Temporal Pattern Mining in Dynamic Environments

von Andreas D. Lattner

Dissertation

zur Erlangung des Grades eines Doktors der
Ingenieurwissenschaften
– Dr.-Ing. –

Vorgelegt im Fachbereich 3 (Mathematik & Informatik)
der Universität Bremen
im März 2007

Datum des Promotionskolloquiums: 30. Mai 2007

Gutachter: Prof. Dr. Otthein Herzog (Universität Bremen)
Prof. Dr. Stefan Wrobel (Universität Bonn)

Acknowledgment

First of all, I would like to thank my doctoral advisor Prof. Dr. Otthein Herzog for his continuous support while I have been writing this dissertation. His motivating and inspiring – usually also challenging – comments in our meetings have been extremely valuable to improve this work and to accelerate the progress. I am very grateful that he gave me the opportunity and the support in conducting my research.

I would also like to express my gratitude to Prof. Dr. Stefan Wrobel, not only for evaluating my thesis, but also for the kind invitation to the Fraunhofer IAIS in Sankt Augustin for presenting my work and the many helpful comments and for fruitful discussions with him and his colleagues.

Furthermore, I want to thank my (partly former) colleagues from the Artificial Intelligence Research Group (AG-KI) at the Universität Bremen for many exciting discussions about their as well as my research. Especially, I would like to thank Prof. Dr. Ingo J. Timm (now working at the Johann Wolfgang Goethe-Universität, Frankfurt am Main) and Prof. Dr. Holger Wache (University of Applied Sciences Northwestern Switzerland, Olten, Switzerland) for the scientific discussions, their support with Prolog, and many motivating words during the ups and downs of a PhD student. I am also very grateful to my other colleagues for giving many valuable comments on my work and for proof-reading parts of my dissertation. In alphabetical order I would like to thank Hanna Bauerdick, Jan D. Gehrke, Dr. Björn Gottfried, Dr. Peter Knirsch (Theoretical Computer Science Research Group, Universität Bremen), and Jörn Witte. I would also like to thank Dr. Andrea Miene (Faserinstitut Bremen e.V.) for providing the RoboCup 2D simulation league matches analyzed in her dissertation. My special thanks go to Dr. Thomas Wagner for mental and physical support – in the form of continuous supply of caffeinated liquids – sharing the room with me in the final phase of my dissertation. I would also like to thank Dr. Tom Wetjen (BASF, Ludwigshafen) for sharing his knowledge how to nicely present algorithms with LaTeX.

I want to express my deep gratitude to the *Virtual Werder 3D* team, namely Carsten Rachuy, Arne Stahlbock, PD Dr. Ubbo Visser, and Tobias Warden, for their great effort in our RoboCup project. Special thanks go to Carsten Rachuy for extending the *SeqDraw* program and to Tobias Warden for the implementation of the *FactProducer* which have been used in this dissertation.

There are many people scattered around the world who supported me while writing this thesis. I would like to express my gratitude to Prof. Dr. Frank Höppner (Fachhochschule Braunschweig / Wolfenbüttel) for interesting discussions about support computations. I also would like to thank Dr. Jan Struyf (Declarative Languages and Artificial Intelligence research group of the Katholieke Universiteit Leuven, Belgium) for his support w.r.t. *ACE/WARMR* as well as his colleagues Dr. Hendrik Blockeel and Dr. Luc Dehaspe for providing *ACE*. I am also very grateful to Dr. Terrance Swift (State University of New York at Stony Brook, USA) for his immediate help with XSB Prolog. Furthermore, I would like to thank Dr. Guido Cervone (George Mason University, Fairfax, VA, USA) for great scientific discussions during our various journeys through Europe and America.

Last but not least, I would like to thank my friends and family for all their support and understanding during this challenging period of my life. They have convinced me that there are many other problems and pleasures beyond scientific ones. I am deeply indebted to Mine Hartog for her unlimited, sacrificial support in every sense and I want to thank her for the permanent warmth she has been giving to me.

Bremen, March 2007

Andreas D. Lattner

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Modern information technology allows for storing and processing huge amounts of data and thus, the interest in taking advantage of the available data by applying data mining methods has increased in the last decades. While knowledge discovery in databases (KDD) has been defined as the whole "process of finding knowledge in data" including preprocessing and interpretation of the results, data mining is one step in this process where data mining methods are applied to actually search for patterns of interest [FPSS96].

Many domains feature a dynamic characteristic and it would be useful to learn temporal patterns, e.g., in stock market analysis, medicine, weather forecast, alarm sequences in telecommunication, access patterns in the WWW, DNA sequences, etc. (cf. [GRS99, PHMA+01, dFGL04]). In the logistics domain, for instance, there might be many different kinds of objects like different transport vehicles (e.g., trucks, ships, or planes), different actors or organizations (e.g., storages, transport companies, manufacturers), highways or tracks, reloading points, etc. Different events can occur like traffic jams, weather events, break down of a transport vehicle, or delay of some goods. It would be valuable to identify repeating temporal patterns that lead to certain situations in order to predict a traffic jam or a delay and initiate some counter-actions in order to avoid financial loss or penalty payments. Such a pattern could, for instance, look like this: If the traffic density is medium and increasing on highway X on a Friday afternoon and the weather is rainy, it is likely that there will follow a traffic jam on highway Y. Similar temporal patterns could be identified in medicine. It might be interesting, for instance, to identify (temporal) relations between certain events like decreasing blood pressure, some medication, low pulse, and a collapse later on. A hierarchical structure of the types of medication might be helpful in order to identify rules abstracting from a concrete active ingredient.

In many domains, temporal information is available or could be easily acquired. This temporal information could be, for instance, represented as an ordered sequence of events (without explicit time points), events with annotated time points, or events

with temporal extent that can also occur concurrently. Such sequential or temporal information is addressed explicitely by temporal data mining approaches, for instance, if times are available when some items in transaction databases have been purchased or a sequential order of some events is known (e.g., [AS95, Höp03, LS06]).

In this thesis, we focus on dynamic environments with one or more agents. Russell and Norvig [RN03, p.32] denote an *agent* "anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators". Wooldridge [Woo99] describes an "intelligent agent" as one that is "capable of flexible autonomous action in order to meet its design objectives". Wooldridge further specifies flexibility as reactivity, pro-activeness, and social ability, i.e., intelligent agents should be able to respond to changes in the environment, to show goal-directed behavior, and to interact with other agents. Referring to Wooldridge and Jennings [WJ95] some researchers follow a "stronger notion of agency", and characterize them by further properties such as knowledge, belief, intention, or emotions. Many different architectures for intelligent agents have been defined – logic-based architectures, reactive architectures, BDI (beliefs, desires, intentions) architectures, and layered architectures (cf. [Woo99]).

A *dynamic environment* is defined as an environment with dynamic aspects, i.e., where the state of the world can change by actions of agents or events (without intentional action of an agent in the environment). The state of the world can change anytime and without performing any action from an agent's point of view. Consequently, actions and events can occur concurrently. An action of an agent could, for instance, be moving to a certain position or unloading some good from a truck. A *dynamic scene* is the concrete observation of a dynamic environment for a specific time interval. For this time interval for each time point the (belief of the) current state of the world – and thus time points of state changes – is known.

Agents in dynamic environments have to deal with world representations that change over time. In order to allow agents to act autonomously and to make their decisions on a solid basis, an interpretation of the current scene is necessary. Scene interpretation can be done by checking if certain patterns match the current belief of the state of the world. If intentions of other agents or events that are likely to happen in the future can be recognized, the agent's performance can be improved as it can adapt its behavior to the situation. If more elaborated technologies like planning should be used by an agent, the representation of the agent's belief including background knowledge for its behavior decision can become very complex, too. It is necessary to represent knowledge about object types and their properties, actual scenes with objects, their attributes and relations. If even more complex scenes with temporal extents shall be described, this additional dimension must also be incorporated in the formalism.

This thesis focuses on the domain of robotic soccer for illustration and parts of the evaluation. Soccer serves as a good example for agents in dynamic environ-

ments. Here, various objects with different roles interact in the dynamic environment, objects are in various relations, actions are performed and events can occur concurrently. The different players can perform actions like repositioning, covering of opponents, passing to a team mate, shooting to the goal, or clearing the ball. Different events or strategic moves can occur, for instance, scoring a goal, offside, or wing play. The players and the ball can be in different (spatial or spatio-temporal) relations, e.g., an object can be closer to the opponent's goal than another or the ball can be approaching a player or a region. Describing, for instance, a *one-two* pass (one player passing to another and then the second passing back to the first) involves a number of consecutive and concurrent actions of and relations between objects like the distances of the ball to the players, the kick actions, and the movements of the ball and the players. The simulation leagues of the RoboCup initiative provide a good testbed and with *Virtual Werder 3D* a team of the RoboCup 3D simulation league is available [LRS+06]. More details about this domain will be presented in Section 2.1.2.

This work is focused on qualitative representations of dynamic scenes as they allow a concise representation of the relevant information. Such a representation provides means to use background knowledge, to plan future actions, to recognize plans of other agents, and is comprehensible for humans at the same time. Quantitative data has to be mapped to a qualitative representation, e.g., by dividing time series into different segments satisfying certain monotonicity or threshold conditions as suggested by Miene et al. [MLVH04, MVH04]. One example is that if the distance between two objects is observed, it can be distinguished between intervals with increasing and decreasing distances representing approaching and departing relations (cf. [MVH04]). Having such a qualitative representation available, a knowledge-based behavior decision can be performed and – as intended in this thesis – frequent temporal patterns can be mined from this representation.

The following sections describe the goal and research questions of this thesis, classify the work into existing research fields and give an overview of the structure of the thesis.

1.1 Goal and Research Questions

The goal of this thesis is to develop a data mining method in order to learn frequent temporal patterns from dynamic scenes. Having in mind rather complex situations with different objects of various types and relations as well as temporal interrelations of actions and events, the approach should provide means to mine complex temporal patterns taking into account these aspects. One important application of temporal patterns is the prediction of future events or situations. Thus, the generated patterns must be transformed into prediction rules for these purposes. The main research

questions addressed in this thesis are:

- **What is an adequate representation of dynamic scenes?** It is important to define an appropriate representation for dynamic scenes as this will be the input to the pattern mining algorithm. Obviously, whatever is missing in the dynamic scene representation cannot be captured by the learned patterns. The representation should support all relevant information about objects, object types, relations, attributes, and time.
- **How should temporal patterns be represented and how can they be mined efficiently?** Similar to the representational issues of the dynamic scene, an adequate representation of temporal patterns must be found. It is also important to come up with a solution to find all frequent patterns from a dynamic scene. Are there any existing approaches that can be used for the intended mining task? What are their weaknesses or limitations?
- **How can we use temporal patterns for prediction?** This research question addresses the transformation of temporal patterns to prediction rules and the application of such rules. The predictive accuracy of such rules must be evaluable for a given dynamic scene.
- **What are interesting prediction rules?** It is important to find an interestingness measure for the evaluation of prediction rules. The number of potentially relevant patterns and prediction rules in a dynamic scene can be prohibitively high and thus, it is necessary to define some criteria for evaluation.

1.2 Classification of this Study

Different research fields are relevant for this thesis. First of all and most obviously this work can be classified into the field of data mining (and thus can be seen as part of a knowledge discovery process) as the goal is the development of a novel pattern mining approach. As it is searched for frequent patterns (and no target concepts for learning are provided by a teacher), it is an unsupervised learning approach in order to identify associations in temporal data. Most similar to the intended solution are association rule mining approaches and their relational and sequential extensions as well as frequent episode discovery approaches (e.g., [AS94, AS95, DT99, Lee06, MTV97, Höp03]).

The work is also motivated and inspired by the field of intelligent agents in dynamic environments like, for instance, soccer playing agents in the RoboCup domain, intelligent vehicles, or intelligent agents in the logistics domain. In all these cases, an autonomous agent has to update and evaluate his belief of the world, make

decisions about his behavior, and perform actions. Prediction rules can be helpful in the behavior decision process. The application domain of simulated robotic soccer is also used for parts of the evaluation of prediction rules in this thesis.

The knowledge representation of the agent's belief of the world, concepts and their interrelations, actual scenes, etc. forms another relevant field of this thesis. This includes particularly the qualitative representation of dynamic scenes including spatial, temporal and spatio-temporal relations of objects.

The main contributions of this thesis are:

- The development of a novel temporal pattern mining approach for relational interval-based dynamic scene descriptions.
- The realization of an efficient prediction rule generation from frequent temporal patterns.
- A new set of time interval relations including a composition table for temporal reasoning.
- A mapping of the learning task to the well-known relational association rule learning algorithm *WARMR*.
- An extensive evaluation of the runtime complexity with artificial test data, a comparison of *WARMR* and *MiTemP*, and the application of the approach to soccer matches of the RoboCup 3D simulation league.

1.3 Overview of the Work

Following this introduction **Chapter 2** provides some basic definitions and foundations of the fields of knowledge discovery and machine learning as well as of agents in dynamic environments. A concrete scenario of the RoboCup domain is used to extract the problem description and a list of requirements on the data mining method to be developed.

In **Chapter 3**, the state of the art is described w.r.t. existing learning approaches and the representation of dynamic scenes. Different learning approaches are presented and checked against the defined requirements. Besides formalisms for representing actions, events, and time in this chapter, some basic formalisms for a qualitative representation of space and time as well as for motion description are also presented.

In **Chapter 4**, the formalism for the representation of dynamic scenes and temporal patterns and the temporal pattern mining algorithm *MiTemP* is introduced. This is the main achievement of the thesis and it includes all relevant definitions of the formalism as well as proofs, algorithms, and complexity examinations. It

is shown how to enumerate the temporal patterns in a potentially infinite pattern space based on a quasi-order and how subsumption relations between patterns can be checked. The definition of an optimal refinement operator avoids redundancies in the pattern generation process during enumeration. In this chapter, it is also described how the existing relational association rule mining algorithm *WARMR* can be used in order to perform the temporal pattern mining task. Different aspects of the learning task have to be transformed into the corresponding format. Some drawbacks of this solution are also discussed in this chapter.

The generation and evaluation of prediction rules is addressed in **Chapter 5**. An evaluation measure for prediction rules is proposed that takes into account – besides frequency of patterns and confidence of prediction rules – different aspects, namely information content, predicate preferences, pattern size, and specificity. The generation of prediction rules takes advantage of the sequential order in the temporal patterns and thus, an efficient prediction rule generation algorithm is set up.

The evaluation of the whole approach is presented in **Chapter 6**. In this chapter, the different experiments are described including practical complexity examination with varying parameters of the algorithm and input data, a comparison of the two solutions using *WARMR* and *MiTemP*, and the creation and use of prediction rules in soccer matches of the RoboCup 3D simulation league.

Chapter 7 summarizes the results of the thesis and closes with some ideas for future work.

Chapter 2

Preliminaries and Requirement Analysis

This chapter states more precisely which goals are to be achieved by this thesis. In the beginning, some definitions and relevant machine learning basics are presented in order to improve the comprehensibility of the requirements and the state of the art presented in Chapter 3. In this context, a brief introduction in the field of agents in dynamic environments is also given. Afterwards, relevant scenarios for the learning approach are presented, focusing on the RoboCup 3D simulation league [ROME06] as a special case of agents in dynamic environments.

Based on an introductory scenario the problem to be solved and the requirements are described in the subsequent sections. The requirements listed in this chapter build the basis for the evaluation of the existing approaches in the subsequent chapter.

2.1 Basics

Before the actual problem is described and the requirements are presented some basic information of the research field and some definitions are provided. The following two subsections address the fields of machine learning and data mining as well as agents in dynamic environments.

2.1.1 Machine Learning and Data Mining

Over the years different definitions of learning have been stated. One definition by Simon is given as follows: "Any change in a system that allows it to perform better the second time on repetition of the same task or on another task drawn from the same population" [Sim83]. Michalski defines learning as "constructing or modifying representations of what is being experienced" [MCM86]. Langley refers to learning

as "the improvement of performance in some environment through the acquisition of knowledge resulting from experience in that environment" [Lan96, p. 5]. A similar but more formal definition by Mitchell is: "A computer program is said to **learn** from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E" [Mit97, p. 2]. However, there are different opinions which topics should be considered as machine learning: While Langley mentions neural networks and genetic algorithms besides instance-based learning, rule induction, and analytic learning as five "paradigms for machine learning" [Lan96, p. 20-22], Kubat et al. refer to the first two fields just as "Close Neighborhood of Machine Learning" [KBM97, p. 58].

Referring to Fayyad et al. [FPSS96, p.3], the term *knowledge discovery in databases* "was coined in 1989 to refer to the broad process of finding knowledge in data, and to emphasize the 'high-level' application of particular *data mining* methods". Fayyad et al. [FPSS96, p.6] define *knowledge discovery in databases* (KDD) as the "non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data". While the term *data mining* is used synonymously to *knowledge discovery in databases* in some cases (e.g., [SA96]), Fayyad et al. define it as a particular step in the knowledge discovery process where data mining methods are applied in order to "find patterns of interest" [FPSS96, p.11]. KDD as well as Machine Learning address the "study of theories and algorithms for systems which extract patterns and models from data", but KDD can be seen as an extension in terms of "finding special patterns (ones that might be interpreted as *useful* or *interesting* knowledge, (...)) in large sets of real-world data" [FPSS96, p. 5].

Learning approaches can be classified w.r.t. different aspects. Besides the learning algorithm and representational issues, aspects like the type of feedback, and incremental vs. non-incremental learning can be distinguished (cf. [Lan96, KBM97, Mit97]). The following subsections address different basic aspects of machine learning tasks.

Type of Feedback

Referring to Russell and Norvig the three major distinguishing classes of the type of feedback are supervised, unsupervised, and reinforcement learning [RN03]. In the case of supervised learning, a "teacher" guides the learning process by determining what to learn. Learning from examples is supervised. Classes and positive (and sometimes also negative) examples of the classes are given to the learner (cf. [CMM83]). In contrast, no classes and learning examples with target classes are provided to the learning algorithm in unsupervised learning. Kubat et al. [KBM97, p. 43] describe the task of unsupervised learning "to generate conceptual taxonomies from non-classified objects". The classes are constituted by the learner as it is the

case with clustering approaches (see, e.g., [JMF99]). In reinforcement learning, a system learns from a reward that rates its performance. Reinforcement learning has been widely used in agents as it "addresses the question of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goals" [Mit97, p. 367].

Representational Issues

One of the most important issues in machine learning is how to represent the input of the learning algorithm (e.g., training instances), background knowledge, and the hypothesis to be learned. Kubat et al. [KBM97, p. 7-10] present a selection of some representations in their review of machine learning methods which is briefly summarized in the following.

The representation language with least complexity and least expressive power is the *propositional calculus* (a.k.a. zero-order logic; [KBM97]). In this case, examples and concepts are represented by conjunctions of Boolean constants. Disjunction and negation are other connectives that might be used [KBM97].

In *attributional logic*, examples and concepts are described by values of predefined attributes. The advantage is a more compact representation as for each attribute just the valid values have to be shown. It is also possible to represent disjunctions of values. Michalski presented a formal basis for such a representation language in the variable-valued logic ([Mic73b]; cf. [KBM97]). Referring to Kubat et al. the *attributional logic* is "significantly more practical than zero-order logic", although the expressiveness is equivalent "in a strict mathematical sense" [KBM97, p. 8].

Horn clauses – a subset of first-order predicate logic – allow for describing complex objects and relations between them. The programming language *Prolog* is based on this representation (cf. [KBM97]). In Inductive Logic Programming systems like *FOIL* and *Progol*, such representations are used [Qui90, Mug95].

Other representations – Kubat et al. additionally mention *second-order logic, frames, grammars*, and *finite automata* [KBM97] – have been used in learning tasks. It is also the case that subsymbolic learning approaches provide different representations for the learned hypotheses. Further representations will be described together with the approaches where needed in Chapter 3.

An additional important question for this thesis is how the temporal dimension is handled or could be represented. Many approaches do not take the temporal dimension into account explicitely but just learn rules from a "snapshot" of the universe of discourse. Relational approaches could represent time by using an additional parameter in the relations. In dynamical environments, the world changes over time and thus, important information can be acquired from repeatedly appearing patterns in this development.

Hypothesis Generation

The goal of learning is to create hypotheses which match the data as well as possible (including unseen data besides training). In the case of learning concepts from examples, the learned concept descriptions should cover all positive and none of the negative instances in the best case, i.e., they should be consistent and complete. In an attribute-valued logic, ten attributes with five possible values would lead to $5^{10} = 9765625$ possible vectors describing instances and even $2^{9765625}$ possible concept descriptions (this is the size of the power set of all possible instances) [KBM97]. This illustrates the problem of computational tractability which is addressed by induction and heuristic search [KBM97].

A typical approach to learning is to perform a search through the space of hypotheses (cf. [Mit97]). There exists a most general hypothesis description and a set of most special descriptions (which usually represents instances in learning from examples). Hypotheses might be more general or more specific than others and this relation forms a generalization lattice of the hypothesis space. A search through this space can be done by generalization and specialization operators [KBM97]. In order to perform the search, existing exhaustive (e.g., depth-first search) or heuristic search algorithms (e.g., best-first or beam search) can be applied (cf. [KBM97]).

Two "Classic Methods of Learning" [KBM97, p. 17] are the divide-and-conquer learning and the progressive coverage. In the first case, the set of examples are partitioned consecutively, e.g., by using an attribute for separation, until a subset satisfies some conditions to be assigned to a class (e.g., *C4.5* [Qui93]). In the progressive coverage, a hypothesis is generalized by adding disjunction parts until all positive examples are covered (e.g., *AQ learning* and *CN2* [Mic69, CN89]).

Incremental Learning

If the learning algorithm can handle one instance at a time, it performs incremental learning; non-incremental learners need all training instances to be presented at once for learning [Lan96]. If a learner is a non-incremental "batch algorithm", it will be necessary to "re-run the entire learning procedure on all data" if new examples are presented [KBM97, p. 31]. Langley also distinguishes between online and offline learning tasks where the instances are presented one at a time or all together simultaneously, respectively [Lan96]. Referring to Langley, "incremental methods lend themselves to online tasks and non-incremental approaches seem most appropriate for offline problems", but it is possible to "adapt a non-incremental method to an online learning task" and to "use an incremental method for an offline learning task" [Lan96, p. 20].

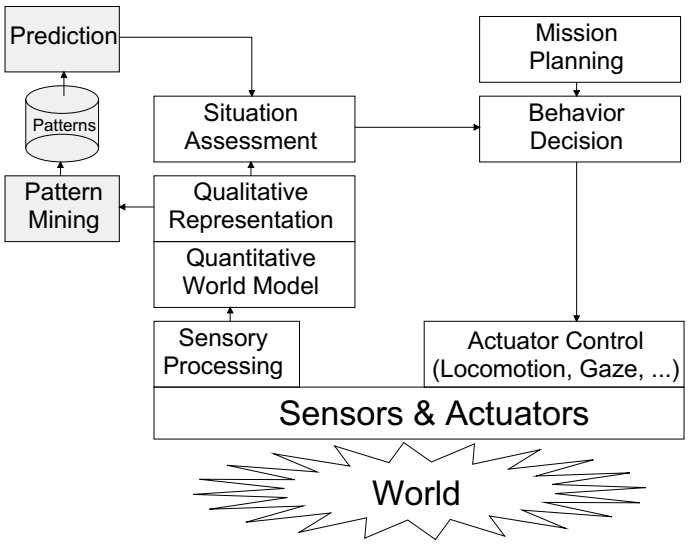

Figure 2.1: Agent architecture

2.1.2 Agents in Dynamic Environments

Agents can be located in different environments. Russell and Norvig [RN03, p.40-43] list a number of different properties of environments, among others, if an environment is fully or partially observable, deterministic or stochastic, static or dynamic, discrete or continuous, single agent or multi-agent. The learning approach to be developed must be able to deal with dynamic environments. In this context, dynamic means that concurrent actions or events[1] can happen while the agent is deliberating. W.r.t. the remaining properties mentioned above, there is no restriction to just one alternative for the approach. However, in the robotic soccer domain the environment is (from the agent's perspective) partially observable, dynamic, continuous, and characterized by multi-agent actions.

Usually, agents in dynamic environments have a set of sensors which they use to perceive the environment (e.g., cameras), and a set of actuators to act in this environment (e.g., a drive for motion). The data perceived by the sensors is processed and a world model is created which represents the agent's current belief of the world. The behavior decision of an agent is based on this world model, i.e., depending on the current situation it chooses the action (or a set of actions) which to its belief let it perform appropriately. There might also be a component for mission planning which also has influence on the behavior decision. The chosen actions of the behavior decision component then lead to actual control of the actuators. Such an architecture can be seen in Figure 2.1 (adapted from Dickmanns, e.g., [Dic03]);

[1]While an action is (intentionally) performed by an agent an event can occur independently from intentional activities, e.g., if it starts to rain during a soccer match.

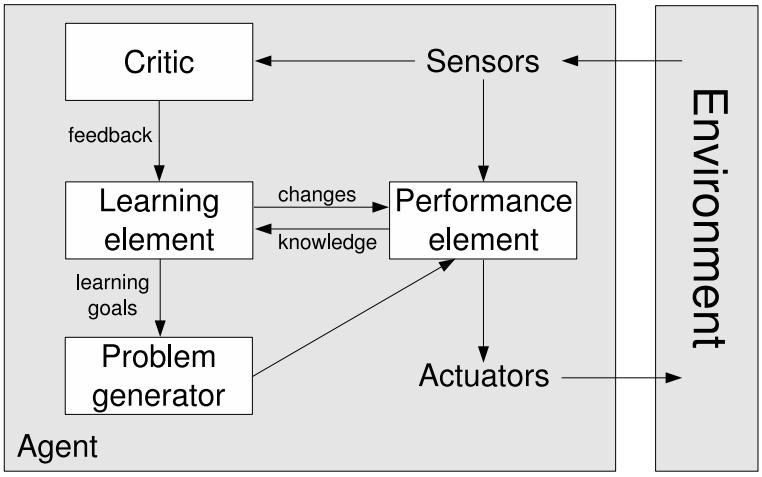

Figure 2.2: Learning agent (adapted from [RN03, p. 53])

the component for a qualitative representation was added to the architecture by Lattner et al. in [LTLH05]. Further discussion about such an explicit qualitative representation and an approach to knowledge-based behavior decision is presented in [LTLH05, LGTH05, GLH05].

The relation of agents in dynamic environments to learning is outlined by the shaded boxes on the left side of Fig. 2.1. Identifying patterns in dynamic scenes can be valuable in order to predict future situations or actions of other agents and thus improve the agent's performance. As it is illustrated in the figure, identified patterns can be stored and used as additional input to the situation assessment component. In the following section, the RoboCup simulation league will be introduced as a special case of agents in dynamic environments. This league is used as a testbed in some of the experiments for the evaluation of the approach.

Russell and Norvig [RN03] divide learning agents into four conceptual components (see Fig. 2.2). This general design is "classic in the machine learning literature" [RN03, p. 56] and can also be found in a similar version in the book of Mitchell [Mit97, p. 12] – although Russell and Norvig focus more on the agent researcher's perspective. The learning element is responsible for making improvements through learning. The performance element selects the actions the agents should perform. The critic gives feedback to the learning element on how the agent performs and thus, the learning element can change the performance element in order to improve the agent's action selection. The task of the problem generator is to suggest exploratory actions that might lead to new and informative experiences [RN03]. Without this element, it could happen that the agent's performance element stagnates and keeps doing the same actions over and over.

2.2 Scenarios

After a short introduction of the RoboCup 3D simulation league a scenario of this league is presented in order to identify relevant aspects for mining temporal patterns in dynamic environments.

2.2.1 RoboCup 3D Simulation League

RoboCup is an international initiative for research and education. It consists of four subfields: RoboCupSoccer, RoboCupRescue, RoboCupJunior, and RoboCup@Home. In different leagues, various research topics like robotics, multi-agent systems, and machine learning are addressed. The first RoboCup was held in 1997 in Nagoya, Japan, and ever since it has been held once a year at different locations. RoboCup-Soccer consists of five leagues: simulation league, small-size league, middle-size league, four-legged league, and humanoid league. In the simulation league, the agents are just simulated, i.e., no real robots exist. This league mainly addresses higher-level tasks like cooperation, communication, and learning. The other leagues are all "real robot" leagues with different hardware and field settings. In the hardware-based leagues, a lot of challenging and important tasks for basic skills like perception, self localization, and actuator control have to be solved. As experiments with real robots are more expensive, time consuming, and additionally still problems with the basic skills can arise, it is obvious that the simulation league provides a better testbed for investigating intensive high-level tasks as learning. Therefore the soccer simulation league was chosen for parts of the evaluation of the presented approach (besides other automatically created synthetic data sets for the evaluation of certain aspects of the learning algorithm).

Among the RoboCup leagues the simulation league provides the most realistic environment w.r.t. the soccer rules, e.g., the number of players, sizes of the field, the goals, and the ball, throw-in, corner kicks, offside, communication, etc. On the other hand, many real robotic problems are suppressed. However, the simulators provide means to add noise to perception and actuator control, i.e., the perceived signals and performed actions are not perfect but interfered with some random signals. The older 2D simulation league consists of a flat world, i.e., no third dimension exists. A proposal for advancing to a 3D soccer simulation league was presented by Kögler and Obst [KO04]. The first competition of this league took place at the RoboCup in 2004 in Lisbon, Portugal.

In the 3D soccer simulation (see Fig. 2.3), the perception of the environment is done by a noisy vision sensor with view direction that can be moved by pan-tilt commands and a listener that can perceive communication of other agents in a restricted range, i.e., for each agent the world is just partially observable. Visual perception provides a list of objects including the dynamic objects – the team mates,

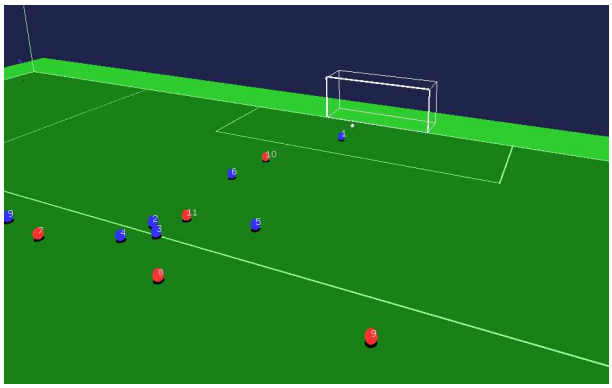

Figure 2.3: RoboCup 3D simulation league

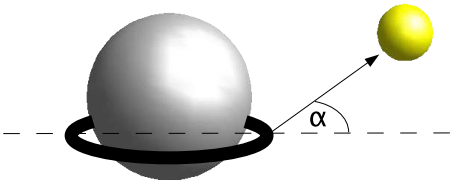

Figure 2.4: Kick effector of an agent in the RoboCup 3D simulation league

opponent players, and the ball – with polar coordinates relative to the own agent's position. The position has to be determined by landmarks, e.g., the marks at the corner points of the field. The world is perceived every 200 ms and the think and act cycles, i.e., behavior decision and actuator control, can be performed concurrently. The agent's body is represented by a sphere. Currently, there are efforts in building more complex physical agent models in order to simulate biped robots which should replace the sphere agents in the long term. The actuators to be used during a regular match are the drive effector (omni-drive) and the kick effector for kicking the ball; before kickoff it is also possible to use a "beam" effector which immediately teleports an agent to a desired position. The kick effector in the current soccer simulation server version is an omni-kick effector, i.e., if the ball is within the kick range it is kicked away from the agent's center (see illustration in Fig. 2.4). Besides the kick power, it is possible to define the kick angle in the z axis, i.e., if the ball should be kicked high or flat on the ground.

The provided information perceived by the sensors has to be processed in order to assess the situation and to decide what action to perform next. Among other facts, it might be interesting to know who is in ball control, who is the fastest player to the ball, and if the ball is in a distance to perform a kick to the goal, or if there are uncovered players, etc. Such information can be extracted from the positional information of the objects (cf. [Mie04] and Section 3.2.3).

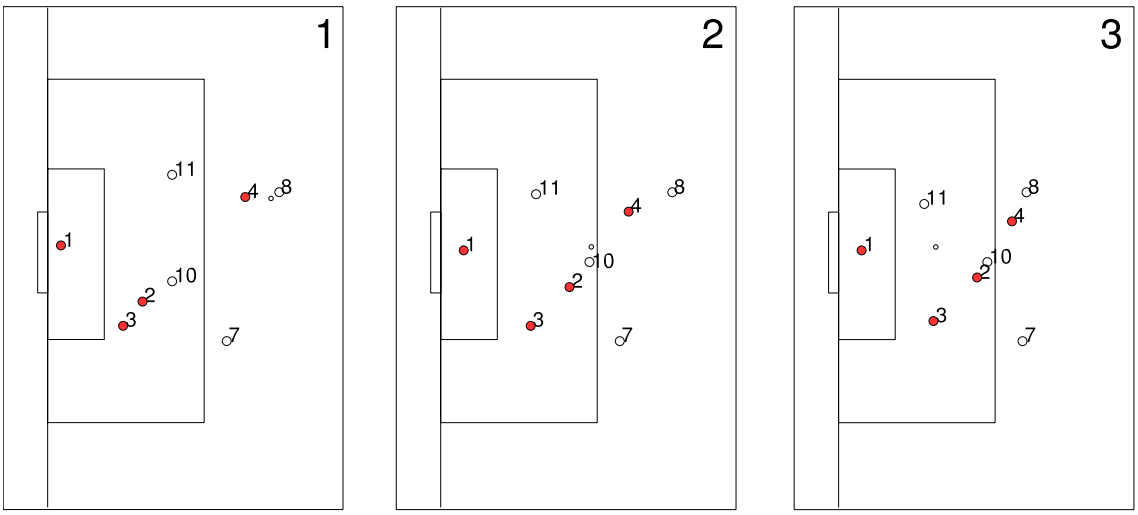

Figure 2.5: Illustration of an attack in a RoboCup soccer match

2.2.2 Typical Soccer Situations

In this section, a scenario of the robotic soccer domain is presented in order to give an idea of the complexity of dynamic scenes and show the potential of learning patterns in dynamic environments. The following scenario also builds the basis for the requirements to be defined in Section 2.4. In the following chapters, the robotic soccer domain also serves for the illustration of concepts and algorithms. The examined scene is an illustration of an attack between two teams of the RoboCup 3D simulation league, namely *Aria3D* (Amirkabir University of Technology, Tehran, Iran) and *Brainstormers3D* (Universität Osnabrück, Germany) which results in a goal for *Aria3D* (Fig. 2.5). It is taken from a sequence (145.48 - 149.98 s) of the final on July 17th 2005 of the RoboCup 2005 in Osaka, Japan which ended 2:0 for *Aria3D*. In this scene, player 8 of the white (hollow) team (*Aria3D*) possesses the ball (Fig. 2.5-1), then passes to player 10 (Fig. 2.5-2), and player 10 finally performs a kick to the *Brainstormers'* goal (Fig. 2.5-3) which leads to the 1:0 for *Aria3D*. If we take a closer look into the scene, we can gather information like:

- Player Red_01 – the goalkeeper – is inside the goal box (Scene 1 - 3).
- Players Red_01, Red_02, Red_03, White_10, and White_11 are inside the penalty area (Scene 1-3).
- Player White_08 possesses the ball, i.e., is closest to the ball and has control of it (Scene 1).
- Player Red_04 is in front of White_08 (Scene 1).

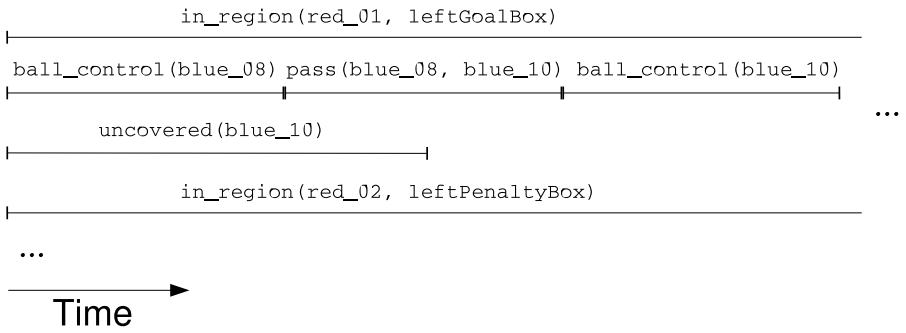

Figure 2.6: Exemplary illustration with validity intervals

- Player Red_04 is the closest player of the red team to the ball and approaches it (Scene 1).
- Players White_10 and White_11 are not covered (Scene 1).
- Player White_11 cannot be reached by a pass as the way is blocked by Red_04 (Scene 1).
- Player White_10 can be reached by a pass (Scene 1).
- Players White_07, White_10, and White_11 are team mates of White_08 who are closer to the opponent's goal (Scene 1).
- Player White_08 passes to White_10 (between Scene 1 and 2).
- Player White_10 possesses the ball, i.e., is closest to the ball and has control of it (Scene 2).
- Player Red_02 is the closest of the red players to the ball and approaches it (Scene 2).
- The ball is close enough to the goal in order to shoot at it (Scene 2).
- Player White_10 shoots at the right corner of the opponent's goal; the ball is moving very fast (between Scene 2 and 3).
- ...

This simple example which only takes a few seconds of a match already gives an idea of the complexity of dynamic scenes. Figure 2.6 gives an exemplary illustration of the scene with time intervals. The time axis is shown from the left to the right. The intervals denote the validity of different predicates. It can be seen that, e.g., `ball_control(white_08)` is directly before `pass(white_08, white_10)`. Different

kinds of information play a role in the description of such a scene (cf. also [Mie04] where RoboCup scenes have been described by a qualitative motion description):

- **Objects:** Soccer agents of both teams, the ball, marks, goal posts, ...
- **Classifications:** Ball vs. player, red vs. white team, roles (e.g., goalkeeper, defender, midfield player, offensive player)
- **Regions:** Goal box, penalty area, center circle, ...
- **Positions:** Inside goal box, inside center circle, ...
- **Directions:** in front of, behind, left, right, ...
- **Distances:** Close, far, reachable, closer to, ...
- **Motion:** Development of distances (e.g., approaches), speed (slow, fast), development of speed (accelerating, decelerating)
- **Events and Actions:** Kick-off, goal, pass, ...
- **Higher-level facts:** Covers player, possesses ball, ...

In order to describe and identify complex patterns, it is necessary to represent temporal interrelations (e.g., before, after, during) of different facts, events, and actions. The durations of events and actions, as well as the validity of facts, can potentially be in arbitrary temporal relations. The identification of frequent temporal relations is an important aspect of pattern mining in dynamic environments.

2.3 Problem Description

The goal of this thesis is to provide means for temporal pattern mining. It is necessary to set up a pattern description language and to develop algorithms in order to find frequent patterns in dynamic scenes. In contrast to predicates in dynamic scenes, the arguments of predicates in patterns do not need to be concrete objects as arguments but can also be variables. An example for a pattern is shown in Fig. 2.7. This pattern says that there are objects X, Y, and Z. X is in ball control directly before a pass between X and Y is performed. Additionally, Y is uncovered and closer to goal than X, and X is attacked by Z.

If a pattern matches a dynamic scene, all variables are bound to concrete objects in the scene for a match. In order to analyze soccer matches, e.g., to identify typical sub-scenes or even to learn about a team's strategy, such matches can be observed and patterns can be extracted. It might be interesting to find answers to questions

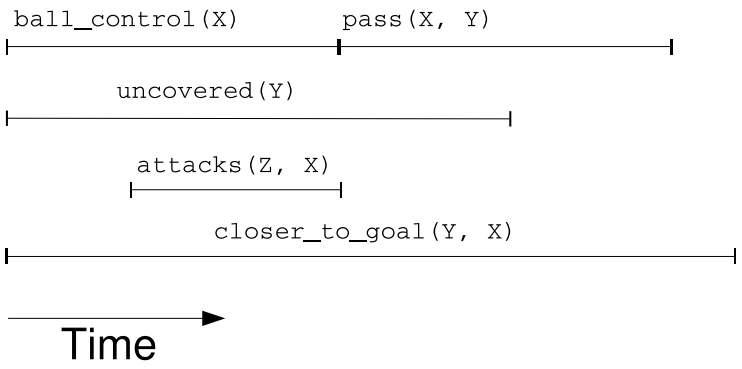

Figure 2.7: Exemplary illustration of a pattern

like: who is chosen as pass partner in which situations, in which situations is a shot performed towards the goal, what are the positions of players in certain situations, what are typical situations for successful or failed passes, is an attack of a team usually performed at a certain side, etc.

Among potential applications for the mining of patterns are (not restricted to robotic soccer, of course): Learning about (causal) relationships in dynamic scenes which might be used for behavior prediction, supporting the user at the task of setting up rules for scene interpretation (e.g., for reporting on a soccer match automatically), and learning strategies of opponents. To remain in the soccer domain, it could be valuable to identify patterns like:

- In many cases (55% of the cases in the dynamic scene), if attacked by an opponent, the player possessing the ball passes to one of his team mates who is closer to the opponent's goal and not covered by any opponent player (cf. Fig. 2.7).
- In most cases (99%) when the ball is shot into the goal, a kick-off event follows.
- The players of a team do not succeed in passing in some cases (20%) if an opponent player is between the ball and the pass partner and in close distance to the pass partner.
- A player always (100%) moves with the ball if he is not attacked by an opponent from the front.
- Before (active) corner kicks, in most cases (90%) there is one player moving to the ball while one of his team mates is moving to the corner of the penalty box. At the corner kick the ball is passed to this player.

Knowing about such patterns could, e.g., be used for setting up the own behavior by trying to avoid certain preconditions that lead to success for the opponent or

to create preconditions that prevent successful opponent's activities. If it can be predicted what is likely to happen, it can also be tried to minimize the risk in the anticipated future situation.

As hard time restrictions usually exist during run-time of a soccer playing agent, we propose a two phase approach where a set of rules is learned offline. These rules could then be validated or proven false online so that the previously learned set could be adapted. Online learning is important in order to adapt the behavior to a new situation or a strategy shift of the opponent. In order to identify patterns which are based on a sufficient amount of data, an offline learning setting seems to be adequate where many matches are taken as input. Starting the learning process in a soccer match from scratch without knowing any patterns would hardly lead to a set of useful patterns at the end of the game. An analogy in real life would be to analyze the behavior of future opponents by examining a number of their matches before the own team has to play against them.

Thus, a set of patterns or prediction rules can be already known by the agent due to previous offline learning or the provision of manually created rules which can be used in the process of behavior decision. An observation of the known rules allows an agent to identify if certain rules match in the current situation (e.g., for the current opponent) and rules could thus be deactivated or refined. While deactivation means that a pattern would not be used any longer for prediction, adaption would lead to small changes in the pattern in a way to better fit the current situation.

It could, e.g., be figured out that at the opponent corner kicks no player moves to the corner of the penalty box any longer (as it might have been learned before) and thus, this prediction rule should be removed or deactivated. Another example is that a rule does not hold the way it has been defined or learned before but might hold after a little modification of the rule, e.g., that an opponent player just dribbles if the additional condition is satisfied that there is not any of his team mates closer to the goal.

The major goal of this thesis is the development of algorithms for the processes of temporal pattern mining and prediction rule generation, namely:

A. **Exhaustive search for frequent temporal patterns:** Given a set of dynamic scenes and some background knowledge about object types and temporal relations find **all** frequent temporal patterns within these dynamic scenes.

B. **Heuristic search for frequent temporal patterns:** Given a set of dynamic scenes and some background knowledge about object types and temporal relations find **many interesting** frequent temporal patterns within these dynamic scenes.

C. **Prediction Rule Generation:** Given a temporal pattern with some predicates create prediction rules with a precondition and a consequence (which

happens later than the precondition) that exceed a certain confidence threshold. Each created prediction rule should be assigned its probability estimate, i.e., its confidence value in the sense of association rule mining (see Section 3.1.2, p. 32).

2.4 Requirements

After having described the general problem, the requirements which should be fulfilled by the approach developed in this thesis are defined in this section. As the focus lies on the learning of situation and behavior patterns in dynamic environments, the requirements mainly address representational and learning algorithm issues. The following subsections describe the requirements for the representation of dynamic scenes, for the representation of the patterns, for the mining of frequent patterns, their evaluation, and for the way how the patterns should be applied for prediction. The requirements at a glance can be seen in Table 2.1.

2.4.1 Representational Issues

The learning algorithm and the intended representation of the patterns to be learned directly depend on the representational expressiveness of the source data – the description of dynamic scenes. As patterns of situations and behaviors should be learned, quite high requirements are demanded: Relations between objects have to be taken into account, the temporal dimension has to be modeled due to the dynamic characteristic of the environment, and different kinds of knowledge have to be considered.

The requirements for the representation of the patterns to be learned address different elements of the representation language like predicates, variables, constants, and conjunctions. It is also required that conceptual information and temporal information have to be integrated into the patterns.

Objects, Properties, and Relations

Dynamic scenes can consist of an arbitrary number of different objects in the general case. In soccer – real as well as simulated robotic soccer – the number of relevant dynamic objects is usually restricted to 23 (two teams of eleven players plus one ball), but the approach must be able to deal with an arbitrary number of objects as it should also be possible to learn patterns in other domains. The objects might appear, disappear, or be part of the scene for the whole time. Each of the objects can be described by a number of properties like, e.g., its size, color, or role. Properties of objects can be described by an attribute value pair, e.g., the `role` of player `red_01` is `goalkeeper`.

No.	Requirement
1	Representational Issues
1.1	Representation of objects, properties, and relational data
1.2	Representation of temporal relations
1.3	Representation of conceptual information
1.4	Representation of background knowledge
2	Generation of Qualitative Abstraction
3	Pattern Matching
4	Search for Frequent Patterns
4.1	Candidate Generation
4.2	Candidate Selection
4.3	Frequency Computation
4.4	Learning Guidance
5	Situation Prediction
5.1	Generation of Situation Prediction Rules
5.2	Application of Situation Prediction Rules
6	Prediction Rule Evaluation

Table 2.1: Summary of requirements

Furthermore, there exist relations between different objects, for instance, spatial relations between objects like `red_07` is `behind` `red_10` or `red_07` is `left_of` `red_10`. These relations do not need to be binary. There could also be relations of a higher arity, e.g., `red_07` is `between` `red_06` and `red_08`. In the following, the term *predicate* is used to denote both – properties and relations. A predicate can be evaluated to true or false.

Temporal Dimension

Some properties of objects and relations between objects might be static, i.e., they might be valid the whole time period of the dynamic scene or even unchangeable (e.g., that `red_10` is an `opponent` of `white_08`). Other properties and relations might just be valid for one or more specific time intervals during the scene, e.g., that a player passes the ball to one of his team mates. Predicates that can have varying truth values over time are called fluents in the situation calculus [Rei01, p. 19]. We also use this term for predicates which are just valid over time intervals.

As it could be seen in the soccer example, it is important that complex temporal interrelations can be modeled. Actions, events, and relations have an extension over time and can appear concurrently. For a detailed description of dynamic scenes, it is necessary to be able to represent temporal relations, like an action, which occurs

before or during another action, e.g., while player `red_07` dribbled to the center of the field player `red_10` moved to the penalty box, or the goalkeeper started moving before the penalty kick action of the opponent player was finished. Relevant work about relations between time intervals has been introduced by Allen and Freksa [All83, Fre92a] and will be discussed in Section 3.2.1.

Conceptual Information

In many cases, objects of different types appear in dynamic scenes. These objects are instances of certain concepts (e.g., `ball` or `player`) and these concepts can be structured in a concept hierarchy. Properties can usually be assigned to a concept, i.e., that only instances of this concept (or one of its sub-concepts) can have such a property. If a property could exist for all objects, it would be assigned to the top-level concept (*object*). For relations the concepts of all appearing objects should be constrainable by defining their domain (e.g., a pass can only happen among *player* instances).

The representation must allow for formulating this information about instances, concepts, domains of properties and relations. The soccer domain has quite flat hierarchies, but it still makes sense to provide information, e.g., that a player can only cover another player (but not a ball). Other domains might request for hierarchies of greater depth (e.g., a taxonomy of participants in the traffic domain).

Background Knowledge

In order to have a concise and understandable representation, it is demanded to represent background knowledge that can be used by inference mechanisms. This has the advantage that it is not necessary to represent all information explicitly but to derive some implicit information from facts and background knowledge. For instance, such a background knowledge rule could state that if an object is inside the left goal box, it is also inside the left penalty box. Another example is that a player is defined to be the last defender if there is no other team mate behind him besides the goalkeeper.

Background knowledge must be taken into account during pattern matching in order to get all matches for a pattern even if not all the information is directly represented by atomic facts.

Representation of Patterns

The basic elements of the patterns to be learned are predicates, variables, and constants. An atomic pattern should just consist of one predicate and a number of variables or constants – as many as the arity of the predicate stipulates. A variable in a pattern does not constrain that a certain object, i.e., a constant must

be part of the predicate. Two examples for atomic patterns are $pass(X, Y)$ and $closerToGoal(red\_07, red\_08)$. In the first example, X and Y denote variables, i.e., each match of arbitrary objects should be counted as a match. In this work, we follow the notation that variable names start with an upper-case letter while constants begin with a lower-case letter. In the second example, red_07 and red_08 are constants, i.e., this pattern should only match between exactly these two objects. It must also be expressible that two arguments in the pattern are identical (i.e., the variables are identical and have to be bound to the same constant).

In order to describe more complex patterns, it must be possible to combine atomic patterns by conjunction operators. A conjunction of atomic patterns should only match if all its elements are satisfied.

The representation of temporal relations is also an important aspect of the patterns of dynamic scenes. As mentioned above, we assume the dynamic scenes to be represented by time intervals which define when certain predicates are valid in the scene. In the learned patterns, temporal relations should be represented in an abstract qualitative way for a concise representation as it is done in interval logics (cf. Section 3.2.1 on page 61).

For each variable in the learned patterns it should be possible to assign a concept or an instance. These concepts do not necessarily need to be leaf concepts but can also be (abstract) concepts in the concept hierarchy for which no direct instances exist. This allows for formulating abstract patterns which hold for instances of a common (super) concept and allows for learning generalizations of similar situations.

Another important aspect is the comprehensibility of the generated rules. In many cases, it is highly desired that rules can be understood (and maybe adapted) by users.

2.4.2 Generation of a Qualitative Abstraction

Although the focus of this thesis is not on the generation of qualitative abstractions, it is an important topic that needs to be mentioned here. In many cases, the information generated by sensors is quantitative like distances, speeds or directions. The learning approach addresses symbolic scene descriptions which change over time. Thus, it is necessary to create a qualitative representation from the quantitative data. For the evaluation part of the pattern mining algorithm where matches of the RoboCup 3D simulation league are analyzed, it is necessary to perform such a qualitative abstraction. This can be done for instance by the generation of monotonicity-based intervals or a threshold-based interval generation for different qualitative classes. This topic is not a major subject of this thesis and has been studied extensively by Miene [Mie04]. Miene's approach is summarized in Section 3.2.3 on page 70.

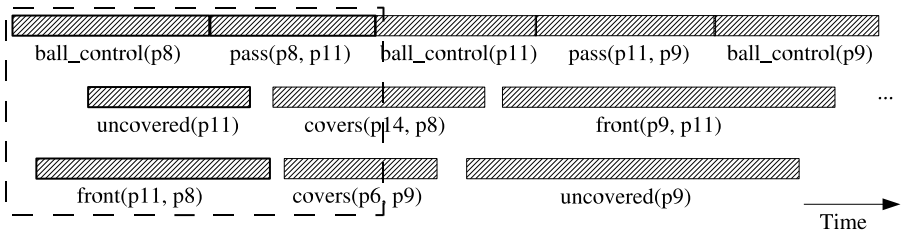

Figure 2.8: Sliding window for pattern matching

2.4.3 Pattern Matching

The process of pattern matching assigns objects to variables of a pattern in a way that all conditions of the pattern are satisfied. In order to restrict matches to some time period, it should be possible to define a maximum window size in which a pattern can match. If the actual predicates of a matched pattern are not within this window size, it should not be counted as a match. Fig. 2.8 illustrates a sliding window which is "moved" over the sequence. Only the predicates with a validity interval inside the sliding window should be considered for pattern matching.

At the current window position in Fig. 2.8 (highlighted by the dotted line), the pattern $pass(X, Y)$ should match when assigning `p8` and `p11` to X and Y, respectively. Inside the window, there is no match for the pattern $pass(X, Y) \wedge uncovered(X)$ (ignoring temporal relations in this example for simplicity) because no assignment satisfies both parts of the pattern (if $p8$ is assigned to X there is no corresponding $uncovered(p8)$ observable).

As the patterns should be learned from a relational representation where also background knowledge can exist in form of rules, this must also be considered during pattern matching. If a predicate in the pattern is not explicitly stored as a fact but can be inferred in combination with the background knowledge, the pattern must still match. It is demanded that all predicates of the pattern are valid for some objects, that same variables are bound by identical objects, that no temporal restriction is violated and all concept restrictions are satisfied by the bound objects.

If a pattern matches, it is important to know which instances match to which variables in the pattern. This is important in order to correctly assess a situation in a dynamic scene, e.g., in order to identify which the relevant objects in the pattern are and what role they play. It might also be important what the exact time periods of the single predicates are. Thus, the result of the pattern matching must be a binding of objects to variables and concrete time points for the start and end of the intervals representing the validity of predicates.

2.4.4 Search for Frequent Patterns

This section defines the requirements for the learning algorithm. Of course, it must be able to deal with the representational issues mentioned above. As the intention of learning in this work is to discover patterns in dynamic scenes and to apply them for improving the performance of the system, supervised approaches are not suitable. We need an unsupervised algorithm that mines patterns without the presentation of any positive or negative examples. The input of the learning algorithm should be a dynamic scene and the result a set of frequent patterns that have been found in the data.

The pattern space must be searched exhaustively in a structured way in order to find all frequent patterns for a dynamic scene. The necessary functionalities are candidate generation, candidate selection, and frequency computation which will be described in the following subsections.

Candidate Generation

In the candidate generation step, new candidate patterns must be generated which might be frequent patterns and thus should be found by the search. Each candidate should be created once at most. The result of such a candidate generation step is a set of new patterns. The candidate generation process should support the structure of the pattern space, i.e., if it is possible to infer that a pattern cannot be frequent (due to its relation to patterns that have been created before) it should not be checked or not even be created.

Candidate Selection

The procedure for candidate selection should process a list of pattern candidates and remove all candidates which have a more general pattern which was identified not to be frequent. This is an important step in order to avoid the (potentially computational expensive) frequency computation for patterns which cannot be frequent due to the monotonously decreasing frequency values of specialized patterns.

Frequency Computation

The procedure for frequency computation must compute the frequency of a pattern (or for a set of patterns) in a dynamic scene by counting the occurrences of the pattern in the scene. As mentioned above, a maximal window size for matching a pattern must be taken into account. It is important, that no predicate is used more than once in a match in order to avoid wrong frequency values. The result of this step is the assignment of frequency values to each of the processed patterns.

Learning Guidance

Due to the complexity of learning, the approach must provide means in order to integrate some guidance to learning. It must be possible to do a pre-selection of the predicates to use or to restrict the pattern language to a subset where certain constraints must be satisfied (e.g., only subsequences that end with a goal-kick in a soccer match).

2.4.5 Situation Prediction

One example for the application of learned patterns is the prediction of situations. If frequent patterns are learned and the components of these patterns are in temporal relations, such patterns can be used for prediction. The learned patterns must be divided into two parts by the prediction rule generation procedure: A precondition and a consequence part as it is the case in association rule mining.

Having a set of temporal association rules, it must be possible to apply the precondition part to some situation and to return the probability of consequences if the preconditions are satisfied. This has to be done by the situation prediction procedure.

2.4.6 Prediction Rule Evaluation

In order to consider only the most interesting prediction rules, it is necessary to evaluate the rules by computing an "interestingness" value for each of them. It is necessary to set up some criteria like frequency, confidence, and size which can be used to evaluate and rank the created prediction rules. The input of the rule evaluation procedure is a set of prediction rules and the result is a rate for each of the patterns.

Chapter 3

State of the Art

The two major areas which have been identified in the previous chapter are the requirements on the representation of dynamic scenes and on the learning of patterns from such a representation. The following two sections present the state of the art of relevant learning approaches and of adequate representation formalisms. In the subsequent section, existing approaches are discussed w.r.t. the identified requirements.

3.1 Learning Approaches

Referring to Langley [Lan96, p. 2], the "interest in computational approaches to learning dates back to the beginnings of artificial intelligence and cognitive science in the mid-1950s". Machine learning is a huge field with many different approaches also including, e.g., artificial neural networks, genetic algorithms, and many propositional learning approaches. The learning task asks for special representational abilities as in principle arbitrary relations between objects can be assigned. Subsymbolic and propositional learning approaches cannot be used directly here as it is required to identify patterns which can represent relations in a comprehensible way. In order to apply these approaches, a transformation or flattening – i.e., creating a propositional representation from a relational one by introducing attributes for object tuples in relations (cf. [De 98]) – of the represented knowledge would have to be done which is not feasible in all cases. The following subsections present different approaches which are particularly suitable for the intended learning task or address some of its aspects based on the requirements. Particularly of interest are approaches that can deal with relational or temporal data, or can unsupervised identify association rules. Historical information about the field of machine learning can be found, for instance, in the machine learning overviews of Carbonell et al. [CMM83] and Langley [Lan96, Chapter 1].

3.1.1 Supervised Inductive Logic Programming

Propositional learners like, for instance, *AQ, C4.5, CN2*, and *RIPPER* [Mic69, Qui93, CN89, Coh95] just learn attribute-value rules or decision trees. Inductive learning of first-order rules that contain variables is often called Inductive Logic Programming (ILP) [Mit97]. The first order representation leads to a much higher expressiveness compared to propositional learners. Some problems among others of propositional learning approaches are [De 98]:

- If different objects have to be observed, properties of both objects must be represented by separate attributes. If two objects switch places in the order, this might lead to problems as it cannot be represented that there should exist one object with certain properties.
- If relations between objects have to be taken into account, these must be flattened by inserting an attribute for each object pair (or tuple with higher arity) where the relation might exist. This can lead to many false facts.
- Another problem is that a variable number of objects cannot be handled properly. This is due to the fact that the number of objects must be known in advance in order to set up the attributes and that the number of attributes can be very large if relations between many objects can exist.

ILP approaches avoid these problems as the first-order representation allows for formulating relations in a more convenient way and the availability of variables makes it possible to deal with a varying number of objects and changing orders of those. They meet the requirement to deal with relational representations and are thus important approaches to be discussed here. In the following, a selection of relevant ILP systems and techniques is presented.

Sequential Covering

In sequential covering approaches, the set of examples is covered sequentially by rules, i.e., a set of rules covering the positive examples (and preferably no negative ones) is created incrementally. An early example for sequential covering is the AQ algorithm by Michalski [Mic69, Mic73a]; the most recent version of the AQ family is *AQ21* [KM05, Woj04]. Due to its relevance to other approaches presented in this thesis, the AQ algorithm is introduced briefly. The following simplified description is adapted from Kubat et al. [KBM97]:

1. Divide all examples into subsets PE and NE for positive and negative examples
2. Choose one example (possibly at random) from PE as seed

3. Find a set of maximally general rules (called *star*) covering the seed but not covering any example from NE

4. Select the desired rule in the star according to a preference criterion

5. Stop, if all positive examples are covered by the new rule and the previously created rules. Otherwise select another uncovered seed from PE and go to 3.

FOIL is a system developed by Quinlan which learns (almost[1]) Horn clauses from relational data [Qui90]. As it is interesting for the approach presented in this thesis, the *FOIL* algorithm is shown in Algorithm 1 (adapted from [Mit97, p. 286]). *FOIL* extends attribute-value learning systems (cf. [Mic69, Mic73a]) to a first-order representation.

FOIL is a supervised approach as target concepts, positive and negative examples are given to the algorithm. *FOIL* performs a sequential covering algorithm where rules are created and added to a disjunctive hypothesis until all positive examples are covered. The outer loop of the algorithm generates a new rule and removes all covered positive examples in each step. This outer loop is similar to the *AQ* algorithm [KBM97]. In the inner loop, a *Prolog* clause of the following form is created [Qui90, p. 244]:

$$P(X_1, X_2, \ldots, X_k) \leftarrow L_1, L_2, \ldots, L_n.$$

where $P(X_1, X_2, \ldots, X_k)$ is the rule head and $L_1, L_2, \ldots, L_n$ are the literals which form the rule preconditions [Mit97].

In the inner loop, a general-to-specific search is performed where literals are added to the rule until no negative example is covered any more. In order to guide the generation of single rules, *FOIL* uses an information-based estimate like *ID3* [Qui90]. The computation of the *FOIL* gain value of adding a literal L to rule R is [Mit97, p. 289]:

$$Foil\_Gain(L, R) = t \left(log_2 \frac{p_1}{p_1 + n_1} - log_2 \frac{p_0}{p_0 + n_0} \right)$$

with p_0 and n_0 as the number of positive and negative bindings of the rule R and p_1 and n_1 are the corresponding bindings of R'. R' is the rule which is created if literal L is added to R. t is the number of still covered positive bindings of R after adding L to R.

FOIL is not incremental [Qui90, p. 264] and there is no explicit representation for the temporal dimension given, i.e., temporal information had to be represented by additional arguments in the relations.

[1]Actually, the learned sets of first-order rules are just similar to Horn clauses because on the one hand they do not allow literals to contain function symbols, but on the other hand, they are more expressive than Horn clauses as they allow literals in the rule body to be negated [Mit97].

Algorithm 1 FOIL

Input: $Target\_predicate, Predicates, Examples$
Output: $Learned\_rules$

$Pos \leftarrow$ those $Examples$ for which the $Target\_predicate$ is $True$.
$Neg \leftarrow$ those $Examples$ for which the $Target\_predicate$ is $False$.
$Learned\_rules \leftarrow \{\}$
while Pos **do**
  /* Learn a new rule */
  $NewRule \leftarrow$ the rule that predicts $Target\_predicate$ with no precondition
  $NewRuleNeg \leftarrow Neg$
  while $NewRuleNeg$ **do**
    /* Add new literal to specialize $NewRule$ */
    $Candidate\_literals \leftarrow$ generate candidate new literals for $NewRule$, based on $Predicates$
    $Best\_literal \leftarrow arg\max_{L \in Candidate\_literals} Foil\_Gain(L, NewRule)$
    Add $Best\_literal$ to preconditions of $NewRule$
    $NewRuleNeg \leftarrow$ subset of $NewRuleNeg$ that satisfies $NewRule$ preconditions
  end while
  $Learned\_rules \leftarrow Learned\_rules + NewRule$
  $Pos \leftarrow Pos \setminus \{$members of Pos covered by $NewRule\}$
end while
return $Learned\_rules$

Inverse Entailment

Muggleton's *Progol* uses the idea that induction can be seen as the inverse of deduction as first stated by Stanley Jevons in the 19th century [Mug95]. Muggleton introduces the mode-directed inverse entailment (MDIE) where the single most specific hypothesis is created which in combination with background data entails the observed data [Mit97]. This hypothesis can be used to limit the search through the hypothesis space. This is also a supervised learning approach.

The algorithm which is realized in the *Progol* system can be summarized by the following steps (adapted from [Mit97, p. 300]:

- The user specifies "mode declarations" which define a restricted language of first-order expressions for the hypothesis space H. Here, the predicate and function symbols to be considered and their types and formats of arguments are stated.

- *Progol* uses a sequential covering algorithm (similar to the AQ algorithm; see [Mug95]) to learn a set of expressions from H that cover the data. For each uncovered example $\langle x_i, f(x_i)\rangle$ the (approximately) most specific hypothesis h_i is searched within H such that $(B \wedge h_i \wedge x_i) \vdash f(x_i)$ where B denotes the

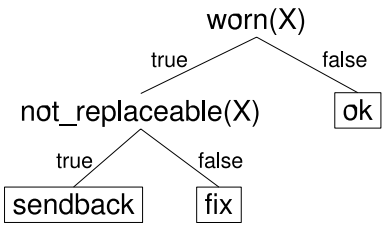

Figure 3.1: Example of a learned first-order decision tree

background knowledge [Mug95]. A user-defined parameter determines how many applications of the resolution rule are done.

- In the next step, *Progol* performs a general-to-specific search bounded by the most general possible hypothesis and the specific bound h_i. Within this bound the hypothesis with minimum description length (i.e., minimum number of literals) is computed. The search is done by an A*-like heuristic [Mit97].

The input to *Progol* is a *Prolog* program with a set of types (and valid values), background knowledge, positive and negative examples for the predicate to learn, and mode declarations which guide the learning process. The output is a set of *Prolog* clauses covering the positive examples. For more details about *Progol* see the original papers and tutorial by Muggleton and Firth [Mug95, Mug97, MF01].

Induction of First-Order Decision Trees

Blockeel and De Raedt [BD98] present an approach to top-down induction of first-order decision trees. Their system *TILDE* is a first-order extension of Quinlan's *C4.5* (see [Qui93]). Blockeel and De Raedt [BD98, p. 287] define a first-order logical decision tree as a "*binary* decision tree in which (1) the nodes of the tree contain a conjunction of literals, and (2) different nodes may share variables, under the following restriction: a variable that is introduced in a node (...) must not occur in the right branch of that node." An example for such a decision tree (adapted from [BD98, p. 288]) can be seen in Fig. 3.1.

The decision-tree induction algorithm in the system *TILDE* behaves like *C4.5*. The only difference is in the computation of the set of tests in the nodes [BD98]. Here, a refinement operator under θ-subsumption is used. A θ-subsumption between two clauses c_1 and c_2 exists if (and only if) there is a variable substitution θ such that $c_1\theta \subseteq c_2$ where $\subseteq$ is the subsumption symbol (cf. [BD98, p. 292]). Top-down induction of decision trees is a divide-and-conquer approach (cf. [KBM97, p. 17]). It is started with an empty decision tree and nodes with decisions are added one after the other. In each node, a decision is inserted that splits the examples, and the

new nodes are processed recursively until only examples of one class are left over or another stop criterion is satisfied.

Similar to the previous two approaches, *TILDE* is a supervised, non-incremental learning algorithm with no explicit representation of the temporal dimension. In experiments, Blockeel and De Raedt [BD98] show that *TILDE* is competitive in terms of efficiency and accuracy with *Progol* and *FOIL*.

Learning Description Logics

The learnability of description logics (DL) is studied by Cohen and Hirsh [CH92, CH94]. Cohen and Hirsh [CH92] define a simple DL "CoreClassic" which is a subset of *CLASSIC* and analyze its learnability. They show that the full logic cannot be tractably learned, but some restrictions (acyclic concept graphs and restriction of the number of vertices in a concept graph) enable tractable learning. They define an algorithm which learns from positive examples. It returns the least concept ("least common subsumer") that subsumes all examples [CH92]. In subsequent work, Cohen and Hirsh [CH94] present extensions to learn the DL *C-CLASSIC* which has more practical relevance. In this work, they also present algorithms that learn from individuals (represented as very specific concepts) and that learn disjunctions of descriptions.

3.1.2 Frequent Pattern Mining and Association Rule Mining

In contrast to the approaches presented in the previous section, association rule mining is unsupervised and identifies frequent rules in data. The created rules are described in a comprehensible way as desired in this thesis. The goal is usually to mine frequent association rules which exceed a certain confidence and frequency threshold in data. For this work, two extensions of association rule mining are particularly of interest: sequential and relational association rule mining as they can deal with special requirements for learning in dynamic and complex environments. Nevertheless, it is important to know basic algorithms like *Apriori* [AS94] in order to understand the extensions. Thus, initial work on rule mining in item sets is introduced first.

Mining Association Rules in Item Sets

Agrawal et al. [AIS93] introduce the problem of association rule mining. The input to the learning algorithm is a database with transactions, e.g., different products bought by customers. The output is a set of rules about associations found in the data, e.g., which products have been bought together. An example for such a mined rule is: 80% of the customers who have bought the albums *Abbey Road* and *Rubber*

ID	A	B	C	D
1	x	x	x	x
2	x	x	x	
3	x	x	x	x
4		x	x	
5	x	x	x	x
6	x	x	x	x

Table 3.1: Item set example

Soul have also bought *The White Album*. This approach is not incremental and cannot deal with relational or temporal representations.

Association rule mining can be divided into two subproblems [AIS93, AS94]:

1. Finding all large item sets and
2. Generating the association rules.

In the first step, all combinations of items whose *support* exceeds some threshold *minsupport* are created. The *support* is the number of transactions that contain the item set. The combinations of items with $support \geq minsupport$ are called "large itemsets"; the combination with frequency below *minsupport* are called "small itemsets" [AIS93, p. 208].

The second step uses the large item sets to create association rules. Following the notation of Agrawal and Srikant, if $ABCD$ and AB are large item sets, the rule $AB \Longrightarrow CD$ is an association rule with confidence $conf = \frac{support(ABCD)}{support(AB)}$. If $conf \geq minconf$, the rule will be kept. As $ABCD$ – the most restricted part of the rule – is a large itemset, the support of the rule also exceeds *minsupport* [AS94]. If the transactions in Table 3.1 are taken as an example, the supports of the two item sets are: $support(AB) = 5$ and $support(ABCD) = 4$, i.e., the confidence of the rule is: $conf(AB \Longrightarrow CD) = \frac{4}{5} = 0.8$.

Agrawal and Srikant [AS94] introduce fast algorithms for association rule mining: *Apriori*, *AprioriTid*, and *AprioriHybrid* (a hybrid of the other two). Due to the importance – also for recent approaches – the *Apriori* algorithm is introduced briefly (cf. [AS94]). It is shown in Algorithm 2. L_i holds the large itemsets and C_i the candidates of the i-th step. In the first step, all large 1-itemsets are generated by counting occurrences of transactions which include this item. In the next steps, candidates for large item sets are generated based on the large item sets in the previous step $(k-1)$. In the candidate generation function, large item sets of the previous steps are combined and then pruned (see Algorithm 3; adapted from [AS94]). Pruning means that all item sets are deleted which have a $(k-1)$-subset that is not in L_{k-1}. Then in a loop all transactions are tested and the support

Algorithm 2 Apriori [AS94]

Input: Transaction database $\mathcal{D}$; minimal support *minsup*
Output: All frequent itemsets

```
1: L_1 = {large 1-itemsets}
2: for (k = 2; L_{k-1} ≠ ∅; k++) do
3:    C_k = Apriori − gen(L_{k-1}) /* new candidates */
4:    for all transactions t ∈ D do
5:       C_t = subset(C_k, t) /* candidates contained in t */
6:       for all candidates c ∈ C_t do
7:          c.count++
8:       end for
9:    end for
10:   L_k = {c ∈ C_k | c.count ≥ minsup}
11: end for
12: return ⋃_k L_k
```

Algorithm 3 Apriori-gen [AS94]

Input: Last level frequent itemsets L_{k-1}
Output: Next level candidates C_k

```
1: /* Join step */
2: insert into C_k
3: select p.item_1, p.item_2, ..., p.item_{k-1}, q.item_{k-1} from L_{k-1}p, L_{k-1}q where
   p.item_1 = q.item_1, ..., p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1};
4: /* Prune step */
5: for all itemsets c ∈ C_k do
6:    for all (k-1)-subsets s of c do
7:       if (s ∉ L_{k-1}) then
8:          delete c from C_k;
9:       end if
10:   end for
11: end for
12: return C_k
```

values for the candidates are computed. All candidates with a support exceeding the *minsup* threshold are kept as large item sets.

AprioriTid also uses the *apriori-gen* function for identifying candidate item sets. The difference to *Apriori* is that it does not use the database $\mathcal{D}$ for counting the support after the initial pass. Instead of the database, it uses the set $\overline{C}_k$. The members of $\overline{C}_k$ are tuples $\langle TID, \{X_k\}\rangle$ where the X_k are potentially large item sets present in transaction TID [AS94]. The advantage is that transactions which do not contain any candidate item set can be ignored. Thus, the size of $\overline{C}_k$ may be smaller than the number of transactions. This is especially expected at high values

for k [AS94]. Nevertheless, for small k the single entries might be larger than the corresponding transaction. *AprioriHybrid* combines *Apriori* and *AprioriTid*. It uses *Apriori* in the beginning and switches to *AprioriTid* when it is expected that $\overline{C}_k$ fits in the memory. Details can be found in [AS94].

Following the work of Agrawal et al. [AS94], further frequent itemset mining algorithms, data structures and implementations are presented (e.g., [ZPOL97]). Recent workshops show that this is still an active research area (e.g., [BGZ04]). A good survey article of the field of association mining can be found in [CR06].

Sequential Association Rule Mining

The original association rule mining algorithms only take itemsets without any order into account. If the order of items is important (due to temporal or spatial arrangements), sequential association rule mining approaches must be used. These approaches are highly relevant to this thesis as one of the requirements is to handle temporal data. The goal is to find sequential patterns, e.g., in order to identify a pattern in what order customers usually rent videos ("Star Wars" before "Empire Strikes Back" and "Return of the Jedi"; cf. [AS95]). Thus, the result of the sequential association rule mining is a set of frequent sequences of itemsets. Two surveys covering sequential pattern mining can be found in [ZB03, LS06].

Agrawal and Srikant [AS95] present three algorithms for mining sequential patterns: *AprioriSome*, *AprioriAll*, and *DynamicSome*. They assume that customer transactions in the database $\mathcal{D}$ consist of a customer ID, a transaction time, and the items purchased in the transaction. A sequence is defined as an ordered list of itemsets. The fraction of customers who bought an itemset defines the support for this itemset. Itemsets with minimum support are also called large itemsets. A large sequence is a list of large itemsets.

Agrawal and Srikant [AS95] split the problem of sequential pattern mining in different phases: sort phase, litemset phase, transformation phase, sequence phase, and maximal phase. In the sort phase, the database is sorted by customer ID and for each customer w.r.t. the transaction time. The identification of large itemsets is similar to the work in [AIS93, AS94] with an adapted definition of support: The support is here not the fraction of the transactions where the itemset is present but the fraction of the customers where it is present in one of their transactions [AS95]. For efficiency reasons each large itemset is mapped to an integer value, i.e., each itemset is treated as a single entity. In the transformation phase, the transactions are replaced by the large itemsets contained in them. The search for sequences is performed in the sequence phase. In the last phase, the maximal sequences are identified by deleting all sequences which are subsequences from the set of sequences.

Two families of algorithms are presented in [AS95]: *count-all* and *count-some*. In

the first case, all sequences are counted. This includes the non-maximal ones which must be pruned out in the maximal phase. In the context of sequential pattern mining, a sequence s is maximal w.r.t. a set of sequences "if s is not contained in any other sequence" [AS95, p. 3]. In the second case (*count-some*), longer sequences are counted first in order to avoid counting non-maximal sequences. *AprioriAll* is an example for a *count-all* algorithm and *AprioriSome* and *DynamicSome* are examples for *count-some* algorithms.

In *AprioriAll*, in each pass the large sequences of the previous pass are used to create new large sequences. Only those candidates with maximum support are kept. *AprioriSome* consists of two phases: the forward phase and the backward phase. In the forward phase, only sequences of certain lengths are counted. It is determined by a function $next(k)$ what sequence length to process in the next step. In the backward phase, the sequences for the skipped lengths are counted. In advance, all sequences are deleted that are contained in other large sequences. *DynamicSome* also only counts sequences of certain lengths in the forward phase. All multiples of a *step* value are counted. In the different steps of the algorithm, sequences are generated by joining sequences of the lower length steps, e.g., if $step = 3$ sequences of length 9 are created by sequences of length 6 and 3 [AS95].

Srikant and Agrawal have generalized this approach in [SA96]. The extended variant allows for adding time constraints in order to specify minimum and maximum time periods between elements in the pattern. The items in an element of a sequential pattern do not need to come from the same transaction anymore; it is sufficient if they belong to transactions within a specified time window. Another extension is the introduction of item taxonomies and thus the possibility to mine generalized patterns which include items from different levels of a taxonomy. Their new algorithm *GSP* (Generalized Sequential Pattern) can handle the extensions. It is also *Apriori*-based and can be divided into the candidate generation and frequent pattern generation phases [ZB03]. The candidate generation phase also consists of a join phase – where patterns of the previous level are combined – and a prune phase – where candidates are removed if they have a "contiguous $(k-1)$-subsequence" which is infrequent w.r.t. the minimal support [SA96]. For support computation in the frequent pattern generation phase it must be checked if a data sequence contains a pattern. In *GSP*, this is checked in so called forward and backward phases in order to find a match satisfying the time constraints. In order to discover rules w.r.t. taxonomies, the itemsets of a transaction are extended by their super items in the taxonomy. As optimization, Srikant and Agrawal [SA96] remove items that do not occur in any pattern and do not count patterns with elements that contain both, an item and one of its ancestors.

Zhang et al. [ZKYC01] address the problem in *GSP* that the database has to be scanned as often as items occur in the longest frequent sequence. They extend the candidate generation function of *GSP* and present the new two-stage algorithm

MFS (Mining Frequent Sequences). Instead of only checking candidates of the same length, as it is the case in *GSP*, their approach allows for checking candidates of various lengths in a database scan. *MFS* first computes a rough estimate of the set of frequent sequences which is later refined. Zhang et al. show in experiments that *MFS* outperforms *GSP* w.r.t. performance as costs for database scans can be saved. In later work, Zhang, Kao, and colleagues present algorithms for an incremental update of frequent sequences [ZKCY02, KZYC05]. They present the algorithms *GSP+* and *MFS+*. The basic idea is that if the support of a frequent pattern in an old database is known, the new support can be computed by scanning the newly added (and recently removed) sequences. Infrequent patterns (w.r.t. the old database) must only be checked in the (possibly large) unchanged part of the database if the frequency in the new part is large enough and the deleted part is small enough [KZYC05].

PrefixSpan (Prefix projected sequential pattern mining) is another approach to sequential pattern mining [PHMA+01]. It reduces the candidate subsequence generation efforts by exploring a prefix projection and thus also reduces the database sizes for future scans which leads to more efficient processing. Pei et al. [PHMA+01] discuss three different kinds of projections: level-by-level, bi-level, and pseudo-projection. In the level-by-level projection, the algorithm starts with the large 1-itemsets and creates a prefix projection for each large itemset where the projected database only consists of the postfixes. In the projected database, the frequent 2-itemsets are mined and then the database is projected again for these itemsets. This procedure is performed recursively until the projected database is empty or no frequent patterns can be generated. In bi-level projection, a (triangular) matrix is created that holds the supports of all level two sequences. The use of this matrix reduces the number of projected databases and thus requires less space. The pseudo-projection does not create a (physical) projection database but represents a projection by a pointer to the sequence and the offset of the postfix in the sequence. The pseudo-projection can only be used if the database can be held in the main memory. The advantages of *PrefixSpan* are that it does not create (and test) any candidate sequence that does not occur in a projected database but only "grows longer sequential patterns from the shorter frequent ones" [PHMA+01, p. 221] and that the projected databases get smaller with each step. Pei et al. show in experiments that their approach is faster than *GSP* while it still mines the complete set of frequent patterns. Pseudo projection has the best performance and bi-projection outperforms the level-by-level projection [PHMA+01].

Pinto et al. [PHP+01] combine *PrefixSpan* with *BUC*, a multi-dimensional pattern mining algorithm, in order to address multi-dimensional sequential pattern mining. They develop three algorithms: *UniSeq*, *Dim-Seq*, and *Seq-Dim*. While the first one "treats all dimension values as sequential items" and only uses *PrefixSpan*

the latter two apply both the sequential and the multi-dimensional pattern mining algorithm. *Dim-Seq* identifies frequent dimension value combinations first and then applies the sequential pattern mining algorithm to the sequences that satisfy the dimension value combination. In the other case, the sequential pattern mining algorithm is applied first and then for each sequential pattern the frequent dimension patterns are created. Pinto et al. [PHP+01] compare the algorithms and conclude that *UniSeq* is the best choice if the total number of sequential items and other dimension values is low. Depending how sparse or dense the dataset is w.r.t. occurring sequential patterns and dimension value combinations, either *Dim-Seq* or *Seq-Dim* is advantageous. *Dim-Seq* is better for sparse dimension value combinations and *Seq-Dim* if the dataset is sparse w.r.t. the sequential patterns. If both factors are dense, *Seq-Dim* is referred to as the better alternative [PHP+01].

Another extension of *PrefixSpan* to sequential pattern mining with time intervals is proposed by Hirate and Yamana [HY06b, HY06a]. Their representation allows "transaction occurrence time intervals", i.e., itemsets with a temporal extension can be represented. The approach allows for distinguishing patterns "with any time interval that are multiples of a used-defined base interval" [HY06b, p. 775]. It is possible to mine patterns like if event A occurs the probability that it will re-occur in the time interval between one or two days later with some probability p.

Zaki [Zak01] proposes another approach to sequential pattern mining. His algorithm *SPADE* (sequential pattern discovery using equivalence classes) is based on a vertical database representation, i.e., each sequence is associated with a list of objects (and their timestamp) "in which it occurs" [Zak01, p. 32]. The frequent patterns can be enumerated through temporal joins or intersections on ID lists. *SPADE* usually needs only three database scans in order to mine the frequent sequences. Two scans are needed for finding the frequent 1-sequences and 2-sequences and the third scan is required to mine all remaining frequent sequences. In *SPADE*, the task of sequence mining is decomposed so that only sub-problems which can be held in the main memory must be processed. A lattice-theoretic approach is used in order to decompose the search space into sub-lattices. The decomposition is done by an equivalence relation where all sequences with a common item prefix are put into an equivalence class. For the actual search of frequent sequences, Zaki compares two methods for enumeration: breadth-first search (BFS) and depth-first search (DFS). While BFS needs more memory to hold all the equivalence classes of a level it has more information available to prune candidate sequences with infrequent parents. The k-item patterns are generated by joining the frequent $(k-1)$ patterns with the common $(k-2)$-prefix [Zak01]. Experiments show that *SPADE* scales well, for instance, w.r.t. the number of sequences and the number of events per sequence, and that it outperforms *GSP* by a factor of two. Parthasarathy et al. [PZOD99] present an extension of *SPADE* to incremental and interactive sequence mining where it is

avoided to rerun the algorithm on the entire dataset if the database is updated or some mining parameters are changed.

Garofalakis et al. [GRS99] present *SPIRIT* – an approach to sequential pattern mining using regular expression constraints. They address the problem that most approaches provide only minimal support for frequent sequential pattern mining as "mechanism (...) for specifying patterns of interest" [GRS99, p.223]. They propose the utilization of regular expressions for a user-defined focus of interest. In their work, they introduce four algorithms to exploit the information provided by the regular expression: *SPIRIT(N)* adapts the *GSP* algorithm [SA96] by requiring each element of a candidate sequence to appear in the regular expression. *SPIRIT(L)* uses an automaton based on the regular expression. All candidates that are not legal w.r.t. any state of the automaton are pruned. The third algorithm *SPIRIT(V)* is even more restrictive and removes each candidate that is not valid w.r.t. any state of the automaton. *SPIRIT(R)* uses the automaton in order to generate the candidates by enumerating the paths in the automaton. Garofalakis et al. perform experiments on synthetic and real-life data and conclude that *SPIRIT(V)* can be seen as the "overall winner" as it provides good performance over different regular expression constraints. *SPIRIT(R)* outperforms the other algorithms for highly restrictive regular expression constraints.

The mining of indirect associations is addressed by Chen et al. [CBL06]. Indirect association patterns are infrequent item pairs which are indirectly associated with a common mediator itemset which is frequent in conjunction with each of the separate items. Chen et al. take the lifespan of the transactions into account in their temporal indirect association mining algorithm *TMG-Growth*. The algorithm first generates a frequency graph in order to identify infrequent item pairs and potential mediators for these pairs. For all infrequent item pairs a mediator graph is constructed and used to generate a complete set of mediators for the item pair.

A recent approach by Kum et al. [KCW06] addresses sequential pattern mining in multi-databases. They propose an approach to approximate sequential pattern mining where patterns are identified that are approximately shared by many sequences. In their algorithm, *ApproxMAP* sequences are first clustered by similarity and then patterns are mined from each cluster. The local summaries of the mining processes in the different databases are then used for the global mining process.

Discovery of Frequent Episodes

Mannila et al. [MTV97] address the discovery of frequent episodes in event sequences. In their work, an event sequence is defined as a sequence of events with an associated

time of occurrence. More formally, an event sequence is defined as a triple (s, T_s, T_e). T_s and T_e are the start and end time of the complete sequence and s is an ordered sequence of events [MTV97]:

$$s = \langle (A_1, t_1), (A_2, t_2), \ldots, (A_n, t_n) \rangle$$

It holds that $A_i \in E$ for all $i = 1, \ldots, n$ where E is a set of event types. Furthermore, $t_i \leq t_{i+1}$ for all $i = 1, \ldots, n-1$ and $T_s \leq t_i < T_e$ for all $i = 1, \ldots, n$.

A time window defines what is still counted as an episode, i.e., how close the occurring events in an episode must be. Different events can occur serially or in parallel in episodes. Mannila et al. define an episode by a triple $(V, \leq, g)$ where V is a set of nodes and $\leq$ is a partial order on V. $g : V \rightarrow E$ maps each node to an event type. They also define subepisode ($\preceq$) and superepisode ($\succeq$) relationships. If $\alpha \preceq \beta$ and $\beta \not\preceq \alpha$, it is written as $\alpha \prec \beta$.

The frequency of an episode is defined by Mannila et al. [MTV97] as:

$$fr(\alpha, s, win) = \frac{|\{w \in \mathcal{W}(s, win) | \alpha \text{ occurs in } w\}|}{|\mathcal{W}(s, win)|}$$

where $\mathcal{W}(s, win)$ is the set of all windows w on the event sequence s with width win. An episode α occurs if there is an injective mapping from nodes of α to events of s. An episode rule is defined as an expression $\beta \Rightarrow \gamma$, where β and γ are episodes with $\beta \preceq \gamma$. The confidence of the episode rule is computed by the fraction $\frac{fr(\gamma, s, win)}{fr(\beta, s, win)}$.

Mannila et al. introduce two algorithms for finding frequent episodes: *WINEPI* and *MINEPI*. *WINEPI* works in a similar way as *Apriori* as it creates candidates for frequent episodes based on the episodes in the previous step. The generated candidates are (more special) superepisodes of the episodes in the step before with only one additional node. Only the frequent patterns are kept after each step, i.e., the patterns whose frequency exceeds a threshold.

The alternative approach *MINEPI* is based on minimal occurrences of episodes. Information about the locations of the minimal occurrences are stored for each frequent episode. With this information the locations of minimal occurrences of an episode α can be computed "as a temporal join of the minimal occurrences of two subepisodes of α" [MTV97]. This has the advantage that frequencies and confidences for episodes with varying window sizes can be computed quickly. For more details see [MTV97].

Mining Patterns from Interval-based Data

Another approach to learning temporal rules is presented by Höppner [Höp01, Höp03]. Here, rules are learned from a sequence of labeled time intervals. Similar to the work of Mannila et al. [MTV97] it is searched for frequent local patterns.

The difference is that in this work events (called "states" by Höppner) have a temporal extension and interval relationships between events can be used in patterns. Allen's interval logic [All83] is used to describe these temporal relations between events. A "state sequence" (according with "event sequence" in the notation of Mannila et al. [MTV97]) is defined as:

$$(b_1, s_1, f_1), (b_2, s_2, f_2), \ldots$$

with $b_i \leq b_{i+1}$ and $b_i < f_i$. s_i are the different states and they hold in the interval $[b_i, f_i)$.

Temporal patterns of size n are defined by a pair (s, R) where $s : \{1, \ldots, n\} \rightarrow \mathcal{S}$ maps index i to the corresponding state and $R \in \mathcal{I}^{n \times n}$ defines the interval relationships [Höp01]. Similar to the subepisode relationship in [MTV97] a subpattern relation $\sqsubseteq$ is defined. A sliding window is used in order to find occurrences of temporal patterns in the sequence.

The discovery of temporal rules is split into candidate generation, support estimation, and rule generation. For candidate generation, the same approach as for association rule mining is used [AS94]. In order to create new candidate patterns (at level $k + 1$), two frequent patterns of the previous step are joined which have the same $(k - 1)$-pattern prefix. Due to a normalized representation of patterns when combining such two patterns there is only one degree of freedom left. This is the temporal interval between the two events which have not been combined in the single patterns in the step before. The normalized patterns additionally allow for reducing the number of interval relations to seven (instead of 13). The application of the transitivity table [All83] leads to a further complexity reduction [Höp01].

Winarko and Roddick [WR05] follow Höppner's definition of temporal patterns [Höp01]. They extend the *MEMISP* algorithm (memory indexing for sequential pattern mining; [LL02]) for mining temporal patterns from interval data. If the database fits into the main memory, only one database scan is required. During this scan, the support of each state is computed and frequent 1-patterns are generated. After the generation of the index set for the frequent 1-patterns a recursive "find-then-index" strategy is applied in order to find all temporal patterns. Winarko and Roddick also introduce a maximal time gap constraint in order to avoid the generation of insignificant patterns where the time gap is too long.

Tatavarty and Bhatnagar [TB05] present another approach that creates patterns with relations between temporal intervals. They address the mining of temporal association rules from frequent patterns in multivariate time series. The input to learning is a multi-dimensional time series, i.e., a set of item sequences. Their methodology first handles each dimension separately and mines frequent substring patterns in each dimension. Then the frequent patterns are clustered in order to

create equivalence classes. In the next step, temporal dependencies (followed by, contains, overlaps) between the clusters are mined.

Relational Pattern and Association Rule Mining

The mining of relational association rules is another extension to the initial mining of frequent itemsets. The focus is to deal with more complex representations than propositional logic (cf. [Stu04]). These approaches can also be seen as inductive logic programming because logical programs are derived. The difference to the approaches presented in Section 3.1.1 is that unsupervised learning is applied, i.e., no examples for concepts are provided to the learner. Besides meeting the requirement of being unsupervised, these approaches can deal additionally with relations which is also an important aspect for this thesis.

Dehaspe and Toivonen combine association rule mining algorithms with ILP techniques. Their system *WARMR* is an extension of *Apriori* for mining association rules over multiple relations [DT99, DT01]. Dehaspe and Toivonen [DT99, p. 19] introduce the notion of *query extensions* which are the "first order equivalent" of association rules. The created rules are – similar to transaction-based association rules – also expressions of the form $X \Rightarrow Y$, but here X and Y are sets of logical atoms of the form $p(t_1, \ldots, t_n)$ where each term t_i is a variable or a function (including constants as functions with an arity of zero). This more expressive representation allows for discovering rules like (cf. [DD97]):

$$\text{likes}(KID, A), \text{has}(KID, B) \Rightarrow \text{prefers}(KID, A, B)$$

Similar to the original association rule mining, the task is to discover all rules with a confidence about a minimum threshold *minconf* and a support above minimum support *minsup*. The difference is that the database is a deductive relational database instead of a simple transaction table.

Due to the importance of *WARMR* for the work presented here, the algorithms are shown in Algorithms 4, 5, and 6 (adapted from [DT99, p. 15-18]). The *WARMR* algorithm is based on *Apriori*, but it exploits the lattice structure of atomsets instead of itemsets. The basic algorithm of *WARMR* is almost identical to *Apriori*. In the beginning, the candidate query set is initialized with the most general query and the sets of frequent and infrequent queries are initialized with empty sets. As long as query candidates exist, the frequencies for these candidates are generated. Depending on the frequency the queries are added to the set of frequent or infrequent queries. The next level candidates are then generated from the current candidate set. The *WARMR-gen* algorithm applies two pruning conditions where all queries are removed that either have some infrequent subsuming query or are equivalent to

Algorithm 4 WARMR [DT99]

Input: Database $\mathbf{r}$; WRMODE language $\mathcal{L}$ and key; threshold $minfreq$
Output: All queries $Q \in \mathcal{L}$ with $frq(Q, \mathbf{r}, key) \geq minfreq$

```
Initialize level d := 1
Initialize the set of candidate queries Q_1 := {? − key}
Initialize the set of infrequent queries I := ∅
Initialize the set of frequent queries F := ∅
while Q_d not empty do
    Find frq(Q, r, key) of all Q ∈ Q_d using WARMR-EVAL
    Move the queries ∈ Q_d with frequency below minfreq to I
    Update F := F ∪ Q_d
    Compute new candidates Q_{d+1} from Q_d, F, and I using WARMR-GEN
    Increment d
end while
Return F
```

Algorithm 5 WARMR-EVAL [DT99]

Input: Database $\mathbf{r}$; set of queries $\mathcal{Q}$; WRMODE key
Output: The frequencies of queries $\mathcal{Q}$

```
for each query Q_j ∈ Q do
    Initialize frequency counter q_j := 0
end for
for each substitution θ_k ∈ answerset(? − key, r) do
    Isolate the relevant fraction of the database r_k ⊆ r
    for each query Q_j ∈ Q do
        if query Q_j θ_k succeeds w.r.t. r_k then
            Increment counter q_j
        end if
    end for
end for
for each query Q_j ∈ Q do
    Return frequency counter q_j
end for
```

one of the previously generated queries [DT99].

Another approach to learning from multi-relational data has been presented by Wrobel [Wro97]. He introduces the *MIDOS* (multi-relational discovery of subgroups) algorithm in order to identify statistically unusual subgroups in multi-relational databases. The mined patterns consist of a conjunction of first-order literals and comparisons on attributes (nominal attributes are restricted to an equality comparator). In order to avoid the "combinatorial explosion of this hypothesis space"

Algorithm 6 WARMR-GEN [DT99]

Input: WRMODE language $\mathcal{L}$; infrequent queries $\mathcal{I}$; frequent queries $\mathcal{F}$; frequent queries $\mathcal{Q}_d$ for level d

Output: Candidate queries $\mathcal{Q}_{d+1}$ for level $d+1$

1: Initialize $\mathcal{Q}_{d+1} := \emptyset$
2: **for** each query $Q_j \in \mathcal{Q}_d$ and for each immediate specialization $Q'_j \in \mathcal{L}$ of Q_j **do**
3: Add Q'_j to $\mathcal{Q}_{d+1}$ unless:
4: (i) Q'_j is more specific than some query $\in \mathcal{I}$, or
5: (ii) Q'_j is equivalent to some query $\in \mathcal{Q}_{d+1} \cup \mathcal{F}$
6: **end for**
7: Return $\mathcal{Q}_{d+1}$

[Wro97, p. 81], *foreign key links* are used as declarative bias, i.e., these links specify the paths where relations can be joined together. Wrobel uses an evaluation function in order to measure the interestingness of subgroup candidates. The *MIDOS* algorithm itself "is a top-down, general-to-specific search that can use breadth-first, depth-first, or best-first or parallel control regimes" [Wro97, p. 84].

Koperski and Han [KH95] present an approach for the discovery of spatial association rules. They define a spatial association rule as a rule where "at least one of the predicates (...) is a spatial predicate"[KH95, p. 52]. Among others, they give the following rule as an example. It indicates that 80% of the large towns in B.C./Canada which are close to the sea are also close to the border to the USA:

$$is\_a(X, large\_town) \wedge g\_close\_to(X, sea) \rightarrow g\_close\_to(X, us\_boundary)\ (80\%)$$

Although this rule is a relational representation Koperski and Han actually do not apply relational association rule mining. A "first-order counterpart of Koperski's method" [ML01, p. 19] will be discussed below. One distinctive characteristic of Koperski's and Han's approach is the approximate spatial computation and knowledge mining at multiple abstraction levels. Their algorithm applies approximate spatial computation which reduces the candidate set for association rules. The definition of conceptual hierarchies (e.g., "sea" is a kind of "water") allows for generating association rules at different levels of granularity.

The work of Malerba and Lisi [ML01] is inspired by Dehaspe's approach to mining association rules from multiple relations (see above and [DT99]). Similar to Koperski and Han they address spatial association rule mining. Their method can use background knowledge of the domain and exploit hierarchical structures of geographic layers. The algorithm is implemented in the *SPADA* (Spatial Pattern Discovery Algorithm) system which works on a deductive database system. In comparison to Koperski's work, the mining algorithm is performed on atomsets as

proposed by Dehaspe [DD97] and thus has a more expressive power w.r.t. the representation. Compared to *WARMR* it allows for exploiting ontological information while searching the pattern space [ML01].

Malerba and Lisi define a refinement operator under θ-subsumption which drives the search through the pattern space [ML01]. As all refinements of an infrequent pattern must also be infrequent, the search can be done without missing relevant patterns as it is also the case in *Apriori*. In a more recent work, Lisi and Malerba [LM04] introduce an upgrade of the learning system *SPADA* which can learn patterns from the language $\mathcal{AL}$-log. It is a hybrid knowledge representation language that integrates the description logic $\mathcal{ALC}$ and the deductive database language *DATALOG*.

Kaminka et al. [KFCV03] address the unsupervised learning of sequential behaviors of agents. They map the observations in the dynamic, continuous multi-variate world state to a sequence of atomic behaviors. Based on this time series of behaviors frequent subsequences are searched. In their work, they compare two different approaches: frequency counts and statistical dependency. They conclude that the latter method might be more suitable for the discovery of coordinated sequential behavior as it is able to reject frequent sequences whose behaviors only co-occurred by chance [KFCV03]. The frequent sequences are stored in a trie. A trie is a "tree-like data structure, which efficiently stores sequences so that duplicated sub-sequences are stored only once, but in a way it allows keeping a count of how many times they had appeared" [KFCV03, p. 118]. A similar approach using a trie for sequence representation has been presented by Huang et al. [HYC03]. A difference between the two approaches is that the one of Huang et al. can deal with co-occurring behaviors of different agents [HYC03].

Stumme [Stu04] combines relational association rules with formal concept analysis (FCA). Formal concept analysis is a method for the analysis of data. By applying FCA, data can be structured into formal abstractions of concepts (cf. [GW98]). One problem of relational association rules is the number of resulting rules including many uninteresting and redundant rules [Stu04]. Stumme uses concept lattices in order to create a condensed representation for frequent itemsets. According to [Stu04], other groups independently also came up with the idea of using these condensed representations of itemsets [PBTL98, ZO98]. The basic idea of condensed representations is to avoid redundant computation and representation of information. In the case of itemset mining, there can be different itemsets which represent the same transactions and thus have identical support values [Stu04]. It is possible to set up an equivalence relation Ψ on the set of itemsets. Two itemsets are equivalent w.r.t. a database if they are contained in the same transactions [Stu04].

Relational Temporal Pattern Mining

De Amo et al. [dFGL04] introduce an *Apriori*-based algorithm for "first-order temporal pattern mining". They use an extended transaction database where each entry consists of a timestamp, a client ID, an item ID, and a group ID. The group IDs are used in order to group a number of transactions of different customers. In their work, they define *multi-sequences* where the original database can be transformed w.r.t. the group IDs and each entry consists of a set of client sequences; time positions with no item are represented by a special "don't care" item denoted by $\perp$. The goal of the mining process is the identification of all frequent *multi-sequence patterns* which are a special case of multi-sequences. For each client there must exist exactly one concrete item ID at some time point (and $\perp$ elements at the remaining positions) and for each time point there must exist at least one concrete item ID. The representation is therefore restricted (e.g., no time gaps or multiple entries for the same client allowed) and it is not clear how redundancies are handled (e.g., if only the order of two clients is switched). De Amo et al. present two algorithms for temporal pattern mining: *PM* (*Projection Miner*) and *SM* (*Simultaneous Miner*). In *PM*, the patterns are decomposed in two propositional patterns in the candidate generation phase as well as the pruning phase and an adaptation of the *GSP* algorithm is used for mining. The second algorithm *SM* avoids such a conversion to the propositional representation and mines the patterns directly from the first-order representation. The algorithms are similar to other *Apriori*-based ones having a join and prune step; the differences are in counting the support for patterns and the candidate generation. The support is incremented for a group ID if the pattern is a so called "sub-multi-sequence" for this group. An experimental comparison of the two algorithms shows that *SM* is three times faster than *PM* [dFGL04]. In their more recent work, de Amo and Furtado [dF05] present the *MSP-Miner* algorithm for first-order temporal pattern mining with regular expression constraints similar to the work of Garofalakis et al. [GRS99].

Jacobs and Blockeel [JB01] apply the ILP association rule learner *WARMR* in order to mine shell scripts from Unix command shell logs. Log files of shells can be seen as a sequence of commands. Frequent patterns from such command sequences can be interpreted as shell scripts. The challenge is to deal with the arguments in commands and use variables in patterns in order to represent that the same argument (e.g., a file name) should be used by a different command. Jacobs and Blockeel use *WARMR* [DT99] for the generation of scripts and also present some methods for speedup by splitting up the learning task and using the so called *minimal occurrence algorithm*. Command sequences are represented by a *stub* relation with unique identifier, execution time, and command (e.g., `stub(1,2008,'cp')`) and *parameter* relations (e.g., `parameter(1,1,'file1')` and `parameter(1,2,'file2')`).

The learning task is split up by first finding the frequent sequences of commands and then taking the parameter information into account. The *minimal occurrence algorithm* takes into account the sequential information in the query generation process and can thus prune some of the patterns that cannot be frequent any more. It also utilizes the identifiers at which sequences of the previous step start for the calculation of occurrences of a new sequence [JB01].

Masson and Jacquenet [MJ03] address the mining of frequent logical sequences. They extend the *SPIRIT* system [GRS99] to discover logical sequences and introduce the *SPIRIT-LOG* algorithms. The major adaptations w.r.t. *SPIRIT* are done in the generation and pruning functions. Candidate generation is extended to handle logical sequences with variables and in the pruning step an inclusion test between logical sequences (including unification of variables) are developed. *SPIRIT-LOG* can only create patterns of contiguous predicates, i.e., no gaps are allowed in the sequential patterns. Furthermore, it is not possible to use background knowledge in the form of clauses (cf. [LD04]).

Lee and De Raedt [LD04, Lee06] introduce the logical language *SeqLog* for the representation of sequential logical data. The sequence itself is represented as a sequence of logical atoms. Additionally, it is possible to specify background knowledge as *DATALOG* style clauses. The mined patterns consist of a sequence of logical atoms. Two adjacent atoms in the pattern can be specified as direct (temporal) neighbors or allow for having other elements between them (denoted by the $<$ symbol). Lee and De Raedt introduce the mining system MineSeqLog which mines the borders of the solution space for an input sequence and a conjunction of monotonic and anti-monotonic constraints on the patterns [LD04]. Lee and De Raedt combine a level-wise algorithm with ideas from version spaces (see, e.g., [Mit97]). More details about this approach can be found in Chapter 4 as it is highly relevant to this thesis.

3.1.3 Similarity-based Approaches

The approaches presented so far are mainly based on logical representations and learning results in explicit hypotheses for classifiers or learned patterns. Another learning paradigm is the so called Instance-based Learning (IBL) where instances are used in order to classify unknown instances. These approaches are based on similarity measures between instances and are sometimes also called "lazy" learning because "they delay processing until a new instance must be classified" [Mit97, p. 230]. One example for an instance-based learning method is the k-nearest neighbor algorithm [CH67]. Further descriptions of (non-relational) IBL algorithms can be found, e.g., in [AKA91, Aha92]. Similarity measures can also be used for unsuper-

vised learning, namely clustering approaches where similar instances are grouped together to clusters (cf. [JMF99]).

Instance-based learners are usually based on a distance (or similarity) measure in the interval $[0, 1]$. A distance measure assigns a real number to each pair of instances [HWB01]:

$$DIST : \mathcal{I} \times \mathcal{I} \rightarrow \mathbb{R}$$

Similarities between objects can be seen as the opposite of their distances and are also bound between 0 and 1:

$$SIM(x, y) := 1 - DIST(x, y)$$

Referring to Horváth et al. [HWB01], a distance measure should ideally be a metric, i.e. (also cf. [Bis95]):

$$\begin{aligned} DIST(x, y) &\geq 0 && (3.1)\\ DIST(x, y) &= 0, \text{ iff } x = y && (3.2)\\ DIST(x, y) &= DIST(y, x) && (3.3)\\ DIST(x, y) &\leq DIST(x, z) + DIST(z, y) && (3.4) \end{aligned}$$

If the triangle inequality (Eq. 3.4) does not hold, the distance measure is called semi-metric [HWB01].

Case-based Reasoning (CBR) is a problem solving paradigm which uses "specific knowledge of previously experienced, concrete problem situations (cases)" [AP94, p. 39]. New problems are solved by finding similar past cases and reusing them in the new situation [AP94]. Referring to Mitchell [Mit97], CBR is based on two basic principles of IBL: using "lazy" learning methods and classifying new query instances by analyzing similar instances. In opposite to IBL where instances are usually represented by real-valued points in an n-dimensional Euclidian space, in CBR instances usually are represented by more powerful symbolic descriptions [Mit97]. As cases are stored and used for problem solving, CBR can be seen as an incremental learning approach and does not need specific learning phases [AP94].

Due to the focus of this work, the following aggregation is limited to approaches which can either deal with relational data or can handle similarities between object-oriented case representations. The approaches presented here compute similarities between objects, i.e., they could be either used for supervised learning in an instance-based learning setting or for unsupervised learning by applying clustering techniques. In both cases, incremental learning can be applied. Instance-based learning in the supervised case is per se incremental as the classifier is represented by stored instances and can be easily extended by new instances. As incremental clustering algorithms exist (see, e.g., [JMF99]), incremental learning is also possible in the unsupervised case.

Relational Instance-Based Learning

Emde and Wettschereck [EW96] introduce the relational instance-based learning algorithm *RIBL* where relational data can be handled as required. They introduce a similarity computation for (function free) first-order representations based on the one presented by Bisson [Bis92]. The similarity between two objects is computed considering their attribute values and the similarity of other objects related to them [EW96]. If objects are related to other objects by the same relation, *RIBL* takes the most similar related objects for similarity computation. A user-specified depth parameter determines how often a recursive similarity computation of the related objects is called. Different similarity measures are defined for the varying value types (e.g., numbers or discrete values). Classification is performed by a distance-weighted k-nearest neighbor algorithm. *RIBL* is extended by Bohnebeck and Horváth et al. [HWB01, Boh01] in order to handle lists and terms in the case representations. Here, the concept of edit distances for an efficient computation of similarities is applied.

Object-oriented Case Representations

Bergmann and Stahl [BS98] present an approach to similarity measures for object-oriented case representations. They address the problem of having instances of different classes which are structured in a class hierarchy, i.e., concept information can be used as stated in one of the requirements. Their similarity measure allows for computing similarities between objects of different classes. They distinguish between intra-class and inter-class similarity. The intra-class similarity computes the similarities between the common properties of two objects. In order to compute this value, the most specific common super class of both instances' classes is taken and a weighted sum of the attributes in this common super class is computed. The inter-class similarity represents the highest possible similarity of arbitrary instances of different classes. A similarity value is assigned to every node in the class hierarchy. If two objects are (direct) instances of the same class, the inter-class similarity is set to one, otherwise the annotated similarity value of the most special super class is taken. The overall similarity is the product of the intra-class and inter-class similarity [BS98].

Structural Similarity

A measure for computing the structural similarity of objects is presented by Jeh and Widom [JW01]. Here, the structural context of objects is taken into account. Relations between objects can be seen as structural information, thus, this approach is also relevant to the thesis. The basic idea behind the similarity measure is that "two objects are similar if they are related to similar objects" [JW01, p. 1]. Their similarity measure is called *SimRank* and is based on a graph representation model.

Objects and their relationships are modeled as a directed graph. The nodes represent objects and the edges the relations between the objects. The basic *SimRank* equation is defined recursively. If two objects are identical, the similarity value one is assigned. If one of the two objects does not have any incoming links, zero is assigned as similarity. In all other cases, the similarities of the in-neighbors (other objects linking to the objects which are compared) are computed [JW01]:

$$s(a,b) = \frac{C}{|I(a)||I(b)|} \sum_{i=1}^{|I(a)|} \sum_{j=1}^{|I(b)|} s(I_i(a), I_j(b))$$

where $I(x)$ is the set of in-neighbors of object x, $|I(x)|$ is the number of in-neighbors of object x, and C a constant between 0 and 1. $I_i(x)$ denote individual in-neighbors of x identified by the index i. This formula computes the average similarity of all pairs of the in-neighbors of object a and the in-neighbors of object b. Jeh and Widom [JW01] also present a bipartite variant of *SimRank* and pruning technologies. They applied the similarity measure to citation information of research papers and transcripts about courses of undergraduate students. A rapid convergence of the similarity values after few iterations ($k \approx 5$) is reported [JW01]. *SimRank* only uses structural information for computing similarities, but the authors note that it can be combined with other domain-specific similarity measures.

Relational Case-Based Reasoning

Armengol and Plaza [AP01] present *LAUD*, a distance measure between relational cases. The cases are represented by so called "feature terms" – a first-order representation. For similarity assessment it is distinguished between features with numerical values and features with symbolic values. In the first case, usual measures are used [AP01, p. 50]:

$$sim(v_1, v_2) = 1 - \frac{|v_1 - v_2|}{b - a}$$

where a and b define the range $[a, b]$ of the values for a certain feature. For features with symbolic values, the similarity is computed by taking the hierarchy of the sorts into account:

$$sim(v_1, v_2) = \begin{cases} 1 & \text{if } v_1 = v_2 \\ 1 - \frac{1}{M}\ level(lub(v_1, v_2)) & \text{otherwise} \end{cases}$$

lub is the least upper bound of the two values in the hierarchy of the symbolic values. M is the maximum depth and the level of the *lub* is M minus the depth of the *lub*. Armengol and Plaza also introduce a similarity measure for set-valued features. For this computation the similarities of all pairs of the value sets are computed. Then,

the pairs with the "best" values (i.e., greatest similarities) are taken and impossible remaining combinations are removed. In the aggregated similarity between two cases, "both the information that they share and the information that they do not share" is taken into account, i.e., the similarity of the shared features is computed and then normalized by the ratio of the shared to the relevant features [AP01, p.53].

3.1.4 Reinforcement Learning

In Reinforcement Learning (RL), an agent learns its behavior through "trial-and-error interactions with a dynamic environment" [KLM96, p. 237]. It differs from supervised learning in the way feedback is given to the learner. The kind of feedback is a reward or reinforcement which may be received just at the end of the agent's performance (e.g., the end of a board game), or more frequently during the agent's life [RN03] but not instantaneously in the general case. RL is, for instance, successfully applied to the board-game Backgammon in Tesauro's *TD-Gammon* system [Tes92]; it plays comparable to the top three human players (cf. [RN03, p. 780]). The adequacy for dynamic environments and the non-supervised feedback qualify this kind of learning as relevant and will be discussed in the following.

Additionally to the agent and the environment there are four main elements of a reinforcement learning system: a policy, a value function, a reward function, and (optionally[2]) a model of the environment [SB98]. The policy determines the behavior of an agent, i.e., it maps perceived states to actions, e.g., by using a function or a lookup table [SB98]. The reward maps the perceived state (or a state-action pair) to a reward value which rates how desirable the current situation is. The value function specifies the expected accumulated reward from the current state to the future [SB98]. The model of a RL system models the world, i.e., for some given state and action it predicts the next state and reward [SB98].

The learning problem in RL can be formulated as a Markov Decision Process [RM02, SB98]. A Markov Decision Process (MDP) consists of [KLM96, pp. 247-248]:

- a set of states $\mathcal{S}$ of the environment,
- a set of available actions $\mathcal{A}$,
- a reward function $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, and
- a state transition function $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \Pi(\mathcal{S})$.

The members of $\Pi(\mathcal{S})$ are probability distributions over the set $\mathcal{S}$, i.e., states are mapped to probabilities [KLM96]. The probability of making a transition from state s to s' when performing action a is formulated as $\mathcal{T}(s, a, s')$.

[2]Model-free approaches can learn a controller without learning a model [KLM96].

The learning problem is to find an optimal policy $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$ [RM02]. It can be distinguished between RL where the model exists, the model should be learned, and model-free approaches [KLM96]. The optimal value of a state is the "expected infinite discounted sum of reward that the agent will gain if it starts in that state and executes the optimal policy" [KLM96, p. 248]. With π as given decision policy it is written:

$$V^*(s) = \max_{\pi} E\Big(\sum_{t_0}^{\infty} \gamma^t r_t\Big)$$

where $\sum_{t_0}^{\infty} \gamma^t r_t$ is the so called infinite-horizon discount model which computes the long-run reward of the agent (see [KLM96] for details). The solution to the simultaneous equations defines this optimal value function:

$$V^*(s) = \max_{a} \Big(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^*(s')\Big), \forall s \in \mathcal{S},$$

i.e., the value of a state s is the actual reward plus the expected discounted value of the following state s'. The optimal policy can be specified as [KLM96, p. 248]:

$$\pi^*(s) = \underset{a}{argmax}\Big(R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') V^*(s')\Big).$$

Two algorithms for finding optimal policies are value iteration and policy iteration. The *value iteration* algorithm searches for the optimal value function in order to find the optimal policy. An iterative algorithm is applied which converges to the correct V^* values. In the *policy iteration* algorithm, the policy is manipulated directly. Initially, an arbitrary policy is chosen. This policy is improved iteratively in each state until no further improvement can be performed. For details of the algorithms see, e.g., [RN03, KLM96, SB98]. In the following, three popular RL approaches are introduced: $TD(\lambda)$, *Q-Learning*, and SARSA(λ). The reinforcement learning section closes with the presentation of a relational extension to RL.

The $TD(\lambda)$ Approach

Referring to Sutton and Barto [SB98], the most central and novel idea in RL is temporal-difference (TD) learning which combines Monte Carlo ideas with dynamic programming ideas. TD can directly learn from experience without having a model of the environment's dynamics. In TD learning given some experience and a policy π, the estimate of the optimal value function V^* is updated. The simplest TD method is $TD(0)$ [SB98]:

$$V(s_t) \leftarrow V(s_t) + \alpha\Big[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\Big].$$

Every time a state is visited its estimated value is updated. This results in new values closer to $r_{t+1} + \gamma V(s_{t+1})$ [KLM96]. If the learning rate is slowly decreased and the policy is held fixed, it is guaranteed that $TD(0)$ converges to the optimal value function [KLM96]. $TD(0)$ is a special case of the general $TD(\lambda)$ algorithm with $\lambda = 0$. The general rule for $TD(\lambda)$ is [KLM96]:

$$V(s_t) \leftarrow V(s_t) + \alpha\Big[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\Big]e(s_t).$$

It is applied to every state (instead of just the previous state) and $e(s_t)$ is the corresponding eligibility function including λ [KLM96]. For a detailed introduction into TD see [Sut88].

Sarsa

Sarsa is an on-policy TD control method and is introduced in [RN94]. It learns an action-value function (in contrast to a state-value function as it was the case in the previous approach). The task is to estimate the values $Q^*(s, a)$ for the behavior policy π, all states s, and all actions a [SB98]. Instead of considering state-to-state transitions and learned values of states, here it is dealt with transitions between state-action pairs and values for such pairs are learned [SB98]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\Big].$$

The update is done for each transition from a non-terminal state. $Q(s_{t+1}, a_{t+1})$ is defined as zero if s_{t+1} is terminal [SB98]. *Sarsa* got its name from the quintuple of events which are used in the update rule: $(s_t, a_t, r_{t+1}, s_{t+1}, a_{t+1})$. In more recent works, variants of *Sarsa* were applied, e.g., using function approximation with tile coding in [SSK05].

Stone and Sutton [SS01] apply Sarsa(λ)-Learning with tile-coding function approximation to learning keepaway[3] behavior in the RoboCup domain under idealized conditions (complete noise-free world state information). In subsequent work, they show that the simplification by the idealized conditions were not necessary in order to learn the keepaway task. In recent work, they also report on a comparison of Sarsa(λ) and Q-Learning for three keepers and two takers. In their experiments, Q-Learning takes more time to converge than *Sarsa* (40-50 hours vs. 15-20 hours) [SSK05]. When scaling to larger problems (5 vs. 4 keepaway) *Sarsa* takes about 70 hours.

[3] At keepaway a team of players who own the ball – the keepers – try to keep it away from the players of the enemy team as long as possible.

Q-Learning

Q-Learning is introduced by Watkins [Wat89, WD92]. It is a model-free reinforcement learning approach. One-step Q-Learning is defined by:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha\Big[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)\Big].$$

In this case, an action-value function Q is learned independent of the policy [SB98], i.e., it is an off-policy TD control algorithm. The optimal policy is learned implicitly in form of the Q-function. The Q-function takes a state-action pair as input and computes the quality of the action in this state (cf. [DD04]).

The Q-function can be usually represented as a table, but when states and actions are characterized by parameters, the number of combinatorial possibilities grows and it might be infeasible to use a tabular representation [DD04]. In order to solve this problem, usually inductive learning is integrated into the algorithm to approximate the Q table [DD04].

The RoboCup team around Martin Riedmiller, the *Brainstormers*, apply reinforcement learning to their soccer playing agents [MR02, RM02, RGH+06]. They describe robotic soccer as Multi-Agent Markov Decision Processes (MMDP). The difference to single-agent learning is in the reward function as every agent receives his own reinforcement signal [RM02]. The decision making task of the agents is decomposed hierarchically by distinguishing decision in individual capabilities (basic skills like intercepting the ball) and higher-level capabilities that enable team play on the tactical level [RM02]. Riedmiller and Merke present some results of the application of an adapted version of Q-Learning. They learn all basic skills of their agents by RL methods: kicking the ball, intercepting the ball, going to a position (avoiding obstacles), holding ball against an opponent, dribbling. On the higher-level they learn how to behave in attack situations. Using learning leads to an improvement of the number of goals achieved in comparison to their hand-crafted team by a factor of 1.6 [RM02].

Relational Reinforcement Learning

Džeroski et al. [DDB98] introduce the relational reinforcement learning (*RRL*) algorithm. It is a combination of Q-Learning and the relational regression tree algorithm *TILDE* ([BD98]; see also Section 3.1.1). The difference between *RRL* and other generalizing Q-Learning approaches is that relational representations are used for states and actions [DD04]. Džeroski [Dže03, p. 306] argues for such a relational representation of states as it is "natural and useful when the environment is complex and involves many inter-related objects". Driessens et al. [DRB01] introduce an extension where the fully incremental relational decision tree algorithm TG is integrated into *RRL*. In recent work, Driessens and Džeroski [DD04] integrated guidance into *RRL* where the use of reasonable policies is used to provide guidance.

3.1.5 Probability-based Approaches

Probability-based theories provide means to assign probabilities to hypotheses and allow for reasoning in uncertain environments – including dynamic environments. After a short introduction into the Bayes theorem (following [Mit97]) and its impact to learning different probabilistic approaches will be presented which can deal with either temporal or relational representations.

Bayesian Learning

The Bayes theorem can be seen as the "cornerstone of Bayesian learning methods" [Mit97, p. 156]. It can be applied to compute the posterior probability $P(h|D)$ of a hypothesis h at given data D if the prior probability of the hypothesis $P(h)$ is known, as well as the probabilities of the appearance of the data $P(D)$ and the data being observed knowing that a given hypothesis is true (cf. [Mit97]):

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}.$$

If all hypotheses are known, it is possible to compute the *maximum a posteriori* (MAP) hypothesis [Mit97, p.157]:

$$\begin{aligned} h_{MAP} &\equiv \underset{h \in H}{argmax}\ P(h|D) && (3.5)\\ &= \underset{h \in H}{argmax}\ \frac{P(D|h)P(h)}{P(D)} && (3.6)\\ &= \underset{h \in H}{argmax}\ P(D|h)P(h) && (3.7) \end{aligned}$$

If the probability of all hypotheses is assumed to be equal, the multiplication in Eq. 3.7 can be omitted in order to compute the maximum likelihood hypothesis at given training data h_{ML} [Mit97]:

$$h_{ML} = \underset{h \in H}{argmax}\ P(D|h).$$

The "most probable *classification*" of new instances given the training data can be computed by the Bayes optimal classifier [Mit97, p. 174]. The predictions of all hypotheses are combined and weighted by their posterior probabilities:

$$P(v_j|D) = \sum_{h_i \in H} P(v_j|h_i)P(h_i|D),$$

where $v_j \in V$ are all possible classifications and $P(v_j|h_i)$ is the probability that the new instance belongs to class v_j when using hypothesis h_i [Mit97].

In naïve Bayesian learning, instances are represented by a conjunction of attribute values and the target function takes a value v_j of a finite set V. The most probable target value can be assigned by [Mit97]:

$$v_{MAP} = \underset{v_j \in V}{argmax}\ P(v_j|a_1, a_2, \ldots, a_n) \tag{3.8}$$

$$= \underset{v_j \in V}{argmax}\ \frac{P(a_1, a_2, \ldots, a_n|v_j)P(v_j)}{P(a_1, a_2, \ldots, a_n)} \tag{3.9}$$

$$= \underset{v_j \in V}{argmax}\ P(a_1, a_2, \ldots, a_n|v_j)P(v_j) \tag{3.10}$$

In naïve Bayesian learning, it is assumed that attribute values are conditionally independent, thus, the probability of observing $a_1, a_2, \ldots, a_n$ can be computed by the product of the individual probabilities. The naïve Bayes classifier is [Mit97]:

$$v_{NB} = \underset{v_j \in V}{argmax}\ P(v_j) \prod_i P(a_i|v_j).$$

Naïve Bayes classifiers are used successfully, e.g., for text classification tasks (cf. [YL99, Seb02]).

Bayesian networks are directed graphs "in which each node is annotated with quantitative probability information" [RN03, p.493]. Russell and Norvig specify a Bayesian network as follows:

1. "A set of random variables makes up the nodes of the network. Variables may be discrete or continuous.

2. A set of directed links or arrows connects pairs of nodes. If there is an arrow from node X to node Y, X is said to be parent of Y.

3. Each node X_i has a conditional probability distribution $P(X_i|Parents(X_i))$ that quantifies the effect of the parents to the node.

4. The graph has no directed cycles (and hence is a directed, acyclic graph, or DAG)." [RN03, p. 493]

Bayesian networks allow for representing dependencies among variables and provide a representation of joint probability distributions. There exist exact and approximate inference mechanisms for Bayesian networks. Details can be found, e.g., in Russell and Norvig's book on Artificial Intelligence [RN03].

Probabilistic Reasoning over Time

Probabilistic reasoning over time addresses reasoning in dynamic environments with uncertainty. Referring to Russell and Norvig, the basic inference tasks are filtering,

prediction, smoothing, and finding the most likely explanation [RN03]. With filtering (or monitoring) the posterior probability distribution of the current state for some evidence is computed. Prediction tasks compute posterior distributions over future states. If a posterior distribution over a past state is computed, it is called smoothing (or hindsight). The task of finding the most likely explanation is to identify a sequence of states that most likely has generated observations in evidence variables.

The different models for probabilistic reasoning over time represent the dynamic aspects by time slices. Each time slice contains sets of observable and unobservable variables. At time t these sets are denoted by X_t (unobservable) and E_t (observable; evidence variables). In order to avoid an unbound number of variables and a potentially infinite number of parents at computations, it is usually assumed that the transition model from each state to its successor is stationary, i.e., identical for all time points and that the Markov assumption holds, i.e., that the evidence variables just depend on a finite history of states. Typically, it is also assumed that the evidence variables just depend on the current state.

Three famous models for probabilistic reasoning over time are: Hidden Markov Models, Kalman Filters, and Dynamic Bayesian Networks. The following brief descriptions are mainly based on Russell and Norvig's chapter of "Probabilistic Reasoning over time" where more information about the approaches can be found [RN03, Chapter 15].

Hidden Markov Models (HMM) consist of one single discrete random variable. The values of this variable represent possible world states. A transition model $P(X_t|X_{t-1})$ is a $S \times S$ matrix T_{ij}, where S is the number of states:

$$T_{ij} = P(X_t = j|X_{t-1} = i).$$

T_{ij} stands for the probability of a transition from state i to j. Another model in HMMs is the observation O_t model, which can also be represented by a matrix. It stands for the probabilities that a specific state leads to certain observations in the evidence variable. The restriction of HMMs to single discrete variables and the matrix representations allow for simplified matrix algorithms in order to solve the tasks mentioned above.

Another approach that can deal with a number of continuous variables is introduced by Kalman [Kal60]. *Kalman filters* can be used "to estimate the state (position and velocity, for example) of a physical system from noisy observations over time" [RN03, p. 551]. Due to the continuous variables conditional densities are needed for the transition and sensor models – Gaussian distributions are used here.

Dynamic Bayesian networks (DBN) extend Bayesian networks in order to model dynamic systems [SDW03]. They can have an arbitrary number of state variables and evidence variables and "include Kalman filters and hidden Markov models as special cases" [RN03, p. 537]. A HMM can be represented by a single state and

evidence variable. DBN are also more general than Kalman filters as they can model arbitrary distributions. Inference in DBN is NP-complete; a famous approximate reasoning method is *Particle Filtering* [SDW03]. In *Particle Filtering*, a number of particles is used to approximate the probability distributions. It is started with generating N particles according to the initial distribution. In each further step for each particle, the next state is generated and the new samples are weighted according to the likelihood with the observations and resamples N new particles [SDW03].

Probabilistic Reasoning with Relational Representations

Flach and Lachiche [FL99, FL01, LF03, FL04] address the naïve Bayesian classification of structured individuals. They present two approaches that upgrade propositional naive Bayes classifiers in order to deal with structured data. In the *1BC* system (see [FL99]), aggregated properties of related objects are employed as attributes of the individual, i.e., a propositionalization is performed [FL04]. In the second approach – the *IBC2* system (see [LF03]) – the probabilistic engine is modified, i.e., no representation transformation is performed, but it is directly worked on the structured representation [FL04]. Both approaches are implemented in the *Tertius* system [FL01, FL04].

Another recent probabilistic approach – Dynamic Probabilistic Relational Models (DPRMs) – is presented by Sanghai et al. [SDW03]. DPRMs extend dynamic Bayesian networks (DBN). They address the drawback of previous approaches which cannot compactly represent many real-world domains, e.g., where multiple objects and classes exist and "objects and relations can appear and disappear over time" [SDW03]. Each time slice in a DPRM is represented by a probabilistic relational model (PRM, [FGKP99]). PRMs allow reasoning with classes, objects, and relations [SDW03]. Sanghai et al. develop approximate inference methods based on Rao-Blackwellisation of relational attributes and abstraction trees and apply them to monitor assembly processes for fault detection.

3.1.6 Artificial Neural Networks

The field of *Artificial Neural Networks* (ANN) is inspired by biological learning systems which are based on complex networks of interconnected neurons [Mit97]. ANN consist of nodes (a.k.a. units) and directed links connecting the nodes. Each link has a numeric weight $W_{j,i}$ that represents the strength of the connection [RN03]. Each node computes the weighted sum of its inputs and applies an activation function (e.g., a threshold function or a sigmoid function) in order to determine its output [RN03]. Some neurons may represent the input to the neural net (e.g., the input of a function) and others the output (e.g., the target value of the intended function).

It is distinguished between feed-forward and recurrent networks. The feed-forward networks do not have any cyclic structures; they can be seen as directed layers of neurons where the neurons of each layer are only connected to neurons of subsequent layers. If there is no hidden layer between the input units and the output units in a network it is called a "perceptron" network; these kind of networks can only learn linearly separable functions [RN03, p. 740].

Learning in neural networks basically is changing the weights of the connections; there are also approaches where changing the structure of networks by adding or removing neurons or connections are addressed, e.g., Mézard and Nadal's tiling algorithm [MN89]. One famous algorithm for multilayer feed-forward networks is the backpropagation algorithm. It starts from the output layer by computing the Δ values (the observed error) for the output units. Then the Δ values are propagated back to the previous layer and the weights between the two layers are updated. More details about backpropagation learning can be found, e.g., in [Mit97, RN03]. There are also techniques for unsupervised learning in ANNs, e.g., learning vector quantization and self-organizing maps (a.k.a. Kohonen maps) [Zel00].

If time series should be processed by an acyclic feed-forward network, the input values of all time points to consider could be used as input. The drawback is that this restricts the size of the window of time; if an "arbitrary window of time in the past" should be somehow accessible a solution can be found in recurrent networks [Mit97, p. 120]. Different variants of recurrent networks have been proposed, e.g., by Jordan and by Elman ([Jor86, Elm90]; cf. [Mit97, Zel00]).

Although ANNs perform quite well in many learning tasks there exists the problem of understanding how the results are achieved. Andrews et al. [ADT95, p. 373] state that for user acceptance of ANNs it is "highly desirable if not essential that an *explanation* capability should become an integral part of the functionality of a trained ANN". This is especially the case in safety critical applications. There have been a number of approaches to extract rules from neural networks (e.g., [TS93, CHD98, SDTB98]). Andrews et al. [ADT95] set up a classification scheme for such approaches. It takes into account the expressive power of the extracted rules, the "translucency", the extent of incorporated training regimes, the quality of the extracted rules, and the algorithm's complexity. The quality is evaluated by accuracy, fidelity (i.e., how good the rules mimic the ANN), consistency, and comprehensibility [ADT95]. Zhou [Zho04] identifies in his article the "fidelity-accuracy dilemma" stating that usually high fidelity and high accuracy are not possible. He suggests to distinguish among rule extraction approaches for accurate and comprehensible learning systems and approaches for understanding the working mechanisms of neural networks [Zho04].

One decompositional approach to rule extraction, i.e., focusing on the extraction of rules at the level of individual units [ADT95], is presented by Towell and Shavlik [TS93]. They introduce the *SUBSET* algorithm which extracts rules from

knowledge-based neural networks (*KBANN*, [TSN90]). In the algorithm, for each hidden and output unit two steps are performed: In the first step, "up to β_p subsets of the positively-weighted incoming links whose summed weight is greater than the bias on the unit" are extracted. In the second step, for each subset $\mathcal{P}$ of β_p "up to β_n minimal subsets of negatively-weighted links whose summed weights (sic!) is greater than the sum of $\mathcal{P}$ less the bias on the unit" are extracted. Then an unused predicate $\mathcal{Z}$ is introduced and for each subset $\mathcal{N}$ of β_n rules are formed: "if $\mathcal{N}$ then $\mathcal{Z}$". Finally, a rule is formed: "if $\mathcal{P}$ and not $\mathcal{Z}$ then <name of unit>" [TS93].

3.2 Representation of Dynamic Scenes

Autonomous agents which act in the real (or a simulated) world perceive information about the environment by sensors. It can be distinguished between passive and active sensors. In opposite to passive sensors which "capture signals that are generated by other sources in the environment" (e.g., cameras), active sensors "send energy into the environment" (e.g., sonar sensors) [RN03, p. 903]. In robotics and intelligent vehicles, typical sensors are cameras, radar systems, laser range scanners, infra-red sensors and the *Global Positioning System* (GPS).

Perception can be defined as the process where sensor measurements are mapped into internal representations of the environment [RN03]. Two major tasks in robotics are localization and mapping. Localization is the task of determining the positions of objects in the environment and mapping addresses the creation of a map if it is not given to the robot in advance. If the robot has to create a map without knowing about its position, the task is called *simultaneous localization and mapping* (SLAM) [RN03]. Russell and Norvig mention temperature, odors, and acoustic signals as other types of perceptions.

The perceived information can be used to form the world model, i.e., the belief of the world of an agent. It should consist of all relevant information about objects in the environment and their states. Typically, the information in the world models is quantitative as the data provided by the sensors or sensory processing routines usually leads to quantitative information like positions, distances, or angles as real number values. However, for the application of higher-level functions qualitative representations are advantageous or even mandatory which leads to the requirement of a qualitative mapping. Components like situation assessment, behavior decision, and planning take advantage of qualitative representations, and the learning approach developed in this thesis is also intended to work with qualitative information.

The remainder of this section presents approaches to qualitative spatial and temporal reasoning, formalisms which allow for the representation of actions and/or events, and some integrating approaches to describe dynamical scenes qualitatively.

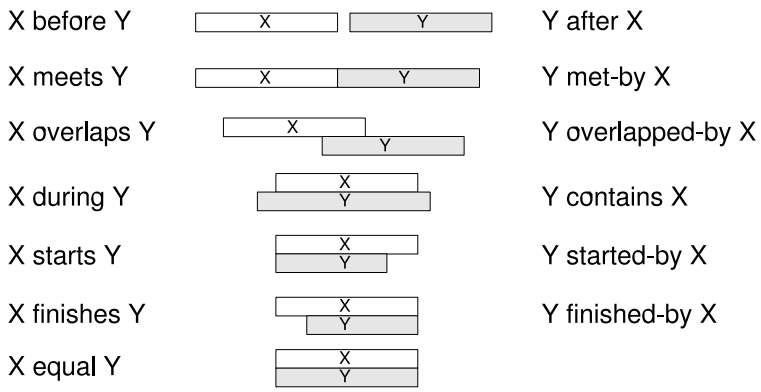

Figure 3.2: Allen's temporal relations [All83]

3.2.1 Qualitative Representations of Space and Time

This section gives a brief introduction to some approaches for qualitative representations of space and time. This field is of interest for this thesis because on the one hand the methods are highly relevant for the generation of a qualitative abstraction of quantitative data, and on the other hand some approaches for the representation of dynamic scenes are based on qualitative representations. As the focus of this thesis is on learning, a rather compact presentation of selected work is done. Further details can be, for instance, found in [CH01].

Motion of objects in dynamic environments takes place in a four dimensional space: Three spatial dimensions and one temporal dimension. Spatiotemporal information can be represented quantitatively by different spatial coordinates at certain time points. Qualitative representations abstract from these concrete coordinates. In the literature, many different approaches for temporal, spatial, and spatiotemporal (motion) descriptions have been proposed. A selection based on [Mie04, Geh05] is presented in the following.

Representation of Temporal Information

Famous representations for the temporal dimension are Allen's temporal logic and Freksa's semi-intervals [All83, Fre92a]. Allen introduces a temporal logic with temporal intervals as primitives [All83]. He defines seven relations between time intervals and together with their inverse relations – except for the *equal* relation where no inverse is needed – 13 disjoint relations are introduced: *before, equal, meets, overlaps, during, starts,* and *finishes* and their inverse relations (see Fig. 3.2). Allen also introduces a composition table to infer which temporal relations between two intervals are possible by mapping the relations of both to a third interval, e.g., if it is known that interval A is *before* B and interval B is *before* C it must hold that

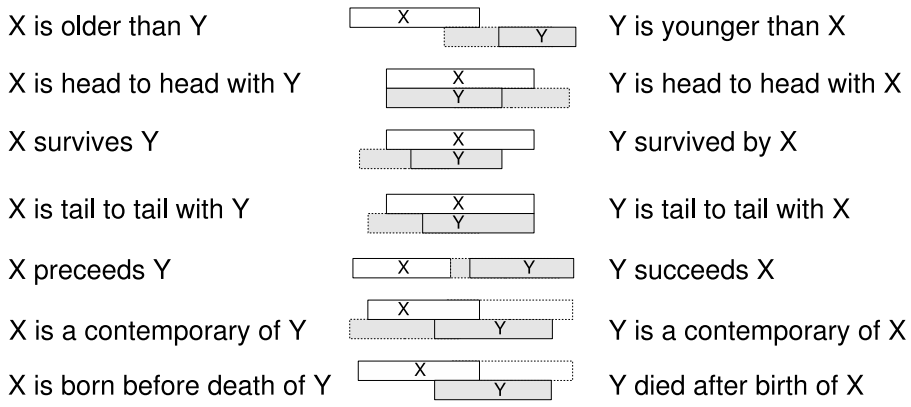

Figure 3.3: Freksa's semi-interval relationships [Fre92a]

A is also *before* C. Allen's composition table contains all possible relation pairs of relations between intervals A and B as well as between B and C and provides a set of consistent relations between A and C [All83]. A generalization of Allen's interval relations to two dimensions is proposed by Gottfried [Got04].

Freksa [Fre92a] introduces a generalization of Allen's approach where he introduces the conceptual neighborhood of qualitative relations and the concept of semi-intervals as basic units. He defines two relations between pairs of events to be conceptual neighbors if "they can be directly transformed into one another by continuously deforming (i.e. shortening, lengthening, moving) the events (in a topological sense)" [Fre92a]. Semi-intervals describe the start and end of intervals. This allows for making statements about intervals even if one of the time points is not known. An example given by Freksa is "X was born before Y's death"; with Allen's representation it had to be expressed as "X lived *before* Y or X' life *meets* Y's life or X' life *overlaps* Y's life or X' life is *finished by* Y's life or X' life *contains* Y's life or X' life is *started by* Y's life or X' life *equals* Y's life or X' life *starts* Y's life or X lived *during* Y's life of X' *finishes* Y's life or X' life is *overlapped by* Y's life" [Fre92a]. Freksa introduces eleven semi-interval relations which are shown in Fig. 3.3. The regions enclosed by the dotted lines in this figure illustrate variants still covered by the corresponding relation.

Representation of Spatial Information

Spatial representations can be classified into approaches that describe topological, ordinal, or metric information. Clementini et al. [CDH97] present a framework for the qualitative representation of 2D positional information. In their work, they describe how orientation and distance information can be discretized into qualita-

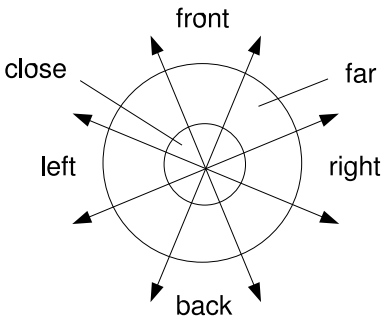

Figure 3.4: Example for qualitative distance and direction classes

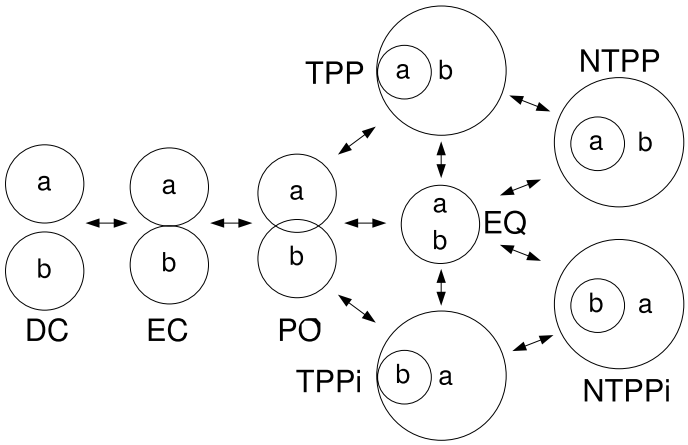

Figure 3.5: Region Connection Calculus (RCC-8) [RCC92]

tive classes (e.g, *front, back, left, right* and *close, far*). An example for qualitative distance and direction classes is given in Fig. 3.4.

A well-known approach to spatial representations based on topological information is the *Region Connection Calculus* (RCC) by Randell et al. [RCC92]. RCC can be used to describe connectivity and overlap relations of regions. RCC-8 distinguishes between *disconnected (DC), externally connected (EC), partial overlapping (PO), tangential proper part (TPP)* (and its inverse, TPPi), *equal (EQ), non-tangential proper part (NTPP)* (and its inverse NTPPi). A similar approach is presented by Egenhofer [Ege91].

Fig. 3.5 shows the RCC-8 relations and their neighborhood graph (adapted from [CH01]). Similar to Allen's interval relations, a composition table is introduced by Randell et al. [RCC92], i.e., if relations between two regions to a third one are known, information about the relation between these two regions can be derived.

Freksa [Fre92b] presents an approach where orientation information based on a direction vector is used to qualitatively describe the position of other points relative

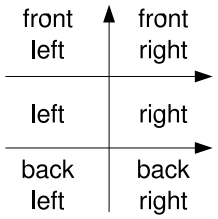

Figure 3.6: Freksa's orientation grid [Fre92b]

to this vector ("orientation grid"). Three lines – one covering the vector and two orthogonal lines at the start and end point of this vector – divide the space into six regions. Including the positions which lie on at least one of the three lines, i.e., which lie on one of the region borders, 15 qualitative orientation relations can be distinguished (Fig. 3.6). In his work, Freksa [Fre92b] develops an inference scheme for orientation-based spatial inferences in analogy to Allen. Gottfried [Got05] introduces calculi for bipartite and tripartite arrangements based on the orientation grid.

Schlieder's panorama approach [Sch93, Sch96b] defines an ordinal arrangement of objects in the order how they are perceived by looking around from a viewpoint (panorama). More detailed information can be achieved if the opposite position of objects is also acquired in the panorama. The panorama is extended by metric information and qualitative directions and distances by Wagner et al. [WVH04].

3.2.2 Formalisms for Representing Actions, Events, and Time

Three of the most known representation formalisms for actions (and time) are introduced in this section: The situations calculus, *STRIPS* representations, and the interval temporal logic.

Situation Calculus

The original formulation of the situation calculus was done in 1969 by McCarthy and Hayes [MH69]. Allen and Ferguson [AF94, p. 4] refer to it as the "general theory of the situation calculus" and describe it as "a point-based temporal logic with a branching time model".

The situation calculus – a second-order language – allows for describing dynamically changing worlds [Rei01]. The world at certain time points is described by situations and changes of the world are done by actions. The initial situation, i.e., before any action was performed, is denoted by the constant S_0 [Rei01]. The binary function symbol $do(\alpha, s)$ denotes the successor situation if action α is performed in situation s. The following examples by Reiter illustrate the execution of actions

in situations: $do(put(A, B), s)$ means that object A is put on top of B in situation s; $do(putDown(A), do(walk(L), do(pickup(A), S_0)))$ denotes the sequence of action $[pickup(A), walk(L), putDown(A)]$, i.e., picking up some object A, walking to L and putting down A again [Rei01, p. 19]. Situations themselves are described by first-order terms.

Actions have preconditions and effects. The preconditions specify the requirements that must be satisfied in order to perform the action. A predicate $Poss(\alpha, s)$ is defined which means that it is possible to perform the action α in situation s [Rei01]. The following example given by Reiter denotes that it is possible for the robot r to pickup object x in situation s if it is not holding anything else (denoted by z in the example), object x is not heavy and the robot is next to the object in this situation [Rei01, p. 20]:

$$Poss(pickup(r, x), s) \supset [(\forall z)\neg holding(r, z, s)] \wedge \neg heavy(x) \wedge nextTo(r, x, s).$$

The effect of dropping a fragile object can be denoted as [Rei01, p. 20]:

$$fragile(x, s) \supset broken(x, do(drop(r, x), s)).$$

This information allows for reasoning what actions are possible in which situations. In a strict sense it does not really allow to imply the possible actions due to the "qualification problem" which says that there might be exceptions where, e.g., picking up does not work– Reiter mentions the colorful examples that it must also hold that $\neg gluedToFloor(x, s) \wedge \neg armsTied(r, s) \wedge \neg hitByTenTonTruck(r, s) \ldots$ [Rei01, p. 21]. It is chosen to ignore the "minor" qualifications and just record the important ones.

Another problem called the "frame problem" is that other than the effect axioms are needed. The so called "frame axioms" specify the action invariants, i.e., the fluents which are not effected by the action [Rei01]. An example is that if a robot r' paints an object x' with color c it does not have any effect on robot r holding object x [Rei01]:

$$holding(r, x, s) \supset holding(r, x, do(paint(r', x', c), s)).$$

The actual problem is that there exists a huge number of these frame axioms. For more details about the frame problem and solutions see, e.g., [Rei01].

In situation calculus, actions are represented as functions from one situation to another and it is not possible to represent information about the duration of actions, to represent effects that do not start at the end of the action, and to represent preconditions that hold beyond the start of the action [AF94]. However, there are extensions of the situation calculus that allow for representing concurrent actions [Pin94].

STRIPS

Referring to Allen and Ferguson [AF94], the *STRIPS* (STanford Research Institute Problem Solver, [FN71]) representation adds additional constraints to the situation calculus. A world model is represented as a collection of first-order predicate calculus formulas [FN71]. Effects of actions are specified by add and delete lists. These lists specify which propositions must be added to or deleted from the world state. It is also possible to define preconditions which must be satisfied to make the action applicable. A typical example from the *Block's World* for an action is [AF94, p. 6]:

```
STACK(a,b)
    preconds: (Clear a) (Clear b)
    delete:   (Clear b)
    add:      (On a b)
```

In order to stack an object a to another object b, the precondition is that both objects are "Clear", i.e., no other objects are on top of them. After performing the action object b is not clear any more and object a is on top of object b.

Similar to situation calculus the actions are effectively instantaneous and it is not possible to make any statements about what is true while an action is executed [AF94]. In *STRIPS*, it is assumed that "the world only changes as the result of a single action by the agent", i.e., it is not possible to represent situations where an action occurs while some other actions (or events) are occurring [AF94].

Interval Temporal Logic

The basic temporal structure in the interval temporal logic is the interval representation of time which is developed by Allen [AF94] (see Section 3.2.1 and [All83, All84]). The temporal logic can be used to describe knowledge about properties, events, and actions[4].

Referring to Allen and Ferguson [AF94, p. 10], the "most obvious way to add times into a logic is to add an extra argument to each predicate". They use time intervals as such arguments which allows for describing a proposition that some predicate P is true (or false) during some interval [AF94].

In the interval temporal logic, events are represented by a variable. If the person $jack34$ lifts the ball $ball26$ onto the table $table5$ at time $t1$, it would be represented as [AF94]:
$\exists e.LIFT(e) \ \wedge \ agent(e) = jack34 \ \wedge \ dest(e) = table5 \ \wedge \ theme(e) = ball26 \ \wedge \ time(e) = t1$.

In this approach, events are divided into types and the types define a set of role functions which specify the arguments to the event instances. In order to

[4] In his theory of action and time, Allen [All84] distinguishes between properties, events, and processes.

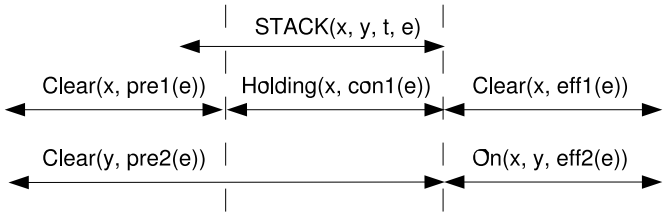

Figure 3.7: Conditions for stacking x on y [AF94]

define when an event occurs, conditions can be formulated as shown in this stacking example [AF94, p.14]:

$$\begin{aligned}
&\forall x, y, e.STACK(x, y, t, e) \supset \\
&\quad \exists j, k, l, m, n.Clear(x, j) \wedge Overlaps(j, t) \wedge \\
&\qquad Holding(x, k) \wedge Finishes(k, t) \wedge Meets(j, k) \wedge \\
&\qquad Clear(x, l) \wedge Meets(t, l) \wedge Clear(y, m) \wedge SameEnd(t, m) \wedge \\
&\qquad On(x, y, n) \wedge Meets(t, n). \qquad (3.11)
\end{aligned}$$

The conditions are also illustrated in Fig. 3.7 (adapted from [AF94, p. 15]).

The representation of events does not include the knowledge of causality, i.e., which properties are caused and which must be true as condition [AF94]. Allen and Ferguson introduce the predicate $Try(\pi, t)$ which is only true if π is executed over t. In the case of stacking, a successful stacking try can be described as follows [AF94]:

$$\begin{aligned}
&\forall x, y, t, j, k.Try(stack(x, y), t) \wedge Clear(x, j) \wedge Overlaps(j, t) \wedge \\
&Clear(y, k) \wedge SameEnd(k, t) \supset \exists e.STACK(x, y, t, e)
\end{aligned}$$

3.2.3 Approaches to Motion Description and Applications

This section presents approaches to motion description where some of the representations mentioned above are used or which are somehow related. Three of the approaches use the situation calculus or related representations and one uses Allen's interval temporal logic.

Different authors point out the advantages of using qualitative representations for agents in dynamic environments as robustness, efficiency, identical representations for similar situations, and comprehensibility (e.g., [FSW04, WVH04, GLH05]).

Application of Qualitative Reasoning to Robotic Soccer

Fraser et al. [FSW04] present an approach of qualitative reasoning to robotic soccer. They describe how to extract qualitative knowledge from quantitative world models,

and how to use the qualitative model for computing and executing plans. They propose a qualitative representation language with ground predicates P for basic facts about the state of the world, e.g., $reachable(A, B)$ [FSW04]. The validity of the n-ary predicates $p \in P$ is checked by evaluating some conditions. An example for the reachable predicate is [FSW04]:

```
COND_inReach(X, M)
  return dist(pos(X, M), pos(self, M)) < 1.
```

In the paper, a number of predicates for representation (e.g., $Left(X)$, $Right(X)$, $InReach(X)$) and some actions (e.g., $DribbleTowards$, $GoTowards$, $GrabBall$, $Kick$) are defined. Classical planning is suggested by Fraser et al. [FSW04] in order to create plans.

Representation of Dynamic Scenes with Situation Calculus

Chella et al. [CFG00] propose a framework for visual knowledge representation. As the aim is to create an understandable representation, a high-level, declarative representation – a KL-ONE like semantic net – is chosen. In their subsequent work, they propose to map the situation calculus on their conceptual representation. In their work, conceptual spaces are used to represent motion. Referring to Chella et al. [CFG01, p. 335], a conceptual space "is a metric space whose dimensions are in some way related to the quantities processed in the subconceptual area", which can be in some cases strictly related to sensory data. Examples of dimensions of a conceptual space are color, pitch, mass, spatial co-ordinates etc. [CFG01].

Chella et al. [CFG01] use the term "knoxel" in order to denote a point in the conceptual space and complex entities are represented by a set of knoxels. Mapped to the situation calculus a scattering in the conceptual space corresponds to an action, a knoxel corresponds to a process, and a configuration of knoxels corresponds to a state [CFG01]. Chella et al. give as example the movement of an object o from position p_1 to p_2. When the object starts to move a scattering occurs in the conceptual space and a knoxel becomes active. This could be represented by $start\_move(o, p_1, p_2)$. Before the object reaches the final position the fluent $moving(o, p_1, p_2)$ is true [CFG01].

As the original version of the situation calculus does not allow for concurrency, Chella et al. use an extension of the situation calculus where actions can be performed concurrently denoted by the +-function [Rei01, Pin94]. This allows for representing concurrent actions like $start\_move\_arm = start\_move\_forearm + start\_move\_upper\_arm$.

Acting and Deliberating using Golog

The group of Lakemeyer uses *Golog* – "a second-order language for reasoning about actions and their effects"[FFL05, p. 24] which is based on Reiter's situation calcu-

lus variant – for deliberation and team control in robotic soccer [FFL05, DFL03]. *Golog* provides imperative control constructs as loops and conditionals, recursive procedures, and constructs for the non-deterministic choice of actions [FFL05]. In a recent extension – *DTGolog* – decision-theoretic planning is introduced. In *DT-Golog*, meaning can be given to primitive actions by using basic action theories. A reward is assigned to situations in order to indicate the desirability of reaching this situation [FFL05]. Additionally, with stochastic actions it is possible to assign probabilities of success to actions.

The language *Readylog* proposed by Ferrein et al. [FFL05] is based on these techniques. It is an extension of *Golog* and allows for specifying the behavior of robots in dynamic real-time domains. It provides a number of control constructs; besides the ones provided by *Golog* mentioned above it is possible, e.g., to define condition-bounded executions, probabilistic actions, and procedures [FFL05]. It is possible to model procedures which are used by the behavior decision. An example for trying to perform a double pass is [FFL05, p. 26]:

$$
\begin{array}{l}
proc\ tryDoublePass(Own, Teammate) \\
\quad lookForFreeSpace(Own, Teammate); directPass(Own, Teammate); \\
\quad pconc(receivePass(Teammate), \\
\qquad interceptPass(closestOpponent(Teammate)), \\
\qquad if\ ballIsKickable(Teammate) \\
\qquad then\ passTo(Teammate, freePosition); \\
\qquad\quad interceptPass(closestOpponent(Own)) \\
\qquad endif \\
\qquad moveToPos(Own, freePosition); receivePass(Own);) \\
endproc
\end{array}
$$

In Readylog, a model-based passive sensing approach is combined with *DT-Golog* in order to perform decision-theoretic planning. More details can be found in [FFL05].

Conceptual Description of Image Sequences

Arens and Nagel [AN05] also address the creation of qualitative conceptual knowledge out of quantitative data. They propose *situation graph trees* (SGTs) as the representation formalism. In their "layer model of a cognitive vision system", three sub systems exist: the interactive system, the vision system, and the conceptual system [AN05]. The interactive system is the bottom-most sub system and enables observing a scene by sensors (e.g., cameras) and might also contain actuators. In the vision system, image data is transformed into a three-dimensional quantitative

description of the scene. The top-most conceptual system the three-dimensional scene description is transformed into "conceptual primitives" [AN05, p. 6].

Situation schemes are the basic units in SGTs and can be instantiated by an observed agent. This representation consists of predicates of a fuzzy, metric temporal logic (developed by Schäfer [Sch96a]) that define the state of an agent or its environment [AN05]. Traffic domain examples for such predicates are *driving*, *standing*, $on\_lseg(Agent, Lseg)$, $following\_anyone\_on(Agent, Patient, LObj)$ [AN05]. The state scheme co-exists with an action scheme where the actions an agent can perform are described. Situation graphs are set up by connecting situation schemes with prediction edges which indicate temporal successor relations. This allows for identifying expected actions (and thus, e.g., expected changes in speed and steering) the vehicle is likely to perform [AN05].

Qualitative Motion Representation

Musto [Mus00] develops a qualitative motion representation which is later extended by Stein [Ste03]. The motion description is based on a two layer architecture. The first layer describes basic motion by denoting a qualitative direction and a qualitative distance the object moved in the given direction. These two components form the *qualitative motion vector* (QMV) and a sequence of *QMVs* describes the motion [Ste03]. The second layer describes motion by a sequence of shapes which are basic elements like "right hand bend". The first layer is closer to the actual measured motion of objects; the second layer rather complies to linguistic descriptions of motion [Ste03].

The *QMVs* are represented by triples of distance, direction, and duration. With very small values for the duration quite exact representations of the actual movement are possible. Motion sequences are denoted as follows:

$$\langle d_1, \alpha_1, \Delta t_1 \rangle \ \langle d_2, \alpha_2, \Delta t_2 \rangle \ \langle d_3, \alpha_3, \Delta t_3 \rangle \ldots$$

where d_i represent the distances, α_i the directions and Δt_i the durations of the single *QMVs*.

This representation can be used for quantitative as well as for qualitative representations. Stein mentions as potential qualitative classes {zero, very-close, close, medium-distance, far, very-far} for distances and $\{zero, north, east, south, west\}$ for directions. Stein [Ste03] also presents the algebra *QMVAlgebra* for calculations on QMV sequences.

Qualitative Motion Description

A qualitative motion description is introduced by Miene [MVH04, Mie04]. In this approach, movements and positions are abstracted to different direction, distance,

and velocity classes. Intervals are created from time-series based on monotonicity and threshold criteria. Such intervals are created for properties of single objects or for relations between object pairs (e.g., *speed of object x is slow*, *distance between x and y is very close*).

Basically, only position information of objects is needed in order to create the qualitative motion description. For each object and time point the following is needed [Mie04, p. 45]:

- the unique object identifier,
- the timestamp, and
- the object's coordinates.

This information is sufficient to describe a number of single object motion properties, and relations between object pairs as well. It is also important to know the classes of the objects to decide which motion properties and relations should be extracted. If the classes of objects are not given (e.g., by a simulation environment), a classification of objects has to be performed by the system.

Information about the motion direction and the velocity of objects can be derived from positional information of adjacent time points (positional difference). The positional change can be computed by subtracting the adjacent values [Mie04]:

$$\underset{t}{\Delta_x} = (x_t - x_{t-1}) \text{ and } \underset{t}{\Delta_y} = (y_t - y_{t-1})$$

An angular direction value can be computed by applying the arc tangent to the fraction of the two relative change values of the positions in the two dimensions; the velocity can be computed by taking into account the length of the positional shift per time step [Mie04]. This leads to two time series about the motion direction and the velocity of the object.

For each object pair, Miene also extracts two time series which represent the relative angle and distance between those two objects. Altogether this method leads to:

- One time series per object with the motion direction (angle of direction),
- one time series per object with the velocity (length of motion vector),
- one time series per object pair with their relative direction (angle), and
- one time series per object pair with their distances (length).

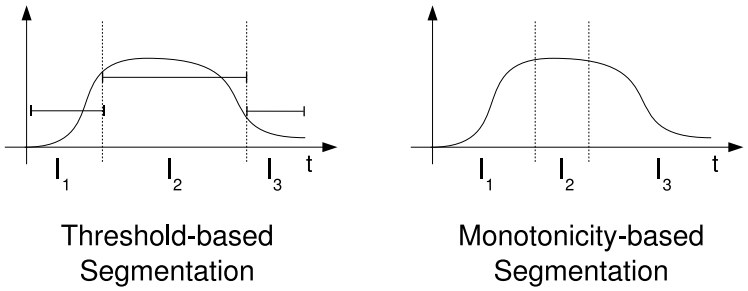

Figure 3.8: Threshold-based and monotonicity-based segmentation [Mie04]

These discretized[5] time series can have false values or outliers, or can be based on noisy sensors. In these cases, it is necessary to preprocess the time series before the qualitative abstraction is performed. Miene presents two approaches: The equalization of "gaps" in the time series (e.g., a complete halt in-between two noticeable movements) by interpolation and smoothing of the time series by applying a mean filter (which is also used in image processing). In mean filtering, a value is replaced by the mean of a fixed number of its predecessor and successor values. Probabilistic approaches like Kalman filters [Kal60] could also be applied.

Miene [Mie04] introduces two methods for the segmentation of time series: Threshold-based segmentation and monotonicity-based segmentation (illustrated in Fig. 3.8). In the threshold-based method, consecutive values are grouped together if they are similar. In Miene's approach, for every value it is decided if it is a member of the previous interval or if a new interval should be created. In the first case, the interval is extended by the actual value. The main idea of this segmentation is to keep track of the mean value of the current interval. If the difference of the next value to the mean value is within a defined threshold, the interval will be extended and the mean value will be updated. Otherwise, a new interval is created.

The monotonicity-based segmentation divides the time series into intervals where monotonicity criteria are satisfied. The intervals are identified as increasing, constant, or decreasing. Consecutive values are compared and a group of adjacent values forms an interval if they have identical properties w.r.t. monotonicity (e.g., *increasing*). A detailed description of the segmentation as well as problems and their solutions can be found in the doctoral thesis of Miene [Mie04].

[5]The values just exist for discrete time points, i.e., it is not a continuous function. See discussion in [Mie04, p. 53].

3.3 Discussion

The learning approaches presented in Section 3.1 are summarized in Tables 3.2, 3.3, and 3.4. The tables consists of relevant information for reviewing the approaches, e.g., the type of feedback, if incremental learning is possible with the approach, the representation of the learning input, the learning algorithm, and the learning result. If a check mark is shown in brackets (✓), then there exists an extension that accomplishes the corresponding aspect.

FOIL, Progol, and *TILDE* are relevant to this thesis as they deal with learning from relational data, but they are all supervised learning approaches, i.e., they do not meet the requirement of being unsupervised. Incremental learning cannot be performed with the systems, but extensions that address this issue are not excluded. Relational data can be represented and the representation of the generated rules would be sufficient w.r.t. the requirements. Temporal data might be represented by assigning time points or intervals to the relations. However, this additional temporal information would be handled just like any other information and thus, no constraining methods that potentially lead to a better efficiency could be used directly.

The basic association rule mining algorithms like *Apriori* are unsupervised as desired but work on a set of transactions without taking relations or temporal information into account. However, the basic algorithm is an efficient approach to level-wise generation of patterns. The extensions to sequential association rule mining take a further step towards the goals of this thesis but can only handle sequences of events or itemsets, i.e., relational data is still excluded. Particularly interesting is the approach by Höppner as it is also possible to have temporal interval relations in the mined patterns. The relational association rule mining approaches (e.g., *WARMR*) are also of great relevance as they satisfy the same requirements as the ILP approaches mentioned above but additionally have the property of being unsupervised. A combination of the sequential and relational extensions would cover most of the defined requirements. The approaches by Masson and Jacquenet [MJ03] as well as by Lee and De Raedt [LD04] are highly relevant to this thesis as they handle sequential and relational data, but in their approaches, events do not have a temporal extent and class information is not represented explicitely.

Although there are similarity-based approaches that can deal with relational data (e.g., *RIBL*) and there exist supervised and unsupervised learning techniques, this class of approaches does not lend itself to be used as starting point for the learning approach to be developed in this thesis because there are actually no comprehensible patterns as learning result. A similar argument conflicts with the use of reinforcement learning. The original representations of states in reinforcement learning are not powerful enough to describe a dynamic scene with relational data, but recent extensions to relational reinforcement learning can deal with relational

Approach	Type of feedback	Incremental	Explicit time represent.	Time intervals	Relations	Background knowledge	Represent. / Input	Algorithm(s)	Learning result
Supervised inductive logic programming									
FOIL [Qui90]	superv.	-	-	-	✓	✓	Relational data	Sequential covering with information-based estimate	∼ Horn clauses
Progol [Mug95]	superv.	-	-	-	✓	✓	Prolog Horn clauses & Mode declarations	Sequential covering with inverse entailment	First-order rules
TILDE [BD98]	superv.	-	-	-	✓	✓	Horn clause logic & refinement modes	Top-down induction of decision trees	First-order decision trees
Cohen and Hirsh [CH94]	superv.	-	-	-	✓	✓	Description Logic	Least common subsumer	Concept descriptions
Frequent pattern and association rule mining									
Apriori(TID) [AS94]	unsup.	-	-	-	-	-	Transactions	Level-wise candidate generation and statistical evaluation	Large itemsets / association rules
AprioriSome / AprioriAll / DynamicSome [AS95]	unsup.	-	✓	-	-	-	Transactions with timestamps	Level-wise candidate generation and statistical evaluation	(Maximal) frequent sequential patterns
GSP [SA96]	unsup.	-	✓	-	-	-	Transactions with timestamps, item hierarchy	Frequent itemset candidate generation and statistical evaluation	Generalized frequent sequential patterns
MFS [ZKYC01]	unsup.	-	✓	-	-	-	Transactions with timestamps	Sample-based estimate itemset mining, candidate generation from estimate	Frequent sequential patterns
GSP+ / MFS+ [ZKCY02, KZYC05]	unsup.	✓	✓	-	-	-	Transactions with timestamps, re-moved/added transactions	Incremental update of sequential patterns	Frequent sequential patterns
PrefixSpan and extensions [PHMA$^+$01, PHP$^+$01, HY06b]	unsup.	(✓)	✓	(✓)	-	-	Transactions with timestamps	Prefix-projected pattern growth	Frequent sequential patterns (with time gaps)
SPADE [Zak01, PZOD99]	unsup.	(✓)	✓	-	-	-	Transactions with timestamps	Lattice search strategies and join operations	Frequent sequential patterns

Table 3.2: Survey of the different learning approaches (1/3)

Approach	Type of feedback	Incremental	Explicit time represent.	Time intervals	Relations	Background knowledge	Represent. / Input	Algorithm(s)	Learning result
SPIRIT [GRS99]	unsup.	-	✓	-	-	-	Transactions with timestamps	Extension of GSP by regular expressions (RE)	Freq. sequential patterns satisfying the REs
MINEPI / WINEPI [MTV97]	unsup.	-	✓	-	-	-	Event sequences	Frequent episode generation and statistical evaluation	Frequent episodes
Höppner [Höp01] / Winarko and Roddick [WR05]	unsup.	-	✓	✓	-	-	Labeled time intervals	Level-wise candidate generation and statistical evaluation	Temporal patterns with interval relations
Tatavarty and Bhatnagar [TB05]	unsup.	-	✓	✓	-	-	Multivariate time series	Substring pattern mining, clustering, temporal relation mining	Frequent substrings and temporal dependencies
WARMR [DT99]	unsup.	-	-	-	✓	✓	Prolog / Deductive relational database	Level-wise atomset candidate generation and statistical evaluation	Frequent atomsets
MIDOS [Wro97]	unsup.	-	-	-	✓	✓	Multi-relational database	Top-down, general-to-specific search	Conjunction of first-order literals
SPADA [ML01]	unsup.	-	-	-	✓	✓	Prolog / Deductive relational database	Frequent atomset candidate generation and statistical evaluation	Spatial association rules
Kaminka et al. [KFCV03]	unsup.	-	✓	-	✓	-	Event sequences	Trie-based frequency counts & statistical dependency methods	Frequent event sequences
De Amo et al. [dFGL04]	unsup.	-	✓	-	(✓)	-	Transactions with timestamps and group IDs	GSP-based	Frequent multi-sequence patterns
SPIRIT-LOG [MJ03]	unsup.	-	✓	-	✓	-	Logical event sequences, regular expression constraints	Level-wise candidate generation	Logical sequences satisfying the REs

Table 3.3: Survey of the different learning approaches (2/3)

data. However, in RL it is also not the case that explicit representations of patterns are learned but policies how an agent should behave in certain situations.

The probabilistic approaches are especially valuable when uncertain information has to be handled. The basic Bayesian networks have been extended in different

Approach	Type of feedback	Incremental	Explicit time represent.	Time intervals	Relations	Background knowledge	Represent. / Input	Algorithm(s)	Learning result
MineSeqLog [LD04, Lee06]	unsup.	-	✓	-	✓	✓	Logical event sequences, background knowledge	Level-wise candidate generation with optimal refinement operator	Set of max. (min.) patterns (not) satisfying anti-monoton. (monoton.) constraints
Similarity-based approaches									
RIBL [EW96]	unsup. / superv.	✓	-	-	✓	-	Function-free Horn clauses	Similarity Measure	Instance collection
Bergmann & Stahl [BS98]	unsup. / superv.	✓	-	-	-	-	Object-oriented case repr. & class hierarchy	Similarity Measure	Instance collection
SimRank [JW01]	unsup. / superv.	✓	-	-	✓	-	Linked objects	Similarity Measure	Instance collection
LAUD [AP01]	unsup. / superv.	✓	-	-	✓	-	Feature term graph	Similarity Measure	Instance collection
Reinforcement learning									
TD, Q-Learning, Sarsa [Sut88, RN94, WD92]	reinf.	✓	✓	-	-	-	States, Actions, Policy, Reward	TD, Sarsa, Q-Learning	Value-function, action-value function, Q-table
RRL [DDB98]	reinf.	✓	✓	-	✓	-	Relational States, Actions, Policy, Reward	Q-Learning & TILDE	First-order decision tree for Q-function
Probability-based approaches									
1BC/1BC2 [FL04]	superv.	-	-	-	✓	-	First-order data without background knowledge	Top-down refinement & statistics	First-order Bayesian classifier
HMM, Kalman filter, DBN; cf. [RN03]		-	✓	-	-	-	Evidence variables	Particle filtering etc.	Probabilistic Model
DPRM [SDW03]		-	✓	-	✓	-	Observed relational objects' attributes	Adapted Rao-Blackwellised Particle Filtering	Dynamic Probabilistic Relational Model
Artificial neural networks									
FF ANN / Recurrent ANN	unsup. / superv.	✓	-	-	-	-	Numerical feature vectors	Backpropagation etc.	Trained neural network with adapted weights

Table 3.4: Survey of the different learning approaches (3/3)

ways that probabilistic reasoning over time (e.g., *Hidden Markov Models, Kalman Filters,* and *Dynamic Bayesian Networks*) and relational data can be handled (e.g., *Probabilistic Relational Models*). A recent extension even unites aspects of the two extension with the *Dynamic Probabilistic Relational Model*. However, these ap-

proaches are not suited to explicitely create the intended patterns which combine relations with different validity intervals, interval relations and generalizations about classes of objects in patterns.

Artificial neural networks can be used for supervised and unsupervised learning and with recurrent networks it is also possible to learn temporal patterns. Different approaches to knowledge extraction exist in order to create explicit understandable rules about the learned networks. However, the networks with their numeric unit representations cannot be used to represent relations between objects adequately.

Altogether it appears to be promising to combine different ideas from the fields of inductive logic programming and association rule mining. In the latter case, the extensions to sequential and relational association rule mining are particularly of interest. Thus, the approach presented in the next chapter is based on ideas of approaches of these fields.

In Section 3.2.2, different formalisms for the representation of action and time have been presented: the situation calculus, the *STRIPS* representation, and the interval temporal logic. Objects, properties, and relational data can be represented by all three formalisms. It is also possible to represent background knowledge in the different logical representations. The representation of temporal relations between the validity of predicates and the duration of actions or events can only be done by the interval temporal logic. The other two approaches assume changes between two world states to be done by single instantaneous actions with no temporal extension. For the situation calculus there exist extensions to handle concurrent actions (e.g., [Pin94]), but it is still not possible to precisely represent the temporal interrelationship between the actions.

Allen and Ferguson [AF94, p. 1-2] list properties of actions and events they feel to be essential:

1. "Actions and events take time."

2. "The relationship between actions and events and their effects is complex."

3. "Actions and events may interact in complex ways when they overlap or occur simultaneously."

4. "External changes in the world may occur no matter what actions an agents plans to do, and may interact with the planned actions."

5. "Knowledge of the world is necessarily incomplete and unpredictable in detail, thus prediction can only be done on the basis of certain assumptions."

The temporal interval logic has the highest expressiveness w.r.t. representing actions and events with temporal extensions and their complex interactions. It meets

all defined requirements and shall thus be the basic formalism for the representation of dynamic scenes in this thesis.

The different approaches to motion description provide a number of interesting ideas for the actual representation of dynamic scenes. Such a representation is highly dependent on the domain, e.g., for the choice of qualitative classes for speed or distances, regions (e.g., lanes in traffic domains or the penalty area in soccer), and background knowledge. In many cases, a mapping of quantitative data to qualitative data must be performed. All the presented approaches need such a qualitative abstraction. The qualitative motion description by Miene [Mie04] appears to be the most elaborated approach as besides well-known qualitative spatial representations also qualitative information about dynamics is generated by the monotonicity-based interval generation in a way that, e.g., acceleration of objects and approaching of object pairs can be recorded. This representation is based on Allen's temporal interval logic and is applied to analyze and interpret dynamic scenes in (robotic) soccer which is also the evaluation domain in this thesis.

Chapter 4

MiTemP: Mining Temporal Patterns

The analysis of the state of the art has shown that none of the single approaches satisfies all the requirements which have been defined in Chapter 2. However, there is a number of approaches which address relevant aspects, especially those which can learn rules from relational representations or can mine temporal association rules. Three approaches are particularly of interest: *WARMR* [DT99] for frequent pattern discovery in first-order logic, Höppner's [Höp03] approach to pattern discovery from interval-based data, and *MineSeqLog* [Lee06] for mining temporal patterns from first-order sequences. Three important aspects are addressed by these works: Dehaspe and Toivonen's approach can deal with relational data, Lee additionally takes sequences of first-order representations into account, and Höppner's approach allows for an interval-based representation of the temporal dimension and learning of temporal rules with interval relations. The approach presented here combines ideas of these works and additionally introduces new aspects as the explicit use of a schema level in order to guide the pattern mining process. More details about the relevant aspects of these approaches are given in the subsequent sections.

In the following, the developed temporal pattern mining approach is described. After providing some relevant definitions for the description of dynamic scenes and patterns different aspects like pattern matching, support computation and the pattern mining algorithm are discussed in detail. The generation and evaluation of prediction rules are addressed in Chapter 5.

4.1 Basic Definitions

The goal of this thesis is the development of an algorithm for temporal pattern mining in dynamic environments. Before the actual mining algorithm is described in detail, a number of relevant definitions is provided for the sake of clarity.

The smallest building blocks of dynamic scenes and temporal patterns are objects and variables:

Definition 4.1 (Object, Variable, Term)**.** Concrete objects in dynamic scenes are denoted as objects. $\mathcal{O}$ is the set of all objects in the universe of discourse. A variable is a placeholder for an object which allows for expressing statements about objects while abstracting from the concrete binding. $\mathcal{V}$ is the set of variables. A term can be either an object or a variable. The set of terms is defined as $\mathcal{T} = \mathcal{O} \cup \mathcal{V}$.

For terms the function $atom : \mathcal{T} \rightarrow \{true, false\}$ is defined as:

$$atom(t) = \begin{cases} \text{true} & \text{if } t \in \mathcal{O} \\ \text{false} & \text{otherwise} \end{cases}$$

□

The function $atom(t)$ can be used in order to check if t denotes a variable or an object.

We follow the notation that variable identifiers start with an upper-case letter and objects start with a lower-case letter as it is the case in many *Prolog* syntaxes and different works in the field (e.g., [Deh98, Lee06]).

Example 4.1. Examples for objects are: *a*, *p7*, *ball*, *klose*, *frings*. These are examples for variables: *X*, *Y*, *Opponent*.

In order to group objects with common properties, we introduce *concepts*. A concept consists of a *concept identifier* and a set of objects which are instances of this concept.

Definition 4.2 (Concept)**.** The set of concept identifiers is defined as $\mathcal{CI}$. A concept $c : \mathcal{CI} \times 2^{\mathcal{O}}$ is defined as a relation between concept identifiers and sets of objects. The most general concept is denoted as *object* and is defined as the set of all objects, i.e., $c_{object} = (object, \mathcal{O})$.

□

We abstract from complex concept descriptions describing common properties of concepts. In this work, a concept is defined by a (given) set of objects and a unique identifier.

Definition 4.3 (*is-a* Relation)**.** Let $c_1 = (ci_1, \mathcal{O}_1)$ and $c_2 = (ci_2, \mathcal{O}_2)$ be two concepts. c_1 is a sub-concept of c_2 if and only if all elements of c_1 are also elements of c_2 and c_2 has at least one element which is not member of c_1:

$$\textit{is-a}(c_1, c_2) \text{ iff } \textit{is-a}(ci_1, ci_2) \text{ iff } \mathcal{O}_1 \subset \mathcal{O}_2.$$

Multiple inheritances of concepts are not allowed:

$$\forall c, c_1, c_2 \in \mathcal{CI} : \textit{is-a}(c, c_1) \wedge \textit{is-a}(c, c_2) \wedge c_1 \neq c_2 \Longrightarrow (\textit{is-a}(c_1, c_2) \vee \textit{is-a}(c_2, c_1)).$$

□

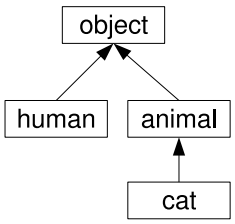

Figure 4.1: Illustration of a concept hierarchy

Without the last condition it would be possible to assign objects to different concepts which do not have a common path in the concept hierarchy.

Example 4.2. Let $\mathcal{O} = \{klose, frings, tweety, sylvester, tom, jerry\}$. The most general concept object is $c_{object} = (object, \mathcal{O})$. Let the concepts *human*, *animal*, and *cat* be defined as $c_{human} = (human, \mathcal{O}_{human})$, $c_{animal} = (animal, \mathcal{O}_{animal})$, and $c_{cat} = (cat, \mathcal{O}_{cat})$, respectively, with $\mathcal{O}_{human} = \{klose, frings\}, \mathcal{O}_{animal} = \{tweety, sylvester, tom, jerry\}$, and $\mathcal{O}_{cat} = \{sylvester, tom\}$. As $\mathcal{O}_{human} \subset \mathcal{O}_{object}$, $\mathcal{O}_{animal} \subset \mathcal{O}_{object}$, and $\mathcal{O}_{cat} \subset \mathcal{O}_{animal}$, it holds that *is-a*$(human, object)$, *is-a*$(animal, object)$, and *is-a*$(cat, animal)$. The concepts form a concept hierarchy as it is illustrated in Fig. 4.1.

The *is-a* relation allows for structuring the concepts hierarchically. As it can be seen, transitivity holds for the *is-a* relation:

Theorem 4.1 (Transitivity of the is-a relation)**.** The *is-a* relation is transitive, i.e., it holds that *is-a*$(c_1, c_2) \wedge$ *is-a*$(c_2, c_3) \Longrightarrow$ *is-a*(c_1, c_3).

□

Proof 4.1. Let $c_1 = (ci_1, \mathcal{O}_1), c_2 = (ci_2, \mathcal{O}_2), c_3 = (ci_3, \mathcal{O}_3)$. As the *is-a* relation is defined by the proper subset relation of the corresponding object sets, it is obvious that $\mathcal{O}_1$ as a proper subset of $\mathcal{O}_2$ must also be a proper subset of $\mathcal{O}_3$. It holds that $\forall x \in \mathcal{O}_1 : x \in \mathcal{O}_2$ and $\forall y \in \mathcal{O}_2 : y \in \mathcal{O}_3$. It follows that $\forall x \in \mathcal{O}_1 : x \in \mathcal{O}_3$. It holds that $\mathcal{O}_1 \subset \mathcal{O}_3$ and thus, *is-a*(c_1, c_3).

■

The membership of objects to concepts is defined by the *instance-of* relation. It can be distinguished between the direct *instance-of* relation which refers to the most special concept where the *instance-of* relation holds for an object, and the general *instance-of* relation which also holds for the super concepts.

Definition 4.4 (*direct-instance-of* Relation)**.** Let $o \in \mathcal{O}$ be an object and $c_1 = (id_1, \mathcal{O}_1)$ be a concept. o is *direct instance of* c_1 if and only if it is an element of the set $\mathcal{O}_1$ and it is not an element of any other more special set:

direct-instance-of(o, id_1) iff $o \in \mathcal{O}_1 \wedge \nexists c_2 = (id_2, \mathcal{O}_2) : o \in \mathcal{O}_2 \wedge$ *is-a*$(c_2, c_1) \wedge c_2 \neq c_1$.

□

Definition 4.5 (General *instance-of* Relation)**.** Let $o \in \mathcal{O}$ be an object and $c_{id} = (id, \mathcal{O}_{id})$ be a concept. o is *instance of* a concept c_{id} if and only if it is an element of the set $\mathcal{O}_{id}$:

$$\textit{instance-of}(o, id) \text{ iff } \textit{instance-of}(o, c_{id}) \text{ iff } o \in \mathcal{O}_{id}.$$

□

Theorem 4.2 (*instance-of* relation to all super concepts)**.** Let o be a direct instance of $c_1 = (id_1, \mathcal{O}_1)$: *direct-instance-of*$(o, id_1)$. It holds that o is also an instance of all super concepts of c_1: $\forall o \in \mathcal{O}_1 \forall c_2 = (id_2, \mathcal{O}_2) : \textit{is-a}(id_1, id_2) \Longrightarrow \textit{instance-of}(o, id_2)$.

□

Proof 4.2. It holds that $\forall x : \textit{direct-instance-of}(x, id_1) \Longrightarrow x \in \mathcal{O}_1$ (Def. 4.4). For all $c_2 = (id_2, \mathcal{O}_2)$ with $\textit{is-a}(id_1, id_2) \Longrightarrow \mathcal{O}_1 \subset \mathcal{O}_2$ (Def. 4.3). Therefore, $\forall x \in \mathcal{O}_1 : x \in \mathcal{O}_2$. By definition of *instance-of* (Def. 4.5) it follows that *instance-of*(x, id_2).

■

Predicates describe relations between objects which hold (or do not hold) for certain time intervals. It is distinguished between *predicate templates* and actual *predicates*. Predicate templates consist of an identifier and a list of concepts whose instances can be used in the predicate. Predicates consist of an identifier, a list of objects, the validity period expressed by the start and end time, and the boolean value.

Definition 4.6 (Predicate Template)**.** $\mathcal{PI}$ is the set of predicate identifiers. Let $id \in \mathcal{PI}$ be a predicate identifier and $ci_1, \ldots, ci_n$ be concept identifiers with $ci_i \in \mathcal{CI}$ and $1 \leq i \leq n$. A *predicate template* pt with arity n is defined as the tuple $pt = (id, (ci_1, \ldots, ci_n))$.

□

Definition 4.7 (Predicate)**.** Let $pt = (id, (ci_1, \ldots, ci_n))$ be a predicate template. A *predicate* p compliant with predicate template pt is defined as a relation

$$\{id\} \times \mathcal{O}_1 \times \ldots \times \mathcal{O}_n \times \mathbb{R} \times \mathbb{R} \times \{true, false\}$$

where $c_i = (ci_i, \mathcal{O}_i), 1 \leq i \leq n$ and $id \in \mathcal{PI}$.

□

The two real numbers in the predicate relation above represent the start and end times of the predicate's validity interval.

A lexicographic order of predicate identifiers is needed for a canonical representation of patterns. More generally, an order is defined on the set of identifiers $\mathcal{ID}$ with $\mathcal{PI} \subseteq \mathcal{ID}$:

Definition 4.8 (Lexicographic Order of Identifiers)**.** For the set of identifiers $\mathcal{ID}$ we define an order relation $<_{lex}$: $\mathcal{ID} \times \mathcal{ID}$. $id_1 <_{lex} id_2$ if and only if id_1 is lexicographical before id_2.

□

It is required that the end time of a valid predicate is greater than the start time:

Definition 4.9 (Valid Predicate)**.** Let $p = (id, o_1, \ldots, o_n, s, e, b)$ be a predicate compliant with $pt = (id, (\mathcal{C}_1, \ldots, \mathcal{C}_n))$. p is a *valid predicate* denoted by $valid(p)$ if and only if $s < e$.

□

For a better understanding, we denote predicates in a more readable way: The predicate $(\text{predicate}, o_1, o_2, \ldots, o_n, s, e, true)$ with start time s, and end time e is represented by $holds(predicate(o_1, o_2, \ldots, o_n), \langle s, e\rangle)$. Analogously to Miene [Mie04], we use Allen's [All84] *holds* relation in order to represent the validity of a predicate. An example for a predicate in this notation is: $holds(inBallControl(p7), \langle 17, 42\rangle)$. For predicate templates the short notation is $predicate(ci_1, \ldots, ci_n)$ with $pt = (predicate, (ci_1, \ldots, ci_n))$. The list of concept identifiers of the predicate template is the argument list, e.g., $inBallControl(player)$.

Definition 4.10 (Inconsistent Predicate Set)**.** A set of predicates $\mathcal{P}$ is defined to be *inconsistent* iff there is any pair of predicates $p_1 = (id, o_1, \ldots, o_n, s_1, e_1, b_1) \in \mathcal{P}$ and $p_2 = (id, o_1, \ldots, o_n, s_2, e_2, b_2) \in \mathcal{P}$ such that $s_1 \leq s_2 \wedge e_1 > s_2$ and $b_1 \neq b_2$. A *contradiction* between two predicates is denoted by $contradicts(p_1, p_2)$.

□

Definition 4.11 (Validity of Sub-intervals)**.** Given a predicate

$$p = holds(predicate(o_1, \ldots, o_n), \langle s, e\rangle)$$

that holds during some time interval $\langle s, e\rangle$, it also holds for all sub-intervals of $\langle s, e\rangle$ (cf. [All84]): $p = holds(predicate(o_1, \ldots, o_n), \langle s, e\rangle) \Longrightarrow \forall s', e' : holds(predicate(o_1, \ldots, o_n), \langle s', e'\rangle)$ with $s \leq s' < e$, $s' < e' \leq e$, and $s' < e'$.

□

As a predicate also holds for all sub-intervals of its validity interval, it is possible to define a minimal predicate set. A minimal predicate set does not have any overlaps of predicates, i.e., each fact is maximally described once for each time point.

Definition 4.12 (Minimal Predicate Set)**.** Let $\mathcal{P}$ be a set of predicates. $\mathcal{P}$ is defined to be *minimal* denoted by $minimal(\mathcal{P})$ iff $\forall p \in \mathcal{P} \nexists q : (s_1 \leq s_2 \wedge e_1 > s_2) \vee (s_2 \leq s_1 \wedge e_2 > s_1)$ with $p = (id, o_1, \ldots, o_n, s_1, e_1, b)$ and $q = (id, o_1, \ldots, o_n, s_2, e_2, b)$.

□

Having defined validity for predicates as well as inconsistency and minimality for predicate sets it is possible to define all possible dynamic scenes:

Definition 4.13 (Set of Valid Predicate Sets)**.** Let $\mathcal{PT} = \{pt_1, \ldots, pt_n\}$ be the set of all predicate templates and $\mathcal{P}_1, \ldots, \mathcal{P}_n$ the corresponding sets of all possible predicates, i.e., the Cartesian products as defined in Def. 4.7. The union of these sets is $\mathcal{P} = \mathcal{P}_1 \cup \ldots \cup \mathcal{P}_n$. The power set $2^{\mathcal{P}}$ forms the set of all possible predicate combinations induced by $\mathcal{PT}$. The *set of valid predicate sets* is defined as

$\mathcal{SPS} = \{\mathcal{PS} \mid \mathcal{PS} \in 2^{\mathcal{P}} \wedge minimal(\mathcal{PS}) \wedge \forall p \in \mathcal{PS} : valid(p) \wedge \forall p \in \mathcal{PS} \nexists q : contradicts(p, q)\} \subset 2^{\mathcal{P}}$.

□

Example 4.3. Three examples for predicates are $p_1 = (approaches,\ klose,\ frings,\ 10, 20,\ true)$, $p_2 = (approaches,\ klose,\ frings,\ 13, 17, false)$, and $p_3 = (approaches,\ ball,\ frings,\ 20, 30, true)$. It holds that $contradicts(p_1, p_2)$ as the same fact is defined to be $true$ and $false$ in the time interval $\langle 13, 17\rangle$.

Two examples for predicate sets are: $\mathcal{P}_1 = \{(approaches,\ klose,\ frings,\ 10, 20,\ true), (approaches,\ klose,\ frings,\ 15, 30, true)\}$ and $\mathcal{P}_2 = \{(approaches,\ klose,\ frings,\ 10,\ 20,\ true), (approaches,\ klose,\ frings,\ 25, 30, true)\}$. The first set $\mathcal{P}_1$ is not minimal as two intervals of the same fact overlap. For the second set it holds that $minimal(\mathcal{P}_2)$.

An example for an invalid predicate is $p = (approaches,\ klose,\ frings,\ 20, 10, true)$ because $20 \not\leq 10$.

Definition 4.14 (Interval Relation Function)**.** The *interval relation function* $ir : \langle\mathbb{N}, \mathbb{N}\rangle \times \langle\mathbb{N}, \mathbb{N}\rangle \rightarrow \mathcal{IR}$ maps time interval pairs to a set of interval relations $\mathcal{IR}$. The mapping of interval pairs to relations must be jointly exhaustive and mutually exclusive.

□

It depends on the used interval relations $\mathcal{IR}$ how the actual mapping from the interval pairs to the interval relation has to be performed. Using, for instance, Allen's interval relations $ir(\langle s_1, e_1\rangle, \langle s_2, e_2\rangle) = b$ (*before*) if and only if $e_1 < s_2$ (see Section 3.2.1 on page 61; [All83]).

Example 4.4. Let $\mathcal{IR} = \{$older, head-to-head, younger$\}$. A valid interval relation function would be, for instance:

$$ir(\langle s_1, e_1\rangle, \langle s_2, e_2\rangle) = \begin{cases} \text{older} & \text{if } s_1 < s_2 \\ \text{head-to-head} & \text{if } s_1 = s_2 \\ \text{younger} & \text{if } s_1 > s_2 \end{cases}$$

A *dynamic scene* describes properties of objects and relations between objects. It can be distinguished between predicates that do change over time – so called *fluents* – and those that do not change over time denoted as *static* predicates. The advantage of distinguishing between fluent and static predicates is that it is not necessary to consider temporal interrelations between static predicates. In other words, the validity intervals of static predicates can be seen as identical infinite intervals (without start and end time) and all fluent predicates happen to be valid during this perpetual "static" interval. The static predicates do not need any validity intervals like the fluent ones do. In this work, we just address fluent predicates.

An important differentiation in the proposed formalism is the separation between schema and instance layer. While the schema defines the set of possible dynamic

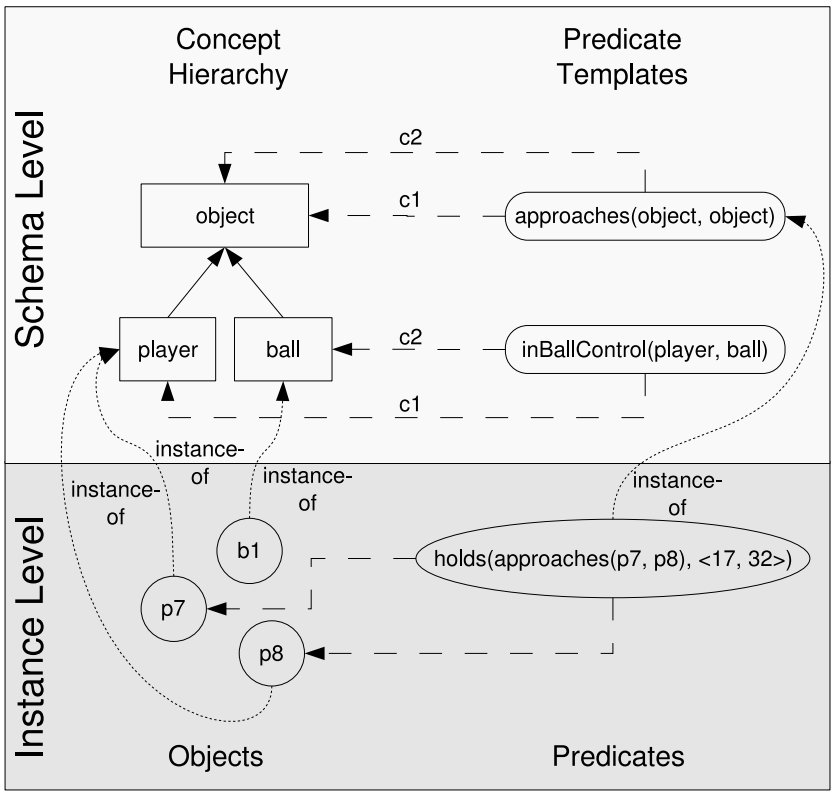

Figure 4.2: Illustration of schema and instance level of a dynamic scene

scenes on an abstract level, the instance layer addresses concrete objects with their concrete properties and relations including validity intervals for fluents. This distinction is necessary in order to provide conceptual information, for instance, about the allowed objects' concepts in predicates. This information can be used in order to guide the mining process, i.e., to cut off certain branches in the candidate pattern generation process. In the schema level, information about concepts, sub-concept relationships, and predicate templates has to be provided. In the instance level, instance-of relations of objects and predicates are represented. The schema and instance levels are illustrated in Fig. 4.2. The concepts *player* and *ball* are sub-concepts of *object*. There are two predicate templates: *approaches* can relate two instances of the concept *object*; *inBallControl* can describe a relation between a *player* and a *ball*. At the instance level three object instances – *b1*, *p7*, and *p8* – are shown. Additionally, a predicate instance of the predicate *approaches* holds from time point 17 to 32 between objects *p7* and *p8*.

Definition 4.15 (Dynamic Scene Schema)**.** Let $\mathcal{CI}$ be the set of concept identifiers, *is-a* : $\mathcal{CI} \times \mathcal{CI}$ be the sub-concept relation which describes the concept hierarchy, $\mathcal{PT}$ be the set of predicate templates, and *ir* be the interval relation function. The

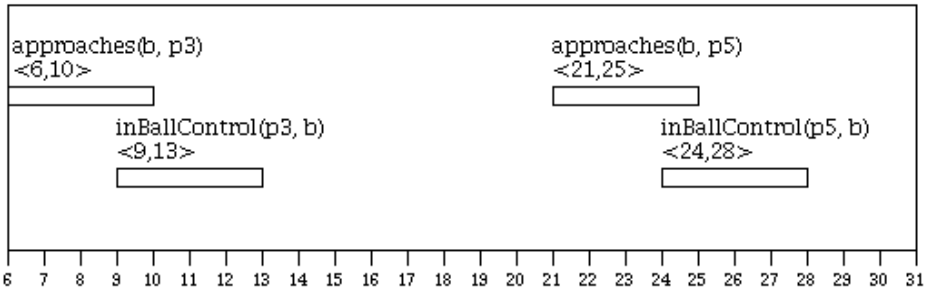

Figure 4.3: Graphical representation of the dynamic scene example

schema of a dynamic scene is defined as the quadruple $dss = (\mathcal{CI}, \textit{is-a}, \mathcal{PT}, ir)$ and consists of all schematic information. $\mathcal{CI}$ consists of at least one element which denotes the most general concept (*object*): $object \in \mathcal{CI}$. It is not allowed to have two different concept identifiers referring to the same set of objects: $\forall ci_1, ci_2 \in \mathcal{CI} : ci_1 \neq ci_2 \Longrightarrow \mathcal{O}_1 \neq \mathcal{O}_2$ with $c_1 = (ci_1, \mathcal{O}_1)$ and $c_2 = (ci_2, \mathcal{O}_2)$.

□

Definition 4.16 (Dynamic Scene)**.** Let $\mathcal{P}$ be a set of predicates, $\mathcal{O}$ be a set of objects in the dynamic scene, *direct-instance-of* : $\mathcal{O} \times \mathcal{CI}$ be the mapping of objects to concept identifiers, and dss be a dynamic scene schema. A *dynamic scene* is described by the quadruple $ds = (\mathcal{P}, \mathcal{O}, \textit{direct-instance-of}, dss)$.

□

An example of the soccer domain corresponding to Fig. 4.2 should clarify the definitions of dynamic scene, dynamic scene schema, predicate template, predicate, and interval relation:

Example 4.5. Let $dss = (\mathcal{CI}, \textit{is-a}, \mathcal{PT}, ir)$ be a dynamic scene schema with $\mathcal{CI} = \{object,\ player, ball\}$, $\textit{is-a}(ball, object)$, $\textit{is-a}(player, object)$, and $\mathcal{PT} = \{approaches(object, object)$, $inBallControl(player,\ ball)\}$. The interval relation function ir is defined as in Example 4.4 with $\mathcal{IR} = \{$older, head-to-head, younger$\}$.

A valid dynamic scene for this schema is $ds = (\mathcal{P}, \mathcal{O}, \textit{direct-instance-of}, dss)$ with $\mathcal{P} = \{p_1, p_2, p_3, p_4\}$, $\mathcal{O} = \{p3, p5, b\}$, $\textit{direct-instance-of}(p3, player)$, $\textit{direct-instance-of}(p5, player)$, $\textit{direct-instance-of}(b, ball)$. The actual dynamic scene – the set of predicates – can be written (in the short form) as:

$$\begin{aligned} p_1 = &\quad holds(approaches(b, p3), &\quad \langle 6, 10\rangle) \\ p_2 = &\quad holds(inBallControl(p3, b), &\quad \langle 9, 13\rangle) \\ p_3 = &\quad holds(approaches(b, p5), &\quad \langle 21, 25\rangle) \\ p_4 = &\quad holds(inBallControl(p5, b), &\quad \langle 24, 28\rangle) \end{aligned}$$

A graphical representation of this dynamic scene is shown in Fig. 4.3.

Up to this point it has been defined how dynamic scenes can be described on an abstract level and how concrete dynamic scenes can look like. The next step is to define a representation for patterns which can occur in dynamic scenes.

An atomic pattern consists only of one predicate pattern. The difference to predicates is that the list of arguments do not need to denote objects and that no start and end times for the validity interval are assigned. In the general case, the elements of the pattern are variables that can be bound to objects while pattern matching. However, it is also allowed to have arguments bound to objects in a pattern already. Atomic patterns can be combined to conjunctive patterns. Atomic patterns and conjunctive patterns are defined as follows:

Definition 4.17 (Atomic Pattern)**.** Let $pt = (id, (ci_1, \ldots, ci_n))$ be a predicate template and p_{arg} be a list of terms $p_{arg} = (t_1, t_2, \ldots, t_n)$. An *atomic pattern* is defined as $ap = (pt, p_{arg})$. All t_i are either elements of $\mathcal{O}$ as defined in the dynamic scene or are elements of $\mathcal{V}$, the set of variables: $\forall t_i \in \mathcal{V} \cup \mathcal{O} = \mathcal{T}$ with $1 \leq i \leq n$.

□

The short notation of atomic patterns is similar to the representation of predicates, but in this case no time interval needs to be represented and the argument list of the atomic pattern can also include variables. An atomic pattern $ap = (predicate, (t_1, \ldots, t_n))$ can also be written as $predicate(t_1, \ldots, t_n)$.

Example 4.6. Two examples for atomic patterns are: $ap_1 = (pt, (X, Y))$ and $ap_2 = (pt, (X,\ p3))$ with predicate template $pt = (approaches, (object, object))$. In the first case, the two arguments of the atomic pattern are variables. In the second case, the first argument is a variable and the second refers to an object. In the short notation, the atomic patterns can be represented by: $approaches(X, Y)$ and $approaches(X, p3)$.

A conjunction of atomic patterns is called *conjunctive pattern*:

Definition 4.18 (Conjunctive Pattern)**.** Let $ap_1, ap_2, \ldots, ap_n$ be atomic patterns. A *conjunctive pattern* is defined by an n-tuple of atomic patterns: $(ap_1, ap_2,\ \ldots,\ ap_n)$. The number of atomic patterns $n = size((ap_1, ap_2, \ldots, ap_n))$ is denoted as the *size* of the pattern.

The number of terms of a conjunctive pattern $cp = (ap_1, \ldots, ap_n)$ of size n can be computed by: $m = length(cp) = \sum_{i=1}^{n} arity_i$ where $ap_i = (pt_i, (t_1, \ldots, t_{arity_i}))$.

The m-tuple of arguments of a conjunctive pattern cp is defined as $arg(cp) = (t_{1,1}, \ldots, t_{1,arity}, t_{2,1}, \ldots, t_{2,arity}, \ldots, t_{n,1}, \ldots, t_{n,arity})$ with $ap_i = (pt_i, (t_{i,1}, \ldots, t_{i,arity}))$.

□

Similarly to the predicates above, we introduce a short notation for conjunctive patterns:

$$predicate\_1(t_{1,1}, t_{1,2}, \ldots, t_{1,arity}) \wedge predicate\_2(t_{2,1}, t_{2,2}, \ldots, t_{2,arity}) \wedge \ldots \\ \ldots \wedge predicate\_n(t_{n,1}, t_{n,2}, \ldots, t_{n,arity})$$

Example 4.7. An example of a conjunctive pattern with two atomic patterns is $cp = (pt_1, (X, Y), (pt_2, (Y, X))$ with $pt_1 = (approaches, (object, object))$, $pt_2 = (inBallControl, (player, ball))$ and in the alternative notation $approaches(X, Y) \wedge inBallControl(Y, X))$. The number of atomic patterns is $size(cp) = 2$ and the number of terms is $length(cp) = 4$.

The order of the predicates in a pattern defines already an implicit temporal order. The reason for this restriction is that the representation of the patterns should be canonical, i.e., there should not exist more than one representation for a pattern. If it was allowed to have an arbitrary order of predicates, it would be easy to come up with examples for redundant patterns (e.g., $uncovered(X)$ *before* $pass(Y, X)$ is identical to $pass(X, Y)$ *after* $uncovered(Y)$). In the proposed solution, a predicate must have an earlier or identical start time as all its succeeding predicates. This has the advantage that just about half of the temporal interval relations have to be taken into account in the pattern mining process. Therefore, we define $\mathcal{IR}_{older} \subseteq \mathcal{IR}$ including those temporal relations where the start time of the first interval s_1 is before the start time of the second interval s_2, i.e., $s_1 < s_2$ and for the "head to head" temporal relations we define $\mathcal{IR}_{\models} \subseteq \mathcal{IR}$ where the start times are equal, i.e., $s_1 = s_2$. The union of these two sets (where $s_1 \leq s_2$) is defined as $\mathcal{IR}_{\leq} = \mathcal{IR}_{older} \cup \mathcal{IR}_{\models}$.

A similar canonical representation for temporal patterns has also been defined by Höppner [Höp03]. As Höppner takes only simple events into account in his solution, it is required that two intervals with identical "states" can only have the temporal relation *before*. In case of the other temporal relations (without inverse ones), there would be a connection or overlap of identical states which is not allowed in Höppner's approach. In these cases, the two intervals with identical state must be merged to one larger interval [Höp03]. In this work, it is not only distinguished between simple states but predicates with a list of arguments. It is allowed that more than one predicate of the same predicate template holds concurrently. The minimality condition defined in Def. 4.12 disallows temporarily overlapping predicates with identical arguments. More details how the canonical representation is guaranteed are provided in Def. 4.23 and respective remarks.

Definition 4.19 (Substitution)**.** Let $X_1, \ldots, X_m$ be variables and $t_1, \ldots, t_m$ be terms with $X_i \in \mathcal{V}, t_i \in \mathcal{T}, 1 \leq i \leq m$. A *substitution* $\theta = \{X_1/t_1, \ldots, X_m/t_m\}$ is defined as a set of variable-term pairs X_i/t_i. If it is applied to a conjunctive pattern cp, all occurrences of variables X_i are substituted by terms t_i, i.e., $cp' = cp\theta$ is equal to cp except that all occurrences of X_i are replaced by t_i.

□

Example 4.8. Let $cp = hasBall(X) \wedge pass(X, Y)$ and $\theta = \{X/frings, Y/klose\}$. Applying θ to cp leads to the substituted conjunctive pattern $cp\theta = hasBall(frings) \wedge pass(frings, klose)$.

As mentioned in the requirements section, different kinds of information should

be taken into account in order to create more specific patterns, namely conceptual information, variable unification, and temporal interval relations. This additional information is described by different kinds of *restrictions*.

A concept restriction determines the least general concept for each variable in a conjunctive pattern, i.e., while pattern matching it is not allowed to bind an object to the variable which is not an instance of this concept (or one of its sub-concepts).

Definition 4.20 (Concept Restriction)**.** Let $cp = (ap_1, \ldots, ap_m)$ be a conjunctive pattern, $\mathcal{T}$ be the set of terms, and $\mathcal{CI}$ be the set of concept identifiers. The *concept restriction* defines for each unique term t_i of cp its least general concept c_i represented by the concept identifier ci_i. The concept restriction is defined by the relation $\mathcal{CR} : \mathcal{T} \times \mathcal{CI}$. Let $args(cp) = (t_1, t_2, \ldots, t_n)$ be the n-tuple of terms in the conjunctive pattern cp and $\mathcal{A} = \{t_i \mid 1 \leq i \leq n\}$ the set of unique terms. It then holds that $\forall a \in \mathcal{A}\ !\exists ci : (a, ci) \in \mathcal{CR}$ with $ci \in \mathcal{CI}$.

□

Example 4.9. Let $cp = approaches(A, B) \wedge inBallControl(C, D)$ be a conjunctive pattern. Valid concept restrictions for cp are, for instance, $\mathcal{CR}_1 = \{(A, ball), (B, player), (C, player), (D, ball))\}$ and $\mathcal{CR}_2 = \{(A, player), (B, object), (C, player), (D, ball)\}$.

Variable unifications define if certain variables in a (conjunctive) pattern should refer to the same object in the assignment during pattern matching, i.e., if variables are unified.

Definition 4.21 (Variable Unification)**.** Let $V_1, V_2 \in \mathcal{V}$ be two variables occurring in conjunctive pattern cp. A *variable unification* of a conjunctive pattern cp is defined as the unification of two different arguments V_1 and V_2 of one or more predicates of cp. A conjunctive pattern cp' with unified variables is created by applying substitution $\theta = \{V_2/V_1\}$ to cp: $cp' = cp\theta$.

□

Example 4.10. Let $cp_1 = approaches(A, B) \wedge inBallControl(C, D)$ be a conjunctive pattern . An example for a variable unification is $cp_2 = approaches(A, B) \wedge inBallControl(A, D)$ where $C = A$, i.e., C has been unified with A.

Binding a variable to a constant (i.e., to an instance) is denoted as instantiation:

Definition 4.22 (Instantiation)**.** Let $V \in \mathcal{V}$ be a variable occurring in the conjunctive pattern cp. A variable V is *instantiated* if it is bound to an instance of the set of objects in the dynamic scene: If $V = o$ with $o \in \mathcal{O}$, all occurrences of the variable in the conjunctive pattern cp must be substituted by o: $cp' = cp\theta$ with $\theta = \{V/o\}$.

□

Example 4.11. Let $cp_1 = approaches(A, B) \wedge inBallControl(A, D)$ be a conjunctive pattern. If A is instantiated to $p3$ ($A = p3$), the resulting conjunctive pattern is $cp_2 = approaches(p3, B) \wedge inBallControl(p3, D)$.

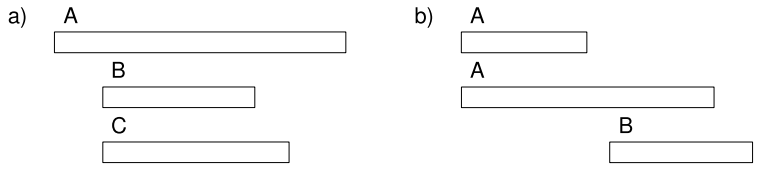

Figure 4.4: Redundancy problems in temporal patterns

A temporal restriction defines the constraints w.r.t. the validity intervals of two predicates in a conjunctive pattern. As mentioned above, the predicate order already defines certain temporal constraints among predicates. A temporal restriction further constrains the set of possible temporal relations between atomic pattern pairs.

Definition 4.23 (Temporal Restriction)**.** Let cp be a conjunctive pattern with $size(cp) = n$. The *temporal restriction* of cp is defined as the set of pairwise temporal relations between all atomic patterns. A *temporal restriction* is defined as the relation $\mathcal{TR} : \mathbb{N} \times \mathbb{N} \times 2^{\mathcal{IR}_{\leq}}$.

For all $(i, j, \mathcal{TR}_{i,j}) \in \mathcal{TR}$ it is required that $1 \leq i < n$ and $i < j \leq n$. For each atomic pattern pair ap_i, ap_j of $cp = (ap_1, \ldots, ap_n)$ where ap_i appears before ap_j in the pattern, i.e., $i < j$, the possible temporal relations between these two intervals are defined by the set $\mathcal{TR}_{i,j} \subseteq \mathcal{IR}_{\leq}$. If the identifier id_j of ap_j is smaller than id_i of ap_i w.r.t. the lexicographic order, i.e., $id_j <_{lex} id_i$, it is required that $\mathcal{TR}_{i,j} \subseteq \mathcal{IR}_{older}$ in order to have a canonical representation of the sequences. If the predicate identifiers are identical, i.e., $id_i = id_j$, and the start times are identical, it is also required that the end time of ap_j is not smaller than the end time of ap_i.

□

The restriction to $\mathcal{IR}_{\leq}$ (instead of $\mathcal{IR}$) in the definition above is due to the implicit temporal order of the atomic patterns.

The temporal restriction constrains the set of possible matches for a pattern. A set of predicates just matches the pattern if – besides the other conditions – the interval relation function of all time interval pairs of the predicate pairs results in an interval relation which is an element of the corresponding set of temporal relations. How patterns can be matched to dynamic scenes is described in detail in Section 4.3.2.

Example 4.12. Two examples should clarify why additional constraints are needed in order to avoid redundant patterns. In the first case (Fig. 4.4a), three intervals denoted by A, B, and C can be found in the scene. Intervals B and C have the same start time. Depending on the chosen interval relations $\mathcal{IR}$ two representations could represent the same pattern if there was no requirement for a lexicographic order. In Section 4.2, we propose a concise set of temporal interval relations that does not take into account the interval end times if two intervals have the same start time (head-to-head relation); these relations are used in the following examples. In

	B	C
A	contains ($\mathcal{TR}_{1,2}$)	contains ($\mathcal{TR}_{1,3}$)
B	–	started-by ($\mathcal{TR}_{2,3}$)

Table 4.1: Visualization of a temporal restriction

this case, these two conjunctive pattern descriptions with their temporal restrictions would represent the same pattern:

1. $cp_1 = A \wedge B \wedge C$ and $\mathcal{TR}_1 = \{(1, 2, \{\text{older \& contemporary}\}), (1, 3, \{\text{older \& contemporary}\}), (2, 3, \{\text{head-to-head}\})\}$.

2. $cp_2 = A \wedge C \wedge B$ and $\mathcal{TR}_2 = \{(1, 2, \{\text{older \& contemporary}\}), (1, 3, \{\text{older \& contemporary}\}), (2, 3, \{\text{head-to-head}\})\}$.

A lexicographic order only allows the first pattern description. In Fig. 4.4b, there is a similar situation. If the end times of the two A intervals are not considered, these two conjunctive pattern descriptions and temporal restrictions could represent the situation:

1. $cp_1 = A \wedge A \wedge B$ and $\mathcal{TR}_1 = \{(1, 2, \{\text{head-to-head}\}), (1, 3, \{\text{before}\}), (2, 3, \{\text{older \& contemporary}\})\}$.

2. $cp_2 = A \wedge A \wedge B$ and $\mathcal{TR}_2 = \{(1, 2, \{\text{head-to-head}\}), (1, 3, \{\text{older \& contemporary}\}), (2, 3, \{\text{before}\})\}$.

As an abbreviation, we write $ap_1 <_{lex} ap_2$ (or $ap_1 =_{lex} ap_2$) if and only if the predicate identifier of the atomic pattern ap_1 is lexicographical before (or equal to) the identifier of the predicate template of ap_2.

Temporal restrictions can be visualized by a triangular matrix (cf. [Höp03, p. 47]). The example in Table 4.1 shows the temporal restriction for the three predicates of Fig. 4.4a where A must *contain* B and C, and B is *started-by* C. In this example, Allen's interval relations are used (cf. Section 3.2.1; [All83]).

For the mining of temporal patterns it is helpful to have a unique order of the elements of a temporal restriction. It is possible to map each atomic pattern index pair to a unique number:

Definition 4.24 (Order on Temporal Restriction Elements)**.** Let cp be a conjunctive pattern with $n = size(cp)$ atomic patterns. For the corresponding temporal restriction $\mathcal{TR}$ it is possible to map each index pair to a unique number by the function $pair\_to\_index : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. The function is defined as $pair\_to\_index(i, j) = j - i + \sum_{k=n-i+1}^{n-1} k$.

□

i	j	$pair\_to\_index(i,j)$
1	2	1
1	3	2
1	4	3
2	3	4
2	4	5
3	4	6

Table 4.2: Index pair mapping for $n = 4$

As a short notation with a single index, we write $\mathcal{TR}_{pair\_to\_index(i,j)} = \mathcal{TR}_{i,j}$ to refer to the temporal relation sets of $\mathcal{TR}$. The number of temporal relations for n elements is equal to the number of possible pairs of n elements: $\frac{n\cdot(n-1)}{2}$.

Example 4.13. Table 4.2 shows an example for the index pair mapping with $n = 4$. For this example $n = 4$ and thus, the number of pairs is $\frac{4\cdot(4-1)}{2} = 6$.

Temporal restrictions are an essential part of temporal patterns:

Definition 4.25 (Temporal Pattern)**.** *Temporal patterns* $tp_i = (cp_i, \mathcal{TR}_i, \mathcal{CR}_i)$ are defined as a triple of a conjunctive pattern $cp_i = (ap_{i,1}, ap_{i,2}, \ldots, ap_{i,size})$, a temporal restriction $\mathcal{TR}_i$, and a concept restriction $\mathcal{CR}_i$.

□

Definition 4.26 (Most General Pattern)**.** The *most general*, empty pattern is denoted as ϵ. As it does not have any atomic pattern, there is also no temporal restriction and no concept restriction.

□

Any temporal pattern p where no interval relation is possible anymore between any of two atomic patterns of the conjunctive pattern is denoted as *inconsistent*. It is also specified that all instances in the conjunctive pattern must be compliant with the corresponding concept in the concept restriction and the corresponding concept in the predicate template. For terms it is required that all concepts restricting the variable (through the concept restriction or the concepts specified in the corresponding predicate template) must mutually subsume each other in at least one direction. If this is not the case, no instance can be found that satisfies all concept specifications as the instance needed to be instance of separate concepts which are not on the same path in the concept hierarchy. This would violate Def. 4.3.

Definition 4.27 (Inconsistent Pattern)**.** Let $p = (cp, \mathcal{TR}, \mathcal{CR})$ be a temporal pattern. p is *inconsistent* (denoted by $p = \perp$) if one of the following conditions is satisfied:

1. There is at least one empty element in the temporal restriction: $\exists \mathcal{TR}_i \in \mathcal{TR} : \mathcal{TR}_i = \emptyset$.

2. There is at least one atomic term in the conjunctive pattern which is not an instance of the corresponding concept in the concept restriction. Let $args(cp_i) = (t_1, t_2, \ldots, t_l)$ be the list of arguments of cp_i. The pattern is inconsistent if $\exists i : t_i \in \mathcal{O} \wedge (t_i, ci_{cr}) \in \mathcal{CR} \wedge \neg \textit{instance-of}(t_i, ci_{cr}))$ with $1 \leq i \leq l$.

3. Furthermore for a term t_m let $\mathcal{C}_m = \{ci_{m,1}, ci_{m,2}, \ldots, ci_{m,j}\}$ be the set of all corresponding concept identifiers of the predicate templates of all occurrences of t_m in the conjunctive pattern and ci_{cr} be the concept identifier in the concept restriction. The pattern is inconsistent if $\exists ci_1, ci_2 \in \mathcal{C}_m \cup ci_{cr} : \neg(\textit{is-a}(ci_1, ci_2) \vee \textit{is-a}(ci_2, ci_1))$.

□

Definition 4.28 (Substitution in Concept Restrictions and Temporal Patterns)**.** Let $\mathcal{CR} = \{(t_1, ci_1), \ldots, (t_n, ci_n)\}$ be a concept restriction and $\theta = \{X_1/t_{\theta,1}, \ldots, X_m/t_{\theta,m}\}$ a substitution. If θ is applied to $\mathcal{CR}$, all occurrences of X_i are replaced by the corresponding term $t_{\theta,i}$, i.e., $\mathcal{CR}' = \mathcal{CR}\theta$ is equal to $\mathcal{CR}$ except that all occurrences of X_i are replaced by $t_{\theta,i}$.

The definition of a substitution can now be extended to temporal patterns. If a substitution θ is applied to a temporal pattern $tp = (cp, \mathcal{TR}, \mathcal{CR})$, the result is a new temporal pattern tp' with a substituted conjunctive pattern:

$$tp\theta = (cp\theta, \mathcal{TR}, \mathcal{CR}\theta) = tp'.$$

□

Example 4.14. Two examples for temporal patterns are: $tp_1 = (\textit{approaches}(A, B) \wedge \textit{inBallControl}(C, D), \{(1, 2, \{before\})\}, \{(A, player), (B, player), (C, player), (D, player)\})$ and $tp_2 = (\textit{approaches}(b, A) \wedge \textit{inBallControl}(A, b), \{(1, 2, \{before\})\}, \{(A, ball), (b, player)\})$.

In the remaining text of this thesis, we use the term *pattern* as an abbreviation of *temporal pattern*. In the following, conjunctive patterns (and single atomic patterns) without restrictions are referred to as *basic patterns*.

A dynamic scene schema can be used to define the space of all potential patterns:

Definition 4.29 (Pattern Language)**.** Given a dynamic scene schema

$$dss = (\mathcal{CI}, \textit{is-a}, \mathcal{PT}, ir)$$

the set of all potential temporal patterns forms the *pattern language* $\mathcal{L}_{tp}$. Let $tp = (cp, \mathcal{TR}, \mathcal{CR})$ be a temporal pattern with the conjunctive pattern cp, the temporal restriction $\mathcal{TR}$, and the concept restriction $\mathcal{CR}$. The number of atomic patterns in the conjunctive pattern is $s = \textit{size}(cp)$ and the number of terms (i.e., the number of variables or objects) in the conjunctive pattern is $l = \textit{length}(cp)$ with $s, l \in \mathbb{N}$. Let $cp = (ap_1, \ldots, ap_s)$, $ap_i = (pt_i, (a_{i,1}, \ldots, a_{i,arity}))$, $pt_i = (pi_i, (ci_{i,1}, \ldots, ci_{i,arity}))$ with

$1 \leq i \leq s$, $args(cp) = (t_1, t_2, \ldots, t_l)$, and $\mathcal{A} = \{t_i \mid 1 \leq i \leq l\}$ be the set of unique terms. $tp \in \mathcal{L}_{tp}$ if and only if:

- $\forall i : pt_i \in \mathcal{PT} \wedge \forall i, k : a_{i,k} \in \mathcal{T}$ with $1 \leq k \leq arity$.
- $\forall i, j$ with $1 \leq i < s$ and $i < j \leq s$: $!\exists(i, j, \mathcal{TR}_{i,j}) \in \mathcal{TR}$ with $\mathcal{TR}_{i,j} \subseteq \mathcal{IR}_{\leq}$ if $pi_i \leq_{lex} pi_j$ and $\mathcal{TR}_{i,j} \subseteq \mathcal{IR}_{older}$, otherwise. Additionally, $\forall tr \in \mathcal{TR} : tr = (q, r, \mathcal{TR}_{q,r})$ with $1 \leq q < s$ and $q < r \leq s$ and $\mathcal{TR}_{q,r} \subseteq \mathcal{IR}_{\leq}$, i.e., it is not allowed that $\mathcal{TR}$ includes any other element.
- $\forall a \in \mathcal{A}$ $!\exists ci : (a, ci) \in \mathcal{CR}$ with $ci \in \mathcal{CI}$. Additionally, it is required that $\forall cr \in \mathcal{CR} : cr = (a_{cr}, ci_{cr})$ with $a_{cr} \in \mathcal{A}$ and $ci_{cr} \in \mathcal{CI}$.

□

4.2 Temporal Relations

Temporal relations are the central component of the patterns in temporal pattern mining. A qualitative representation of temporal relations between intervals can abstract from unnecessary details and leads to a concise representation of similar structures in sequences. As described in Section 3.2.1 (p. 61), Allen [All83] has proposed thirteen relations in order to describe qualitatively how two intervals can be interrelated. Allen has furthermore introduced a composition table in order to perform reasoning about temporal intervals. As the major intention for temporal pattern mining in this work is the prediction of actions or events, we focus on a smaller set of temporal interval relations. This has the additional advantage that the complexity of the pattern space is reduced.

The proposed set of temporal interval relations combines Allen's and Freksa's (semi-) interval relations [All83, Fre92a]. The five relations are: *before*, *older & contemporary*, *head-to-head*, *younger & contemporary*, and *after* and are denoted by $<, <_c, \models, >_c, >$. The motivation for this set of temporal relations is on the one hand to keep the size of the temporal relation set small and on the other hand to provide relevant relations for prediction rules focusing on the start times of the intervals. It is necessary to distinguish if an interval (of a pattern) starts before another one in order to perform a temporal prediction. Replacing the set of temporal relations and the interval relation function (e.g., by Allen's interval relations) can be easily done as long as the set is mutually exclusive and jointly exhaustive.

The relations are illustrated in Fig. 4.5. Let s_1 and e_1 be the start and end time of interval i_1 and s_2 and e_2 be the start and end time of interval i_2, respectively. Depending on the start and end times, the two intervals i_1 and i_2 are in different

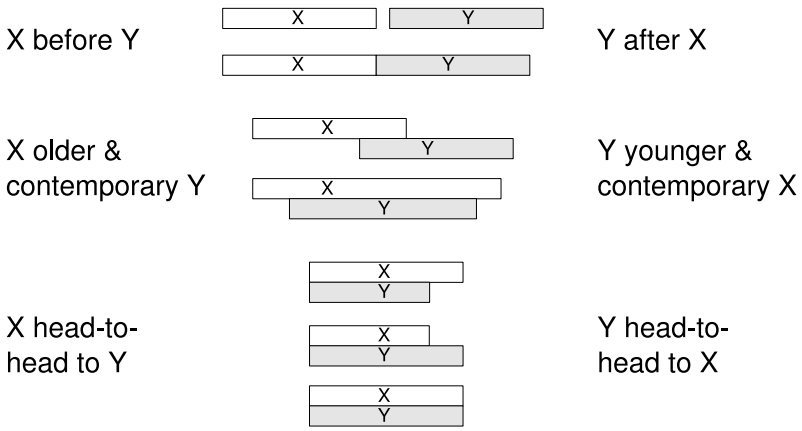

Figure 4.5: Interval relations in temporal patterns

temporal relations:

$$\begin{aligned} i_1 < i_2 &\iff e_1 \leq s_2 \\ i_1 <_c i_2 &\iff s_1 < s_2 \wedge e_1 > s_2 \\ i_1 \models i_2 &\iff s_1 = s_2 \\ i_1 >_c i_2 &\iff s_1 > s_2 \wedge s_1 < e_2 \\ i_1 > i_2 &\iff e_1 \geq s_2 \end{aligned} \tag{4.1}$$

The temporal relations are mutually exclusive and jointly exhaustive, i.e., two intervals can only be in one of the five temporal relations and for each interval pair there is a mapping to a temporal relation.

Definition 4.30 (Converse Interval Relations)**.** Let $i_1 = \langle s_1, e_1 \rangle$ and $i_2 = \langle s_2, e_2 \rangle$ be two intervals. The *converse interval relations* denoted by the "hat" symbol ($\hat{x}$) are defined as follows:

$$\begin{aligned} i_1 < i_2 &\iff i_2 > i_1, && \text{i.e., } \hat{<} = > \\ i_1 <_c i_2 &\iff i_2 >_c i_1, && \text{i.e., } \hat{<}_c = >_c \\ i_1 \models i_2 &\iff i_2 \models i_1, && \text{i.e., } \hat{\models} = \models \\ i_1 >_c i_2 &\iff i_2 <_c i_1, && \text{i.e., } \hat{>} = < \\ i_1 > i_2 &\iff i_2 < i_1, && \text{i.e., } \hat{>}_c = <_c \end{aligned} \tag{4.2}$$

Let $\mathcal{X}$ be a set of interval relations. The converse set of $\mathcal{X}$ is defined as: $\widehat{\mathcal{X}} = \{\hat{x} \mid x \in \mathcal{X}\}$.

□

A r_1 B \ B r_2 C	$<$	$<_c$	$\models$	$>_c$	$>$
before	$<$	$<$	$<$	$<$, $<_c$, $\models$, $>_c$	$<$, $<_c$, $\models$, $>_c$, $>$
older & contemporary	$<_c$	$<$, $<_c$	$<$, $<_c$	$<_c$	$<_c$, $\models$, $>_c$	$<_c$, $\models$, $>_c$, $>$
head-to-head	$\models$	$<$, $<_c$	$<$, $<_c$	$\models$	$>_c$	$>$
younger & contemporary	$>_c$	$<$, $<_c$	$<$, $<_c$, $\models$, $>_c$, $>$	$>_c$, $>$	$>_c$, $>$	$>$
after	$>$	$<$, $<_c$, $\models$, $>_c$, $>$	$>_c$, $>$	$>_c$, $>$	$>_c$, $>$	$>$

Table 4.3: Composition table for the temporal relations

The corresponding interval relation function ir is defined as:

$$ir(\langle s_1, e_1\rangle, \langle s_2, e_2\rangle) = \begin{cases} \text{before} & \text{if } e_1 \leq s_2 \\ \text{older \& contemporary} & \text{if } s_1 < s_2 \wedge e_1 > s_2 \\ \text{head-to-head} & \text{if } s_1 = s_2 \\ \text{younger \& contemporary} & \text{if } s_1 > s_2 \wedge s_1 < e_2 \\ \text{after} & \text{if } e_1 \geq s_2 \end{cases}$$

In order to reason about temporal interval relations, we have also defined a composition table (see Table 4.3). For instance, if it is known that an interval A is *before* ($<$) B and interval B is *head-to-head* ($\models$) to C, the temporal relation between A and C must be *before* (first row, $\models$-column). If the temporal relations of two intervals to a third one are known, on average there are only $\frac{55}{25} = 2.2$ possible temporal relations left between these two intervals (instead of five potential interval relation candidates).

Definition 4.31 (Composition of Interval Relations)**.** The *composition* of two interval relations r_1 and r_2 is defined by the corresponding entry in the composition table. The operator is denoted as $\circ$ and $r_1 \circ r_2 \subseteq \mathcal{IR}$ with $r_1, r_2 \in \mathcal{IR}$.

The composition of two sets of interval relations is defined as

$$\mathcal{R}_1 \circ \mathcal{R}_2 = \bigcup_{x \in \mathcal{R}_1, y \in \mathcal{R}_2} x \circ y.$$

□

The composition operator can be used for constraint propagation w.r.t. a temporal restriction:

Definition 4.32 (Temporal Constraint Propagation)**.** Let $\mathcal{TR}$ be a temporal restriction. A single constraint propagation step $prop : \mathbb{N} \times \mathbb{N} \times \mathcal{IR} \rightarrow \mathbb{N} \times \mathbb{N} \times \mathcal{IR}$ is defined as: $\mathcal{TR}' = prop(\mathcal{TR})$ such that $\forall i, j : (i, j, \mathcal{TR}_{i,j}) \in \mathcal{TR} \Longrightarrow !\exists (i, j, \mathcal{TR}'_{i,j}) \in$

$\mathcal{TR}'$ and $\forall i,j : \mathcal{TR}'_{i,j} = \mathcal{TR}_{i,j} \cap \bigcap_k \mathcal{TR}_{i,k} \circ \mathcal{TR}_{k,j} \cap \overbrace{\bigcap_l \mathcal{TR}_{j,l} \circ \mathcal{TR}_{l,i}}^{\wedge}$ with $i < j, k \neq i, k \neq j, l \neq i, l \neq j$ and $\mathcal{TR}_{m,n} = \overbrace{\mathcal{TR}_{n,m}}^{\wedge}$ if $m > n$.

We also define the multiple application of constraint propagation $prop^n(\mathcal{TR})$ as:

$$\begin{aligned} prop^0(\mathcal{TR}) &= \mathcal{TR} \\ prop^1(\mathcal{TR}) &= prop(\mathcal{TR}) \\ prop^n(\mathcal{TR}) &= prop(prop^{n-1}(\mathcal{TR})) \end{aligned}$$

The most restricted temporal relation generated by constraint propagation is denoted as $prop_{spec}(\mathcal{TR})$. It is defined as $prop_{spec}(\mathcal{TR}) = prop^k(\mathcal{TR})$ with the minimal value for $k \geq 0$ such that $prop^k(\mathcal{TR}) = prop^{k-1}(\mathcal{TR})$.

□

Example 4.15. Using these temporal relations the sequence shown in Fig. 4.4 on page 90 is represented as $p_1 = ((A, B, C), \{(1, 2, \{ \text{ older \& contemporary } \}), (1, 3, \{ \text{ older \& contemporary } \}), (2, 3, \{ \text{ head-to-head } \})\}, \mathcal{CR})$.

The relevant subsets of $\mathcal{IR} = \{<, <_c, \models, >_c, >\}$ for this pattern mining approach are $\mathcal{IR}_{older} = \{<, <_c\}$, $\mathcal{IR}_{\models} = \{\models\}$, and $\mathcal{IR}_{\leq} = \{<, <_c, \models\}$.

4.3 Support Computation

In order to identify relevant patterns, it is necessary to set up some criteria that reduce the pattern space to the interesting patterns. Support is a typical measurement in pattern mining. In the case of frequent itemset mining, the support counts the transactions in the database that *support* a frequent itemset [AIS93]. A support or frequency threshold specifies which patterns are taken into account for further processing in the incremental process of large pattern generation and for the creation of association rules. Besides the support, further aspects for the evaluation of the interestingness of prediction rules are discussed in Section 5.2. In this chapter, we focus on frequency in order to specify relevant patterns for the mining process.

As sequences can have arbitrary lengths, the matches of patterns should be restricted in a way that a pattern does not match to a set of predicates which hold at time periods that are temporarily far away from each other. Similar to Höppner [Höp03] and Mannila et al. [MTV97] we introduce the concept of a *time window* with a window size:

Definition 4.33 (Time Window, Observable Predicates)**.** Let $\mathcal{P}$ be the set of predicates in a dynamic scene. A *time window* at the temporal position w_s with width w

restricts the set of *observable predicates* for this position. The set of *observable predicates* is defined as $\mathcal{P}_{obs} = \{p \mid p = (id, o_1, \ldots, o_n, s, e, b) \in \mathcal{P} \wedge s \leq (w_s + w) \wedge e > w_s\}$. □

4.3.1 Discussion of Variants for Support Computation

The definition of the support is not unproblematic for a complex pattern representation as chosen in this thesis. Before the actual definition of support for our approach is given, different variants and their advantages and disadvantages are discussed. Due to the relational data, different objects in scenes, and due to the temporal dimension it is not obvious what a good estimation of the support can be. Agrawal et al. [AS94] and Dehaspe [Deh98], for instance, count itemsets and matches of queries. Höppner decided to use an observation time semantics with a sliding window. The different approaches are discussed briefly in the following.

Match Counting with Key Parameter

In the task of frequent pattern discovery in logic, Dehaspe [Deh98] introduced an extra *key* parameter in order to determine what is counted. Entities are uniquely identified by each binding of the variables in *key* [Deh98, p. 34]. Such a key parameter could, e.g., be different *player* objects of the predicate $inBallControl(player)$ in a pattern. In this case, the support is defined by the number of different player objects (which are actually in ball control) in all matches of the pattern. A disadvantage of this support definition is that the key parameter must be part of each pattern in order to get a support greater than zero. Thus, it is not possible to compare two different patterns if they do not share this key parameter. An advantage of this approach is its clear semantics: It is clearly stated *what* is counted (e.g., player instances that are in ball control).

Match Counting without Key Parameter

The difficulty with the counting key parameter is that it must be part of every pattern. Intuitively, it would be more elegant to find a solution that counts occurrences of patterns in a scene without this restriction. At first glance, it should be possible to only count occurrences of patterns by taking each predicate into account once only. The simple example in Fig. 4.6 would lead to two matches for the pattern $A \wedge B$ without any ambiguity as the temporal gap between the predicate instances of the two matches is large. The dashed line illustrates the window. Two other (still quite simple) examples shall illustrate the problems of this counting approach.

Fig. 4.7 shows two sequences where predicates A and B appear. If the pattern $A \wedge B$ has to be matched, in the first sequence it can be seen that either one or two matches could be found depending on the order of the pattern matching process.

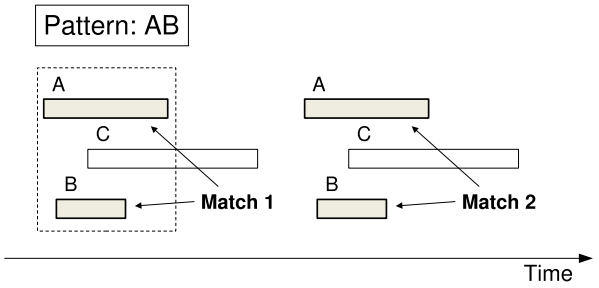

Figure 4.6: Counting without key parameter

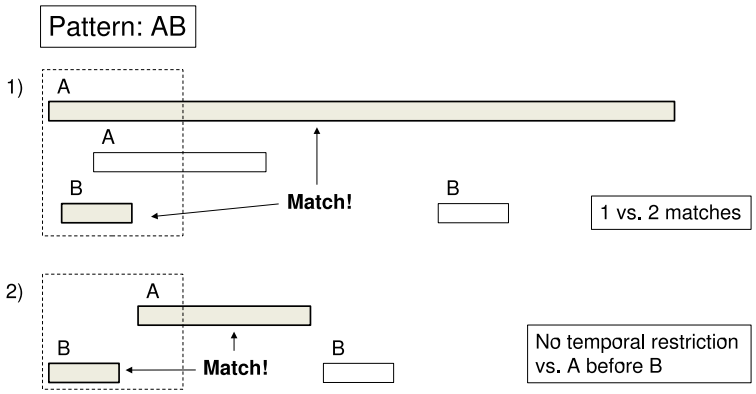

Figure 4.7: Assignment and match reuse problem

If the first match is taken in a greedy way – i.e., taking the predicate instances in the order they appear – the match would happen to consist of the first A and the first B predicate instance as marked by the shaded bars. However, if the first A was combined with the second B and the first B with the second (shorter) A we could get two valid matches! This simple example already illustrates what we call the *assignment problem*. This problem has also been identified in [Höp03, p. 51]. In the worst case, this could even lead to a violation of the anti-monotonicity condition of the support, i.e., that the support of a more special pattern would be higher than the one of its more general parents. In this case, the completeness of the pattern mining algorithm could not be guaranteed any longer, i.e., that frequent patterns might be excluded because they are assumed to be infrequent due to infrequent more general patterns. This might also lead to missed frequent patterns in the ongoing steps.

A way to handle this problem is to identify one of the *optimal* matchings, i.e., to find an assignment that leads to the maximal possible support. However, this would lead to efficiency problems as all assignment combinations had to be checked.

A second problem is illustrated in the second sequence in Fig. 4.7. Here, again the pattern $A \wedge B$ has to be matched. If we do not have any further restrictions,

the greedy match would again lead to an assignment of the first appearing predicate instances of A and B as marked by the shaded bars. If we add a restriction that A must be *before* B, the match of the more general pattern (without restriction) does not match the pattern anymore. However, there is still a valid match if the A predicate was combined with the second B predicate. This is problematic as we cannot just reuse the matches of the more general pattern(s) and thus cannot avoid a complete scan of the sequence without risking to miss matches! This problem cannot even be solved by identifying one of the optimal matches. We refer to this problem as the *match reuse problem*. However, it is possible to restrict the search in the sequence by just searching in the neighborhoods (specified by the window size) of the matches of the more general patterns.

Computing the support without a key parameter leads not only to the problem of high computational cost or possible loss of monotonicity. The semantics of the matches is not as clear as it appears at the first glance. It is not motivated why certain predicate instances should be grouped together to a match (besides maximizing the support). Depending on the application it might even be desired to allow some predicate instances to be reused (and thus, one or more counting attributes should be used).

It is possible to come up with further modifications, e.g., a dynamic selection of a counting parameter. One idea is to select from each pattern the predicate with minimal occurrences in the sequence as the counting parameter. But this can also lead to strange effects: A minimal change in the support of single predicates leading to a another counting parameter can result in enormous changes of the support value of the pattern[1]. A way to avoid this effect would be to dynamically select the "minimal predicate" (the one with least occurrences) from the set of matches as key parameter, but with such a definition of support the semantics gets even more incomprehensible as the support computation could be based on different key values for different patterns.

Observation Time with Strict Window Semantics

Höppner [Höp03] chose a different way to compute the support. His measurement is based on an observation time semantics. The support is defined by the length of all time intervals where a pattern can be observed for a given window size. It is assumed that only a part of the sequence can be seen, i.e., the sliding window determines what can be observed. Only if the pattern can be matched by the given information in the current window, the interval of the window is taken into account for the support computation. Having the complete length of the sequence and the summed-up length of the intervals where the pattern holds, the likelihood

[1]Personal communication with Frank Höppner; e-mail correspondence October 26 - November 10, 2005

of observing the pattern in a randomly chosen time window can be computed easily.

Besides the clear semantics of this support another advantage is that it is not necessary to collect all possible matches for a pattern in the sequence. If one match has been found at a window position, the matching can be stopped and the window can be moved to the next position (where the observation changes w.r.t. the window, i.e., where a predicate "leaves" or "enters" the window). On the other hand, it cannot be distinguished between the cases if just one or many matches occur at a window position.

Observation Time with Memory

As it has already been outlined by Höppner [Höp03, p. 56], time points that have been passed by the time window could be memorized and thus still be used for the matching process even if they are not visible at the current window position. The advantage of this extension is that information that actually is provided in the sequence and can easily be stored could be used to identify more pattern instances even if they do not fit in the selected window size.

Maximal match length

It should also be briefly discussed what the effect of a slightly different support definition would be. If the interval of the whole match (from the earliest start time point to the latest end time point of the predicate) was taken as basis for support computation, huge support values could be generated. Here, again, it was not guaranteed that the support value decreases if the pattern complexity increases! If a pattern is specialized by adding a predicate, the support could increase. For instance, if the pattern $((A, B), \{(1, 2, \{before\})\}, \mathcal{CR})$ would be extended to $((A, B, C), \{(1, 2, \{before\}), (1, 3, \{before\}), (2, 3, \{before\})\}, \mathcal{CR})$ the latest end time of a match (by predicate C) could cover a region that has not been covered by the more general pattern. Another problem is that depending on the match the support values could differ enormously. In order to have a fair support computation, it would be necessary – again – to check all matches and select the one with the highest (or lowest) length. These problems show that this variant is not really an alternative.

Due to the problems with the support definitions based on match counting – with or without key parameter – the observation time semantics (with memory) is chosen in this thesis for support computation. The convincing advantages of using observation time as support are the clear semantics and the better efficiency as not all matches have to be collected or maybe even further processed. The anti-monotonicity property for this support definition holds (as it will be shown in Section 4.4) and the support intervals of previous steps (i.e., of more general patterns) can

be reused in order to restrict the search to parts of the temporal sequence during support computation.

The next section addresses the matching of patterns as this is needed for the support definition presented in Section 4.3.3.

4.3.2 Pattern Matching

After having defined dynamic scenes, their schemata, and temporal patterns, we can define how to match such patterns to a dynamic scene. Pattern matching is essential for the computation of the support of a pattern. Basically, a match can be seen as a successful query to a database [Deh98]. In order to match a temporal pattern, all predicates in the conjunction must occur within a defined window size, the temporal restrictions between these predicates must be satisfied, and for the variable assignment the concept restriction must not be violated.

Definition 4.34 (Pattern Match)**.** Let $\mathcal{P}$ be the set of predicates in a dynamic scene and $p = (cp, \mathcal{TR}, \mathcal{CR})$ be a temporal pattern. A *match* $\mathcal{M} : \mathbb{N} \times \mathcal{P}$ of pattern p is given if and only if

- there exists a (potentially empty) substitution θ leading to $p' = p\theta = (cp', \mathcal{TR}, \mathcal{CR}')$ with $cp' = cp\theta = (ap'_1, \ldots, ap'_n)$ and $\mathcal{CR}' = \mathcal{CR}\theta$ such that a valid assignment for each atomic pattern $ap'_i = (pt_i, (a_{i,1}, \ldots, a_{i,m}))$ with $1 \leq i \leq n$ and $pt_i = (pi_i, (ci_{i,1}, \ldots, ci_{i,m}))$ to a corresponding predicate $p_i = (pi_i, a_{i,1}, \ldots, a_{i,m}, s_i, e_i, true) \in \mathcal{P}$ of the dynamic scene is given where both refer to the same predicate identifier pt_i and all arguments $a_{i,1}, \ldots, a_{i,m}$ are identical. The match $\mathcal{M}$ is defined as $\mathcal{M} = \{(1, p_1), \ldots, (n, p_n)\}$. It is required that no predicate is assigned more than once: $\forall i, j : p_i \neq p_j$ with $i \neq j$ and $1 \leq i \leq n$ and $1 \leq j \leq n$;

- the match is within the sliding window range. Let w be the sliding window width and s_{max}, e_{min} be the maximal start time end minimal end time of the predicates in the match. It is required that $s_{max} - e_{min} \leq w$, i.e., all predicates must be observable within a window position;

- the concept restriction is satisfied. Let $\mathcal{O}_{match} = (o_1, o_2, \ldots, o_l) = (a_{1,1}, \ldots, a_{1,m}, a_{2,1}, \ldots, a_{n,1}, \ldots, a_{n,m},)$ be the list of objects in the assigned predicates and let $cr = (ci_1, ci_2, \ldots, ci_l)$ be the tuple of concept identifiers of the concept restriction of the pattern with $arg(cp) = (a_1, \ldots, a_l)$ and $(a_k, ci_k) \in \mathcal{CR}$. It is required that $\forall k : \textit{instance-of}(o_k, ci_k)$ with $1 \leq k \leq l$;

- the temporal restriction $\mathcal{TR}$ is satisfied. It is required that $\forall r, s : ir(\langle s_r, e_r \rangle, \langle s_s, e_s \rangle) \in \mathcal{TR}_{r,s}$ with $(r, s, \mathcal{TR}_{r,s}) \in \mathcal{TR}$, $1 \leq r < n$, and $r < s \leq n$.

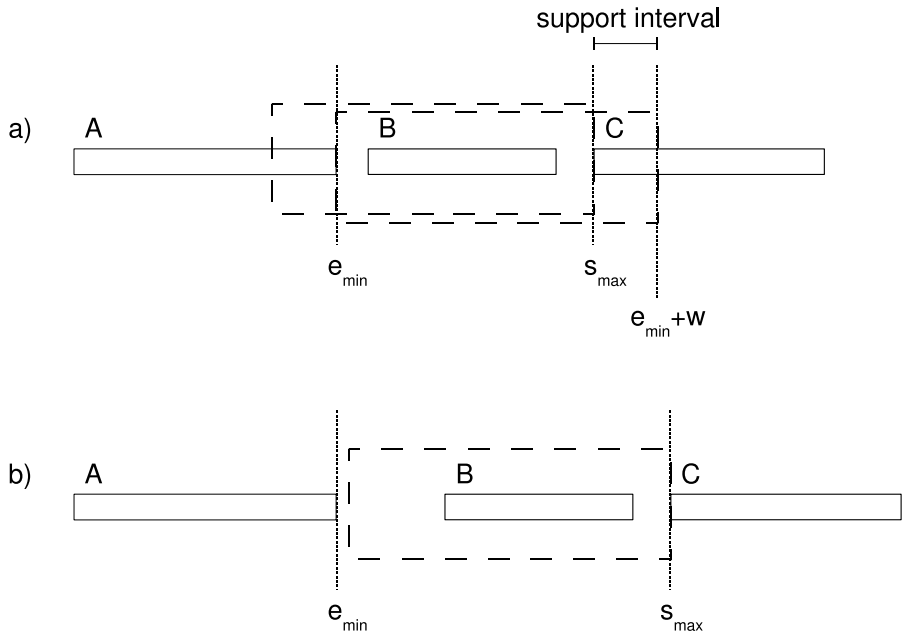

Figure 4.8: Illustration of the sliding window

The set of all matches of a pattern p for all window positions is denoted by $matches(p)$.

□

Fig. 4.8 illustrates how a pattern matches within a sliding window. Let the pattern to be matched be $p = (A \wedge B \wedge C, \{(1, 2, \{before\}), (1, 3, \{before\}), (2, 3, \{before\})\}, \{\})$. In Fig. 4.8a the difference between the minimal end time and the maximal start time is below the window size and thus, the pattern matches in the interval from $(s_{max}, e_{min} + w]$. If the temporal distances between the intervals are higher (Fig. 4.8b), not all three predicates can be observed within the sliding window and thus, the pattern does not match.

Example 4.16. Let the window size be defined as $w = 12$. If we take into account the predicate sequence shown in Fig. 4.9 and would like to match the temporal pattern $tp = (closerToGoal(X, Y) \wedge uncovered(X, X) \wedge pass(Y, X), \{(1, 2, \{$ older & contemporary $\}), (1, 3, \{$ older & contemporary $\}), (2, 3, \{$ older & contemporary $\})\}, \{(X, object), (Y, object)\})$, this would be a valid match: $\mathcal{M} = \{(1, holds(closerToGoal(p8,\ p9),\ \langle 11, 19\rangle)),\ (2, holds(uncovered(p8, p8), \langle 13, 21\rangle)),\ (3, holds(pass(p9, p8), \langle 15, 17\rangle))\}$ with $\theta = \{X/p8,\ Y/p9\}$. The maximal start time and minimal end time of the match are $s_{max} = 15$, $e_{min} = 17$. $s_{max} - e_{min} \leq w$, i.e., all predicates fit the given maximal window size.

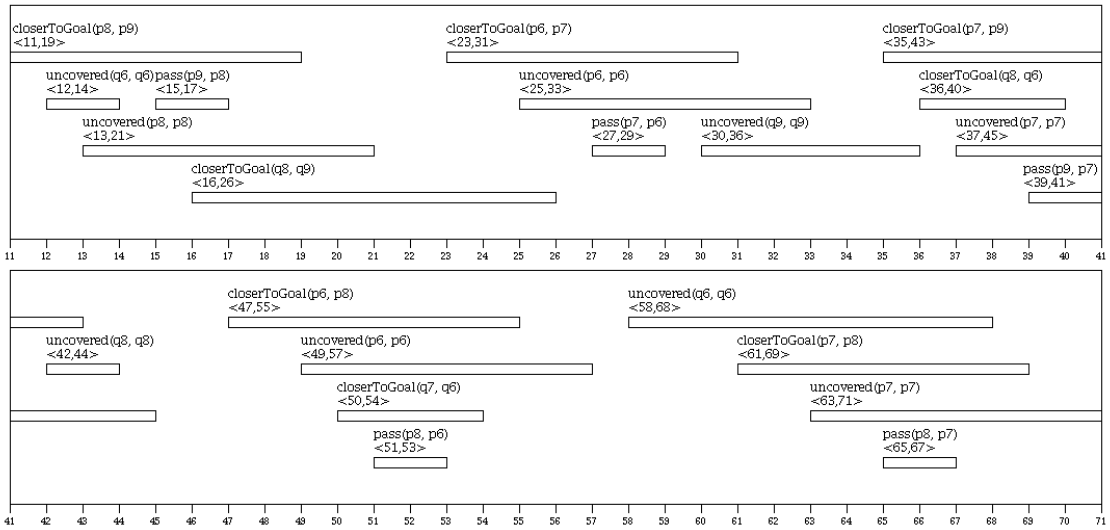

Figure 4.9: Visualization of the predicate sequence

4.3.3 Support and Frequency Definition

The following support definition computes the length of intervals where at least one match for a pattern can be found for a given window size. The frequency is the probability to find a match of a pattern at a random window position for a given dynamic scene and window size (cf. [Höp03]).

Definition 4.35 (Support)**.** Let p be a temporal pattern, ds the dynamic scene, and $\mathcal{M}$ the set of matches. The validity interval of a single match $m_i \in \mathcal{M}$ is defined as $v_i = (s_{max_i}, e_{min_i} + w]$ with s_{max_i} being the maximal start time and e_{min_i} the minimal end time of all predicate instances in m_i. The *support* of p w.r.t. ds is defined as the length of the union of all validity intervals of the matches:

$$supp(p) = length\left(\bigcup_{k=1}^{|\mathcal{M}|} v_k\right).$$

□

The length of n discontinuous intervals $i_i = [s_i, e_i]$ with $1 \leq i \leq n$ can be computed by summing up the lengths of the single intervals: $\sum_{i=1}^{n}(e_i - s_i)$. It is required that none of the intervals overlaps with another: $\forall i : e_i \leq s_{i+k}$ with $k > 0$. Overlapping intervals can be merged together easily by replacing the overlapping intervals by a single new one with minimal start and maximal end time:

$$merge([s_1, e_1], [s_2, e_2]) := [min(s1, s2), max(e1, e2)] \text{ if } e1 \geq s2$$

Definition 4.36 (Frequency)**.** Let p be a temporal pattern with support $supp(p)$, seq_{length} be the sequence length of the dynamic scene, and w be the window size. If the support value is divided by the sequence length of the dynamic scene plus the window size (sliding window size is added at the start of the sequence), we get the frequency of the pattern: $freq(p) = \frac{supp(p)}{seq_{length}+w}$.

□

Example 4.17. Let the dynamic scene and its schema be defined as it has been done in Example 4.5 on page 86 (also see Fig. 4.3). For the pattern $p = (approaches(X, Y) \wedge inBallControl(Y, X), \{(1, 2, \{$ older & contemporary $\})\}, \{(X, ball), (Y, player)\})$ the following set of matches can be found for window size $w = 10$: $\mathcal{MS} = \{\mathcal{M}_1, \mathcal{M}_2\}$ with $\mathcal{M}_1 = \{(1, holds(approaches(b, p3), \langle 6, 10\rangle)), (2, holds(inBallControl(p3, b), \langle 9, 13\rangle))\}$ and $\mathcal{M}_2 = (1, \{holds(\ approaches(b,\ p5), \langle 21, 25\rangle)),\ (2, holds(\ inBallControl(p5, b), \langle 24, 28\rangle))\}$. The match intervals for $\mathcal{M}_1$ and $\mathcal{M}_2$ are $(9, 20]$ and $(24, 35]$, respectively. The support for this pattern is 22. As the maximal support of the dynamic scene is $seq_{length} + w = 22 + 10 = 32$, the frequency of the pattern is $freq(p) = \frac{22}{32} = 0.6875$.

4.4 Generalizations and Specializations of Patterns

Similar to Dehaspe [Deh98] we define a quasi-order "more general than" for the set of patterns. Referring to Dehaspe [Deh98, p. 80] for a given set $\mathcal{L}$ and a binary relation $\succeq$ on $\mathcal{L}$, $\langle \mathcal{L}, \succeq \rangle$ is a quasi-order if and only if $\succeq$ is reflexive and transitive:

$$\forall x \in \mathcal{L} : x \succeq x \qquad \text{(Reflexivity)}$$
$$\forall x, y, z \in \mathcal{L} : x \succeq y \text{ and } y \succeq z \Rightarrow x \succeq z \qquad \text{(Transitivity)}$$

Our pattern description consists of the basic pattern, the temporal restriction, and the concept restriction. All these parts must be taken account in the definition of the generalization (sub-pattern) relation:

Definition 4.37 (Generalization Relation)**.** Let $p_1 = (cp_1, \mathcal{TR}_1, \mathcal{CR}_1)$ and $p_2 = (cp_2, \mathcal{TR}_2, \mathcal{CR}_2)$ be two temporal patterns with conjunctive pattern sizes $s_1 = size(cp_1)$ and $s_2 = size(cp_2)$ and $cp_1 = ap_{1,1} \wedge \ldots \wedge ap_{1,s_1}$ and $cp_2 = ap_{2,1} \wedge \ldots \wedge ap_{2,s_2}$ p_1 *subsumes* p_2 if and only if $s_1 \leq s_2$ and there is an injective mapping $\mu : \{1, \ldots, s_1\} \mapsto \{1, \ldots, s_2\}$ such that $\forall i, j : i < j \Longrightarrow \mu(i) < \mu(j)$ (the order must be preserved).

Furthermore, $\forall i \in \{1, \ldots, s_1\} : pi_{1,i} = pi_{2,\mu(i)}$ with $ap_{1,i} = (pi_{1,i}, args_{1,i})$ and $ap_{2,\mu(i)} = (pi_{2,\mu(i)}, args_{2,\mu(i)})$. There must exist a substitution θ such that $(ap_{1,1}, \ldots, ap_{1,s_1})\theta = (ap_{2,\mu(1)}, \ldots, ap_{2,\mu(s_1)})$. It is also required that the temporal relation sets of p_1 are supersets of the mapped ones in p_2: $\forall i, j \in \{1, \ldots, s_1\} : \mathcal{TR}_{i,j} \supseteq \mathcal{TR}_{\mu(i),\mu(j)}$ with $i < j$, $(i, j, \mathcal{TR}_{i,j}) \in \mathcal{TR}_1$, and $(\mu(i), \mu(j), \mathcal{TR}_{\mu(i),\mu(j)}) \in \mathcal{TR}_2$.

Let $t_1 = arg(cp_1) = (t_{1,1}, \ldots, t_{1,l_1})$ and $t_2 = arg((ap_{2,\mu(1)}, \ldots, ap_{2,\mu(s_1)})) = (t'_{2,1}, \ldots, t'_{2,l_2})$ be the tuples of terms of the conjunctive patterns. Then $\forall i : (t_{1,i}, ci_{1,i}) \in \mathcal{CR}_1 \Longrightarrow (t_{2,i}, ci_{2,i}) \in \mathcal{CR}_2 \wedge (is\text{-}a(ci_{2,i}, ci_{1,i}) \vee ci_{2,i} = ci_{1,i})$.

□

In other words, a pattern p_1 subsumes another one p_2 if there exists an order-preserving mapping of the elements in the conjunctive pattern of the general pattern to the special pattern so that the conjunctive pattern subsumes the conjunction of the mapped elements. Additionally, the temporal restrictions for all predicate pairs of the general pattern must be supersets of the corresponding temporal restriction in the special pattern and the concept restrictions for each argument of the conjunctive pattern must subsume the corresponding concept in the concept restriction of the special pattern. A proper "more general than" relation $\succ$ exists iff:

$$p_1 \succ p_2 \text{ iff } p_1 \succeq p_2 \ \wedge \ p_2 \not\succeq p_1.$$

We also define direct, proper more general than relations and denote them by $\succ_1$:

$$tp_1 \succ_1 tp_2 \iff tp_1 \succ tp_2 \wedge \nexists tp_3 : tp_1 \succ tp_3 \wedge tp_3 \succ tp_2$$

The $\succ_1$-relation can be used for enumerating the pattern space. It allows for systematically searching for frequent patterns starting from the most general pattern ϵ.

The pattern mining algorithm has to perform a search through the pattern space. We define five different refinement operations in order to specialize a pattern: lengthening, temporal refinement, variable unification, concept refinement, and instantiation. If applied to a pattern p, each of these operations leads to a set of specialized patterns which are subsumed by p. These operations are similar to those defined by Lee [Lee06], but in our case predicates have a temporal extent and thus, interval relations are used for temporal restrictions and Lee's operations do not include the concept refinement as defined here. We follow Lee's notation w.r.t. the refinement operations [Lee06].

Definition 4.38 (Refinement Operator)**.** A *refinement operator* $\rho : \mathcal{L}_{tp} \rightarrow 2^{\mathcal{L}_{tp}}$ maps a pattern p of the pattern language to a set of patterns such that it holds $\rho(p) \subseteq \{p' \mid p \succ_1 p'\}$.

□

Definition 4.39 (Lengthening)**.** Let $tp = ((ap_1, \ldots, ap_m), \mathcal{TR}, \{(t_1, ci_1), \ldots, (t_n, ci_n)\})$ be a temporal pattern. The *lengthening* operator ρ_L adds an atomic pattern to the conjunctive pattern: $\rho_L(((ap_1, \ldots, ap_m), \mathcal{TR}, \{(t_1, ci_1), \ldots, (t_n, ci_n)\})) = \{((ap_1, \ldots, ap_{i-1}, ap', ap_i, \ldots, ap_m), \mathcal{TR}', \{(t_1, ci_1), \ldots, (t_n, ci_n), (V_1, ci'_1), \ldots, (V_{arity}, ci'_{arity})\})\}$ with $ap' = (pi', (V_1, \ldots, V_{arity}))$, $pt' = (pi', (ci'_1, \ldots, ci'_{arity}))$, and $pi' \in \mathcal{PI}$. Variables $V_1, \ldots, V_{arity}$ must not occur in any of the previously existing atomic patterns and must be mutually unequal. The new temporal restriction is defined as

$\mathcal{TR}' = \{(1, 2, \mathcal{TR}'_{1,2}), \ldots, (m, m+1, \mathcal{TR}'_{m,m+1})\}$ with:

$$\mathcal{TR}'_{k,l} = \begin{cases} \mathcal{TR}_{k,l} & \text{if } k, l < i \\ \mathcal{TR}_{k,l-1} & \text{if } k < i, l \geq i \\ \mathcal{TR}_{k-1,l-1} & \text{if } k, l \geq i \\ \mathcal{IR}_{older} & \text{if } (k = i \vee l = i) \wedge ap_{new,k} >_{lex} ap_{new,l} \\ \mathcal{IR}_{\leq} & \text{if } (k = i \vee l = i) \wedge ap_{new,k} \leq_{lex} ap_{new,l} \end{cases}$$

with $1 \leq k < (m+1)$ and $1 < l \leq (m+1), k < l$.

□

The new temporal restriction in the definition above keeps the temporal relations for the existing atomic pattern pairs and introduces new interval relation sets for the new atomic pattern in combination with all existing ones. Depending on the lexicographic order, the new interval relation set is set to $\mathcal{IR}_{older}$ or $\mathcal{IR}_{\leq}$.

Definition 4.40 (Temporal Refinement)**.** Let $p = (cp, \mathcal{TR}, \mathcal{CR})$ be a temporal pattern with $n = size(cp)$ atomic patterns. The set of *temporal refinements* for p is defined as $\rho_T = \{(cp, \mathcal{TR}', \mathcal{CR}) \mid \exists! i, j : (i, j, \mathcal{TR}'_{i,j}) \in \mathcal{TR}' \wedge |\mathcal{TR}'_{i,j}| + 1 = |\mathcal{TR}_{i,j}| \wedge \mathcal{TR}'_{i,j} \subset \mathcal{TR}_{i,j} \wedge \forall k, l : (k \neq i \vee l \neq j) \Longrightarrow (k, l, \mathcal{TR}_{k,l}) \in \mathcal{TR}'\}$ with $i < j$ and $k < l$.

□

Definition 4.41 (Variable Unification)**.** Let $tp = (cp, \mathcal{TR}, \mathcal{CR})$ be a temporal pattern and V_1, V_2 be variables occurring in cp. The *variable unification* operator unifies two variables V_1 and V_2 with $V_1 \neq V_2$: $\rho_U((cp, \mathcal{TR}, \mathcal{CR})) = \{(cp\theta, \mathcal{TR}, \mathcal{CR}') \mid \theta = \{V_1/V_2\}\}$ where V_1 and V_2 occur in cp.

In the concept restriction, the entry of V_1 must be removed and V_2 must be updated according to the previously defined concepts of V_1 and V_2: $\mathcal{CR}' = \mathcal{CR} \setminus \{(V_1, ci_1), (V_2, ci_1)\} \cup \{(V_2, ci'_2)\}$ with $ci'_2 = ci_2 \iff \textit{is-a}(ci_2, ci_1)$ and $ci'_2 = ci_1 \iff \textit{is-a}(ci_1, ci_2)$. The variable to be unified must be compatible: $\textit{is-a}(ci_1, ci_2) \vee \textit{is-a}(ci_2, ci_1)$.

□

Definition 4.42 (Concept Refinement)**.** Let $(cp, \mathcal{TR}, \{(t_1, ci_1), \ldots, (t_n, ci_n)\})$ be a temporal pattern. A *concept refinement* replaces one of the n concepts in the concept restriction by one of its direct sub-concepts: $\rho_C((cp, \mathcal{TR}, \{(t_1, ci_1), \ldots, (t_n, ci_n)\})) = \{(cp, \mathcal{TR}, \mathcal{CR}') \mid \mathcal{CR}' = \{(t_1, ci_1), \ldots, (t_{i-1}, c_{i-1}), (t_i, ci'_i), (t_{i+1}, ci_{i+1}), \ldots, (t_n, ci_n)\}, \textit{is-a}(ci'_i, ci_i)\}$ with $1 \leq i \leq n$.

□

It is possible to calculate the number of concept refinements for a term in a temporal pattern by computing the distance of the current concept to the most special concept of the predicate templates of the pattern:

Definition 4.43 (Concept Refinement Level)**.** Let $tp = ((ap_1, \ldots, ap_n), \mathcal{TR}, \mathcal{CR})$ be a temporal pattern with atomic patterns $ap_i = (pt_i, (t_{i,1}, \ldots, t_{i,arity}))$ and $pt_i = (pi_i, (ci_{i,1}, \ldots, ci_{i,arity})) \in PT$. The set of concept identifiers for a term t in the temporal pattern tp is then defined as $\mathcal{CI}_t = \{ci_{j,k} \mid t_{j,k} = t\}$. The most special concept is ci_{spec} if and only if $\forall ci \in \mathcal{CI}_t : \textit{is-a}(ci_{spec}, ci)$. As it is not allowed that two concepts mutually subsume each other (Def. 4.15), the distance between two concepts in the concept hierarchy is defined as: $dist(ci_1, ci_2) := |\{ci' \in CI \mid \textit{is-a}(ci_1, ci') \wedge ci' \neq ci_2\}|$.

The concept refinement level for a concept with identifier ci w.r.t. a temporal pattern tp is defined as: $crl(t, ci, tp) := dist(ci, ci_{spec})$.

□

Definition 4.44 (Instantiation)**.** Let $tp = (cp, \mathcal{TR}, \mathcal{CR})$ be a temporal pattern, V be a variable occurring in cp, and $o \in \mathcal{O}$ be an object occurring in the dynamic scene. The binding of a variable V to an object o is denoted as *instantiation*. Similar to a variable unification, the set of refined patterns is defined by a substitution: $\rho_I((cp, \mathcal{TR}, \mathcal{CR})) = \{(cp\ \theta, \mathcal{TR}, \mathcal{CR}\ \theta) \mid \theta = \{V/o\}\}$ where V must occur in cp and $o \in \mathcal{O}$.

□

Example 4.18. Let p be a temporal pattern $p = (approaches(A, B) \wedge inBallControl(C, D),\ \{(1, 2, \{\text{older, head-to-head}\})\}, \{(A, object), (B, object), (C, player), (D, ball)\})$. Then these are examples of valid refinements:

- Lengthening: $p_L = (approaches(A, B) \wedge inBallControl(C, D) \wedge inBallControl(E, F),\ \{(1, 2, \{\text{older, head-to-head}\}), (1, 3, \{\text{older, head-to-head}\}), (2, 3, \{\text{older, head-to-head}\})\}, \{(A, object), (B, object), (C, player), (D, ball), (E, player), (F, ball))\})$
- Temporal refinement: $p_T = (approaches(A, B) \wedge inBallControl(C, D), \{(1, 2, \{\text{older}\})\},\ \{(A, object), (B, object), (C, player), (D, ball)\})$
- Unification: $p_U = (approaches(A, B) \wedge inBallControl(B, D), \{(1, 2, \{\text{older, head-to-head}\})\}, \{(A, object), (B, player), (D, ball)\})$
- Concept Refinement: $p_C = (approaches(A, B) \wedge inBallControl(C, D), \{(1, 2, \{\text{older, head-to-head}\})\}, \{(A, ball), (B, object), (C, player), (D, ball)\})$
- Instantiation: $p_I = (approaches(A, B) \wedge inBallControl(C, b), \{(1, 2, \{\text{older, head-to-head}\})\}, \{(A, object), (B, object), (C, player), (b, ball)\})$

The combination of the five refinement operations form the refinement operator on the pattern language $\mathcal{L}_{tp}$: $\rho(p) = \rho_L(p) \cup \rho_T(p) \cup \rho_U(p) \cup \rho_C(p) \cup \rho_I(p)$.

One important property in pattern mining approaches is the anti-monotonicity w.r.t. the support of specialization operators. It is used, e.g., in Apriori in order

to prune the search space [AS94]: If an itemset is found to be not frequent, all its specializations cannot be frequent due to the anti-monotonicity property.

Theorem 4.3 (Anti-monotonicity of the refinement operator)**.** All refinements of a pattern have the same or a smaller support than the pattern itself, i.e., $\forall sp \in \rho(p) : supp(p) \geq supp(sp)$.

□

Proof 4.3. In order to prove the anti-monotonicity property, it must be shown that for each of the five refinement operations the support of the resulting patterns cannot increase in comparison to the support of the original pattern p. It is obvious that a pattern can only have a match where a generalization of the pattern has a match as for each atomic pattern in the conjunctive pattern we must find a matching predicate instance so that a valid substitution of the conjunctive pattern can be found and the temporal restriction and concept restriction are satisfied. There is only one operation which changes the size of the conjunctive pattern, namely the lengthening operation. In all other cases, the size of the conjunctive pattern does not change. We show for both cases that the support cannot increase.

1. Temporal refinement, variable unification, concept refinement, instantiation: In all these refinements, the size of the pattern does not change. A refined pattern p' can only have a match where p also has a match, i.e., $matches(p') \subseteq matches(p)$. For each match $m \in matches(p)$ there are only two possibilities: It either also satisfies the additional refinement, i.e., $m \in matches(p')$ or it does not, i.e., $m \notin matches(p')$. In particular, these cases are possible:

   (a) Temporal refinement: Let $\mathcal{TR}'_{i,j} \subset \mathcal{TR}_{i,j}$ be the refined temporal restriction with $\mathcal{TR}'_{i,j} = \mathcal{TR}_{i,j} \setminus tr$ and match $\mathcal{M} = \{(1, pred_1), \ldots, (n, pred_n)\}$ with $pred_i = (pi, o_1, \ldots, o_m, s_i, e_i, b)$. If $ir(\langle s_i, e_i\rangle, \langle s_j, e_j\rangle) = tr$ then $\mathcal{M} \notin matches(p')$, otherwise $\mathcal{M} \in matches(p')$.

   (b) Variable unification: Let V_1 and V_2 be the two variables in p before unification. For each match m of the pattern p a substitution θ with $\{V_1/o_1, V_2/o_2\} \subseteq \theta$ has been performed. After unification, only those matches of p where both variables are substituted by the same object can also be matches of p', i.e., if and only if $o_1 = o_2$ $m \in matches(p')$.

   (c) Concept refinement: Let $\mathcal{CR} = \{(o_1, ci_1), \ldots, (o_i, ci_i), \ldots, (o_n, ci_n)\}$ and $\mathcal{CR}' = \{(o_1, ci_1), \ldots, (o_i, ci'_i), \ldots, (o_n, ci_n)\}$ be the concept restrictions of p and p', respectively, with $is\text{-}a(ci'_i, ci_i)$. If the corresponding object o_i is still an instance of the sub-concept c'_i, i.e., $instance\text{-}of(o_i, c'_i)$, then $\mathcal{M} \in matches(p')$.

   (d) Instantiation: Let V be the variable in p that has been instantiated by o_1 and let o_2 be the instance that has been used in the substitution θ

with $\{V/o_2\} \subseteq \theta$ in order to match then $\mathcal{M} \in matches(p')$ if and only if $o_1 = o_2$.

2. Lengthening: As lengthening increases the size of the conjunctive pattern, the matches of p cannot be directly used as matches for p'. The reason is that p' has one more atomic pattern and there is no corresponding predicate in the existing matches of p. Nevertheless, p' can only have matches that *extend* a match $\mathcal{M} \in matches(p)$ as, of course, the conjunctive pattern without the added atomic pattern ap_{new} must also match. Thus, $matches(p)$ can be seen as the relevant set of sub-matches where matches of p' can occur. Ignoring the temporal restriction and concept restriction for simplicity a match $\mathcal{M}'$ can only be a valid match of p' if there is a match $\mathcal{M} \in matches(p)$ with $\forall i : (i, pred_i) \in \mathcal{M} \Longrightarrow \exists j : (j, pred_i) \in \mathcal{M}'$ with $i \leq j$ and there exists a predicate $(k, pred_{new}) \in \mathcal{M}'$ with $pred_{new} = (pi, o_1, \ldots, o_n, s, e, true)$ which occurs concurrently to the match interval of $\mathcal{M}$. It is required that $e \geq (s_{max} - w)$ and $s < e_{min} + w$.

   If a predicate satisfying these conditions exists, it must be shown that it cannot extend the validity interval of match $\mathcal{M}$. The validity interval of a match $\mathcal{M}$ is $v_i = (s_{max_i}, e_{min_i} + w]$ by definition. The following cases can occur:

   - $s \leq s_{max_i}$: Then $s'_{max_i} = s_{max_i}$, i.e., the lower bound of the validity interval does not change.
   - $s > s_{max_i}$: Then $s'_{max_i} > s_{max_i}$, i.e., the interval length decreases.
   - $e \geq e_{min_i}$: Then $e'_{min_i} = e_{min_i}$, i.e., the upper bound of the validity interval does not change.
   - $e < e_{min_i}$: Then $e'_{min_i} < e_{min_i}$, i.e., the interval length decreases.

Thus, it has been shown that the validity intervals of the matches of p' can only be within the validity intervals of the matches of p and it follows that $supp(p) \geq supp(p')$.

■

Fig. 4.10 illustrates an existing match (A and B) and three cases of an additional predicate C. C_1 and C_3 are not relevant as they do not occur concurrently to the previous match interval. C_2 illustrates the only situation where the new interval lies within the previous match interval. As shown in the proof above, it does not matter if C_2 starts before or ends after the match interval.

From Def. 4.36 it follows directly that $\forall sp \in \rho(p) : freq(p) \geq freq(sp)$. It also holds that any pattern that can be generated by more than one application of

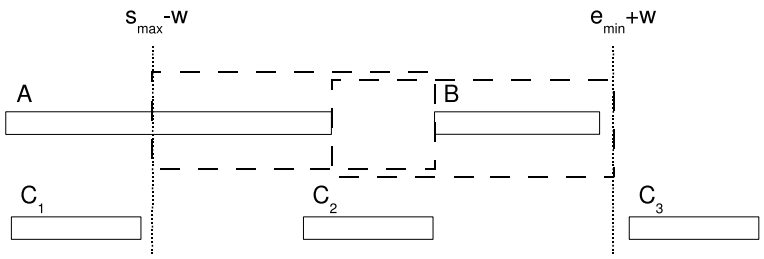

Figure 4.10: Different cases for a match applying the lengthening operation

the refinement operator must have a frequency equal to or less than the original pattern's frequency:

$$p \preceq q \Longrightarrow freq(p) \leq freq(q)$$

4.5 Pattern Mining

Similar to the definition of large itemsets in association rule mining (see Section 3.1.2 and [AIS93]), we define patterns to be *frequent* if their frequency exceeds a threshold $minfreq > 0$. The support of a pattern in terms of association rule mining is the number of transactions that support the pattern. In our case, the support is computed by the cumulated length of intervals where at least one match of the pattern can be found in the dynamic scene.

Temporal pattern mining denotes the process of generating temporal patterns from dynamic scene descriptions which satisfy certain criteria. Besides constraints about the size of the patterns to mine, the discovered patterns should appear frequently in the dynamic scene.

Definition 4.45 (Temporal Pattern Mining)**.** The process of temporal pattern mining identifies all temporal patterns for a dynamic scene ds which exceed a given frequency threshold $minfreq$, i.e., $\mathcal{L}_{tp} = \{tp_i \mid freq(tp_i) \geq minfreq\}$.

□

4.5.1 Optimal Refinement Operator

The pattern mining algorithm performs a general-to-specific search through the pattern space. It starts with the empty, most general pattern ϵ and applies the refinement operator successively in order to generate frequent pattern candidates. We are looking for an optimal refinement operator, i.e., a refinement operator that can be used to find all frequent patterns and creates each pattern once at most. An optimal refinement operator is one that satisfies completeness and non-redundancy

properties [Lee06]. Following Lee's notation we define completeness and redundancy [Lee06, p. 93]:

Definition 4.46 (Completeness)**.** A refinement operator ρ is *complete* w.r.t. language $\mathcal{L}_{tp}$ if and only if $\rho^*(\epsilon) = \mathcal{L}_{tp}$ where ρ^* denotes the transitive closure of ρ.

□

Definition 4.47 (Non-redundancy)**.** A refinement operator ρ is *non-redundant* w.r.t. a language $\mathcal{L}_{tp}$ if and only if for all patterns $p \in \mathcal{L}_{tp}$ there exists at most one single path $\epsilon = p_0, p_1, \ldots, p_n = p$ such that $\forall i : p_{i+1} \in \rho(p_i)$ with $0 \leq i \leq n$.

□

If there exists exactly one single path for each pattern p, the refinement operator is optimal, i.e., non-redundant and complete.

Theorem 4.4 (Non-optimality of refinement operator ρ)**.** The refinement operator ρ is not optimal.

□

Proof 4.4. The proof of non-optimality of ρ is done by contradiction. Let us assume that ρ was optimal. Then it must be complete and non-redundant. Non-redundancy means that there is only one path from the most general empty pattern to all patterns in $\mathcal{L}_{tp}$. It is easy to find a counter-example where two different paths can be found to a pattern p. If the dynamic scene schema consists only of one predicate template $pass(object, object)$, then the pattern $p = (pass(p_1, p_1), \mathcal{TR}, \{(p_1,\ object)\})$ can be obtained by these two paths ($p \xrightarrow{\rho_X} q$ denotes that q can be created by applying ρ_X to p):

$$\begin{aligned}
\epsilon &\xrightarrow{\rho_L} (pass(V_1, V_2), \mathcal{TR}, \{(V_1, object), (V_2, object)\}) \\
&\xrightarrow{\rho_U} (pass(V_1, V_1), \mathcal{TR}, \{(V_1, object)\}) \\
&\xrightarrow{\rho_I} (pass(p_1, p_1), \mathcal{TR}, \{(p_1, object)\})
\end{aligned}$$

$$\begin{aligned}
\epsilon &\xrightarrow{\rho_L} (pass(V_1, V_2), \mathcal{TR}, \{(V_1, object), (V_2, object)\}) \\
&\xrightarrow{\rho_I} (pass(p_1, V_2), \mathcal{TR}, \{(p_1, object), (V_2, object)\}) \\
&\xrightarrow{\rho_I} (pass(p_1, p_1), \mathcal{TR}, \{(p_1, object)\})
\end{aligned}$$

The temporal restrictions have been omitted for clarity. The existence of more than one path to a pattern $p \in \mathcal{L}_{tp}$ is a contradiction to the non-redundancy property and thus to the optimality property of ρ.

■

As it has been shown that the refinement operator ρ is not optimal, now an adaption is presented that leads to non-redundancy. An order between the different refinement operations and an order within the refinement operations is set up in order to create each pattern only once. Lee [Lee06] has introduced a *refinement level vector* which can be used to track the process of refinement. In this work, an adapted version of such a refinement level vector is introduced and used to control the mining process. As it has been shown by Lee [Lee06], this vector is also helpful to prove the optimality of a refinement operator.

In order to reduce complexity, we constrain the space for temporal restrictions. In the previous definition of the pattern language, it is possible to mine patterns where the temporal restriction has more than one element for atomic pattern pairs, i.e., it was possible to have $|\mathcal{TR}_{i,j}| > 1$. A temporal pattern with n atomic patterns and $k = |\mathcal{IR}_{\leq}|$ temporal relations would lead to $\frac{n\cdot(n-1)}{2} \cdot (2^k - 1)$ possible combinations (if no composition table was used) as there are $\frac{n\cdot(n-1)}{2}$ predicate pairs and for each predicate pair there are $(2^k - 1)$ possible sets of temporal relations (the power set of the number of temporal relations minus one as the empty set is not allowed in temporal patterns). A pattern with five predicates ($n = 5$) and seven temporal relations ($k = 7$ without inverses) as it is the case with Allen's interval relations would already lead to $\frac{5\cdot(5-1)}{2} \cdot (2^7 - 1) = 1270$ combinations (for each pattern). In the constrained variant, only those patterns are relevant where each temporal constraint for a predicate pair consists of one temporal relation, i.e., $|\mathcal{TR}_{i,j}| = 1$. In this case, the number of possible combinations is reduced to $\frac{n\cdot(n-1)}{2} \cdot k$. With $n = 5$ and $k = 7$ there are only $\frac{5\cdot(5-1)}{2} \cdot 7 = 70$ possible combinations.

The first step towards a non-redundant refinement operator is to assure that each refinement operation itself is non-redundant. Non-redundancy w.r.t. a refinement operations means that applying only this single operation multiple times cannot lead to redundant patterns. Analogously to Lee, we restrict each refinement operation:

Definition 4.48 (Non-redundant Refinement Operations)**.** The following restrictions specify *non-redundant* variants of the refinement operations defined in Def. 4.39 - 4.44:

- Lengthening ($\dot{\rho}_L$): It is only allowed to add atomic patterns to the end of a conjunctive pattern: $\dot{\rho}_L((ap_1, \ldots, ap_m), \mathcal{TR}, \{(t_1, ci_1), \ldots, (t_n, ci_n)\}) = \{((ap_1, \ldots, ap_m, ap_{m+1}), \mathcal{TR}', \{(t_1, ci_1), \ldots, (t_n, ci_n), (V_1, ci'_1), \ldots, (V_{arity}, ci'_{arity})\})\}$ with $ap' = (pi', (V_1, \ldots, V_{arity}))$, $pt' = (pi', (ci'_1, \ldots, ci'_{arity}))$, and $pi' \in \mathcal{PI}$. Further restrictions w.r.t. uniqueness of variables and the temporal restriction are still required as defined in Def. 4.39. Let $args((ap_1, \ldots, ap_m)) = (t_1, \ldots, t_k)$. It is required that all new variables are lexicographically greater than all existing variables and that the new variables among themselves also keep the order: $\forall i, j : t_j <_{lex} V_i$ with $1 \leq i \leq arity$ and $1 \leq j \leq m$ as well as $\forall i, j : i < j \Longrightarrow V_i <_{lex} V_j$.

- Temporal restriction ($\dot{\rho}_T$): It is only allowed to restrict the first occurring temporal restriction with more than one element, i.e., $|\mathcal{TR}_{i,j}| > 1$ with $(i, j, \mathcal{TR}_{i,j}) \in \mathcal{TR}$ such that $\forall k, l : pair\_to\_index(k, l) < pair\_to\_index(i, j) \Longrightarrow |\mathcal{TR}_{k,l}| = 1$ and $(k, l, \mathcal{TR}_{k,l}) \in \mathcal{TR}$. Furthermore, in the refinement step the new restriction must only consist of one element of the previous set, i.e., $\mathcal{TR}'_{i,j} = \{tr\}, tr \in \mathcal{TR}_{i,j}$.

- Unification ($\dot{\rho}_U$): It is only allowed to unify one of the variables to the right of the last multiple occurrence of a variable (i.e., after the previous unification's position) and it can only be unified with one of the previously occurring variables in the conjunctive pattern. Let $arg(cp) = (t_1, \ldots, t_i, \ldots, t_n)$ with $1 \leq i \leq n$. A variable $t_i \in \mathcal{V}$ can be unified if and only if $\nexists j, k : j \leq i \wedge k > i \wedge t_k = t_j$.

- Concept refinement ($\dot{\rho}_C$): It is only allowed to specialize those concepts that do not have a refined concept to their "right", i.e., after a certain concept position has been refined the concepts to the left cannot be changed anymore. Let $\{t_1, \ldots, t_n\}$ be the set of terms in the conjunctive pattern of the temporal pattern to be refined. It is only allowed to refine the concept of those terms t_i where $\nexists t_j : t_j >_{lex} t_i \wedge crl(t_j, tp) > 0$ (see Def. 4.43 on page 108 for the definition of the concept refinement level crl). The lexicographic order on the variables ensures that variable identifiers which appear later in the conjunctive pattern are lexicographically greater than the variable identifiers before. (Due to the order of the refinement operators all terms in the conjunctive pattern must be variables; after an instantiation the concept refinement is not used anymore as we will see below; cf. Def. 4.50.)

- Instantiation ($\dot{\rho}_I$): It is only allowed to instantiate a variable if there has not been any instantiation to the "right" of this variable (excluding those instantiations that also have an occurrence of the same instance before the position of the variable to instantiate). Let $arg(cp) = (t_1, \ldots, t_n)$ be the n-tuple of terms of the conjunctive pattern. An instantiation for term t_i can be performed only if $\forall j > i : t_j \in \mathcal{O} \Longrightarrow \exists k < i : t_k = t_j$.

  It is also required that the object o is a direct instance of the corresponding concept identifier ci with $(t_i, ci) \in \mathcal{CR}$, i.e., *direct-instance-of(o, ci)*. Additionally, a variable can only be instantiated to an object that does not occur in the conjunctive pattern so far, i.e., it is required that $\forall i : t_i \neq o$ with $1 \leq i \leq n$.

□

These restrictions avoid redundant paths w.r.t. each single refinement operation as they could happen before. In the previous definitions, there was no restriction in which order the refinements are performed, i.e., it was possible, for instance, to first unify variable V_1 and then V_2 or the other way round by applying ρ_U creating

the same pattern. Similar redundant paths to identical patterns have been possible with the other previous refinement operations, too.

Following Lee [Lee06] we also introduce a refinement level vector:

Definition 4.49 (Refinement Level Vector)**.** Let $tp = (cp, \mathcal{TR}, \mathcal{CR})$ be a temporal pattern. The *refinement level vector* of tp is denoted as $\vec{v}(tp) = (l, t, u, c, i)$ where

- $l = size(cp)$ is the number of atomic patterns in tp,
- t is the number of consecutive totally restricted temporal restrictions, i.e., where $|\mathcal{TR}_{i,j}| = 1$ starting at the $\mathcal{TR}_{1,2}$, i.e., $\min(\{k \mid \forall l \leq k : |\mathcal{TR}_l| = 1\})$,
- u is the number of terms (i.e., variables or objects) in tp, i.e., m with $arg(cp) = (t_1, \ldots, t_m)$ minus the number of distinct terms in tp: $n = |\{t_i \mid 1 \leq i \leq m\}|$, i.e., $u = m - n$,
- c is the sum of path distances of all distinct terms' concepts to the corresponding most general basic concept of the term. Let $\{t_1, \ldots, t_n\}$ be the set of distinct terms in tp, and let $\{ci_1, \ldots, ci_n\}$ be the corresponding concept identifiers with $(t_i, ci_i) \in \mathcal{CR}$. The concept refinement level (see Def. 4.43) can be used to compute the sum of all path distances: $c = \sum_{i=1}^{n} crl(t_i, ci_i, tp)$, and
- i is the number of distinct instances in tp. Let $arg(cp) = (t_1, \ldots, t_n)$ be the n-tuple of terms in the conjunctive pattern. The number of distinct instances is defined as: $i = |\{t_i \mid 1 \leq i \leq n \wedge t_i \in \mathcal{O}\}|$.

□

Each position in the refinement level vector $\vec{v}(tp)$ thus indicates how often the different refinement operations have been applied in order to create temporal pattern tp. The empty pattern ϵ does not have any atomic patterns, temporal restrictions, variables, concept restrictions, or instances. The refinement level vector of ϵ is defined as $\vec{v}(\epsilon) = (0, 0, 0, 0, 0)$.

Example 4.19. The temporal pattern $tp = (uncovered(X), pass(Y, X), \{(1, 2, \{\text{older \& contemporary}\})\}, \{(X, object), (Y, object)\})$ has the refinement level vector $\vec{v}(p) = (2, 1, 1, 0, 0)$ as there has been two lengthening operations, one temporal refinement, one unification, and zero concept refinements and instantiations.

The refinement level vector can now be used to force an order on the refinement operations in the united refinement operator (cf. [Lee06]):

Definition 4.50 (Optimal Refinement Operator)**.** The optimal refinement operator

is defined as:

$$\dot{\rho}(p) = \text{set-union of } \begin{cases} \dot{\rho}_L(p) & \text{if } \vec{v}(p) = (l,0,0,0,0) \\ \dot{\rho}_T(p) & \text{if } \vec{v}(p) = (l,t,0,0,0) \\ \dot{\rho}_U(p) & \text{if } \vec{v}(p) = (l,t,u,0,0) \wedge t = \frac{l\cdot(l-1)}{2} \\ \dot{\rho}_C(p) & \text{if } \vec{v}(p) = (l,t,u,c,0) \wedge t = \frac{l\cdot(l-1)}{2} \\ \dot{\rho}_I(p) & \text{if } \vec{v}(p) = (l,t,u,c,i) \wedge t = \frac{l\cdot(l-1)}{2} \end{cases}$$

□

The additional condition that $t = \frac{l\cdot(l-1)}{2}$ forces that unifications, concept refinements, and instantiations can only be performed after all temporal relations between the atomic patterns have been restricted.

In the same way, it is possible to define the inverse refinement operator:

Definition 4.51 (Inverse Optimal Refinement Operator)**.** The *inverse optimal refinement operator* $\dot{\rho}^{-1}$ is defined as:

$$\dot{\rho}^{-1}(p) = \begin{cases} \dot{\rho}_I^{-1}(p) & \text{if } i > 0 \\ \dot{\rho}_C^{-1}(p) & \text{if } i = 0 \wedge c > 0 \\ \dot{\rho}_U^{-1}(p) & \text{if } i = 0 \wedge c = 0 \wedge u > 0 \\ \dot{\rho}_T^{-1}(p) & \text{if } i = 0 \wedge c = 0 \wedge u = 0 \wedge t > 0 \\ \dot{\rho}_L^{-1}(p) & \text{if } i = 0 \wedge c = 0 \wedge u = 0 \wedge t = 0 \wedge l > 0 \end{cases}$$

□

Theorem 4.5 (Operation and refinement level vector correspondence)**.** The values in the refinement level vector $\vec{v}(p) = (l,t,u,c,i)$ correspond to the number of applications of the refinement operation $\dot{\rho}_L, \dot{\rho}_T, \dot{\rho}_U, \dot{\rho}_C$, and $\dot{\rho}_I$ needed to create p from the empty pattern ϵ.

□

Proof 4.5. For each of the values in the refinement level vector $\vec{v}(p) = (l,t,u,c,i)$ it can be shown that it corresponds to the number of refinements (cf. [Lee06]):

- $\dot{\rho}_L$ is the only operation that changes the length of the conjunctive pattern. For ϵ it holds that $l = 0$ as $\vec{v}(\epsilon) = (0,0,0,0,0)$ and each application of $\dot{\rho}_L$ adds one atomic pattern to the temporal pattern it holds that the number of applications of $\dot{\rho}_L$ corresponds to the length of the conjunctive pattern, namely l.

- The number of restricted temporal relations between predicates can only be increased by $\dot{\rho}_T$. As it is only allowed to restrict the first $\mathcal{TR}_{i,j}$ with $|\mathcal{TR}_{i,j}| > 1$, the number of consecutive restricted temporal relations corresponds to the number of applications of $\dot{\rho}_T$.

- The number of terms in a conjunctive pattern changes with different refinement operations, but u is defined as the number of terms minus the number of distinct terms. Thus, as lengthening ($\dot{\rho}_L$) just introduces new, distinct variables u does not change. Temporal refinement and concept refinement ($\dot{\rho}_T$ and $\dot{\rho}_C$) do not change the number of terms or the terms themselves and thus do not affect u. The instantiation operation does not change the number of terms but just replaces a variable by an object. As all occurrences of the variable are replaced, the number of distinct terms stays the same and u does not change. The only operation that changes u is the unification ($\dot{\rho}_U$). If two variables are unified, the number of distinct terms change. As it is only possible to unify distinct variables (due to the order in $\dot{\rho}_U$), the number of distinct terms is reduced by exactly one and thus, u increases by one each time the operator is applied. It follows that u corresponds to the number of unification operations.

- The concept refinement ($\dot{\rho}_C$) is the only operation that changes the concept restriction. As the application of $\dot{\rho}_C$ replaces one concept of a variable by one of its direct sub-concepts, c is increased by one at each application of $\dot{\rho}_C$.

- As the instantiation operator ($\dot{\rho}_I$) is the only one adding instances to the conjunctive pattern and it is only allowed to add an instance that does not occur in the previous term, the number of distinct instances is increased by one at each application of this refinement. Thus, it follows that the number of distinct instances corresponds to the number of applications of $\dot{\rho}_I$.

■

Each refinement operation increments only one of the counters in $\vec{v}(p) = (l, t, u, c, i)$. Therefore, the sum $l + t + u + c + i$ of the single counters of the refinement level vector of a pattern p is equal to the number of refinement operations performed from ϵ in order to form p. As abbreviation we write $|\vec{v}(p)| = l + t + u + c + i$. Analogously, the inverse refinement operator decrements the corresponding counter of a refinement operation by one as just one refinement operation is "taken back".

In the next section, we present how knowledge about temporal relations and about concepts and predicate templates can be used to constrain the search space. If this knowledge is applied, not all statements in the proof above hold anymore because some intermediate patterns are skipped. However, as it will be shown later these changes do not interfere with the optimality property of the refinement operator (Proof 4.7).

Definition 4.52 (Pattern Space of Frequent Patterns)**.** The changed pattern space consists only of frequent patterns and those which are completely constrained w.r.t. the temporal restriction. The intermediate patterns (before the temporal restriction is completely constrained) must also be legal members of the pattern space: $\mathcal{L}_{tp} =$

$\{p \mid freq(p) \geq minfreq \wedge \vec{v}(p) = (l,t,u,c,i) \wedge (t = \frac{l\cdot(l-1)}{2} \vee u = c = i = 0)\}$.

□

The condition $(t = \frac{l\cdot(l-1)}{2} \vee u = c = i = 0)$ assures that the temporal restriction is either completely constrained or that none of the later refinement steps (unification, concept refinement, instantiation) has been applied so far.

Theorem 4.6 (Optimality of the refinement operator $\dot{\rho}$)**.** The refinement operator $\dot{\rho}$ is optimal for $\mathcal{L}_{\dot{tp}}$.

□

Proof 4.6. In order to prove optimality for $\dot{\rho}$, it must be shown that $\dot{\rho}$ is complete and non-redundant. The refinement level vector $\vec{v}$ as well as the inverse refinement operator $\dot{\rho}^{-1}$ can be used for the proof and the proof schema of Lee [Lee06, p. 98 - 99] can be adapted:

1. Completeness

   Let $p \in \mathcal{L}_{\dot{tp}}$ be an arbitrary temporal pattern with $p \neq \epsilon$. The number of refinement steps performed in order to create p from ϵ is $k = |\vec{v}(p)|$ with $k \neq 0$. For each pattern p there is only one inverse operation possible in $\dot{\rho}^{-1}$ as the inverse refinement operations are only applicable mutually disjoint due to the conditions (Def. 4.51). Furthermore, for each inverse refinement operation the result is a single pattern by definition. Each application of the inverse refinement operator decrements the sum of the refinement level vector by one. After k applications of $\dot{\rho}^{-1}$ to p we have a single pattern p' with a refinement level of zero and thus $\vec{v}(p') = (0,0,0,0,0)$. The only element of $\mathcal{L}_{\dot{tp}}$ with this property is ϵ, i.e., $p' = \epsilon$. Thus, it has been shown that for each pattern p there exists a path of inverse refinement operations to the most general pattern and it follows that by application of the corresponding original (non-inverse) refinement operations p can be generated from ϵ, i.e., $\forall p \in \mathcal{L}_{\dot{tp}} \setminus \epsilon : p \in \dot{\rho}^*(\epsilon)$.

2. Non-redundancy

   Non-redundancy can be shown in a similar way by contradiction. Let us assume that $\dot{\rho}$ is not non-redundant, i.e., there is a pattern $p \in \mathcal{L}_{\dot{tp}}$ where two different paths from ϵ to p exist, i.e., there are two paths $\epsilon = p_0, p_1, \ldots, p_m = p$ and $\epsilon = q_0, q_1, \ldots, q_n = p$ with $p_{i+1} \in \dot{\rho}(p_i)$ and $q_{i+1} \in \dot{\rho}(q_i)$. As $p_m = q_n = p$, it holds that $\vec{v}(p_m) = \vec{v}(q_n)$. Therefore, the number of refinement steps is $k = |\vec{v}(p_m)| = |\vec{v}(q_n)|$ and it follows that $m = n$. If we apply the inverse refinement operator k times to p_m and q_m, we get the paths $p = p'_m, p'_{m-1}, \ldots, p'_0 = \epsilon$ and $p = q_m, q'_{m-1}, \ldots, q'_0 = \epsilon$ with $p'_i = \dot{\rho}^{-1}(p'_{i+1})$ and $q'_i = \dot{\rho}^{-1}(q'_{i+1})$. As the inverse refinement operator only creates a single pattern, it must hold that $p'_i = q'_i$ and therefore $p_i = q_i$ with $0 \leq i \leq m$. This is a contradiction to the assumption that there exist two distinct paths from ϵ to p.

■

4.5.2 Application of Knowledge

In the previous sections, temporal patterns and a quasi-order on the pattern space have been defined and it has been shown how the pattern space can be searched efficiently by an optimal refinement operator. In this section, it is shown how the set of relevant patterns can be mined by leaving out those that cannot be frequent due to the anti-monotonicity property of the support and by applying knowledge about temporal intervals and concepts from the schema definition. As it will be shown, this knowledge can be used to reduce the number of patterns to be checked or even generated in the pattern mining process.

The goal of the mining process is not to enumerate all possible patterns but to identify all *interesting* patterns. While many different interpretations of *interestingness* are possible, we focus on a typical measure, namely the *support* or *frequency* as mentioned above. The goal of temporal pattern mining is to find all patterns $p \in \mathcal{L}_{tp}$ which have a frequency above the frequency threshold, i.e., the relevant pattern set to be mined is defined as $\{p \mid p \in \mathcal{L}_{tp} \wedge freq(p) \geq minfreq\}$.

Here, the set of patterns to be mined should be even further restricted. Although we have defined a generalization relation (Def. 4.37), it is possible to create patterns p_1 and p_2 with $p_1 \prec p_2$ and $freq(p_1) = freq(p_2)$ for all possible dynamic scenes compliant with a given dynamic scene schema dss. The following examples illustrate this fact.

Example 4.20. Let $p_1 = (pass(p1, p2), \mathcal{TR}, \{(p1, player), (p2, player)\})$ and $p_2 = (pass(p1, p2), \mathcal{TR}, \{(p1, object), (p2, object)\})$ with *direct-instance-of*$(p1, player)$ and *direct-instance-of*$(p2,\ player)$. From the generalization definition above it follows that $p_1 \prec p_2$. However, as the two arguments of $pass$ are already instantiated to $p1$ and $p2$ the concept restriction $(p1, object), (p2, object)$ is without effect and it holds that $matches(p_1) = matches(p_2)$ and $freq(p_1) = freq(p_2)$.

Example 4.21. Let $p_1 = (closerToGoal(X, Y), uncovered(X), pass(Y, X), \{(1, 2,$ $\{before\}),\ (2, 3, \{before\}), (1, 3, \{before\})\}, \mathcal{CR})$ and $p_2 = (closerToGoal(X, Y),$ $uncovered(X), pass(Y,\ X), \{((1, 2, \{before\}), (2, 3, \{before\}), (1, 3, \{$before, older & contemporary, head-to-head$\})\},\ \mathcal{CR})$. From the generalization definition above it follows that $p_1 \prec p_2$, but the composition table (Table 4.3 on page 96) shows that if $\mathcal{TR}_{1,2} = \{before\}$ and $\mathcal{TR}_{2,3} = \{before\}$ it must hold that $\mathcal{TR}_{1,3} = \{before\}$, i.e., the other two options older & contemporary and head-to-head are irrelevant and thus, it holds that $matches(p_1) = matches(p_2)$ and $freq(p_1) = freq(p_2)$.

In more extreme cases, it would even be possible to create an inconsistent pattern that cannot have any match at all, for instance, because the combination of temporal restrictions is not possible. Therefore, we introduce additional concepts to our formalism to handle these aspects.

Definition 4.53 (Most Special Equivalent Pattern)**.** Let $p = (cp, \mathcal{TR}, \mathcal{CR})$ be a temporal pattern. The function $spec : \mathcal{L}_{tp} \rightarrow \mathcal{L}_{tp}$ maps a pattern p to its *most special*

equivalent pattern $p_{spec} = (cp, \mathcal{TR}', \mathcal{CR}')$. The (potentially) specialized temporal restriction is defined as $\mathcal{TR}' = prop_{spec}(\mathcal{TR})$ (see Def. 4.32 on page 96).

The specialized concept restriction takes for each term t_i in the conjunctive pattern the most specialized concept of the corresponding predicate templates, previous concept restriction, or direct concept of the instance. Let $t_1, \ldots, t_n$ be the list of terms in the conjunctive pattern and $\mathcal{C}_m = \{ci_{m,1}, \ldots, ci_{m,k}\} \cup ci_{m,prev}$ be the set of concept identifiers of the corresponding positions in the predicate templates with $\mathcal{CR} = \{(t_1, ci_{1,prev}), \ldots, (t_n, ci_{n,prev})\}$. If t_m is atomic, i.e., $t_m \in \mathcal{O}$, the corresponding most specialized concept is $ci_{m,spec}$ with *direct-instance-of*$(t_m, ci_{m,spec})$. Otherwise, $ci_{m,spec}$ is the most special concept of $\mathcal{C}_m$, i.e., $ci_{m,spec} \in \mathcal{C}_m$ and $\forall ci_m \in \mathcal{C}_m : ci_m \neq ci_{m,spec} \Longrightarrow$ *is-a*$(ci_m, ci_{m,spec})$. $\mathcal{CR}'$ is defined as $\mathcal{CR}' = \{(t_1, ci_{1,spec}), \ldots, (t_n, ci_{n,spec})\}$.

If a pattern is inconsistent, the function maps to the *inconsistent pattern* $\perp$ (cf. Def. 4.27).

□

Definition 4.54 (Temporal Pattern Equivalence Relation)**.** Two patterns p_1 and p_2 are *equivalent* denoted by $p_1 \sim p_2$ iff their most special equivalent patterns are identical: $spec(p_1) = spec(p_2)$.

The *equivalence class* of a pattern p is defined as the set of all patterns that are equivalent to p: $[p] = \{q \mid q \in \mathcal{L}_{tp} \wedge p \sim q\} \subseteq \mathcal{L}_{tp}$.

□

In order to avoid checking equivalent patterns multiple times and to avoid checking inconsistent patterns (which cannot have any match and thus must have a support and frequency of zero), the available knowledge should be used to create the most special pattern of a pattern's equivalence class. In order to create this most special pattern, the temporal restriction and the concept restriction must be examined. For the temporal restriction, the composition table must be used to remove all temporal relations that are not possible anymore. This can be done by constraint propagation as shown in Section 4.2.

For the concept restriction, the most restricted concept must be identified for each position taking into account the predicate templates and instantiation. A pattern is inconsistent if during this process an empty temporal restriction was created for a pattern pair or if the set of concepts for some argument of the conjunctive pattern cannot be mutually subsumed (i.e., if there is a contradiction that a variable should be instance of two different concepts which do not lie on the same path in the concept hierarchy).

With these new definitions it is possible to re-define the temporal pattern mining task: The goal is to find the set of all most special equivalent patterns of frequent patterns: $\mathcal{P}_{freq} = \{p \mid p' = spec(p) \wedge p \in \mathcal{L}_{tp} \wedge freq(p) = freq(p') \geq minfreq\} \cup \epsilon$.

The mining process can be defined as:

$$\begin{aligned}
\mathcal{P}_{freq} &= \mathcal{P}_0 \cup \ldots \cup \mathcal{P}_\infty \\
\mathcal{P}_0 &= \{\epsilon\} \\
\mathcal{C}_{i+1} &= \{p \mid p \in \dot{\rho}(p') \wedge p' \in \mathcal{P}_i\} \\
\mathcal{P}_{i+1} &= \{p \mid spec(p) = p' \wedge p' \in \mathcal{C}_1 \cup \ldots \cup \mathcal{C}_{i+1} \wedge freq(p) \geq minfreq \\
&\quad \wedge |\vec{v}(p)| = (i+1)\}
\end{aligned}$$

Theorem 4.7 (Optimality of the equivalence-based mining operator)**.** The refinement operator $\dot{\rho}$ with the minimal frequency condition and the maximal specifity condition is optimal for $\mathcal{P}_{freq}$.

□

Proof 4.7. The proof must show that all changes comply with the completeness and optimality properties. The new pattern space is restricted to those patterns that are frequent and maximally specialized within their equivalence class.

1. Completeness

   As only those patterns are taken into account whose frequency is above the $minfreq$ threshold for generating patterns of the next refinement level, it is clear that less patterns are generated by this additional condition. At the same time it is only required to find those patterns that exceed the $minfreq$ threshold. Due to the anti-monotonicity property of the frequency (Def. 4.3) for any given pattern p with $freq(p) < minfreq$ none of the patterns subsumed by p can be relevant for the mining process as it holds that $p' \preceq p \Longrightarrow freq(p') \leq freq(p)$ and thus $freq(p') < minfreq$. It follows that the minimal frequency condition does not interfere with completeness.

   In order to show that no relevant pattern is missed after applying the specialization function, the different refinements must be examined:

   - Lengthening: Specialization has no effect, i.e., $\forall p' : p' \in \dot{\rho}_L(p) \Longrightarrow p' = spec(p')$, i.e., no pattern is left out.
   - Temporal refinement: After restricting the set of possible temporal relations between a predicate pair, it is possible that the temporal constraint propagation procedure further specializes the pattern. The following cases have to be distinguished:
     - Case 1: Nothing is specialized, i.e., $p = spec(p)$.
     - Case 2: Any of the temporal relation sets is empty after specialization, i.e., $\exists(i, j, \emptyset) \in \mathcal{TR}$ In this case, the pattern is inconsistent and has got a frequency of zero and thus is not relevant for the mining process as it is defined that $minfreq > 0$.

  - Case 3: $n > 0$ of the directly succeeding temporal relation sets $\mathcal{TR}_{m+1}, \ldots, \mathcal{TR}_{m+n}$ of the recently restricted temporal relation $\mathcal{TR}_m$ have been restricted such that they all only consist of one element, i.e. $\forall k : |\mathcal{TR}_k| = 1$ with $1 \leq k \leq (m+n)$. This can be seen as n temporal refinement operations without taking into account the remaining options that have been available in the temporal relation sets before. However, all these pruned options would have led to the inconsistent pattern anyway. This specialization only keeps the relevant combination but skips the intermediate refinement operations.
  - Case 4: If the directly succeeding temporal relation set is not completely restricted, i.e., $|\mathcal{TR}_{m+1}| > 1$, then the refinement level of the pattern does not change. All temporal relations that have been removed by the constraint propagation algorithm would have led to an inconsistent pattern, i.e., the search space (of upcoming refinement steps) is only pruned at the irrelevant parts of the search space.

- Unification: The unification of two variables can lead to a specialization in the concept restriction. Let c_1 and c_2 be the corresponding concepts in the concept restriction of the unified variables. The following cases can be distinguished:
  - Case 1: Nothing is specialized, i.e., $p = spec(p)$.
  - Case 2: Neither $is\text{-}a(c_1, c_2)$ nor $is\text{-}a(c_2, c_1)$. In this case, the pattern is inconsistent as it is not possible that an object is instance of two different concepts without common path in the concept hierarchy. This pattern and the ones subsumed by it are infrequent and thus irrelevant for the mining task.
  - Case 3: One concept subsumes the other one, i.e., $is\text{-}a(c_g, c_s)$ with $c_g = c_1 \wedge c_s = c_2$ or $c_g = c_2 \wedge c_s = c_1$. This prunes all patterns where the more general concept would have been specialized to any c' with $is\text{-}a(c', c_g) \wedge \neg is\text{-}a(c_s, c')$. The set of these concepts $\mathcal{C}_{intermediate} = \{c' \mid is\text{-}a(c', c_g) \wedge \neg is\text{-}a(c_s, c')\}$ can be divided into those that lie on the path from c_g to c_s, i.e., $\mathcal{C}_{path} = \{c_p \mid c_p \in \mathcal{C}_{intermediate} \wedge is\text{-}a(c_p, c_s)\}$ and those that are outside this path $\mathcal{C}_{inc} = \{c_{inc} \mid c_{inc} \in \mathcal{C}_{intermediate} \wedge \neg is\text{-}a(c_p, c_s)\}$. For those concepts in $\mathcal{C}_{inc}$ it is clear that they would have led to a contradiction, i.e., pruning them away does not affect any relevant pattern. The other concepts in $\mathcal{C}_{path}$ lie on the path and therefore these patterns are also elements of the equivalence class $[p']$ with $p' = spec(p)$. Leaving these patterns out, only prunes patterns with the same equivalent maximal specialization pattern p' that has been found already.

- Concept restriction: The concept restriction just restricts the concept of

a term in the conjunctive pattern. The patterns where the concept had not been set to this concept would still be members of the equivalence class and thus, leaving them out does not interfere with completeness as the most special equivalent pattern for this equivalence class has been found already.

- Instantiation: The instantiation operation is just allowed for direct instances of the concept in the concept restriction. Thus, no specialization at the concept restriction is performed, i.e., $p = spec(p)$.

2. Non-redundancy

   For non-redundancy, the additional condition leaving out the patterns with a frequency below $minfreq$ is irrelevant as patterns are just left out and the valid refinements are not affected. It has to be shown that the use of the specialization function does not affect non-redundancy for the different refinements:

   - Lengthening: This operation is never called after any of the other refinements was called and is not affected by the specialization function.
   - Temporal refinement: This refinement operator restricts the temporal relations consecutively from left to right without leaving out any predicate pair. The specialization operation only prunes options for future refinements; if any of the previously processed predicate pairs is further restricted (from one temporal relation to zero temporal relations) the pattern must be inconsistent and is not relevant for the mining task.
   - Unification: After applying the lengthening and temporal refinement operations, all resulting patterns are mutually different (lengthening results in unique sequences and for each unique sequence different completely constrained temporal restrictions are generated). The unification process itself also creates just different unification patterns. Redundancy could thus only occur w.r.t. the concept restriction. As the concept restriction is immediately specialized after unification, the most special equivalent pattern is only created once (and the intermediate patterns are left out as described above in the completeness part of the proof).
   - Concept refinement, instantiation: As none of the influencing operations (that can be affected by the specialization function) can be applied after these operations, the non-redundancy is not affected.

As it has been shown that none of the relevant patterns is left out by the changes and non-redundancy is still given, it follows that $\dot{\rho}$ with the minimal frequency condition and the maximal specifity condition is optimal for $\mathcal{P}_{freq}$.

■

Algorithm 7 *MiTemP-main* (Pattern Generation)

Input: $ds = (\mathcal{P}, \mathcal{O}, \textit{direct-instance-of}, dss)$, win_{size}, $size_{min}$, $size_{max}$, $minfreq$, $maxlevel$ /* dynamic scene, window size, minimal and maximal pattern size, min. frequency, max. refinement level */

Output: All frequent patterns $\mathcal{P}_{freq}$ with $size_{min} \leq size \leq size_{max}$

1: Init $\mathcal{P}_{freq} = \emptyset, i = 1$
2: $C_i \leftarrow \textit{create\_single\_predicate\_patterns}()$ /* Create candidates for each pred. template */
3: **while** $C_i \neq \emptyset \wedge i \leq maxlevel$ **do**
4: $\quad support[|C_i|] \leftarrow$ MiTemP-support(ds, win_{size}, C_i)
5: $\quad L_i = \{c_j \in C_i \mid \frac{support(c_j)}{max\_supp} \geq minfreq\}$
6: $\quad \mathcal{P}_{freq} \leftarrow \mathcal{P}_{freq} \cup \{l \in L_i \mid size_{min} \leq size(l) \leq size_{max} \wedge \vec{v}(l) = (v_l, v_t, v_u, v_c, v_i) \wedge (v_t = \frac{v_l \cdot (v_l - 1)}{2} \vee v_u = v_c = v_i = 0)\}$
7: $\quad i \leftarrow i + 1$
8: $\quad C_i = \textit{MiTemP-gen-lengthening}(L_{i-1}) \cup \textit{MiTemP-gen-temp-refinement}(L_{i-1}) \cup \textit{MiTemP-gen-unification}(L_{i-1}) \cup \textit{MiTemP-gen-concept-refinement}(L_{i-1}) \cup \textit{MiTemP-gen-instantiation}(L_{i-1})$
9: **end while**

4.5.3 Temporal Pattern Mining Algorithms

In the previous sections, the temporal pattern mining task has been introduced formally and the optimality for the refinement operator has been shown. The optimality holds also for the adapted version that takes as much knowledge as possible into account by only creating the most special representative of a set of equivalent patterns, and pruning away those patterns that are known to be infrequent (or inconsistent). In this section, algorithms for the temporal pattern mining process are presented, including the main loop, the five refinement operations, support computation, and the generation of the specialized patterns.

Main Loop and Refinement Operations

The main loop of the algorithm is shown in Algorithm 7. The inputs are the dynamic scene, the size of the sliding window, and the minimal frequency. Additionally, minimal and maximal sizes for the conjunctive pattern as well as a maximal refinement level can be defined in order to restrict the search space. The pattern space is traversed level-wise starting from the empty pattern and successively performing a general-to-specific search. In each step of the *while* loop the refinement operations are applied to the frequent patterns of the previous step. The relevant patterns are stored in P_{freq} and the frequent patterns of the current level are used for candidate generation in the next step.

As the specialization function *spec* is applied to different patterns, it can happen

that the resulting equivalent pattern is on a higher refinement level, i.e., $spec(p) = p'$ with $|\vec{v}(p')| > |\vec{v}(p)|$. This has no effect on the result of the mining algorithm. Only the lengthening refinement steps take more than one pattern into account for next level candidate generation, but in these cases, the generation of the most special equivalent pattern has no effect (it is identical to the original pattern; specialization is thus not even invoked), i.e., no level is missed out. In the other cases, only single patterns are refined and thus, it does not affect the result if some refinement steps are "skipped" due to equivalence of the intermediate patterns and independence between the different paths in the search space.

The refinement operations are shown in Algorithms 8 - 12. All algorithms are only applied to the subset of the frequent patterns of the previous step L_{i-1} where the operation is allowed (Def. 4.50). It is checked if the conditions for applying the operations are satisfied in order to keep the specified order of the optimal refinement operator.

The lengthening operation differs from the remaining refinement operations as the next level candidates are generated by combining two of the previous frequent patterns with a common $(n-1)$-prefix. In the other cases, the new candidates are just generated from one frequent pattern of the previous level.

Initially, single predicate patterns are generated for each predicate template in the main algorithm (Algorithm 7). The following lengthening refinement steps in Algorithm 8 always take two frequent patterns of the previous level into account. If they have a shared $(n-1)$-prefix and if all their $(n-1)$-subsets are frequent, a new pattern is added to the candidate set. In this algorithm, for each generated pattern a new initial temporal restriction $\mathcal{TR}'$ and concept restriction $\mathcal{CR}'$ is created. The temporal relation $\mathcal{TR}' = init\_tr(cp)$ consists of a set of potential interval relations for all atomic pattern pairs, i.e., $\forall i, j :!\exists(i, j, \mathcal{TR}'_{i,j}) \in \mathcal{TR}'$ with $1 \leq i \leq (n-1)$ and $i \leq j \leq n$ where $n = size(cp)$ with $cp = (ap_1, \ldots, ap_n)$. Due to the implicit order of the atomic patterns, $\mathcal{TR}'_{i,j} = \mathcal{IR}_{\leq}$ if $pi_i \leq_{lex} pi_j$ and $\mathcal{TR}'_{i,j} = \mathcal{IR}_{older}$ if $pi_i >_{lex} pi_j$ with $ap_k = (pi_k, args_k)$. The initial concept restriction $\mathcal{CR}' = init\_cr(cp)$ consists of all variable/concept pairs for all atomic patterns in the conjunctive pattern. Let $V_1, \ldots, V_m$ be all variables in the conjunctive pattern and $ci_1, \ldots, ci_m$ be all corresponding concepts in the predicate templates with $m = length(cp)$, then $\mathcal{CR}' = \{(V_1, ci_1), \ldots, (V_m, ci_m)\}$.

The candidate generation of the temporal refinement operator is shown in Algorithm 9. The next position in the temporal restriction is identified and for each available interval relation for this atomic pattern pair a new pattern is generated and added to the candidate set. As the selection of the interval relation might allow futher restrictions w.r.t. other interval relation sets of other atomic pattern pairs, the specialization function for the temporal restriction is called (see Algorithm 15). The result might be an inconsistent temporal restriction or a new temporal restriction but might also be the same temporal restriction as before.

Algorithm 8 *MiTemP-gen-lengthening* (Lengthening Candidate Generation)

Input: $\mathcal{L}_{i-1}$ /* Frequent patterns of the previous step */
Output: New candidate patterns C_i

Init $C_i = \emptyset$
$\mathcal{F}_{i-1} = \{l \in \mathcal{L}_{i-1} \mid lengthening\_allowed(l)\}$
for $(p_m = (cp_m, \mathcal{TR}_m, \mathcal{CR}_m) \in \mathcal{F}_{i-1})$ **do**
  for $(p_n = (cp_n, \mathcal{TR}_n, \mathcal{CR}_n) \in \mathcal{F}_{i-1}$ with $n \geq m)$ **do**
    if $cp_m = (ap_{m,1} \wedge ap_{m,2} \wedge \ldots \wedge ap_{m,i-2} \wedge ap_{m,i-1}) \wedge cp_n = (ap_{m,1} \wedge ap_{m,2} \wedge \ldots \wedge ap_{m,i-2} \wedge ap_{n,i-1})$ **then**
      $cp_{new1} = (ap_{m,1} \wedge \ldots \wedge ap_{m,i-2} \wedge ap_{m,i-1} \wedge ap_{n,i-1})$
      $cp_{new2} = (ap_{m,1} \wedge \ldots \wedge ap_{m,i-2} \wedge ap_{n,i-1} \wedge ap_{m,i-1})$
      /* Add if all subsets are frequent (prune step) */
      if for all $(i-1)$ subsets of cp_{new1} there exists a frequent pattern in $\mathcal{F}_{i-1}$ **then**
        $p_{new1} = (cp_{new1}, \mathcal{TR}', \mathcal{CR}')$ with $\mathcal{TR}' = init\_tr(cp_{new1})$ and $\mathcal{CR}' = init\_cr(cp_{new1})$
        $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup p_{new1}$
      end if
      if $m \neq n$ **then**
        /* Avoid duplicate if m = n */
        if for all $(i-1)$ subsets of cp_{new2} there exists a frequent pattern in $\mathcal{F}_{i-1}$ **then**
          $p_{new2} = (cp_{new2}, \mathcal{TR}', \mathcal{CR}')$ with $\mathcal{TR}' = init\_tr(cp_{new2})$ and $\mathcal{CR}' = init\_cr(cp_{new2})$
          $\mathcal{C}_i \leftarrow \mathcal{C}_i \cup p_{new2}$
        end if
      end if
    end if
  end for
end for
return C_i

The algorithm for unification candidate generation can be seen in Algorithm 10. For each pattern to be refined, it is checked for the highest index where a unification with a previous argument has been performed before. Only those positions after this index are taken into account for candidate generation. For each following position all variables of the previous indices are used in order to create candidate patterns. It is also checked if two variables are unifiable by testing if one of the corresponding concepts subsumes the other.

The concept refinement candidate generation is shown in Algorithm 11. For each pattern it is checked for the term with the highest ID w.r.t. the lexicographical identifier order among all those terms that have been conceptually refined already. Only this term and the terms whose identifier is lexicographically greater are refined.

Algorithm 9 *MiTemP-gen-temporal-refinement* (Temporal Refinement Candidate Generation)

Input: $\mathcal{L}_{i-1}$ /* Frequent patterns of the previous step */
Output: New candidate patterns C_i

Init $C_i = \emptyset$
$\mathcal{F}_{i-1} = \{l \in \mathcal{L}_{i-1} \mid temporal\_refinement\_allowed(l)\}$
for all $p \in \mathcal{F}_{i-1}$ with $p = (cp, \mathcal{TR}, \mathcal{CR})$ **do**
  $l = v_l$ with $\vec{v}(p) = (v_l, v_t, v_u, v_c, v_i)$
  $ri = v_t + 1$ /* temporal relation set index to refine */
  init $\mathcal{TR}' = \mathcal{TR}$
  for all $tr \in \mathcal{TR}_{ri}$ **do**
    $\mathcal{TR}'_{ri} = \{tr\}$
    $\mathcal{TR}'_{spec} \leftarrow$ *MiTemP-temp-rest*$(\{(ri_1, ri_2)\}, \mathcal{TR}', l)$ with $\mathcal{TR}'_{ri_1,ri_2} = \mathcal{TR}'_{ri}$
    if $\mathcal{TR}'_{spec} \neq fail$ **then**
      $p' = (cp, \mathcal{TR}'_{spec}, \mathcal{CR})$
      $C_i \leftarrow C_i \cup p'$
    end if
  end for
end for
return C_i

A new pattern is generated and added to the set of pattern candidates for each possible direct sub-concept of the current concept in the concept restriction.

Algorithm 12 shows the candidate generation procedure for instantiation. First, it is checked for the highest index such that no instantiation has been performed after this index. For this position and the positions behind, the term is replaced by all direct instances of the corresponding concept in the concept restriction.

Support Computation

The support computation procedure is shown in Algorithm 13. The input to the algorithm is the dynamic scene, the size of the sliding window, and a list of temporal patterns where the support should be computed. There are two loops – one for the sliding window positions and the other for the list of patterns to be checked. In the beginning, for each pattern the support interval list is initialized by the empty set and the next time when the pattern should be checked is initialized by $-\infty$. The motivation is that if a pattern is known to match also after the current sliding window position, it is not necessary to check for a match. Thus, the *next_to_check* variable keeps track of the next position where the pattern should be checked. It is also possible to have positions where a pattern cannot match (as one of its more general patterns does not match at these positions). This is tested by calling *potential_match*.

Algorithm 10 *MiTemP-gen-unification* (Unification Candidate Generation)

Input: $\mathcal{L}_{i-1}$ /* Frequent patterns of the previous step */
Output: New candidate patterns C_i

Init $C_i = \emptyset$
$\mathcal{F}_{i-1} = \{l \in \mathcal{L}_{i-1} \mid unification\_allowed(l)\}$
for all $p \in \mathcal{F}_{i-1}$ with $p = (cp, \mathcal{TR}, \mathcal{CR})$ **do**
  $t = (t_1, \ldots, t_n)$ is the list of arguments of cp
  $prev_{unif}$ is the index of the last unification
  for all $j \in \{prev_{unif} + 1, \ldots, n\}$ **do**
    for all $V_k \in setof(\{t_1, \ldots, t_{j-1}\})$ **do**
      if $(t_j, ci_1) \in \mathcal{CR} \wedge (V_k, ci_2) \in \mathcal{CR} \wedge (is\text{-}a(ci_1, ci_2) \vee is\text{-}a(ci_2, ci_1))$ **then**
        $cp' = cp\theta$ with $\theta = \{t_j/V_k\}$ /* substitution of variable */
        $\mathcal{CR}'_{spec} = specializeConceptRestriction(cp', \mathcal{CR})$
        if $\mathcal{CR}'_{spec} \neq fail$ **then**
          $p' = (cp', \mathcal{TR}, \mathcal{CR}'_{spec})$
          $C_i \leftarrow C_i \cup p'$
        end if
      end if
    end for
  end for
end for
return C_i

Only at those positions where a match can occur, the pattern matching procedure is called. If a match has been found, its match interval is added to the set of match intervals of the pattern and the *next_to_check* time is updated by the end time of the identified match (see Def. 4.35 on page 104). It should be noted that just the first match is taken into account (instead of collecting all matches of the pattern). This saves time and still leads to a correct support computation as it is checked for the pattern again as soon as the end time of the validity interval of the identified match is reached. After reaching the end of the dynamic scene, the lengths of the support interval sets constitute the support of the checked patterns.

After having processed all patterns to be checked, the sliding window is moved to the next position where a relevant change occurs (w.r.t. to the sliding window), i.e., the sliding window start time is moved to the next position where either a new fact enters the sliding window or an observed fact leaves the sliding window. This procedure takes advantage of the fact that patterns that matched in the previous step must also match until some predicate leaves the window and patterns that did not match in the previous step can only match if some new predicate enters the window. The next relevant position is provided by *next_change*.

Algorithm 11 *MiTemP-gen-concept-refinement* (Concept Refinement Candidate Generation)

Input: $\mathcal{L}_{i-1}$ /* Frequent patterns of the previous step */
Output: New candidate patterns C_i

Init $C_i = \emptyset$
$\mathcal{F}_{i-1} = \{l \in \mathcal{L}_{i-1} \mid concept\_refinement\_allowed(l)\}$
Init $\mathcal{CR}' = \mathcal{CR}$
$t = args(cp) = (t_1, \ldots, t_m)$ is the tuple of terms in cp
for all $p \in \mathcal{F}_{i-1}$ **do**
  $\mathcal{CR} = (c_1, \ldots, c_n)$ with $p = (cp, \mathcal{TR}, \mathcal{CR})$
  $start$ is the highest ID of $\mathcal{CR}$ where a concept refinement step has been applied
  for all $k \in \{start, \ldots, n\}$ **do**
    if $\nexists prev : prev < k \wedge t_k = t_{prev}$ **then**
      $\mathcal{C}_{sub} = \{x \mid is\text{-}a(x, c_k)\}$
      for all $c' \in \mathcal{C}_{sub}$ **do**
        $\mathcal{CR}'_k = c'$
        $p' = (cp, \mathcal{TR}, \mathcal{CR}')$
        $C_i \leftarrow C_i \cup p'$
      end for
    end if
  end for
end for
return C_i

Generation of the Most Special Equivalent Pattern

Algorithm 14 shows how the most special equivalent pattern can be computed. The algorithm initiates the temporal constraint propagation procedure (Algorithm 15) and the specialization of the concept restriction (Algorithm 17). In both cases, the upmost specialized temporal restriction and concept restriction are generated. It is also checked if the current pattern is inconsistent. In this case, the *inconsistent pattern* – $\perp$ – is returned. For the specialization of the temporal restriction, initially all interval relations for all atomic pattern pairs must be checked. Thus, all possible atomic pattern pairs are generated and used as input to the temporal constraint propagation algorithm.

Algorithm 15 corresponds to the constraint propagation algorithm presented by Allen [All83, p. 835]. The input to the algorithm is a set of atomic pattern index pairs representing the interval relations that have been changed (or should be checked). For each interval pair (i, j) all remaining intervals k are tested if the values of the corresponding interval relation set between i and k are consistent with the intervals by taking into account the composition table through j (analogously this is done for j and k by comparing the set with the compositions through

Algorithm 12 *MiTemP-gen-instantiation* (Instantiation Candidate Generation)

Input: $\mathcal{L}_{i-1}$ /* Frequent patterns of the previous step */
Output: New candidate patterns C_i

Init $C_i = \emptyset$
$\mathcal{F}_{i-1} = \{l \in \mathcal{L}_{i-1} \mid instantiation\_allowed(l)\}$
$t = args(cp) = (t_1, \ldots, t_m)$ is the list of terms in cp with $p = (cp, \mathcal{TR}, \mathcal{CR})$
for all $p \in \mathcal{F}_{i-1}$ **do**
  $start = \max(\{j \mid 1 \leq j \leq m \wedge \nexists prev : prev < j \wedge t_j = t_{prev} \wedge \neg atom(t_j))\}$
  for all $k \in \{start, \ldots, m\}$ **do**
    if $\nexists prev : prev < k \wedge t_k = t_{prev} \wedge \neg atom(t_k)$ **then**
      $\mathcal{I} = \{x \mid direct\text{-}instance\text{-}of(x, ci_k) \wedge (t_k, ci_k) \in \mathcal{CR}\}$
      for all $inst \in \mathcal{I}$ **do**
        $cp' = cp\theta$ with $\theta = \{t_k/inst\}$ /* substitution of variable */
        $\mathcal{CR}' = \mathcal{CR} \setminus \{(t_k, ci_k)\} \cup (inst, ci_k)$
        $p' = (cp', \mathcal{TR}, \mathcal{CR}')$
        $C_i \leftarrow C_i \cup p'$
      end for
    end if
  end for
end for
return C_i

i). *potentialTemporalRelations* (shown in Algorithm 16) corresponds to the constraints relation *Constraints* in [All83, p. 835]. Given the set of possible interval relations between A and B and between B and C it creates the union of all possible interval relations for intervals A and C w.r.t. the composition table by taking into account all possible value combinations.

If the set of possible interval relations between the atomic patterns ap_i and ap_k or between ap_j and ap_k could be restricted, the corresponding pair is added to the set of pairs to be processed in order to propagate the changes in the constraint network.

The specialization of the concept restriction is presented in Algorithm 17 on page 134. The algorithm identifies the most special concept for all terms in the conjunctive pattern. If the term is an object (i.e., if it is atomic), it is checked if it is consistent with the current corresponding concept in the concept restriction. If this is not the case, the algorithm returns *fail*. Otherwise, the concept for this term is replaced by its most special concept, i.e., the concept where it is in the *direct-instance-of* relation.

If the term is not atomic but a variable, all concepts of the corresponding positions in the predicate templates are collected and put together with the corresponding concept in the concept restriction. Out of this set the concept which is subsumed by all other concepts is selected as new (most special) concept for this term in the

Algorithm 13 *MiTemP-support* (Support Computation)

Input: $ds = (\mathcal{P}, \mathcal{O}, i, dss)$, win_{size}, $\mathcal{PL}$ /* dynamic scene, window size, pattern list */
Output: Support values $support(p_i)$ for all patterns $p_i \in \mathcal{PL}$

1: /* s_{min} is the earliest start and e_{max} is the latest end time */
2: Init $supp\_intervals(p_i) = \emptyset$, $next\_to\_check(p_i) = -\infty$
3: Init $\mathcal{P}_{win} = \emptyset$, $w_{start} = s_{min} - win_{size}$
4: **while** $w_{start} \leq e_{max}$ **do**
5: **for** $p_i \in \mathcal{PL}$ **do**
6: **if** $(potential\_match(p_i, w_{start}) \wedge next\_to\_check(p_i) \leq w_{start})$ **then**
7: $m \leftarrow pattern\_match(p_i, w_{start}, win_{size})$
8: **if** $m \neq null$ **then**
9: $supp\_intervals(p_i) \leftarrow supp\_intervals(p_i) \cup get\_support\_interval(m)$ /* Add newly covered interval */
10: $next\_to\_check(p_i) \leftarrow get\_min\_end\_time(m)$
11: **end if**
12: **end if**
13: $w_{start} \leftarrow next\_change(w_{start})$ /* move to next position with a change in the scene */
14: **end for**
15: **end while**
16: **for** $p_i \in \mathcal{PL}$ **do**
17: $support(p_i) \leftarrow length(supp\_intervals(p_i))$
18: **end for**

Algorithm 14 *spec* (Most Special Equivalent Pattern Generation)

Input: Temporal pattern $p = (cp, \mathcal{TR}, \mathcal{CR})$
Output: The most special equivalent temporal pattern $p' = (cp, \mathcal{TR}', \mathcal{CR}')$

1: Init the set of predicate index pairs $\mathcal{PIP} = \{(i_1, j_1), (i_2, j_2), \ldots, (i_m, j_m)\})$ with $n = size(cp)$ and $m = \frac{n \cdot (n-1)}{2}$
2: $\mathcal{TR}' = MiTemP\text{-}temp\text{-}restr(\mathcal{PIP}, \mathcal{TR}, n)$
3: $\mathcal{CR}' = specializeConceptRestriction(cp, \mathcal{CR})$
4: **if** $\mathcal{TR}' = fail \vee \mathcal{CR}' = fail$ **then**
5: **return** $\perp$
6: **else**
7: **return** $p' = (cp, \mathcal{TR}', \mathcal{CR}')$
8: **end if**

concept restriction. If the algorithm found out that there are two concepts that do not subsume each other in any direction, the pattern is inconsistent (see Def. 4.27) and the algorithm returns *fail*.

Algorithm 15 *MiTemP-temp-restr* (Temporal Constraint Propagation)

Input: A set of predicate pair indices $\mathcal{T} = \{(i_1, j_1), \ldots, (i_n, j_n)\}$, a temporal restriction $\mathcal{TR}$, the number of predicates l

Output: The refined temporal restriction $\mathcal{TR}'$

Initialize set of changed temporal relations $\mathcal{T}' = \emptyset$
Initialize $\mathcal{TR}' = \mathcal{TR}$
for all $(i, j) \in \mathcal{T}$ **do**
  $\mathcal{T} \leftarrow \mathcal{T} \setminus \{(i, j)\}$
  for all $k \in \{1, \ldots, l\} \setminus \{i, j\}$ **do**
    $i' = \min(i, k), k' = \max(i, k)$/* switch indices if needed */
    $\mathcal{TR}_{tmp} = potentialTemporalRelations((i', k'), j, \mathcal{TR}') \cap \mathcal{TR}_{i',k'}$
    if $\mathcal{TR}_{tmp} = \emptyset$ **then**
      return fail /* inconsistent temporal restriction */
    else if $\mathcal{TR}_{tmp} \subset \mathcal{TR}_{i',k'}$ **then**
      $\mathcal{TR}'_{i',k'} = \mathcal{TR}_{tmp}$ /* replace by restricted set */
      $\mathcal{T}' \leftarrow \mathcal{T}' \cup (i', k')$ /* add index pair for check */
    end if
    $j' = \min(j, k), k' = \max(j, k)$/* switch indices if needed */
    $\mathcal{TR}_{tmp} = potentialTemporalRelations((j', k'), i, \mathcal{TR}') \cap \mathcal{TR}_{j',k'}$
    if $\mathcal{TR}_{tmp} = \emptyset$ **then**
      return fail /* inconsistent temporal restriction */
    else if $\mathcal{TR}_{tmp} \subset \mathcal{TR}_{j',k'}$ **then**
      $\mathcal{TR}'_{j',k'} = \mathcal{TR}_{tmp}$ /* replace by restricted set */
      $\mathcal{T}' \leftarrow \mathcal{T}' \cup (j', k')$ /* add index pair for check */
    end if
  end for
end for
return $\mathcal{TR}'$

4.5.4 Complexity Examination

This section provides a brief discussion about the complexity of the pattern space and the support computation algorithm. Let n be the size of a temporal pattern, k be the number of predicate templates, a be the maximal arity of the predicate templates, i be the number of instances in the dynamic scene, c be the number of concepts in the dynamic scene schema, and t be the number of relevant temporal relations for the temporal restrictions with $t = |\mathcal{IR}_{\leq}|$.

For a fixed size n of the conjunctive pattern the number of possible combinations of the predicate templates (relevant order, multiple occurrences) leads to the number of possible basic patterns, namely k^n basic patterns.

The number of temporal relations is related to the size of the conjunctive pattern. If the conjunctive pattern consists of n atomic patterns, there are $\frac{n \cdot (n-1)}{2}$ temporal

Algorithm 16 *potentialTemporalRelations*

Input: A predicate index pair (i, j), a third predicate index k for comparison, a temporal restriction $\mathcal{TR}$

Output: A set $\mathcal{TR}_{ret}$ of potential temporal relations between predicates at indices i and j

Initialize $\mathcal{TR}_{ret} = \emptyset$
if $i < k$ **then**
  $\mathcal{TR}'_{i,k} = \mathcal{TR}_{i,k}$
else
  $\mathcal{TR}'_{i,k} = inverse(\mathcal{TR}_{k,i})$ /* set of inverse elements */
end if
if $k < j$ **then**
  $\mathcal{TR}'_{k,j} = \mathcal{TR}_{k,j}$
else
  $\mathcal{TR}'_{k,j} = inverse(\mathcal{TR}_{j,k})$
end if
for all $r_1 \in \mathcal{TR}'_{i,k}$ **do**
  for all $r_2 \in \mathcal{TR}'_{k,j}$ **do**
    $\mathcal{TR}_{ret} \leftarrow \mathcal{TR}_{ret} \cup composition\_table(r_1, r_2)$
  end for
end for
return $\mathcal{TR}_{ret}$

relations. As each temporal relation is forced to consist of one element only, in the mining task there are $t \cdot \frac{n \cdot (n-1)}{2}$ combinations of temporal relations. Note that this formula does not take into account impossible combinations leading to inconsistent patterns and the cases where $\mathcal{IR}_{older}$ is used instead of $\mathcal{IR}_{\leq}$.

For the worst case estimation, it is assumed that there are $v = n \cdot a$ variables in the conjunctive pattern, i.e., it is assumed that each of the predicates has maximal arity a. The number of possible variable unifications is equal to the number of all possible clusters of arbitrary sizes for v objects. This leads to the Bell number [Bel34] (cf. Table 4.4). The number of all possible variable combinations for v variables can be computed by the formula (cf. [Pit97]):

$$\sum_{m=1}^{v} \left\{ {v \atop m} \right\} = \sum_{m=1}^{v} \frac{1}{m!} \sum_{j=1}^{m} (-1)^{m-j} \binom{m}{j} j^v$$

The Bell numbers for some different v are shown in Table 4.4.

The worst case estimation of the number of possible concept refinement combinations for all variables is $c^v = c^{n \cdot a}$, i.e., each variable could be refined to each concept in the dynamic scene schema.

The computation for the possible instantiations is similar to the one for the

Algorithm 17 *specializeConceptRestriction*

Input: Conjunctive pattern cp and concept restriction $\mathcal{CR} = \{(t_1, ci_1), \ldots, (t_n, ci_n)\}$
Output: The upmost specialized concept restriction $\mathcal{CR}'$

```
1: Init CR' = CR
2: for all terms t_i ∈ {t_1, ..., t_n} do
3:    if atom(t_i) then
4:       if ¬instance-of(t_i, ci_i) then
5:          return fail
6:       end if
7:       direct-instance-of(t_i, ci_spec) /* Get direct concept of object */
8:    else
9:       Collect all concepts of the arguments of the corresponding predicate templates
         of t_i in the set C
10:      Init ci_spec = ci_i
11:      for all c ∈ C do
12:         if is-a(c, ci_spec) then
13:            ci_spec = c
14:         else if ¬is-a(ci_spec, c) then
15:            return fail
16:         end if
17:      end for
18:   end if
19:   CR' ← CR' \ {(t_i, ci_i)} ∪ (t_i, ci_spec)
20: end for
21: return CR'
```

v	1	2	3	4	5	6	7	8	9	10
B_v	1	2	5	15	52	203	877	4140	21147	115975

Table 4.4: Bell numbers

concept refinements, but there exists the additional constraint that each object is assigned to a variable once at most. Thus, we have the number of combinations like selecting v' values from a set of i values with relevant order (and no multiple uses of the values). As it is allowed to have zero instantiations up to the maximal number of instances i, the number of possible instantiations is $\sum_{v'=0}^{i} \binom{i}{v'} \cdot v'!$. Actually, if there are less variables than instances, i.e., $n \cdot a < i$ the number of combinations is smaller: $\sum_{v'=0}^{n \cdot a} \binom{i}{v'}$.

During pattern mining, patterns of different sizes are generated. The overall

estimation for the number of patterns therefore is:

$$\sum_{n=1}^{size_{max}} \left(\overbrace{k^n}^{\text{length.}} \cdot \overbrace{t \cdot \frac{n \cdot (n-1)}{2}}^{\text{temp. ref.}} \cdot \overbrace{\sum_{m=1}^{v} \frac{1}{m!} \sum_{j=1}^{m} (-1)^{m-j} \binom{m}{j} j^v}^{\text{unification}} \cdot \overbrace{c^v}^{\text{conc. ref.}} \cdot \overbrace{\sum_{v'=0}^{i} \binom{i}{v'} \cdot v'!}^{\text{instantiation}} \right)$$

The complexity of the support computation depends on the number of window positions to check, the number of patterns, and the number of combinations to check at each window position. As mentioned above, it is only necessary to check the window positions where a predicate enters or leaves the sliding window. Let $p = |\mathcal{P}|$ be the number of predicates in the dynamic scene, w be the size of the sliding window, and $sl = e_{max} - s_{min} + w$ be the length of the dynamic scene. In the case of discrete time points (integers), the maximal number of window positions to check is $min(2 \cdot p, sl)$ (each predicate has one start and one end time – if they are all mutually different there are $2 \cdot p$ time points to check). In the worst case, all possible window positions occur in the dynamic scene and all sl positions must be checked.

At each window position all tp patterns must be checked (if it is not already known that they match or do not match at this position). In the worst case, for each pattern all possible combinations of the predicates in the current sliding window must be tested. Let q be the maximal number of occurring predicates in a window position and let n be the size of the pattern to check. The number of possible mappings between predicates in a window position and the atomic patterns in the conjunctive pattern is equal to the number of combinations to draw n of q elements (ignoring the order and without allowing to draw elements repeatedly): $\binom{q}{n} = \frac{q!}{n! \cdot (q-n)!}$ [BSMM99, p. 744]. For each legal mapping (i.e., where the conjunctive pattern could be unified with the mapped part of the dynamic scene) it must be checked if the temporal restriction and the concept restriction are satisfied. The complexity for these tests are $\frac{n \cdot (n-1)}{2}$ and $n \cdot a$, respectively. These are the maximal numbers of temporal relations and variables to check for consistency with the restrictions. Altogether the worst case complexity for one support scan is:

$$\overbrace{sl}^{\text{seq. length}} \cdot \overbrace{tp}^{\text{\#patterns}} \cdot \overbrace{\binom{q}{n}}^{\text{\#pred. comb.}} \cdot \left(\overbrace{\frac{n \cdot (n-1)}{2}}^{\text{\#temp. rel.}} + \overbrace{n \cdot a}^{\text{\#var.}} \right)$$

In the O notation, this leads to the complexity class $O(q!) \cdot O(n^2 + n)$. As it can be seen, the complexity increases linearly with the sequence length and the number of patterns to check; the check of the concept restriction also just increases linearly

w.r.t. the maximal arity of predicates. The check of the temporal restriction has quadratic complexity w.r.t. the pattern size n. The most crucial part is the number of predicate combinations in a sliding window position which depends on the number of observable predicates.

4.6 Mining Temporal Patterns with WARMR

As the temporal validity intervals of predicates can be seen as just another dimension of relations, it should be possible to transfer the learning problem to relational association rule mining. Intuitively, it seems to be unhandy but feasible to add information about start and end time to every predicate and to regard it as additional dimensions. However, it will be interesting to compare the patterns found by *MiTemP* and *WARMR*.

In this section, we show how *WARMR* can be used in order to mine temporal patterns from the defined representation. *WARMR* has been described in Section 3.1.2 on page 42 and the relevant algorithms have been shown in Algorithms 4, 5, and 6. Before it is shown how the learning problem described in this thesis can be transformed to a learning task for *WARMR*, some general information about the representation of knowledge and learning settings in *WARMR* is presented.

4.6.1 Learning Task Definition in WARMR

The description of learning tasks for *WARMR* consists of the knowledge base (KB), background knowledge (BG), and learning settings including the language bias for the patterns (*queries* in the terms of *WARMR*) to be learned. An implementation of *WARMR* is provided in the *ACE* data mining system maintained by the Declarative Languages and Artificial Intelligence (DTAI) research group of the KU Leuven [BDD+02, BDR+06]. *ACE* provides a number of different relational data mining algorithms including *WARMR* [DT99]. The following description is a brief summary of the information regarding the definition of learning tasks provided in the *ACE* user's manual [BDR+06].

Knowledge Base and Background Knowledge

The knowledge for learning with *WARMR* can be divided into two kinds: First, information for learning related to examples is provided in the knowledge base. This is a set of relevant relations for the mining task. An example from the soccer domain could be the relations between dynamic objects, e.g., qualitative distances, orientation information, higher-level relations: `distance(p1, p2, close)`, `in-front-of(p4, p2)`, `covers(p6, q7)`. Background knowledge consists of the

set of predicates that can be used to derive new information. The informal description in Blockeel et al. [BDR+06, p. 13] is that a "predicate or relation can be considered to be background knowledge if adding an example to the set of examples does not change the definition of that predicate". It could, e.g., be described that a player is just uncovered if there is no object covering him: `uncovered(X) :- not(covers(_Y, X))`. The different kinds of knowledge are represented in *Prolog* syntax.

Learning Settings and Language Bias

For setting up the learning bias in *WARMR*, it is necessary to define *rmode* statements. These statements define how a query can be extended during the generation of new query candidates, i.e., they describe the pattern space. Basically, a *rmode* consists of a predicate and specifications for the arguments, i.e., if an existing (+), new (-), or unique variable (\) or if a constant should be used. It is also possible to define constraints which must be satisfied in order to add such an atom to the query. There are many other possibilities to specify the language bias which are out of scope of this thesis. More details about the bias and other settings can be found, for instance, in Dehaspe's doctoral thesis and in the *ACE* user's manual [Deh98, BDR+06].

4.6.2 Transformation of the Learning Task

In this section, it is presented how *WARMR* can be used in order to mine temporal patterns as described in the previous sections of this chapter. As it will be discussed later, there are no means to handle temporal information explicitely. In the following, it is described how the schema and sequence information can be represented, how the refinement operations are realized, and how the support can be computed in the same way as defined above.

Sequence and Schema Information

As shown in Section 4.1, the description of dynamic scenes contains schematic information about concepts, the predicate templates, and their domains as well as the dynamic predicates with the start and end times of the validity intervals. The domain definitions determine the concepts for each argument of a predicate template, i.e., instances of these predicate templates must have objects as arguments which are instances of the corresponding concepts of the domain definition.

The transformation of the concept hierarchy and corresponding instances is straight forward. The `is-a` and `direct-instance-of` relations can be directly described in the knowledge base. Transitive clauses for querying instances of concepts and sub-concepts of a given concept can be defined in the background knowledge.

The `holds` predicates representing the validity intervals of relations are now represented by relations with an additional argument `i(S,E)` which stands for the time interval with start and end times `S` and `E`. The predicate $holds(pass(p8, p7), \langle 32, 45 \rangle)$ can be represented by `pass(1, p8, p7, i(32, 45))` where the first argument is a unique ID for the predicate, followed by the arguments of the relation (players `p8` and `p7`), and the time interval with start time `32` and end time `45`. The unique ID is necessary in order to avoid multiple uses of predicates in the pattern matching process.

Refinement Operators

The different refinement operations described above have to be modeled by the *rmodes*. For lengthening (i.e., extending a query by a predicate), a *rmode* must be defined for each predicate template. In order to avoid the same predicate being used more than once in a match, it must be guaranteed that the predicate ID variable differs from all other predicate ID variables of this query. This can be forced by the backslash modifier. The *rmode* `rmode(closerToGoal(\Id, -X0, -X1, -I, +Index)` specifies that a `closerToGoal` predicate can be added to the query where the `Id` must be unique (i.e., a new variable unequal to all existing ones), `X0`, `X1`, and `I` which represent the two players and the time interval must be new variables, and that `Index` must be an existing variable. The variable `Index` is used to check if a matched predicate lies within the scope of the sliding window. More details about the support computation can be found below.

WARMR also supports type declarations for arguments of atoms. This information is used to avoid unallowed variable unifications. The types cannot be hierarchically structured, i.e., it is not possible to map the concept hierarchy presented above directly to these types. Nevertheless, the types are helpful in order to introduce different types for interval, index, predicate ID, and object variables. Having defined these types, *WARMR* does not try to unify the variables of different types. In order to handle the concept hierarchy, a transitive *instance-of* relation in the background knowledge is used which also correctly identifies that objects are also instances of super concepts of their direct concept (*direct-instance-of* relation).

Temporal relations between intervals are represented by clauses which check if the temporal relation actually holds for the interval pair, i.e., for each temporal relation (using the interval relations presented in Section 4.2: `before`, `olderContemp`, and `headToHead`) a clause exists and a *rmode* is created. In order to refine a pattern by adding a temporal constraint, one of the temporal clauses is added to the query by relating two intervals of existing predicates of the query to each other. Thus, in the *rmode* specifications (e.g., `rmode(before(+I1, +I2))` with `I1` and `I2` being of the type *interval*) the two interval variables must be existing ones (qualified by the + in the *rmode* declaration) in order to set up a temporal constraint for them.

Unification is handled by a special unification clause (`unif(X, Y)`) which unifies two existing variables in the previous query. The *rmode* declarations of *ACE* also provide means to define *rmodes* which do not introduce a new variable in the new atom but reuse an existing one. However, our intended solution should also cover the instantiation of variables (i.e., using constants). Setting up *rmodes* for all cases (unification, constants, and new variables) and their combinations in predicates with an arbitrary (potentially large) number of arguments would have led to a huge number of *rmodes* for the predicates. Thus, if a new predicate is added to the query all actual arguments of the relation are new variables in the beginning. These can be unified with another variable or can be bound to a constant in further refinement steps.

For instantiation, a *rmode* definition allows a variable to be bound to an instance. The set of instance candidates depends on the predicate where the variable occurs, thus, for each argument of all predicate templates there must be a distinct *rmode* definition (e.g., `eq_obj_uncovered_01(+X, C)` where `01` represents the second argument of the `uncovered` predicate). Only those instances are taken into account which appear at least in one of the predicates at the variable's position in the dynamic scene, i.e., no "impossible" query will be generated here. For each argument of each predicate template, a clause for finding the possible objects is setup (e.g., `useful_constant_uncovered_01(C)`). It is necessary to define all these different `useful_constant` clauses as the sets of relevant objects can differ for each argument.

Concept refinement is performed by adding `instanceOf` predicates, constraining a variable to a certain concept (or one of its sub-concepts). A constraint definition makes sure that for each variable only one `instanceOf` predicate will be added. Additional constraints ensure that a variable will be used only for instantiation or concept refinement and that unified variables are not refined at all (in order to avoid redundant or possibly inconsistent patterns), e.g.,

```
constraint(unif(_,Y), not_occurs(eq_obj_uncovered_01(Y, _))).
constraint(instanceOf(X,_), not_occurs(eq_obj_uncovered_01(X, _))).
constraint(eq_obj_uncovered_01(Y, _), not_occurs(unif(_,Y))).
constraint(eq_obj_uncovered_01(Y, _), not_occurs(instanceOf(Y,_))).
```

The first constraint specifies that a unification can only be performed if there has not been a instantiation for this variable so far. The second constraint only allows concept refinements as long as no instantiation has been performed. The third and fourth constraints specify that a variable is just instantiated as long as it has neither been unified nor conceptually refined.

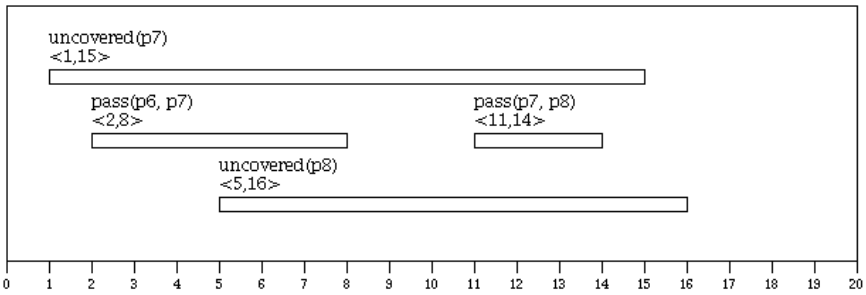

Figure 4.11: Example for *WARMR* conversion

Support Computation

WARMR needs a counting attribute which is used for support computation, i.e., the number of different values of this attribute where a query matches determines the support of the query. In our case, the support is defined to be the number of temporal positions where within a sliding window a match for a pattern exists. In order to let *WARMR* compute the intended support, a predicate `currentIndex` has been introduced which is used as counting attribute. For each existing temporal position for the sliding window, a `currentIndex` predicate is needed in the knowledge base. It represents the end times of all relevant sliding window positions. For a specific index value, only those predicates inside the sliding window are taken into account for matching. The clause `validInterval` in the background knowledge checks if a predicate is relevant w.r.t. the sliding window position.

4.6.3 Sample Representation

Let the dynamic scene schema of this example consist of two concepts *object* and *player* with *is-a*(*player*, *object*) and two predicate templates $pt_1 = (uncovered, (player))$ and $pt_2 = (pass, (player, player))$, and the interval relation function be defined as in Section 4.2 on page 94. Furthermore, let the dynamic scene consist of these four predicates (also illustrated in Fig. 4.11):

```
holds(uncovered(p7), 1, 15).
holds(pass(p6, p7), 2, 8).
holds(uncovered(p8), 5, 16).
holds(pass(p7, p8), 11, 14).
```

$\mathcal{O} = \{p6, p7, p8\}$ and the objects in the dynamic scene are all player instances, i.e., *direct-instance-of*$(p6, player)$, *direct-instance-of*$(p7, player)$, and *direct-instance-of*$(p8, player)$.

Converted to *WARMR* input this dynamic scene and dynamic scene schema results in three parts. The knowledge base is:

```
directInstanceOf(p6, player).
directInstanceOf(p7, player).
directInstanceOf(p8, player).
uncovered(1, p7, i(1, 15)).
pass(2, p6, p7, i(2, 8)).
uncovered(3, p8, i(5, 16)).
pass(4, p7, p8, i(11, 14)).
currentIndex(1).
currentIndex(2).
(...)
```

The background knowledge is defined as:

```
unif(X, Y) :-
        X = Y.

validInterval(S, E, WindowEnd) :-
        windowSize(WindowSize),
        WindowStart is (WindowEnd - WindowSize),
        E > WindowStart,
        S =< WindowEnd.

before(i(_S1, E1), i(S2, _E2)) :-
        E1 < S2.

olderContemp(i(S1, E1), i(S2, _E2)) :-
        S1 < S2,
        E1 >= S2.

headToHead(i(S1, _E1), i(S2, _E2)) :-
        S1 == S2.

pass(Id, O0, O1,  i(S, E), WindowEnd) :-
        pass(Id, O0, O1,  i(S, E)),
        validInterval(S, E, WindowEnd).

useful_constant_pass_O0(C) :-
        findall(X, pass(_,X,_,_), ConstList), setof(Y, member(Y,ConstList), ConstSet), !,
        member(C, ConstSet).

eq_obj_pass_O0(X, Y) :-
        X = Y.

useful_constant_pass_O1(C) :-
        findall(X, pass(_,_,X,_), ConstList), setof(Y, member(Y,ConstList), ConstSet), !,
        member(C, ConstSet).

eq_obj_pass_O1(X, Y) :-
        X = Y.

uncovered(Id, O0,  i(S, E), WindowEnd) :-
        uncovered(Id, O0,  i(S, E)),
        validInterval(S, E, WindowEnd).

useful_constant_uncovered_O0(C) :-
        findall(X, uncovered(_,X,_), ConstList), setof(Y, member(Y,ConstList), ConstSet), !,
        member(C, ConstSet).

eq_obj_uncovered_O0(X, Y) :-
        X = Y.

concept(X) :-
        isA(X,_).
```

```
subConceptOf(X, Y) :-
        isA(X, Y).

subConceptOf(X, Y) :-
        isA(Z,Y),
        subConceptOf(X,Z).

instanceOf(Inst, Concept) :-
        directInstanceOf(Inst, Concept).

instanceOf(Inst, Concept) :-
        directInstanceOf(Inst, SubConcept),
        subConceptOf(SubConcept, Concept).
```

The file with the learning settings (including the *rmode* definitions) is:

```
typed_language(yes).
type(currentIndex(index)).
type(before(interval, interval)).
type(olderContemp(interval, interval)).
type(headToHead(interval, interval)).

type(unif(object, object)).
rmode(unif(+X, +Y)).
constraint(unif(X,Y), not_occurs(unif(_,X))).
constraint(unif(X,Y), X\==Y).

type(pass(id, object, object, interval, index)).
rmode(pass(\Id, -X0, -X1, -I, +Index)).
type(useful_constant_pass_O0(_)).
type(eq_obj_pass_O0(object, _)).
rmode(#(C: useful_constant_pass_O0(C), eq_obj_pass_O0(+X, C))).
constraint(eq_obj_pass_O0(X, Y), occurs(pass(_,X,_,_,_))).
constraint(unif(_,Y), not_occurs(eq_obj_pass_O0(Y, _))).
constraint(instanceOf(X,_), not_occurs(eq_obj_pass_O0(X, _))).
constraint(eq_obj_pass_O0(Y, _), not_occurs(unif(_,Y))).
constraint(eq_obj_pass_O0(Y, _), not_occurs(instanceOf(Y,_))).

type(useful_constant_pass_O1(_)).
type(eq_obj_pass_O1(object, _)).
rmode(#(C: useful_constant_pass_O1(C), eq_obj_pass_O1(+X, C))).
constraint(eq_obj_pass_O1(X, Y), occurs(pass(_,_,X,_,_))).
constraint(unif(_,Y), not_occurs(eq_obj_pass_O1(Y, _))).
constraint(instanceOf(X,_), not_occurs(eq_obj_pass_O1(X, _))).
constraint(eq_obj_pass_O1(Y, _), not_occurs(unif(_,Y))).
constraint(eq_obj_pass_O1(Y, _), not_occurs(instanceOf(Y,_))).

type(uncovered(id, object, interval, index)).
rmode(uncovered(\Id, -X0, -I, +Index)).
type(useful_constant_uncovered_O0(_)).
type(eq_obj_uncovered_O0(object, _)).
rmode(#(C: useful_constant_uncovered_O0(C), eq_obj_uncovered_O0(+X, C))).
constraint(eq_obj_uncovered_O0(X, Y), occurs(uncovered(_,X,_,_))).
constraint(unif(_,Y), not_occurs(eq_obj_uncovered_O0(Y, _))).
constraint(instanceOf(X,_), not_occurs(eq_obj_uncovered_O0(X, _))).
constraint(eq_obj_uncovered_O0(Y, _), not_occurs(unif(_,Y))).
constraint(eq_obj_uncovered_O0(Y, _), not_occurs(instanceOf(Y,_))).

rmode_key(currentIndex(I)).
root(currentIndex(I)).
rmode(before(+I1, +I2)).
```

```
rmode(olderContemp(+I1, +I2)).
rmode(headToHead(+I1, +I2)).
constraint(headToHead(X,Y), not(X=Y)).

type(concept(_)).
type(instanceOf(object, _)).
rmode(#(C: concept(C), instanceOf(+X, C))).

constraint(instanceOf(X,Y), not_occurs(instanceOf(X,_))).
constraint(instanceOf(X,Y), not_occurs(unif(_,X))).
constraint(unif(_,X), not_occurs(instanceOf(X,Y))).
minfreq().
warmr_maxdepth().
```

4.6.4 Discussion

We showed that the validity intervals of predicates can be represented by one additional argument with the start and end time of the validity interval in relations, and how the relational association rule mining algorithm *WARMR* can be used for mining temporal patterns.

Although mining the intended temporal patterns with *WARMR* is possible, there are some drawbacks of this solution. First of all, redundant patterns with identical matches and patterns which cannot have a support greater than zero are created. This could be avoided by exploiting the implicit information about variables' concepts and temporal interrelations, i.e., by using a composition table as shown in Table 4.3 to remove all impossible temporal relations due to already known temporal relations between predicates as it has been described in Section 4.5.2.

Another drawback is the matching process. The realization of the sliding window by an extra predicate does not take advantage of the sequential structure of the dynamic scene representation. If only the currently visible predicates were taken into account while pattern matching, the support computation would be faster as no irrelevant predicates had to be checked. However, it should be stated that *WARMR* is a generic approach to relational association rule mining and not specialized to temporal representations.

The algorithms that have been presented in Section 4.5.3 utilize the given information about temporal relations as well as hierarchical concept information and take advantage of the temporal structure for the pattern matching in window positions. An automated converter of *MiTemP* learning tasks to a *WARMR* compatible representation is presented in Chapter 6 where also an experiment comparing *WARMR* and *MiTemP* is shown.

Depending on the dynamic scene schema and the minimal frequency, the number of mined patterns can be very large. Thus, it is necessary to have some support in evaluating the created patterns. In the next chapter, it is described how prediction rules can be generated from patterns and how the interestingness of prediction rules can be measured.

Chapter 5

Prediction Rule Generation

The identification of frequent patterns – as presented in the previous chapter – itself might be already interesting in order to learn about correlations and noticeable characteristics in temporal data. An extension to association rules allows for using the learned patterns for the prediction of future situations. Recalling our soccer scenario in Chapter 2 it would, for instance, be useful to equip a soccer playing agent with the ability of predicting future situations or predicting the opponent's behavior. The process of generating prediction rules from temporal patterns is presented in Section 5.1. Section 5.2 introduces means to compute an interestingness measure for prediction rules in order to rank the rules before they are provided to the user (or some other system that might utilize the created rules). In the subsequent Section 5.3, it is described how the rules can actually be used for prediction.

5.1 From Patterns to Prediction Rules

In Section 3.1.2 (p. 32), it has been described how association rules can be generated from the identified large itemsets in a transaction database (i.e., those sets of items which appear together often enough in the transaction database so that the support threshold is exceeded). Large itemsets can be split into two parts and a statement is given in how many cases the second part (the consequence) can also be observed when the first one appears in a transaction. From a given large itemset $ABCD$ it is possible, for instance, to create an association rule $AB \Longrightarrow CD$ (0.8) saying in 80% of the cases where AB was bought in a transaction, CD is also part of this transaction. This concept of association rules can easily be adapted in order to create prediction rules from temporal patterns.

Having an explicit representation of temporal relations allows for introducing a special case of association rules, namely prediction rules. If it is known that some parts of a temporal pattern appear *later* than others, it is possible to focus on association rules where the consequence part happens later, i.e., the start times of

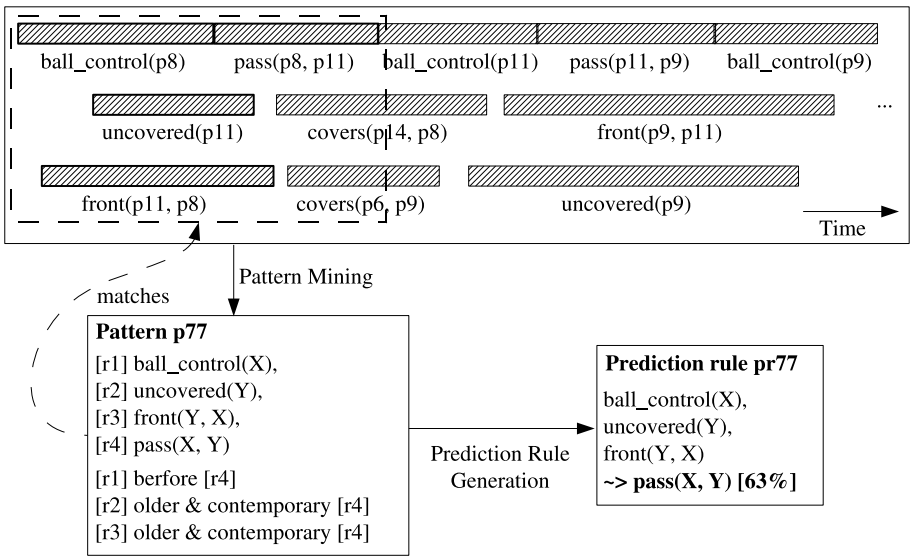

Figure 5.1: Pattern and prediction rule generation

the predicates in the consequence are greater than the start times of the precondition part. Of course, it is still possible to create arbitrary association rules (ignoring the temporal order), but from the predictor's point of view it makes sense to focus on those rules that can be used for the estimation how future situations might look like. Fig. 5.1 illustrates the process of generating prediction rules. After the identification of frequent patterns, these patterns are split into a precondition and consequence part (separated by ~> in Fig. 5.1).

The conjunction of atomic patterns in the temporal patterns has already an implicit temporal order. Every atomic pattern ap_j with a greater index than atomic patterns ap_i (i.e., $j > i$) has a greater or equal start time than ap_i by definition. This fact can be utilized for an efficient prediction rule generation. A prediction rule is defined as:

Definition 5.1 (Prediction Rule)**.** Let $tp_{prec} = (cp_{prec}, \mathcal{TR}_{prec}, \mathcal{CR}_{prec})$, $tp_{cons} = (cp_{cons}, \mathcal{TR}_{cons}, \mathcal{CR}_{cons})$, and $tp = (cp, \mathcal{TR}, \mathcal{CR})$ be temporal patterns. $pr = (tp_{prec}, tp_{cons})$ is a *prediction rule* of tp if and only if:

- the consequence pattern tp_{cons} is equal to the temporal pattern itself: $tp_{cons} = tp$,
- the precondition pattern tp_{prec} subsumes the consequence: $tp_{prec} \succ tp$,
- the conjunctive pattern of the precondition consists of the first n elements of the consequence pattern: $cp_{prec} = (ap_1, ap_2, \ldots, ap_n)$ and $cp = (ap_1, ap_2, \ldots, ap_n, \ldots, ap_m)$ with $0 < n < m$,

- and the temporal relations between the atomic patterns in the precondition must be equal to the corresponding temporal relations in the consequence patterns: $\forall i,j : (i,j,\mathcal{TR}_{i,j}) \in \mathcal{TR}_{prec} \Longrightarrow (i,j,\mathcal{TR}_{i,j}) \in \mathcal{TR}$ with $1 \leq i \leq n$ and $i < j \leq n$.

The prediction rule $pr = (tp_{prec}, tp_{cons})$ can also be denoted by $tp_{prec} \Rightarrow tp_{cons}$.

□

We adapt the notation of Dehaspe [Deh98] for *query extensions* and use the shorter notation with $\rightsquigarrow$. We write $((ap_1, ap_2, \ldots, ap_n \rightsquigarrow ap_{n+1}, \ldots, ap_m), \mathcal{TR}, \mathcal{CR})$ for the prediction rule (tp_{prec}, tp_{cons}) with $tp_{prec} = ((ap_1, ap_2, \ldots, ap_n), \mathcal{TR}_{prec}, \mathcal{CR}_{prec})$ and $tp_{cons} = ((ap_1, ap_2, \ldots, ap_m), \mathcal{TR}, \mathcal{CR})$.

In transaction databases, the initially proposed definition of the *confidence* of an association rule is computed by dividing the support of the whole itemset by the support of the precondition part (e.g., $conf(AB \Longrightarrow CD) = \frac{support(ABCD)}{support(AB)}$ for the example on page 33) [AS94]. This definition can be easily adapted to the temporal patterns in this thesis:

Definition 5.2 (Confidence)**.** Let $pr = (tp_{prec}, tp_{cons})$ be a prediction rule. The *confidence* of a $pr = (tp_{prec}, tp_{cons})$ is computed by the fraction of the support of the consequence pattern and the precondition pattern:

$$conf(pr) = \frac{supp(tp_{cons})}{supp(tp_{prec})}.$$

The threshold $minconf$ defines which prediction rules are regarded as relevant and only those exceeding the minimal confidence should be collected.

□

Höppner [Höp03, p. 86] has identified a problem of this confidence definition: "The greater the (temporal) extent of the pattern, the smaller the probability of observing the pattern in the sliding window. Consequently, the confidence of a rule decreases as the extent of the rule pattern increases." As the temporal extent of the consequence pattern is usually greater than the extent of the precondition, low confidence values can occur. Höppner has shown the extreme case where the gap between the precondition and the unique part of the consequence increases and thus, the confidence value approaches zero.

Due to this "counterintuitive" outcome for the confidence values, Höppner [Höp03, p. 86] introduces a modified rule semantics: "Given a randomly selected sliding window that contains an instance of the premise pattern, then with probability p this window overlaps a sliding window that contains the rule pattern." For a given premise A and consequence B if A can be seen in a sliding window position and the beginning of B is inside the window, it is counted as a regular match of the rule. Our definition of *confidence* of prediction rules follows Höppner's definition [Höp03, p.87]:

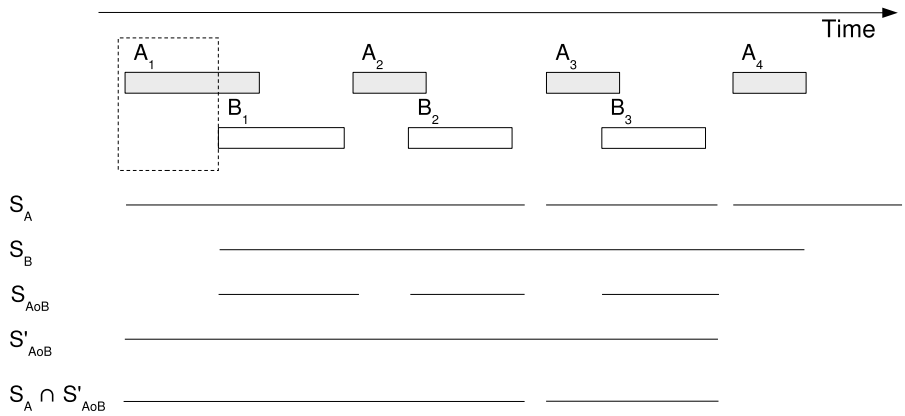

Figure 5.2: Example for observation intervals

Definition 5.3 (Confidence with Modified Rule Semantics)**.** Let $pr = (tp_{prec}, tp_{cons})$ be a prediction rule. The *modified confidence* of pr is computed by

$$conf_{mrs}((tp_{prec}, tp_{cons})) = \frac{length(\mathcal{S}_{prc} \cap \mathcal{S}'_{cons})}{length(\mathcal{S}_{prec})}$$

where $\mathcal{S}_X$ is the support interval set of X and $\mathcal{S}'_X := \mathcal{S}_X \cup \{t - w \mid t \in \mathcal{S}_X\}$.

□

As described in [Höp03, p. 87], $\mathcal{S}'_X$ can be interpreted as pattern X "is visible or will be visible within time w". Fig. 5.2 shows observation intervals for predicates A, B, and A *before* B (cf. [Höp03, p. 66]). It also illustrates how the observation interval is extended by the modified rule semantics for the consequence where A overlaps B. The bottom row shows the intersection of the precondition and modified consequence support intervals. If in a sequence each precondition is followed by the consequence, a confidence value of one is obtained with the modified rule semantics [Höp03].

Using exhaustive search the support intervals and support values have already been computed during frequent pattern generation, i.e., there is no additional computational cost for generating them. However, if some bias has been used in order to restrict the set of patterns to be generated, it might happen that a precondition pattern is generated which does not satisfy the defined bias and thus has not been processed during support computation. Prediction rules with such preconditions might still be relevant and cannot be ignored without risking to lose an important rule. The only secure way to handle these cases is to do a support computation for these (precondition) patterns that have not been processed before in the frequent pattern generation phase.

The temporal relations among predicates shall be used for a structured generation of prediction rules. As mentioned above, it is important that no element of the

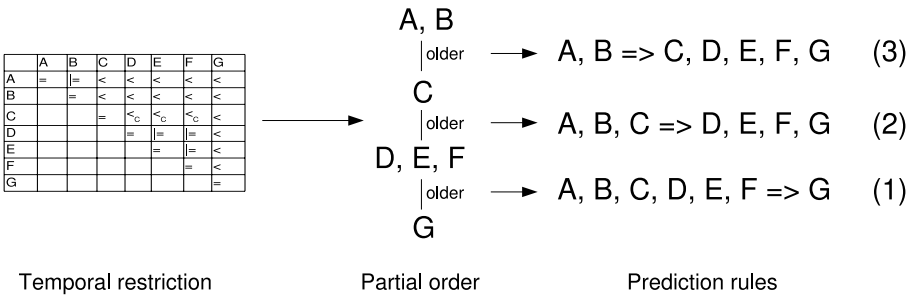

Figure 5.3: Generation of prediction rules

consequence of a prediction rule begins before all elements of the preconditional part have begun. This restriction can be used to reduce the number of prediction rules to be generated considerably and to guide the process of prediction rule generation.

The temporal structure of the temporal patterns can be used in order to efficiently generate prediction rules satisfying this property. The conjunctive pattern already consists of an ordered sequence of atomic patterns. Each atomic pattern must have a later (or equal) start time than its direct predecessor (and also than all other predecessors). Thus, a natural way to split temporal patterns into precondition and consequence is to separate the conjunctive pattern only at these positions. Without this restriction, it would be possible to generate many more rules as all subsets of the atomic patterns in the conjunctive pattern could be used as precondition (leaving at least one predicate in the precondition and consequence). Taking advantage of the sequential structure leads to linear costs w.r.t. the size of the pattern. Let n be the size of the pattern. Then it is only possible to split the pattern at $n-1$ positions into precondition and consequence. If some adjacent atomic patterns are temporarily definitely in a *head-to-head* relation, it makes no sense to separate this set of atomic patterns in the prediction rule. As all these predicates must occur at the same time, they should all be part of either the precondition or the consequence.

Fig. 5.3 illustrates how prediction rules can be generated. The temporal pattern in the example is only split at those positions where adjacent atomic patterns are in an *older* relation, i.e., where the start time of the first predicate must be smaller than the start time of the second predicate. In this case, three prediction rules (illustrated on the right hand side) are generated. A and B as well as D, E, and F are in *head-to-head* relations and thus are not split.

If the prediction rules are generated starting from the "right" side of the pattern, it is possible to leave out prediction rules that cannot exceed the minimal confidence threshold. The first prediction rule to check is the one where all but the last atomic pattern form the precondition. The following prediction rules are generated by moving the last atomic pattern from the precondition to the consequence part. If a prediction rule does not exceed the *minconf* threshold, the upcoming prediction

Algorithm 18 *TePaMi-prediction* (Prediction Rule Generation)

Input: $p = (cp, \mathcal{TR}, \mathcal{CR})$, *minconf* /* Pattern to create prediction rules, minimal confidence threshold */

Output: $\mathcal{PR}$ /* A set of prediction rules with confidence above *minconf* */

```
1: Init PR = ∅
2: Init p_prec = p
3: while size(p_prec) > 1 do
4:    p_prec ← remove_last_predicate(p_prec) /* New pattern without last atomic pattern */
5:    conf = supp'(p) / supp(p_prec)
6:    if conf ≥ minconf then
7:       PR ← PR ∪ (p_prec, p)
8:    else
9:       break /* Not necessary to check further prediction rules */
10:   end if
11: end while
12: return PR
```

rules do not need to be generated and checked as they must also have a confidence value below *minconf*.

Theorem 5.1 (Monotonicity of the confidence values)**.** Let tp be a temporal pattern and $(tp_{prec,1}, tp), \ldots, (tp_{prec,n}, tp)$ be the list of prediction rules that can be generated from tp where $tp_{prec,1}$ consists of all but the last atomic pattern of tp. Each following $tp_{prec,i}$ leaves out one more atomic pattern. It holds that $conf((tp_{prec,i}, tp)) \geq conf((tp_{prec,i+1}, tp))$ for all i with $0 < i < n$.

□

Proof 5.1. The confidence of $(tp_{prec,i}, tp)$ is computed by $\frac{supp(tp)}{supp(tp_{prec,i})}$. As $tp_{prec,i+1}$ just leaves out one atomic pattern it holds that $tp_{prec,i+1} \succ tp_{prec,i}$, i.e., that it is more general. From Theorem 4.3 it follows that $supp(tp_{prec,i+1}) \geq supp(tp_{prec,i})$ and thus, it holds that $conf((tp_{prec,i}, tp)) = \frac{supp(tp)}{supp(tp_{prec,i})} \geq \frac{supp(tp)}{supp(tp_{prec,i+1})} = conf((tp_{prec,i+1}, tp))$. Due to transitivity, the following prediction rules can also just have identical or smaller confidence values.

■

The algorithm for prediction rule generation is shown in Algorithm 18. The input parameters are the pattern that is to be processed and the minimal confidence *minconf*. Initially, the result set for the prediction rules is set to the empty set and the precondition pattern is set to the original input pattern p. Then, in a while loop in each step the last atomic pattern is removed from the previous precondition pattern as long as at least one atomic pattern is left as precondition. If the confidence of a prediction rule is greater than *minconf*, the pattern is added to the set of

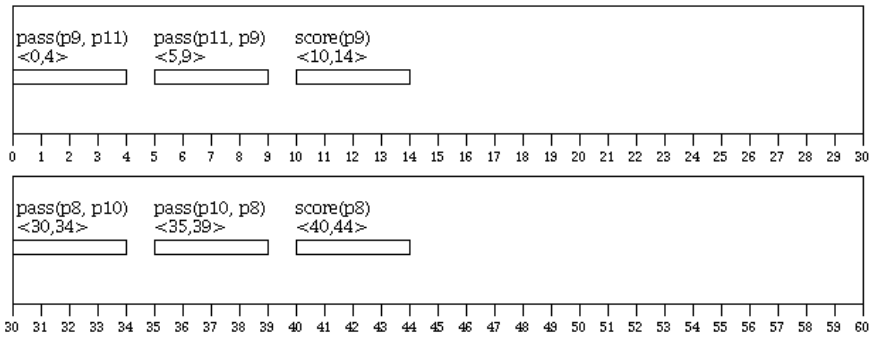

Figure 5.4: Example sequence for prediction rule generation

prediction rules. Otherwise, the while loop is left and the actual set of prediction rules for the given pattern is returned.

Example 5.1. For the dynamic scene in Fig. 5.4 the frequent pattern mining algorithm would find (among others) the temporal pattern $tp = ((pass(X,Y), pass(Y,X), score(X)), \{(1,2,before), (1,3,before), (2,3,before)\}, \{(X,object), (Y,object)\})$.

The two generated prediction rules are:

1. $pr_1 = (((pass(X,Y), pass(Y,X)), \{(1,2,before)\}, \{(X,object), (Y,object)\}), tp)$ (also written as $((pass(X,Y), pass(Y,X) \rightsquigarrow score(X)), \{(1,2,before), (1,3,before), (2,3,before)\}, \{(X,object), (Y,object)\}))$

2. $pr_2 = (((pass(X,Y)), \{\}, \{(X,object), (Y,object)\}), tp)$ (also written as $((pass(X,Y) \rightsquigarrow pass(Y,X), score(X)), \{(1,2,before), (1,3,before), (2,3,before)\}, \{(X,object), (Y,object)\}))$

5.2 Evaluation of Prediction Rules

So far, two measures have been introduced in order to determine which patterns and prediction rules are regarded as relevant. The minimal frequency threshold filters out all temporal patterns that do not occur often enough in the dynamic scene and the minimal confidence threshold takes care that just those prediction rules are kept where the consequence occurs frequently enough when the precondition is present. Support (or frequency) and confidence are two typical measures for frequent itemsets and association rules and have already been used for association rule mining from transaction databases [AIS93, AS94]. The number of identified rules can easily become too large to be investigated by experts and the identified rules might not be equally interesting (cf. the discussion in [Höp03, p.87]). Different researchers address this problem; good surveys for interestingness measures can be found, for instance, in the papers of Tan et al. [TKS02] and McGarry [McG05].

Interestingness measures can be classified into objective and subjective measures [McG05]. While subjective measures are based on subjective knowledge of the user objective, measures are usually based on statistics in order to identify relevant patterns. Referring to McGarry, the disadvantage of the subjective (user-driven) approach is "that it constrains the discovery process to seek only what the user can anticipate or hypothesize" [McG05, p. 58]. Objective measures are only based on "statistical strength or correlations" and thus might not be interesting to the user, for instance, if previously known strong rules are identified. Objective criteria for rule evaluation could be coverage, complexity, confidence, completeness; whereas unexpectedness, actionability, and novelty are mentioned by McGarry as subjective criteria [McG05, p. 44].

Tan et al. [TKS02] present an extensive survey of interesting measures for association patterns. In their work, they compare 21 interestingness measures and examine if certain properties are satisfied for these measures. Among many others they present their findings for mutual information, the J-measure, support, confidence, Piatetsky-Shapiro's, and Klösgen. They conclude that there is "no measure that is consistently better than others in all cases" and that in many cases there is a high correlation among the different measures [TKS02, p. 40].

In this work, we follow Höppner's decision to use the J-measure, which is – referring to Berthold and Hand [BH99] – a promising measure for rule evaluation (cf. [Höp03]). The J-measure has been introduced by Smyth and Goodman [SG91] in order to define the information content of a probabilistic rule. For a probabilistic rule of the form "If $Y = y$ then $X = x$ with probability p" the J-Measure is defined as [SG91]:

$$J(X;Y=y) = p(y) \cdot \underbrace{\left(p(x|y) \cdot log_2\left(\frac{p(x|y)}{p(x)}\right) + (1-p(x|y)) \cdot log_2\left(\frac{(1-p(x|y))}{(1-p(x))}\right)\right)}_{j(X;Y=y)}$$

Here, $j(X;Y=y)$ is the cross entropy between the a posteriori and a priori belief about X. The first part of the J-measure ($p(y)$) can be seen as "preference for generality or simplicity in our rules" [SG91, p. 164]. Thus, maximizing the J-Measure maximizes the "simplicity of hypothesis y, and goodness-of-fit between y and a perfect predictor of X". Referring to McGarry, the "J-measure is particularly suited to discriminating between the *prior* probability and the *posterior* probability of an event" [McG05, p. 55].

Transferring the J-measure to prediction rules for a prediction rule $pr = (tp_{prec}, tp_{cons})$ $p(y)$ is the frequency of the precondition tp_{prec}, $p(x)$ is the frequency of the consequence pattern tp_{cons}, and $p(x|y)$ is the confidence of pr. The information

content of a prediction rule can be computed by:

$$info(pr) = freq(tp_{prec}) \cdot (conf(pr) \cdot log_2 \left(\frac{conf(pr)}{freq(tp_{cons})} \right) + (1 - conf(pr)) \cdot log_2 \left(\frac{(1 - conf(pr))}{(1 - freq(tp_{cons}))} \right))$$

Example 5.2. Let the frequency of the precondition pattern tp_{prec} be $p(y) = freq(tp_{prec}) = 0.8$, the frequency of the consequence pattern tp_{cons} be $p(x) = freq(tp_{cons}) = 0.5$, and the confidence of the prediction rule be $p(x|y) = conf((tp_{prec}, tp_{cons})) = 0.7$. The information content of this prediction rule is then:

$$info((tp_{prec}, tp_{cons})) = 0.8 \cdot \left(0.7 \cdot log_2 \left(\frac{0.7}{0.5} \right) + 0.3 \cdot log_2 \left(\frac{0.3}{0.5} \right) \right) = 0.095$$

For the overall rating of a prediction rule $pr = (tp_{prec}, tp)$, we take into account three additional aspects (besides frequency, confidence, and information content): The size of the pattern, the specificity of the pattern, and a predicate preference measure. The relative size is computed by the fraction of the pattern size and the maximal pattern size, i.e.:

$$rel\_size(pr) = \frac{size(tp)}{max\_size}$$

Similarly, the relative specificity is computed by dividing the refinement level by the maximal refinement level:

$$rel\_spec(pr) = \frac{|status(tp)|}{max\_specifity}$$

The predicate preference is an evaluation measure of the used predicates in the pattern. For each predicate template a preference factor must be specified. The predicate preference is then computed by the sum of the preference values of the predicates in the pattern divided by the pattern size:

$$pred\_pref(pr) = \frac{1}{size(tp)} \sum_{i=1}^{n} pref(pt_i)$$

with $tp = ((ap_1, \ldots, ap_n), \mathcal{TR}, \mathcal{CR})$, $ap_i = (pt_i, p_{arg,i})$ and $pref(pt_i) \in [0, 1]$.

The overall rating of a prediction rule is computed by the weighted sum of the single evaluation aspects:

$$eval(pr) = \omega_{supp} \cdot supp(tp) + \omega_{conf} \cdot conf(pr) + \omega_{info} \cdot info(pr) + \omega_{size} \cdot rel\_size(pr) + \omega_{spec} \cdot rel\_spec(pr) + \omega_{pred} \cdot pred\_pref(pr)$$

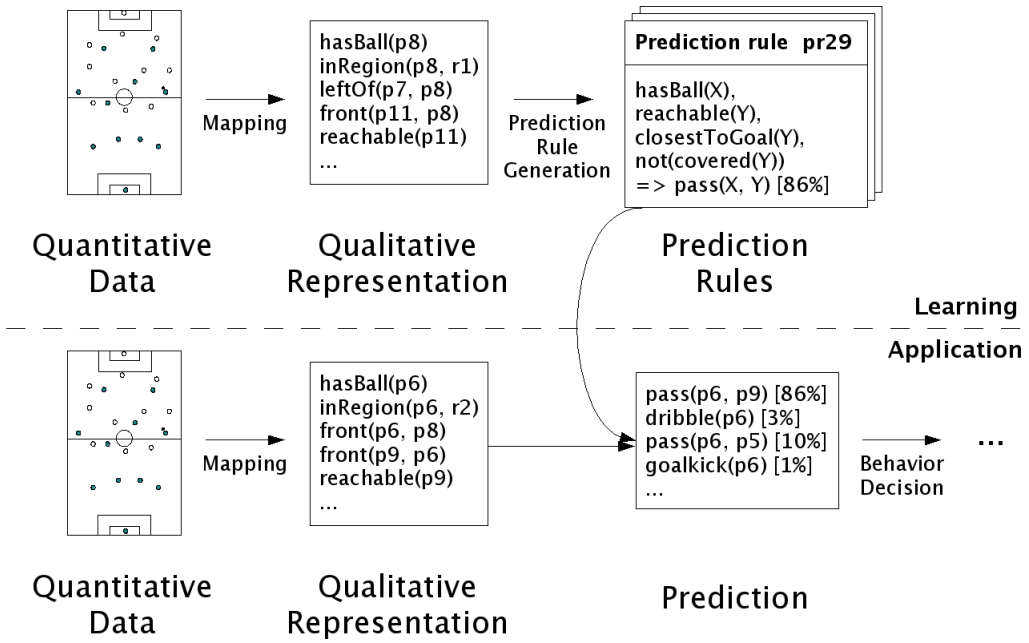

Figure 5.5: Application of prediction rules

The user can specify how strong the single aspects should be taken into account in the overall measure. If any of the weights is set to zero, the corresponding measure has no influence to the overall evaluation value. Of course it is possible to extend or adapt the evaluation measure easily, for instance, by replacing the J-Measure by some other interestingness measure. The different weights allow for adapting the evaluation measure to individual preferences for pattern mining tasks. It is possible, for instance, to assign different weights to the distinct predicate templates so that patterns with more interesting predicates are preferred. The evaluation measure can be used to rank the created prediction rules, i.e., the most interesting prediction rules could, e.g., be presented to the user first.

5.3 Application of Prediction Rules

It has been shown how relevant prediction rules can be generated from frequent temporal patterns and how the quality of the generated rules can be estimated so that it is possible to rank the rules w.r.t. the expected relevance to the user. This section shows how the prediction rules can be used.

Fig. 5.5 illustrates the principle process of generating and applying prediction rules. In both phases – learning as well as application – a qualitative representation of the dynamic scene has to be generated. In the learning phase the prediction rules are generated (as described above). In this phase, a selection of relevant prediction

rules has to be performed. Note, that it is also possible to change or extend the set of prediction rules (or even to set up prediction rules manually from scratch without learning). In the application phase, the stored prediction rules are checked and if the precondition of a prediction rule matches the consequence is identified as potential future part of the dynamic scene (provided with a confidence value which estimates the probability that the prediction might be correct).

As mentioned above, prediction rules consist of a precondition pattern and a consequence pattern. The confidence of a prediction rule is the observed frequency of window positions where the precondition pattern is followed by the consequence part. In order to apply a prediction rule, it has to be checked if the precondition pattern matches in a certain situation. If a match was found, it is expected that the consequence of the pattern also occurs (with a certain probability) in the current sliding window (or at least begins within the temporal distance of the sliding window width in the case of the modified rule semantics). If the precondition and the consequence share common variables, it is expected that they are bound to the same object.

Chapter 6

Evaluation

This chapter presents a detailed evaluation of the concepts developed in the previous chapters. The goals of the evaluation are manifold. Section 6.1 presents a simple example that can also be checked "manually" in order to see if the implementation leads to the expected results. In the following Section 6.2, different variations of parameters are tested in order to get an estimation of practical complexity w.r.t. the number of generated patterns. Section 6.3 compares the frequent pattern mining approaches *WARMR* and *MiTemP* which have been presented in Chapter 4 in order to see if there is an advantage of the new approach in comparison to *WARMR*. Finally, some experiments for generating and applying prediction rules in the RoboCup domain are presented in Section 6.4. All files with the learning input and output as well as console logs of the different experiments can be found on the DVD provided with this thesis. The content of the DVD is described in Appendix B.

The implementation of *MiTemP* has been realized with *XSB Prolog* [SSW+06] – a freely available logic programming and deductive database system developed at the Computer Science Department of the Stony Brook University, New York, USA, hosted by Sourceforge[1]. Among others, *XSB* provides the features of tabled resolution and indexing techniques for efficient access to predicates. The implementation of *MiTemP* features three methods for reducing the number of generated patterns:

1. If certain refinement types are not important, it is possible to disable them. Each refinement type can be disabled separately. However, lengthening is mandatory as it is the only way to add atomic patterns to the conjunctive pattern.

2. A bias for the relevant patterns to be mined can be defined by a disjunctive list of sub-patterns. Only those patterns are taken into account that are still compliant with at least one of the sub-patterns specified in the bias. It is

[1] `http://xsb.sourceforge.net/`

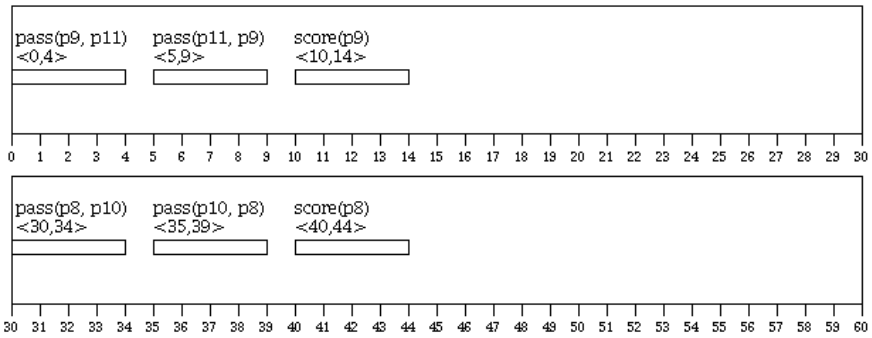

Figure 6.1: Illustration of the simple example

possible, e.g., to restrict the refinement to those patterns that are subsumed by a pattern with the conjunctive pattern {`pass(X, Y), score(Y)`}. It is also allowed to use constants in the bias, e.g., {`pass(X, klose), score(klose)`}.

3. The third option to reduce the number of patterns is a random selection of a maximum number of patterns to be refined for each refinement level. Only n of the frequent patterns of the previous level are refined. Using this option, not all – but a subset of the – frequent patterns of a dynamic scene are found. If the number of frequent patterns in the previous level is below n, all patterns are refined.

All experiments have been performed on an AMD Athlon 64 4000+ 2.4 GHz PC (1024 MB Cache, FSB 2000 MHz, 3 GB main memory, average hard disk read seek time of 8.9 ms, and average hard disk latency of 4.17 ms[2]) with SuSE Linux 10.0 (64 Bit) as the operating system.

6.1 A Simple Example

In order to illustrate the implementation of our approach, we re-use the example of Section 5.4 which is shown again in Fig. 6.1. The dynamic scene schema consists of two predicate templates (*pass* and *score*) and the concept "hierarchy" consists of the single most general concept *object*. The *pass* can be performed between two *objects* and the *score* predicates just take one *object* as argument. The dynamic scene consists of six predicates with the four instances – *p8, p9, p10*, and *p11* – as arguments. The dynamic scene is set up in a way that the pattern *pass(X, Y) before pass(Y, X) before score(X)* occurs twice (Fig. 6.1).

[2]Specification of the Samsung SP1614N hard drive at `http://www.samsung.com`.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% MiTemP parameters
mitempParam(windowSize, 10).
mitempParam(minFrequency, 0.1).
mitempParam(minConfidence, 0.8).
mitempParam(minPatternSize, 1).
mitempParam(maxPatternSize, 3).
mitempParam(maxLevel, 14).

mitempParam(wFreq, 1.0).
mitempParam(wConf, 1.0).
mitempParam(wInfo, 1.0).
mitempParam(wRelSize, 1.0).
mitempParam(wRelSpec, 1.0).
mitempParam(wPredPref, 1.0).

mitempPredPref(pass, 0.7).
mitempPredPref(score, 1.0).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Predicate templates

predicate(pass(object, object)).
predicate(score(object)).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Example description of a dynamic scene

directInstanceOf(p8, object).
directInstanceOf(p9, object).
directInstanceOf(p10, object).
directInstanceOf(p11, object).

holds(pass(p9, p11), 0, 4).
holds(pass(p11, p9), 5, 9).
holds(score(p9), 10, 14).
holds(pass(p8, p10), 30, 34).
holds(pass(p10, p8), 35, 39).
holds(score(p8), 40, 44).
```

Figure 6.2: Simple example input file: simpleExample.P

The input file for the learning program is shown in Fig. 6.2. The first part of the input file consists of a number of parameters for learning: the size of the sliding window, the minimal frequency and confidence, the minimal and maximal pattern size (i.e., number of atomic patterns in the conjunctive pattern), and the maximal refinement level. The different weights for prediction rule evaluation are set to 1.0 and the *score* predicate is set to be more important than the *pass* predicate (1.0 vs. 0.7). The two predicate templates define which predicates can be used while pattern mining and what the arguments' concepts must be. The `directInstanceOf` entries define the objects and the `holds` predicates describe the fluents in the scene.

In the first step – the frequent pattern generation phase – the program creates the pattern candidates and computes their frequencies level-wise. Listing the complete output of the generated patterns in this thesis would take too much space. Thus,

```
pattern(1,[score(_h211652)],temp,conceptRestr(object)).
pattern(2,[pass(_h211691,_h211692)],temp,conceptRestr(object,object)).
pattern(3,[score(_h211732),score(_h211736)],temp(tr([before,olderContemp,headToHead])),
          conceptRestr(object,object)).
pattern(4,[score(_h211786),pass(_h211790,_h211791)],temp(tr([before,olderContemp])),
          conceptRestr(object,object,object)).
pattern(5,[pass(_h211840,_h211841),score(_h211845)],temp(tr([before,olderContemp,headToHead])),
          conceptRestr(object,object,object)).
pattern(6,[pass(_h211896,_h211897),pass(_h211901,_h211902)],temp(tr([before,olderContemp,
          headToHead])),conceptRestr(object,object,object,object)).
...
pattern(418,[pass(_h235926,p9),pass(p9,_h235926),score(_h235926)],temp(tr([before],[before]),
          tr([before])),conceptRestr(object,object,object,object,object)).
pattern(419,[pass(_h235993,p10),pass(p10,_h235993),score(_h235993)],temp(tr([before],[before]),
          tr([before])),conceptRestr(object,object,object,object,object)).
pattern(420,[pass(_h236060,p11),pass(p11,_h236060),score(_h236060)],temp(tr([before],[before]),
          tr([before])),conceptRestr(object,object,object,object,object)).
```

Figure 6.3: Simple example created patterns: createdPatterns.P

only a snippet is presented in Fig. 6.3. The complete output can be found on the DVD in the file `/eval/simpleExample/createdPatterns.P`. The patterns shown in Fig. 6.3 also include the infrequent ones (all created patterns are shown). As it can be seen in the output file, the expected patterns are created by the program. It starts with all the single predicate patterns (`score` and `pass`), then in the next step creates the level two patterns, i.e., those with size two ({`score, score`}, {`score, pass`}, {`pass, score`}, {`pass, pass`}) and also the unification patterns (for `pass` only as `score` is an unary predicate) and instantiation patterns where variables are bound to instances. Temporal refinement cannot be applied in refinement step two as there have only been single predicate patterns in the previous step and consequently, there are no interval relations to be restricted. Concept refinement cannot be applied to the whole example as there is exactly one concept in the dynamic scene schema, namely `object`.

In the output, the pattern with ID 5, for instance, consists of a conjunctive pattern with a *pass* and a *score* predicate. The temporal relation between these atomic patterns is not restricted (i.e., it could be any of {before, olderContemp, headToHead}) and the concept restriction is set to the concept `object` in all cases.

In the second step, for each frequent pattern all prediction rules are generated (and only those with a confidence above the minimum threshold are kept). Once again, due to the large number of prediction rules, only a part of the generated prediction rules is shown in Fig. 6.4. The prediction rules are ordered by the overall evaluation measure starting from the prediction rule with the worst rate. For each prediction rule, besides the rule itself and the temporal and concept restrictions, the different evaluation values are computed. The following abbreviations are used in the output:

- **f:** frequency;

```
=====
[pass(_h611890,p10)] => [pass(_h611909,_h611910)]
temp(tr([before]))
conceptRestr(object,object,object,object)

Eval: 0.4791
(f: 0.1852, c: 1.0000, j: 0.0368, s: 0.6667, r: 0.2857, p: 0.7000)
=====
[pass(_h611993,p11)] => [pass(_h612012,_h612013)]
temp(tr([before]))
conceptRestr(object,object,object,object)

Eval: 0.4791
(f: 0.1852, c: 1.0000, j: 0.0368, s: 0.6667, r: 0.2857, p: 0.7000)
=====
...
=====
[pass(_h257737,_h257738),pass(_h257738,_h257743)] => [score(_h257737)]
temp(tr([before],[before]),tr([before]))
conceptRestr(object,object,object,object,object)

Eval: 0.6155
(f: 0.1852, c: 1.0000, j: 0.1362, s: 1.0000, r: 0.5714, p: 0.8000)
=====
[pass(_h257866,_h257867),pass(_h257871,_h257866)] => [score(_h257866)]
temp(tr([before],[before]),tr([before]))
conceptRestr(object,object,object,object,object)

Eval: 0.6155
(f: 0.1852, c: 1.0000, j: 0.1362, s: 1.0000, r: 0.5714, p: 0.8000)
=====
[pass(_h258201,_h258202),pass(_h258202,_h258201)] => [score(_h258201)]
temp(tr([before],[before]),tr([before]))
conceptRestr(object,object,object,object,object)

Eval: 0.6274
(f: 0.1852, c: 1.0000, j: 0.1362, s: 1.0000, r: 0.6429, p: 0.8000)
```

Figure 6.4: Simple example created prediction rules: testrun_output.txt (snippet)

- **c:** confidence;
- **j:** J-measure;
- **s:** relative size;
- **r:** relative specifity (refinement level);
- **p:** predicate preference

The overall evaluation value is shown above these single values.

The last prediction rule has received the highest overall evaluation value of 0.6274. It is a rule that can be easily recognized when taking a look to the dynamic scene (Fig. 6.1): If there is a pass from X to Y followed by a pass from Y to X, object X will score. The temporal relations between all these atomic patters are

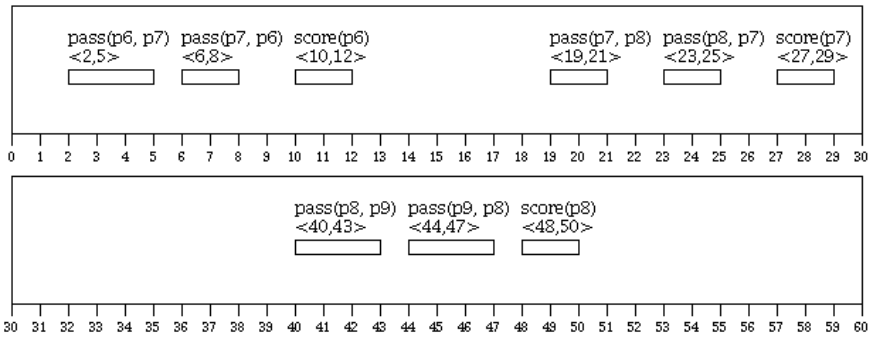

Figure 6.5: Simple example: Test scene for rule application

before relations, and the concepts of the variables are *objects* in all cases. The frequency of this prediction rule (or its corresponding pattern) is 0.1852, the confidence is 1.0, the J-measure value is 0.1362, the relative size is 1.0, the relative specifity is 0.6429, and the predicate preference is 0.8. This prediction rule corresponds to the intended rule that was set up in the simple example input file.

In order to test the prediction, a test scene is set up which is shown in Fig. 6.5. The intended test rule appears three times in the sequence (with different interval times and object identifiers used in the predicates). From the learned prediction rules, the best one is chosen (i.e., the last one in Fig. 6.4). The output of the application of the prediction rule is shown in Fig. 6.6. The output consists of the prediction rule (internal representation and a more readable format), the prediction intervals, and the prediction quality for the test scene. The prediction rule says that if a pass is performed from an player to another one and then the other way round, the initial player will score. The prediction intervals are those intervals where the precondition pattern has matched. The prediction quality is the relation of positions where precondition and consequence appear together to all the positions where the precondition is observable. In this example, the precondition matches in the intervals $\langle 6, 15\rangle$, $\langle 23, 31\rangle$, and $\langle 44, 53\rangle$. In all these cases, the consequence can also be observed (within the sliding window or at least starting within the sliding window) thus, the quality is 1.0, i.e., 100%. With this example, it has been shown that the pattern mining and prediction rule generation implementations work as expected.

6.2 Experiments with Synthetic Data

The aim of the experiments with synthetic data is to get some insights about the practical complexity of the learning approach. The learning program is applied to a number of artificially generated dynamic scenes with varying parameters in order to

```
Prediction quality for dynPredRule(0.6274,0.1362,1.0000,0.1852,1.0000,0.6429,0.8000,
  pattern(id,[pass(_h15366,_h15367),pass(_h15367,_h15366)],temp(tr([before])),
          conceptRestr(object,object,object,object),status),
  pattern(364,[pass(_h15366,_h15367),pass(_h15367,_h15366),score(_h15366)],
          temp(tr([before],[before]),tr([before])),conceptRestr(object,object,object,object,object),
          status(3,3,temp(2,3),3,varPos(3,1),0,varPos(-1,-1),0,varPos(-1,-1))))

[pass(_h15366,_h15367),pass(_h15367,_h15366)] => [score(_h15366)]
temp(tr([before],[before]),tr([before]))
conceptRestr(object,object,object,object,object)

Eval: 0.6274
(f: 0.1852, c: 1.0000, j: 0.1362, s: 1.0000, r: 0.6429, p: 0.8000)

Precondition matches at: [i(6,15),i(23,31),i(44,53)]
Quality => 1.0000
```

Figure 6.6: Simple example rule application: testrun_output_test.txt

investigate the complexity behavior w.r.t. these parameters. In order to generate the synthetic data, a test set generator, namely *TestDataCreator*, has been developed. This program takes a number of parameters as input and creates a random test set compliant with the parameters. Among others, the parameters are: the number of predicates in the scene, the number of predicate templates, the number of concepts, the number of instances, the size of the sliding window and the average number of concurrent predicates within a sliding window. Seven parameters have been chosen in order to see how the learning algorithm works with varying values:

- Minimal frequency: $[0.1, 0.15, 0.2, 0.25, 0.3, 0.35, 0.4]$
- Number of concepts: $[1, 2, 3, 4, 5, 6, 7]$
- Number of instances: $[2, 3, 4, 5, 6, 7, 8]$
- Maximal pattern size: $[2, 3, 4, 5, 6, 7, 8]$
- Number of predicate templates: $[2, 3, 4, 5, 6, 7, 8]$
- Number of predicates (sequence length): $[100, 150, 200, 250, 300, 350, 400]$
- Window size: $[10, 15, 20, 25, 30, 35, 40]$

In all experiments, the remaining parameters are left unchanged in almost all cases. The default settings for the parameters are:

- Minimal frequency: 0.3
- Number of concepts: 1

- Number of instances: 3
- Minimal and maximal pattern size: 3
- Number of predicate templates: 3
- Number of predicates (sequence length): 100
- Window size: 10
- Maximal refinement level: 10

The idea behind these values is to keep the default setting simple for all values except the one to be evaluated. In almost all experiments, only one value differs from these default settings. In the experiment with varying numbers of concepts, the number instances has been set to the maximal occurring value of concepts, i.e., seven instead of three as in the other experiments. For each of the seven experiments the number of created patterns, the number of frequent patterns, and the CPU time for a run have been recorded. Each experiment has been repeated ten times with randomly generated dynamic scenes characterized by the properties mentioned above. The average of the ten runs has been computed for each value of the analyzed value. The results are shown in Fig. 6.7 - 6.20. Altogether 490 test runs have been performed in this part of the evaluation (7 parameters $\times$ 7 values $\times$ 10 runs).

In the first experiment, the minimal frequency parameter is changed for different dynamic scenes (Fig. 6.7 and 6.8). The graphs show that the number of generated patterns and the consumed CPU time decreases rapidly with increasing minimal frequency values. A quite low number of frequent patterns is reached quickly; the values almost do not change after the minimal frequency threshold of 0.2. This result is compliant with the expectations: The higher the minimal frequency threshold, the less frequent patterns are found. This has a direct influence on the number of created patterns, which also decreases.

The result of the experiment with a varying number of concepts seems counter-intuitive at the first glance (Fig. 6.9 and 6.10). The number of created patterns decreases with an increasing number of concepts. The variety grows if more concepts are present, but in the developed approach, instantiation refinements are only done for the direct instances of a variable's concept in the concept restriction. Consequently, if more concepts are present (and instances might be direct instances of concepts in the lower parts of the concept hierarchy), there are less instantiation refinements possible until concept refinements have been created, i.e., the branching factor is lower. This leads to a decreasing number of patterns. The number of frequent patterns stays at a stable level as the actual frequent patterns in the data are the same (but with different concepts). The increasing CPU time can be explained

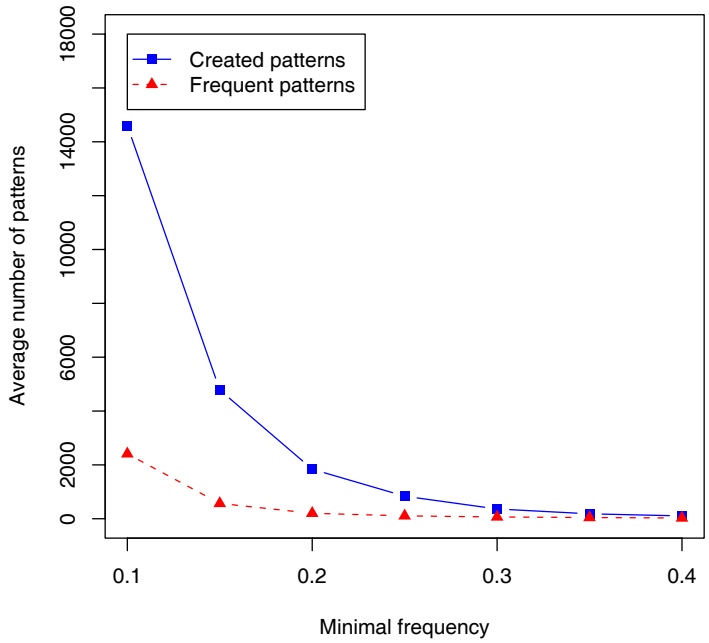

Figure 6.7: Number of patterns for varying minimal frequencies

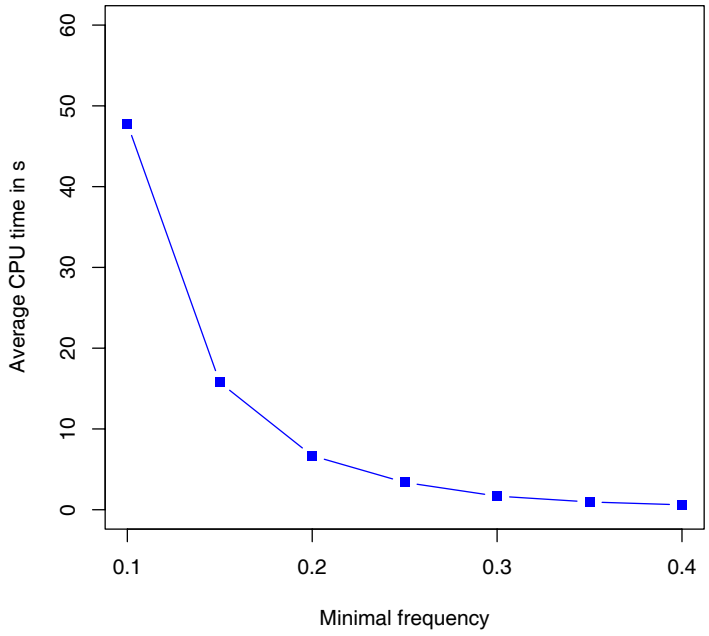

Figure 6.8: CPU time for varying minimal frequencies

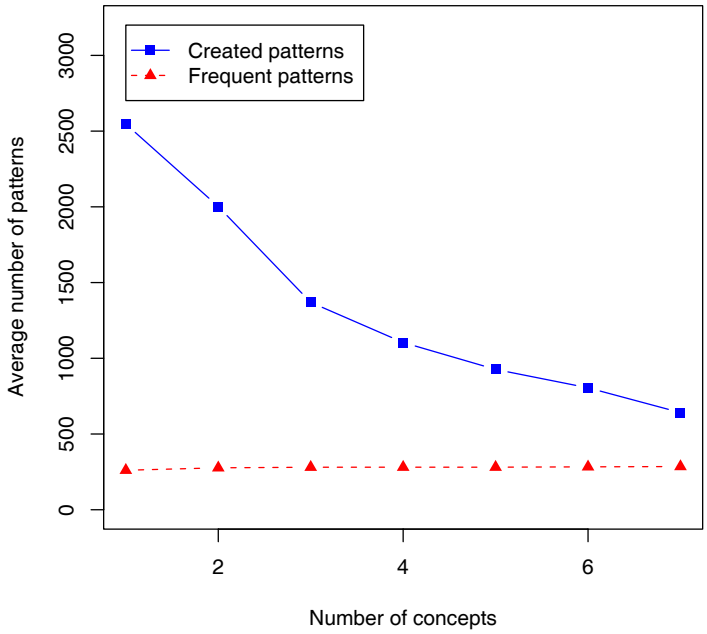

Figure 6.9: Number of patterns for varying number of concepts

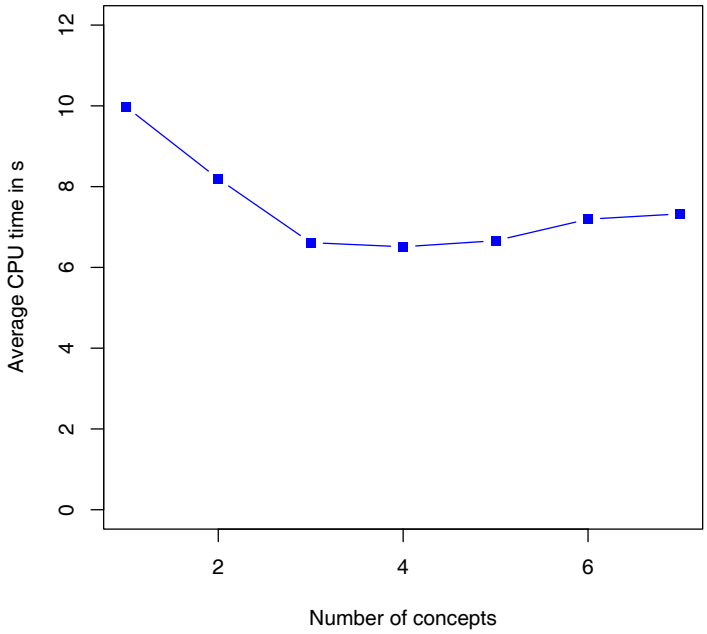

Figure 6.10: CPU time for varying number of concepts

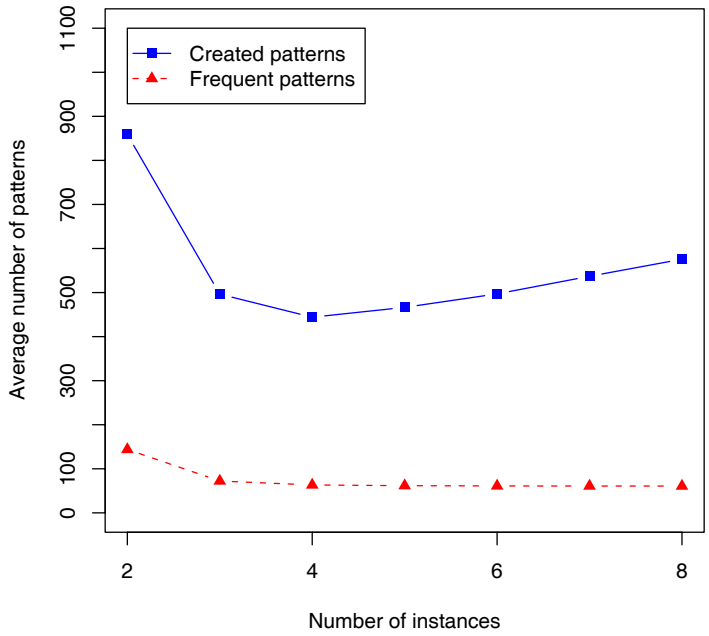

Figure 6.11: Number of patterns for varying number of instances

by higher efforts in checking instance-of relations with longer paths in the concept hierarchy.

In the case of the increasing number of instances (Fig. 6.11 and 6.12), another phenomenon can be observed: The number of created patterns first decreases and than increases w.r.t. an increasing number of instances. This can be explained by two interfering situations. On the one hand, an increasing number of instances leads to a decreasing number of frequent patterns as in the artificially generated data more instances are chosen randomly and thus, patterns with identical instances do not occur that often. On the other hand, the number of potential patterns grows with an increasing number of instances. Thus, more patterns are created when the instantiation of variables comes into play. The CPU time correlates to the number of created patterns.

The biggest changes for the varying number of maximal pattern sizes are between the values 2 and 4 (Fig. 6.13 and 6.14). Then the number of created and frequent patterns does not change any more. The reason is that the created dynamic scenes do not have any frequent patterns with more atomic patterns involved. The values in the beginning of the graph show that a very fast growth is possible with increasing pattern sizes as long as frequent patterns of this size occur in the scene. The CPU times also stay constant when the number of created patterns does not increase

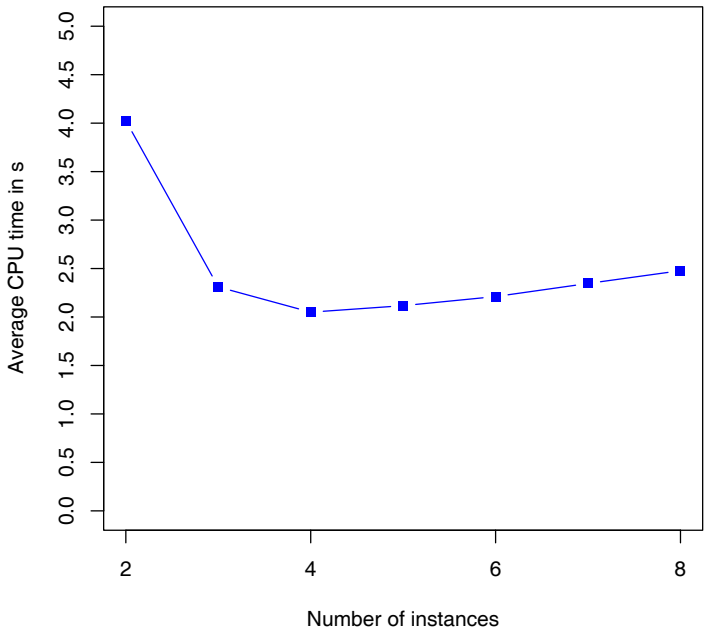

Figure 6.12: CPU time for varying number of instances

anymore.

The situation with a varying number of predicate templates (Fig. 6.15 and 6.16) is similar to the varying number of instances. An increase in the number of predicate templates leads (rather counter-intuitively) to a decrease in the number of generated patterns. The reason is the same: If the number of randomly generated predicates is split up on different predicate templates, the number of frequent patterns decreases as it is the case in the experiments here. The CPU time also correlates to the number of generated patterns.

If the number of predicates (i.e., the sequence lengths) is increased without changing the remaining parameters, the expectation is to have a constant number of created patterns and frequent patterns while the CPU time increases linearly as longer sequences have to be checked for support computation. The result shows that the latter two expectations can be observed in this experiment (Fig. 6.17 and 6.18). The number of created patterns decreases in the beginning of the experiments. The maximal possible support is lower for shorter sequences. An increase of 50 in the number of predicates can thus lead to quite big effects w.r.t. to the frequency of patterns. For the greater values (250 and above), a more stable behavior w.r.t. to the generated patterns can be seen.

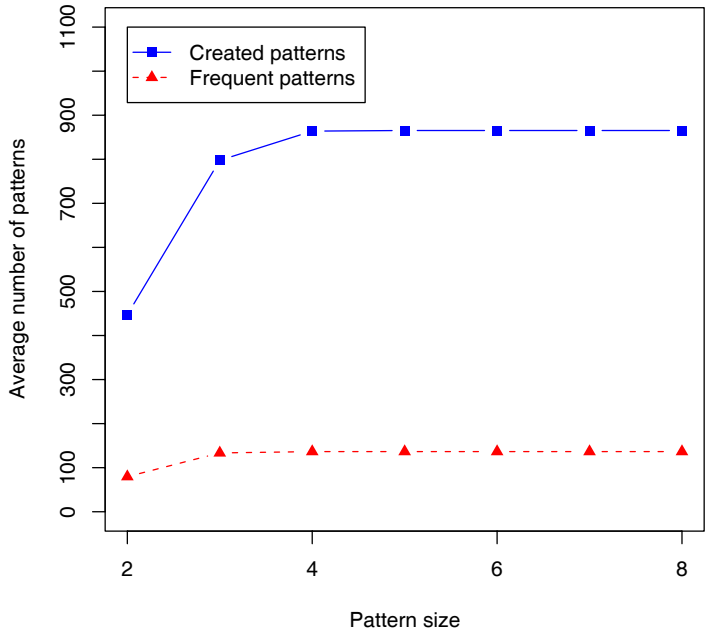

Figure 6.13: Number of patterns for varying pattern sizes

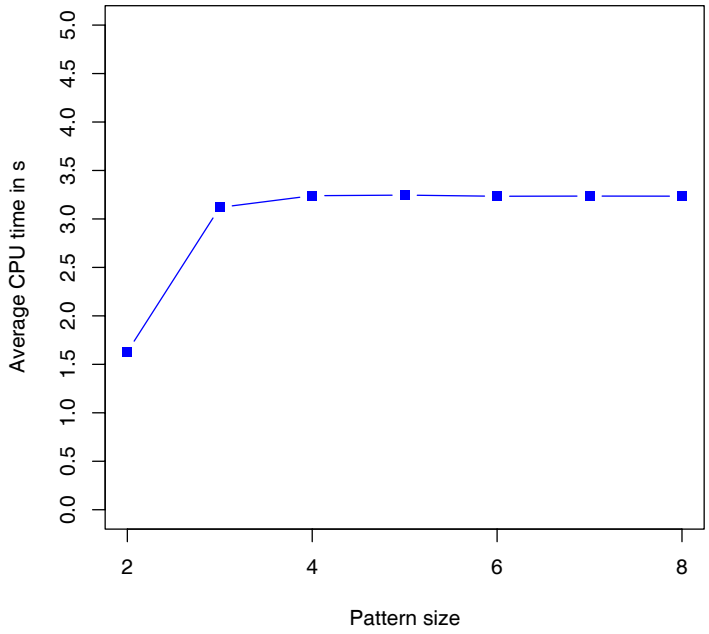

Figure 6.14: CPU time for varying pattern sizes

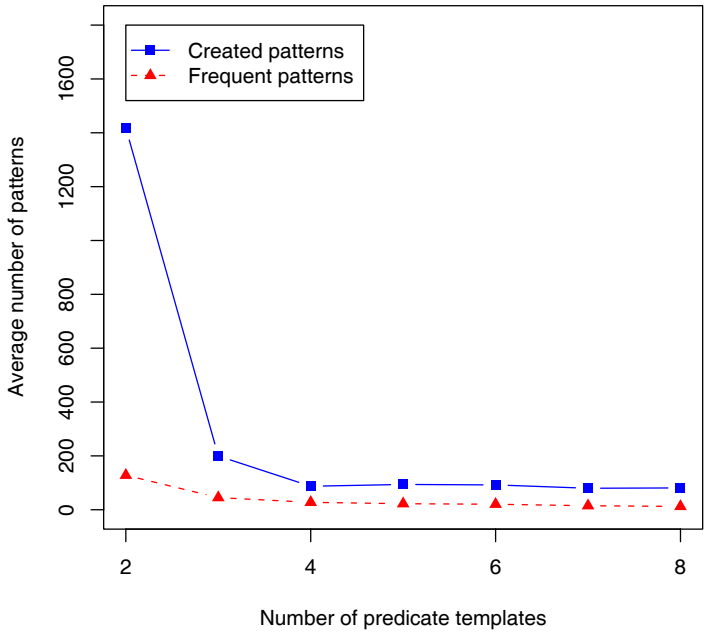

Figure 6.15: Number of patterns for varying numbers of predicate templates

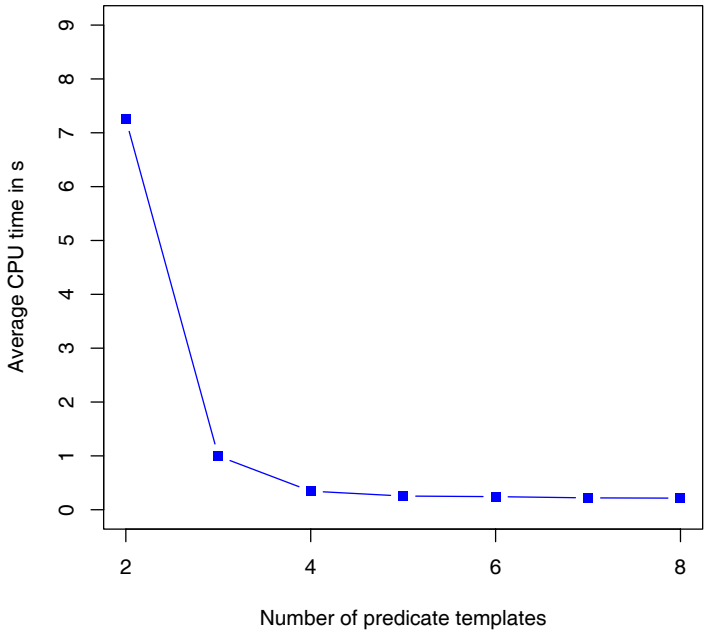

Figure 6.16: CPU time for varying numbers of predicate templates

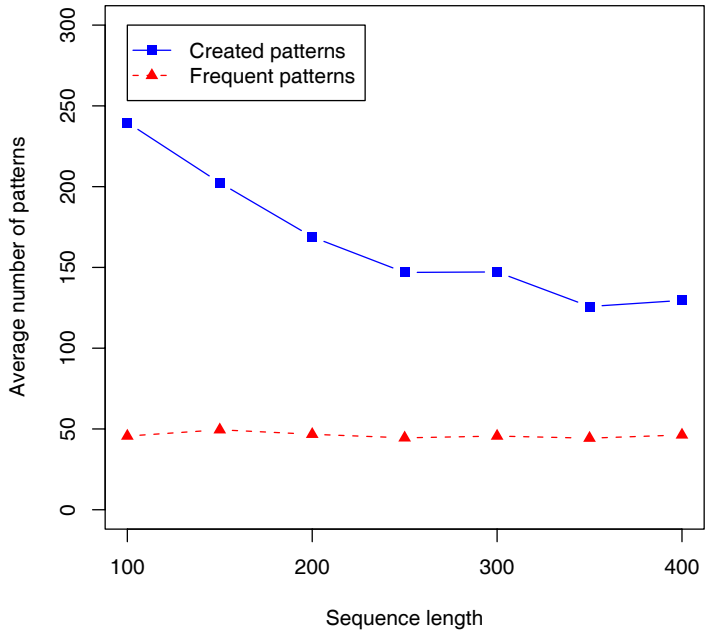

Figure 6.17: Number of patterns for varying sequence lengths

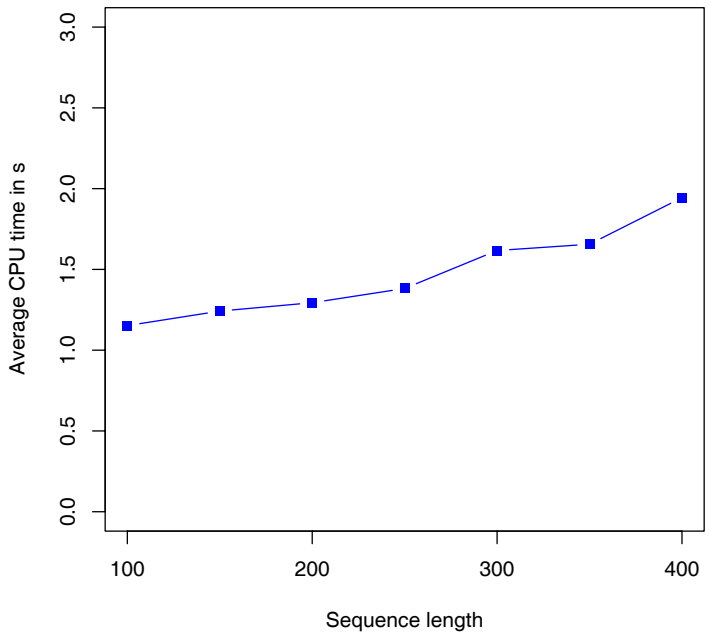

Figure 6.18: CPU time for varying sequence lengths

An increasing window size leads to a greater number of predicates observable within the sliding window. The consequence is that the number of frequent patterns should increase. This can be seen in the results of the experiments where the window size is increased from 10 up to 40 with step size 5 (Fig. 6.19 and 6.20). The complexity evaluation in Section 4.5.4 on page 132 has identified the number of predicates to be checked within a sliding window position as the crucial part which is supported by the graphs. The growth of the frequent patterns also leads to a growth of the number of generated patterns as it can be seen in Fig. 6.19.

The input and output files of the experiments with the synthetic data can be found in the folder `/eval/synthetic` on the DVD provided with this thesis.

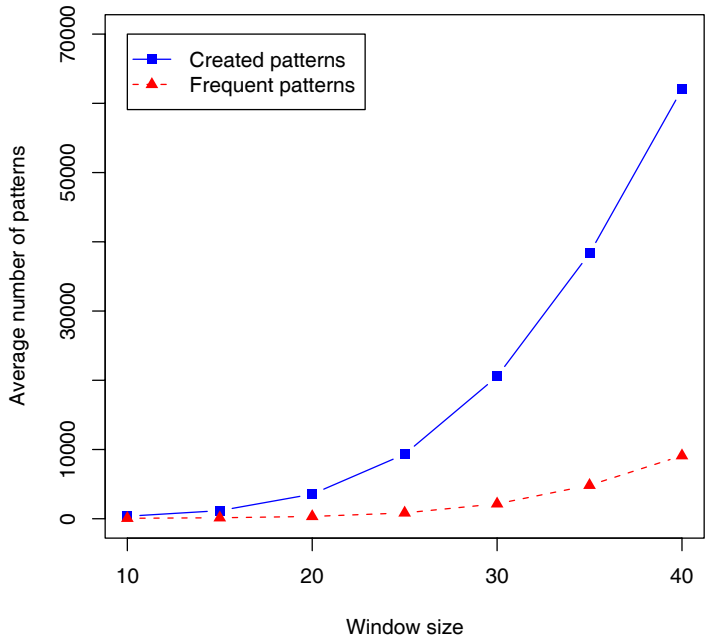

Figure 6.19: Number of patterns for varying window sizes

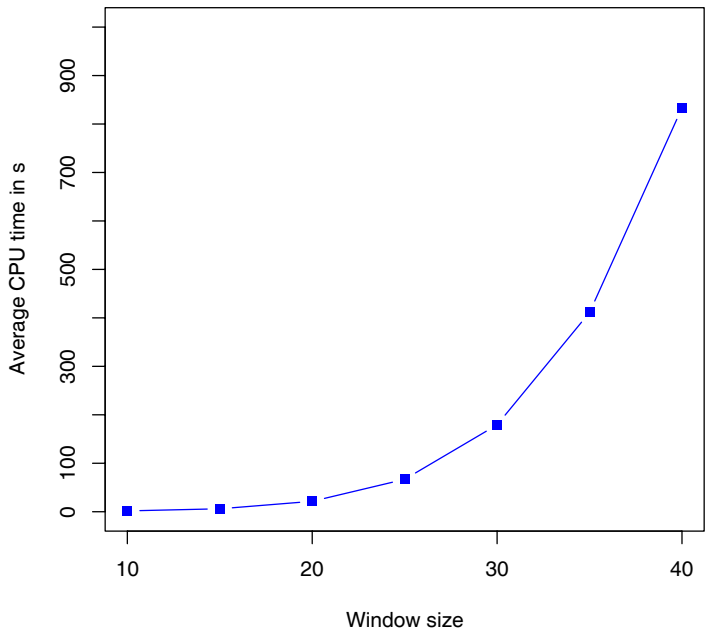

Figure 6.20: CPU time for varying window sizes

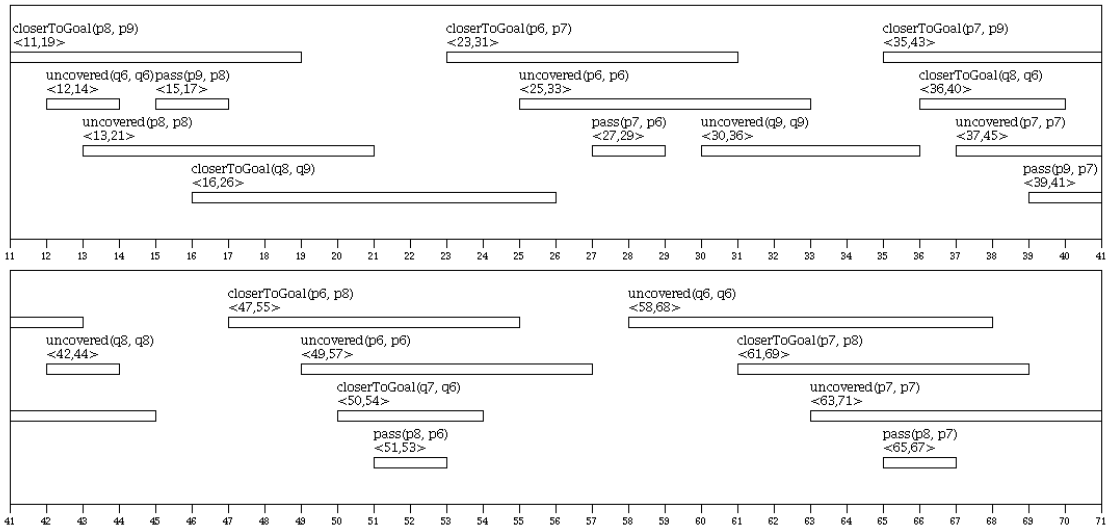

Figure 6.21: Test scene for *WARMR* comparison

6.3 Comparison of WARMR and MiTemP

In Section 4.6, it has been shown how *WARMR* can also be used for mining temporal patterns as intended in this thesis. The goal of this evaluation part is to check whether *MiTemP* and *WARMR* generate identical patterns (with identical frequency values) and to compare *MiTemP* and *WARMR* w.r.t. the number of generated patterns. Preliminary results of this comparison have been presented in [LH06].

For the comparison of the two approaches, the converter *miTemP2warmr* was developed (written in Perl). It takes a *MiTemP* input file and creates *ACE* input files so that the *WARMR* implementation in the *ACE* system can be used for pattern mining [BDD+02, BDR+06]. For the evaluation, the scene shown in Fig. 6.21 was used (cf. Example 4.16). The *MiTemP* input file is shown in Fig. 6.22. The files created by the converter are shown in Fig. 6.23 (settings), 6.24 (knowledge base), and 6.25 (background knowledge). The experiment was performed with different parameters for the maximal refinement level varying from two to seven; the input file snippets shown in Fig. 6.23 - 6.25 illustrate the input for the maximal refinement level of four. In all experiments, the following settings are used: a sliding window size of twelve, a minimal frequency of 0.2, and a minimal pattern size of one. The maximal pattern size corresponds to the maximal refinement level in all cases. All input as well as the output files of these experiments can be found in the folder `/eval/warmrComparison` on the DVD.

```
% MiTemP parameters
mitempParam(windowSize, 12).
mitempParam(minFrequency, 0.2).
mitempParam(minPatternSize, 1).
mitempParam(maxPatternSize, 4).
mitempParam(maxLevel, 4).

% Predicate templates
predicate(closerToGoal(object,object)).
predicate(pass(object,object)).
predicate(uncovered(object,object)).

% Concept hierarchy
isA(team1, object).
isA(team2, object).

% Dynamic scene
directInstanceOf(p6, team1).
directInstanceOf(p7, team1).
directInstanceOf(p8, team1).
directInstanceOf(p9, team1).
directInstanceOf(q6, team2).
directInstanceOf(q7, team2).
directInstanceOf(q8, team2).
directInstanceOf(q9, team2).

holds(uncovered(q6, q6), 12, 14).
holds(pass(p9, p8), 15, 17).
holds(closerToGoal(p8, p9), 11, 19).
holds(uncovered(p8, p8), 13, 21).
holds(closerToGoal(q8, q9), 16, 26).
holds(pass(p7, p6), 27, 29).
holds(closerToGoal(p6, p7), 23, 31).
holds(uncovered(p6, p6), 25, 33).
holds(uncovered(q9, q9), 30, 36).
holds(closerToGoal(q8, q6), 36, 40).
holds(pass(p9, p7), 39, 41).
holds(closerToGoal(p7, p9), 35, 43).
holds(uncovered(q8, q8), 42, 44).
holds(uncovered(p7, p7), 37, 45).
holds(pass(p8, p6), 51, 53).
holds(closerToGoal(q7, q6), 50, 54).
holds(closerToGoal(p6, p8), 47, 55).
holds(uncovered(p6, p6), 49, 57).
holds(pass(p8, p7), 65, 67).
holds(uncovered(q6, q6), 58, 68).
holds(closerToGoal(p7, p8), 61, 69).
holds(uncovered(p7, p7), 63, 71).
```

Figure 6.22: *WARMR* comparison input file: example_small.P

```
typed_language(yes).
type(currentIndex(index)).
type(before(interval, interval)).
type(olderContemp(interval, interval)).
type(headToHead(interval, interval)).

type(unif(object, object)).
rmode(unif(+X, +Y)).
constraint(unif(X,Y), not_occurs(unif(_,X))).
constraint(unif(X,Y), X\==Y).

type(closerToGoal(id, object, object, interval, index)).
rmode(closerToGoal(\Id, -X0, -X1, -I, +Index)).
type(useful_constant_closerToGoal_O0(_)).
type(eq_obj_closerToGoal_O0(object, _)).
rmode(#(C: useful_constant_closerToGoal_O0(C), eq_obj_closerToGoal_O0(+X, C))).
constraint(eq_obj_closerToGoal_O0(X, Y), occurs(closerToGoal(_,X,_,_,_))).
constraint(unif(_,Y), not_occurs(eq_obj_closerToGoal_O0(Y, _))).
constraint(instanceOf(X,_), not_occurs(eq_obj_closerToGoal_O0(X, _))).
constraint(eq_obj_closerToGoal_O0(Y, _), not_occurs(unif(_,Y))).
constraint(eq_obj_closerToGoal_O0(Y, _), not_occurs(instanceOf(Y,_))).

type(useful_constant_closerToGoal_O1(_)).
type(eq_obj_closerToGoal_O1(object, _)).
rmode(#(C: useful_constant_closerToGoal_O1(C), eq_obj_closerToGoal_O1(+X, C))).
constraint(eq_obj_closerToGoal_O1(X, Y), occurs(closerToGoal(_,_,X,_,_))).
constraint(unif(_,Y), not_occurs(eq_obj_closerToGoal_O1(Y, _))).
constraint(instanceOf(X,_), not_occurs(eq_obj_closerToGoal_O1(X, _))).
constraint(eq_obj_closerToGoal_O1(Y, _), not_occurs(unif(_,Y))).
constraint(eq_obj_closerToGoal_O1(Y, _), not_occurs(instanceOf(Y,_))).

type(pass(id, object, object, interval, index)).
(...)
type(uncovered(id, object, object, interval, index)).
(...)

rmode_key(currentIndex(I)).
root(currentIndex(I)).
rmode(before(+I1, +I2)).
rmode(olderContemp(+I1, +I2)).
rmode(headToHead(+I1, +I2)).
constraint(headToHead(X,Y), not(X=Y)).

type(concept(_)).
type(instanceOf(object, _)).
rmode(#(C: concept(C), instanceOf(+X, C))).

constraint(instanceOf(X,Y), not_occurs(instanceOf(X,_))).
constraint(instanceOf(X,Y), not_occurs(unif(_,X))).
constraint(unif(_,X), not_occurs(instanceOf(X,Y))).
minfreq(0.2).
warmr_maxdepth(4).
```

Figure 6.23: *WARMR* settings file experiment.s

```
isA(team1, object).
isA(team2, object).
directInstanceOf(p6, team1).
directInstanceOf(p7, team1).
directInstanceOf(p8, team1).
directInstanceOf(p9, team1).
directInstanceOf(q6, team2).
directInstanceOf(q7, team2).
directInstanceOf(q8, team2).
directInstanceOf(q9, team2).
uncovered(1, q6, q6, i(12, 14)).
pass(2, p9, p8, i(15, 17)).
closerToGoal(3, p8, p9, i(11, 19)).
uncovered(4, p8, p8, i(13, 21)).
closerToGoal(5, q8, q9, i(16, 26)).
pass(6, p7, p6, i(27, 29)).
closerToGoal(7, p6, p7, i(23, 31)).
uncovered(8, p6, p6, i(25, 33)).
uncovered(9, q9, q9, i(30, 36)).
closerToGoal(10, q8, q6, i(36, 40)).
pass(11, p9, p7, i(39, 41)).
closerToGoal(12, p7, p9, i(35, 43)).
uncovered(13, q8, q8, i(42, 44)).
uncovered(14, p7, p7, i(37, 45)).
pass(15, p8, p6, i(51, 53)).
closerToGoal(16, q7, q6, i(50, 54)).
closerToGoal(17, p6, p8, i(47, 55)).
uncovered(18, p6, p6, i(49, 57)).
pass(19, p8, p7, i(65, 67)).
uncovered(20, q6, q6, i(58, 68)).
closerToGoal(21, p7, p8, i(61, 69)).
uncovered(22, p7, p7, i(63, 71)).
currentIndex(11).
currentIndex(12).
(...)
currentIndex(81).
currentIndex(82).
```

Figure 6.24: *WARMR* knowledge base file experiment.kb

```
unif(X, Y) :-
        X = Y.

validInterval(S, E, WindowEnd) :-
        windowSize(WindowSize),
        WindowStart is (WindowEnd - WindowSize),
        E > WindowStart,
        S =< WindowEnd.

before(i(_S1, E1), i(S2, _E2)) :-
        E1 < S2.
olderContemp(i(S1, E1), i(S2, _E2)) :-
        S1 < S2,
        E1 >= S2.
headToHead(i(S1, _E1), i(S2, _E2)) :-
        S1 == S2.

windowSize(12).

closerToGoal(Id, O0, O1,  i(S, E), WindowEnd) :-
        closerToGoal(Id, O0, O1,  i(S, E)),
        validInterval(S, E, WindowEnd).

useful_constant_closerToGoal_O0(C) :-
        findall(X, closerToGoal(_,X,_,_), ConstList), setof(Y, member(Y,ConstList), ConstSet), !,
        member(C, ConstSet).

eq_obj_closerToGoal_O0(X, Y) :-
        X = Y.

useful_constant_closerToGoal_O1(C) :-
        findall(X, closerToGoal(_,_,X,_), ConstList), setof(Y, member(Y,ConstList), ConstSet), !,
        member(C, ConstSet).

eq_obj_closerToGoal_O1(X, Y) :-
        X = Y.

pass(Id, O0, O1,  i(S, E), WindowEnd) :-
(...)
uncovered(Id, O0, O1,  i(S, E), WindowEnd) :-
(...)

concept(X) :-
        isA(X,_).

subConceptOf(X, Y) :-
        isA(X, Y).

subConceptOf(X, Y) :-
        isA(Z,Y),
        subConceptOf(X,Z).

instanceOf(Inst, Concept) :-
        directInstanceOf(Inst, Concept).

instanceOf(Inst, Concept) :-
        directInstanceOf(Inst, SubConcept),
        subConceptOf(SubConcept, Concept).
```

Figure 6.25: *WARMR* background knowledge file experiment.bg

Max. ref. level	#created *WARMR* patterns	#created *MiTemP* patterns	#frequent *WARMR* patterns	#frequent *MiTemP* patterns	#redundant *WARMR* patterns	#redundant *MiTemP* patterns	#common patterns *WARMR*/ *MiTemP*	#unique patterns *WARMR*	#unique patterns *MiTemP*	Coverage of *MiTemP* patterns in *WARMR*	Coverage of *WARMR* patterns in *MiTemP*
2	4	27	3	14	0	0	3	0	11	$\frac{3}{14}$ = 21.43%	$\frac{3}{3}$ = 100.0%
3	62	122	38	59	1	0	38	0	21	$\frac{38}{59}$ = 64.41%	$\frac{38}{38}$ = 100.0%
4	382	457	96	193	13	0	88	8	105	$\frac{88}{193}$ = 45.60%	$\frac{88}{96}$ = 91.67%
5	2229	1407	287	525	32	0	287	0	238	$\frac{287}{525}$ = 54.67%	$\frac{287}{287}$ = 100.0%
6	9255	3571	943	1153	287	0	874	69	279	$\frac{874}{1153}$ = 75.80%	$\frac{874}{943}$ = 92.68%
7	42319	7280	1897	2089	1078	0	1684	213	405	$\frac{1684}{2089}$ = 80.61%	$\frac{1684}{1897}$ = 88.77%

Table 6.1: Results of the test runs with different maximal refinement levels

In order to compare the results, another converter for the *WARMR* output and *Prolog* clauses for reading and comparing the patterns generated by the two approaches were implemented. The patterns generated by *WARMR* are processed in a way that their most special equivalent pattern representation is created. Redundant patterns are then counted and removed. It is checked for common and unique patterns in the two sets of patterns generated by *MiTemP* and *WARMR*. For all common patterns, the frequency values are compared in order to test that there is no mismatch in the two support computations. In the experiments, all frequency values of the common patterns are identical as desired.

The results of the experiments with the different maximal refinement levels are shown in Table 6.1. Two graphs visualize some of the results: Fig. 6.26 compares the number of created patterns for the different refinement levels and Fig. 6.27 shows the number of frequent and redundant created patterns for the two approaches at the different refinement levels. As it can be seen, the number of created patterns grows much faster w.r.t. the refinement level with *WARMR*. At the maximal refinement level seven, more than 42000 patterns have been generated by *WARMR*. At level seven, *MiTemP* has generated 7280 patterns, i.e., only about one fifth of those generated by *WARMR*. At the same time, the number of identified *frequent* patterns is higher with *MiTemP* compared to *WARMR*. The two graphs of the identified frequent patterns shown in Fig. 6.27 appear to be quite similar. However, it should be noted that some of the patterns generated by *WARMR* are "incomplete" in the sense that derivable information is not generated (e.g., specialization by temporal reasoning or concept refinement after instantiation) as it is the case in *MiTemP*. This information is generated by the *WARMR* comparison clauses in *MiTemP*. Thus,

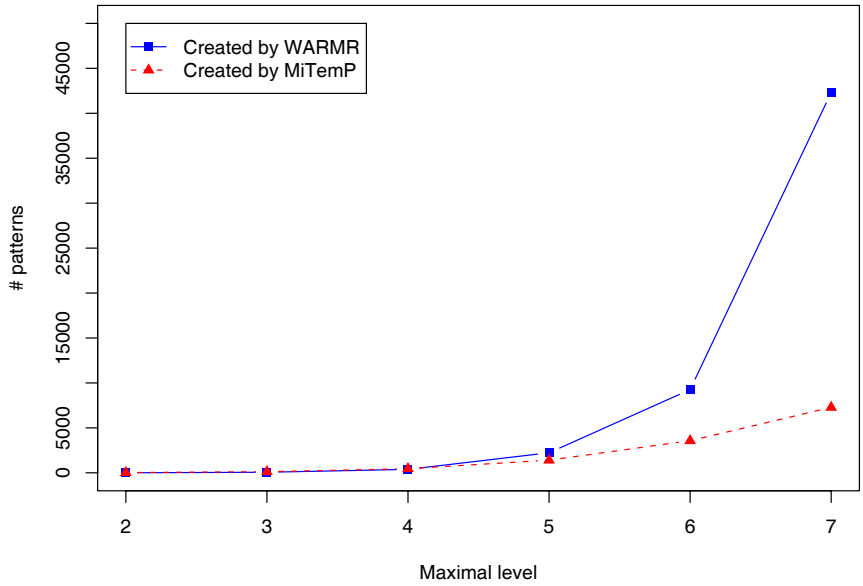

Figure 6.26: Number of created patterns

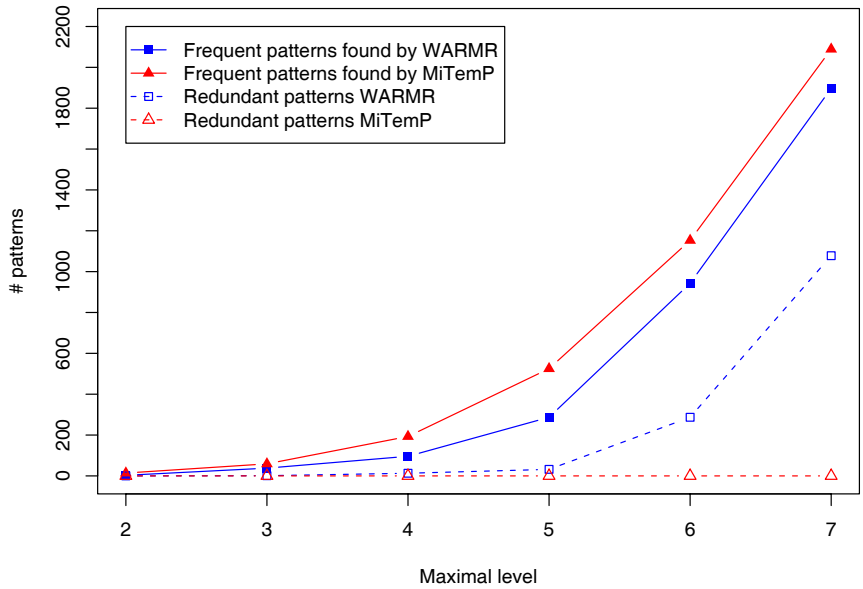

Figure 6.27: Number of frequent and redundant patterns

with the *WARMR*-based solution it can happen easily that a pattern is specialized to one of its equivalent representations. This explains the number of redundant patterns created by *WARMR* (also shown in Fig. 6.27). As it can be seen, no redundant patterns are generated by *MiTemP*.

Another interesting observation is that *WARMR* also generates some patterns "earlier" (i.e., at a lower maximal refinement level) than *MiTemP*. A closer look to the patterns that have been generated by *WARMR* but not (yet) by *MiTemP* has shown that the reason is the handling of the instantiation refinements. *MiTemP* allows an instantiation of a variable if the corresponding concept in the concept restriction has been specialized to the direct concept of an instance. The solution with *WARMR* does not have this restriction. Consequently, in some patterns an instantiation is performed without having restricted the concept restriction to the corresponding concept of the instance. One example is the following pattern identified by *WARMR* in refinement level seven but not yet found by *MiTemP*:

```
pattern([closerToGoal(p6,p8),closerToGoal(q7,var_1)],
        temp(tr([olderContemp])),
        conceptRestr(team1,team1,team2,object))
```

The corresponding query identified by *WARMR* is:

```
freq(7,1011,[currentIndex(A),closerToGoal(B,C,D,E,A),
             closerToGoal(F,G,H,I,A),not(F=B),
             eq_obj_closerToGoal_O0(G,p6),eq_obj_closerToGoal_O0(C,q7),
             eq_obj_closerToGoal_O1(H,p8),
             olderContemp(I,E)],0.222222222222222).
```

In order to generate this pattern with *MiTemP*, nine refinements are necessary. Two lengthening operations for the two atomic patterns, one temporal refinement for the restriction of the interval relation, three concept restrictions (to the concepts `team1` or `team2`), and finally three instantiations.

MiTemP also finds these kinds of patterns but in a later refinement step. Conceptually, it would not be a problem to adapt *MiTemP* in a way to allow direct instantiations. The intention behind this restriction was to avoid the generation of many patterns with instantiations while it is not even known if the union of these instances (in a common concept) leads to a frequent pattern.

The results presented in Table 6.1 on page 179 also show that the proportion of identified frequent patterns to generated patterns is much higher with *MiTemP*. At level seven, for instance, the proportion of *WARMR* is 4.48% while *MiTemP* has a value of 28.70%.

Id	File(s)	League	Match	Result	Date
Match 1	match_2d_1_1, match_2d_1_2	2D	TsinghuAeolus vs. FC Portugal	7:0	06/22/2002
Match 2	match_2d_2_1, match_2d_2_2	2D	FC Portugal vs. Puppets	6:0	06/21/2002
Match 1	match_3d_1_1, match_3d_1_2	3D	Virtual Werder vs. Mithras	0:0	06/14/2006
Match 2	match_3d_2_1, match_3d_2_2	3D	MRL vs. ZJU Base	0:2	06/15/2006
Match 3	match_3d_3_1, match_3d_3_2	3D	WrightEagle vs. Aria	0:1	06/16/2006
Match 4	match_3d_4_1, match_3d_4_2	3D	WrightEagle vs. FC Portugal	0:0	06/18/2006
Match 5	match_3d_5_1, match_3d_5_2	3D	SEU vs. Rezvan	0:1	06/18/2006

Table 6.2: Overview of the used RoboCup 2D/3D simulation league matches

6.4 Learning Prediction Rules from RoboCup Matches

The goal of this part of the evaluation is to show how prediction rules can be generated from RoboCup soccer matches. Input to be learned from are matches of the RoboCup 2D and 3D simulation leagues. Preliminary results w.r.t. learning patterns from RoboCup matches can be found in [LMVH06].

Table 6.2 gives an overview of the soccer matches used in the experiments. In the case of the 2D simulation league, the two matches analyzed by Miene [Mie04] are used (in a slightly changed representation). These matches have taken place at the RoboCup 2002 in Fukuoka, Japan. The remaining five matches are taken from the 3D simulation league of the recent RoboCup 2006 in Bremen, Germany.

In the case of the 2D experiments, these predicates are available in the dynamic scene descriptions (generated by Miene's approach [Mie04]):

- approaches: A player approaches the ball.
- meets: A player meets the ball.
- departs: A player departs from the ball.
- pass: A successful pass between two players is performed.
- lostpass: A failed pass has been performed.
- dribbling: A player moves with the ball.
- ballControl: A player has the ball control.
- attacks: A player attacks another player that has ball control.

```
(...)
holds(front(p10, p7), 7, 8).
holds(front(p10, p9), 7, 8).
holds(meets(p10), 1, 8).
holds(closerToGoal(p10, p9), 1, 13).
holds(closerToGoal(p10, p11), 1, 13).
holds(pass(p10, p7), 8, 13).
holds(front(p7, p10), 13, 18).
holds(meets(p7), 13, 18).
holds(closerToGoal(p7, p9), 13, 19).
holds(closerToGoal(p7, p10), 13, 19).
(...)
```

Figure 6.28: Snippet of match_2d_1_1.P

- uncovered: No opponent is close to a player.
- closerToGoal: A player is closer to the opponent's goal than the other.
- front: A player is in front of the other.

For the 3D simulation league the tool *FactProducer* for the extraction of qualitative scene descriptions from log files has been developed. It extracts this information:

- team: The corresponding team of a player.
- dist: The distance between two objects (kickable, very close, close, medium, far).
- bearing: The direction of a player (north, north west, west, ...).
- playmode: The current playmode of the match (goalkick, kickoff, playon, ...)
- pass: A successful pass between two players is performed.
- failPass: A failed pass has been performed.
- clear: A player cleared the ball.
- selfAssist: A player moves with the ball (kind of dribbling).
- inPassRegion: A player is in the region where a pass is performed.

Each match is split into two parts, one for each half of the match. The scene files with the first half are annotated with a "_1", the ones with the second half with a "_2". `match_3d_1_1`, for instance, is the first half of the first 3D simulation league match (*Virtual Werder* vs. *Mithras*). Figures 6.28 and 6.29 show snippets of two of the input files in the *MiTemP* format. The complete data sets and the

```
(...)
holds(dist(b,q8,close), 1175, 1179).
holds(bearing(q8,northEast), 1178, 1180).
holds(dist(b,q8,kickable), 1180, 1183).
holds(pass(q1,q8), 1149, 1180).
holds(inPassRegion(p6), 1139, 1156).
holds(inPassRegion(p10), 1139, 1190).
holds(inPassRegion(p11), 1139, 1153).
holds(inPassRegion(p9), 1150, 1156).
holds(dist(q8,p10,medium), 1139, 1154).
holds(dist(q8,p11,medium), 1149, 1162).
holds(dist(q8,p10,close), 1155, 1158).
(...)
```

Figure 6.29: Snippet of match_3d_1_1.P

learning output can be found on the DVD in the folders `/eval/robocup/2d` and `/eval/robocup/3d`, respectively.

In the experiments, the window size is set to 20 and patterns of the sizes from 1 to 3 are generated. The maximal refinement level is set to 20 and the minimal frequencies are 0.005 in all experiments. At the 2D experiments, two runs have been performed: In the first run, prediction rules are generated from the first half of the first match and in the second run, the first half of the second match is used as learning input. The experiments with the matches of the 3D simulation league are set up similarly. In this case, for each of the five matches the first half is used for learning in five different runs.

In the first three refinement levels, all generated frequent patterns are taken into account for next level candidate generation in order to avoid missing any frequent atomic pattern combination as the lengthening operator needs a set of frequent $(n-1)$ patterns. For later refinement levels, 200 patterns are chosen randomly from the set of frequent patterns of the current level using the third method mentioned above for reducing the number of generated patterns. In order to avoid results due to a single random selection, all experiments are repeated three times (referred to as "run 1", "run 2", and "run 3").

After mining the frequent patterns and creating the corresponding prediction rules (compliant with the defined minimal frequency and confidence thresholds; in this experiments the minimum confidence of 0.5 is used), the generated prediction rules are applied to the different dynamic scenes – including the one used for learning (these values have not been used for the computation of the average accuracies). For both parts of the experiment – 2D as well as 3D – all created prediction rules are taken into account in the evaluation.

It should be noted that selecting a minimal confidence value of 0.5 also generated prediction rules that might have a low accuracy as 0.5 on the training data already. If only rules with higher prediction accuracies are desired, the minimal confidence value must be increased.

Training	Match 1-1	Match 1-2	Match 2-1	Match 2-2	Average on unseen parts
Match 1-1	(77.12%)	68.32%	65.09%	71.43%	68.28%
Match 2-1	75.05%	67.39%	(73.56%)	74.12%	72.19%
Total					70.24%

Table 6.3: Accuracy of all prediction rules in 2D RoboCup matches - Run 1

Training	Match 1-1	Match 1-2	Match 2-1	Match 2-2	Average on unseen parts
Match 1-1	(76.63%)	67.70%	64.43%	70.68%	67.60%
Match 2-1	75.48%	68.10%	(74.09%)	74.59%	72.72%
Total					70.16%

Table 6.4: Accuracy of all prediction rules in 2D RoboCup matches - Run 2

Training	Match 1-1	Match 1-2	Match 2-1	Match 2-2	Average on unseen parts
Match 1-1	(76.56%)	67.99%	64.23%	71.17%	67.80%
Match 2-1	75.07%	68.08%	(74.32%)	74.38%	72.51%
Total					69.85%

Table 6.5: Accuracy of all prediction rules in 2D RoboCup matches - Run 3

Training	Match 1-1	Match 1-2	Match 2-1	Match 2-2	Average on unseen parts
Match 1-1	(76.77%)	68.00%	64.58%	71.09%	67.89%
Match 2-1	75.20%	67.86%	(73.99%)	74.36%	72.47%
Total					70.18%

Table 6.6: Accuracy of all prediction rules in 2D RoboCup matches - Average

The average prediction accuracy for all created prediction rules for the three runs of the 2D simulation league matches is shown in Tables 6.3, 6.4, and 6.5. The average values of all three runs can be seen in Table 6.6. The values in brackets denote the data sets that have been used for learning. As mentioned above, these values are less than 100% as the minimal confidence of 0.5 has been used at the prediction rule generation procedure, i.e., the program has also generated prediction rules that do not have perfect prediction quality on the training data.

In the three single runs, the overall accuracy is 70.24%, 70.16%, and 69.85%. The average prediction accuracy for the generated rules on unseen data is 70.18%; the prediction rules generated from the second match have a higher accuracy (67.89% vs. 72.74%). As it can be seen, the prediction rules generated from the first half of the second match also work well for the first match (Table 6.6).

The results of the corresponding experiments with the data of the 3D simulation league for the single runs are shown in Tables 6.7, 6.8, and 6.9. The average accuracies are shown in Table 6.10. Using all generated rules, the average prediction accuracy on unseen data is 63.42% – lower than in the experiments with the 2D simulation league data. In the best case, the prediction rules learned from Match

Train. match	Match 1-1	Match 1-2	Match 2-1	Match 2-2	Match 3-1	Match 3-2	Match 4-1	Match 4-2	Match 5-1	Match 5-2	Avg. on unseen parts
1-1	(87.24%)	54.19%	70.02%	64.42%	62.41%	63.35%	56.41%	62.83%	64.54%	64.74%	62.55%
2-1	45.42%	55.56%	(81.98%)	64.84%	64.89%	70.22%	62.02%	66.72%	69.86%	63.27%	62.53%
3-1	41.60%	53.78%	70.05%	63.86%	(80.66%)	73.36%	66.6%	66.71%	70.17%	60.89%	63.00%
4-1	38.85%	54.28%	73.72%	69.11%	71.38%	74.34%	(80.67%)	70.88%	73.41%	64.45%	65.60%
5-1	46.12%	61.12%	77.05%	68.38%	72.71%	77.10%	71.62%	72.15%	(79.94%)	67.79%	68.23%
Total											64.38%

Table 6.7: Accuracy of all prediction rules in RoboCup matches (3D) - Run 1

Train. match	Match 1-1	Match 1-2	Match 2-1	Match 2-2	Match 3-1	Match 3-2	Match 4-1	Match 4-2	Match 5-1	Match 5-2	Avg. on unseen parts
1-1	(84.63%)	52.57%	67.13%	59.27%	59.66%	60.80%	55.12%	58.61%	62.05%	62.9%	59.79%
2-1	44.82%	56.08%	(83.09%)	67.65%	66.43%	71.43%	64.70%	69.54%	71.73%	65.36%	64.19%
3-1	47.94%	53.10%	73.63%	65.53%	(77.51%)	73.75%	65.63%	69.60%	72.99%	64.67%	65.20%
4-1	39.86%	52.67%	68.58%	65.59%	69.11%	73.58%	(79.75%)	69.51%	72.10%	58.93%	63.33%
5-1	29.76%	49.45%	71.26%	57.66%	68.62%	71.27%	66.10%	68.2%	(78.62%)	60.57%	60.32%
Total											62.56%

Table 6.8: Accuracy of all prediction rules in RoboCup matches (3D) - Run 2

Train. match	Match 1-1	Match 1-2	Match 2-1	Match 2-2	Match 3-1	Match 3-2	Match 4-1	Match 4-2	Match 5-1	Match 5-2	Avg. on unseen parts
1-1	(85.14%)	51.00%	66.54%	59.44%	58.40%	60.47%	54.48%	59.04%	59.63%	63.19%	59.13%
2-1	36.27%	46.55%	(81.33%)	62.00%	63.42%	68.71%	63.00%	64.86%	67.55%	60.74%	59.23%
3-1	55.54%	59.48%	77.99%	69.21%	(79.75%)	76.20%	69.80%	74.11%	75.38%	68.99%	69.63%
4-1	40.96%	53.29%	74.88%	72.20%	74.63%	74.86%	(83.94%)	74.02%	74.84%	66.50%	67.35%
5-1	32.72%	53.28%	74.69%	61.12%	66.57%	72.53%	65.75%	66.41%	(78.39%)	62.80%	61.76%
Total											63.42%

Table 6.9: Accuracy of all prediction rules in RoboCup matches (3D) - Run 3

3-1 have an accuracy of 65.95% on unseen data. In these experiments, it can be seen that the learned rules are more characteristic w.r.t. the training data. The average accuracies are significantly higher for the training data compared to the unseen data (approximately 81.7% vs. 63.4%).

The matches of the 3D simulation league are also used for another experiment. Table 6.11 shows the different accuracies of all prediction rules for different refinement levels. The average values for each level are also plotted in Fig. 6.30. It can be observed that the average accuracy on unseen data decreases monotonically with an increasing refinement level. At refinement level eight the average accuracy is 65.52%, at level twelve it is only 42.50%. Thus, it can be concluded that more characteristic rules for a certain soccer match can be found at higher refinement levels.

However, it is possible to identify both – rules that are characteristic for a match

Train. match	Match 1-1	Match 1-2	Match 2-1	Match 2-2	Match 3-1	Match 3-2	Match 4-1	Match 4-2	Match 5-1	Match 5-2	Avg. on unseen parts
1-1	(85.67%)	52.59%	67.90%	61.04%	60.16%	61.54%	55.34%	60.16%	62.07%	63.61%	60.49%
2-1	42.17%	52.73%	(82.13%)	64.83%	64.91%	70.12%	63.24%	67.04%	69.71%	63.12%	61.99%
3-1	48.36%	55.45%	73.89%	66.20%	(79.31%)	74.44%	67.34%	70.14%	72.85%	64.85%	65.95%
4-1	39.89%	53.41%	72.39%	68.97%	71.71%	74.26%	(81.45%)	71.47%	73.45%	63.29%	65.43%
5-1	36.20%	54.62%	74.33%	62.39%	69.30%	73.63%	67.82%	68.92%	(78.98%)	63.72%	63.44%
Total											63.46%

Table 6.10: Accuracy of all prediction rules in RoboCup matches (3D) - Average

Level	Match 1-1	Match 1-2	Match 2-1	Match 2-2	Match 3-1	Match 3-2	Match 4-1	Match 4-2	Match 5-1	Match 5-2	Avg. on unseen parts
08	(87.59%)	56.62%	74.69%	66.83%	64.7%	66.53%	58.57%	66.21%	67.39%	68.16%	65.52%
09	(86.54%)	48.42%	63.65%	57.66%	56.44%	55.36%	48.14%	55.35%	58.75%	57%	55.64%
10	(86.12%)	42.43%	55.71%	51.36%	49.61%	48.93%	42.88%	48.73%	51.29%	50.57%	49.06%
11	(86.37%)	40.85%	53.24%	52.36%	47.84%	48.33%	42.71%	46.75%	49.72%	46.6%	47.60%
12	(87.29%)	36.48%	45.32%	51.42%	41.92%	43.38%	38.09%	42.16%	45.17%	38.59%	42.50%

Table 6.11: Accuracy for different refinement levels (3D, Run 1)

as well as rules that are more generic and also hold in other matches. In following experiments, rules have been sorted w.r.t. different quality criteria. In the first case, the goal was to find general prediction rules that work well for all matches. The quality measure therefore is only the average accuracy on both – the training match as well as the other matches. An example for such a general rule is shown in Fig. 6.31. The rule says that if the ball is in far distance to some player, then passed from another player (`_h1161`), this player will be in far distance to the ball later on (the temporal relations among the predicates are older & contemporary in all cases). The accuracy of this prediction rule is 79.43% on all matches.

In order to identify specific rules for a match, the quality of the generated prediction rules has been computed by the fraction of the accuracy for the rule on the first and second half of the training match divided by the average accuracy of the other matches. Fig. 6.32 shows such a characteristic rule that has an average accuracy of 50.83% on the training match and an average accuracy of only 19.97% on the remaining matches. It says that after a pass between `_h1384` and `_h1385` there is an object `_h1404` in far distance to the ball, and this object will be `close` to the receiver of the ball later on. This is a rule that can be explained by a certain behavior of the *Virtual Werder 3D* agents that use a *cover* skill in order to stay close to opponents that might get the ball passed by a team mate (all temporal relations are older & contemporary).

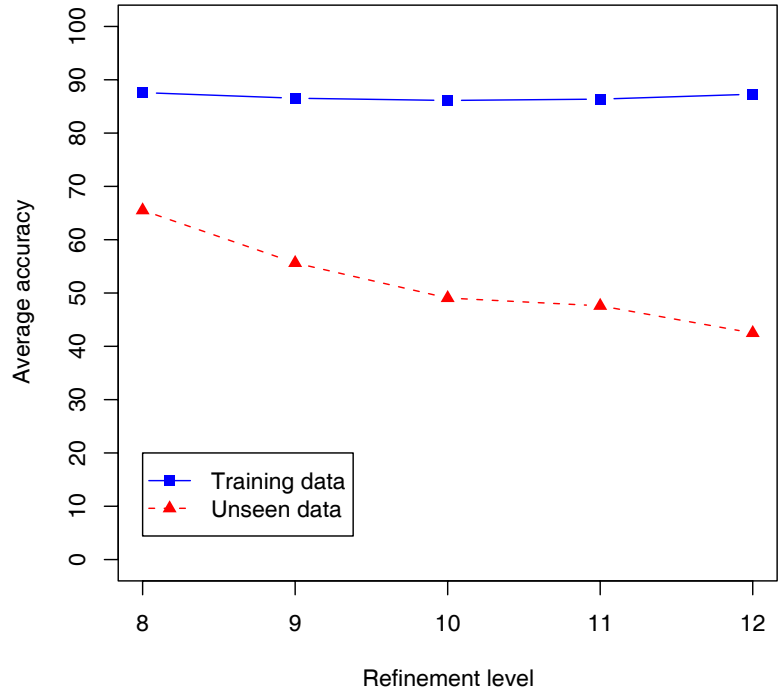

Figure 6.30: Accuracy for different refinement levels: Training vs. unseen data

```
[dist(_h1155,_h1156,far),pass(_h1161,_h1162)] => [dist(_h1155,_h1161,far)]
temp(tr([olderContemp],[olderContemp]),tr([olderContemp]))
conceptRestr(ball,player,distance,player,player,ball,player,distance)

Eval: 0.5848
(f: 0.0119, c: 0.8966, j: 0.0003, s: 1.0000, r: 0.6000, p: 1.0000)
```

Figure 6.31: General prediction rule generated from match_3d_1_1.P

```
[pass(_h1384,_h1385)] => [dist(_h1403,_h1404,far),dist(_h1385,_h1404,close)]
temp(tr([olderContemp],[olderContemp]),tr([olderContemp]))
conceptRestr(player,player,ball,player,distance,player,player,distance)

Eval: 0.5276
(f: 0.0109, c: 0.5490, j: 0.0056, s: 1.0000, r: 0.6000, p: 1.0000)
```

Figure 6.32: Characteristic prediction rule generated from match_3d_1_1.P

6.5 Discussion

In this chapter, the evaluation of the temporal pattern mining approach has been presented. The introductory example has shown how the programs for frequent pattern mining, prediction rule generation, and prediction rule application work and that the frequent patterns and prediction rules are generated as intended. In a number of experiments with synthetic data, different aspects of the learning algorithm have been investigated. Some of the results have not been intuitive at the first glance. In some cases, the generation of more patterns would have been expected with increasing parameter values but due to less *frequent* patterns (for instance, if a constant number of predicates is distributed over a varying number of predicate templates) the result can be the opposite. These rather simple examples on synthetic data already show the complexity in temporal pattern mining. The window size – and in correlation the number of predicates present within a sliding window position – has been identified as a "bottleneck" of the approach.

The comparison of the *WARMR*-based solution with *MiTemP* in the third experiment had two goals: to test if the mining task can be mapped to *WARMR* and to find out if the additional knowledge used by *MiTemP* is of advantage. It has been successfully shown that both approaches create the same patterns with the same support/frequency values as desired. The advantage of *MiTemP* by bringing in knowledge about temporal relations and the concept hierarchy could also be shown. *WARMR* generates many redundant patterns in the pattern mining process while this is avoided in *MiTemP* by using the most special representation of a pattern. At level seven, *WARMR* has already generated more than 42000 patterns while *MiTemP* has only created 7280. At the same time, *MiTemP* has identified more relevant (frequent) patterns than *WARMR*. This is a significant advantage which is especially valuable if support computation is expensive (e.g., for long dynamic scenes).

In the final experiment, the learning approach has been applied to scenes of the RoboCup 2D and 3D simulation league. Frequent patterns as well as prediction rules have been generated and applied to (mostly) unseen scenes. It has been shown how the approach can be used in this domain and it could be seen that prediction rules with average accuracies of 70.18% and 63.46% on unseen data in the 2D and 3D experiments could be generated. Of course, it is possible to filter out those prediction rules with higher accuracies (by increasing the minimal confidence) or to select prediction rules on further matches as presented above in order to get characteristic prediction rules for a specific soccer match or general rules that can be used for the prediction on different matches.

Chapter 7

Summary and Perspective

Dynamic scenes with many different objects and interrelations changing over time demand complex representations. The identification of frequent patterns and prediction rules in such scenes would be very valuable as associations in the data could be discovered or a system's performance could even be improved by utilizing the new information in the behavior decision process. In this work, a novel approach to temporal pattern mining in dynamic environments has been proposed.

In the requirements analysis, different demands on representation formalisms for dynamic scenes as well as for patterns to be mined from dynamic scenes have been identified from a soccer scenario of the RoboCup simulation league. Among other demands the representation of objects and relations and the temporal validity of these relations have been detected to be crucial. As various actions and events in dynamic scenes can occur concurrently, it is important that the representation also supports this concurrency. A comprehensive investigation of the state of the art has led to a number of relevant approaches covering parts of the requirements. Particularly of interest were the approaches dealing with association rule mining, especially the extensions for sequential or temporal data as well as the extensions which can also handle relational data. *WARMR* [DT99] can be used to mine relational association rules but has no direct means to represent the temporal dimension. The work of Höppner [Höp03] which addresses rule learning from interval-based temporal data uses interval relations in order to represent temporal interrelations among states in the input data. A third relevant approach by Lee [Lee06] allows for mining first-order sequential rules but does not support concurrently occurring intervals of relations.

Based on the defined requirements and on the analyzed approaches, the novel temporal pattern mining approach *MiTemP* has been developed. It can mine temporal patterns from interval-based relational data. The relevant concepts of the approach have been defined formally in order to establish a formal basis for this work. Besides formal definitions of dynamic scenes and their schemata as well as

definitions for patterns and prediction rules, a partial order has been defined for the generalization relation between patterns. Following ideas of Lee [Lee06], an optimal refinement operator has been defined that guarantees a complete and non-redundant generation of frequent patterns from the given representation. In the conceptual chapter of this thesis, it has also been shown how *WARMR* can be used in order to mine temporal patterns albeit it generates many redundant ones.

The pattern mining approach starts with the most general (empty) pattern and successively performs refinement operations to those patterns that still exceed the minimal frequency threshold. Five refinement operations have been set up: lengthening, temporal refinement, unification, concept refinement, and instantiation. Existing approaches (e.g., *WARMR*) also allow for restricting types of variables but do not provide means to handle concept hierarchies as presented in this thesis. Concept restrictions can represent the information that specific variables can only be bound to instances of certain concepts in the concept hierarchy.

In order to represent temporal relations in patterns, a new concise set of mutual exclusive and jointly exhaustive interval relations has been set up by combining ideas from Allen's and Freksa's interval relations. For the interval relations, a composition table has been set up and an algorithm for guaranteeing path consistency in the temporal constraint graph has been set up. The reason for the new set of interval relations was to reduce complexity and to focus on important relations for prediction rules. However, the set of interval relations can be easily replaced without changing the mining algorithm, e.g., by using Allen's interval relations.

It can easily happen that a huge number of patterns is generated. Therefore, different means to restrict the relevant pattern space have been introduced. It is possible to disable single refinement types (e.g., no instantiation) or to restrict the maximal refinement level. If it is known before mining that only certain patterns with some predicates are of interest, a bias can be set up consisting of partial conjunctive patterns. If such a bias is defined, the patterns that are inconsistent with this bias are filtered out during pattern mining. Another way to reduce the number of patterns is a selection of patterns to be refined at each refinement level. In the *MiTemP* implementation, a random selection of n patterns can be done before the refinement is performed.

In order to utilize the identified frequent patterns for prediction, an algorithm for prediction rule generation has been introduced. Prediction rules are like other association rules with the restriction that the consequence of the rule must be past the precondition. As it has been shown, this has the advantage of only linear effort w.r.t. the conjunctive pattern size. Due to the anti-monotonicity property of the frequency, the prediction rule generation procedure can even be stopped if the minimal confidence value cannot be reached any more. Thus, an efficient method for creating prediction rules from temporal patterns has been introduced. Furthermore, different criteria for the evaluation of prediction rules have been set up: frequency,

confidence, information value (J-measure), predicate preference, size, and specifity. The different criteria are put together by a weighted sum to an overall evaluation value. If certain aspects should not be considered in the evaluation measure, the corresponding weights can be set to zero.

The learning approach has been implemented in *XSB Prolog*. For evaluation also a number of further scripts and programs have been developed. Among others, the JAVA program *SeqDraw* for the visualization of interval representations of dynamic scenes, the *TestDataCreator* (also JAVA) for the generation of synthetical test data, and the *FactProducer* (C++) for the generation of a qualitative, interval-based representation of 3D simulation league log files have been implemented[1]. The evaluation consists of four parts: a simple example for illustration purposes, a number of experiments with synthetical data, a comparison of *WARMR* and *MiTemP*, and the generation of prediction rules from RoboCup soccer matches. The experiments have shown the successful implementation of the developed temporal pattern mining approach as well as the prediction rule generation and application to dynamic scenes from the RoboCup domain. The advantage of *MiTemP* in comparison to *WARMR* has been shown; bringing in the knowledge about temporal relations and concepts can reduce the number of patterns to be generated (and checked during pattern matching) significantly.

Starting from the results of this thesis, many different directions for future research can be envisioned. First of all, the complexity of the mining algorithm has been identified as one of the crucial aspects of the approach. This problem can be addressed by applying optimizations which preserve completeness or by developing some heuristics or restrictions that lead to a partial set of frequent patterns. One possible solution for increasing efficiency without the loss of completeness could be a restructuring of the pattern's query in the pattern matching procedure. First checking those predicates that help to find out that there is actually no match for the pattern (e.g., the least frequent or probable predicate) could reduce the backtracking steps while searching for a valid match. If completeness is not important and it is sufficient to find *some* of the frequent patterns in the different refinement levels, only a selection of the previously frequent patterns could be further refined. In the current implementation, this selection can only be done randomly. It would be valuable to have an estimation which of the patterns found so far would be most interesting for being refined, i.e., the refinements of which patterns are expected to lead to interesting frequent patterns or prediction rules. It could be helpful to bring in statistical information about occurrences of predicates or objects for the decision which patterns are promising and which are not.

Another possibility for reducing the complexity is pruning: If it is sufficient to find the n best patterns w.r.t. some evaluation function and if it can be guaranteed

[1] *SeqDraw* has been implemented in cooperation with Carsten Rachuy; the C++ version of the *FactProducer* has been realized by Tobias Warden.

that some pattern and its specializations cannot be within the best n patterns anymore, the pattern (and its specializations) could be pruned. The challenge would be to set up the system in a way that anti-monotonicity of the evaluation values is given or by examining boundary values (if the best possible case for evaluation values of specializations of a pattern cannot exceed the evaluation value of the best n patterns, the branch can be pruned).

The branching factor of the search tree in the hypothesis space can grow quickly due to the many different kinds of refinements and their potentially large number of variants (e.g., if a large concept hierarchy or many instances are present). A way to address this problem could be a hybrid solution: Mining parts of the patterns as presented in this thesis and then applying classical propositional learning algorithms in order to identify further regularities in the data. A statistical analysis of the matches of patterns – e.g., by counting the occurring instances and their concepts – could also lead to efficiency improvements as many pattern candidates could be checked all at once resulting in a set of frequent more specialized patterns. However, if separate statistics for the bindings of the variables in the pattern were counted, a new solution for checking combinations of these specialized patterns would be needed.

As it has been mentioned in the conceptual chapter of this work, it has been focused on *fluent* predicates, i.e., on those predicates with a temporal validity interval. In many cases, scenes also consist of static information, i.e., facts that cannot or do not change over time (e.g., the team membership of a player at a soccer match). Conceptually, such information can be represented by large validity intervals starting before the first start time in the scene and ending past the latest end time, but this would not be a very good solution as unnecessary complexity is brought into the mining process. The reason is that the mining algorithm has additionally to check the temporal relations between the static and other intervals. It would be helpful to introduce separated types of predicates – static and fluent ones – where the static predicates are left out in the temporal refinement process.

So far, the exclusion of certain patterns has only been performed by utilizing the composition table for the interval relations. An interesting extension of the system would be a more general approach that allows for setting up constraints among predicates and thus avoids the generation (or support computation) of illegal patterns. If it is known that an object cannot be *in front of* and *behind* (w.r.t. the spatial position) another object at the same time, it is not necessary to check such a pattern.

If the result of the pattern mining process is too large to be easily inspected by the user, post-processing for structuring the created patterns could be helpful. As potentially many similar patterns can be generated, a similarity measure for patterns could be set up to apply a clustering approach in order to group similar patterns. If just one or two representatives of each cluster are presented to the user,

the inspection could be eased. The challenging task would be the definition of a reasonable similarity (or distance) measure.

In this thesis, an approach for mining frequent patterns from dynamic scenes has been presented. It would be interesting to develop further concepts in order to identify relevant characteristics of a group of scenes. This would go rather in the direction of classification, for instance, to find common patterns and prediction rules in soccer matches of *Werder Bremen* in comparison to matches of other teams. A first idea in this direction is to identify prediction rules on a set of training matches and check the prediction accuracy on further training matches selecting those rules that work well for *Werder Bremen* but do not for the other teams as it has been shown in the evaluation chapter.

The result of the pattern mining algorithm and the prediction rule generation is a set of patterns and rules that are not related in any way. Some of the rules with identical consequence might work well in different parts of the scene. It would be interesting to create a composite prediction rule that combines a number of prediction rules with the same consequence in a way that the upmost coverage for a prediction is given in a scene.

Another direction of future research is to apply the approach to different application domains in order to find out if valuable patterns or prediction rules can be generated. One interesting domain would be logistics where many different objects of various kinds are in different relations. In logistics, the temporal dimension is also of great interest and finding temporal patterns or prediction rules could help to avoid unfavorable situations like using a route where the probability for a traffic jam is high due to the current situation. Of course, in many other domains like, for instance, medicine, telecommunication, or the WWW it would be interesting to mine temporal patterns and prediction rules, too.

Bibliography

[ADT95] Robert Andrews, Joachim Diederich, and Alan B. Tickle. Survey and critique of techniques for extracting rules from trained artificial neural networks. *Knowledge-Based Systems*, 8(6):373–389, December 1995.

[AF94] James F. Allen and George Ferguson. Actions and events in interval temporal logic. Technical Report 521, University of Rochester, New York, Computer Science Department, July 1994.

[Aha92] David W. Aha. Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms. *International Journal of Man-Machine Studies*, 36(2):267–287, 1992.

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun N. Swami. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, Washington, D.C., May 1993.

[AKA91] David W. Aha, Dennis Kibler, and Marc K. Albert. Instance-based learning algorithms. *Machine Learning*, 6(1):37–66, January 1991.

[All83] James F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, November 1983.

[All84] James F. Allen. Towards a general theory of action and time. *Artificial Intelligence*, 23(2):123–154, 1984.

[AN05] Michael Arens and Hans-Hellmut Nagel. Quantitative movement description based on qualitative knowledge about behavior. *Künstliche Intelligenz*, 2:5–11, 2005.

[AP94] Agnar Aamodt and Enric Plaza. Case-based reasoning: foundational issues, methodological variations, and system approaches. *AI Communications*, 7(1):39–59, 1994.

[AP01] Eva Armengol and Enric Plaza. Similarity assessment for relational CBR. In *Case-Based Reasoning Research and Development: ICCBR 2001*, pages 44–58. Springer-Verlag Berlin Heidelberg, 2001.

[AS94] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases, VLDB*, pages 487–499, September 1994.

[AS95] Rakesh Agrawal and Ramakrishnan Srikant. Mining sequential patterns. In *Proceedings of the IEEE International Conference on Data Engineering*, pages 3–14, Taipei, Taiwan, March 1995.

[BD98] Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2,):285–297, 1998.

[BDD+02] Hendrik Blockeel, Luc Dehaspe, Bart Demoen, Gerda Janssens, Jan Rammon, and Henk Vandecasteele. Improving the efficiency of inductive logic programming through the use of query packs. *Journal of Artificial Intelligence Research*, 16:135–166, 2002.

[BDR+06] Hendrik Blockeel, Luc Dehaspe, Jan Rammon, Jan Struyf, Anneleen Van Assche, Celine Vens, and Daan Fierens. *The ACE Data Mining System, User's Manual*. Katholieke Universiteit Leuven, Belgium, February 16 2006.

[Bel34] Eric T. Bell. Exponential numbers. *The American Mathematical Monthly*, 41(7):411 – 419, August/September 1934.

[BGZ04] Roberto Bayardo, Bart Goethals, and Mohammed J. Zaki, editors. *Proceedings of FIMI'04, ICDM 2004 Workshop on Frequent Itemset Mining Implementations*, Brighton, UK, November 1 2004.

[BH99] Michael R. Berthold and David J. Hand, editors. *Intelligent Data Analysis*. Springer, 1999. Cited in [Höp03].

[Bis92] Gilles Bisson. Learning in FOL with a similarity measure. In *Proceedings of the 10th National Conference on Artificial Intelligence (AAAI92)*, pages 82–87, San Jose, CA, July 12-16 1992. Morgan Kaufman.

[Bis95] Gilles Bisson. Why and how to define a similarity measure for object-based representation systems. In N.J.I. Mars, editor, *Towards Very Large Knowledge Bases*, pages 236–246. IOS Press, Amsterdam, 1995.

[Boh01] Uta Bohnebeck. *mRNA-Signalstrukturerkennung mittels Knowledge Discovery*. PhD thesis, Universität Bremen, 2001. DISKI 249, Akademische Verlagsgesellschaft, Berlin.

[BS98] Ralph Bergmann and Armin Stahl. Similarity measures for object-oriented case representations. In *Proceedings of European Workshop on Case-Based Reasoning (EWCBR'98)*, pages 25–36, 1998.

[BSMM99] Il'ja N. Bronstein, Konstantin A. Semendjajew, Gerhard Musiol, and Heiner Mühlig. *Taschenbuch der Mathematik, 4. Auflage*. Verlag Harri Deutsch, Frankfurt am Main, 1999.

[CBL06] Ling Chen, Sourav S. Bhowmick, and Jinyan Li. Mining temporal indirect associations. In W.K. Ng, M. Kitsuregawa, and J. Li, editors, *PAKDD 2006*, volume 3918 of *Lecture Notes in Artificial Intelligence*, pages 425–434. Springer-Verlag Berlin Heidelberg, 2006.

[CDH97] Eliseo Clementini, Paolino Di Felice, and Daniel Hernández. Qualitative representation of positional information. *Artificial Intelligence*, 95(2):317–356, 1997.

[CFG00] Antonio Chella, Marcello Frixione, and Salvatore Gaglio. Understanding dynamic scenes. *Artificial Intelligence*, 123(1-2):89–132, October 2000.

[CFG01] Antonio Chella, Marcello Frixione, and Salvatore Gaglio. Symbolic and conceptual representation of dynamic scenes: Interpreting situation calculus on conceptual spaces. In *Proceedings of the 7th Congress of the Italian Association for Artificial Intelligence on Advances in Artificial Intelligence (AI*IA 2001), LNAI 2175*, pages 333–343. Springer-Verlag Berlin Heidelberg, 2001.

[CH67] Thomas M. Cover and Peter E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, January 1967.

[CH92] William W. Cohen and Haym Hirsh. Learnability of description logics. In *Proceedings of the 5th Annual Workshop on Computational Learning Theory*, Pittsburgh, Pennsylvania, 1992. ACM Press.

[CH94] William W. Cohen and Haym Hirsh. Learning the CLASSIC description logic: Theoretical and experimental results. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fourth International Conference (KR94)*. Morgan Kaufmann, 1994.

[CH01] Anthony G. Cohn and Shyamanta M. Hazarika. Qualitative spatial representation and reasoning: An overview. *Fundamenta Informaticae*, 46(1-2):1–29, 2001.

[CHD98] Stephen Chalup, Ross Hayward, and Joachim Diederich. Rule extraction from artificial neural networks trained on elementary number classification tasks. In *Proceedings of the 9th Australian Conference on Neural Networks*, pages 265–270, Brisbane, Australia, 1998.

[CMM83] Jaime G. Carbonell, Ryszard S. Michalski, and Tom M. Mitchell. An overview of machine learning. In Michalski et al. [MCM83], pages 405–428.

[CN89] Peter Clark and Tim Niblett. The CN2 induction algorithm. *Machine Learning*, 3(4):261 – 283, 1989.

[Coh95] William W. Cohen. Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning*, Lake Taho, California, 1995.

[CR06] Aaron Ceglar and John F. Roddick. Association mining. *ACM Computing Surveys*, 38(2), 2006.

[DD97] Luc Dehaspe and Luc De Raedt. Mining association rules in multiple relations. In S. Džeroski and N. Lavrač, editors, *Proceedings of the 7th International Workshop on Inductive Logic Programming*, volume 1297, pages 125–132. Springer-Verlag, 1997.

[DD04] Kurt Drissens and Sašo Džeroski. Integrating guidance into relational reinforcement learning. *Machine Learning*, 57(3):271–304, December 2004.

[DDB98] Sašo Džeroski, Luc De Raedt, and Hendrik Blockeel. Relational reinforcement learning. In J. Shavlik, editor, *Proceedings of the 15th International Conference on Machine Learning (ICML'98)*, pages 136–143. Morgan Kaufmann, 1998.

[De 98] Luc De Raedt. Attribute-value learning versus inductive logic programming: The missing links. In David Page, editor, *Inductive Logic Programming, 8th International Workshop, ILP-98*, volume 1446 of *Lecture Notes in Computer Science*, Wisconsin, USA, July 22-24 1998. Springer.

[Deh98] Luc Dehaspe. *Frequent Pattern Discovery in First-Order Logic*. PhD thesis, Katholieke Universiteit Leuven, Belgium, 1998.

[dF05] Sandra de Amo and Daniel A. Furtado. First-order temporal pattern mining with regular expression constraints. In *Proceedings of the 20th Brazilian Symposium on Databases (SBBD'2005)*, pages 280–294, 2005.

[dFGL04] Sandra de Amo, Daniel A. Furtado, Arnaud Giacometti, and Dominique Laurent. An Apriori-based approach for first-order temporal pattern mining. In *Proceedings of the 19th Brazilian Symposium on Databases (SBBD'2004)*, pages 48–62, 2004.

[DFL03] Frank Dylla, Alexander Ferrein, and Gerhard Lakemeyer. Acting and deliberating using golog in robotic soccer, a hybrid architecture. In *3rd International AAAI Workshop on Cognitive Robotics*, 2003.

[Dic03] Ernst D. Dickmanns. An advanced vision system for ground vehicles. In *Proceedings of 1st International Workshop on In-Vehicle Cognitive Computer Vision Systems (IVCCVS)*, pages 1–12, Graz, Austria, April 3 2003.

[DRB01] Kurt Driessens, Jan Ramon, and Hendrik Blockeel. Speeding up relational reinforcement learning through the use of an incremental first-order decision tree algorithm. In *Proceedings of the 12th European Conference on Machine Learning, LNAI 2167*, pages 97–108. Springer, Berlin, 2001.

[DT99] Luc Dehaspe and Hannu Toivonen. Discovery of frequent DATALOG patterns. *Data Mining and Knowledge Discovery*, 3(1):7 – 36, March 1999.

[DT01] Luc Dehaspe and Hannu Toivonen. Discovery of relational association rules. In *Relational Data Mining*, pages 189 – 208. Springer-Verlag New York, Inc., 2001.

[Dže03] Sašo Džeroski. Relational reinforcement learning for agents in worlds with objects. In *Adaptive Agents and MAS, LNAI 2636*, pages 306–322. Springer-Verlag Berlin Heidelberg, 2003.

[Ege91] Max Egenhofer. Reasoning about binary topological relations. In O. Gunther and H.-J. Schek, editors, *Second Symposium on Large Spatial Databases*, volume 525 of *Lecture Notes in Computer Science*, pages 143–160. Springer-Verlag, Zurich, Switzerland, August 1991.

[Elm90] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14:179–211, 1990.

[EW96] Werner Emde and Dietrich Wettschereck. Relational instance-based learning. In L. Saitta, editor, *Proceedings of the 13th International Conference on Machine Learning*. Morgan Kaufman, 1996.

[FFL05] Alexander Ferrein, Christian Fritz, and Gerhard Lakemeyer. Using golog for deliberation and team coordination in robotic soccer. *Künstliche Intelligenz*, 1:24–30, 2005.

[FGKP99] Nir Friedman, Lise Getoor, Daphne Koller, and Avi Pfeffer. Learning probabilistic relational models. In T. Dean, editor, *Proceedings of the Sixteenth International Joint Conferences on Artificial Intelligence (IJCAI99)*, pages 1300–1309, Stockholm, Sweden, 1999.

[FL99] Peter A. Flach and Nicolas Lachiche. 1BC: A first-order bayesian classifier. In *Proceedings of the 9th International Workshop on Inductive Logic Programming (ILP'99), LNAI 1634*, pages 92–103. Springer, 1999.

[FL01] Peter A. Flach and Nicolas Lachiche. Confirmation-guided discovery of first-order rules with Tertius. *Machine Learning*, 42(1-2), January 2001.

[FL04] Peter A. Flach and Nicolas Lachiche. Naive bayesian classification of structured data. *Machine Learning*, 57(3):233–269, December 2004.

[FN71] Richard E. Fikes and Nils J. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4):189–208, 1971.

[FPSS96] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery: An overview. In Usama M. Fayyad, Gregory Piatetsky-Shapiro, Padhraic Smyth, and Ramasamy Uthurusamy, editors, *Advances in Knowledge Discovery and Data Mining*, pages 1–34. MIT Press, 1996.

[Fre92a] Christian Freksa. Temporal reasoning based on semi-intervals. *Artificial Intelligence*, 54(1–2):199–227, 1992.

[Fre92b] Christian Freksa. Using orientation information for qualitative spatial reasoning. In A. U. Frank, I. Campari, and U. Formentini, editors, *Proceedings of the International Conference on Theories and Methods of Spatio-Temporal Reasoning in Geographic Space*, pages 162–178, 1992.

[FSW04] Gordon Fraser, Gerald Steinbauer, and Franz Wotawa. Application of qualitative reasoning to robotic soccer. In *18th International Workshop on Qualitative Reasoning*, Illinois, USA, August 2004.

[Geh05] Jan D. Gehrke. Qualitative Szenenrepräsentation für intelligente Fahrzeuge. Master's thesis, Department for Mathematics and Computer Science, Universität Bremen, 2005.

[GLH05] Jan D. Gehrke, Andreas D. Lattner, and Otthein Herzog. Qualitative mapping of sensory data for intelligent vehicles. In *Workshop on Agents in Real-Time and Dynamic Environments at the 19th International Joint Conference on Artificial Intelligence (IJCAI-05)*, pages 51–60, 2005.

[Got04] Björn Gottfried. Reasoning about intervals in two dimensions. In W. Thissen, P. Wieringa, M. Pantic, and M. Ludema, editors, *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, pages 5324–5332. IEEE Press, 2004.

[Got05] Björn Gottfried. *Shape from Positional-Contrast*. PhD thesis, Universität Bremen, 2005. Published in 2007, DUV - Deutscher Universitäts-Verlag, Wiesbaden, ISBN 978-3-8350-6070-8.

[GRS99] Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. SPIRIT: Sequential pattern mining with regular expression constraints. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99)*, pages 223–234, Edinburgh, Scotland, UK, September 1999.

[GW98] Bernhard Ganter and Rudolf Wille. Applied lattice theory: formal concept analysis. In *G.Grätzer: General lattice theory*, pages 591–605. Birkhäuser Verlag, Basel, 2nd edition, 1998.

[HWB01] Tamas Horváth, Stefan Wrobel, and Uta Bohnebeck. Relational instance-based learning with lists and terms. *Machine Learning*, 43(1/2):53–80, April 2001.

[HY06a] Yu Hirate and Hayato Yamana. Generalized sequential pattern mining with item intervals. *Journal of Computers (JCP)*, 1(3):51–60, June 2006.

[HY06b] Yu Hirate and Hayato Yamana. Sequential pattern mining with time intervals. In W.K. Ng and M. Kitsuregawa, editors, *PAKDD 2006*, volume 3918 of *Lecture Notes in Artificial Intelligence*, pages 775–779. Springer-Verlag Berlin Heidelberg, 2006.

[HYC03] Zhanxiang Huang, Yang Yang, and Xiaoping Chen. An approach to plan recognition and retrieval for multi-agent systems. In Mikhail Prokopenko, editor, *Workshop on Adaptability in Multi-Agent Systems, First RoboCup Australian Open 2003 (AORC-2003)*, Sydney, Australia, 2003. CSIRO.

[Höp01] Frank Höppner. Learning temporal rules from state sequences. In *Proceedings of the IJCAI'01 Workshop on Learning from Temporal and Spatial Data*, pages 25–31, Seattle, USA, 2001.

[Höp03] Frank Höppner. *Knowledge Discovery from Sequential Data*. PhD thesis, Technische Universität Braunschweig, 2003.

[JB01] Nico Jacobs and Hendrik Blockeel. From shell logs to shell scripts. In C. Rouveirol and M. Sebag, editors, *ILP 2001*, volume 2157 of *LNAI*, pages 80–90. Springer-Verlag Berlin Heidelberg, 2001.

[JMF99] Anil K. Jain, M. Narasimha Murty, and Patrick J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[Jor86] Michael I. Jordan. Attractor dynamics and parallelism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, pages 531–546, 1986. Cited in [Mit97].

[JW01] Glen Jeh and Jennifer Widom. SimRank: A measure of structural-context similarity. Technical report, Computer Science Department, Stanford University, 2001.

[Kal60] Rudolf E. Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME, Journal of Basic Engineering*, 82 (Series D):35–45, 1960.

[KBM97] Miroslav Kubat, Ivan Bratko, and Ryszard S. Michalski. A review of machine learning methods. In Ryszard S. Michalski, Ivan Bratko, and Miroslav Kubat, editors, *Machine Learning and Data Mining - Methods and Applications*. John Wiley & Sons Ltd, 1997.

[KCW06] Hye-Chung Kum, Joong Hyuk Chang, and Wei Wang. Sequential pattern mining in multi-databases via multiple alignment. *Data Mining and Knowledge Discovery*, 12:151–180, 2006.

[KFCV03] Gal Kaminka, Mehmet Fidanboylu, Allen Chang, and Manuela Veloso. Learning the sequential coordinated behavior of teams from observation. In Gal Kaminka, Pedro Lima, and Raul Rojas, editors,

RoboCup 2002: Robot Soccer World Cup VI, LNAI 2752, pages 111–125, Fukuoka, Japan, 2003.

[KH95] Krzysztof Koperski and Jiawei Han. Discovery of spatial association rules in geographic information databases. In *Proceedings of the 4th International Symposium on Advances in Spatial Databases, SSD*, pages 47–66, Portland, Maine, 1995.

[KLM96] Leslie P. Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, 4:237–285, 1996.

[KM05] Kenneth A. Kaufman and Ryszard S. Michalski. From data mining to knowledge mining. In C. R. Rao, J. L. Solka, and E. J. Wegman, editors, *Handbook in Statistics, Vol. 24: Data Mining and Data Visualization*, pages 47–75. Elsevier/North Holland, 2005.

[KO04] Marco Kögler and Oliver Obst. Simulation league: The next generation. In Daniel Polani, Andrea Bonarini, Brett Browning, and Kazuo Yoshida, editors, *RoboCup 2003: Robot Soccer World Cup VII*, volume 3020 of *Lecture Notes in Artificial Intelligence*, pages 458 – 469. Springer, Berlin, Heidelberg, New York, 2004.

[KZYC05] Ben Kao, Mighua Zhang, Chi-Lap Yip, and David W. Cheung. Efficient algorithms for mining and incremental update of maximal frequent sequences. *Data Mining and Knowledge Discovery*, 10:87–116, 2005.

[Lan96] Pat Langley. *Elements of Machine Learning*. Morgan Kaufmann Publishers, Inc., San Francisco, California, 1996.

[LD04] Sau Dan Lee and Luc De Raedt. Constraint based mining of first order sequences in SeqLog. In Rosa Meo, Pier Luca Lanzi, and Mika Klemettinen, editors, *Database Support for Data Mining Applications*, volume 2682 of *Lecture Notes in Computer Science*, pages 154–173. Springer Berlin / Heidelberg, 2004.

[Lee06] Sau Dan Lee. *Constrained Mining of Patterns in Large Databases*. PhD thesis, Albert-Ludwigs-Universität Freiburg, 2006.

[LF03] Nicolas Lachiche and Peter A. Flach. 1BC2: A true first-order bayesian classifier. In *Proceedings of the 12th International Conference on Inductive Logic Programming, LNAI 2583*, pages 133–148. Springer-Verlag, July 2003.

[LGTH05] Andreas D. Lattner, Jan D. Gehrke, Ingo J. Timm, and Otthein Herzog. A knowledge-based approach to behavior decision in intelligent vehicles. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV'05)*, pages 466–471, Las Vegas, USA, June 6-8 2005.

[LH06] Andreas D. Lattner and Otthein Herzog. Constraining the search space in temporal pattern mining. In Martin Schaaf and Klaus-Dieter Althoff, editors, *LWA 2006, Lernen - Wissensentdeckung - Adaptivität*, number 1/2006 in Hildesheimer Informatikberichte, pages 314–321, Hildesheim, Germany, October 11-13 2006.

[LL02] Ming-Yen Lin and Suh-Yin Lee. Fast discovery of sequential patterns by memory indexing. In *Proceedings of the 4th International Conference on Data Warehousing and Knowledge Discovery*, volume 2454 of *Lecture Notes in Computer Science*, pages 150–160, 2002.

[LM04] Francesca A. Lisi and Donato Malerba. Inducing multi-level association rules from multiple relations. *Machine Learning*, 55(2):175–210, May 2004.

[LMVH06] Andreas D. Lattner, Andrea Miene, Ubbo Visser, and Otthein Herzog. Sequential pattern mining for situation and behavior prediction in simulated robotic soccer. In Ansgar Bredenfeld, Adam Jacoff, Itsuki Noda, and Yasutake Takahashi, editors, *RoboCup-2005: Robot Soccer World Cup VIII*, volume 4020 of *Lecture Notes in Computer Science*, pages 118–129. Springer Verlag, Berlin, 2006.

[LRS+06] Andreas D. Lattner, Carsten Rachuy, Arne Stahlbock, Tobias Warden, and Ubbo Visser. Virtual Werder 3D team documentation 2006. Technical Report 36, TZI, Universität Bremen, August 2006.

[LS06] Srivatsan Laxman and P. S. Sastry. A survey of temporal data mining. *SADHANA - Academy Proceedings in Engineering Sciences*, 31(2):173–198, April 2006.

[LTLH05] Andreas D. Lattner, Ingo J. Timm, Martin Lorenz, and Otthein Herzog. Knowledge-based risk assessment for intelligent vehicles. In *Proceedings of the International Conference on Integration of Knowledge Intensive Multi-Agent Systems (KIMAS '05)*, pages 191–196, Waltham, MA, USA, April 18-21 2005.

[McG05] Ken McGarry. A survey of interestingness measures for knowledge discovery. *Knowledge Engineering Review*, 20(1):39–61, 2005.

[MCM83] Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors. *Machine Learning - An Artificial Intelligence Approach*. Tioga Publishing Company, Palo Alto, 1983.

[MCM86] Ryszard S. Michalski, Jaime G. Carbonell, and Tom M. Mitchell, editors. *Machine Learning - An Artificial Intelligence Approach (Volume II)*. Morgan Kaufmann Publishers, Inc., Los Altos, 1986.

[MF01] Stephen Muggleton and John Firth. CProgol4.4: A tutorial introduction. In Saso Dzeroski and Nada Lavrac, editors, *Relational Data Mining*. Springer-Verlag, September 2001.

[MH69] John McCarthy and Patrick J. Hayes. Some philosophical problems from the standpoint of artificial intelligence. In B. Meltzer and D. Mitchie, editors, *Machine Intelligence*, volume 4, pages 463–502. American Elsevier Publishing Co., Inc., 1969. Cited in [AF94].

[Mic69] Ryszard S. Michalski. On the quasi-minimal solution of the general covering problem. In *Proceedings of the V International Symposium on Information Processing (FCIP 69)*, volume A3, pages 125–128, Bled, Yugoslavia, October 8-11 1969.

[Mic73a] Ryszard S. Michalski. AQVAL/1: Computer implementation of a variable-valued logic system VL_1 and examples of its application to pattern recognition. In *Proc. International Joint Conference on Pattern Recognition*, pages 3–17, 1973.

[Mic73b] Ryszard S. Michalski. Discovering classification rules using variable-valued logic system VL_1. In *Proceedings of the 3rd International Conference on Artificial Intelligence, IJCAI*, pages 162 – 172, 1973.

[Mie04] Andrea Miene. *Räumlich-zeitliche Analyse von dynamischen Szenen*. Akademische Verlagsgesellschaft, Berlin, 2004. DISKI 279. PhD thesis, Universität Bremen.

[Mit97] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[MJ03] Cyrille Masson and François Jacquenet. Mining frequent logical sequences with SPIRIT-LOG. In S. Matwin and C. Sammut, editors, *ILP 2002*, volume 2583 of *Lecture Notes in Artificial Intelligence*, pages 166–182. Springer-Verlag Berlin Heidelberg, 2003.

[ML01] Donato Malerba and Francesca A. Lisi. An ILP method for spatial association rule mining. In *Working notes of the First Workshop on Multi-Relational Data Mining*, pages 18–29, Freiburg, Germany, 2001.

[MLVH04] Andrea Miene, Andreas D. Lattner, Ubbo Visser, and Otthein Herzog. Dynamic-preserving qualitative motion description for intelligent vehicles. In *Proceedings of the IEEE Intelligent Vehicles Symposium (IV '04)*, pages 642–646, June 14-17 2004.

[MN89] Marc Mézard and Jean-Pierre Nadal. Learning in feedforward layered networks: the tiling algorithm. *Journal of Physics*, 22:2191–2203, 1989.

[MR02] Artur Merke and Martin Riedmiller. Karlsruhe Brainstormers - a reinforcement learning way to robotic soccer. In *RoboCup-2001: Robot Soccer World Cup V*. Springer, Berlin, 2002.

[MTV97] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1:259–289, 1997.

[Mug95] Stephen Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13(3-4):245– 286, 1995.

[Mug97] Stephen Muggleton. Learning from positive data. In *Proceedings of the Sixth International Workshop on Inductive Logic Programming, LNAI 1314*. Springer-Verlag, 1997.

[Mus00] Alexandra Musto. *Qualitative Repräsentation von Bewegungsverläufen*. PhD thesis, TU München, 2000.

[MVH04] Andrea Miene, Ubbo Visser, and Otthein Herzog. Recognition and prediction of motion situations based on a qualitative motion description. In Daniel Polani, Brett Browning, Andrea Bonarini, and Kazuo Yoshida, editors, *RoboCup 2003: Robot Soccer World Cup VII*, volume 3020 of *Lecture Notes in Computer Science*, pages 77–88. Springer, 2004.

[PBTL98] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Pruning closed itemset lattices for association rules. In *Proceedings of the BDA French Conference on Advanced Databases*, October 1998.

[PHMA+01] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the 12th IEEE International Conference on Data Engineering*, pages 215–224, Heidelberg, Germany, April 2001.

[PHP+01] Helen Pinto, Jiawei Han, Jian Pei, Ke Wang, Qiming Chen, and Umeshwar Dayal. Multi-dimensional sequential pattern mining. In *Proceedings of the 10th International Conference on Information and Knowledge Management*, pages 81–88, Atlanta, Georgia, USA, 2001.

[Pin94] Javier A. Pinto. *Temporal Reasoning in the Situation Calculus*. PhD thesis, Department of Computer Science, University of Toronto, 1994.

[Pit97] Jim Pitman. Some probabilistic aspects of set partitions. *The American Mathematical Monthly*, 104(3):201–209, March 1997.

[PZOD99] Srinivasan Parthasarathy, Mohammed J. Zaki, Mitsunori Ogihara, and Sandhya Dwarkadas. Incremental and interactive sequence mining. In *Proceedings of the 8th International Conference on Information and Knowledge Management (CIKM'99)*, pages 251–258, Kansas City, MO, USA, November 1999.

[Qui90] J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990. Kluwer Academic Publishers, Boston.

[Qui93] J. Ross Quinlan. *C4.5 - Programs for Machine Learning*. Morgan Kaufmann Publishers, Inc., 1993.

[RCC92] David A. Randell, Zhan Cui, and Anthony G. Cohn. A spatial logic based on regions and connection. In Bernhard Nebel, Charles Rich, and William Swartout, editors, *Proceedings of the 3rd International Conference on Principles of Knowledge Representation and Reasoning*, pages 165–176. Morgan Kaufmann, 1992.

[Rei01] Raymond Reiter. *Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems*. Massachusetts Institute of Technology, 2001.

[RGH+06] Martin Riedmiller, Thomas Gabel, Roland Hafner, Sascha Lange, and Martin Lauer. Die Brainstormers: Entwurfsprinzipien lernfähiger autonomer Roboter. *Informatik-Spektrum*, 29(3):175–190, June 2006.

[RM02] Martin Riedmiller and Artur Merke. Using machine learning techniques in complex multi-agent domains. In *I. Stamatescu, W. Menzel, M. Richter and U. Ratsch (eds.): Perspectives on Adaptivity and Learning, LNCS, Springer*. 2002.

[RN94] Gavin A. Rummery and Mahesan Niranjan. On-line Q-learning using connectionist systems. Technical report, Engineering Department, Cambridge University, 1994.

[RN03] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach (Second Edition)*. Prentice Hall, Pearson Education, Inc., Upper Saddle River, New Jersey, 2003.

[ROME06] Markus Rollmann, Oliver Obst, Jan Murray, and Hesham Ebrahimi. *RoboCup Soccer Server 3D Manual*, July 2006.

[SA96] Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *5th International Conference on Extended Database Technology*, March 1996.

[SB98] Richard S. Sutton and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT Press, Cambridge, MA, 1998. Online version available at: `http://www.cs.ualberta.ca/~sutton/book/ebook/the-book.html`.

[Sch93] Christoph Schlieder. Representing visible locations for qualitative navigation. In Núria Piera Carreté and Madan G. Singh, editors, *Proceedings of the III IMACS International Workshop on Qualitative Reasoning and Decision Technologies, CIMNE*, pages 523–532, Barcelona, 1993.

[Sch96a] Karl H. Schäfer. *Unscharfe zeitlogische Modellierung von Situationen und Handlungen in Bildfolgenauswertung und Robotik*. PhD thesis, Universität Karlsruhe (TH), Juli 1996. Cited in [AN05].

[Sch96b] Christoph Schlieder. Ordering information and symbolic projection. In S.K. Chang, E. Jungert, and G. Tortora, editors, *Intelligent image database systems*, pages 115–140. World Scientific, Singapur, 1996.

[SDTB98] Ingo Schellhammer, Joachim Diederich, Michael Towsey, and Claudia Brugman. Knowledge extraction and recurrent neural networks: An analysis of an elman network trained on a natural learning task. In D. M. W. Powers, editor, *NeMLaP3/CoNLL98: New Methods in Language Processing and Computational Natural Language Learning*, pages 73–78, 1998.

[SDW03] Sumit Sanghai, Pedro Domingos, and Daniel Weld. Dynamic probabilistic relational models. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, Acapulco, Mexico, 2003.

[Seb02] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Computing Surveys*, 34(1):1–47, 2002.

[SG91] Padhraic Smyth and Rodney M. Goodman. Rule induction using information theory. In *Knowledge Discovery in Databases*, pages 159–176. MIT Press, 1991.

[Sim83] Herbert A. Simon. Why should machines learn? In Michalski et al. [MCM83], chapter 2, pages 25–37.

[SS01] Peter Stone and Richard S. Sutton. Scaling reinforcement learning toward RoboCup soccer. In *Proceedings of the 18th International Conference on Machine Learning*, 2001.

[SSK05] Peter Stone, Richard S. Sutton, and Gregory Kuhlmann. Scaling reinforcement learning toward RoboCup soccer. In *Adaptive Behavior*, 2005. To appear.

[SSW+06] Konstantinos Sagonas, Terrance Swift, David S. Warren, Juliana Freire, Prasad Rao, Baoqiu Cui, Ernie Johnson, Luis de Castro, Rui F. Marques, Steve Dawson, and Michael Kifer. *The XSB System Version 3.0 - Volume 1: Programmer's Manual, Volume 2: Libraries, Interfaces, and Packages*, July 26 2006.

[Ste03] Klaus Stein. *Qualitative Repräsentation und Generalisierung von Bewegungsverläufen*. PhD thesis, TU München, 2003.

[Stu04] Gerd Stumme. Iceberg query lattices for datalog. In *LWA 2004: Lernen – Wissensentdeckung – Adaptivität, GI-Workshopwoche*, pages 211–219, 2004.

[Sut88] Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, August 1988.

[TB05] Giridhar Tatavarty and Raj Bhatnagar. Finding temporal association rules between frequent patterns in multivariate time series. In *ICDM 2005 Workshop on Temporal Data Mining: Algorithms, Theory and Applications (TDM 2005)*, pages 127–136, Houston, Texas, USA, November 2005.

[Tes92] Gerald Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8(3-4):257–277, 1992.

[TKS02] Pang-Ning Tan, Vipin Kumar, and Jaideep Srivastava. Selecting the right interestingness measure for association patterns. In *Proceedings of the Eight International Conference on Knowledge Discovery and Data Mining (KDD'02)*, pages 32–41. ACM Press, 2002.

[TS93] Geoffrey G. Towell and Jude W. Shavlik. Extracting refined rules from knowledge-based neural networks. *Machine Learning*, 13(1):71–101, October 1993.

[TSN90] Geoffrey G. Towell, Jude W. Shavlik, and Michiel O. Noordewier. Refinement of approximate domain theories by knowledge-based neural networks. In *Proceedings of the Eight National Conference on Artificial Intelligence (AAAI-90)*, pages 861 – 866, Boston, MA, 1990.

[Wat89] Christopher J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge, England, 1989.

[WD92] Christopher J. C. H. Watkins and Peter Dayan. Technical note: Q-learning. *Machine Learning*, 8(3-4):279–292, May 1992.

[WJ95] Michael Wooldridge and Nicholas R. Jennings. Intelligent agents: Theory and practice. *Knowledge Engineering Review*, 10(2), 1995.

[Woj04] Janusz Wojtusiak. AQ21 user's guide. Technical Report MLI 04-3, Machine Learning and Inference Laboratory, George Mason University, Fairfax, VA, 2004 2004. (Updated in September 2005).

[Woo99] Michael Wooldridge. Intelligent agents. In Gerhard Weiss, editor, *Multiagent Systems – A Modern Approach to Distributed Artificial Intelligence*, pages 27–78. The MIT Press, 1999.

[WR05] Edi Winarko and John F. Roddick. Discovering richer temporal association rules from interval-based data: Extended report. Technical Report SIE-05-003, School of Informatics and Engineering, Flinders University, Adelaide, Australia, March 2005.

[Wro97] Stefan Wrobel. An algorithm for multi-relational discovery of subgroups. In J. Komorowski and J. Zytkow, editors, *PKKD'97*, pages 78–87. Springer-Verlag Berlin New York, 1997.

[WVH04] Thomas Wagner, Ubbo Visser, and Otthein Herzog. Egocentric qualitative spatial knowledge representation for physical robots. *Robotics and Autonomous Systems*, 49:25–42, 2004.

[YL99] Yiming Yang and Xin Liu. A re-examination of text categorization methods. In *Proceedings of the 22nd annual international ACM SIGIR conference on research and development in information retrieval*, pages 42–49. ACM Press, 1999.

[Zak01] Mohammed J. Zaki. SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42(1-2):31–60, 2001.

[ZB03] Qiankun Zhao and Sourav S. Bhowmick. Sequential pattern mining: A survey. Technical Report 2003118, CAIS, Nanyang Technological University, Singapore, 2003.

[Zel00] Andreas Zell. *Simulation neuronaler Netze*. R. Oldenbourg Verlag, München / Wien, 2000. 3rd edition; unchanged reprint of the book published by Addison Wesley Longman Verlag GmbH (1994).

[Zho04] Zhi-Hua Zhou. Rule extraction: Using neural networks or for neural networks? *Journal of Computer Science and Technology*, 19(4):249–253, March 2004.

[ZKCY02] Minghua Zhang, Ben Kao, David Cheung, and Chi-Lap Yip. Efficient algorithms for incremental update of frequent sequences. In *Proceedings of the 6th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*, pages 186–197, Taiwan, 2002.

[ZKYC01] Minghua Zhang, Ben Kao, Chi-Lap Yip, and David Cheung. A GSP-based efficient algorithm for mining frequent sequences. In *Proceedings of the 2001 International Conference on Artificial Intelligence (IC-AI 2001)*, Las Vegas, Nevada, USA, 2001.

[ZO98] Mohammed J. Zaki and Mitsunori Ogihara. Theoretical foundations of association rules. In *3rd SIGMOD'98 Workshop on Research Issues in Data Mining and Knowledge Discovery (DMKD)*, pages 71–78, Seattle, WA, June 1998.

[ZPOL97] Mohammed J. Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New algorithms for fast discovery of association rules. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 283–296, 1997.

Appendix A

Symbols

The following list gives an overview of the most important symbols used in this thesis:

$\bot$	Inconsistent pattern
$\hat{}$	Converse interval relation
$\circ$	Composition of interval relations
$\succeq$	Generalization relation for patterns ("more general than")
$\succ$	Proper generalization relation for patterns
$\succ_1$	Direct proper generalization relation for patterns
$<_{lex}$	Lexicographic order on identifiers
ϵ	Most general (empty) pattern
$\theta = \{X_1/t_1, \ldots, X_m/t_m\}$	Substitution
$\mathcal{CI}$	Set of concept identifiers
$cp = (ap_1, \ldots, ap_n)$	Conjunctive pattern
$\mathcal{CR}$	Concept restriction
$ds = (\mathcal{P}, \mathcal{O}, \textit{direct-instance-of}, dss)$	Dynamic scene
$dss = (\mathcal{CI}, \textit{is-a}, \mathcal{PT}, ir)$	Dynamic scene schema
$holds(pred(\ o_1, o_2, \ldots, o_n), \langle s, e\rangle)$	Predicate (alternative representation)
$\mathcal{ID}$	Set of identifiers
$\mathcal{IR}$	Set of interval relations
$\mathcal{IR}_{\leq}$	Set of interval relations with $s_1 \leq s_2$
$\mathcal{IR}_{\models}$	Set of interval relations with $s_1 = s_2$
$\mathcal{IR}_{older}$	Set of interval relations with $s_1 < s_2$
$ir(\langle s_1, e_1\rangle, \langle s_2, e_2\rangle)$	Interval relation function
$\mathcal{L}_{tp}$	Pattern language

$\mathcal{L}_{tp}$	Pattern language (for frequent and most special patterns)
$\mathcal{MS}$	Set of matches
$\mathcal{M}_i$	Single match
$\mathbb{N}$	Set of natural numbers
$\mathcal{O}$	Set of objects
$\mathcal{P}$	Set of predicates
$\mathcal{P}_{freq}$	Set of all most special frequent patterns
$p = (\text{pred}, o_1, o_2, \ldots, o_n, s, e, true)$	Predicate
$\mathcal{PI}$	Set of predicate identifiers
$pr = (tp_{prec}, tp_{cons})$	Prediction rule
$pt = (id, (ci_1, \ldots, ci_n))$	Predicate template
$\mathbb{R}$	Set of real numbers
ρ	Refinement operator
ρ_C	Concept refinement
ρ_I	Instantiation
ρ_L	Lengthening
ρ_T	Temporal refinement
ρ_U	Variable unification
$\dot{\rho}_x$	Non-redundant refinement operation
$\dot{\rho}_x^{-1}$	Inverse, non-redundant refinement operation
$\langle s_1, e_1 \rangle$	Interval
$\mathcal{SPS}$	Set of valid predicate sets
$\mathcal{T}$	Set of terms
$tp = (cp, \mathcal{TR}, \mathcal{CR})$	Temporal pattern
$tp_1 \sim tp_2$	Equivalence of temporal patterns
$[tp]$	Equivalence class of temporal pattern
$tp_{prec} \Rightarrow tp_{cons}$	Prediction rule (alternative representation)
$\mathcal{TR}$	Temporal restriction (for all predicate pairs)
$\mathcal{TR}_{i,j}$	Temporal restriction (between predicates with indices i and j)
$\mathcal{TR}_k$	Temporal restriction (between two predicates using a single index)
$\mathcal{V}$	Set of variables
$\vec{v}(tp) = (l, t, u, c, i)$	Refinement level vector

Appendix B

Evaluation Data on the DVD

The input and output files as well as the recorded console output of the different experiments can be found on the DVD provided with this thesis and are described in this part of the appendix. Printing the contents of all these files in this dissertation would take too much space. The basic structure of the DVD is as follows:

Base directory	Experiment
`/eval/simpleExample`	Simple Example
`/eval/synthetic`	Synthetic Data
`/eval/warmrComparison`	Comparison of *WARMR* and *MiTemP*
`/eval/robocup`	RoboCup Experiments

The following subsections describe the content of the DVD for the different experiments.

B.1 Simple Example

The simple example consists only of one folder with the following files:

File	Description
`createdPatterns.P`	Temporal patterns created by *MiTemP*
`createdPredRules.P`	All prediction rules created by *MiTemP*
`createdPredRulesSimpleExample.P`	Best prediciton rule used for prediction on unseen scene
`miTemP_output.txt`	Output of *MiTemP*
`simpleExample.P`	Dynamic scene input for learning
`simpleExampleTest.P`	Dynamic scene input for prediction
`testrun_output_test.txt`	Console output for prediction run
`testrun_output.txt`	Console output for learning

B.2 Synthetic Data

The experiments with the synthetic data are arranged in seven sub directories; one for each of the varied parameters:

Directory	Content
/eval/synthetic/minFreq	Varying minimal frequency
/eval/synthetic/numConcepts	Varying number of concepts
/eval/synthetic/numInstances	Varying number of instances
/eval/synthetic/patternSize	Varying pattern sizes
/eval/synthetic/predTempl	Varying number of predicate templates
/eval/synthetic/seqLength	Varying number of predicates
/eval/synthetic/windowSize	Varying window sizes

The principle structure of the single sub directories is similar for all seven sub experiments. They consist of a script for generating the random input data (createtestdata.sh), 70 files with dynamic scenes (ten random scenes for each of the seven parameters), and 70 corresponding files with the captured console output as well as the generated patterns (synthetic_<subexp>_<param>_<run>.txt and synthetic_<subexp>_<param>_<run>.P). The extracted CPU times, numbers of created patterns, and numbers of frequent patterns can be found in the files result_cpu_time.txt, result_created_patterns.txt, and result_freq_patterns.txt. The following tables give an overview of all files in the corresponding folders.

B.2.1 Varying Minimal Frequency

File	Description
createtestdata.sh	Script for random input data generation
miTemP_output_synthetic_minFreq_01_01.txt	*MiTemP* output, parameter 1, random data set 01
miTemP_output_synthetic_minFreq_01_02.txt	*MiTemP* output, parameter 1, random data set 02
...	
miTemP_output_synthetic_minFreq_01_10.txt	*MiTemP* output, parameter 1, random data set 10
miTemP_output_synthetic_minFreq_02_01.txt	*MiTemP* output, parameter 2, random data set 01
...	
miTemP_output_synthetic_minFreq_07_10.txt	*MiTemP* output, parameter 7, random data set 10
result_cpu_time.txt	Extracted CPU times for all runs
result_created_patterns.txt	Extracted number of patterns for all runs
result_freq_patterns.txt	Extracted number of frequent patterns for all runs
synthetic_minFreq_01_01.P	Input data, parameter 1, random data set 01
synthetic_minFreq_01_01.txt	Console output, parameter 1, random data set 01
synthetic_minFreq_01_02.P	Input data, parameter 1, random data set 02
synthetic_minFreq_01_02.txt	Console output, parameter 1, random data set 02
...	
synthetic_minFreq_01_10.P	Input data, parameter 1, random data set 10
synthetic_minFreq_01_10.txt	Console output, parameter 1, random data set 10
synthetic_minFreq_02_01.P	Input data, parameter 2, random data set 01
synthetic_minFreq_02_01.txt	Console output, parameter 2, random data set 01
...	
synthetic_minFreq_07_10.P	Input data, parameter 7, random data set 10
synthetic_minFreq_07_10.txt	Console output, parameter 7, random data set 10
tmp_mitempPatterns_synthetic_minFreq_01_01.P	Frequent patterns, parameter 1, random data set 01
tmp_mitempPatterns_synthetic_minFreq_01_02.P	Frequent patterns, parameter 1, random data set 02
...	
tmp_mitempPatterns_synthetic_minFreq_01_10.P	Frequent patterns, parameter 1, random data set 10
tmp_mitempPatterns_synthetic_minFreq_02_01.P	Frequent patterns, parameter 2, random data set 01
...	
tmp_mitempPatterns_synthetic_minFreq_07_10.P	Frequent patterns, parameter 7, random data set 10

B.2.2 Varying Number of Concepts

File	Description
createtestdata.sh	Script for random input data generation
miTemP_output_synthetic_numConcepts_01_01.txt	*MiTemP* output, parameter 1, random data set 01
miTemP_output_synthetic_numConcepts_01_02.txt	*MiTemP* output, parameter 1, random data set 02
...	
miTemP_output_synthetic_numConcepts_01_10.txt	*MiTemP* output, parameter 1, random data set 10
miTemP_output_synthetic_numConcepts_02_01.txt	*MiTemP* output, parameter 2, random data set 01
...	
miTemP_output_synthetic_numConcepts_07_10.txt	*MiTemP* output, parameter 7, random data set 10
result_cpu_time.txt	Extracted CPU times for all runs
result_created_patterns.txt	Extracted number of patterns for all runs
result_freq_patterns.txt	Extracted number of frequent patterns for all runs
synthetic_numConcepts_01_01.P	Input data, parameter 1, random data set 01
synthetic_numConcepts_01_01.txt	Console output, parameter 1, random data set 01
synthetic_numConcepts_01_02.P	Input data, parameter 1, random data set 02
synthetic_numConcepts_01_02.txt	Console output, parameter 1, random data set 02
...	
synthetic_numConcepts_01_10.P	Input data, parameter 1, random data set 10
synthetic_numConcepts_01_10.txt	Console output, parameter 1, random data set 10
synthetic_numConcepts_02_01.P	Input data, parameter 2, random data set 01
synthetic_numConcepts_02_01.txt	Console output, parameter 2, random data set 01
...	
synthetic_numConcepts_07_10.P	Input data, parameter 7, random data set 10
synthetic_numConcepts_07_10.txt	Console output, parameter 7, random data set 10
tmp_mitempPatterns_synthetic_numConcepts_01_01.P	Frequent patterns, parameter 1, random data set 01
tmp_mitempPatterns_synthetic_numConcepts_01_02.P	Frequent patterns, parameter 1, random data set 02
...	
tmp_mitempPatterns_synthetic_numConcepts_01_10.P	Frequent patterns, parameter 1, random data set 10
tmp_mitempPatterns_synthetic_numConcepts_02_01.P	Frequent patterns, parameter 2, random data set 01
...	
tmp_mitempPatterns_synthetic_numConcepts_07_10.P	Frequent patterns, parameter 7, random data set 10

B.2.3 Varying Number of Instances

File	Description
createtestdata.sh	Script for random input data generation
miTemP_output_synthetic_numInstances_01_01.txt	*MiTemP* output, parameter 1, random data set 01
miTemP_output_synthetic_numInstances_01_02.txt	*MiTemP* output, parameter 1, random data set 02
...	
miTemP_output_synthetic_numInstances_01_10.txt	*MiTemP* output, parameter 1, random data set 10
miTemP_output_synthetic_numInstances_02_01.txt	*MiTemP* output, parameter 2, random data set 01
...	
miTemP_output_synthetic_numInstances_07_10.txt	*MiTemP* output, parameter 7, random data set 10
result_cpu_time.txt	Extracted CPU times for all runs
result_created_patterns.txt	Extracted number of patterns for all runs
result_freq_patterns.txt	Extracted number of frequent patterns for all runs
synthetic_numInstances_01_01.P	Input data, parameter 1, random data set 01
synthetic_numInstances_01_01.txt	Console output, parameter 1, random data set 01
synthetic_numInstances_01_02.P	Input data, parameter 1, random data set 02
synthetic_numInstances_01_02.txt	Console output, parameter 1, random data set 02
...	
synthetic_numInstances_01_10.P	Input data, parameter 1, random data set 10
synthetic_numInstances_01_10.txt	Console output, parameter 1, random data set 10
synthetic_numInstances_02_01.P	Input data, parameter 2, random data set 01

synthetic_numInstances_02_01.txt	Console output, parameter 2, random data set 01
...	
synthetic_numInstances_07_10.P	Input data, parameter 7, random data set 10
synthetic_numInstances_07_10.txt	Console output, parameter 7, random data set 10
tmp_mitempPatterns_synthetic_numInstances_01_01.P	Frequent patterns, parameter 1, random data set 01
tmp_mitempPatterns_synthetic_numInstances_01_02.P	Frequent patterns, parameter 1, random data set 02
...	
tmp_mitempPatterns_synthetic_numInstances_01_10.P	Frequent patterns, parameter 1, random data set 10
tmp_mitempPatterns_synthetic_numInstances_02_01.P	Frequent patterns, parameter 2, random data set 01
...	
tmp_mitempPatterns_synthetic_numInstances_07_10.P	Frequent patterns, parameter 7, random data set 10

B.2.4 Varying Pattern Sizes

File	Description
createtestdata.sh	Script for random input data generation
miTemP_output_synthetic_patternSize_01_01.txt	*MiTemP* output, parameter 1, random data set 01
miTemP_output_synthetic_patternSize_01_02.txt	*MiTemP* output, parameter 1, random data set 02
...	
miTemP_output_synthetic_patternSize_01_10.txt	*MiTemP* output, parameter 1, random data set 10
miTemP_output_synthetic_patternSize_02_01.txt	*MiTemP* output, parameter 2, random data set 01
...	
miTemP_output_synthetic_patternSize_07_10.txt	*MiTemP* output, parameter 7, random data set 10
result_cpu_time.txt	Extracted CPU times for all runs
result_created_patterns.txt	Extracted number of patterns for all runs
result_freq_patterns.txt	Extracted number of frequent patterns for all runs
synthetic_patternSize_01_01.P	Input data, parameter 1, random data set 01
synthetic_patternSize_01_01.txt	Console output, parameter 1, random data set 01
synthetic_patternSize_01_02.P	Input data, parameter 1, random data set 02
synthetic_patternSize_01_02.txt	Console output, parameter 1, random data set 02
...	
synthetic_patternSize_01_10.P	Input data, parameter 1, random data set 10
synthetic_patternSize_01_10.txt	Console output, parameter 1, random data set 10
synthetic_patternSize_02_01.P	Input data, parameter 2, random data set 01
synthetic_patternSize_02_01.txt	Console output, parameter 2, random data set 01
...	
synthetic_patternSize_07_10.P	Input data, parameter 7, random data set 10
synthetic_patternSize_07_10.txt	Console output, parameter 7, random data set 10
tmp_mitempPatterns_synthetic_patternSize_01_01.P	Frequent patterns, parameter 1, random data set 01
tmp_mitempPatterns_synthetic_patternSize_01_02.P	Frequent patterns, parameter 1, random data set 02
...	
tmp_mitempPatterns_synthetic_patternSize_01_10.P	Frequent patterns, parameter 1, random data set 10
tmp_mitempPatterns_synthetic_patternSize_02_01.P	Frequent patterns, parameter 2, random data set 01
...	
tmp_mitempPatterns_synthetic_patternSize_07_10.P	Frequent patterns, parameter 7, random data set 10

B.2.5 Varying Number of Predicate Templates

File	Description
createtestdata.sh	Script for random input data generation
miTemP_output_synthetic_predTempl_01_01.txt	*MiTemP* output, parameter 1, random data set 01
miTemP_output_synthetic_predTempl_01_02.txt	*MiTemP* output, parameter 1, random data set 02
...	
miTemP_output_synthetic_predTempl_01_10.txt	*MiTemP* output, parameter 1, random data set 10

miTemP_output_synthetic_predTempl_02_01.txt	*MiTemP* output, parameter 2, random data set 01
...	
miTemP_output_synthetic_predTempl_07_10.txt	*MiTemP* output, parameter 7, random data set 10
result_cpu_time.txt	Extracted CPU times for all runs
result_created_patterns.txt	Extracted number of patterns for all runs
result_freq_patterns.txt	Extracted number of frequent patterns for all runs
synthetic_predTempl_01_01.P	Input data, parameter 1, random data set 01
synthetic_predTempl_01_01.txt	Console output, parameter 1, random data set 01
synthetic_predTempl_01_02.P	Input data, parameter 1, random data set 02
synthetic_predTempl_01_02.txt	Console output, parameter 1, random data set 02
...	
synthetic_predTempl_01_10.P	Input data, parameter 1, random data set 10
synthetic_predTempl_01_10.txt	Console output, parameter 1, random data set 10
synthetic_predTempl_02_01.P	Input data, parameter 2, random data set 01
synthetic_predTempl_02_01.txt	Console output, parameter 2, random data set 01
...	
synthetic_predTempl_07_10.P	Input data, parameter 7, random data set 10
synthetic_predTempl_07_10.txt	Console output, parameter 7, random data set 10
tmp_mitempPatterns_synthetic_predTempl_01_01.P	Frequent patterns, parameter 1, random data set 01
tmp_mitempPatterns_synthetic_predTempl_01_02.P	Frequent patterns, parameter 1, random data set 02
...	
tmp_mitempPatterns_synthetic_predTempl_01_10.P	Frequent patterns, parameter 1, random data set 10
tmp_mitempPatterns_synthetic_predTempl_02_01.P	Frequent patterns, parameter 2, random data set 01
...	
tmp_mitempPatterns_synthetic_predTempl_07_10.P	Frequent patterns, parameter 7, random data set 10

B.2.6 Varying Number of Predicates

File	Description
createtestdata.sh	Script for random input data generation
miTemP_output_synthetic_seqLength_01_01.txt	*MiTemP* output, parameter 1, random data set 01
miTemP_output_synthetic_seqLength_01_02.txt	*MiTemP* output, parameter 1, random data set 02
...	
miTemP_output_synthetic_seqLength_01_10.txt	*MiTemP* output, parameter 1, random data set 10
miTemP_output_synthetic_seqLength_02_01.txt	*MiTemP* output, parameter 2, random data set 01
...	
miTemP_output_synthetic_seqLength_07_10.txt	*MiTemP* output, parameter 7, random data set 10
result_cpu_time.txt	Extracted CPU times for all runs
result_created_patterns.txt	Extracted number of patterns for all runs
result_freq_patterns.txt	Extracted number of frequent patterns for all runs
synthetic_seqLength_01_01.P	Input data, parameter 1, random data set 01
synthetic_seqLength_01_01.txt	Console output, parameter 1, random data set 01
synthetic_seqLength_01_02.P	Input data, parameter 1, random data set 02
synthetic_seqLength_01_02.txt	Console output, parameter 1, random data set 02
...	
synthetic_seqLength_01_10.P	Input data, parameter 1, random data set 10
synthetic_seqLength_01_10.txt	Console output, parameter 1, random data set 10
synthetic_seqLength_02_01.P	Input data, parameter 2, random data set 01
synthetic_seqLength_02_01.txt	Console output, parameter 2, random data set 01
...	
synthetic_seqLength_07_10.P	Input data, parameter 7, random data set 10
synthetic_seqLength_07_10.txt	Console output, parameter 7, random data set 10
tmp_mitempPatterns_synthetic_seqLength_01_01.P	Frequent patterns, parameter 1, random data set 01
tmp_mitempPatterns_synthetic_seqLength_01_02.P	Frequent patterns, parameter 1, random data set 02
...	
tmp_mitempPatterns_synthetic_seqLength_01_10.P	Frequent patterns, parameter 1, random data set 10
tmp_mitempPatterns_synthetic_seqLength_02_01.P	Frequent patterns, parameter 2, random data set 01

...	
tmp_mitempPatterns_synthetic_seqLength_07_10.P	Frequent patterns, parameter 7, random data set 10

B.2.7 Varying Window Size

File	Description
createtestdata.sh	Script for random input data generation
miTemP_output_synthetic_windowSize_01_01.txt	*MiTemP* output, parameter 1, random data set 01
miTemP_output_synthetic_windowSize_01_02.txt	*MiTemP* output, parameter 1, random data set 02
...	
miTemP_output_synthetic_windowSize_01_10.txt	*MiTemP* output, parameter 1, random data set 10
miTemP_output_synthetic_windowSize_02_01.txt	*MiTemP* output, parameter 2, random data set 01
...	
miTemP_output_synthetic_windowSize_07_10.txt	*MiTemP* output, parameter 7, random data set 10
result_cpu_time.txt	Extracted CPU times for all runs
result_created_patterns.txt	Extracted number of patterns for all runs
result_freq_patterns.txt	Extracted number of frequent patterns for all runs
synthetic_windowSize_01_01.P	Input data, parameter 1, random data set 01
synthetic_windowSize_01_01.txt	Console output, parameter 1, random data set 01
synthetic_windowSize_01_02.P	Input data, parameter 1, random data set 02
synthetic_windowSize_01_02.txt	Console output, parameter 1, random data set 02
...	
synthetic_windowSize_01_10.P	Input data, parameter 1, random data set 10
synthetic_windowSize_01_10.txt	Console output, parameter 1, random data set 10
synthetic_windowSize_02_01.P	Input data, parameter 2, random data set 01
synthetic_windowSize_02_01.txt	Console output, parameter 2, random data set 01
...	
synthetic_windowSize_07_10.P	Input data, parameter 7, random data set 10
synthetic_windowSize_07_10.txt	Console output, parameter 7, random data set 10
tmp_mitempPatterns_synthetic_windowSize_01_01.P	Frequent patterns, parameter 1, random data set 01
tmp_mitempPatterns_synthetic_windowSize_01_02.P	Frequent patterns, parameter 1, random data set 02
...	
tmp_mitempPatterns_synthetic_windowSize_01_10.P	Frequent patterns, parameter 1, random data set 10
tmp_mitempPatterns_synthetic_windowSize_02_01.P	Frequent patterns, parameter 2, random data set 01
...	
tmp_mitempPatterns_synthetic_windowSize_07_10.P	Frequent patterns, parameter 7, random data set 10

B.3 Comparison of WARMR and MiTemP

The input and output of the experiment for the comparison of *WARMR* and *MiTemP* is divided into six directories; for each refinement level a separate directory:

Directory	Description
/eval/warmrComparison/level02	Maximal refinement level 2
/eval/warmrComparison/level03	Maximal refinement level 3
/eval/warmrComparison/level04	Maximal refinement level 4
/eval/warmrComparison/level05	Maximal refinement level 5
/eval/warmrComparison/level06	Maximal refinement level 6
/eval/warmrComparison/level07	Maximal refinement level 7

The structure of the six directories is identical. Therefore, the principle structure is described only once in the following table:

File	Description
example_small.P	Input dynamic scene in *MiTemP* format
experiment.bg	Automatically created *ACE* background knowledge file
experiment.kb	Automatically created *ACE* knowledge base file
experiment.s	Automatically created *ACE* settings file
miTemP_output.txt	*MiTemP* output
run_ace_result.txt	Console output of *WARMR* run
run_xsb_result_compare.txt	Console output of comparison of *WARMR* and *MiTemP*
run_xsb_result.txt	Console output of *MiTemP* run
tmp_mitempPatterns.P	Patters created by *MiTemP*
tmp_warmrPatterns.P	Patterns created by *WARMR* (*MiTemP* format for comparison)
tmp_warmrPatterns_var.P	Patterns created by *WARMR* (intermediate format without anonymous variables)
warmr3/experiment.freq_queries.out	Patterns created by *WARMR* (*ACE* output)

B.4 RoboCup Experiments

The RoboCup experiments are divided into two sub directories, one for the experiments with the 2D simulation league matches (`/eval/robocup/2d`) and the other for the experiments with the 3D simulation league matches (`/eval/robocup/3d`). In both cases – 2D as well as 3D – three different runs have been performed for all different matches.

B.5 2D Simulation League

In the 2D simulation league experiments the two matches have been divided into four dynamic scenes (each consists of one half of a match). These dynamic scenes can be found in the base directory of the 2D experiments. Furthermore, in six sub directories the results of the three different runs with two different training scenes (Match 1-1, Match 2-1):

File	Description
match_2d_1_1.P	*MiTemP* input (match 1, half 1)
match_2d_1_2.P	*MiTemP* input (match 1, half 2)
match_2d_2_1.P	*MiTemP* input (match 2, half 1)
match_2d_2_2.P	*MiTemP* input (match 2, half 2)
run1/exp1/	Results of run 1 (Training: Match 1-1)
run1/exp2/	Results of run 1 (Training: Match 2-1)
run2/exp1/	Results of run 2 (Training: Match 1-1)
run2/exp2/	Results of run 2 (Training: Match 2-1)
run3/exp1/	Results of run 3 (Training: Match 1-1)
run3/exp2/	Results of run 3 (Training: Match 2-1)

All six sub directories consist of the following files:

File	Description
`averagesAll.txt`	Average prediction accuracies on the different dynamic scenes
`createdPatterns.P`	All created patterns
`createdPredRules.P`	Created prediction rules
`miTemP_output.txt`	*MiTemP* output while pattern mining
`runExp_applyAll_1_1.txt`	Results of prediction rule application for Match 1-1
`runExp_applyAll_1_2.txt`	Results of prediction rule application for Match 1-2
`runExp_applyAll_2_1.txt`	Results of prediction rule application for Match 2-1
`runExp_applyAll_2_2.txt`	Results of prediction rule application for Match 2-2
`runExp_trainPred.txt`	Console output of prediction rule generation
`runExp_train.txt`	Console output of pattern mining

B.6 3D Simulation League

The experiments with the 3D simulation league matches consist of five different matches which have been divided into ten dynamic scenes (one for each half). Each of the five first halves have been used for training and all experiments have been repeated three times (`run1`, `run2`, and `run3`):

File	Description
`match_3d_1_1.P`	*MiTemP* input (match 1, half 1)
`match_3d_1_2.P`	*MiTemP* input (match 1, half 2)
`match_3d_2_1.P`	*MiTemP* input (match 2, half 1)
`match_3d_2_2.P`	*MiTemP* input (match 2, half 2)
`match_3d_3_1.P`	*MiTemP* input (match 3, half 1)
`match_3d_3_2.P`	*MiTemP* input (match 3, half 2)
`match_3d_4_1.P`	*MiTemP* input (match 4, half 1)
`match_3d_4_2.P`	*MiTemP* input (match 4, half 2)
`match_3d_5_1.P`	*MiTemP* input (match 5, half 1)
`match_3d_5_2.P`	*MiTemP* input (match 5, half 2)
`run1/exp1/`	Results of run 1 (Training: Match 1-1)
`run1/exp2/`	Results of run 1 (Training: Match 2-1)
`run1/exp3/`	Results of run 1 (Training: Match 3-1)
`run1/exp4/`	Results of run 1 (Training: Match 4-1)
`run1/exp5/`	Results of run 1 (Training: Match 5-1)
`run2/exp1/`	Results of run 2 (Training: Match 1-1)
`run2/exp2/`	Results of run 2 (Training: Match 2-1)
`run2/exp3/`	Results of run 2 (Training: Match 3-1)
`run2/exp4/`	Results of run 2 (Training: Match 4-1)
`run2/exp5/`	Results of run 2 (Training: Match 5-1)
`run3/exp1/`	Results of run 3 (Training: Match 1-1)
`run3/exp2/`	Results of run 3 (Training: Match 2-1)
`run3/exp3/`	Results of run 3 (Training: Match 3-1)
`run3/exp4/`	Results of run 3 (Training: Match 4-1)
`run3/exp5/`	Results of run 3 (Training: Match 5-1)

The directory with the results of the first run and the first experiment differs from the remaining ones as in this case the experiments with varying maximal refinement levels have been performed additionally:

File	Description
averageLevel08.txt	Average prediction accuracies (refinement level 08)
averageLevel09.txt	Average prediction accuracies (refinement level 09)
averageLevel10.txt	Average prediction accuracies (refinement level 10)
averageLevel11.txt	Average prediction accuracies (refinement level 11)
averageLevel12.txt	Average prediction accuracies (refinement level 12)
averagesAll.txt	Average prediction accuracies (all refinement levels)
bestCreatedPredRules.P	Best characteristic prediction rules
bestCreatedPredRulesPosNeg.P	Best general prediction rules
createdPatterns.P	All created patterns (all ref. levels)
createdPredRules08.P	Created prediction rules (refinement level 08)
createdPredRules09.P	Created prediction rules (refinement level 09)
createdPredRules10.P	Created prediction rules (refinement level 10)
createdPredRules11.P	Created prediction rules (refinement level 11)
createdPredRules12.P	Created prediction rules (refinement level 12)
createdPredRules.P	All created prediction rules (all ref. levels)
miTemP_output.txt	*MiTemP* output
qual_match_3d_1_1.P	Prediction accuracies of all rules for Match 1-1
qual_match_3d_1_2.P	Prediction accuracies of all rules for Match 1-2
qual_match_3d_2_1.P	Prediction accuracies of all rules for Match 2-1
qual_match_3d_2_2.P	Prediction accuracies of all rules for Match 2-2
qual_match_3d_3_1.P	Prediction accuracies of all rules for Match 3-1
qual_match_3d_3_2.P	Prediction accuracies of all rules for Match 3-2
qual_match_3d_4_1.P	Prediction accuracies of all rules for Match 4-1
qual_match_3d_4_2.P	Prediction accuracies of all rules for Match 4-2
qual_match_3d_5_1.P	Prediction accuracies of all rules for Match 5-1
qual_match_3d_5_2.P	Prediction accuracies of all rules for Match 5-2
runExp_applyAll_1_1_level08.txt	Console output for prediction rule application (Match 1-1, ref. level 08)
runExp_applyAll_1_1_level09.txt	Console output for prediction rule application (Match 1-1, ref. level 09)
runExp_applyAll_1_1_level10.txt	Console output for prediction rule application (Match 1-1, ref. level 10)
runExp_applyAll_1_1_level11.txt	Console output for prediction rule application (Match 1-1, ref. level 11)
runExp_applyAll_1_1_level12.txt	Console output for prediction rule application (Match 1-1, ref. level 12)
runExp_applyAll_1_1.txt	Console output for prediction rule application (Match 1-1, all ref. levels)
runExp_applyAll_1_2_level08.txt	Console output for prediction rule application (Match 1-2, ref. level 08)
runExp_applyAll_1_2_level09.txt	Console output for prediction rule application (Match 1-2, ref. level 09)
runExp_applyAll_1_2_level10.txt	Console output for prediction rule application (Match 1-2, ref. level 10)
runExp_applyAll_1_2_level11.txt	Console output for prediction rule application (Match 1-2, ref. level 11)
runExp_applyAll_1_2_level12.txt	Console output for prediction rule application (Match 1-2, ref. level 12)
runExp_applyAll_1_2.txt	Console output for prediction rule application (Match 1-2, all ref. levels)
runExp_applyAll_2_1_level08.txt	Console output for prediction rule application (Match 2-1, ref. level 08)
runExp_applyAll_2_1_level09.txt	Console output for prediction rule application (Match 2-1, ref. level 09)
runExp_applyAll_2_1_level10.txt	Console output for prediction rule application (Match 2-1, ref. level 10)
runExp_applyAll_2_1_level11.txt	Console output for prediction rule application (Match 2-1, ref. level 11)
runExp_applyAll_2_1_level12.txt	Console output for prediction rule application (Match 2-1, ref. level 12)
runExp_applyAll_2_1.txt	Console output for prediction rule application (Match 2-1, all ref. levels)
runExp_applyAll_2_2_level08.txt	Console output for prediction rule application (Match 2-2, ref. level 08)
runExp_applyAll_2_2_level09.txt	Console output for prediction rule application (Match 2-2, ref. level 09)
runExp_applyAll_2_2_level10.txt	Console output for prediction rule application (Match 2-2, ref. level 10)
runExp_applyAll_2_2_level11.txt	Console output for prediction rule application (Match 2-2, ref. level 11)
runExp_applyAll_2_2_level12.txt	Console output for prediction rule application (Match 2-2, ref. level 12)
runExp_applyAll_2_2.txt	Console output for prediction rule application (Match 2-2, all ref. levels)
runExp_applyAll_3_1_level08.txt	Console output for prediction rule application (Match 3-1, ref. level 08)
runExp_applyAll_3_1_level09.txt	Console output for prediction rule application (Match 3-1, ref. level 09)
runExp_applyAll_3_1_level10.txt	Console output for prediction rule application (Match 3-1, ref. level 10)
runExp_applyAll_3_1_level11.txt	Console output for prediction rule application (Match 3-1, ref. level 11)
runExp_applyAll_3_1_level12.txt	Console output for prediction rule application (Match 3-1, ref. level 12)
runExp_applyAll_3_1.txt	Console output for prediction rule application (Match 3-1, all ref. levels)
runExp_applyAll_3_2_level08.txt	Console output for prediction rule application (Match 3-2, ref. level 08)

runExp_applyAll_3_2_level09.txt	Console output for prediction rule application (Match 3-2, ref. level 09)
runExp_applyAll_3_2_level10.txt	Console output for prediction rule application (Match 3-2, ref. level 10)
runExp_applyAll_3_2_level11.txt	Console output for prediction rule application (Match 3-2, ref. level 11)
runExp_applyAll_3_2_level12.txt	Console output for prediction rule application (Match 3-2, ref. level 12)
runExp_applyAll_3_2.txt	Console output for prediction rule application (Match 3-2, all ref. levels)
runExp_applyAll_4_1_level08.txt	Console output for prediction rule application (Match 4-1, ref. level 08)
runExp_applyAll_4_1_level09.txt	Console output for prediction rule application (Match 4-1, ref. level 09)
runExp_applyAll_4_1_level10.txt	Console output for prediction rule application (Match 4-1, ref. level 10)
runExp_applyAll_4_1_level11.txt	Console output for prediction rule application (Match 4-1, ref. level 11)
runExp_applyAll_4_1_level12.txt	Console output for prediction rule application (Match 4-1, ref. level 12)
runExp_applyAll_4_1.txt	Console output for prediction rule application (Match 4-1, all ref. levels)
runExp_applyAll_4_2_level08.txt	Console output for prediction rule application (Match 4-2, ref. level 08)
runExp_applyAll_4_2_level09.txt	Console output for prediction rule application (Match 4-2, ref. level 09)
runExp_applyAll_4_2_level10.txt	Console output for prediction rule application (Match 4-2, ref. level 10)
runExp_applyAll_4_2_level11.txt	Console output for prediction rule application (Match 4-2, ref. level 11)
runExp_applyAll_4_2_level12.txt	Console output for prediction rule application (Match 4-2, ref. level 12)
runExp_applyAll_4_2.txt	Console output for prediction rule application (Match 4-2, all ref. levels)
runExp_applyAll_5_1_level08.txt	Console output for prediction rule application (Match 5-1, ref. level 08)
runExp_applyAll_5_1_level09.txt	Console output for prediction rule application (Match 5-1, ref. level 09)
runExp_applyAll_5_1_level10.txt	Console output for prediction rule application (Match 5-1, ref. level 10)
runExp_applyAll_5_1_level11.txt	Console output for prediction rule application (Match 5-1, ref. level 11)
runExp_applyAll_5_1_level12.txt	Console output for prediction rule application (Match 5-1, ref. level 12)
runExp_applyAll_5_1.txt	Console output for prediction rule application (Match 5-1, all ref. levels)
runExp_applyAll_5_2_level08.txt	Console output for prediction rule application (Match 5-2, ref. level 08)
runExp_applyAll_5_2_level09.txt	Console output for prediction rule application (Match 5-2, ref. level 09)
runExp_applyAll_5_2_level10.txt	Console output for prediction rule application (Match 5-2, ref. level 10)
runExp_applyAll_5_2_level11.txt	Console output for prediction rule application (Match 5-2, ref. level 11)
runExp_applyAll_5_2_level12.txt	Console output for prediction rule application (Match 5-2, ref. level 12)
runExp_applyAll_5_2.txt	Console output for prediction rule application (Match 5-2, all ref. levels)
runExp_applyGood_1_1.txt	Console output of characteristic rule application (Match 1-1)
runExp_applyGood_1_2.txt	Console output of characteristic rule application (Match 1-2)
runExp_applyGood_2_1.txt	Console output of characteristic rule application (Match 2-1)
runExp_applyGood_2_2.txt	Console output of characteristic rule application (Match 2-2)
runExp_applyGood_3_1.txt	Console output of characteristic rule application (Match 3-1)
runExp_applyGood_3_2.txt	Console output of characteristic rule application (Match 3-2)
runExp_applyGood_4_1.txt	Console output of characteristic rule application (Match 4-1)
runExp_applyGood_4_2.txt	Console output of characteristic rule application (Match 4-2)
runExp_applyGood_5_1.txt	Console output of characteristic rule application (Match 5-1)
runExp_applyGood_5_2.txt	Console output of characteristic rule application (Match 5-2)
runExp_applyGoodPosNeg_1_1.txt	Console output of general rule application (Match 1-1)
runExp_applyGoodPosNeg_1_2.txt	Console output of general rule application (Match 1-2)
runExp_applyGoodPosNeg_2_1.txt	Console output of general rule application (Match 2-1)
runExp_applyGoodPosNeg_2_2.txt	Console output of general rule application (Match 2-2)
runExp_applyGoodPosNeg_3_1.txt	Console output of general rule application (Match 3-1)
runExp_applyGoodPosNeg_3_2.txt	Console output of general rule application (Match 3-2)
runExp_applyGoodPosNeg_4_1.txt	Console output of general rule application (Match 4-1)
runExp_applyGoodPosNeg_4_2.txt	Console output of general rule application (Match 4-2)
runExp_applyGoodPosNeg_5_1.txt	Console output of general rule application (Match 5-1)
runExp_applyGoodPosNeg_5_2.txt	Console output of general rule application (Match 5-2)
runExp_qualFactorPosNeg.txt	Console output for computation of best general rules
runExp_qualFactor.txt	Console output for computation of best characteristic rules
runExp_qual_match_3d_1_1.txt	Console output for computation of prediction accuracies (Match 1-1)
runExp_qual_match_3d_1_2.txt	Console output for computation of prediction accuracies (Match 1-2)
runExp_qual_match_3d_2_1.txt	Console output for computation of prediction accuracies (Match 2-1)
runExp_qual_match_3d_2_2.txt	Console output for computation of prediction accuracies (Match 2-2)
runExp_qual_match_3d_3_1.txt	Console output for computation of prediction accuracies (Match 3-1)
runExp_qual_match_3d_3_2.txt	Console output for computation of prediction accuracies (Match 3-2)
runExp_qual_match_3d_4_1.txt	Console output for computation of prediction accuracies (Match 4-1)
runExp_qual_match_3d_4_2.txt	Console output for computation of prediction accuracies (Match 4-2)

runExp_qual_match_3d_5_1.txt	Console output for computation of prediction accuracies (Match 5-1)
runExp_qual_match_3d_5_2.txt	Console output for computation of prediction accuracies (Match 5-2)
runExp_trainPred08.txt	Console output of prediction rule generation (refinement level 08)
runExp_trainPred09.txt	Console output of prediction rule generation (refinement level 09)
runExp_trainPred10.txt	Console output of prediction rule generation (refinement level 10)
runExp_trainPred11.txt	Console output of prediction rule generation (refinement level 11)
runExp_trainPred12.txt	Console output of prediction rule generation (refinement level 12)
runExp_trainPred.txt	Console output of prediction rule generation (all ref. levels)
runExp_train.txt	Console output of pattern mining (all ref. levels)

The remaining 14 directories (experiment 2-5 of run 1 and all experiments of runs 2 and 3) consist of the same files:

File	Description
averagesAll.txt	Average prediction accuracies (all refinement levels)
bestCreatedPredRules.P	Best characteristic prediction rules
bestCreatedPredRulesPosNeg.P	Best general prediction rules
createdPatterns.P	All created patterns (all ref. levels)
createdPredRules.P	All created prediction rules (all ref. levels)
miTemP_output.txt	*MiTemP* output
runExp_applyAll_1_1.txt	Console output for prediction rule application (Match 1-1, all ref. levels)
runExp_applyAll_1_2.txt	Console output for prediction rule application (Match 1-2, all ref. levels)
runExp_applyAll_2_1.txt	Console output for prediction rule application (Match 2-1, all ref. levels)
runExp_applyAll_2_2.txt	Console output for prediction rule application (Match 2-2, all ref. levels)
runExp_applyAll_3_1.txt	Console output for prediction rule application (Match 3-1, all ref. levels)
runExp_applyAll_3_2.txt	Console output for prediction rule application (Match 3-2, all ref. levels)
runExp_applyAll_4_1.txt	Console output for prediction rule application (Match 4-1, all ref. levels)
runExp_applyAll_4_2.txt	Console output for prediction rule application (Match 4-2, all ref. levels)
runExp_applyAll_5_1.txt	Console output for prediction rule application (Match 5-1, all ref. levels)
runExp_applyAll_5_2.txt	Console output for prediction rule application (Match 5-2, all ref. levels)
runExp_trainPred.txt	Console output of prediction rule generation (all ref. levels)
runExp_train.txt	Console output of pattern mining (all ref. levels)

Index