RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN
Fakultät für Mathematik, Informatik und Naturwissenschaften
Lehr- und Forschungsgebiet Informatik V - Wissensbasierte Systeme

---

READYWORLD

A Qualitative World Model
for Autonomous Soccer Agents
in the READYLOG Framework

Diplomarbeit

von
Stefan Schiffer

vorgelegt im Fach Informatik
im Sommer 2005

1st Supervisor:	Prof. Gerhard Lakemeyer, Ph. D.
2nd Supervisor:	Prof. Dr. Thomas Seidl
Advisor:	Dipl. Inform. Alexander Ferrein

Hiermit versichere ich, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet sowie Zitate kenntlich gemacht habe.

Aachen, August 2005 .....

(Stefan Schiffer)

Acknowledgments

First of all, I would like to express my gratitude to Alexander Ferrein for giving me the opportunity to integrate a qualitative world model into READYLOG and for his valuable suggestions, comments, and advice. Special thanks goes to Stefan Jacobs who answered a lot of my questions concerning READYLOG and who also gave me very useful tips and tricks during the design and the implementation phase. Additionally, I thank the ALLEMANIACS ROBOCUP team for their support and their comments. They gave me the opportunity to evaluate parts of my work at the German Open 2005 in Paderborn. Finally, I would like to thank Shangning Postel and my family who encouraged me during the last few years, in general, and during this thesis, in particular.

Contents

List of Figures

Chapter 1

Introduction

In this diploma thesis we develop a qualitative world model for autonomous soccer agents. This qualitative world model provides abstractions from numerical data to more general concepts which derive from human soccer theories. As these theories naturally evolve from human cognition facilities, this qualitative world model not only supports general human soccer concepts but it is also meant to ease agent specifications for its designer.

1.1 General Background

In this thesis we are concerned with autonomous agents. These agents are supposed to play soccer in the ROBOCUP domain. The ROBOCUP domain provides a testbed with a wide range of standard problems autonomous agents have to deal with. Playing soccer is a high-level task which incorporates several low-level skills, namely basic abilities like localization, collision-avoidance, and motion planning.
Since the agents we are dealing with are able to perform basic abilities quite reliably the important question how to specify an agent's behavior upon these abilities arises. This means, in the context of ROBOCUP we have to find a way to let one or more agents play soccer successfully. This thesis is carried out in the context of a hybrid approach to behavior specification using the logical programming language READYLOG [Fri03] which integrates programming and decision-theoretic planning.

1.2 Motivation

We already stated that we deal with autonomous agents, that is agents that act on their own. Nonetheless, it is still the designer of an autonomous agent who initially specifies the behavior exhibited by the agent. Usually, the designer has expertise in the task the agent should accomplish. This expertise consists of valuable information on how to use basic abilities to achieve a particular task. In order to make use of this knowledge the designer has to express it in such a way that an agent system can cope with it.
The autonomous agents we are concerned with are mainly mobile robots. These robots acquire information about their environment through sensors. The systems of soccer agents, in general, and the systems of mobile robots, in particular, are characterized by a high amount of numerical values. That is to say, the information acquired by sensors as well as instructions given to the robot's execution system are represented, stored, and processed in form of numerical data.

The designer's knowledge, however, is most often not available in form of numerical values but it is more abstract in nature. For this reason, a designer who is trying to transfer his expertise to the specification of an autonomous agent has to transform his domain knowledge into numerical data an agent can process. This transformation is always an inconvenient and time-consuming process as the designer has to do it by hand. We want to overcome this inconvenience in this thesis.
Therefore we decided to take a closer look at human soccer. Generally, the success of a human soccer team largely depends on the individual abilities of each player as well as on the overall strategy constituting the tactical team behavior. Looking back on robotic soccer we ask ourselves how we can make it possible to take advantage of strategies and tactics already developed in human soccer more easily.
As stated in [DFL+05] we can obtain a formalization of human soccer strategies. These strategies are described in an abstract fashion, incorporating diagrams which show basic ideas of tactical moves by a set of examples. These figures are accompanied by textual information which often only provide fuzzy concepts and vague descriptions. Spatial relations, for instance, which are essential in soccer are denoted by terms like 'left', 'front', and 'far'. From now on, these things will be referred to as *qualitative* information. To be able to adapt descriptions of soccer moves which can be found in current soccer literature it seems reasonable to have similar descriptions within the system of an agent. That is, an agent system should be able to cope with the same vague descriptions and abstract representations. Moreover, it should also be able to represent the information about its environment in terms of such qualitative notions.
We use a hybrid approach to behavior specification that employs planning. Planning requires a model of the world the agent is situated in. When we talk about modeling the world we have to decide in which way we want to represent the (real) world for our agent. In the case of a mobile robot we need to maintain information like its position on a map and the positions of objects within the area of operation. Commonly, this model is created from the set of sensor inputs upon which we have to rebuild the robot's model of the world. The sensory information are represented in form of numerical values, e.g. positions within a coordinate system.
We are talking about agents which use planning to fulfill the high-level task of playing soccer. Coaches of human soccer teams talk about possible actions based on overall game tactics in terms of qualitative notions and abstract descriptions. We want to adapt these notions and descriptions to our agents. Within the description of possible actions an agent has to plan with we could formulate something like 'going to the *left side* of the playing field'. But this is an information the robot cannot cope with, yet. Our goal is to enable the use of such descriptions for agents in our framework. In order to formulate behavior strategies in the way coaches of human soccer teams do we need to build up a system which is capable of dealing with qualitative notions such as 'left', 'front', and 'far'.

1.3 Goals and Contributions

Currently, the world model of an soccer agent only consists of numerical data such as its position in a coordinate system, the quantitative distance to obstacles in the surrounding of an agent, or the position of the ball. As motivated in the previous section this thesis aims at building up a qualitative world model for autonomous soccer agents.
Therefore, we are going to investigate human soccer theory. We want to obtain an organizational structure of the technical knowledge present in current soccer literature in order to

determine which qualitative notions are needed to enable the transfer of human expertise in soccer to the specification of an autonomous agent.
To be more precise, we want to raise the level of abstraction within the world model of our soccer agents. That is to say, we want to provide qualitative representations for the information contained in an agent's world model. We are going to do this in a bottom-up manner. We start with the computation of qualitative representatives and attributes from numerical values which are commonly present in the context of autonomous soccer agents, in general, and mobile robots, in particular. By combining several basic predicates we can define more complex ones.
Our approach is meant to improve the agent specification process in several ways. First of all, we want to enable the designer of an agent to express his knowledge of soccer and soccer tactics within an agent specification in almost the same way this knowledge is available to the designer himself. That means, instead of being forced to 'translate' some kind of abstract moves into numerical values he can easily transfer his expertise in the domain to the specification of an agent's behavior.
Furthermore, we seek to exploit the facilities of our high-level component for our task of playing soccer to a greater extend. That is to say, instead of considering a high amount of numerical values we perform planning in our framework upon fewer but more abstract values. This allows for the specification of overall strategies and enables planning with tactical patterns.
Since a mobile robot perceives its environment through sensors and sensors are naturally affected with noise the data available for the robot is not always stable. We want to encapsulate basic forms of uncertainty with our qualitatively abstracted representations and thus provide a more stable model of the environment.

1.4 Outline

The outline of this thesis is as follows: Chapter 2 introduces the framework of this thesis. This includes autonomous agents and approaches on behavior specification as well as the ROBOCUP domain. In Chapter 3 we will give an overview of some related work. Chapter 4 describes the agent specification framework used in this thesis, that is the logical programming language READYLOG in which the high-level control is encoded. Afterwards, we will investigate the soccer domain in Chapter 5 and obtain a description of qualitative information as we deal with in this thesis. We present our approaches to qualitative data representation and reasoning with these predicates in Chapter 6. An exemplary application of our work is presented in Chapter 7. We describe implementational details in Chapter 8. Finally, we draw a conclusion of our work and we give an outlook on possible future work in Chapter 9.

Chapter 2

Framework

In this chapter we start with an introduction to autonomous agents and present three approaches on how to specify the behavior of an agent as well as ways to combine these approaches. Then, we present the ROBOCUP domain, that is the testbed used for the evaluation of different attempts to agent specification.

2.1 Autonomous Agents

Autonomous agents and multi-agent systems (MAS) are major research topics in artificial intelligence, in form of software agents as well as in form of mobile robots. These agents are used to perform a growing variety of tasks such as negotiating in electronic commerce, carrying out mail, and even playing soccer. These tasks commonly require several so-called low-level tasks like localization and movement with collision-avoidance. The low-level tasks can already be carried out quite satisfyingly. But in order to have a robot executing a high-level task like playing soccer the main question is how to let the agent do this in a reliable and intelligent manner. In general, there exist three major paradigms to specify an agent's behavior: *programming*, *learning*, and *planning*.

2.1.1 Programming

The *programming* approach explicitly specifies the agent's behavior for every situation the agent may find itself in. Hence, the agent's behavior is completely determined by the programmer. Normally, this is done by using commonly known control constructs like loops, conditionals, and procedures.
Positive aspects of this approach are that the programmer can control each detail of the agent's program. The program's execution is usually fast because the agent does not have to compute much. The agent has to classify the current state and then it executes the associated action which is computationally inexpensive, in general.
The main limitation of this approach is as follows: if the agent faces a situation which does not match any of those situations specified in its program it simply does not know what to do. The complexity of the program increases along with the complexity of the task. Thus, the design of an agent for a complex task gets more and more complicated as the programmer has to specify the behavior for each and every possible situation. Since playing soccer definitively is a very complex task, specifying an agent only according to the programming approach is almost impossible.

2.1.2 Learning

The *learning* approach tries to overcome the programmer's difficulties of considering every situation the agent may have to face. There are several algorithms to approximate a function which maps from situations to actions. The programmer has to specify a set of examples from which the agent learns and generalizes. Afterwards, the designer has to test if the learnt function suits the agent's needs. Special learning paradigms have been introduced which leave the learning task completely up to the agent. One of these approaches is *reinforcement learning*. Within such approaches an agent explores its environment on its own and receives positive or negative rewards due to its actions in the domain. It then has to associate this reward to its actions in order to be able to find an optimal course of actions.
Different learning methods have been applied successfully to several problems. Tasks like speech or pattern recognition, picture classification, or the dribbling in ROBOCUP are well studied examples of applying learning to autonomous agent problems. Currently, several methods are used in realistic domains like recognition of postal codes on envelopes. In general, the error rate is negligible and it is better than in the programming approach. Examples for learning methods are artificial neural networks or the decision tree learning algorithm. For a general view on learning methods and where they have been applied, please confer [RN03, DHS00, HTF01].
The advantage of learning is the resulting mapping from situations to actions. At runtime, the current situation is presented to the agent and the desired action is generated from the learnt generalization. This query is generally computational inexpensive as it is in the programming approach. Another advantage is that the programmer does not have to specify the complete program. He only has to program the interface to the learning method which is applied to the problem. With supervised learning approaches he further has to collect the set of examples which are shown to the agent. This set has to be chosen carefully because it is the foundation for the agent's behavior.
The main disadvantage of this approach is that the programmer has to give a complete set of situations to the agent, so that all desired actions in all situations are shown at least once to the learning algorithm. The learning algorithm has to generalize from the set of examples to build a mapping from situations to actions. To be able to cope with situations which are slighty different from the examples shown this generalization is of great importance. It is a major problem to obtain sufficiently large sets for many learning problems. Many different algorithms for learning have been proposed for specific problems but there currently is no general solution to the learning problem. If the agent learns something wrong or behaves strangely, it is hard to locate the fault in the agent's program which causes this behavior. Another drawback is that the whole learning procedure has to be repeated if the task of the agent changes.

2.1.3 Planning

In the *planning* approach to behavior specification we do not explicitly specify how the agent has to behave. Instead, the agent is left alone with the decision on an appropriate course of action.
For that purpose we need to create and maintain a model of the world our agent should act in. Furthermore, we need to specify an initial state for the agent to start in as well as a set of possible actions along with their effects on the world. Then we can define a goal state for the agent to reach, leaving it up to the agent to find (i.e. to search for) a feasible way to achieve this

goal by using sequences of actions from the specified set of actions.
But, the comfort to let an agent plan its actions on its own comes along with a huge computational complexity. Instead of simply following the strict instructions given by the programmer, the agent has to reason about different possible courses of action. It can do this, for example, by *projecting* the effects of all possible courses of action up to a certain *horizon*. When such projections have been generated, the agent can determine the expected rewards for each of them and select the most promising for execution. Projection, however, takes time. The complexity depends directly on the number of possible actions determining the branching factor of the thereby defined *search tree* that is traversed in planning.
Unfortunately, there are additional problems that can arise in realistic domains. Mobile robotics, for example, is an application where multiple forms of uncertainty exist. In domains with uncertainty an action can have multiple outcomes. The branching factor of the search tree is then further increased by the number of possible outcomes an action can have. This fact immediately restrains the application of planning in real-time systems when time to decide on the next action is limited.

2.1.4 Combined Approaches

There are approaches which combine the above paradigms. One of these has been presented in [Bee99]. They propose the STRUCTURED REACTIVE CONTROLLERS which combine physical actions, perception, planning, and communication into one framework. It is a hybrid robot control architecture which integrates deliberation, action, discrete actions, and continuous control processes. Robot controllers are realized as concurrent percept driven plans. Features like conditionals, loops, program variables, and subroutines are available as well as high-level constructs like interrupts and monitors both realizing and synchronizing parallel behavior.
An approach that is of particular interest for this thesis was published in 2000 by Boutilier et al. [BRST00]. They propose DTGOLOG which combines programming and planning. They integrate Markov Decision Processes (MDPs) [Put94] into the logic programming language GOLOG [LRL$^+$97]. GOLOG is based on the situation calculus, a second order logic which can be used to model worlds where changes in the world are only due to actions. It offers common programming constructs like sequences, conditionals, and procedures. GOLOG's main benefit is the underlying natural projection mechanism for actions or sequences of actions.
DTGOLOG integrates planning into programming by leaving certain choices to the program. At any time in the program the programmer may specify a set of actions or procedures and let the agent pick the *best* one during program execution instead of determining what to do next explicitly. This is helpful in cases where the programmer only has a general idea about the structure of the problem but not of the particular solution. Therefore, he defines models of possible actions and associates appropriate rewards to certain goal situations in order to guide the agent's search.
From the programming point of view, DTGOLOG introduces non-determinism with an underlying optimization theory. From the planning point of view, DTGOLOG enables the programmer to specify and guide the search of the agent for a plan by restraining the set of all possible plans and therefore it decreases the computational resources needed.
This thesis is carried out in the context of a hybrid approach using the logical programming language READYLOG [Fri03] which integrates programming and decision-theoretic planning. We will describe the situation calculus, GOLOG, and READYLOG in greater detail in Chapter 4.

2.2 ROBOCUP

There are several testbeds in which the results of recent research in different approaches can be evaluated. One of these testbeds is the ROBOCUP domain [TRF05].

> "RoboCup (...) is an international research and education initiative. It is an attempt to foster AI and intelligent robotics research by providing a standard problem where wide range of technologies can be integrated and examined, as well as being used for integrated project-oriented education.
> For this purpose, RoboCup chose to use soccer game as a primary domain, and organizes RoboCup: (The Robot World Cup Soccer Games and Conferences). (...) RoboCup is a task for a team of multiple fast-moving robots under a dynamic environment." [TRF05]

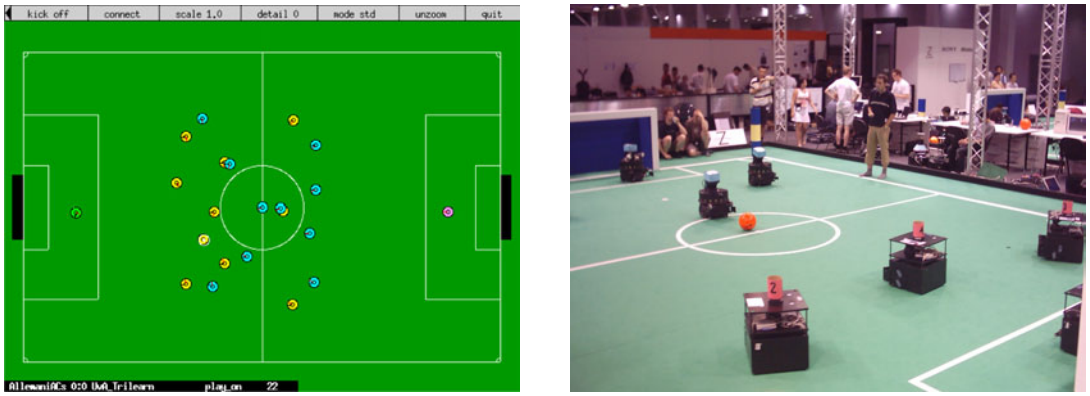
In the ROBOCUP domain autonomous agents are supposed to play soccer. There are different leagues for different kinds of agents. For example, in the Sony Legged Robot League a team consists of four Sony Aibo dog-like robots playing on a field of approximately two times four meters. In our work we consider the SOCCER SIMULATION LEAGUE and the MIDDLE SIZE LEAGUE which we will describe in more detail in the following. In all leagues the agents are completely autonomous. That is to say there is no human interaction during the games, in particular, except for emergency situations. There is an annual world cup as well as several regional tournaments where research groups meet to test and evaluate their approaches by competing with each other.

SOCCER SIMULATION LEAGUE

In the ROBOCUP SOCCER SIMULATION LEAGUE a team consist of eleven software agents playing in a simulated environment. This virtual soccer environment is provided by the so-called SOCCER SERVER [NMHF97]. We depict a screen shot of the SOCCER SERVER in Figure 2.1(a). The duration of a match is two times five minutes. The rules are mainly the same as in human soccer, derived from the official FIFA rules. There are off-sides, throw-ins, and corner kicks, just to name some. Additionally, there are rules that regulate specific aspects of the simulation which are not present in human soccer. For example, it is forbidden to use inter-process communication, although several software agents may be run on the same machine. The agents can broadcast short messages via the SOCCER SERVER. These messages can be received by surrounding players. However, this way of communication is very limited. For instance, it is not possible to let one agent set up an entire team strategy and command all other players accordingly. Consequently, each player has to make his own decision, usually without any knowledge about his teammate's intentions.

MIDDLE SIZE LEAGUE

A team in the ROBOCUP MIDDLE SIZE LEAGUE consists of autonomous robots with mobile and cognitive features. These robots can move freely in their environment (mobile), they use sensors to perceive information from the world (cognitive), and they act without immediate human interference (autonomous). The robots measure 40cm $\times$ 40cm $\times$ 80cm (width, length, height) at most. Two teams of up to five robots play on a field of currently about six times twelve meters. We depict a picture of a game in the MIDDLE SIZE LEAGUE in Figure 2.1(b).



(a) SOCCER SIMULATION LEAGUE (b) MIDDLE SIZE LEAGUE

Figure 2.1: The ROBOCUP domain.

As in the SOCCER SIMULATION LEAGUE the rules are derived from the official FIFA rules but they contain several league specific modifications. There are, for instance, special requirements for the environment. The current rules prescribe coloring one goal yellow and the other one blue. Poles at the corners of the field have similar color codings to make them easy to distinguish for vision systems. In contrast to the SOCCER SIMULATION LEAGUE, the mid-size league allows robots to communicate unrestrictedly using wireless communication. The set of allowed sensors is not strictly specified. Only satellite based localization (GPS) and modifications of the environment, for example by installing active radio transmitters at certain points around the field, are not allowed. Charging, that is pushing opponent robots intentionally, is disallowed. This is especially reasonable as weight and power of the robots differ immensely which implies that heavier robots can push their lighter opponents away. Manual interference on the field is strictly forbidden as well as controlling robots remotely. In case of malfunctioning, robots may be taken out for repairs and be put back into play after at least 30 seconds.

The RoboCup domain provides a testbed with a wide range of standard problems agents have to deal with. Playing soccer incorporates several low-level tasks, namely basic abilities like localization, collision-avoidance, and motion planning. Having one or more agents at hand which are able to perform these basic tasks mentioned above we reach a point where we have to specify an agent's behavior. This means, in the context of RoboCup we have to find a way to let one or more agents play soccer successfully.

Chapter 3

Related Work

In the following we give an overview of the work which is related to this thesis. There is various work on qualitative spatial data representation and qualitative reasoning as well as some work from the field of computational geometry. Finally, we discuss several important publications concerned with soccer.

3.1 Qualitative Spatial Representation and Reasoning

Humans have the ability to perceive spatial objects and reason about their relations. However, this is totally different for computers. One of the most common problems from the computational point of view lies in the difficulty to identify and to manipulate qualitative spatial representations. For example, although the pixels in a digital image implicitly define the locations of spatial objects in a scene, the task at hand might require a more qualitative characterization of the configuration. For instance, it may be of interest whether a particular object is *far away* or not.

The processing of spatial data is a key task in many different applications, such as in geographic information systems (GIS), meteorological and physical analysis, in computer-aided design (CAD) systems, and in protein structure databases. Another important application is robot navigation. Consider, for example, a GIS system. Normally, it has large amounts of numeric information about spatial entities like highways and buildings. But it might require query mechanisms in order to determine qualitative relationships such as those between a proposed route and landmarks along this route efficiently. Qualitative spatial reasoning (QSR) provides representational primitives and inference mechanisms for these tasks. In the previous two decades, a number of qualitative constraint calculi have been developed which are used to represent and reason about spatial configurations.

Cohn and Hazarika give an overview of major qualitative spatial representation and reasoning techniques in their paper [CH01]. They survey the main aspects of qualitative representations and they also consider methods for qualitative reasoning. Their survey covers ontological aspects and topological approaches as well as methods on distance, orientation, and shape. Besides, they mention possible applications for qualitative spatial reasoning which include geographical information systems, robot navigation, computer vision, engineering design, and many others. They evaluate the usefulness of different reasoning approaches with respect to their potential application areas. One of the most well known works on qualitative spatial reasoning is the region connection calculus (RCC).

3.1.1 Region Connection Calculus

In 1992 Randell, Cui, and Cohn presented a calculus for reasoning about regions and connections called RCC [RCC92]. The fundamental approach bases on extended spatial entities, that is regions, and the relations, namely connections, between them. The RCC allows for representing shape and structure of spatial objects. In contrast to systems from classic geometry the approach pursued in the RCC takes regions as primitives rather than points. The relation $C(x, y)$ states whether region x and region y are connected or not. The RCC is founded on two basic axioms on C, namely reflexivity $\forall x[C(x, x)]$ and symmetry $\forall x, y[C(x, y) \rightarrow C(y, x)]$, plus some axioms and definitions on the main spatial relations. With these axioms one can define predicates and functions that allow for the description of topological knowledge.

All theorems and functions in the RCC are defined in first order logic. Thus, reasoning in the RCC can be performed by theorem proving. But, since first order logic is generally undecidable there is no effective way of reasoning with the complete version of the RCC in terms of computational complexity. Nevertheless, several efforts have been made to find subsets of the RCC which are more tractable.

For example, the RCC-8, a subset of the RCC, provides a set of eight basic relations between two regions x and y: disconnected $DC(x, y)$, part of $P(x, y)$, proper part of $PP(x, y)$, identical with $EQ(x, y)$, overlaps $O(x, y)$, partially overlaps $PO(x, y)$, externally connected $EC(x, y)$, tangential proper part $TPP(x, y)$, and non tangential proper part $NTPP(x, y)$. These eight relations are jointly exhaustive and pairwise disjoint. The eight relations of the RCC-8 are depicted in Figure 3.1.

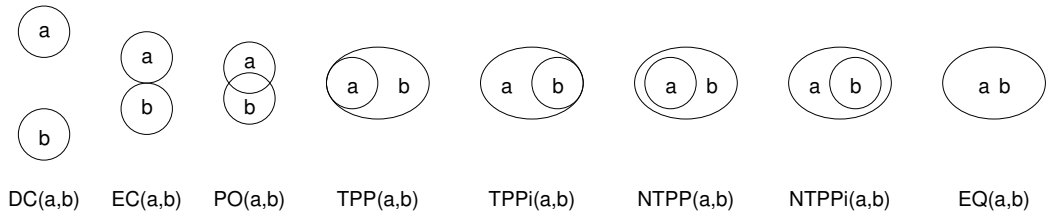


Figure 3.1: The eight axioms of the RCC-8.

The RCC-8 has been well studied by Nebel and Renz [RN99]. They proved that reasoning in the RCC-8 is NP-complete, in general, and they identify a maximal tractable subset called $\mathcal{H}_8$ that contains all basic relations. They observed that this language subset is sufficient to model important topological structures but they also noticed that it can be too weak for some other purposes. For a detailed account on this we refer to [RN99].

Although the RCC provides a powerful method to describe and reason about spatial structures, we are not going to use this calculus or any of its derivatives. Firstly, we do not intend to use regions for representing spatial objects in the ROBOCUP domain. Besides, even if we would represent objects on the playing field as spatially expanded regions it will normally not be the case that any two of these objects will overlap somehow. Second, we think, that we do not rely on the generality provided by the RCC. From our point of view, the description of spatial settings needed for our purposes can be achieved more easily. At least, for our purposes we cannot afford the computational costs of the RCC considering the benefit it might have for our work.

3.1.2 Positional Information

A well known approach to qualitative representation of positional information was proposed by Freksa and Zimmermann in [FZ92]. It bases on a directional orientation information. The approach is motivated by considerations on how spatial information is available to humans and to animals: directly through their perception. Thus, cognitive considerations about the knowledge acquisition process build the basis here.
Qualitative orientation information in two-dimensional space is given by the relation between a vector and a point. The vector consists of a start point A and an end point B. It represents the orientation of a possible movement. Now, imagine a line through A and B. Moreover, consider two lines, one orthogonally going through A and B each. These three lines form an *orientation grid* which has the form of a *double-cross*. Different positions of an additional third point C can be described as follows. In relation to the line through A and B a point C can either be on the *left* of this line, *straight* on this line, or on the *right* of this line. The lines through A and B allow for further distinctions of the position of point C. It can either be in *front* of the line through B, just on the orthogonal line through B, *neutral* in between the two orthogonal lines, on the line through A, or *back*, that is behind the line through A. Altogether this leads to 15 different orientation relations. Figure 3.2 depicts the grid presented above and the iconic representations of the different possible positions of a point C in this grid.

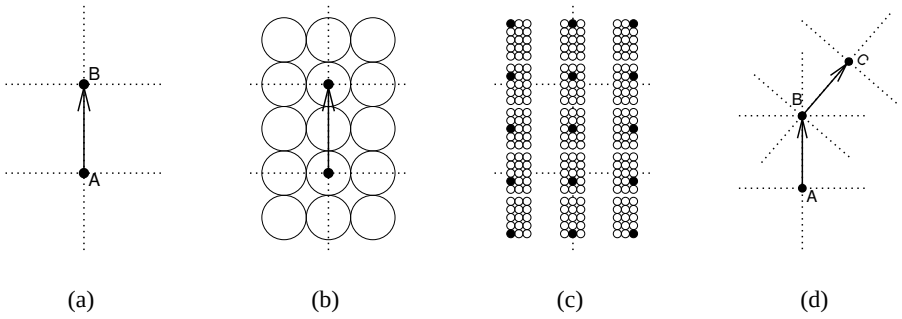


Figure 3.2: The double-cross calculus by Freksa and Zimmermann. Figure (a) shows the orientation vector from A to B. Figure (b) shows the possible position of an additional point C. Figure (c) shows the iconic notation for each of the possible positions of C. A possible composition of two orientation relations is depicted in figure (d).

Reasoning with these relations is possible through four operations, namely *inversion*, *homing*, *shortcut*, and *composition*. Freksa [Fre92] uses two different types of composition, one of them named *coarse composition* and the other named *fine composition*. While the coarse version produces very imprecise results the fine version is enhanced with rudimentary distance information. These supplementary distance information allow for a more exact composition.
The approach presented in [Fre92] and [FZ92] is quite intuitive, not only because it is based on human cognition. But, any relation is based on a vector between two points which cannot be taken for granted in the context of our work. Spatial settings in soccer not always involve a movement but they also describe static situation. Furthermore, we need to represent inherent spatial characteristics of a player in terms of qualitative predicates like the view direction. These inherent traits do also not provide a motion vector to which we could apply Freksa's model. Thus, it is not always applicable for us. Nonetheless, we can use the orientation grid for some sort of tile coding. We will get back to that later in Chapter 6.

Another approach to qualitative orientation relation was introduced by Hernandez [Her91]. Instead of using geometric models which are very precise Hernandez establishes a cognitive model of space. He argues that cognitive spatial concepts are qualitative in nature and preciseness is normally not needed in cognitive models. The qualitative notions in cognitive space are modeled by a relative representation of spatial knowledge. This relative representation does not commit to all aspects of a situation and hence is under-determined. Although the representation might correspond to many 'real' situations it avoids falsifying effects of an exact geometric approach which are likely due to the common limited acuity of perception. Hernandez states that the direct modeling of qualitative statements allows for a more efficient way to handle partial and uncertain spatial information. He also claims that it can lead to more intuitive user interface, for example, in Geographic Information Systems. The orientation relation is meant to represent where objects are placed relatively to each other. The framework proposes to model qualitative orientation with angular intervals. The number of intervals, that is the number of distinctions is determined by a *level of granularity*. An orientation relation states where a *primary object* is located in relation to a *reference object*. Further, there is a *reference frame* which determines the direction in which the primary object is located in relation to the reference object. Hernandez presents methods for changing the reference frame as well as methods for composing relations.

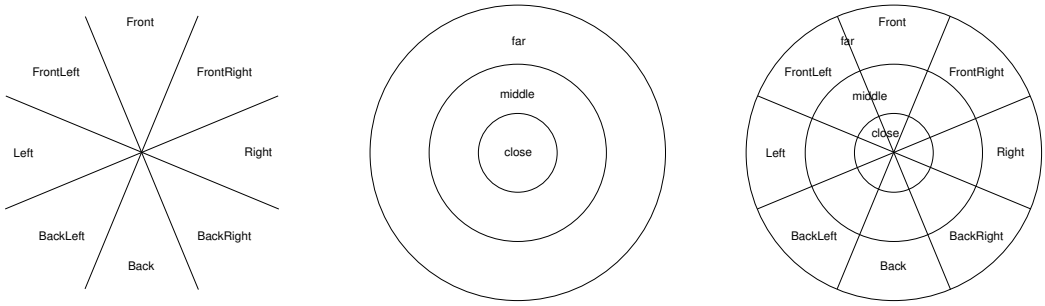
A basic method for qualitative distances was discussed in 1995 by Hernandez, Clementini, and Felici in [HCF95]. Three elements are needed to establish a distance relation, namely a *primary object*, a *reference object*, and a *frame of reference*. The distance $\mathrm{dist}(A, B)$ between the reference object A and the primary object B is expressed as one out of a set of distance relations. These relations are formed by partitioning the plane into regions. These partitions represent the distinctions being made. The context in which the distinctions are made is represented by the frame of reference. It accounts for contextual information such as type and scale of the distance relations as well as for the *distance system* which contains the distance relations and a set of structure relations describing how the distance relations relate to each other. Among their representation of distances through geometric intervals they describe basic inference mechanisms such as composition and switching between different frames of reference.

Two years later, in 1997, Clementini, Felici, and Hernandez presented a unified framework for qualitative representation of positional information in two-dimensional space in [CFH97]. In order to represent positional information they combine the distance relation and the orientation relation mentioned above. Again, they also introduce basic inference and reasoning mechanisms. To give a rough impression we depicted the orientation relation, the distance relation, and their combination in Figure 3.3.

We are going to discuss the three approaches mentioned above in much greater detail in Chapter 6.

3.1.3 Robotic Applications

In [MRL$^+$00] Müller et al. present an application of qualitative spatial representations to robot navigation. They consider the following scenario: In a hospital, a patient should visit a certain room for a medical examination. Since the patient is handicapped a wheelchair is used to get to the examination room. Normally, the patient would be guided by a nurse. But, due to economy of time, the hospital is equipped with intelligent power wheelchairs. The nurse is able to instruct the wheelchair where to go so that the patient can automatically be transported



(a) (b) (c)

Figure 3.3: Combination of the orientation relation (a) and the distance relation (b) to a unified model (c).

to the examination room.

Beside basic modules that cope for controlling the hardware the system architecture used in [MRL+00] employs three higher-level modules responsible for tasks like environment perception, self-localization, motion behavior choice, and behavior execution. The so-called *Situation Detector* extracts features in the robot's environment. These features are mapped to certain landmarks. A module named *Navigator* matches these landmarks to the route description and generates a behavior instruction. Lastly, this instruction is executed by the third high-level module *Behaviors*.

To extract the qualitative notions Müller et al. use methods presented in [MSE+00] which mainly base on the approach by Clementini et al. [CFH97]. We mentioned this approach in the previous section. They generate qualitative motion vectors by using qualitative distance and orientation relations. Then these qualitative vectors are generalized to simplify the motion track. By this method they stress on the coarse form and only regard the major directional changes and the overall shape of the course of motion. For a detailed account on the algorithms applied we refer to [MSE+00].

The navigator matches landmark information computed by the situation detector with the initial route description which specifies the target track. Depending on the current situation, the navigator determines a behavior allowing the wheelchair to follow the route. In [MRL+00] the given route descriptions is supposed to consist of a *starting point*, *reorientations*, *paths/progressions* which can be defined by additional landmarks or approximate distances and a *goal*. The starting point is the current position of the robot and the goal is the end of the route. The route is then specified by a sequence of tuples of the following kind:

$$< \; [\, \{ \, \text{controlmarks} \, \} \, \text{router} \,] \, \text{reorientation} \; >$$

where *reorientation* is a directional instruction such as *turn right*, *router* is a landmark where the directional change should be performed, and *controlmarks* are used to describe locations where no turn should take place.

Imagine the wheelchair is given the command

> "Follow this corridor, after you passed the elevator take the corridor branching off on the right-hand side. Then take the next corridor branching off on the left hand side and stop at its end."

In the above formalization this would look like

$$\begin{array}{lll} < Elevator & CorridorRight & TurnRight > \\ < & CorridorLeft & TurnLeft > \\ < & DeadEnd & Stop > \end{array}$$

Since the robot is not able to perform operations like $\mathrm{TurnRight}$ directly it employs basic behaviors like $\mathrm{FollowRightWall}$ to which the above route description is converted. We will not detail this any further here and instead refer to [MRL+00].
Müller et al. conducted several test cases in an office environment. They state that although the extraction of features is not always as reliable as desired the system sketched above is a promising method. Nonetheless, there remains room for improvements in order to introduce the system into real hospitals.

3.2 Algorithmic Geometry

In the context of computational geometry we want to mention [AK00] where Aurenhammer and Klein talk about Voronoi diagrams and their various applications. They give a detailed overview of the construction algorithms and possible applications of Voronoi diagrams and their dual, the Delaunay triangulation. One of their potential applications for us is to model reachability in distinct forms. That topic is of great importance in this thesis as reachability is an integral element of nearly all strategic specifications in soccer. For instance, we need to determine if a region is reachable with a dribble action or if a pass can reach a teammate. Aurenhammer and Klein present different possibilities of constructing Voronoi diagrams and Delaunay triangulations and they also point out basic and advanced properties of both structures.

A possible application of Voronoi diagrams which is of particular interest for our work is presented in [DAA+05]. In this paper, Dashti et al. propose an approach to dynamic positioning for autonomous soccer agents called DPVC. Their approach which bases on Voronoi cells for each player is meant to achieve a dynamic positioning of players on the field such that there is no restriction for a player's position as there normally is when using home positions. Therefore, they first compute the Voronoi cell of each player. Then, the player is instructed to move towards the center of its Voronoi cell. This positioning instructions yield to a distribution of the players on the field in such a way that the areas covered by each player are almost equally sized.
Moreover, Dashti et al. propose to combine the movement instruction vector with an attraction vector to direct a player into a certain direction of interest. This direction is most often determined by an object or a position of particular importance like the ball or a goal. They state that their approach is more flexible than other methods used for positioning since it does not rely on a home position for each player. That is because fixed home positions restrain the area of influence of player whereas the dynamically computed Voronoi cells do not. Their experiments show that there is a slight improvement concerning the number of players that can receive a pass.

The Computational Geometry Algorithms Library (CGAL) [CGAL05] is a C++ library of geometric algorithms. It is developed by a consortium of eight research teams from different

countries. Its goal is to provide robust, flexible, and efficient implementations of geometric algorithms and data structures. We use CGAL for the computation of triangulations, especially Delaunay triangulations and their dual, Voronoi diagrams. [BDP+02] describes how triangulations are realized in CGAL. Beside basic and Delaunay triangulations CGAL also offers methods and algorithms for regular, constraint, and constraint Delaunay triangulations which may be of use at a later point of time.

3.3 Soccer

In [DFL+05] Dylla et al. discuss a top-down approach to model soccer knowledge as it is presented in current theory books on human soccer. They try to formalize soccer theory in order to use the formal description of soccer moves for agents in different leagues and for different platforms in ROBOCUP. As described in [DFL+05], there are several textbooks dealing with soccer theory. One of those books is Lucchesi's *Coaching the 3-4-1-2 and 4-2-3-1* [Luc01]. It focuses more on tactical patterns than on training lessons. Thus, we can use it to extract soccer moves for the use with ROBOCUP soccer agents.
Another book on soccer tactics is Talaga's *Fußballtaktik* [Tal79]. After introducing the origins of soccer tactics he illustrates different tactical systems. Besides, he characterizes various styles of play predominant in different regions. Talaga covers overall tactical aspects of a soccer team as well as individual strategic patterns along with notes on how to give training lessons. He concludes with observations on tactical behaviors used at several soccer world cups.
Both these books incorporate quite a number of diagrams to illustrate the constituent movements of a tactical pattern.

The Visual Translator system (VITRA) [HW94] is a natural language generation system which is directly grounded in perceptual input. VITRA generates natural language descriptions of dynamic scenes from multiple domains including automobile traffic and soccer games. Semantic representations are extracted from video image sequences. Detailed domain knowledge is used to categorize spatial relations between objects and dynamic events. Higher level propositions are formed from these representations which are mapped to natural language using a rule-based text planner. Within the VITRA project there is a system called SOCCER [AHR88] which generates natural language descriptions of a soccer game. In contrast to our work the systems developed in the VITRA project (only) use video sequences for their input. We are not coping with images sequences but we use the sensory information of an autonomous soccer agent instead. Although, basic notions could be of use such as the structure of tactical moves as well as their natural language description.

More closely related to the ROBOCUP domain there are several systems for an automatic commentary of a soccer game. Mainly three groups developed systems for automatic real time commentary. These systems, namely BYRNE, ROCCO, and MIKE together won the scientific award at ROBOCUP 98 [AK99]. An overview on the three systems can be found in [ABTI+00].
The system BYRNE [BL99] focuses more on the generation of facial expressions and emotions and will therefore not be discussed in detail here.
ROCCO [VAHR99] evolved from the SOCCER system already mentioned above. Instead of the video input used in the SOCCER system ROCCO works on information provided by the ROBOCUP soccer simulation server [NMHF97]. Based on elementary events (like *kick* or *catch*) and geometric data provided by the SOCCER SERVER, ROCCO performs a high-level

scene analysis by an incremental event-recognition. Declarative concepts of event represent a priori knowledge about typical occurrences in a scene. These concepts are organized in an abstract hierarchy, grounded on specialization and temporal decomposition. There exist simple recognition automatons for each concept. These automatons are used to instantiate events based on the underlying scene. Events and state predicates (like *HasBall*) are then used to define higher-level event concepts. For example, a *ball transfer* into the penalty area is defined through several sub-concepts as follows:

$$
\begin{array}{llll}
(Type : HasBall & Time : start & Agent : agent) & \\
(Type : Kick & Time : start & Agent : agent) & \\
(Type : Region & Time : start & Object : Ball & Region : NotPenaltyArea) \\
(Type : Region & Time : end & Object : Ball & Region : PenaltyArea) \\
(Type : HasBall & Time : end & Agent : recipient) &
\end{array}
$$

And as a further condition, there must not be an event of the form

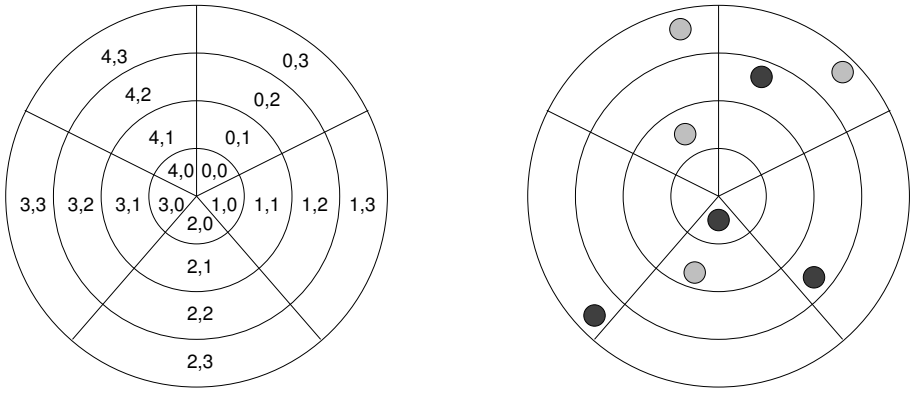
$$
\begin{array}{lccccc}
 & (Type : HasBall & & Time : time & & Agent : agent) \\
\text{with} & start & < & time & < & end
\end{array}
$$

Other concepts are defined in a similar way. We can make use of the definition of high-level concepts since the structure helps us to formulate high-level action sequences. Upon the concepts mentioned above ROCCO generates spoken commentary. For that purpose it uses a discourse planner as well as a content selection module. We will not describe how this is done here as it is of lesser concern for our work.
MIKE [TINF$^+$98] also uses the information provided by the SOCCER SERVER as its input. MIKE consists of six analysis modules that are running concurrently. They can be used to follow and interpret the actions of multiple agents. Upon their analysis the six modules post propositions of information gathered to a pool. MIKE then selects certain events from this pool and generates natural language accordingly while events that were not chosen disappear after a certain amount of time. One of the six analysis modules is highly related to our work as it makes use of Voronoi diagrams. Their application enables MIKE to determine the defensive areas covered by a player as well as to assess the overall positioning. Voronoi diagrams are calculated every $100ms$ for each team. Not only since nowadays the power of computers has increased we expect that we can use Voronoi diagrams in our approach frequently, too. Players are considered to be free if they are positioned as close as possible to a Voronoi vertex of the diagram of the opposing team. Furthermore, triangular shapes in a Voronoi diagram indicate a tight formation since the average shape of a Voronoi area is hexagonal. We will try to use these two observations above in our framework as they seem to provide qualitative insight from a tactical point of view.

In [Fra98] Frank presents a global statistics-based analysis of a corpus of newspaper match reports taken from the Internet pages of the Times newspaper. The analysis is meant to give insights to designers of RoboCup teams. Furthermore, it should help to obtain formalizations which can be used for designing algorithms that control autonomous soccer agents. Frank argues that any factor which is important for the success of a soccer playing team should be mentioned at some time in a game report. With his analysis he aims at identifying important features of the game as well as to discover more detailed objectives than 'scoring goals'. The extensive studies of the soccer language in newspaper reports might enable us to obtain a repertoire of qualitative notions commonly used and thus being relevant for our task. This may

also ease the improvement of interaction between team members for us. We will cover Frank's analysis in more extend in Section 5.2.

In his diploma thesis [Rie03] Riedel evaluates the comparison of world situations in RoboCup soccer games. He develops and studies different approaches to situation classification and to similarity measures. The general intention is as follows. Consider an autonomous soccer agent which uses planning for decision making. The world evolves during the generation of a plan for the current situation. By the time the planning is finished the situation may differ in such a way that the plan cannot be executed any longer. To determine if the generated plan can still be executed Riedel compares the two situations at the beginning and at the end of the planning. He develops and evaluates three different models for the comparison: a graph model, a cone model, and a triple model. All models abstract the world model information and then compute whether or not two situations differ in a significant way. Furthermore, all models are ball centered, that is to say they take the ball as their central entity. The graph model tries to represent a situation as a graph. Different types of objects are represented by different types of vertices. Edges denote possible actions, each one weighted with a probability of success. The triple model takes three distinct properties of a situation. Similarities in each of these properties are then summed up to an overall similarity. The properties used are possession of the ball, possible pass ways, and the grade of endangerment for the player being in possession of the ball. Finally, the cone model bases on a regional approach abstracting object positions to regions. This creates robustness to minimal changes in the player's position as these minimal changes do not cause a plan to be inexecutable. The underlying cone model is depicted in Figure 3.4.



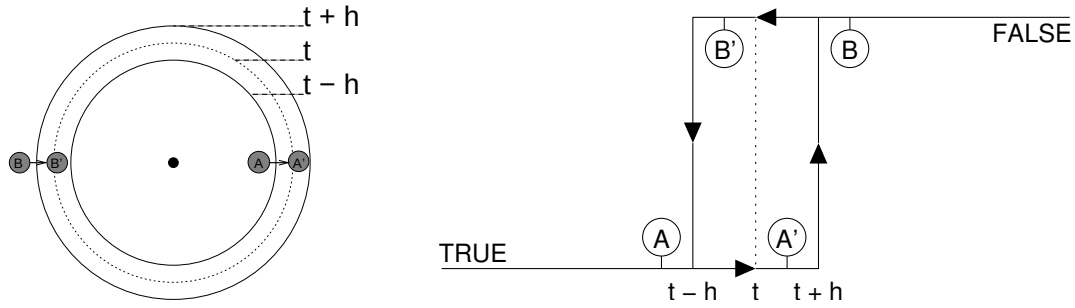
(a) cone model (b) classification

Figure 3.4: The cone model (a) is used for similarity measurement in [Rie03] and an example situation (b) which is classified.

The abstraction made in [Rie03] may help us to determine appropriate levels of granularity for our qualitative predicates as well as to identify relevant properties of a situation in a soccer game.

In [FSW04] Fraser, Steinbauer, and Wotawa describe how to extract qualitative information from numerical world model data. Just like in our work they chose robotic soccer as the real world example domain to apply their qualitative data representation. Fraser et al. use a

hybrid system similar to ours. It combines reactive low-level control with deliberative high-level decision making using classical AI approaches. Their planning approach uses STRIPS-like preconditions for every action. These preconditions are facts about perceived states of the world which are represented by boolean predicates either being true or false such as the visibility of an object or the reachability of the ball.
The mapping from quantitative values to their qualitative representation turns out to be not as easy as it might have looked like. Fraser et al. exemplarily consider the *inReach* predicate which is grounded in the distance to the ball in order to determine whether it is reachable for the robot or not. Due to the unreliable perception of a robot a simple threshold does not suffice as it leads to very unstable results. Hence, Fraser et al. propose the utilization of a so-called *hysteresis function*. Systems following a hysteresis function can be seen as systems that remember decisions taken previously. Once a predicate evaluates to true, it remains true unless a significant change in the world occurs.



(a) evaluation of ball positions

(b) illustration of the hysteresis function

Figure 3.5: A hysteresis function for the *inReach* predicate.

Steinbauer, Weber, and Wotawa discuss first results of the above work in [SWW05]. They present results of several experiments conducted. The symbol grounding with hysteresis proposed in [FSW04] has proven useful to decrease the number of undesired changes in truth values of predicates to a minimum. This improvement in terms of stability in a robot's knowledge increases the performance of the decision making process. Although, some issues still remain unsolved. Firstly, the size of the hysteresis cannot be determined generally, yet. Secondly, the conjunction of a large number of predicates using hysteresis has not been investigated sufficiently. Nonetheless, the concept of hysteresis may prove useful in our approach as well.

In [SOM02] Stolzenburg, Obst, and Murray investigate the use of qualitative motion data in RoboCup, i.e. qualitative velocity in the RoboCup Simulation league. Within their case study they compare four approaches to ball interception, namely a qualitative method, a numerical method, a strategy based on reinforcement learning, and a naive approach. Their focus is the qualitative representation of motion which they say was not covered that much in research on qualitative spatial reasoning. The conducted experiments they present in [SOM02] reveal that the qualitative method is less successful with ball interception than the numerical and the learning approach because the execution system still heavily depends on precise values to be successful. However, their idea of qualitative velocity could be useful for other tasks such as situation classification where less accuracy is needed.

Chapter 4

READYLOG

For the high-level control, both the ALLEMANIACS' SOCCER SIMULATION team and the MIDDLE SIZE team use the logic programming language READYLOG [Fri03, FFL04] which is an extension of GOLOG [LRL+97]. GOLOG is based on the *situation calculus* [McC63], a logical language for reasoning about actions and their effects in a dynamically changing world. In this chapter we introduce the underlying situation calculus first, then we describe the basic GOLOG interpreter. Afterwards, we present the READYLOG interpreter with focus on the extensions being made to suit the idea of combining decision-theoretic planning and programming under real-time constraints. Lastly, we sketch some shortcomings together with improvements already proposed. Along with this we propose the integration of a qualitative world model into the READYLOG framework and describe the benefits we hope to gain from this integration.

4.1 The Situation Calculus

The situation calculus is a logical second order language proposed by John McCarthy in 1963 [McC63]. It was designed to represent dynamically changing worlds. The general idea is that all changes in the world are only due to the execution of *actions* which may be parameterized. Actions are denoted by function symbols. Possible world histories which are sequences of actions are represented in a first order term called *situation*. The sequence of actions in a history is obtained by reading off the actions from right to left. A specific situation of the world is the *initial situation* S_0. This is the situation where no actions have occurred yet. Properties of the world that vary from one situation to another are represented by *fluents* that change over time. We distinguish two types of fluents: *relational fluents* and *functional fluents*. The former are used for relations and the latter are used for functions. Both take a situation term as their last argument. The execution of an action is represented by a special binary function symbol $do(\alpha, s)$ which denotes the successor situation to s. It results from applying action α in situation s. Associated with each action α there is a *precondition axiom* which is a collection of necessary and sufficient conditions under which the action can be executed. It is represented by the predicate symbol $Poss$; $Poss(\alpha, s)$ denotes that it is possible to perform action α in situation s. Additionally, there is an *effect axiom* for each action which describes the impact of this action on the world.

Example

To give an impression of the system mentioned above we outline a small example: [1]
"The Monkey and Banana Problem"

> A monkey in a cage wants to get some bananas that hang from the ceiling. The monkey cannot reach the bananas directly. However, there is a box in the cage; by pushing the box underneath the bananas and by climbing up the box, the monkey could get the bananas.

An example history for the monkey world could be:

$$[\text{climb\_box}(), \text{push\_box}(b, c), \text{goto}(a, b)]$$

which denotes the according situation

$$do(\text{climb\_box}(), do(\text{push\_box}(b, c), do(\text{goto}(a, b), S_0)))$$

where goto, push_box, and climb_box are primitive actions.
Now imagine the push_box action. A precondition for pushing the box in situation s can be modeled as follows:

$$Poss(\text{push\_box}(a, b), s) \equiv \text{at}(Box, a, s) \wedge \text{at}(Monkey, a, s) \wedge \neg\text{heavy}(Box).$$

This means that pushing the box from a to b is possible if and only if the box is at a, the monkey is at a, and the box has not been classified as too heavy to push.
An effect axiom for the above example of pushing the box could be, for instance:

$$Poss(\text{push\_box}(x, y), s) \quad \supset \quad \text{at}(Box, y, do(\text{push\_box}(x, y), s)).$$

Unfortunately, there exists the so-called *frame problem* with the situation calculus concerning the maintenance of a valid world model as some things in the world change while others do not. We describe this problem in the next section.

4.1.1 The Frame Problem

McCarthy observed that for axiomatizing a dynamically changing world more than just precondition and effect axioms are required. So-called *frame axioms* are also necessary to specify the fluents of the domain which remain unaffected due to a performed action. This means we have to specify both the *action effects* and the *action invariants*. For instance, climbing up a box does not change its location:

$$Poss(\text{climb\_box}(), s) \wedge \text{at}(Box, x, s) \quad \supset \quad \text{at}(Box, x, do(\text{climb\_box}(s)).$$

The real problem arises due to the number of necessary frame axioms. Let A be the number of different actions and let F be the number of fluents. The resulting number of frame axioms needed is $2 \times A \times F$. In less simple domains this is a very huge number of axioms the programmer has to specify.

[1] Note that in our formulas symbols starting with an upper case letter denote constants, while variables start with a lower case letter. One exception is the special symbol Now which is always replaced by the current situation and, in particular, is not part of the language. Furthermore, all free variables are implicitly universally quantified.

A solution to the frame problem was proposed by R. Reiter in 1991, [Rei91]. It takes advantage of the fact that effect axioms can be written as

$$\begin{aligned} Poss(a,s) \wedge \gamma_F^+(\vec{x},a,s) &\supset F(\vec{x},do(a,s)) \\ Poss(a,s) \wedge \gamma_F^-(\vec{x},a,s) &\supset \neg F(\vec{x},do(a,s)) \end{aligned}$$

where $\gamma_F^+(\vec{x},a,s)$ denotes the positive effect axioms for action a and fluent $F(\vec{x},s)$, and $\gamma_F^-(\vec{x},a,s)$ the negative effect axioms, respectively. All conditions are characterized in which action a can cause a fluent $F(\vec{x})$ to become true or false in the successor situation. Therefore, the solution of the frame problem bases on a completeness assumption. *All* causal laws are a result of performing action a, no matter if it affects the truth value of a fluent or not. With this completeness assumption it suffices to specify only the effect axioms and to derive all the remaining successor state axioms automatically. We generate all successor state axioms for each fluent in the following form:

$$\begin{aligned} Poss(a,s) \supset\ & [F(\vec{x},do(a,s)) \equiv \gamma_F^+(\vec{x},a,s) \vee \\ & (F(\vec{x},s) \wedge \neg\gamma_F^-(\vec{x},a,s))]. \end{aligned}$$

If action a is possible in situation s, the fluent's value $F(\vec{x})$ in the following situation $do(a,s)$ is either determined by its effect axiom or it is not influenced by an effect axiom and thus remains unchanged.

4.1.2 Basic Action Theory

Levesque et al. [LPR98] formulated a *basic action theory* $\mathcal{D}$ to describe the world and its dynamics:

$$\mathcal{D} = \Sigma \cup \mathcal{D}_{ss} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$$

with

- Σ the set of (domain independent) foundational axioms for situations (e.g. $S_0 \neq do(a,s)$);
- $\mathcal{D}_{ss}$ a set of successor state axioms, one for each fluent;
- $\mathcal{D}_{ap}$ a set of action precondition axioms, one for each action a;
- $\mathcal{D}_{una}$ a set of unique name axioms for actions;
- $\mathcal{D}_{S_0}$ a set of axioms describing the world in the initial situation S_0.

Using the basic action theory we can derive the value of each fluent in the current situation by what is called *regression*. Roughly speaking, regression for a given fluent f and situation $do(a_n, do(\ldots do(a_1, S_0)\ldots)$ works like this: Applying the successor state axiom of f once, will describe the current value of f possibly relative to its value in the previous situation. If the value does not depend on the previous situation, we are done. Otherwise, the value in the previous situation can again be computed by applying the successor state axiom to that situation. This way, the regression will recursively end up in situation S_0 where the value of fluent f can be determined from the axiomatization of S_0.
Naturally, the algorithm described above takes more time the longer the situation term gets. This can be a problem for realistic domains, especially for those where the agent has a nonterminating program to run. Fortunately, there has been approaches to circumvent this problem. One of these approaches will be described in detail in Section 4.3.

4.2 GOLOG

GOLOG is a logic programming language based on the situation calculus and it is implemented in Prolog. It was first introduced by Levesque et al. [LRL+97]. It does not only offer constructs known from ordinary programming languages like loops, conditionals, and recursive procedures, but also less standard constructs like the nondeterministic choice of actions. We briefly present all constructs in the following list where all e_i are legal GOLOG programs:

- nil the empty program;
- a a primitive action;
- $[e_1, \ldots, e_n]$ sequences;
- $?(c)$ tests: if condition c is true, proceed;
- $if\,(c, e_1, e_2)$ conditionals: if condition c is true, proceed with e_1, else proceed with e_2;
- $while(c, e)$ loops: while condition c is true repeat e;
- $e_1|e_2$ nondeterministic choice: do e_1 or e_2;
- $star(e)$ nondeterministic repetitions: repeat e an arbitrary number of times;
- $pi(v, e)$ nondeterministic choice of argument v: choose an arbitrary term t and proceed with e, where all occurrences of v are substituted by t;
- $proc(n, e)$ procedures (also recursive);

One way to define the semantics of these symbols is the introduction of an *evaluation semantics*, originally used by Levesque et al. in [LRL+97]. They introduce a relation $Do(e, s, s')$ which states that executing program e in situation s will result in a new situation s'. The constructs shown above can be seen as abbreviations for formulas in the situation calculus. One example for a translation from GOLOG to the situation calculus is that of primitive actions:

$$Do(a, s, s') \stackrel{\text{def.}}{=} Poss(a[s], s) \wedge s' = do(a[s], s)$$

The formula states that applying action a in situation s will lead to situation s' if and only if action a is possible in situation s. $a[s]$ denotes that all parameters of action a are parameterized through the fluent values of situation s.
All GOLOG constructs can be axiomatized in the situation calculus in that way. For example, the nondeterministic choice of two programs e_1 or e_2 is represented by a simple disjunction in the situation calculus:

$$Do((e_1|e_2), s, s') \stackrel{\text{def.}}{=} Do(e_1, s, s') \vee Do(e_2, s, s')$$

where e_1 and e_2 are both valid GOLOG programs.
Another example is that of conditionals. Conditionals can be formulated in the situation calculus as follows:

$$Do(if\,(c, e_1, e_2), s, s') \stackrel{\text{def.}}{=} (c[s] \wedge Do(e_1, s, s')) \vee (\neg\, c[s] \wedge Do(e_2, s, s'))$$

which states that interpreting the conditional results in s' if either the condition c holds and e_1 is executed, or if condition c does not hold and program e_2 is executed.
Some of the GOLOG constructs have to be axiomatized in second order logic. The number of iterations in a loop, for example, cannot be defined in advance. This means that a loop always defines a set of programs. For a complete example, the implementation, and a detailed description of the second order problems in GOLOG confer [LRL+97].
Most important, GOLOG programs are evaluated by proving them:

$$Axioms \models Do(program, S_0, \sigma)$$

where σ is a ground situation term. The user supplies a precondition axiom for each action and a successor state axiom for each fluent; he completely specifies the initial situation of the world to the system. Afterwards, the user defines the $program$ which represents the behavior the agent should exhibit in the system. This means we have to prove that there exists a situation s which satisfies:

$$Axioms \models (\exists s) Do(program, S_0, s)$$

The result is a ground term of the form $do(a_n, \ldots do(a_2, do(a_1, S_0)) \ldots)$ which can be interpreted as a sequence of actions $[a_1, a_2, \ldots, a_n]$. By executing this sequence of actions the program satisfies the user's program and solves the given assignment.

The original system has some fundamental drawbacks. Up to this point, there is no way for an agent to sense its environment. The agent depends on the complete definition of the initial situation and on the accurate specification of the actions. Properties of the world that are not specified within the initial situation cannot be sensed during execution. Another essential drawback is that the system is not able to handle re-planning tasks. If the agent finds a plan it starts executing it and it sticks to it no matter how the real world evolves. Furthermore, actions which are not up to the agent, so-called *exogenous actions*, are also not handled by this interpreter version of GOLOG. Lastly, continuous processes tend to be a problem, too. The original GOLOG system is not able to handle fluents that change continuously over time. Many of these drawbacks were tackled and some solutions were already proposed. We are going to detail this in the next section.

4.3 READYLOG

Because the expressiveness of GOLOG was not strong enough to model important properties of realistic domains like mobile robotics, several extensions to GOLOG were proposed. They allow for dealing with continuous change [GL00a], concurrency [DGLL00], exogenous and sensing actions [DGL99], and probabilistic projections into the future [Gro00]. There are several interpreter versions like PGOLOG, INDIGOLOG, CCGOLOG, or CONGOLOG which applied one particular aspect to the original GOLOG interpreter. Jansen integrated all features necessary to control robots or simulated agents in a ROBOCUP domain into an interpreter called ICPGOLOG in [Jan02]. That is, progression, on-line execution, probabilism, continuous change, and concurrency were integrated into this interpreter. It was used to control soccer agents which participated in the ROBOCUP soccer simulation league. Another extension called DTGOLOG [BRST00] introduces decision-theoretic planning. Fritz enhanced ICPGOLOG by integrating decision-theoretic planning and several off-line optimizations into READYLOG [Fri03].

In the following we will list the features which have been integrated into the READYLOG interpreter:

On-line: The READYLOG interpreter is an *on-line* interpreter. Programs are executed right during interpretation. The interpreter works *incrementally*, that is, after each step in the program the interpreter commits to the decision by executing the respective action in the real world.
This idea was first proposed and implemented by Levesque and De Giacomo [DGL98] because the interpretation of long programs resulted in unacceptable delays. The introduction of choice points in the program (for example nondeterministic choices) requires the interpretation and projection of all program branches up to the end to see which outcome satisfies e.g. a final test. The problem gets worse as the number of choice points and the program size increase. Moreover, programs cannot be executed when controlling a non-stopping job like playing soccer with a mobile robot. A complete projection of the program is impossible. Therefore, Levesque and DeGiacomo introduced an off-line operator Σ which is able to project an arbitrary GOLOG program without incremental execution. The program $a; \Sigma\,(b|c); d; t?$, for example, executes action a in the real world and it then checks in an off-line fashion if either executing the sequence of actions $b; d;$ or executing the sequence $c; d;$ leads to a satisfaction of the test $t?$. If a branch is found that satisfies $t?$ the according sequence of actions is executed in the real world.
Another important point is the natural support of *sensing*. Certain aspects of the world may be unknown to the program but they are needed to perform reasonable actions in the world. For example, a robot has to sense the state of doors, its battery status, and its location to perform reasonable actions. The fluents associated with these properties change over time and, in general, cannot be predicted at runtime. They can only be sensed during execution by using sensors. Thus, a sensing action "$\mathrm{sense}_{\mathrm{sensor}}$" was introduced.
Additionally, *exogenous actions* are taken into account. These are actions beyond the control of the agent. It can neither execute them nor predict their occurrence. Although, these actions modify the world and, hence, they change values of fluents. Like the ICPGOLOG interpreter, READYLOG supports exogenous actions as long as their *effects* are modeled in advance. Whenever an exogenous action occurs, the interpreter executes the model of the action which describes the effects of this action and the fluents' values are changed according to this model. Exogenous actions can be compared with sensing actions because both provide information about the world to the agent. The difference lies in their occurrence: whereas exogenous actions can be seen as events raised by the world and are beyond the agent's control, sensing actions can be interpreted as querying information from the world.

Continuous Change: Grosskreutz and Lakemeyer were the first to propose a notion for representing *continuously changing* properties of the world and they implemented this in CC-GOLOG [GL00a]. This extension allows for reasoning about continuous fluents that change over time with linear approximation, both on- and off-line. The value of a continuous fluent is expressed as a function of time. So-called *t-functions* compute the changing of a continuous fluent over time. A binary function $\mathrm{val}(f, s)$ evaluates the t-function f at a given time in situation s and returns the fluent's value at that time.
For instance, consider the position of an agent in one-dimensional space. With the properties above, we can model the agents position by specifying its initial position x_0 at time t_0 and its velocity v_x. The above t-function can now compute the agent's position at time t with the

formula $x = x_0 + (t - t_0) \cdot v_x$. For more examples and further details we refer to [Gro02].

Concurrency: Concurrency is understood as interleaving two programs and does not consider actions being truly simultaneous (i.e. parallel primitive action execution). This concept was first implemented in the CONGOLOG interpreter by De Giacomo, Lesperance, and Levesque proposed in [DGLL00]. The semantics of concurrency in READYLOG however was adapted from Grosskreutz [Gro02]. Instead of the prioritized concurrency from CONGOLOG, a concept of time was introduced. For constructs like $conc(e_1, e_2)$, the program which has earlier transition termination time is favored.

Probability Theory: In realistic domains *uncertainty* is an essential component. One form is the uncertainty about how the world is going to evolve. Further, an agent does not know whether the execution of an action succeeds or not, not only since it cannot predict the occurrence of exogenous actions that may influence the outcome. A way of representing probabilistic effects is to list all possible outcomes with their respective probability.
This way of modeling probabilism in GOLOG was integrated in PGOLOG [GL00b] by Grosskreutz. He introduces a new construct $\mathrm{prob}(p, e_1, e_2)$ which means that with probability p program e_1 is chosen and with probability $1 - p$ program e_2 is chosen. This construct allows to perform a *probabilistic projection* in PGOLOG and it is intended for off-line projection only. The user is able to query the probability that a certain condition holds or that a certain state is reached after executing a program containing prob statements. He can then also use this feature for decision making.

Progression: Following the ideas of Lin and Reiter [LR97], Jansen implemented a mechanism of *progressing* the knowledge base in ICPGOLOG [Jan02]. Because the space consumption grows rapidly with execution of actions and the time to perform regression increases heavily in dynamic environments, the progression of the knowledge base is indispensable for realistic domains.
The general idea of progression is as follows: once an action is performed in the real world, it usually cannot be taken back. Situations specified in the situation term before performing the last action become irrelevant for reasoning. Thus, it is reasonable to *progress* the knowledge about what was true in the initial situation S_0 to what is true in the situation where the last action was committed. This allows for computing formulas using fluents which are based on a smaller situation term through an indefinite long lasting program run.

Decision Theory: Decision-theoretic planning describes planning as finding a *policy* which is an optimal course of action based on a given *reward function*. Instead of solving a given assignment by a sequence of primitive actions, policies are returned. Policies contain rules about which action should be applied in which set of states to have the best outcome for the reward function.
Formally, this is done with Markov Decision Processes (MDPs) [Put94]. MDPs consist of a set of *actions*, a set of *states*, and a *transition function* which describes the probability of which states results in what other states when applying an action. A *reward function* describes the value of each state and action. It was first integrated in DTGOLOG by Boutilier et al. [BRST00]. For a more detailed account on the integration of decision-theoretic planning into READYLOG confer [Fri03].

Options: Because of the need for hierarchical planning and for abstraction from primitive actions, the idea of *macros* was proposed. Macros are fixed sequences of actions which are intended for frequent use. In stochastic settings *options* denote a concept similar to macros. It generalizes from action sequences to action policies [SPS99]. Options are supposed to save computational effort on frequently reappearing tasks. They were implemented in READYLOG [Fri03, FFL03]. Fritz et al. describe how options are precomputed by an offline preprocessor. They states that they save exponential computational effort at runtime.

4.4 A Soccer Example

The READYLOG framework was successfully used in several ROBOCUP events. The ALLEMANIACS team uses the features described above to implement the high-level control for each robot. To give an idea of how high-level control is implemented in that framework, we depict an example program in Figure 4.1.

The procedure $\mathrm{proc\_attacker\_bestInterceptor}$ is called if the robot having the role *attacker* is the best positioned player to gain control over the ball. At the beginning of the procedure two opportunistic possibilities are checked. If the robot detects an easy scoring situation it should try to score immediately. If there is a throw-in situation at the end of the field, a hard coded 'corner kick' routine is called. If none of these is the case, the agent decides on how to intercept the ball which can either be a direct or an indirect interception. After deciding how the ball should be intercepted, the program starts to deliberate what to do, when the ball is caught. With a horizon of four, that is all leaves that are deeper in the tree are pruned, the program chooses between dribbling, kicking, move-kicking, or turning with the ball. The solve statement starts decision-theoretic planning and the nondet statement is used for a nondeterministic choice between the given choices of action.

Let us consider an exemplary situation for the above program. We depicted the situation in Figure 4.2. It shows the positions of the ball (the orange circle in the lower left), of our robots (black squares labeled with the AllemaniACs logo), and of the opponents (black squares labeled with a blue line) based on the fused information from all of our robots. The robot named Kirk is the attacking player and it is running the program from Figure 4.1. We assume that none of the if-cases in the first part of the program applies so that the agent starts decision-theoretic planning with the solve-statement.

Firstly, the attacker (Kirk) evaluates three possibilities to choose from where itself conducts an action. It can either directly perform a kick towards the goal, it can carry out a $\mathrm{dribble}$, or it can execute a $\mathrm{move\_kick}$, that is to begin with a dribble towards the goal and then shoot at it.

As the second set of actions to choose from the agent can decide to perform a basic form of team-play. It can turn to one of two directions which is selected by the $\mathrm{pickBest}$ statement. Subsequently, the $\mathrm{bestInterceptor\_Offline}$ which in our case is the robot labeled $\mathrm{Stoerte}$ can decide to either $\mathrm{dribble}$ or $\mathrm{move\_kick}$ itself towards the goal after intercepting the ball.

Within decision-theoretic planning each one of the possibilities above is simulated and the resulting situation is evaluated with a *reward function*. All actions a modeled as uncertain. That is to say, they have a success and a failure case. Especially intercepting the ball and dribbling with it are considered to be difficult, so that the probability for a failure of both these actions is assumed to be 0.8. The reward function assigns a value to each of the projected

```
proc( proc_attacker_bestInterceptor,
  if( f_easy_scoring_situation, /* first check opportunities */
      proc_easy_scoring_situation,
      if( f_far_throwin_situation,
          proc_far_throwin_situation_active,
          /* if none applies: */
          [ if( func_distance( agentPos, ballPos ) < 0.8,
                /* do a direct intercept */
                [intercept_ball_nonblocking( own, 2 ),
                 set( directIntercept, false )],
                [?( and( [ballPos = [BX, BY],
                          field_size( FX, FY )] ) ),
                 /* intercept ball */
                 if( and( [BX > FX/2.0 - 1.5, BY > FY/2.0 - 0.5,
                           not( isDribblable )] ),
                     intercept_ball( own, -1.2*sign( ball_y ), 3 ),
                     intercept_ball( own,
                                     angleTo( ballPos,
                                              bestTarget ), 3 )
                   ) /* end if */
                ]), /* end if */
      /* while doing that: deliberate! */
      solve( [continueSkill( currentSkill ),
        nondet(
          [kick( own, 50 ),
           dt_dribble_or_move_kick( own ),
           dt_dribble_to_points( own, 1 ),
           if( isKickable,
               pickBest( var_turnAngle, [-3.1, 3.1],
                 [turn_relative( own, var_turnAngle, 3 ),
                  intercept_ball( bestInterceptor_Offline,
                                  angleTo( ballPos,
                                           bestTarget ),
                                  2 ),
                  dt_dribble_or_move_kick( bestInterceptor_Offline )
                 ] ),
           [] ) /* end of if */
        ]) /* end of nondet */
      ], 4 ) /* end of solve, horizon 4 */
   ]) /* end of if, far_throw_in */
  ) /* end of if, easy scoring situation */
). /* end of procedure */
```

Figure 4.1: The bestInterceptor program performed by an offensive player.

(a) exemplary situation

```
solve( [continueSkill( currentSkill ),
   nondet( [ kick( own, 50 ),
             dt_dribble_or_move_kick( own ),
             dt_dribble_to_points( own, 1 ),
             if( isKickable,
                 pickBest( var_turnAngle, [-3.1, 3.1],
                   [turn_relative( own, var_turnAngle, 3 ),
                    intercept_ball( bestInterceptor_Offline,
                                    angleTo( ballPos, bestTarget ), 2 ),
                    dt_dribble_or_move_kick( bestInterceptor_Offline )
                   ] ), [] ) /* end pickBest/if */
   ])
], 4 ) /* end of nondet/solve with horizon 4 */
```

(b) complete solve statement

Figure 4.2: An exemplary situation and the corresponding READYLOG code.

(a) individual choices

```
nondet( [ kick( own, 50 ),
          dt_dribble_or_move_kick( own ),
          dt_dribble_to_points( own, 1 ),
          [...]
]) /* end of nondet */

proc(dt_dribble_or_move_kick(own),
     nondet([[move_kick(own, bestTarget, 3)],
             [dribble_to(own, bestTarget, 3)]
     ]) /* end of nondet */
).
```

(b) nondet statement

Figure 4.3: The set of choices for individual actions to take for Kirk.

(a) team-play choices

```
nondet( [ [...],
   if( isKickable,
       pickBest( var_turnAngle, [-3.1, 3.1],
         [turn_relative( own, var_turnAngle, 3 ),
          intercept_ball( bestInterceptor_Offline,
                          angleTo( ballPos, bestTarget ), 2 ),
          dt_dribble_or_move_kick( bestInterceptor_Offline )
         ] ),
   [] ) /* end of if */
]) /* end of nondet */
```

(b) pickBest statement

Figure 4.4: The set of possible team-plays between Kirk and Stoerte.

future situations. The value of each of these final situations is multiplied with the probability that the actions leading to this situation succeed.

The projection into the future is done by applying models of the actions. These models describe an action's effects on the world. For example, the model for the move-kick action is described by driving one meter to the front and then accelerating the ball towards the target.

The $\mathrm{bestTarget}$ denotes the best corner of the goal to shoot at. In our example situation this obviously is the middle of the goal's right half since it is not covered by the goalie.

Let us see to what decision the agent gets in our example. The initial situation has a base reward of 5000.

We first consider the choices depicted in Figure 4.3. A direct kick in the above situation (indicated by the purple line) would cause the ball to advance towards the opponent's goal ending near to the opponent's goal keeper. The associated reward is based on the intersection with the goal line and is computed as 9250. A $\mathrm{move\_kick}$ (denoted by the yellow line half dashed and half solid) would cause the ball to be pushed one meter in direction of the $\mathrm{bestTarget}$ and then getting shot towards it. The reward associated to the resulting situation is 14922. It is higher than with the direct kick since the target is farther away from the opponent's goalie. The possibility to $\mathrm{dribble}$ (showed as a dashed red wiggly line) results in almost the same situation but it is less likely to succeed. That is because the ball gets lost more easily due to eventual circular movements in a dribble action.

Furthermore, there are choices to conduct a basic form of team-play as we depicted it in Figure 4.4. Turning with the ball (dotted purple lines) which succeeds with probability 1.0 will lead to a situation with reward 4169. Next, the supporter (Stoerte) is assumed to intercept the ball. This is only successful with probability 0.2. The reward for the new situation is the same as before since the ball is not moved in our model. However, the intercept has negative costs, as the ball is not far away from the supporter and it is in front of the opponent's goal where an intercept always seems a good idea. Finally, the supporter now in possession of the ball is assumed to score a goal by performing the $\mathrm{move\_kick}$ or the $\mathrm{dribble}$ skill. The resulting situation, with the ball in the opponent's goal, gets a reward of 10000. But, since initially there is an opponent very close to the ball the overall reward is not as high as it was for the individual actions.

Altogether, the highest expected reward of 14922 is computed for the resulting policy starting with a direct $\mathrm{move\_kick}$:

```
[ marker(nil=nil, true),
    [ move_kick(3, [5.3249998, -0.4217660], 3, 80),
      [ exogf_Update,
        [ if(near(ballPos, [5.3249998, -0.4217660], 0.5)=true)
          then
          | []
          | [ (..), [] ], [] ] ] ] ]
```

An important remark on the interpreter has to be made. In the ROBOCUP domain it turned out that the action models and the reward function are very crucial for an agent's behavior. A designer has to take great care specifying them. The agent which is almost exclusively guided by the reward function, is very sensitive to eventual mistakes. They can lead to unintended behavior easily.

4.5 Improvements

Although READYLOG has already proven to be applicable in domains with real-time constraints there are some shortcomings which leave room for improvements. We briefly sketch these shortcomings along with improvements already proposed. Then, we mention how our work may help to integrate and apply some of them.

Ferrein, Fritz, and Lakemeyer propose the use of options in [FFL03]. As stated above options are policies computed offline in advance and they are meant to be applied online later in appropriate situations. This presupposes that we are able to determine when and if an option can be applied by comparing the preconditions of an option to the current situation in the 'real' world. Since ROBOCUP is a continuous domain we are currently not able to compare situations properly since minimal changes in one or more of the continuous fluents result in a different state. Although such a minimal change would not fundamentally change the setting it would cause an option already computed for a similar situation not to be applicable anymore.
With qualitative predicates we can abstract a continuous fluent which can take infinitely many values to a set of few, more abstract values. Thus, the precondition of an option could be expressed in terms of these qualitative values and would then be more comparable to the current situation which can be expressed in terms of qualitative values, too.
Qualitative predicates could not only enable us to use options, but it could also allow us to re-use plans already computed. We could, for instance, apply this to re-use a plan which has been generated for a particular situation in a soccer game if the current situation is similar to the former one.

Jacobs proposes the use of caching in [Jac05] to save time in decision-theoretic solving of MDPs in READYLOG. Caching means to save and re-use the calculation of intermediate results to speed up the computation.
The method implemented in [Jac05] saves the complete transition table for each state in a MDP. The transition table is a way to represent the relation $T(s, a, s')$. That is, performing action a in state s leads to state s'. This relation is extended by associating the value $v_{T(s,a,s')}$ to it. Every time the value of such a transition is computed it is stored along with the according transition. States are saved as the values of each fluent of s and s'. Please keep in mind that each state is completely described by all the values assigned to all fluents in the current situation. If the same transition $T(s, a, s')$ is expanded again at a later point in time the value is already known. It is read from the cache and re-used without performing the same calculation again.
The method above is applicable for discrete domains like the grid-world but not for continuous ones. That is because chances of two particular states being equal is most unlikely in domains with continuous fluent values since they can take infinitely many values. If we have a state abstraction like the one achieved with qualitative representations and predicates similar situations would be represented by the same set of values. This may allow for the application of caching also with continuous fluents.

With its combination of programming and decision-theoretic planning READYLOG allows an agent's designer to guide the search performed in planning. The language enables him to restrain the set of possible plans by using conditionals and restrictive preconditions in his specification that need to be true. Due to the continuous character of soccer such restrictions currently have to be formulated in form of numerical values.

If continuous fluents and combinations of these fluents were abstracted to qualitative predicates this would enable the designer to formulate his knowledge more easily. That is because these qualitative notions provide an even more natural description of the properties in the domain. He would, for instance, no longer have to think about distance in terms of metric values but could simply state things like *close*, *far*, or *in reach* instead. Our goal in this thesis is to provide such predicates and representations for properties in an agent's world.
Furthermore, the state abstraction which can be achieved with qualitative predicates could make the decision making process more tractable. That is because the number of possible states the agent has to consider in his search would decrease to a great extend if there were only a few values in terms of qualitative representations instead of infinitely many when using quantitative numerical values.

Since READYLOG with its underlying situation calculus already provides facilities to project into the future, that is reason about actions and their effects, we do not want to use one of the common calculi for qualitative spatial reasoning. Instead, due to a hybrid approach which holds quantitative and qualitative data side by side, we use the situation calculus to do reasoning. We are going to detail this in Section 6.5.

Chapter 5

Soccer Theory and RoboCup

We already mentioned that it is reasonable to use existing work on human soccer theory, in general, and tactical patterns, in particular, when specifying the behavior of an autonomous soccer agent. In the following we investigate two books on human soccer theory in order to derive an ontology on soccer first, second, to identify qualitative predicates used within the description of soccer moves, and third, to adapt essential elements of soccer tactics for teams with less than eleven players. The analysis is mainly based on Lucchesi's *Coaching the 3-4-1-2 and 4-2-3-1* [Luc01] and Talaga's *Fußballtaktik* [Tal79]. We chose these books because they focus more on the presentation of tactical aspects than on training lessons.

5.1 Organization of Soccer Knowledge

Soccer is a rather simple, flowing game. Nonetheless, only the teams that are most organized and prepared can achieve victory. Therefore, apart from several individual skills required for each player, the success of a soccer team strongly depends on strategy and tactics. Usually, a team uses a specific *formation* or follows a certain *system of play*. Each system then consists of specific tactical patterns and strategies. Descriptions and discussions of these strategies can be found in current soccer literature. We are now going to analyze this literature in order to derive an organizational structure and significant elements.

Soccer strategies are mostly described in a more or less abstract way. That is, they provide a set of examples that sum up the main ideas of tactical patterns. Usually, examples are represented by figures or diagrams showing a typical (and perhaps idealized) situation in which a particular pattern can be applied. Furthermore, movement indicators for single players encode the actions to take. The single players normally represent a player of a certain type, that is to say a specific role in the overall team behavior. We depict an ontological structure of roles in a soccer team in Figure 5.1.

Unfortunately, it is not that straightforward to transform an idea encoded in a figure into a specification for an autonomous soccer agent. That is because, first of all, these figures make use of implicit semantic notions which are part of a soccer ontology. Elements of such an ontology are, for example, strategic positions, tactical roles of a player, or semantic regions on the playing field. All these elements are abstract and qualitative in nature. We can adopt such abstract descriptions from current literature more easily if we have *qualitative* concepts in our specification framework, too, for both the soccer environment and corresponding concepts for the actions taking place therein. This enables us to work in a more abstract way.

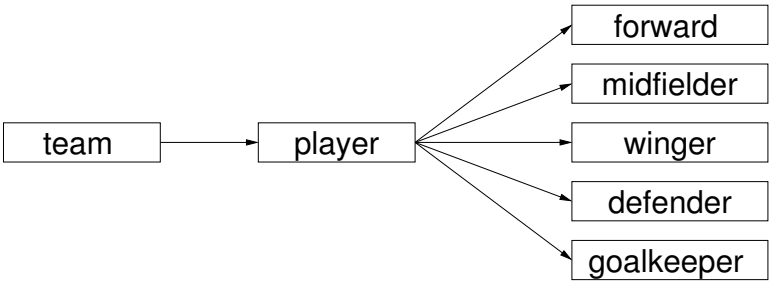


Figure 5.1: An ontology for the organization of a soccer team

In order to derive an overall structure of a soccer game we are now going to investigate the course of a soccer game in more detail.

Course of a Soccer Game

The course of a soccer game is divided into two different phases, namely the defensive phase and the offensive phase. According to [Luc01] the offensive phase itself can be structured by dividing it into four sub-phases. The first phase is *gain(ing) ball possession* followed by a phase of *build(ing) up play* which gets a *final touch* in order to *create a scoring opportunity*. Finally, the offensive phase is completed with the *shooting* phase. This ontological structure is depicted in Figure 5.2.

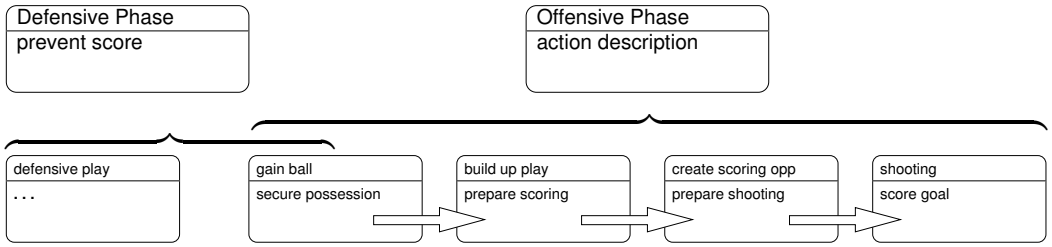


Figure 5.2: A top level ontology for the course of a soccer game

The two phases *build up play* and *create scoring opportunity* are characterized by a higher amount of tactical elements than the other two. Therefore, we are now going to focus on these tactically more significant two phases.

Build up Play

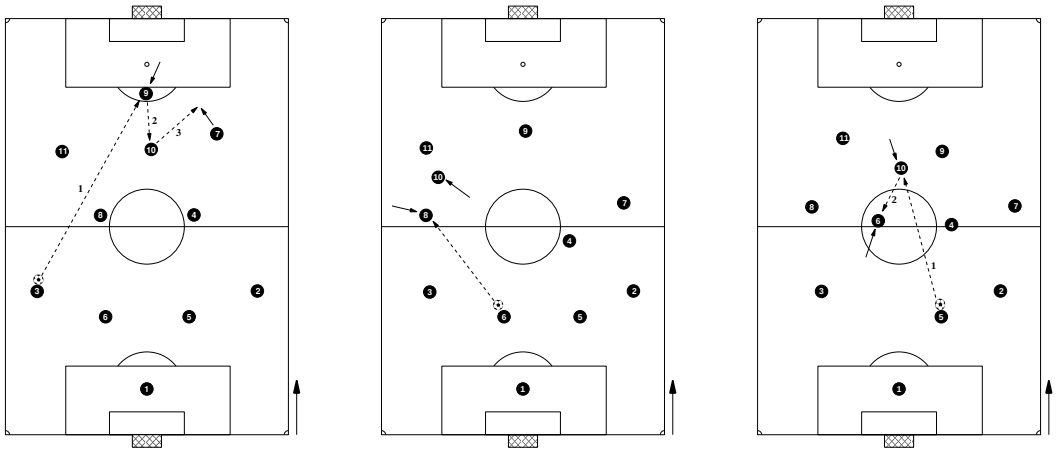
The most important part from a tactical point of view may be the *build up play* phase. The team's objective in this phase is to take the ball towards the opponent's goal in order to establish a setting which allows for creating a scoring opportunity. There are a number of ways to build up a play:

- immediately with a long pass
  The long pass enables the team to take the ball up-field towards the opposing goal very quickly. There is an immediate reversal of play and the risk of losing the ball near one's own penalty area is very low. However, the long pass is difficult to receive, so the opponent may be able to steal the ball more easily. Moreover, as there is not much time, the team cannot move forward in a coordinated way.

- deliberately with a diagonal pass
  The diagonal pass allows for a coordinated way to move forward with the ball and it is easy to receive. The time to get close to the opponent's goal is longer than with the long pass. Thus, chances of losing the ball in a dangerous area are higher.

- deliberately with a deep pass and a subsequent back pass
  This way of building up play requires very good timing as it involves three players who have to move in a coordinated way. If such a move is carried out successfully it allows the team to move forward up-field without great risks if they lose the ball.

We depict one exemplary tactical move for each of the three patterns to build up play mentioned above in Figure 5.3.



(a) A long pass

(b) A diagonal pass

(c) A deep pass and a subsequent back pass

Figure 5.3: Three different ways to build up a play. Dashed lines represent pass ways and solid lines denote a player's movement. The bold arrow next to the field indicates the direction of play.

The decision which pattern to choose certainly depends on the opposing team as well as on the particular situation. That is to say, when playing against an opposing team which has many players in the midfield one would perhaps favor to build up a play with a long pass whereas with a team leaving lots of space uncovered in the midfield one would prefer the deep pass.
Basically, all the possibilities mentioned above are meant to take the ball *up-field* and establish a more offensive setting for the team while remaining in possession of the ball. The ball is either taken forward from the *defense* to the *midfield* section or in the case of the long pass directly to the *offense* section. Both can be done through the *center* of the playing field or by using the *wings* of the pitch. Depending on how the play was built up there are several ways to create a scoring opportunity.

Create Scoring Opportunity

This is the phase where the attacking team tries to place a player in position to shoot at the goal. This can be achieved with one of the following movements:

Cut: The player with the ball is positioned behind the receiving teammate. The receiving player moves forward, i.e. cuts in, in order to achieve an unmarked position. Then the player with the ball makes a (deep) pass to the unmarked player.

Overlap: The player with the ball is positioned ahead of the receiving teammate. The receiving player comes from behind and moves alongside his teammate having the ball. Then the player with the ball makes a deep pass to the unmarked teammate.

Wall pass: The player who acts as a wall in the wall pass move is positioned ahead of the receiving teammate with his back turned to the opposing goal. The receiving player comes from a back position and moves up-field to receive the ball unmarked. A third player passes the ball to the wall passing player who passes the ball to the receiving player.

Combination: A combination is a move which involves two players and enables one of them to shoot. The most common combinations are: *give and go*, *give and follow*, or a *dummy movement*.

Cross pass: A cross pass is usually made from the wing towards the center of the field. The player receiving the cross pass has to be the first to reach the ball in order to shoot. This may require an individual move to unmark himself in advance.

Assist: An assist is a pass that enables the receiver to shoot at the goal.

Dribble: In a dribble the player with the ball beats his (direct) opponent and thus creates a situation of numerical superiority for his team.

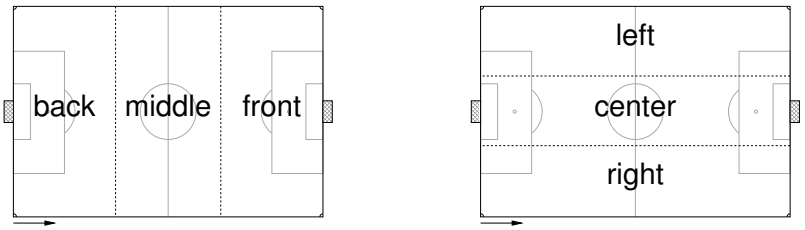
Subsuming, the creation of a scoring opportunity is about establishing ways in which the attacking team can place a player in a position to shoot after an assist. Such a *good* position is either an *unmarked* position *behind* an opponent or a *free* position *near* the opponent's goal, that is in the opposing *penalty area* or at least *close* to it.

We have given a short analysis of the overall structure of a soccer game above which helps us to specify an overall team behavior. Besides, we sketched several possibilities for carrying out each of the sub phases. We want to stress the fact that, throughout the above considerations, the organization of soccer knowledge, in general, and the specifications of tactical patterns, in particular, make extensive use of qualitative notions. That is, spatial as well as other concepts used by soccer experts are quite fuzzy and rather coarse. In the following, we concentrate on these qualitative notions.

5.2 Qualitative Predicates

We now want to consider examples of the description of soccer moves in nowadays soccer theory books more closely. How can these descriptions be used for the behavior specification of an autonomous agent in the RoboCup domain? A major task is to identify and evaluate which qualitative predicates are used within these descriptions, either implicitly or explicitly, and thus are required to specify such a soccer move for autonomous soccer agents. The descriptions in [Luc01] base on several fundamental concepts. Most of them already include qualitative aspects. These concepts are investigated in greater detail now.

Any tactical move presupposes a particular configuration it is used for. The configuration in which a tactical pattern can or rather should be applied is often depicted with a figure illustrating the move's main idea. That is to say, a figure does not show the exact positions of the participating players but it abstracts to a prototypical representation. Thus, a scenario drawn in such a figure stands as a representative for many similar situations.
Naturally, the field of activity in a soccer game is the pitch. All relevant objects are located on it and all actions of interest are bound to it, too. In the majority of cases positions on the field mentioned within the specification of tactical patterns are referred to in terms of a specific part of the field. That is, the playing field is divided into tactical regions. There are at least three rows, that is to say the *defense area*, the *midfield*, and the *offense area*. Since we are looking at the playing field from our team's perspective, we denote these areas by *back*, *middle*, and *front* respectively. Additionally, there are not less than three lanes, namely *left*, *center*, and *right*. By combining the two subdivisions from above we obtain a partitioning of the pitch into nine regions. These regions not only correspond to the different roles of the players in a team but they are also used as addresses in terms of coarse tactical positions. We depict these two field partitionings in Figure 5.4.



(a) zones (b) sides

Figure 5.4: Partitioning of the playing field into zones (a) and sides (b).

Beside the positional information on each object a tactical pattern also contains information about the interplay between multiple objects. There exist several relations between the objects on the field. These relations form the basis for the description of a tactical move. Every move presupposes a spatial configuration for which it can be applied. Such a configuration consists of relative and global positioning relations. For the global case this position relation may be a relation between tactical or semantical regions on the field of play. Regarding the relative case we notice alignment specifications both between players of both types (teammates and opponents) and the ball and between players and integral positions of the field such as the goals or the corners. A player can, for instance, be located *left* and *in front* of another object.
Furthermore, there exist several characteristics associated to an object. These attributes account for spatial properties as well as for tactical peculiarities. That is, for example, a player which is not guarded by opponents is denoted *unmarked* or a region not occupied by any player of the opposing team is referred to as being *free*.

Often, there are additional textual information accompanying a drawing. Similar to the figures, the text also contains qualitative notions and concepts. But, since these notions are expressed in words they are naturally more explicit in written text and we can identify them directly. Basically, the textual descriptions are made up of the same predicates we already acquired in our analysis above.

Table 5.1: A subset of significant words taken from the analysis of newspaper game reports in [Fra98].

category	words
general	side, free, long
basic skills	skills, run, touch, pass, turn, position, possession
attack	attack, header, forward, shot, cross, area, opening, opportunity, wing
midfield	midfield, central, set up, flank, control space
defense	defense, back, clear, mark, save
set pieces	penalty, corner, set, free kick, throw-in

In Section 3.3 we already took a look at Frank's study of newspaper soccer game reports [Fra98]. Now, we want to detail this for our purposes. Frank groups the most frequent words found in a statistical analysis of 2402 articles into the following eight basic features: teamwork, basic skills, playing conditions, objectives, attack, set pieces, midfield, and defense. We are now going to evaluate these categories in order to identify concepts, and notions that are vital for the specification of tactical settings and patterns. Understandably, not all of the words mentioned are of use for us. Hence, we do only consider the ones we can utilize. They are listed in Table 5.1.
Words concerned with environmental properties such as the weather or the ambience created by the audience are not taken into account here. That is because we consider them to be irrelevant for current autonomous agents as they are not really affected from this so far.
As in literature on human soccer theory we notice the frequent use of qualitative notions in Frank's analysis of newspaper reports. Most of them, as in the description of tactical patterns above, are concerned with spatial properties. Others deal with the ontological structure derived in Section 5.1.

Recapitulatory, the main qualitative aspects in soccer theory mostly deal with space. Hence, we have to establish methods and mechanism that provide qualitative representations for spatial data at first.
Most important, we need to represent several forms of positions. That is, positions are referred to from an allocentric point of view as well as relatively to a specific player. The latter can either be in an allocentric or in an egocentric form. Moreover, there are designated semantic regions and positions on the playing field. We have to model the topology on which these elements are based. For instance, a player's position is often referred to as being on the *left side* of the field or being located in the *front part*. As a result of these various forms of positional information which are used in parallel we need to be able to switch between these different forms of pose representation.
The interaction between players demands for spatial and other relations among them and also presupposes several characteristics attributed to them. We identified attributes associated with a particular player and relations between two or more players as well as relations between players and the ball. The most significant attributes are used to characterize, for example, the amount of space available to a player and the exposure to opponent players. These spatial attributes are referred to with concepts like a player being *free* or *unmarked*. Further, relations needed within the specification of a pattern of play include distinct forms of *reachability*. This can be a teammate which is reachable by a pass or it can be a certain position which is reachable by

a dribbling. Other important relations describe the *relative alignment* among several objects on the pitch. The alignment relations are mostly described by spatial properties on how two objects are placed relatively to each other or on how they are positioned related to the field of play. That is, for instance, a player being *in front* or *to the left* of another player.

In order to have a basis for the extraction of qualitative predicates we just investigated the elements of tactical patterns in soccer. All events happen on the playing field and they typically involve one or more *objects*. These objects are either static, e.g. the two goals and the four corner posts or they are dynamic which is the case for the ball and the players. Dynamic objects always have a current state which includes data like the current speed or the direction of a possible movement. Players additionally have a view direction, that is the direction they are looking to.
In order to recognize and classify a specific situation (in terms of a tactical setting) a human soccer player solely relies on his own perception. This is almost the same with soccer agents, either implemented in software or embodied in a mobile robot. All perceptual input is integrated into an agent's world model. This world model stores all properties of the playing field and the objects located on it. Its elements can be organized into a class hierarchy according to their attributes. We depict this hierarchy in Figure 5.5.

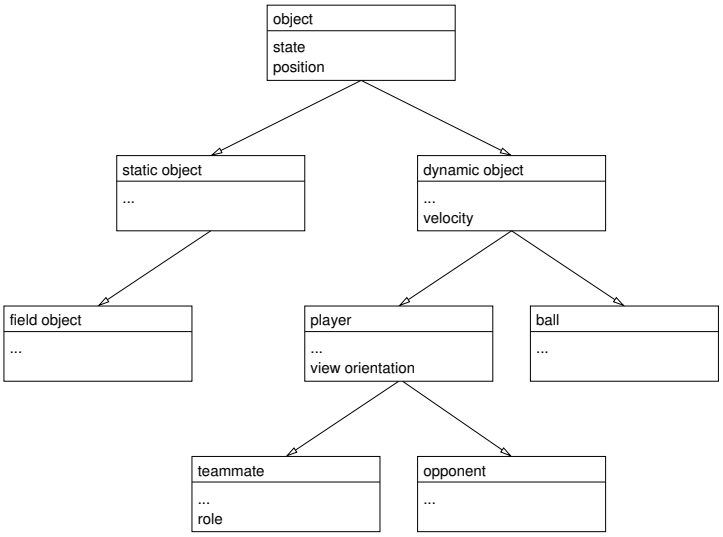


Figure 5.5: World model class hierarchy of the soccer ontology

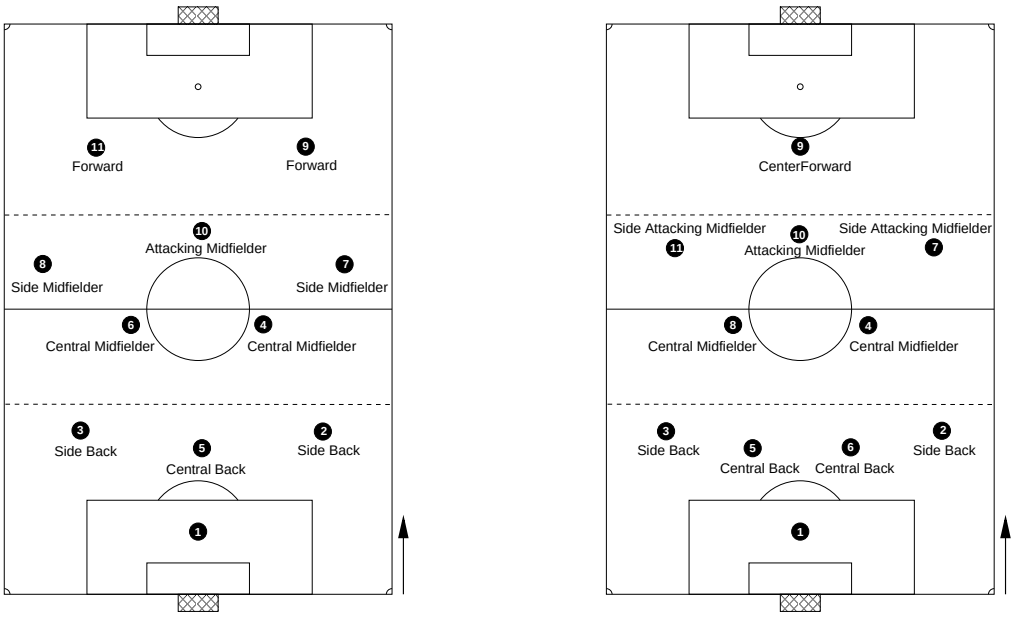
5.3 Adaptation for ROBOCUP

Modern soccer theory addressing coaches of human soccer teams presupposes that a player has several basic abilities and physical skills. These abilities cannot be taken for granted with autonomous agents, in general, and robotic soccer agents, in particular. Besides, strategies and tactical moves in soccer theory books like [Luc01] and [Tal79] are designed for teams of eleven players, whereas a team in the Middle-Size league consists of only five robots. That is why we need to reformulate those strategies in order to fit a team with less than eleven players.
For that purpose, we are now going to formalize the essential components of several tactical patterns. We base our considerations on ideas presented in [DFL+05]. We try to identify key

issues of similar moves through abstraction. That is to say, we abstract from single players in a team of eleven teammates to their respective role or position, whichever is important for the pattern at hand. This makes it possible to instantiate the specific pattern in different leagues almost independent from the actual number of players.
Following [DFL+05] a soccer strategy can be represented as a tuple

$$str = \langle RD, CBP \rangle$$

where RD is a set of *role descriptions* and CBP is a set of *complex behavior patterns*. A role description specifies the required abilities of each player position in relation to the CBP. The complete set of role descriptions can be divided into three subsets, namely the role description for offensive players RD_O, the description for midfield players RD_M, and RD_D, the description for defensive players. This distinction seems to be appropriate as it corresponds both to the field partitions and to the phases of a game revealed earlier. Besides, pursuing our intention to abstract from each single player in an eleven player strategy we can merge several players' role descriptions into groups. These groups serve as a prototype for a role description which is (almost) independent from the overall number of players. Figure 5.6 depicts the player positions in the two systems of play discussed in [Luc01]. The division of roles just mentioned is indicated by two dashed lines splitting the playing field into three parts.



(a) The 3-4-1-2 system of play

(b) The 4-2-3-1 system of play

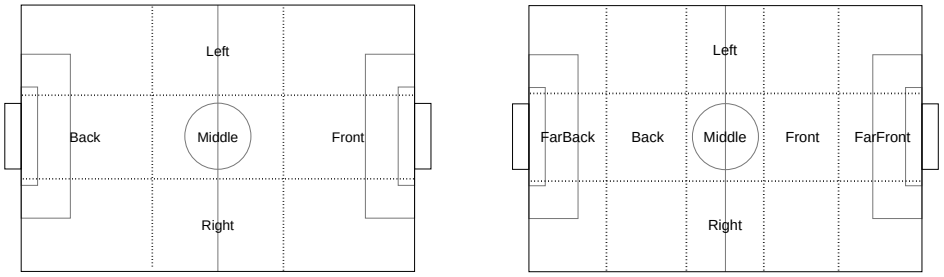
Figure 5.6: Player positions in the 3-4-1-2 and in the 4-2-3-1 taken from [Luc01].

The tactical patterns in [Luc01] base on eleven players. Each one of these players has a specific role. Associated to each role there is a role description which specifies the abilities and the tasks that have to be performed by an individual player carrying this role. But, we have just reduced the set of role descriptions to only three, namely RD_O, RD_M, and RD_D. Consequently, we also have to reduce the complex behavior patterns associated with each of the roles to a set of fewer, more abstract ones. Since we sized down the roles to the groups *attack*, *midfield*,

and *defense*, the interaction between different roles in the behavior patterns are reduced to interactions between these three groups just mentioned.

To retain finer distinctions from the individual behavior patterns in an eleven player setup we differentiate the abstracted patterns for the three newly established coarser groups by the side the particular player is currently playing on. Due to the symmetry of the playing field this can often be limited to a distinction between actions taking place *in the middle* and actions taking place *on the side* of the field. Although, sometimes it is necessary to explicitly have a *left* and a *right* side of the field. With the latter, we can formulate the *on the side* property as a logical disjunction of *left* and *right*.

For our reduced set of role descriptions it is normally sufficient to make three distinctions along the length of the pitch. But, within the specification of tactical patterns this coarse partitioning may not allow for a sufficiently accurate formulation. Therefore, it is reasonable to provide an additional fine grained partitioning which subdivides the playing field into five zones. We depicted the resulting tactical regions of both cases in Figure 5.7.



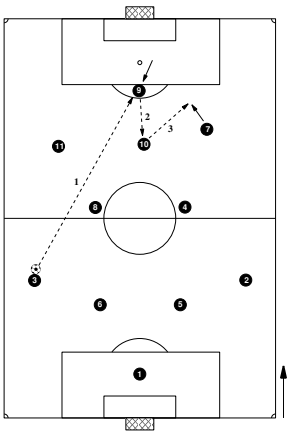
(a) A coarse partitioning of the pitch

(b) A fine grained partitioning of the pitch

Figure 5.7: Tactical regions on the playing field

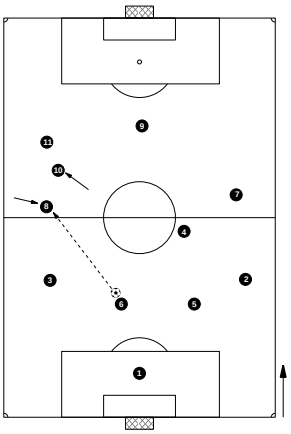
An Example

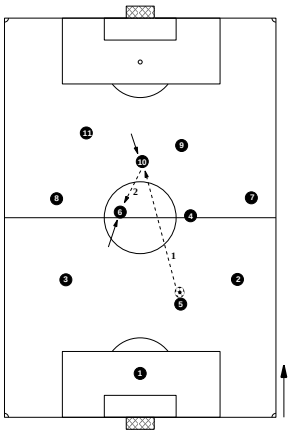
To get an impression on how READYWORLD could help us specifying soccer moves let us consider the three possible ways to build up a play discussed in Section 5.1.



The first way to build up play is with a long pass. We immediately notice that the term *long* is one of the coarse, qualitative notions we need to establish in order to adapt human soccer theory for our autonomous soccer agents. That is, we need a representation for two points or regions being *far* away from each other. Formulated differently, we could state this as passing the ball from a *back* position to a *front* position.

The second way to build up play is with a diagonal pass. This time, the term *diagonal* is of qualitative nature and thus has to be provided in our framework. Diagonal means passing to the side being opposite to the current one. Hence, we need an *opposite* operator for the region system we want to establish. In contrast to the long pass, the diagonal pass is not supposed to reach the *front* part of the pitch but should instead be passed to the midfield.





The last possibility to build up play is with a *deep* pass (dashed line labeled with 1) followed by a subsequent *back* pass (dashed line labeled with 2). The term *deep* is one of the fuzzy concepts in soccer. It is used to denote the space *behind* or *in between* a group of opponent players. Thus, a *deep* pass is a pass through a line of players of the opposing team that should, nonetheless, be received safely by a teammate. Therefore, the endpoint of such a pass has to be the most *free* position available *in between* or *behind* the group of opponents. Note that the notion of *behind* means further up-field with respect to the direction of play.

We now try to adapt as much of these descriptions as possible by integrating their most essential parts into one pattern. First of all, one thing all three possibilities have in common, is that the ball is located in the *back* part on the pitch. According to our abstraction of roles in Section 5.3 it is a player currently having a defensive role which is about to initiate the pattern to build up play. There are three possible ways to build up the play now. We already characterized the possibility of a long pass as 'bringing' the ball to the *front* part of the pitch. Therefore, in this case the agent chooses to pass to a teammate who is located in the attacking zone. For the two other possibilities the pass' destination is the midfield. The agent can either make a diagonal pass, that is the case if the target position is on the *opposite* side of the field, or it can simply pass to a free area on the same side or in the pitch's center. To illustrate our adaptation of the build up play patterns for the MIDDLE SIZE LEAGUE we depicted a diagram similar to the ones in [Luc01] in Figure 5.8.

In order to get an impression of how a specification of the above pattern would look like in our programming language READYLOG we formulated an exemplary program in Figure 5.9. Please note that we do not follow the strict READYLOG syntax here since we just want to give a rough idea on how a possible specification may look like. We are going to present 'real' READYLOG programs in Chapter 7.

Let us briefly step through the specification. In the beginning the agent checks whether it is in possession of the ball. If this is the case it retrieves a free side within the offense part of the playing field. Furthermore, it determines which teammate is a potential $\mathrm{Passpartner}$ on this side in the offensive zone. Subsequently, it starts decision-theoretic planning by using the solve statement. The agent now computes the best course of action for the current situation. It can choose between several actions. If the $\mathrm{PassPartner}$ can be reached with a pass ($\mathrm{isPassReachable}$)

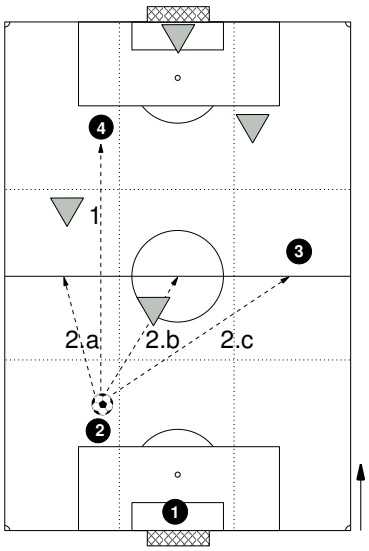


Figure 5.8: A diagram on how to build up a play in the MIDDLE SIZE LEAGUE.

```
proc( build_up_play_defender,
  if( haveBall(ownNumber),
      [ getFreeSide( offense, FreeSide ) ,
        getPassPartner( offense, FreeSide, PassPartner ),
        solve([ if( isNotKickable(ownNumber), intercept),
          nondet( [ if( isKickable(ownNumber),
                        if( isPassReachable( ownNumber, PassPartner ),
                            passTo( ownNumber, PassPartner )
                        ) /* endif */
                    ) , /* endif */
                    pickBest( bestSide, [leftSide,
                                         middleSide,
                                         rightSide],
                              nondet([ [ dribbleTo( ownNumber,
                                                    middleZone,
                                                    bestSide ),
                                         kickTo( ownNumber,
                                                 middleZone,
                                                 bestSide ) ] ]) ),
                    [ intercept,
                      kickTo( ownNumber, middleZone, middleSide ) ]
                  ]) /* end nondet */
          ], 3, func_Reward ) /* end solve with horizon 3 */
      ],
      intercept  /* else intercept */
  ) /* end if haveBall */
). /* end proc build_up_play */
```

Figure 5.9: The buildUpPlay_defender program.

it can perform a pass to this pass partner. The agent can also use the $\mathrm{pickbest}$ statement to obtain the best side in the $\mathrm{middleZone}$. It can either dribble to this position or kick the ball to it. Alternatively, the agent can also intercept the ball and kick it to the $\mathrm{middleSide}$ in the $\mathrm{middleZone}$ of the field. The decision on what to do is mainly based on the reward function which is specified by $\mathrm{func\_Reward}$. We are not going into detail on the reward function here. You may notice, that this specification contains several qualitative elements such as *middleZone*, *leftSide*, and *offense* as well as qualitative predicates such as *isPassReachable*. We simply transfered them from the specification in [Luc01] which we adopted. In the following chapter we will investigate how these notions and terms can be provided by a module called READYWORLD which is to be implemented within the scope of this thesis.

Chapter 6

Qualitative Data Representation

In the following we want to present different possible approaches and formalisms to qualitative data representation and reasoning. We focus on spatial concepts which we consider to be applicable for autonomous soccer agents in a real-time environment.

6.1 Positional Information

Based upon a representation mechanism for qualitative orientation presented by Hernandez in [Her91] and a basic method for qualitative distances discussed in 1995 in [HCF95] by Hernandez, Clementini, and Felici in their 1997 paper [CFH97] Clementini et al. present a unified framework which allows for qualitative representation of positional information. This is done by combining the orientation and distance relations. The position of a *primary object* is represented by a pair of distance and orientation relations with respect to a *reference object*. Both relations depend on a so-called *frame of reference* which accounts for several factors such as size of objects, different points of view, and so on. The framework also features basic reasoning capabilities such as the composition of spatial relations as well as switching between different frames of reference. As we are using Clementini's approach in this thesis, we will now describe it in greater detail.

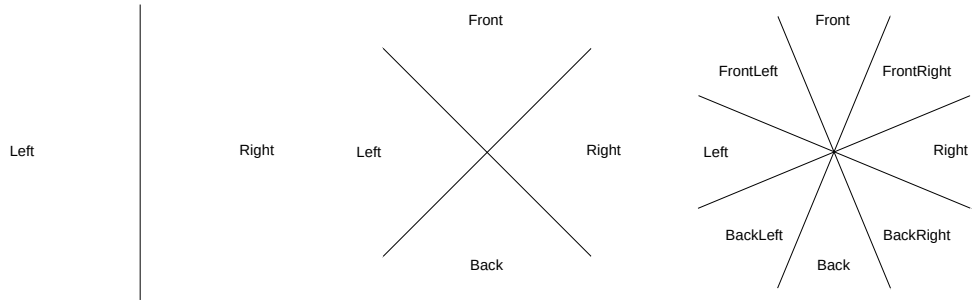
6.1.1 Orientation Relation

Orientation relations are used to describe where objects are placed relatively to each other. Based on the fundamental observation of how three points in the plane relate to each other an orientation relation can be defined in terms of three basic concepts: the *primary object*, the *reference object*, and the *frame of reference* which contains the *point of view*.
The point of view and the reference object are connected by a straight line. The view direction is then determined by a vector from the point of view to the reference object. The location of a primary object is expressed with regard to the view direction as one of a set of relations. The number of distinctions made is determined by the *level of granularity*. We are going to detail this in the following.

Level of granularity

There are different levels of granularity for orientation relations. On the first level the point of view and the reference object are connected by a straight line such that the primary object can

be to the left, to the right, or on that line. Thus the first level partitions the plane into two half-planes. On the second level there would be four partitions, the third level would have eight, and so on. Figure 6.1 depicts the first three levels of granularity for the orientation relation.



(a) level 1 (b) level 2 (c) level 3

Figure 6.1: Different levels of granularity for the orientation relation

A well known example of qualitative directions are the points of a compass which, in the simple case, correspond to the second level: a place can be *north*, *east*, *south*, or *west*. In some cases, we notice even finer distinctions in terms of intermediate directions such as *north-east* or *south-west*. This setting of finer distinctions corresponds to the third level in the above model.

Frames of reference

We already mentioned the term *frame of reference*. It accounts for contextual aspects such as the *front* side of the reference object. Different frames of reference can be classified into three basic types: they are called *intrinsic* if the orientation is given by some inherent property of the reference object, *extrinsic* if a particular orientation is imposed by external factors such as motion, or *deictic* if the orientation is given by the point of view from which the reference object is seen. Based on the frame of reference there is a 'front' side of the reference object. Independent from the level of granularity there is a uniform circular neighboring structure. In general, at a level of granularity k the set $\{\alpha_0, \alpha_1, \ldots, \alpha_n\}$ denotes the $n+1$ orientation relations where $n = 2^k - 1$. Having a reference object A and a primary object B the orientation of B with respect to A is denoted by $\theta_{AB} = \theta(A, B)$. It can take any of the values mentioned in the set above.
Each orientation has a *successor* such that $succ(\alpha_0) = \alpha_1$, $succ(\alpha_1) = \alpha_2$, ..., $succ(\alpha_n) = \alpha_0$ and a *predecessor* with $pred(\alpha_0) = \alpha_n$, $pred(\alpha_1) = \alpha_0$, ..., $pred(\alpha_n) = \alpha_{n-1}$. In addition each orientation α_i has an *opposite* orientation which is obtained by applying $(n+1)/2$ times the function $succ$ to α_i. The minimal number of steps needed to get from orientation α_i to α_j along the circular neighboring structure is called *range*. The range between opposite orientations is $(n+1)/2$. Two orientations are called *orthogonal* if the range between them is $(n+1)/4$ (except for the basic level 1). Each orientation has two orthogonal orientations to it.

6.1.2 Distance Relation

Analogously to the orientation relation establishing a *distance relation* requires three elements: the primary object, the reference object, and the frame of reference.

First of all, we like to describe commonly used distance systems. In a metric space, the distance between points is defined by the following three axioms:

$$dist(P_1, P_1) = 0 \qquad \text{(Reflexivity)}$$
$$dist(P_1, P_2) = dist(P_2, P_1) \qquad \text{(Symmetry)}$$
$$dist(P_1, P_2) + dist(P_2, P_3) \leq dist(P_1, P_3) \qquad \text{(Triangle Inequality)}$$

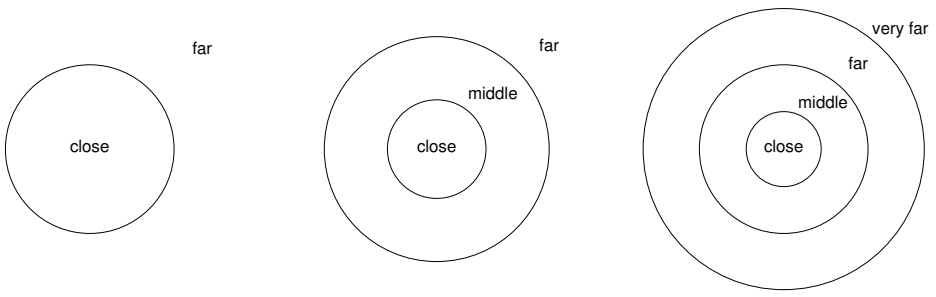
Conventional concepts of distance normally rely on coordinates. The distance between points $P_i = (x_{i1}, x_{i2}, \ldots, x_{in})$ in a n-dimensional vector space can be expressed in terms of the Minkowsky L_p-metric [PS85]:

$$d_p(P_1, P_2) = \Big(\sum_{j=1}^{n} |x_{1j} - x_{2j}|^p\Big)^{1/p}$$

Well known examples are, for instance, the *Manhattan distance* which is defined by the L_1-metric or the *Euclidean distance* which is defined by the L_2-metric.
If we want to use a qualitative concept of distance instead, we usually do not have any coordinate system. The distance of two points expressed in a qualitative way often depends not only on their absolute positions but also on cultural and experimental factors and on the frame of reference.

Level of granularity

Similar to the orientation relation we can distinguish distances at various levels of granularity. An arbitrary level n of granularity with $n+1$ distinctions yields to the set $Q = \{q_0, q_1, \ldots, q_n\}$ of qualitative distances. Given a reference object RO these distances partition the space around RO such that q_0 is the distance closest to RO and q_n the one farthest away.



(a) level 1 (b) level 2 (c) level 3

Figure 6.2: Different levels of granularity for the distance relation

The qualitative distance between the reference object RO and a primary object PO both belonging to a set O of objects is a function $d : O \times O \rightarrow Q$. Let us consider A as the reference object and B as the primary object then $d_{AB} = d(A, B)$ denotes the distance from A to B. There is a total order of magnitude for the elements of Q saying $(q_0 < q_1 < \ldots < q_n)$ which allows us to define a *successor* being the next symbol in the above list, that is $succ(q_i) = q_{i+1}$ for each $i < n$ and $succ(q_n) = q_n$. Accordingly, there is a *predecessor* giving the previous symbol of the list, that is $pred(q_i) = q_{i-1}$ for each $i > 0$ and $pred(q_0) = q_0$. Furthermore

we can introduce a function *ordinal* defined as follows: $ord : Q \rightarrow \{1 \ldots n+1\}$, such that $ord(q_i) = i+1$, and $ord^{-1}(i) = q_{i-1}$, except $ord^{-1} = q_n$ for $i > n$, $ord^{-1}(i) = q_0$ for $i \leq 1$.

Distance system

In order to be able to compare the magnitude of distances, i.e. how they relate to each other, we need to specify several structure relations. They are organized in a so-called *distance system* D defined as:

$$D = (Q, \mathcal{A}, \mathcal{I})$$

where

- Q is the totally ordered set of distance relations;
- $\mathcal{A}$ is an *acceptance function* defined as $\mathcal{A}$: $Q \times O \rightarrow I$, such that, given a reference object RO, $\mathcal{A}(q_i, RO)$ returns the geometric interval $\delta_i \in I$ corresponding to the distance relation q_i;
- $\mathcal{I}$ is an algebraic structure with operations and order relations defined over a set of intervals I. $\mathcal{I}$ defines the *structure relations* between intervals.

Each distance relation can be associated to an *acceptance area*. These areas can be uniquely identified with a series of consecutive intervals $\delta_0, \delta_1, \ldots, \delta_n$ which are called *distance ranges*. There are various interpretations of the intervals $\delta_0, \delta_1, \ldots, \delta_n$. In the case of a strict interpretation we have sharp boundaries and therefore there are points $a_1, a_2, \ldots, a_n \in \mathcal{R}^+$ that make up the intervals $[0, a_1], [a_1, a_2], \ldots, [a_{2n}, +\infty]$. However, cognitive considerations suggest that this is not always the case since intervals may have indeterminate boundaries. The intervals can, then, either be *overlapping* or *non-exhaustive*.
For the overlapping case two consecutive intervals share a common region. This can be represented by as $[0, a_1], [a_2, a_3], \ldots, [a_{2n}, \infty]$ with $a_{2i} \leq a_{2i-1}$ and $a_{2i} > a_{2i-2}$. For the non-exhaustive case, where there is a void space between the acceptance areas of two consecutive distance relations, the intervals would be represented by $[a_0, a_1], [a_2, a_3], \ldots, [a_{2n}, +\infty]$ with $a_0, \ldots, a_{2n} \in \mathcal{R}^+$ and $a_{2i} > a_{2i-1}$ for $i \geq 1$. Any of the above interpretations is possible with the model presented so far as it only uses the notion of consecutive intervals inherited from the total order of $\mathcal{Q}$.
In the following, we briefly sketch an exemplary algebraic structure $\mathcal{I}$. For a detailed discussion of algebraic structures $\mathcal{I}$ we refer to [CFH97]. Consider $\mathcal{I} = (\mathcal{I}, +, \leq, \ll)$ which defines a sum operation ($+$) and two order relations ($\leq, \ll$) over the set I of closed intervals over $\mathcal{R}^+$.
Given two intervals the binary operator $+$ returns the minimal interval that contains both given intervals. Given two intervals $[a, b]$ and $[c, d]$ the operation $+ : I \times I \rightarrow I$ is defined by:

$$[a, b] + [c, d] = [\min(a, c), \max(b, d)]$$

For comparing intervals we have two order relations. Let $\|i\|$ be the function returning the length of an interval i. Then the order relation $\leq$ is defined as follows:

$$i_1 \leq i_2 \Leftrightarrow \|i_1\| \leq \|i_2\|, \forall i_1, i_2 \in I$$

If two intervals have the same length, we say they are *congruent* ($\cong$):

$$i_1 \cong i_2 \Leftrightarrow \|i_1\| = \|i_2\|, \forall i_1, i_2 \in I$$

Let us consider an indistinguishability relation ($\approx$) between the length of intervals which states that no relevant predicate distinguishes between them. We indicate it with $\|i_1\| \approx \|i_2\|$. It allows us to define an order relation *much less than* between lengths:

$$\|i_1\| + \|i_2\| \approx \|i_2\| \Leftarrow \|i_1\| \ll \|i_2\|$$

Then, the order relation *much less than* ($\ll$) between intervals can be defined as follows:

$$i_1 \ll i_2 \Leftrightarrow \|i_1\| \ll \|i_2\|, \forall i_1, i_2 \in I$$

We can also say that the bigger interval *absorbs* the smaller one.
We are now able to define *structure relations* over the domain of intervals. By applying the sum operation to all possible combinations of intervals we obtain a set of intervals Δ which is a subset of I. $\Delta_{h..i}$ is the sum of consecutive intervals δ_k from δ_h to δ_i. If the sum starts from the origin, that is $h = 0$, we abbreviate this to Δ_i which is the distance range from the origin up to and including δ_i. The structure relations of the distance system are the order relations holding between all intervals in Δ.
If the structure relations of the distance domain follow a recurrent pattern they are called *homogeneous*. Examples for distance systems with different homogeneous structure relations are depicted in Figure 6.3.

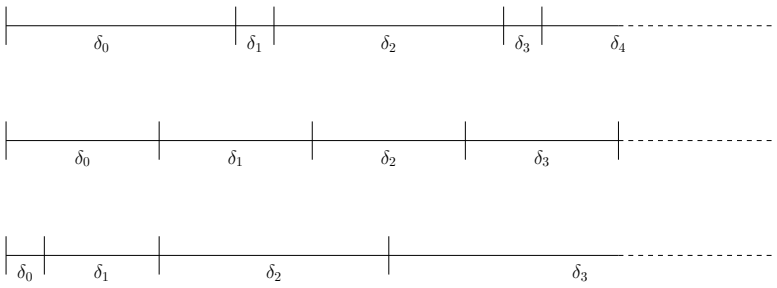


Figure 6.3: Different distance ranges with homogeneous properties

The structure relations are arbitrary, in general. Now we present a restricted set of properties that seems appropriate for our context. By applying several restrictions to the structure of intervals we can achieve fewer indeterminacy in the composition of relations. The constrained properties are motivated by cognitive considerations.
Is is a common phenomenon to make finer distinction in the neighborhood of the reference object than in the periphery. Therefore, a *monotonicity* restriction is established. It constraints the distance domain to increasingly bigger ranges so that any given interval is bigger or equal than the previous one:

$$\delta_0 \leq \delta_1 \leq \delta_0 \leq \ldots \leq \delta_n$$

It is also possible to exclude equally spaced intervals and demand more distant relations to correspond to considerably bigger ranges. This can be achieved by a *range restriction* which states that any given interval is bigger than the entire range from the origin to the previous interval (denoted by Δ):

$$\delta_i \geq \Delta_{i-1} \forall i > 0$$

This can be further emphasized to obtain differences in the *order of magnitude*. It allows us to disregard smaller intervals δ_i with respect to much bigger intervals δ_j for $i < j$. For a given

difference p between the orders of two distance relations q_i and q_j, with $i \leq p \leq (j-i)$, the following holds:

$$(\mathrm{ord}(q_j) - \mathrm{ord}(q_i)) \geq p \Rightarrow \delta_j \gg \delta_i, \forall i, j \geq 0, i < j$$

Thus, a given interval absorbs all the smaller intervals including its immediate predecessor.

Frames of reference

As in the description of qualitative orientation where the frame of reference was meant to fix the 'front' side of the reference object we need to attend frames of reference for the distance relation, too. According to [CFH97] we make use of a more general concept of a frame of reference defined as follows. The distance frame of reference is made up of three components:

$$\mathrm{FofR} = (D, S, T)$$

where

- D is a distance system:
  We already talked about distance systems in the previous section. The distance system contains contextual information which determine relevant distinctions and their structure.
- S is a scale:
  Scale can be understood not only as the proportional relationship of representation to the things it represents but also as proportion between different spatial extensions. The former is the case, for example, with legends like they are used in maps.
- T is a type:
  Analogously to orientation the type can either be
  - *intrinsic*:
    The distance is determined by an inherent characteristics of the reference object. A characteristics can be, for example, its size, shape, or topology.
  - *extrinsic*:
    The distance is determined by some external factor. This could be, for example, the arrangement of objects, the traveling time, or the costs involved.
  - *deictic*:
    The distance is determined by an external point of view. Often this is the case if there is an observer perceiving objects. This is not always meant in the sense of sight but can also be the case for how an individual builds a mental map of the space regarded.

Although each component can principally be combined with any of the other ones all these three components are somehow related to each other. Often the type of FofR influences the scale which itself modifies the distance system.
Since the RoboCup domain is always bound to the playing field (including a certain amount of space for the border area) distances are determined by an external point of view. First, there is a maximal possible distance. That is, no distance can be greater than the length of the diagonal of the pitch. Second, all distances used within the description and the specification of tactical

(or other) patterns in soccer are expressed in relation to the overall size of the pitch. Hence, the type of the frame of reference in soccer, in general, and in ROBOCUP, in particular, is *deictic*.
Following the same motivation as above we can make some remarks about scale. In soccer we can assume that all objects on the pitch have nearly the same size. Therefore, we do not have to consider proportions between different spatial extensions. Moreover, we think it is justifiable to regard the operating range of distinct players being similar enough to discard eventual differences. This is, of course, not always true, neither for two players of different teams nor for two players of the same team. But, since possible differences are disregarded in the descriptions of tactical patterns in human soccer theory books, too, we simply ignore them in our framework.
Lastly, we have to choose an appropriate distance system for our domain. Please, recall that the distance system contains several structure relations in order to enable us to describe how the distance relations relate to each other. We choose a *homogeneous* partitioning, that is, the distance domain follows a recurrent pattern. Instead of using equally spaced distance intervals we think it is more appropriate to make more distant relations correspond to a bigger range. That is not only because the perception of objects and situations farther away is less accurate but also because of a temporal aspect. Since the time needed to get to a place far away is longer than the time needed to get to place close to one's current position there is more time for properties in the environment to change.

Following our above considerations let us examine our way to represent distance in some more detail. Motivated by the cognitively plausible fact that it is common to make finer distinctions in the neighborhood we want to achieve monotonicity. Therefore, we determine our distance intervals as follows. Let base be a scaling factor. Each distance interval is then greater by factor base than its direct predecessor.
As we already stated there is a maximal distance $\mathrm{dist}_{\max}$ due to the fixed size of the pitch. We compute its value from the diagonal of the rectangle formed by the playing field and its additional boundary border.

$$\mathrm{dist}_{\max} = \sqrt{(\mathrm{field}_{\mathrm{width}} + 2 \cdot \mathrm{field}_{\mathrm{border}})^2 + (\mathrm{field}_{\mathrm{length}} + 2 \cdot \mathrm{field}_{\mathrm{border}})^2}$$

We further parameterize our distance system with the *level of granularity* n. At level n there are $n + 1$ distance relations. As you may notice we include an additional interval representing the distance relation $\mathrm{out\_of\_reach}$ which corresponds to all distances greater than the maximal distance $\mathrm{dist}_{\max}$.
The maximal distance can then also be represented by

$$\mathrm{dist}_{\max} = \|\delta_0\| + \|\delta_1\| + \ldots + \|\delta_{n-1}\|$$

Now, we have to subdivide the maximal distance into n intervals δ_i satisfying a monotonicity restriction. We formulated our monotonicity restriction by requiring that each interval is bigger than the preceding interval by factor base. Starting with the first interval δ_0 the length of each subsequent interval δ_i is acquired by multiplying the length of the preceding interval δ_{i-1} with factor base. This can also be expressed as

$$\|\delta_i\| = \|\delta_0\| \cdot \mathrm{base}^i$$

As a result, we can write the above formula as

$$\mathrm{dist}_{\max} = \|\delta_0\| \cdot \mathrm{base}^0 + \|\delta_0\| \cdot \mathrm{base}^1 + \ldots + \|\delta_0\| \cdot \mathrm{base}^{n-1}$$

Unfortunately, we do not know the length of any of the intervals yet. Therefore, we compute the *sum of powers*, that is the sum of the base to the power of the distance relation's index over all distance relations (excluding the additional relation).

$$\mathrm{SumOfPowers} = \sum_{i=0}^{i<n-1} \mathrm{base}^i$$

With this value we can now obtain the length of the first distance interval by

$$\|\delta_0\| = \frac{\mathrm{dist}_{\max}}{\mathrm{SumOfPow}}$$

We can then compute the length of all subsequent intervals as described above.
For simplicity we have chosen a strict interpretation of our distance relations. This leads to sharp boundaries for our distance intervals. Thus, we need exactly n values to represent $\mathrm{number\ of\ distances}$ many intervals δ_i. These intervals are then constituted as follows:

$$[0, a_1], [a_1, a_2], \ldots, [a_{n-1}, a_n], [a_n, +\infty]$$

To obtain the qualitative distance for a quantitative distance value we simply determine to which of the intervals it belongs.

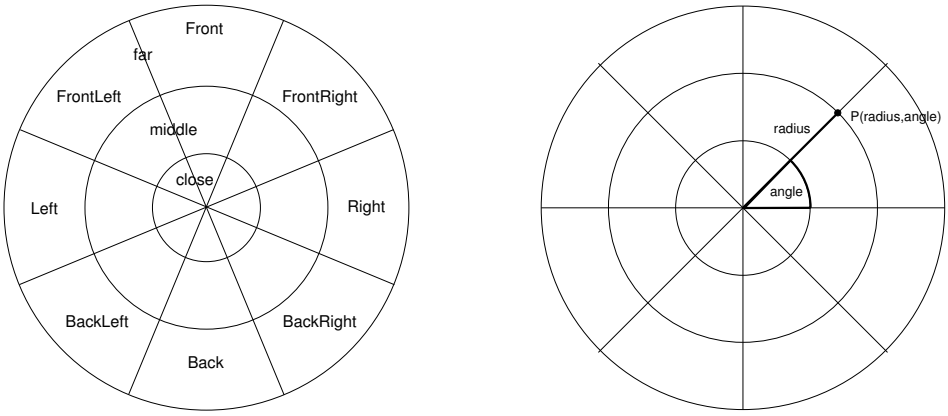
6.1.3 Combination of Orientation and Distance Relation

In order to describe (and reason about) positional information we now investigate the combination of the distance and the orientation relation. Putting together the two relations presented above allows us to represent the location of a primary object B relatively to a reference object A.
From a quantitative point of view, the combined description of a position can be seen as the representation of a point in polar coordinates. The two dimensional polar coordinate system involves the distance from the origin and an angle. A point p in polar coordinates is defined by the distance r from the origin to the point and the angle φ measured from the horizontal x-axis to the line from the origin to p in the counterclockwise direction. Thus, the position of a point p is described as (r, φ). This description corresponds to a combination of the distance relation and the orientation relation which we presented in the previous sections. We depict an illustration in Figure 6.4.
Most of the time, we use the Cartesian coordinate system to represent a position. As we have just seen, the combination of our qualitative distance and orientation relation corresponds to the definition of a point in polar coordinates. Fortunately, there is an easy way to switch between the two systems. If we choose a Cartesian coordinate system with the same origin as with the polar coordinates and if we have the x-axis in direction of the polar coordinates, we have the following formulas for the transformation between the two systems

$$x = r\cos(\varphi) \qquad\qquad y = r\sin(\varphi)$$

and

$$r = \sqrt{x^2 + y^2} \qquad\qquad \varphi = \arctan\frac{y}{x} + \pi \cdot u_0(-x) \cdot \mathrm{sgn}(y)$$



(a) The combination of the distance and the orientation relation

(b) A point p defined in polar coordinates

Figure 6.4: The combination of distance and orientation relation compared to the polar coordinate system.

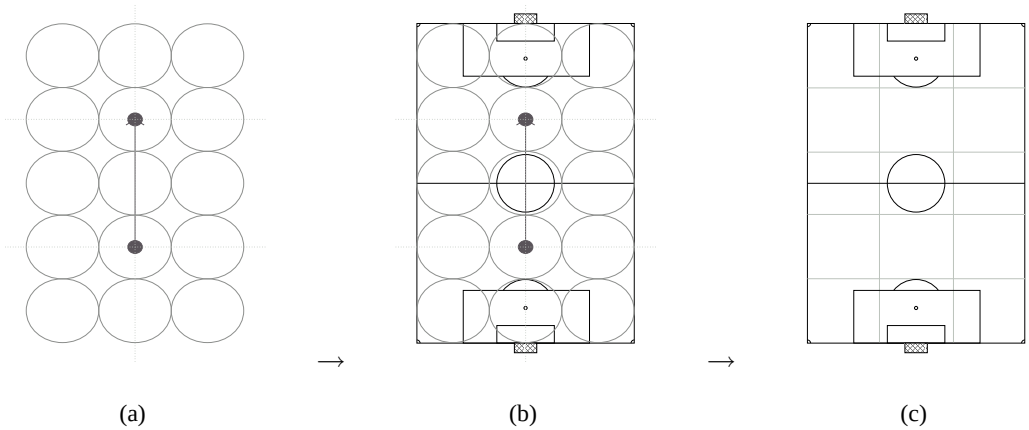
where u_0 is the *Heaviside function* [1] with $u_0(0) = 0$ and sgn is the *signum function*. Here, we use the functions u_0 and sgn as logical switches instead of a distinction of different cases. Most mathematical libraries have a bivariate arcus tangent function $\mathrm{atan2}(y, x)$ that returns the correct value of φ for any x and y.
With the above formulas we are able to transform positional information given in terms of distance and orientation relations to positions in the Cartesian coordinate system we currently employ in our framework. We will detail the execution of this transformation as well as its applications in Section 6.5.
Beside the relative positional information described so far positions are also used globally. We are going to investigate a possible representation approach for this in the next section.

6.2 Semantic Regions

As we already observed in our analysis of soccer theory in Chapter 5 one of the qualitative concepts applied frequently is that of semantic regions on the playing field. These regions are often used as tactical positions corresponding to player roles. That is, there are three zones, namely *back*, *midfield*, and *forward*. Furthermore, there are three sides, viz *left*, *center*, and *right*. Combining the two partitionings above results in a subdivision of the field into nine regions. This seems to be enough to cover the role description task. Although, when specifying soccer moves for a team of autonomous soccer agents, it may be necessary to have a finer distinction. Thus, to retain flexibility we are going to build a representation framework that is adjustable to different demands.
In Chapter 3 we presented Freksa's work on qualitative orientation [Fre92, FZ92]. It is well founded on considerations on human cognition in terms of spatial perception. The orientation grid used for the representation of orientational alignment bases on a (movement) vector going

[1] The Heaviside function, sometimes called the unit step function, is a discontinuous function whose value is zero for negative arguments and one for positive arguments. The value of $u(0)$ can be defined freely; it is often indicated as an index to u, that is u_0 in our case.

from a point A to a point B. Unfortunately, we do not have an explicit movement in the context of global positioning on the playing field. Although, we can regard the direction of play as a vector. From a tactical point of view the focus of a game is to advance from a defensive situation in a team's own half to an offensive one in the opponent's half. Thus, we can take the center of each team's half as the start and end point of our imaginary vector. If we place this vector onto the playing field Freksa's orientation grid yields to 15 regions. The regions and their derivation are depicted in Figure 6.5.



(a) (b) (c)

Figure 6.5: Semantic regions on the playing field. Figure (a) shows the orientation grid taken from [Fre92]. Figure (b) shows the grid embedded into a soccer field. The resulting semantic regions on the playing field are shown in Figure (c).

The regions originating from embedding Freksa's orientation grid onto the playing field are separable by the field's two dimensions. That is, we can carry out a subdivision along of the field along its length and along its width each. This procedure results in *zones* for the length of the field and in *sides* for the width.

Since the *center* of the soccer field is pivotal regarding tactical patterns we ensure that it is available in our qualitative representation, too. Therefore, we take it as the center of both, our zoning and our subdivision into sides. Thus, both the number of zones and the number of sides are odd. We manage the segments in each dimension by their respective indices. The indices are computed starting at zero for the *central* segment.

There is no obvious reason not to choose equidistant intervals for our representation of regions. That is because from a global point of view the playing field's regions do not differ in terms of size or shape. Principally they form an abstraction grid which consists of equally formed sectors subsuming the set of positions of a certain area.

Since we want our regioning to be as flexible as possible we parameterize it by the desired number of distinctions for each dimension of the pitch. Hence, we can determine the length and the width of each region with the following formulas

$$\|\mathrm{region}\|_x = \frac{\mathrm{field}_{\mathrm{width}}}{\text{number of sides}}$$

and

$$\|\mathrm{region}\|_y = \frac{\mathrm{field}_{\mathrm{length}}}{\text{number of zones}}$$

The system sketched above consisting of zones and sides having its center in the middle of the playing field roughly corresponds to a coordinate system. Zones and sides form perpendicular axes with the zones corresponding to the x-axis and the sides corresponding to the y-axis of a Cartesian coordinate system. Thus, we can still specify an object's position in a coordinate system like manner, but by using zones and sides we achieve a qualitative description. In the following section we are going to elaborate on possible applications of the qualitative approach to regions we just presented.

Applications

The main application of the regional model described above is the representation of abstract positional information. Due to the abstraction taking place by referring to a player's position in terms of his zone and his side several similar situations are represented by the same qualitative values. All these situations differ in terms of their numerical values.
For us, this is useful in several ways. First, we are able to abstract from particular situations to more general ones. When we formulate the preconditions required to initiate a tactical move with this qualitative description we cover multiple similar settings in terms of their description in numerical values. That is, we can make use of the coarse and fuzzy positional descriptions in current human soccer theory. Second, we ease the specification process since we are now able to use terms that are much closer to a natural language description than the numerical values are. Moreover, in the majority of cases a tactical instruction just cannot be formulated with precise positions but instead always refers to a set of positions by using the qualitative abstraction of regions such as *front* or *left*. Hence, the descriptions in current soccer literature can be transfered to an autonomous agent's specification more easily with the above region framework.

Beside the greater comfort within the specification of tactical patterns we can use the zoning and siding information to build more complex predicates. Consider, for instance, the course of a soccer game as we described it in Section 5.1. The overall positioning on the pitch is, alongside the ball possession, one of the fundamental indicators upon which to classify whether a team is currently in a defensive or an offensive situation. Moreover, it is an important information whether the game's focus is on one of the sides of the playing field or in the center.
With the zonal information we can provide a qualitative predicate which we call the *game setting*. It expresses where on the pitch the gist of play is. It can be used to derive positioning instructions or simply to call appropriate sub-procedures in the agent's program.
We compute the game setting by a weighted sum of the zone indices of all players and of the ball:

$$\mathrm{weight}_{\mathrm{team}} \cdot \sum_{i \in \mathrm{team}} \mathrm{zone}(i) + \mathrm{weight}_{\mathrm{opp}} \cdot \sum_{j \in \mathrm{opps}} \mathrm{zone}(j) + \mathrm{weight}_{\mathrm{ball}} \cdot \mathrm{zone}(\mathrm{ball})$$

The result of the above formula can take any value in whole numbers. The center of the possible values lies at a value of zero which states that the focus of play is located in the midfield. Negative values indicate that the majority of objects is located in the own half of the field whereas positive values denote the objects to be in the opposing team's half. In order to be able to perform a classification of the game's focal point in terms of a situation being *offensive*, *balanced*, or *defensive* we need to establish a threshold. Upon this threshold we can decide to which of the above class the current situation belongs.

Analogously to the game setting we can determine the gist of play with respect to the sides of the pitch. We call this the *game edge* and compute it with the following formula:

$$\text{weight}_{\text{team}} \cdot \sum_{i \in \text{team}} \text{side}(i) + \text{weight}_{\text{opp}} \cdot \sum_{j \in \text{opps}} \text{side}(j) + \text{weight}_{\text{ball}} \cdot \text{side}(\text{ball})$$

The information provided by the resulting value is classified into the categories *left*, *center*, and *right* in the same way we described for the game setting. The game edge renders useful, for instance, if we need to decide which side of the pitch is less occupied and can thus be used to advance into the opponent's field half with lesser risk of being attacked.

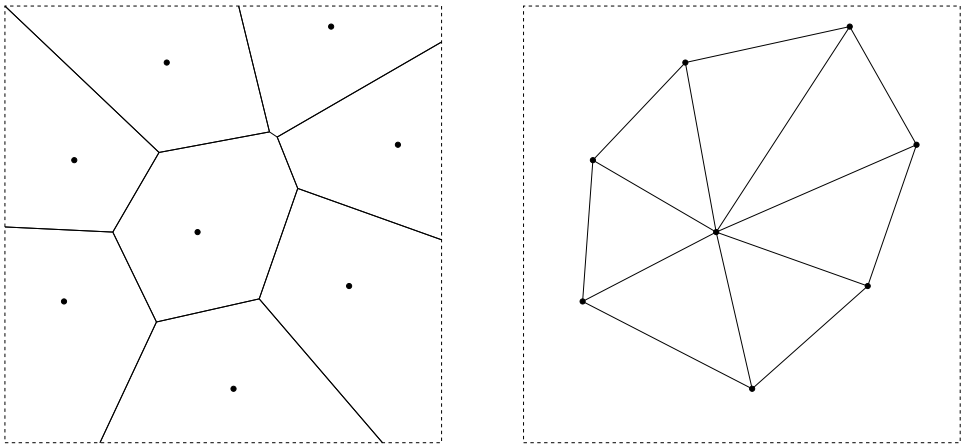
Apart from the more or less static regions established in this section there are regional aspects which can be useful to determine spatial properties and to represent spatial knowledge. In the next section we present a mathematical structure, namely that of Voronoi diagrams, that allows for computing and representing some of these dynamical regional aspects.

6.3 Voronoi Diagrams

The origins of Voronoi diagrams date back to the 17th century. Descartes, for example, used a decomposition of space into convex regions to describe his idea of the extension of regions in the solar system. The principle idea is the following. Let a space M and a set S of sites p in M be given. Further, a site p exerts an *influence* on a point x in M. Then the *region* of p consists of all points x for which the influence of p is the strongest, over all $s \in S$.
The concept sketched above has independently emerged and proven useful in various fields of science. The mathematicians Dirichlet and Voronoi were the first to formally introduce it. Today's standard names for structures of the form described above are hence called *Dirichlet tessellation* or *Voronoi diagram*. We depict an example Voronoi diagram in Figure 6.6(a).
The dual of a Voronoi diagram is a structure in which any two point sites are connected that have a common boundary in the Voronoi diagram. This structure was first introduced by Voronoi himself. Later it was extended by Delaunay and is therefore commonly referred to as *Delaunay triangulation*. Figure 6.6(b) shows an exemplary triangulation. Delaunay used the so-called empty circle method: Consider all triangles formed by the sites such that the circumcircle of each triangle is empty of other sites. The set of edges of these triangles forms the Delaunay triangulation of the sites.
Besides applications in other fields of science, the Voronoi diagram and its dual can be used for solving numerous and very different problems in the field of computational geometry. The most prominent application of Voronoi diagrams surely is the determination of the closest site of an arbitrary point in the plane. This type of query is used to retrieve a *nearest neighbor*. It is of interest in many commercial settings such as the coverage of shops for a certain area. We are going to discuss applications in our framework in Section 6.3.3. The Delaunay triangulation, for instance, can be used to determine largest empty circles within a set of point sites. It can also be used to find closest pairs of sites. Let us now survey the formal definition of Voronoi diagrams and their dual, the Delaunay triangulation.

6.3.1 Definition

Let S be a set of $n \geq 3$ distinct point sites $p_1, p_2, \ldots, p_n$ in the plane. Denote the Euclidean distance between two points p and q by $dist(p, q)$. For points $p = (p_x, p_y)$ and $q = (q_x, q_y)$



(a) Voronoi diagram (b) Delaunay triangulation

Figure 6.6: An example of a Voronoi diagram and its dual, the Delaunay triangulation of 7 points in the Euclidean plane

in the plane we have $dist(p,q) = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2}$ (confer Section 6.1.2). Let $\overline{pq}$ be the line segment from p to q and let $\overline{A}$ be the closure of a set A. For $p, q \in S$ let

$$B(p,q) = \{x \mid dist(p,x) = dist(q,x)\}$$

be the *bisector* of p and q. $B(p,q)$ is the perpendicular line through the center of the line segment $\overline{pq}$. It separates the half-plane

$$D(p,q) = \{x \mid dist(p,x) < dist(q,x)\}$$

containing p from the half-plane $D(q,p)$ containing q. We call

$$\mathrm{VR}(p,S) = \bigcap_{q \in S, q \neq p} D(p,q)$$

the *Voronoi region* of p with respect to S. Finally, the *Voronoi diagram* of S is defined by

$$V(S) = \bigcup_{p,q \in S, p \neq q} \overline{\mathrm{VR}(p,S)} \cap \overline{\mathrm{VR}(q,S)}.$$

By definition, each Voronoi region $\mathrm{VR}(p,S)$ is the intersection of $n-1$ open half-planes containing the site p. Therefore, $\mathrm{VR}(p,S)$ is open and convex. Different Voronoi regions are disjoint.
The common boundary of two Voronoi regions belongs to $V(S)$ and is called a *Voronoi edge* if it contains more than one point. If the Voronoi edge e borders the regions of p and q then $e \subset B(p,q)$ holds. Endpoints of Voronoi edges are called *Voronoi vertices*. They belong to the common boundary of three or more Voronoi regions.

Let us now consider the dual of a Voronoi diagram, the *Delaunay triangulation*. The Delaunay triangulation $\mathrm{DT}(S)$ of a set of points S is obtained by connecting with a line segment any two points $p, q \in S$ for which a circle C exists that passes through p and q and does not contain any other site of S in its interior or boundary. The edges of $\mathrm{DT}(S)$ are called *Delaunay edges*.

6.3.2 Algorithms

As we stated above, the Voronoi region for each point site p in a Voronoi diagram (with n point sites) can be computed as the common intersection of all half-planes $D(p, q)$ with $p \neq q$. This would take us $O(n \log n)$ time per Voronoi region. Since the complete Voronoi diagram has n point sites it would take us $O(n^2 \log n)$ time to compute the whole Voronoi diagram. Fortunately, there is an algorithm commonly known as *Fortune's algorithm* - named after its inventor - which computes the Voronoi diagram in $O(n \log n)$ time. As Fortune's algorithm is optimal, there is no algorithm that computes a Voronoi diagram faster. We now briefly sketch this algorithm also known as *plane sweep algorithm*.

The strategy is to sweep a vertical line - the so-called *sweep line* - from left to right over the plane. In doing so vertices and edges of point sites already visited are emitted. But the edges of a point site already start left of it. To circumvent this problem we proceed as follows. Instead of maintaining the intersection of the Voronoi diagram with the sweep line SL, we maintain information about the part of the Voronoi diagram of the point sites left of SL that cannot be changed by point sites that lie to the right of SL.

The distance of a point q left of SL to any site located right of SL is greater than the distance of q to SL. Hence, the nearest site of q cannot lie right of SL if q is at least as near to some site p_i left of SL as q is to SL. The locus of points which are closer to some site p_i left of SL than to SL is bounded by a parabola. Thus, the locus of points that are closer to any site left of SL than to SL itself is bounded by parabolic arcs. This sequence of parabolic arcs is called *beach line*. The beach line is y-monotone, that is every horizontal line intersects it in only one point. We maintain the beach line BL as we move our sweep line SL. The *breakpoints* between the different parabolic arcs of the beach line lie on edges of the Voronoi diagram since these breakpoints have exactly the same distance to two of the point sites.

The structure of BL changes when one of two types of event occur. That is, when a new parabolic arc appears on it and when a parabolic arc shrinks to a point and disappears. A new arc appears when the sweep line hits a new point site. This is called *site event*. An arc disappears when at least three parabolic arcs meet in one point. This is called *circle event*.

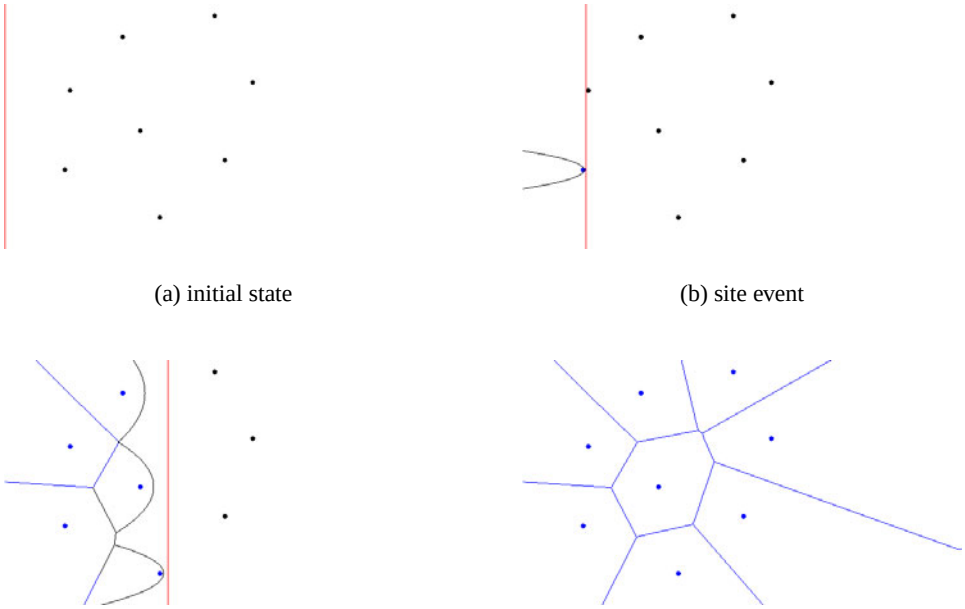
To be more precisely, while the sweep line is moving from left to right the following two types of events occur:

Site Event The sweep line hits a new point site. A new parabolic arc appears on the beach line and a new edge of the Voronoi diagram starts to be traced out.

Circle Event A parabola disappears from the beach line. That happens when three (or more) parabolic arcs intersect in one point. This point then is a new vertex v of the Voronoi diagram. Suppose the three intersecting parabolas belong to the three point sites p_a, p_b, and p_c than v is the center of a circle through p_a, p_b, and p_c.

We depicted an exemplary run of Fortune's algorithm in Figure 6.7. As soon as all point sites have been detected $\mathrm{V}(S)$ is obtained by removing the beach line and extending all spikes to infinity.

There are several other algorithms to construct a Voronoi diagram. For a more detailed account we refer to [AK00].



(a) initial state (b) site event

(c) circle event (d) final state

Figure 6.7: An exemplary run of Fortune's algorithm. (a) shows the initial situation. (b) depicts the situation when the first site event happened. In (c) a circle event occurred. (d) is the final situation after the sweep line passed the whole plane.

6.3.3 Applications

In the following we will point out some of the numerous applications of the Voronoi diagram $\mathrm{V}(S)$ and its dual, the Delaunay triangulation $\mathrm{DT}(S)$ that are of use for us in representing and computing qualitative properties.

Reachability

We already stated that we consider the concept of reachability to be central for the description and the execution of tactical patterns in soccer. There are several forms of reachability, namely the goto reachability, the pass reachability, and the dribble reachability. For all these different reachability relations the individual abilities of a single robot or agent are relevant since even within the same league players of different teams may have unequal capabilities in terms of speed and mobility. As for the interception of a pass, for example, this certainly influences the truth value of an $\mathrm{is\_reachable}(x, y)$ relation. Nevertheless, there are some basic properties which are shared by all of the above relations in multiple settings.
We think the concept of Voronoi diagrams to be applicable for modeling a simplified model of the reachability relations required. Remember that in a set of point site S the Voronoi region of a point site s consists of all points p that are closer to s than to any other point site of S. Furthermore, in the dual to the Voronoi diagram $\mathrm{V}(S)$, the Delaunay triangulation $\mathrm{DT}(S)$, two point sites are connected if and only if they share a common boundary in the Voronoi diagram. Let us now consider the Voronoi diagram and the Delaunay triangulation for the geometric structure of all players on a soccer playing field. We take the players as point sites in the plane and construct $\mathrm{V}(S)$ and $\mathrm{DT}(S)$ with the Euclidean distance thereupon. Then, the Voronoi

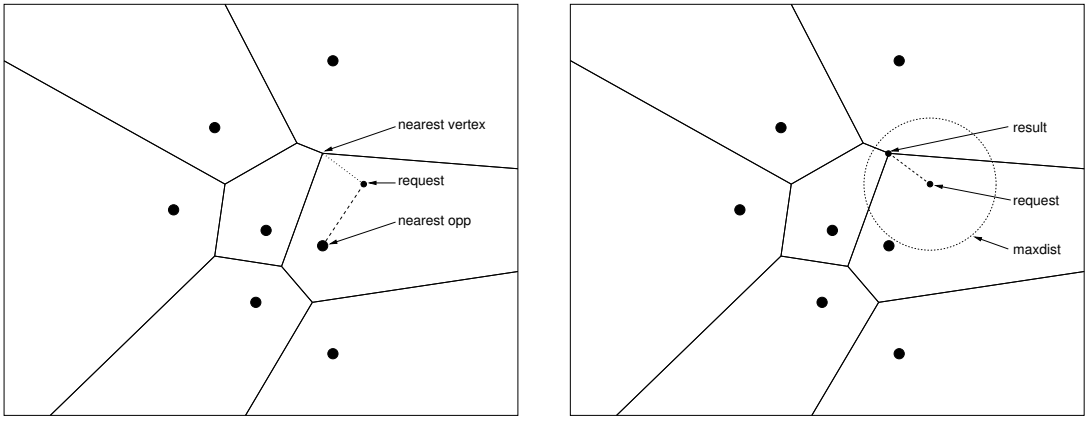
region of each player is the set of points closer to this player than to any other player. Our idea of modeling reachability with the aid of $\mathrm{V}(S)$ and $\mathrm{DT}(S)$ is to consider players to be reachable to each other if their Voronoi regions share a common boundary, that is, they are connected in the Delaunay triangulation $\mathrm{DT}(S)$.

Free Space

The notion of *free space* is another important aspect which can frequently be found in the description of spatial settings and tactical patterns in soccer. The term *free space* denotes an area which is not occupied by any of the players of the opposing team. We think that Voronoi diagrams can help us to determine such free spaces. The cells in a Voronoi diagram represent the area consisting of all points which are closer to the corresponding point site than to any other point site. Consequently, the Voronoi edges are formed by points that are equally far away from the two point sites the edge is between and vertices reflect points in the plane which have the maximal possible distance to even three or more of the surrounding point sites. We now want to take advantage of this fact.

Consider the Voronoi diagram being constructed upon all opponent players on the field. In this diagram, the vertices correspond to those points on the pitch that three or more opponents at the same time are farthest away from. In the following we will not take the edges into account. Instead, we include the four corner points of the playing field into the Voronoi diagram. This enforces that we have not less than three point sites so that there always is at least one Voronoi vertex. The Voronoi vertices directly match our interpretation of positions in free space as we sketched it above. We provide two distinct ways to employ these vertices.

First, we consider a classification request. Given a point on the playing field, we can ask how 'free' this point is. To answer such a request we compute the point's distance to the nearest point site in the opponent's Voronoi diagram as well as its distance to the nearest Voronoi vertex. The ratio between these two values is a good criterion on how 'free' the given point is. Of course, the ratio is not always a sufficient indicator since it does not reflect the absolute distance to an opponent. Therefore, we additionally take account of a minimal distance which has to be exceeded for a position to be classified as being free.



(a) classification request

(b) point request

Figure 6.8: Different requests for free space.

Second, we can answer inquiries for free point positions. Most times it is reasonable to specify a region of interest in which to search for a free position. For simplicity we assume that this region of interest is specified by a position in the pitch's coordinate system (along with a maximal distance to search up to). Given this query, we determine the nearest Voronoi vertex to the position. If the distance between the query point and the vertex is lower than the maximal distance we return the vertex' position. Otherwise, we return the position of a point lying on a line starting at the query point going into the direction of the nearest Voronoi vertex. We depict an illustration for both queries in Figure 6.8.

Ball Possession

Within the course of a soccer game it is a vital information whether or not one's team is in possession of the ball. That is, again, an information we can provide by utilizing the structure of Voronoi diagrams.
The simplest way to answer the question of ball possession is to check if the ball is located in the Voronoi region of a player who belongs to one's own team. This is, of course, not always correct. For example, if the player whose Voronoi cell the ball belongs to is not facing the ball it might be the case that another player who has a greater distance to the ball but who is directly facing it can reach it more quickly. It is, however, possible to take this additional information into account and to refine the predicate accordingly.

6.4 Additional Features

Apart from the spatial representations mentioned and modeled so far there are other predicates that would be useful and could help a designer within the specification of an autonomous soccer agent. Most of them are at least partially concerned with spatial properties as well.

Passway Vacancy

As an additional qualitative predicate of particular interest in the soccer context we now consider something we call *passway vacancy*. We denote a qualitatively abstracted classification of the amount of space available along a potential pass way by this predicate. That is to say, we classify the degree of exposure of a line segment going from point P_{start} to point P_{end} by examining possible sticking points or points of interception.
We derive our classification by considering a ratio on how likely an interception is. Consider a straight line from P_{start} to P_{end}. We compute the minimal distance of each opposing player to this line, that is either the length of a line perpendicular to the passway or the distance to the pass way's nearest end point. Further, we compute the distance from each opponent to the starting point of the pass way. Then, we calculate the ratio between this two values. That is to say, we determine if the opponent is so close to the pass way that it can intercept a ball passed along the pass way. Figure 6.9 shows an illustration of the calculations performed when determining a $\mathrm{PasswayVacancy}$.

Marking

As we already pointed out in Chapter 5 another information that is beneficial for the specification of tactical patterns can be provided by the *unmarked* predicate. It is used to state whether

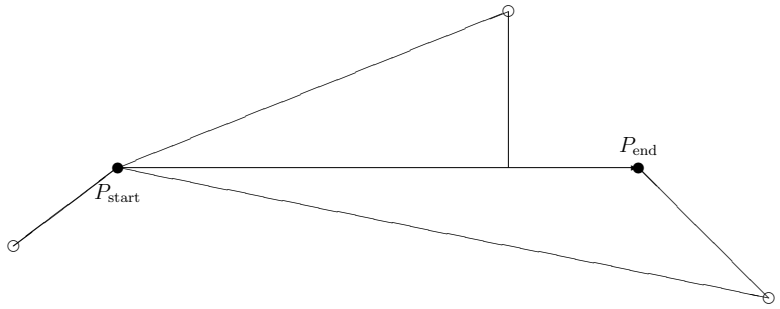


Figure 6.9: An exemplary of the calculation performed to determine the *passway vacancy*.

a player is *free* or covered by an opponent. This information is essential, for instance, to decide if a pass to this player makes sense or not. A simple way to answer this question is to calculate if there are any opponents within a certain distance to the player desired.

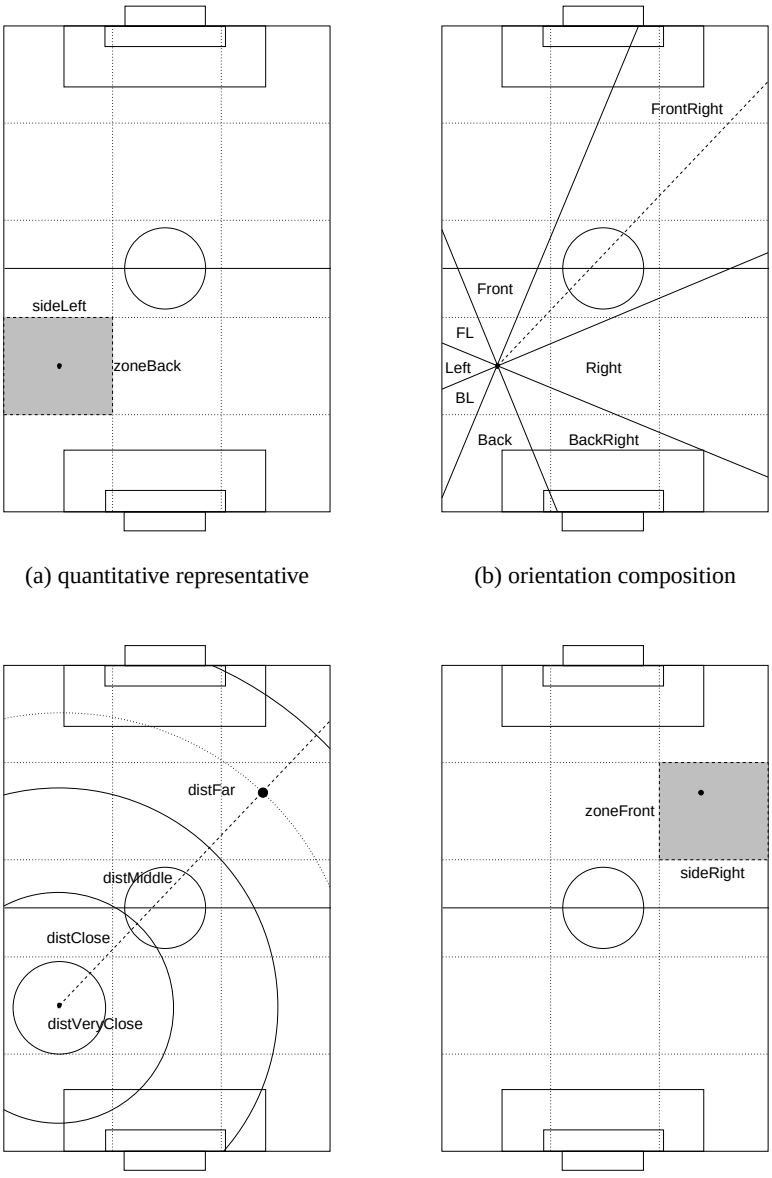
6.5 Reasoning Approach

With the representations and predicates presented so far it is possible to describe situations, that is, spatial settings, in a qualitative manner. But, in order to derive new information and instructions from these descriptions it is necessary to reason with qualitative spatial notions and predicates. Consider, for instance, a situation where the agent is located in the back part of the field on the left side. If we want to know where the agent will end up when driving far to the $\mathrm{front} - \mathrm{right}$ direction we need to perform some kind of reasoning.
In Section 3.1 we already took a look at some systems that deal with qualitative spatial reasoning. Most of them have in common, that they employ a special calculus for reasoning. But, most of these calculi are computationally expensive. Reasoning in the RCC-8, for instance, is NP-complete, in general, like Nebel and Renz show in [RN99]. Therefore, unlike the other approaches mentioned so far, we are not using any calculi for spatial reasoning. Instead, we take advantage of the fact that our world representation always contains numerical data and we try to use the reasoning facilities provided in our READYLOG framework.

Naturally, the sensory information acquired by an agent is numerical. Moreover, numerical data is and will always be required for the execution system of an agent, in general, and for the actuators of our robots, in particular. Thus, we will not be able to set aside all these numerical information. This does not necessarily turn out as a disadvantage, though. So far, the world model of a soccer agent in our framework only consisted of quantitative data.
Reasoning in READYLOG with these quantitative data works through its internal projection mechanisms. That is, every action has a model which describes the effects of this action on the world. If we start planning, these models are applied to the data that the agent had when it started planning. Consider, for example, an agent being at the global coordinates (x_{curr}, y_{curr}). Let $\mathrm{go}(\mathrm{x_{rel}}, \mathrm{y_{rel}})$ be an action that lets the agent go by (x_{rel}, y_{rel}) relatively to its current position. A model of this $\mathrm{go}(\mathrm{x_{rel}}, \mathrm{y_{rel}})$ action in READYLOG would thus project the agent's position for the above go action to the location $(x_{currl} + x_{rel}, y_{curr} + y_{rel})$.
This works, however, for quantitative data. But, we just enhanced the world model by qualitative values. We presented how we compute qualitative abstractions upon numerical data in the previous sections of this chapter. Within an agent's behavior specification which uses qual-

itative predicates it will be necessary to be able to reason about these qualitative notions, too. Therefore, beside the possibility to retrieve a qualitative abstraction for a numerical value we also provide an additional mechanism to access a numerical representative value for each of the qualitatively abstracted values. This enables us to apply READYLOG's projection facilities to our newly established qualitative notions, too.
If we want to compose, for instance, two qualitative distances, we proceed as follows: First, we retrieve quantitative representatives for each of the two distances. Then, we compute the composition of the quantitative values by a simple addition as before. Finally, we re-transform the result into a qualitatively abstracted value. This can be done in the same way we compute the qualitative abstraction in the first place.
Almost all qualitative abstractions we presented in this chapter are based on intervals. That is, in the set of qualitative values each of these values is made up of an upper and a lower bound. We can calculate a representative for a qualitative abstracted value by simply taking the center of its describing interval. This way, we obtain a quantitative value. Furthermore, by taking the center, we also minimize the possible error, since the overall maximal error in such an interval is half of the interval's length at most.

The approach presented above enables us to reason about compositions of different types of qualitative information for all cases where a possibility to compose their quantitative representatives exists.
In Section 6.1 we introduced the representation of positional information through a combination of orientation and distance taken from [CFH97]. We also mentioned that this combination corresponds to the polar coordinate system. Furthermore, there are simple mathematical ways to make a transformation between the Cartesian and the polar coordinate system. We already stated the corresponding formulas in Section 6.1.3. And, in Section 6.2 we described our approach to semantic regions and we showed that it has a coordinate system like representation. Together with the hybrid system we just brought up this allows us to combine information that are given in different representations.
Let us reconsider the example from the beginning of this section. The agent is located in the back part of the field on the left side. We want to know where the agent will end up by driving far to the $\mathrm{front}-\mathrm{right}$ direction. Following our above approach we retrieve representatives for all of the qualitative notions to reason about. First, we need to obtain the representative for the position $[\mathrm{zoneBack}, \mathrm{sideLeft}]$. Then we retrieve a numerical value for the $\mathrm{FrontRight}$ orientation as well as metric distance for the Far abstraction. With the formulas for switching between the polar and the Cartesian coordinate system we are able to obtain relative x and y coordinates which we can add to the agent's initial position. Lastly, we abstract the resulting quantitative position to our qualitative regions again. This yields to $[\mathrm{zoneFront}, \mathrm{sideRight}]$ in our example. To get a visual impression we depicted the procedure described above in Figure 6.10.
READYWORLD provides procedures which transform quantitative values into qualitative representations, and vice versa. Thus any of the existing models can be enhanced to handle qualitative data by integrating a lookup which transforms qualitative data into quantitative ones where it is needed. If this is the case, the output of the model is re-transformed to a qualitative measure after finishing the computations within this model.



(a) quantitative representative

(b) orientation composition

(c) distance composition

(d) qualitative abstraction

Figure 6.10: An exemplary projection of qualitative values in READYLOG.

Chapter 7

A Qualitative Soccer Agent

In this chapter we present exemplary parts of a prototypical soccer agent that makes use of the qualitative predicates we presented in Chapter 6. The agent specification is done in the READYLOG framework which READYWORLD has been integrated in. First, we investigate the integration of qualitative predicates in READYLOG. Then, we depict some general results about the integration of READYWORLD. Afterwards, we consider the results of an agent in the MIDDLE SIZE LEAGUE and we also consider its application in the SOCCER SIMULATION LEAGUE.

7.1 Qualitative Concepts in READYLOG

The qualitative concepts and predicates which we presented in Chapter 6 are computed by the module READYWORLD. The agent specification in our framework is done in READYLOG, that is the logical programming language we introduced in Chapter 4. If we want to use the concepts, functions, and predicates provided by READYWORLD within a READYLOG program we have to establish a connection between the two somehow.

In our framework we realized this with the so-called KNOWLEDGE LIBRARY (KL). The KL provides the qualitative predicates and functions by a string to function mapping. That is to say, there is a PROLOG predicate $\mathrm{kl\_get\_value}(\mathrm{string}, \mathrm{return\_value})$ which passes a query string to the KL which in turn dispatches the query to a READYWORLD function. The return value is then passed back to READYLOG via the KL. We do not go into detail here but we focus on the application. For a more detailed view on implementational aspects we refer to Chapter 8.

Following our reasoning approach which bases on the fact that we hold hybrid data, we provide two classes of functions. The first class of funtions is used to switch between quantitative data and their corresponding qualitative abstraction. For instance, the designer can retrieve the side of an arbitrary object by simply feeding its y-coordinate to the function $\mathrm{getSide}$ which then returns the qualitatively abstracted value. The second class offers functions for retrieving quantitative representatives for qualitative predicates. They can be used to retrieve a numerical representative, for example, in order to instruct the camera to look in a certain direction. In such a case the programmer could write $\mathrm{getOriVal}(\mathrm{oriLeft})$. We depict some prominent functions for each class in Figure 7.1.

In the following we discuss some general properties of the newly established qualitative facilities we added to the READYLOG framework.

```
zone(  X, Zone ) → [...] ;
side(  Y, Side ) → [...] ;
dist(  D, Dist ) → [...] ;
 ori(  O,  Ori ) → [...] ;
```

(a) Functions to retrieve qualitative representations for quantitative data

```
zoneX( Zone, X ) → [...] ;
sideY( Side, Y ) → [...] ;
distD( Dist, D ) → [...] ;
 oriO(  Ori, O ) → [...] ;
```

(b) Functions to retrieve quantitative representatives for qualitative values

```
bool GetTeammateIsReachable( int number );
bool GetHaveBall();
int  GetBestInterceptor();
int  GetGameSetting();
int  GetGameEdge();
int  GetPasswayVacancy( Coord from, Coord to );
```

(c) Qualitative predicates

Figure 7.1: Some prominent predicates and functions to retrieve qualitative representations for quantitative data

7.2 General Discussion

One of our aims was to achieve more comfort within the specification of autonomous soccer agents. We meant to do this, first, by providing a more natural vocabulary in the programming language in terms of qualitative notions and concepts that are commonly present in human cognition. Second, we want to enable more comfort in the specification by providing notions that abstract from numerical values and thus cover for multiple similar situations. Both achievements should enable a designer to transfer his knowledge of soccer into the program of an agent more easily. With the concepts we established in the previous chapter we think we accomplished both these goals. That is because we made most of the key issues available that are used in the specification of soccer tactics and we also placed general qualitative notions and mechanisms for spatial descriptions and operations at the disposal.

One of the key issues in human soccer theory certainly is the overall structure of a soccer game. We took a look at these structural properties in Chapter 5. There, we already stated that the investigations on the course of a soccer game help us to specify the overall structure of an agent's program. But, in order to do this we need to determine the current state of the game. That is to say, we need to classify the current situation in matters of the subphase we are situated in.
For this purpose we can use both, combinations of basic predicates and complex predicates provided by READYWORLD. In the above case of determining the current subphase within the course of a soccer game these are the predicates $\mathrm{gameSetting}$, $\mathrm{gameEdge}$, and $\mathrm{haveBall}$. The most striking indicator doubtlessly is the possession of the ball. Thus, we can say that we are in the defensive subphase if we do not have the ball. But, the ball's position on the field and the overall positioning of the players also add to the decision on the current subphase.
If we do have the ball the current positioning is even more decisive. Having the ball in the defense area surely requires to build up the play whereas having the ball in the midfield rather

demands for the creation of a scoring oppurtunity. Being in possession of the ball in the offensive part of the playing field is a situation in which to try to score. The constructs of READYLOG together with our newly established qualitative information now allow for an easy way to transfer this knowledge into an agent's specification. We depict a code fragment where this is exemplarily done in Figure 7.2.

```
if( f_HaveBall, %% check ball possession
    or( [ %% check setting offense/midfield/defense
          if( f_Setting = settingDefense,
              proc_BuildUpPlay
          ),
          if( f_Setting = settingMidfield,
              proc_CreateScoringOpportunity
          ),
          if( f_Setting = settingOffense,
              proc_Score
          )
        ] ),
    %% else: (re)gain ball possession
    proc_GainBallPossession
  ),
```

Figure 7.2: A code fragment on how to switch between different sub-phases in a soccer game.

Following the toplevel classification within the course of the game we can call corresponding sub-procedures in our agent's program. Since we distinguish between different roles in our specification, we can subsequently differentiate by the roles to call specific procedures implementing the according complex behavior patterns.

Up to this point, the above considerations are mostly independent from the league. In the following we study the MIDDLE SIZE LEAGUE in greater detail in order to investigate how a potential application of the enhancements provided by READYWORLD may look like.

7.3 Middle-Size Robot League

To demonstrate the application and the intended practicality of our approach we are now going to investigate parts of an autonomous agent for a mobile robot in the ROBOCUP MIDDLE SIZE LEAGUE. The agent specification makes use of the qualitative notions and concepts developed in this thesis.

7.3.1 General Remarks

In the MIDDLE SIZE LEAGUE maintaining a fully functional team of up to five mobile robots is not always that easy. That is because the robots are mostly experimental systems where recent research is tested. Moreover, to keep pace with other teams one has to adapt changes very quickly. This leads to losses regarding the stability since there is not enough time to conduct

exhaustive tests. Nonetheless, we tried to evaluate our work in the MIDDLE SIZE LEAGUE due to the fact that this is the league we (the ALLEMANIACS) are competing in.
In order to be independent from potential hardware failures or other difficulties we established a simulation environment. We elaborate on that in Chapter 8. With the simulation framework we are able to set up a complete team of robots with which we can test an agent's behavior in an arbitrary situation. Most of the examples presented here were evaluated in this simulation environment but could have been done in a real game situation just as well. Nonetheless, we chose the simulation environment because it offers a higher degree of reproducibility. Moreover, it would have been a very high logistic effort to organize two fully functional teams in the MIDDLE SIZE LEAGUE for a 'real' game.
In our analysis we will mainly consider set pieces. Set pieces are situations where the game is paused, that is the case for corner kicks, free kicks, goal-kicks, throw-ins, and for penalties. The game continues with the ball being at a fixed position. We do that since set pieces are more static. That provides us with more well-defined situations. Hence, these situations are easier to investigate and to present.

7.3.2 BuildUpPlay

We already stated that we have a slightly more static and well-defined situation when we are dealing with set pieces. Following the different phases in the course of a soccer game we firstly consider a situation in which a build up play is to be made.
One of those situations in terms of set pieces is the goal-kick for our own team. Normally, the player having the role *defender* is the one to execute the goal-kick. The remaining players, namely the *supporter* and the *attacker*, are teammates that are able to receive the ball. In Chapter 5 we explored three possible ways for building up a play. We adapted them for the ROBOCUP domain. A diagram depicting the different possibilities to build up a play in the set piece of a goal-kick for the own team is shown in Figure 7.3.

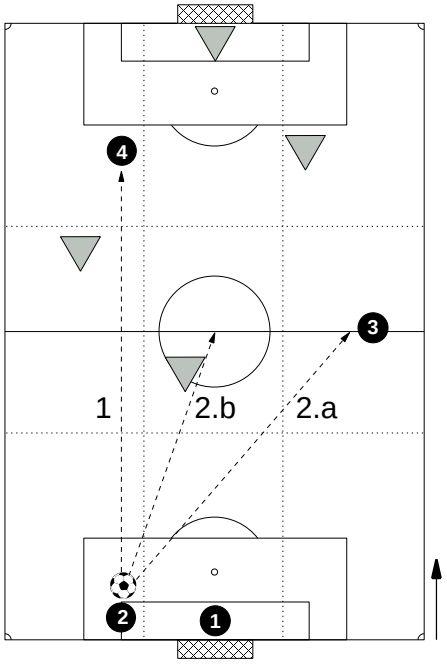


Figure 7.3: Diagram adapted for MIDDLE SIZE LEAGUE

The diagram contains the set of actions that can be performed by the player executing the goal-kick. They are indicated by different arrows. We included all three possible ways to build up a play mentioned in [Luc01]. That is either a long pass to the front part of the field (dashed arrow labeled with 1), a diagonal pass to the opposite site of the current ball position (dashed arrow labeled with $2.a$), or a deep pass to the center of the pitch (dashed arrow labeled with $2.b$).
With the enhancements in terms of qualitative notions which we achieved in this thesis we are now able to transfer this schematic information to an agent specification with only little effort. Let us examine the corresponding specification in READYLOG in order to show how the qualitative enhancements we aimed at can be used in our framework. Figure 7.4 shows the program code which is executed by the player having the role *defender* for the own goal kick situation.
The procedure `proc_BuildUpPlay` is called when our own team has a goal-kick. That is, if the ball went out of bounds over the goal line and an opposing player was the last one touching the ball.
In the first part of the procedure the agent retrieves values for some variables and assigns them within the test statement $?(\cdot)$. In terms of qualitative representations, it retrieves and assigns the current *side* of the ball to the variable `SideBall`. Then the agent having the role `Defender` is instructed to intercept the ball.[1] It is told to do this facing the orientation `oriFront`.
Subsequently, the agent starts deliberating with the solve statement in line 13. The player being `Supporter` is told to go to the global position denoted by `[zone(zoneMiddle),side(-SideBall)]`. That is to say, it should go to the opposite side of the ball (`-SideBall`) in the middle of the playing field. The agent having the role `Attacker` is instructed to go to the front part of the pitch on the same side the ball is at.
Finally, in line 24 the agent's program uses the $\mathrm{pickBest}$ statement to choose the best argument for a particular action. That is to say, the agent decides to call the action `pass_to` for the agent being `Defender` with the best value for variable `var_target` taken from the list of arguments specified in lines 24 to 29. This list of positions corresponds to the set of possibilities to build up a play we depicted in Figure 7.3.
The procedure `proc_BuildUpPlay` is, of course, only a simple example. But, we think it illustrates that with the resources provided by the work of this thesis we are now able to specify an agent's behavior with greater ease than before. That is not only because we can transfer expertise with less effort but also due to the more natural character of the predicates provided. Moreover, the abstractions being made by the qualitative notions yield to a generalization which renders it possible to disregard diverse subtleties an agent's designer is normally confronted with.

As we already said in Section 4.4 the agent's decision is substantially determined by the reward function. As you can see in line 33 we are using the function `f_Reward_BuildUpPlay` as the reward function for our `proc_BuildUpPlay` program. We list it in Figure 7.5.
The reward function retrieves information about the situation and evaluates it. In our example, this information contains the positions of the defending player, the position of the ball, and the target of the action $\mathrm{pass\_to}$. Please note, that the reward function evaluates a projected situation; that is to say the fluent values were already modified according to the models of the actions planned. Hence, the `TargetPos` contains the position to which the defender wants to

[1] Please note that the intercept action has the suffix `_nonblock` which denotes that the agent does not wait for the intercept to get finished but instead proceeds with its program.

```
proc( proc_BuildUpPlay,
     [% gather information
      ?( and([  ballPos = [BallX,BallY],
                rw_side(BallY,SideBall),
                %writeln([SideBall]),
                Attacker      = numberByRole(attacker),
                Supporter     = numberByRole(supporter),
                Defender      = numberByRole(defender)
             ])
      ),
      intercept_ball_nonblock(Defender, ori(oriFront),
                              drivemodeSlowAllowBackward),
      solve( [ [ % SUPPORTER %%%%%
                 goto_global(Supporter,
                             [zone(zoneMiddle),
                              side(-SideBall)],
                             drivemodeSlowAllowBackward),
                 % ATTACKER %%%%%
                 goto_global(Attacker,
                             [zone(zoneFront),
                              side(SideBall)],
                             drivemodeModerateForward)
               ],
               pickBest(var_target, [ [zone(zoneFront),
                                       side(SideBall)],
                                      [zone(zoneMiddle),
                                       side(-SideBall)],
                                      [zone(zoneMiddle),
                                       side(sideMiddle)] ],
                        % DEFENDER %%%%%
                        pass_to(Defender, var_target, 3)
                       ) %% end pickBest
             ], 4,  f_Reward_BuildUpPlay), %% end solve
      waitForNextCycle
     ]
    ).
```

Figure 7.4: A possible program to build up a play performed by a defensive player.

```
function( f_Reward_BuildUpPlay, V,
          and([ DefenderNumber = numberByRole(defender),
                DefenderPos = ecf_agentPos(DefenderNumber),
                DefenderPos = [X,Y],
                TargetPos   = ballPos,
                globalPasswayVacancy( DefenderPos,
                                      TargetPos,
                                      PasswayVacancy ),
                PassDist    = func_distance( DefenderPos,
                                             TargetPos ),
                VacyReward  = (PassDist * PasswayVacancy),
                V = VacyReward
              ])
        ).
```

Figure 7.5: The reward function used for proc_BuildUpPlay.

pass (if possible) and the `BallPos` is the projected position of the ball according to whether or not the pass succeeded.
Again, the reward function presented here is only a simplistic example. Nevertheless, it reflects the benefits of our qualitative enhancements quite well. That is because we think it gets obvious how straightforward the evaluation of a situation can be with the aid of qualitative predicates. In our case that is the predicate `projectedPasswayVacancy` which classifies the vacancy of the pass way used to build up a play. There is no longer any need to worry about geometrical considerations. Instead, we can simply combine the values of properties we think to be relevant for the evaluation.

Let us investigate the execution of the program `proc_BuildUpPlay` in Figure 7.4, now. Therefore, we consider an example situation in which to build up a play in Figure 7.6. We constructed the situation in the simulation environment. It is, however, a situation that occurs just the same in a real game between to teams in the Middle-Size.
There are eight robots on the field, four of each team. All ALLEMANIACS' robots are executing the procedure `proc_BuildUpPlay` which was invoked upon the playmode for a goal-kick for the own team.
When decision-theoretic planning is started with the solve statement READYLOG evaluates all possible courses of action specified within. That is, the Attacker (Kirk) is going to position [zoneFront, sideBall] and the Supporter (Cook) is going to [zoneMiddle, −sideBall]. Then, the Defender (Hook) can perform a pass to one of the positions specified in the list in the pickBest statement, namely to the position of the attacker, to the position of the defender, or to the position denoted with [zoneMiddle, sideMiddle].
The reward function computes an estimate rating of each resulting situation. This is done by retrieving the qualitative position of the ball as well as the value of the PasswayVacancy predicate. The straightforwardness illustrates one of the fundamental achievements we aimed at. We are able to use qualitative characteristics of a situation instead of being forced to perform geometric calculations by ourselves. This allows us to transfer our abstract criteria on how good a situation is very easily.

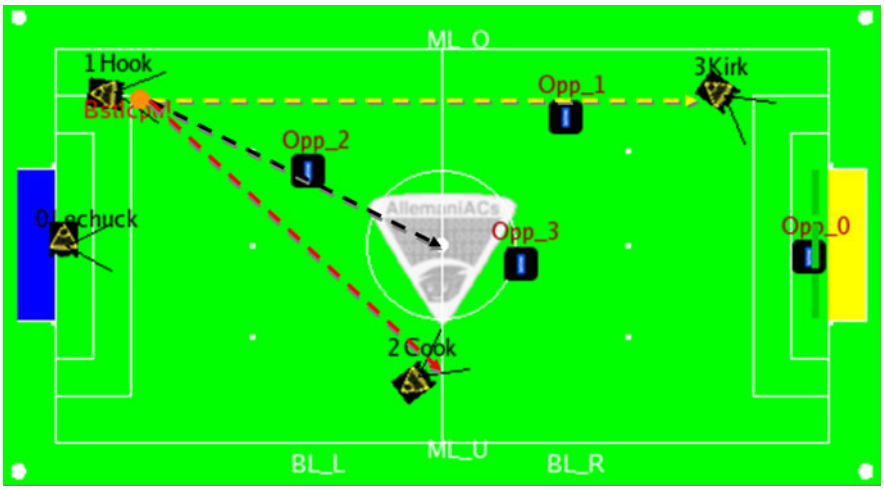


Figure 7.6: Example situation in which to build up a play in the MIDDLE SIZE LEAGUE

Let us briefly step through the evaluation process of our example situation. After projecting the two goto actions in the beginning of the solve statement we evaluate three pickBest possibilities. In our reward function we can retrieve the position that is currently evaluated via the PassTarget fluent. We call the function `projectedPasswayVacancy` to retrieve the vacancy of the pass way to the current pass target. A higher value indicates lesser endangerment for the pass. Furthermore, we retrieve the zone of the ball, that is its qualitatively abstracted position with respect to the length of the field. The higher this value is the more the ball is in the front part of the pitch. We compute the overall reward for a situation as the sum of BallZone and PasswayVacancy. That is to say, the lesser the endangerment for the pass was and the farther the ball gets towards the opponent's goal the better the situation is rated. In our example the highest reward is computed for the pass to the supporter's position (dashed red line). That is because this pass leads to the highest PasswayVacancy even though the ball would be more in the front part of the pitch when passing to the Attacker (dashed yellow line). We depicted a trace of the READYLOG interpreter in Figure 7.7.

7.3.3 Refinements

There are still multiple ways to refine the specification of the BuildUpPlay program discussed in the previous section. One way to do this is to integrate more qualitative predicates with very little effort. As a matter of fact it is actually reasonable to consider more properties of the situation in which to decide what to do.
Additional information that is meaningful to the decision is concerned with the conditions associated with the player receiving the defender's pass. That is, for instance, the information whether or not the receiver is marked by an opponent. We provide a qualiative predicate containing this information, that is the unmarked predicate. We have distinct possibilities to integrate the predicate into the specification. The first way is to include it in the reward function. We choose this possibility if we only want to uprate those situations where the receiving player is unmarked. If we, instead, want to exclude that the defender passes to a teammate which is marked we would change the program by adding an `if` statement. That is to say, when applying the policy generated in planning the defender only executes the pass if the receiving player is unmarked.

```
 --------        solve  -------------
Horizon: 4
Prog: [[goto_global(2, [zone(zoneMiddle), side(-(-1))],
                    drivemodeSlowAllowBackward),
        goto_global(3, [zone(zoneFront), side(-1)],
                    drivemodeModerateForward)],
        pickBest(var_target,[[zone(zoneFront),side(-1)],
                             [zone(zoneMiddle),side(-(-1))],
                             [zone(zoneMiddle),side(sideMiddle)]],
                 pass_to(1, var_target, 3))]
S: [send(nextSkill(1),
    intercept_ball(ori(oriFront),drivemodeSlowAllowBackward)),
    exogf_Update, s0]
(...)
 --------        solve DONE     -------------
Policy:
[ goto_global(2, [zone(zoneMiddle), side(-(-1))],
              drivemodeSlowAllowBackward),
  [ if(true) then
        ~~ [ goto_global(3, [zone(zoneFront), side(-1)],
        ~                drivemodeModerateForward),
        ~ [ if(true) then
        ~       ~~ [ pass_to(1, [0.0, 1.75], 3),
        ~       ~    [ exogf_Update,
        ~       ~      [ if(near(ballPos,
        ~       ~        ~        [5.175, -0.781321052759893],
        ~       ~        ~        0.5)=true) then
        ~       ~        ~~ []
        ~       ~        ~~ [ (..), [] ], [] ] ] ]
        ~       ~~ [], [] ] ]
        ~~ [], [] ] ]

Times: [0.0599999999999998, 0.0, 0.0599999999999987]
Value: 2.89806668071002
TermProb: 1.0
```

Figure 7.7: Trace of the READYLOG interpreter for the proc_BuildUpPlay program.

Another possible refinement is to make the program more flexible. Instead of instructing the attacking player to go to a fixed position like $[\mathrm{zoneFront}, \mathrm{sideBall}]$ we can also retrieve a free position in the front part of the pitch. READYWORLD would process such a query with the Voronoi-based methods presented in Chapter 6.

Lastly, a potential variant to refine our agent specification is to take the further course of the game into account. This can, for example, be done by evaluating the chance for the receiving player to shoot at the goal. We could account for this within the reward function by retrieving the $\mathrm{PasswayVacancy}$ from the receiving player to the goal.

7.3.4 KickTrick

Another set piece situation of particular interest is the free kick. In the MIDDLE SIZE LEAGUE the free kick is accounted for fouls just as in human soccer. Thus, it has to be executed in various situations.

We developed an unorthodox procedure to execute a free kick with our Middle-Size robots. We call this procedure $\mathrm{KickTrick}$. It makes use of the fact that our robots have a rectangular shape and that they are build very robust. The idea is as follows: The $\mathrm{Attacker}$ intercepts the ball at the position where the free kick is to be executed. Meanwhile, the $\mathrm{Supporter}$ chooses a free position nearby. Then the $\mathrm{Attacker}$ shoots in direction of the $\mathrm{Supporter}$. The $\mathrm{Supporter}$ positions itself in such a way that it can reflect the ball towards the opponent's goal.

The procedure described above demands for the abstract notions we developed in our work. That is because the positions of the teammates working together cannot be determined by a fixed geometrical scheme. Let us consider the READYLOG procedure `proc_FreeKick` in order to illustrate this. We depict it in Figure 7.8.

In the beginning of the procedure we retrieve the qualitatively abstracted position of the ball. Upon these information we choose potential positions for the reflecting player to go to. We take four positions into account. They are constructed as follows: First, we take the current zone of the ball and the successing zone, that is the next zone being closer to the opponent's goal. Second, we consider the side at which the ball is located and we determine the residual sides. Then we combine these zones and sides and store the resulting positions in a list called `PickList`. Subsequently, we call the $\mathrm{pickBest}$ statement in order to compute the 'best' positions to execute the $\mathrm{KickTrick}$ procedure at.

Let us now consider the READYLOG procedure `proc_KickTrick` which we list in Figure 7.9. The procedure `proc_KickTrick` specifies the actions to be performed to execute the $\mathrm{KickTrick}$. That is, we first determine the angle in which to intercept the ball and the angle in which to reflect it. Then the two players, namely the $\mathrm{Attacker}$ and the $\mathrm{Supporter}$ move according to our idea for the $\mathrm{KickTrick}$.

For all procedures which incorporate decision-theoretic planning using the solve statement we need to specify a reward function. For the KickTrick procedure we use the function `f_Reward_KickTrick` which we list in Figure 7.10.

The reward function classifies the resulting situation by utilizing several qualitative predicates. That is to say, the situation rating is based on the $\mathrm{PasswayVacancy}$ of the pass ways between the $\mathrm{Attacker}$ and the $\mathrm{Supporter}$ as well as between the $\mathrm{Supporter}$ and the goal.

```
proc( proc_FreeKick,
   [ ?( and([ globalBallSide(SideBall),
              globalBallZone(ZoneBall),
              FwdZoneBall = Succ(ZoneBall),
              lif( SideBall = sideLeft,
                   [ SideOne = sideMiddle,
                     SideTwo = sideRight],
                   lif( SideBall = sideRight,
                        [ SideOne = sideMiddle,
                          SideTwo = sideLeft],
                        lif( SideBall = sideMiddle,
                             [ SideOne = sideRight,
                               SideTwo = sideLeft]) ) ),
               PickList = [[zone(ZoneBall),side(SideOne)],
                           [zone(ZoneBall),side(SideTwo)],
                           [zone(FwdZoneBall),side(SideOne)],
                           [zone(FwdZoneBall),side(SideTwo)]] ])
      ),
     solve([ pickBest( var_target, PickList,
                       proc_KickTrick(var_target) )
           ], 7,  f_Reward_KickTrick)
   ]
 ). %% end proc_Own_FreeKick
```

Figure 7.8: The READYLOG specification for the proc_FreeKick procedure.

```
proc( proc_KickTrick(ReflectPos),
   [ ?( and([  ReflectPos   = [ReflectX,ReflectY],
               ReflectOri   = f_Ori_Reflect(ballPos,
                                            ReflectPos,
                                            bestTarget),
               ReflectPose  = [ReflectX,ReflectY,ReflectOri],
               InterceptOri = f_Ori_BallToPos(ReflectPos),
               Attacker     = numberByRole(attacker),
               Supporter    = numberByRole(supporter) ])
      ),
      % SUPPORTER %%%%%
      goto_global( Supporter, ReflectPose,
                   drivemodeModerateAllowBackward),
      % ATTACKER %%%%%
      intercept_ball( Attacker, InterceptOri,
                      drivemodeModerateForward),
      waitForPlaymode( playmodePlayOn ),
      kick( Attacker, 100)
   ]
 ).
```

Figure 7.9: The READYLOG specification for the proc_KickTrick procedure.

```
function( f_Reward_KickTrick, V,
      and([ KickerPos    = ballPos,
            ReflectorNum = numberByRole(supporter),
            ReflectorPos = ecf_agentPos(ReflectorNum),
            TargetPos    = bestTarget,
            ReflectAngle = f_Angle_ReflectFromAtTo(KickerPos,
                                                   ReflectorPos,
                                                   TargetPos),
            AngleReward  = 1.57 - abs(1.047-ReflectAngle),
            globalPasswayVacancy( KickerPos, ReflectorPos,
                                  PasswayVacancyKick ),
            DistKick  = func_distance( KickerPos, ReflectorPos ),
            globalPasswayVacancy( ReflectorPos, TargetPos,
                                  PasswayVacancyTrick ),
            DistTrick = func_distance( ReflectorPos, TargetPos ),
            VacancyReward = (DistKick * PasswayVacancyKick) +
                            (DistTrick * PasswayVacancyTrick),
            geom_angle( KickerPos, ReflectorPos, KickAngle ),
            geom_angle( KickerPos, f_Pos_OwnGoal, GoalAngle ),
            GoalReward = abs( KickAngle - GoalAngle ),
            V = VacancyReward + (10 * AngleReward) + GoalReward ])
 ).
```

Figure 7.10: The READYLOG specification for the function f_Reward_KickTrick.

7.4 Soccer Simulation League

We mentioned that we require the module READYWORLD to be as flexible and platform independent as possible. To verify our achievement in matters of platform independence we applied READYWORLD in our SOCCER SIMULATION LEAGUE framework. Therefore, we implemented a platform specific connection to the league's low level to retrieve data available in this league. We also integrated the KNOWLEDGE LIBRARY into the existing framework so that we can process requests in their textual form.
The essential observation is that it actually works. We are not going into detail here. Instead, we only mention that - upon our work - there is currently a new agent being developed. Moreover, our framework is employed for action selection. The qualitative abstractions which we provide are used to simplify the decision on which action to take.

Chapter 8

Implementation

Within the work of this thesis the software module READYWORLD has been developed. READYWORLD is a library that provides a qualitative world model including basic inference mechanisms. It works on the low-level data provided by the particular agent platform it is used for. As we intend to use READYWORLD in both the SOCCER SIMULATION LEAGUE and the MIDDLE SIZE LEAGUE, generality has been an important aspect within the development. READYWORLD is designed to be as league and platform independent as possible. This allows us to apply it to any other league more easily by just replacing the connection to the low-level layer where the numerical values of an agent's perception of the world are retrieved.

8.1 Framework

The research topic of this thesis' framework is intelligent decision making of autonomous agents. For us, this means that an agent should be able to make plans for future courses of actions instead of only deciding what to do based on the current situation. For this purpose we use the language READYLOG which we already described in Chapter 4. Planning is generally costly. Moreover, in agent soccer it is better to make a sub-optimal decision on which action to take than to make no decision. Therefore, we believe that some form of reactive control is still needed in order to keep reactivity. Hence, the aim is to combine planning with reactive action selection to get the best performance for the high-level control of the agent. Before we discuss the implementation of this approach in detail we give an overview of our current decision module architecture.

8.1.1 The DR-ARCHITECTURE

While deliberation has many advantages for decision making of an autonomous agent, it has the disadvantage of being rather slow compared to systems which generate actions purely reactive. In [DFL02] a hybrid architecture was proposed which allows for the combination of deliberation with reactivity. We give a brief overview of this architecture here. For more information we refer to [DFL02]. Figure 8.1 shows the DR-ARCHITECTURE.
From the sensory input of an agent we build a quantitative world model. All decision making components work on the information contained in this world model. The decision module in the DR-ARCHITECTURE is divided into three sub-modules.
To be able to settle an action immediately the *Reactive Component* computes the next action to be executed based on the current game situation. As proposed in [KFL04] the reactive action

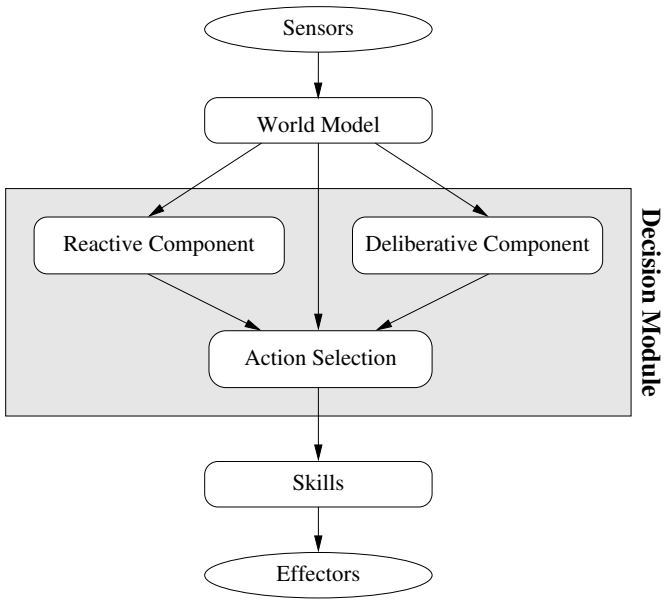


Figure 8.1: The ALLEMANIACS DR-ARCHITECTURE

choice is formulated as a classification problem. We apply Quinlan's C4.5 [Qui93] to learn the assignment from world situations to actions.
In contrast, the *Deliberative Component* calculates a plan by projecting into the future and choosing among the possible action sequences. The choice for the best action sequence is supported by an optimization theory. We apply decision-theoretic planning to calculate an optimal policy in the READYLOG framework.
Because we have an immediate action and a plan concurring for execution one needs to decide which of both to use. Therefore, an *Action Selection* module is currently being developed. The idea is to have a flexible selection mechanism which adapts to the environment during a match. The approach is to use reinforcement learning methods to learn the best strategy for the action selection.
The *skill* chosen for execution is then passed on to the execution system. *Skills* are the basic actions that the agent can perform in the world. In the MIDDLE SIZE LEAGUE they are defined in the *skill module* which we are going to consider in Section 8.1.3. In the SOCCER SIMULATION LEAGUE they are provided by a basic agent framework developed by Kok and de Boer [dBK02].
The DR-ARCHITECTURE as we sketched it above is used in both the SOCCER SIMULATION LEAGUE and the MIDDLE SIZE LEAGUE. Only the connection to the sensory system and the connection to the execution system are different. Furthermore, the set of basic abilities varies slightly between the two leagues. In the following, we present the software system of the MIDDLE SIZE LEAGUE as well as of the SOCCER SIMULATION LEAGUE in some more detail.

8.1.2 SOCCER SIMULATION LEAGUE

We now briefly outline the software system which the ALLEMANIACS use in the SOCCER SIMULATION LEAGUE.
The soccer game is simulated by the so-called SoccerServer [NMHF97]. Each participating

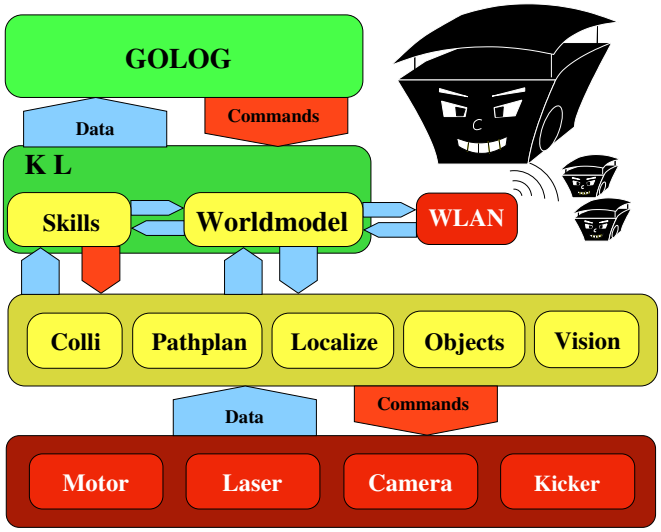


Figure 8.2: ALLEMANIACS MIDDLE SIZE LEAGUE software system

agent connects to this server via a network connection. The server simulates the game and provides a model of the physical environment. This model consists of auditive and visual information for each agent as well as information on its body such as the agent's current strength. To achieve a more realistic model of an agent's perception uncertainty is added to the provided information by artificially affecting them with noise. The simulated data are transmitted to each player in discrete time intervals called cycles. An agent can send action commands such as *dash*, *turn*, or *kick* to the server where they are executed if possible. The ALLEMANIACS' client uses the world model and the basic playing skills of the UvA-Trilearn agent developed by Jelle Kok and Remco de Boer [dBK02]. As already stated before the ALLEMANIACS feature a hybrid architecture which contains a reactive component. This reactive component uses C4.5, a decision tree system described in [Qui93].

8.1.3 MIDDLE SIZE LEAGUE

Now we want to give a brief overview of the RCSOFT. This is the software system which we use to run the mobile robots of the MIDDLE SIZE LEAGUE. We distinguish between software running locally on each robot and control software running externally on an additional computer.

Local Software

The local software system uses a three-layered architecture. It consists of a low-level layer where sensory inputs and actuator outputs are controlled, a mid-level where modules like collision avoidance and localization are located and a high-level module for making plans about future courses of actions using READYLOG. The architecture is depicted in Figure 8.2.
For the low-level control task there exist modules for motor control, for receiving the data from the laser scanner, for supplying video streams from the cameras, and for controlling the kicking device. On the mid-level data obtained on the low-level is used for robot control. The software features a module for self-localization which uses the 360° laser range finder following the Monte Carlo approach. We are able to localize with high accuracy in the ROBOCUP envi-

ronment integrating a whole 2D-sweep from the laser range finder. Another module working on the data from the laser range finder performs object recognition. It provides information on dynamic objects on the field. There exists a central module for collision avoidance which also uses the data provided by the laser scanner. An A^* search is used to calculate a collision free path to a given target point. The ball, the goals and the flag-posts on the playing field are detected by two vision modules using color segmentation and knowledge about the objects' shape.

The *world model* consists of information like ball and player positions as well as some other internal state information such as the current play mode. We provide two kinds of world model, a local one which is constructed from the own perception and a global one which is the result of several fusion processes running on an external computer.

The *skill module* encapsulates the robot's behavior system. All basic behaviors like kicking and dribbling are implemented here. This module has access to modules controlling those actuators that are needed to perform soccer behaviors. It also has access to all world model information relevant for its tasks.

Finally, there is the so-called knowledge library (KL). It integrates the complete world model. The basic idea is to have one central component that provides all relevant information for high-level decision making. It processes queries like $\mathrm{position}(\mathrm{teammate}, \mathrm{region}(\mathrm{front}, \mathrm{left}))$ and returns the corresponding value. The skill module is implemented as an instance of the KL. On the skill level this allows to use all the predicates provided by the world model. By this, moreover, we achieve a form of a *least commitment* execution system. The skill module is able to resolve predicates like $\mathrm{kick}(\mathrm{bestGoalCorner})$ by itself. This is especially important when a time gap between plan generation in READYLOG and plan execution in the skill module exists.

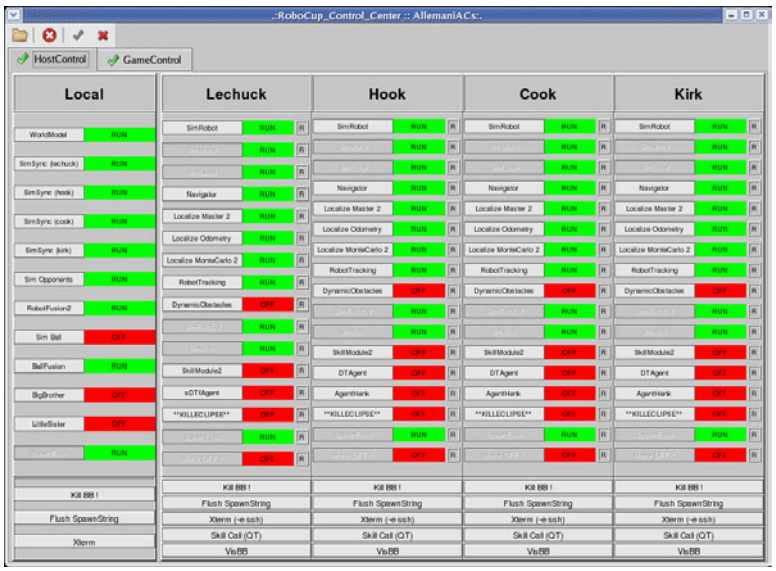
For interprocess communication we use a blackboard communication which internally uses a shared memory. Instead of communicating directly in a blackboard architecture all modules place the results of their particular task in a central area (the blackboard) and fetch new input from there. This allows every module to see and access all available data. Furthermore, this renders it possible to easily exchange different modules carrying out the same task. For communication between different computers the blackboard uses UDP.

External Control Software

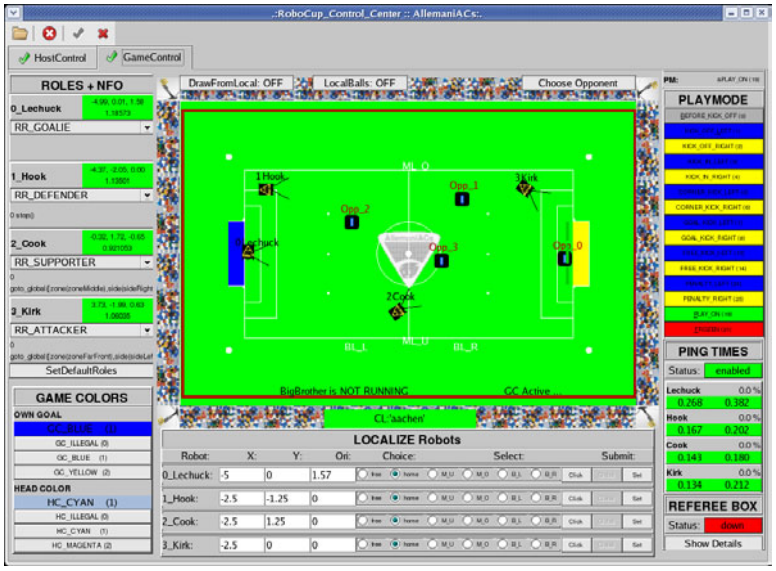
As already mentioned in the previous section, there are some modules running on an external computer for controlling the robot team and calculating parts of the global world model.

One of these external modules is the RoboCup Control Center (RCCC). This module is used to control the team and it is therefore connected to the robots via wireless LAN. With this module we are able to control the software modules running locally as well as those running on each robot. Furthermore, we are able to retrieve and set information on the current game state. That is, for instance, the current playmode or the colors of the own and the opponent's goal. Every module on each robot can be started or stopped. Moreover, we get information on the status of a particular module, e.g. if it is running or if it crashed for some reason. To indicate the current game state to the robots we can choose between different play modes like *play on* or *kick off left*. For getting information on the robot's internal state we visualize the robot's pose estimate and the opponent's pose estimates as well as the action a robot is currently performing. Figure 8.3 shows a screen shot of the RCCC.

Moreover, we provide modules for fusing the robots' perceptions of the ball and of tracked



(a) Software Control Window



(b) Game State Window

Figure 8.3: The RoboCup Control Center. Figure 8.3(a) shows the software control window. The graphical user interface is configurable via an XML file enabling the integration of new modules into the software control preferably easily. Figure 8.3(b) shows the game state window.

opponent players in order to enhance our global world model.

RCSOFT Simulator

Since maintaining the hardware platform of a multi robot team is sometimes difficult it is reasonable to provide a simulation environment. This allows us to test newly developed software components even if the robots are not available at that time.
With its blackboard communication the software architecture described above allows for a creation of a simple simulation system quite easily. Since all modules communicate only through previously defined interfaces each module can be replaced by a module simulating its purpose. Hence, in order to build a simulator the modules on the lowest level which normally control the hardware platform have to be replaced by modules that simulate a physical environment.

SimBall

In the context of this thesis we developed parts of the simulation framework. That is, firstly, we provided a module which simulates a ball. On the central station the user specifies the initial position and the initial movement of the ball. This information is then passed on to the robots. Within the robot simulator there is a ball simulation process with inserts this virtual ball position into the robot's local world model. Physical interaction between the robots and the simulated ball is computed on the central computer. The new position and movement information gets redistributed back to the robots again.

SimOpponents

Furthermore, we developed an opponent simulator. The user has the possibility to specify the positions of abritrarily many opponent players on the playing field. This is done by generating simulated reflections within the laser readings of each of our robots.
For this purpose, on the central computer a list of objects to be simulated is maintained. This list is then distributed to all (simulated) robots. Within the robot simulator these objects are then integrated into the simulated laser readings. This way, all software modules except for the hardware modules remain the same as in a real setup.

8.2 READYWORLD

We are now going to present implementational details of the module READYWORLD which we developed in the work of this thesis.

8.2.1 Software Design

READYWORLD is designed module-based in order to achieve an easier exchangeability of single methods and mechanisms. This also makes it less difficult to compare different modules for a specific task. We depict the overall structure of READYWORLD in Figure 8.4.
Each module is designed preferably flexible by means of parametrization. We realize the parametrization by a configuration object which stores and provides all configurational data. This object retrieves all necessary parameters from an XML-file. When starting READYWORLD it is possible to specify an configuration file via a command line argument. This

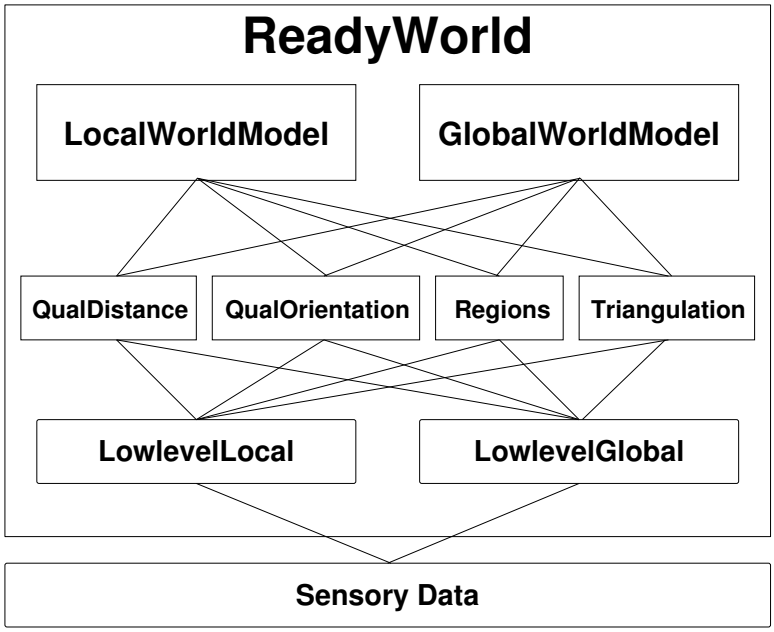


Figure 8.4: Software architecture of the READYWORLD module

allows us to easily use READYWORLD with different sets of parameter values. The configuration object is passed to all modules which need to access parameter values. The parameter set includes values such as the level of granularity of the distance and the orientation relation, the number of zones, and the number of sides.

On the lowest level in the READYWORLD architecture there is an abstraction layer consisting of two classes. These classes are used to retrieve the sensory information of an agent. That is, the two classes `lowlevel_local` and `lowlevel_global` encapsulate the access to the information provided by the agent's base system. As their names already suggest, there is one class for local information and a second class for global information which can, for example, be supplied by internal fusion processes. The low-level classes then make these information available via a standardized interface consisting of all data potentially useful to READYWORLD. Data that are not provided by the particular platform are simply marked as not being available. We depict an UML diagram of this abstraction layer in Figure 8.5.

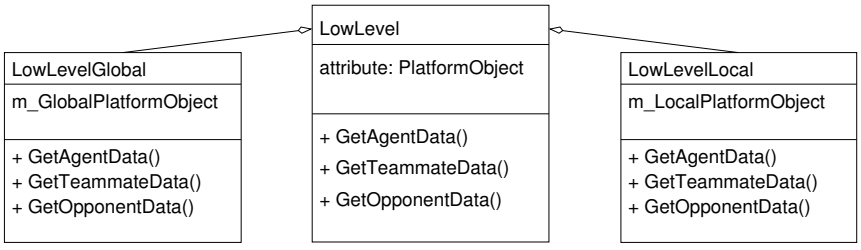


Figure 8.5: UML diagram of the lowest level in the READYWORLD architecture.

On the middle level there are two separate classes for a standardized local world model and a standardized global world model. They retrieve all data from the corresponding low-level classes described above. It is on this level where the first qualitative predicates are computed. That is, each world model class contains three triangulation modules which construct a Delau-

nay triangulation and a Voronoi diagram for all teammates, all opponents, and all players each. A UML diagram of the world model level is depicted in Figure 8.6.

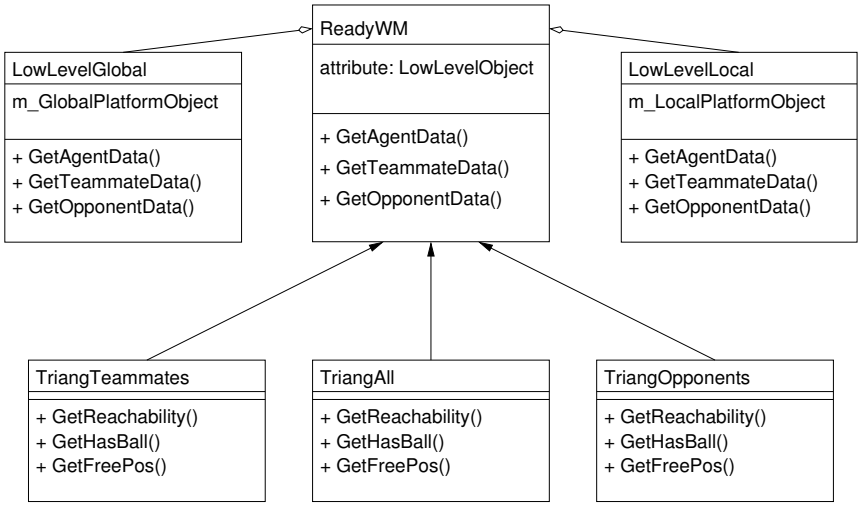


Figure 8.6: UML diagram of the world model level in the READYWORLD architecture.

Each world model stores its available data in a unified structure derived from the world model class hierarchy that we acquired in Section 5.2. This class hierarchy is reflected in the inheritance structure of the object classes we provide. There is a base class `object` which all objects are derived from. Every object has a position associated to it. It can either be static or dynamic. We got separate classes for each object in the ROBOCUP domain each derived from the corresponding super class. Each sub-class has additional information assigned to it according to its properties.
Lastly, there is the class `ReadyWorld` which constitutes the topmost level within the architecture. This top-level class provides all functions available to the outside. That is, it holds both the local and the global world model classes as an instance and it encapsulates all function calls to them. Additionally, there are different sub-modules for qualitative computation. These sub-modules include a class for qualitative orientation, a class for qualitative distance, and two classes for the semantic regions on the playing field, namely a class for zones and a class for sides. In Figure 8.7 we show a UML diagram of the top-level layer.

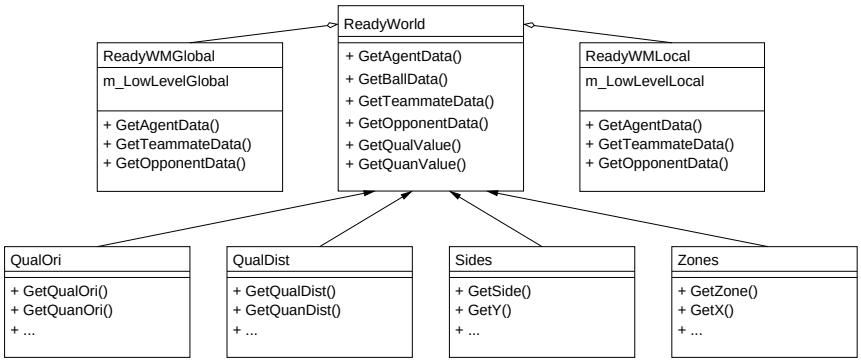


Figure 8.7: UML diagram of the topmost level in the READYWORLD architecture.

It is indispensable for autonomous agents that the sensory data it works on are consistent with each other. To be able to guarantee this consistency the top-level object provides a synchronized

`Update` method. When this update request is called READYWORLD launches an incremental update procedure. It starts in the lowest level by retrieving the latest data from the platform specific objects. These data are subsequently conformed to the internal structure. Afterwards, the world model level classes retrieve the standardized information to update their internal structure and they also recompute the values of all those predicates depending on multiple data sets.

8.2.2 Graphical User Interface

In order to visualize the data READYWORLD is working with, we implemented a graphical user interface. Almost all processed information and all generated information are displayed herein. There are separate parts for local and for global data as well as for general information such as the size of the playing field. Moreover, game overall predicates like the team's ball possession, the gameSetting, and the gameEdge are visualized.
Both, the local and the global part show the pitch and all objects on it. The user may activate several drawing routines which allow for displaying the regions on the pitch, the Voronoi diagram and the Delaunay triangulation. The latter two can be displayed for all teammates, all opponents, and for all players each. Furthermore, there is a table indicating the 'aliveness' of each teammate. It also contains information about the reachability between each two teammates as well as their qualitative distance to each other. In Figure 8.8 we depict an exemplary screenshot of the graphical user interface as we sketched it above.

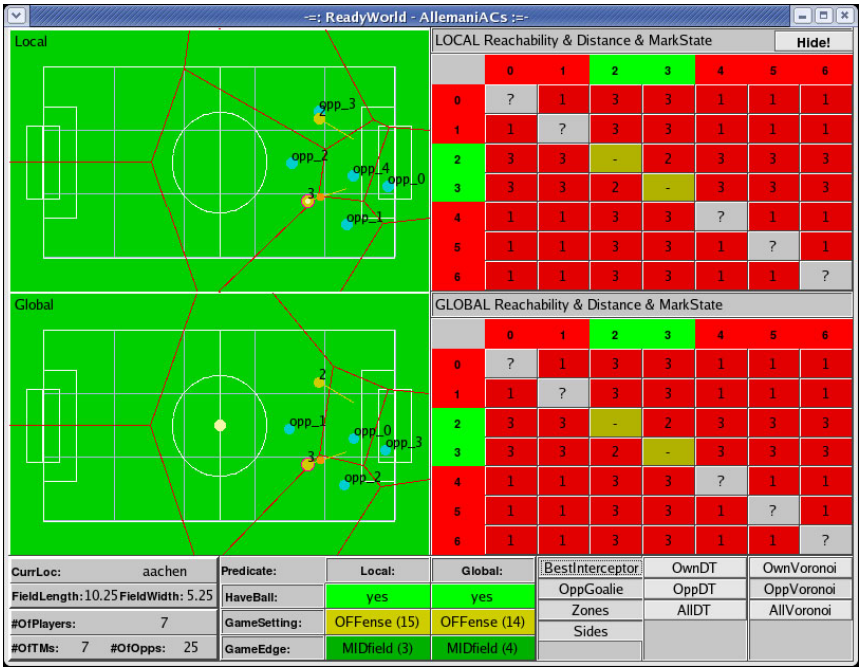


Figure 8.8: The graphical user interface provided by READYWORLD

READYWORLD's GUI was initially designed to be used for debugging purposes. But, it can also help an agent's designer in several ways. First, the designer can figure out which information are available to a single robot as well as to the whole team. Secondly, he can determine which qualitative information are assigned to a particular situation which should enable him to develop an agent's program more easily in terms of a qualitative description of situations and

settings. The GUI can be switched on and off during runtime. Thus, the computational efforts spent on its display can be saved when it is no longer needed.

8.3 Integration

READYWORLD is compiled as a shared library. This enables every other module in the overall software system architecture to embed it preferably easily.

Low-level

We already mentioned that the connection to the low-level system is done in separate classes. This enables a straigtforward application of READYWORLD to different platforms. In order to connect a particular platform to the READYWORLD framework the only parts that have to be modified are the two low-level classes, namely `lowlevel_local` and `lowlevel_global`. There is an predefined interface which has to be met. Information that cannot be provided by the platform can be set to void values. The low-level specify an update method which is meant to be used to fetch the latest system information. This is required to allow for consistent data. The world model level then retrieves actual data via the predefined data access methods.
In order to make the information provided by READYWORLD available for our high-level component we have to establish a connection to READYLOG. This is done through the KNOWLEDGE LIBRARY which we now describe in some more detail.

KNOWLEDGE LIBRARY

For the integration of READYWORLD with our high-level component we use the so-called KNOWLEDGE LIBRARY (KL). The KL is realized as a library and provides a query server for diverse kinds of information. In particular, it can process symbolic queries like $\mathrm{bestTarget}$ or $\mathrm{zone}(\mathrm{zoneLeft})$. A query is mapped to a function or to a quantitative values. The corresponding value then is returned to the inquiring module. The queries can also be nested. The KL parses the query string and decomposes it if necessary.
To keep the areas of application as broad as possible the KL comes with an abstract type concept. This allows us, for instance, to use the KL within the execution system, too. Hence, we can use abstract symbolic notions also for the execution system. Furthermore, we use it to establish *least commitment* facilities in our framework which is beneficial for queries like $\mathrm{bestGoalCorner}$. This is especially important when there exists a time gap between plan generation in READYLOG and plan execution in the execution system. That is because the query is processed at very last possible point in time. The KNOWLEDGE LIBRARY integrates the complete world model. Thus, we have one central component which provides all relevant information for high-level decision making as well as for the execution system.

Chapter 9

Conclusion and Future Work

In this chapter we conclude the thesis. We summarize our contributions and recapitulate the benefits we achieved. Afterwards, we propose topics for potential future work.

9.1 Conclusion

We have designed and implemented READYWORLD, a qualitative world model for autonomous soccer agents. It enhances the world model used so far with qualitative predicates. Moreover, it allows for representing and describing spatial knowledge in an abstract and unified form. With our work we increased the comfort in agent specification. That is because we based our qualitative representations and predicates on cognitive considerations.
Modeling intelligent behavior is a complex tasks for designers of autonomous agents. Therefore, we posed the question of how we can make use of expertise in the task at hand. With our work we aimed at narrowing the gap between the numerical character of current agent systems and the abstract and general form of the knowledge a designer of an agent usually has. Our work now allows for a more natural and abstract way to specify an agent's behavior; as an example domain we presented the robotic soccer domain.
The fundamental idea behind our approach was to enable an agent's designer to transfer his expertise in the domain at hand to the specification of an agent more easily. The domain of application in our work is the robotic soccer domain. Therefore, we investigated human soccer theory. Our analysis revealed that we need to provide qualitative predicates and notions in order to make use of tactical insights developed for human soccer. We analyzed books on human soccer theory and also took a look at an analysis of newspaper reports on human soccer games. First, we derived an ontology for the soccer domain and identified the organizational structure of a soccer game as well as significant qualitative predicates which are frequently used in the referred related work.
Upon these investigations we established a framework consisting of mechanisms which are able to represent and describe spatial knowledge in terms of qualitative notions and concepts. We based our methods on existing approaches, most of them being developed with respect to cognitive considerations. We provided methods for representing positional information in form of orientation and distance as well as in form of semantical regions on the playing field. Moreover, we designed own methods to acquire qualitative spatial knowledge suiting our purposes. We studied Voronoi diagrams and employed them to model qualitative predicates such as reachability and free space. Additionally, we obtained further abstract information on a

soccer game like the current focus of play.
We also estalished mechanism to reason about the qualitative notions we provide. Instead of employing one of the calculi commonly used for qualitative spatial reasoning we took advantage of the reasoning facilities of the underlying agent programming framework. Therefore, we provided mechanisms to retrieve quantitative representatives for our qualitatively abstracted values. This allows us to use the existing projection mechanism in READYLOG to obtain qualitative descriptions of possible future situations. Thus, we are able to avoid the computational complexity adherent with common calculi.
In the READYLOG framework it is possible to combine programming and planning. A designer is able to guide the search in planning by specifying conditions which have to be true for a particular action to be taken. With the contributions of our work we simplified this task. The designer is now able to transfer his expertise in the domain with little effort. That is because he can use almost the same concepts and qualitative notions which are common in his knowledge of (human) soccer theory. We are able to specify the overall strategy and let the agent choose the best tactical pattern on its own.
We presented parts of an prototypical agent implementation which makes use of the newly established qualitative enhancements developed within the work of this thesis. We demonstrated the comfort in behavior specification we achieved through our work by providing abstract concepts that are closer to human cognition than numerical values are.

9.2 Future Work

The framework we established in this thesis gives impetus to future research in different directions. Several improvements to the READYLOG framework have been proposed. Most of them can be applied to discrete domains but there are problems in domains with continuous fluents. This is due to the fact, that in continuous domains the state space is far too big. With the state space abstraction we provide by our work, we may be able to apply some of these improvements.
Consider, for instance, the idea of options proposed by Fritz et al. [FFL03]. Options are not yet applicable to domains with continuous state space since it is impossible to formulate when an previously computed option can be applied. That is because the continuous fluent can take infinitely many values so that the chance of a fluent taking the value the particular option was computed for is not very likely. With the abstraction mechanisms provided by READYWORLD we render it possible to apply options in the continuous domain of ROBOCUP, too. That is to say, we can compute an option for qualitatively abstracted values and we can then determine very easily when to apply this option.
If options prove to be applicable with the help of the abstraction mechanisms developed in this thesis there are further applications. We could try to take the options already computed as new set of actions. It may be an interesting topic for further research on how decision-theoretic planning can be done upon these options.
Another improvement which was proposed in [Jac05] was the idea of caching intermediate results in decision-theoretic planning. Again, this is only applicable in discrete domains, yet. The set of values that a discrete fluent can take is sufficiently small to render it likely that two particular states consist of the same fluent assignment. Only in such cases the cached result can be re-used. If we use our qualitatively abstracted values instead of continuous fluents, chances of two states being identical definitively increases. Hence, caching may be enabled upon these

qualitative fluents.
We already mentioned that there is an ongoing work which is concerned with the task of action selection in the RoboCup domain. The idea is to use reinforcement learning methods to determine whether to execute a plan generated by our high-level component or to execute a reactive behavior. In order to decide which of the two concurring actions to execute their outcomes are observed and classified. This classification can be eased by using the abstraction mechanisms we presented in the scope of this thesis.
We employed Voronoi diagrams to model several properties like reachability and free space. There are extensions and specializations of Voronoi diagrams and Delaunay triangulations which deserve further investigations. They may allow to include additional dynamic aspects of the world such as the velocities of objects.
It is always good for a soccer team to be as flexible as possible. We investigated human soccer theory within our work to identify key issues of descriptions made therein. With our framework we are now able to transfer human insight into soccer to our autonomous robotic soccer agents. Thus, it is now possible to adapt even more tactical patterns and strategic moves from human soccer to the RoboCup domain. It may, for instance, be profitable to create a playbook for our agents, that is to say a collection of tactical patterns and moves. This would allow a team of agents to choose one of many possible strategies depending on the type of opponent they are playing against. With the aid of such a playbook agents could also switch their strategy depending on the current score.

To sum up, we provide a framework which achieves the improvements we intended. It allows for the abstraction of numerical information in the context of soccer. Moreover, it facilitates future benefits since it can have various further applications as we mentioned above.

Bibliography

[ABTI+00] Elisabeth André, Kim Binsted, Kumiko Tanaka-Ishii, Sean Luke, Gerd Herzog, and Thomas Rist. Three RoboCup Simulation League Commentator Systems. *AI Magazine*, 21(1):57–66, 2000.

[AHR88] Elisabeth André, Gerd Herzog, and Thomas Rist. The simultaneous interpretation of real world image sequences and their natural language description: The system Soccer. In Y. Kodratoff, editor, *ECAI 88. Proceedings of the 8th European Conference on Artificial Intelligence, Munich*, pages 449–54, John Wiley and Sons, Inc., UK, 1988.

[AK99] Minoru Asada and Hiroaki Kitano, editors. *RoboCup-98: Robot Soccer World Cup II*, London, UK, 1999. Springer-Verlag.

[AK00] Franz Aurenhammer and Rolf Klein. Voronoi diagrams. In Jörg-Rüdiger Sack and Jorge Urrutia, editors, *Handbook of Computational Geometry*, pages 201–290. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[BDP+02] Jean-Daniel Boissonnat, Olivier Devillers, Sylvain Pion, Monique Teillaud, and Mariette Yvinec. Triangulations in CGAL. *Comput. Geom. Theory Appl.*, 22:5–19, 2002.

[Bee99] Michael Beetz. Structured reactive controllers. In *AGENTS 99: Proceedings of the Third International Conference on Autonomous Agents*, pages 228–235, ACM Press, New York, NY, USA, 1999.

[BL99] Kim Binsted and Sean Luke. Character design for soccer commentary. In *RoboCup-98: Robot Soccer World Cup II*, pages 22–33, Springer-Verlag, London, UK, 1999.

[BRST00] Craig Boutilier, Ray Reiter, Mikhail Soutchanski, and Sebastian Thrun. Decision-theoretic, high-level agent programming in the situation calculus. In *Proceedings of the 7th Conference on Artificial Intelligence (AAAI-00) and of the 12th Conference on Innovative Applications of Artificial Intelligence (IAAI-00)*, pages 355–362, AAAI Press, Menlo Park, CA, July 30– 3 2000.

[CFH97] Eliseo Clementini, Paolino Di Felice, and Daniel Hernandez. Qualitative representation of positional information. *Artificial Intelligence*, 95(2):317–356, 1997.

[CGAL05] CGAL Computational Geometry Algorithm Library. website: www.cgal.org, 2005.

[CH01] Anthony G. Cohn and Shyamanta M. Hazarika. Qualitative Spatial Representation and Reasoning: An Overview. *Fundamenta Informaticae*, 46(1-2):1–29, 2001.

[DAA+05] Hesam Addin Torabi Dashti, Nima Aghaeepour, Sahar Asadi, Meysam Bastani, Zahra Delafkar, Fatemeh Miri Disfani, Serveh Mam Ghaderi, Shahin Kamali, Sepideh Pashami, and Alireza Fotuhi Siahpirani. Dynamic Positioning based on Voronoi Cells (DVPC). In *Proceedings of the RoboCup 2005 Symposium*, 2005. to appear.

[dBK02] Remco de Boer and Jelle Kok. The Incremental Development of a Synthetic Multi-Agent System: The UvA Trilearn 2001 Robotic Soccer Simulation Team. Diploma thesis, University of Amsterdam, Amsterdam, The Netherlands, 2002.

[DFL02] Frank Dylla, Alexander Ferrein, and Gerhard Lakemeyer. Acting and deliberating using golog in robotic soccer – a hybrid architecture. In *3rd International Cognitive Robotics Workshop (CogRob02)*. AAAI Press, 2002.

[DFL+05] Frank Dylla, Alexander Ferrein, Gerhard Lakemeyer, Jan Murray, Oliver Obst, Thomas Röfer, Frieder Stolzenburg, Ubbo Visser, and Thomas Wagner. Towards a League-Independent Qualitative Soccer Theory for RoboCup. In *RoboCup 2004: Robot World Cup VIII*, number 3276 in Lecture Notes in Artificial Intelligence, pages 611–618. Springer, 2005.

[DGL98] Giuseppe De Giacomo and Hector Levesque. An Incremental Interpreter for High-Level Programs with Sensing. Technical report, Department of Computer Science, University of Toronto, Toronto, Canada, 1998.

[DGL99] Giuseppe De Giacomo and Hector Levesque. An incremental interpreter for high-level programs with sensing. In Hector J. Levesque and Fiora Pirri, editors, *Logical Foundation for Cognitive Agents: Contributions in Honor of Ray Reiter*, pages 86–102. Springer, Berlin, 1999.

[DGLL00] Giuseppe De Giacomo, Yves Lespérance, and Hector Levesque. ConGolog, A Concurrent Programming Language Based on the Situation Calculus. *Artificial Intelligence*, 121(1–2):109–169, 2000.

[DHS00] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification*. John Wiley and Sons, Inc., New York, USA, second edition, November 2000.

[FFL03] Alexander Ferrein, Christian Fritz, and Gerhard Lakemeyer. Extending DTGolog with Options. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 1394–1395. Morgan Kaufmann, 2003.

[FFL04] Alexander Ferrein, Christian Fritz, and Gerhard Lakemeyer. On-line Decision-Theoretic Golog for Unpredictable Domains. In *Proc. of 27th German Conference on Artificial Intelligence*, pages 322–336. Springer, 2004.

[Fra98] Ian Frank. Football in Recent Times: What We Can Learn from the Newspapers. In Hiroaki Kitano, editor, *RoboCup-97: Robot Soccer World Cup I*, volume 1395 of *Lecture Notes in Computer Science*, pages 216–230, Springer-Verlag, London, UK, 1998.

[Fre92] Christian Freksa. Using Orientation Information for Qualitative Spatial Reasoning. In U. Formentini A. U. Frank, I. Campari, editor, *Theories and methods of spatio-temporal reasoning in geographic space*, pages 162–178. Springer, Berlin, 1992.

[Fri03] Christian Fritz. Integrating Decision-Theoretic Planning and Programming for Robot Control in Highly Dynamic Domains. Diploma thesis, RWTH Aachen University, Aachen, Germany, 2003.

[FSW04] Gordon Fraser, Gerald Steinbauer, and Franz Wotawa. Application of Qualitative Reasoning to Robotic Soccer. In *18th International Workshop on Qualitative Reasoning*, pages 173–178, Illinois, USA, August 2004.

[FZ92] Christian Freksa and Kai Zimmermann. On the utilization of spatial structures for cognitively plausible and efficient reasoning. In *IEEE International Conference on Systems Man and Cybernetics*, pages 261–266, Chicago, 1992.

[GL00a] Henrik Grosskreutz and Gerhard Lakemeyer. cc-Golog: Towards More Realistic Logic-Based Robot Controllers. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 476–482. AAAI Press / The MIT Press, 2000.

[GL00b] Henrik Grosskreutz and Gerhard Lakemeyer. Turning High-Level Plans into Robot Programs in Uncertain Domains. In *ECAI 2000, Proceedings of the 14th European Conference on Artificial Intelligence*, pages 548–552. IOS Press, 2000.

[Gro00] Henrik Grosskreutz. Probabilistic projection and belief update in the pGOLOG framework. In *The 2nd International Cognitive Robotics Workshop, 14th European Conference on Artificial Intelligence*, pages 34–41, 2000.

[Gro02] Henrik Grosskreutz. *Towards More Realistic Logic-Based Robot Controllers in the GOLOG Framework*. PhD thesis, RWTH Aachen University, 2002.

[HCF95] Daniel Hernandez, Eliseo Clementini, and Paolino Di Felice. Qualitative distances. In W Kuhn and A Frank, editors, *Spatial Information Theory: a theoretical basis for GIS*, number 988 in LNCS, pages 45–58. Springer-Verlag, 1995.

[Her91] Daniel Hernandez. Relative Representation of Spatial Knowledge: The 2-D Case. In D. M. Mark and A. U. Frank, editors, *Cognitive and Linguistic Aspects of Geographic Space*, pages 373–385. Kluwer, Dordrecht, 1991.

[HTF01] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, New York, NY, USA, 1st edition, 2001.

[HW94] Gerd Herzog and Peter Wazinski. VIsual TRAnslator: Linking Perceptions and Natural Language Descriptions. *Artificial Intelligence Review*, 8(2-3):175–187, 1994.

[Jac05] Stefan Jacobs. Applying READYLOG to Agent Programming in Interactive Computer Games. Diploma thesis, RWTH Aachen University, Aachen, Germany, 2005.

[Jan02] Norman Jansen. Entwurf eines Rahmenwerks zur Entwicklung von deliberativen Komponenten für unsichere, hochgradig dynamische Umgebungen mit Echtzeitanforderung. Diploma thesis, RWTH Aachen University, 2002.

[KFL04] Savas Konur, Alexander Ferrein, and Gerhard Lakemeyer. Learning decision trees for action selection in soccer agents. In *Proceedings of Workshop on Agents in Dynamic and Real-time Environments, 16th European Conference on Artificial Intelligence*, 2004.

[LPR98] Hector Levesque, Fiora Pirri, and Ray Reiter. Foundations for the situation calculus. *Linkoping Electronic Articles in Computer and Information Science*, 3(018), 1998.

[LR97] Fangzhen Lin and Ray Reiter. How to Progress a Database. *Artificial Intelligence*, 92:131–167, 1997.

[LRL+97] Hector Levesque, Ray Reiter, Yves Lesperance, Fangzhen Lin, and Richard Scherl. GOLOG: A Logic Programming Language for Dynamic Domains. *Journal of Logic Programming*, 31(1-3):59–84, 1997.

[Luc01] Massimo Lucchesi. *Coaching the 3-4-1-2 and 4-2-3-1*. Reedswain Publishing, 2001.

[McC63] John McCarthy. Situations, Actions and Causal Laws. Technical report, Stanford University, 1963.

[MRL+00] Rolf Müller, Thomas Röfer, Axel Lankenau, Alexandra Musto, Klaus Stein, and Andreas Eisenkolb. Coarse qualitative descriptions in robot navigation. In *Spatial Cognition II, Integrating Abstract Theories, Empirical Studies, Formal Methods, and Practical Applications*, pages 265–276, Springer-Verlag, London, UK, 2000.

[MSE+00] Alexandra Musto, Klaus Stein, Andreas Eisenkolb, Thomas Röfer, Wilfried Brauer, and Kerstin Schill. From motion observation to qualitative motion representation. In *Spatial Cognition II, Integrating Abstract Theories, Empirical Studies, Formal Methods, and Practical Applications*, pages 115–126, Springer-Verlag, London, UK, 2000.

[NMHF97] Itsuki Noda, Hitoshi Matsubara, Kazuo Hiraki, and Ian Frank. Soccer Server: A Tool for Research on Multi-Agent Systems. *Applied Artificial Intelligence*, 12(2):233–250, 1997.

[PS85] Franco P. Preparata and Michael I. Shamos. *Computational geometry: An Introduction*. Springer-Verlag, New York, NY, USA, 1985.

[Put94] Martin L. Puterman. *Markov Decision Processes: Discrete Dynamic Programming*. Wiley, New York, NY, 1994.

[Qui93] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, San Francisco, CA, USA, 1993.

[RCC92] David A. Randell, Zhan Cui, and Anthony Cohn. A Spatial Logic Based on Regions and Connection. In Bernhard Nebel, Charles Rich, and William Swartout, editors, *KR'92. Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*, pages 165–176. Morgan Kaufmann, San Mateo, California, 1992.

[Rei91] Ray Reiter. The Frame Problem in the Situation Calculus: A Simple Solution (sometimes) and a Completeness Result for Goal Regression. In Vladimir Lifschitz, editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 359–380. Academic Press, San Diego, CA, 1991.

[Rie03] Björn Riedel. Die Entwicklung von Ähnlichkeitsmaßen für den Vergleich von Spielsituationen im RoboCup. Diploma thesis, RWTH Aachen University, 2003.

[RN99] Jochen Renz and Bernhard Nebel. On the complexity of qualitative spatial reasoning: a maximal tractable fragment of the region connection calculus. *Artificial Intelligence*, 108(1-2):69–123, 1999.

[RN03] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition, 2003.

[SOM02] Frieder Stolzenburg, Oliver Obst, and Jan Murray. Qualitative Velocity and Ball Interception. In Matthias Jarke, Jana Köhler, and Gerhard Lakemeyer, editors, *KI-2002: Advances in Artificial Intelligence – Proceedings of the 25th Annual German Conference on Artificial Intelligence*, number 2479 in LNAI, pages 283–298, Springer, Berlin, Heidelberg, New York, 2002.

[SPS99] Richard S. Sutton, Doina Precup, and Satinder P. Singh. Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1-2):181–211, 1999.

[SWW05] Gerald Steinbauer, Jörg Weber, and Franz Wotawa. From the real-world to its qualitative representation – practical lessons learned. In Rinner B., Hofbaur M., and Wotawa F., editors, *International Workshop on Qualitative Reasoning*, pages 186–191, Graz, Austria, 2005.

[Tal79] Jerzy Talaga. *Fußballtaktik*. Sportverlag, Berlin, 2nd edition, 1979.

[TINF+98] Kumiko Tanaka-Ishii, Itsuki Noda, Ian Frank, Hideyuki Nakashima, Koiti Hasida, and Hitoshi Matsubara. MIKE: An Automatic Commentary System for Soccer. In *ICMAS '98: Proceedings of the 3rd International Conference on Multi Agent Systems*, page 285, IEEE Computer Society, Washington, DC, USA, 1998.

[TRF05] The RoboCup Federation. website: www.robocup.org, 2005.

[VAHR99] Dirk Voelz, Elisabeth André, Gerd Herzog, and Thomas Rist. Rocco: A RoboCup Soccer Commentator System. In *RoboCup-98: Robot Soccer World Cup II*, pages 50–60, Springer-Verlag, London, UK, 1999.