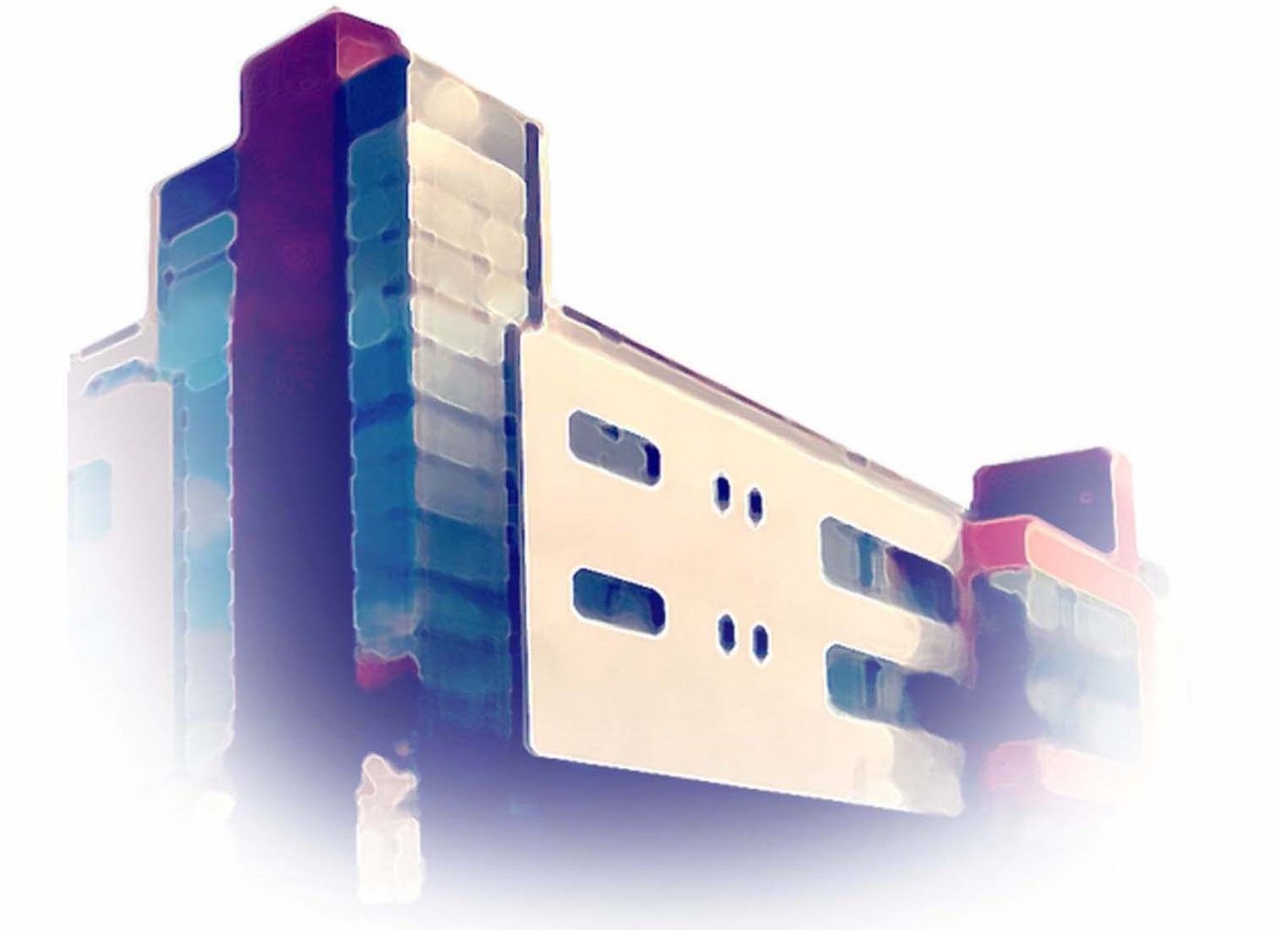
Timo Steffens

Feature-based declarative opponent-modelling in multi-agent systems



PICS

Publications of the Institute of Cognitive Science

Volume 4-2004

ISSN: 1610-5389

Series title: PICS
Publications of the Institute of Cognitive Science

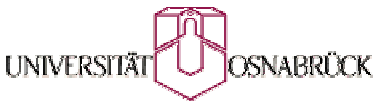
Volume: 4-2004

Place of publication: Osnabrück, Germany

Date: September 2004

Editors: Kai-Uwe Kühnberger
Peter König
Petra Ludewig

Cover design: Thorsten Hinrichs



Feature-based declarative opponent-modelling in multi-agent systems

Magisterarbeit im Fach Computerlinguistik und Künstliche Intelligenz
zur Erlangung des Magister Artium
im Fachbereich Sprach- und Literaturwissenschaft
der Universität Osnabrück

vorgelegt von:
Timo Steffens
geb. 14.06.1977 in Düsseldorf

16. Juli 2002

Contents

Chapter 1

Introduction

> One of the basic factors that effects the behavior of agents in MAS is the knowledge that they have about each other.
>
> *David Carmel*

As multi-agent-systems (MAS) grow in importance and are applied in many different domains [14], also the diversity of the types of agents in these systems grows. Agent designers can no longer hard-code all possible interaction situations into their software, because there are many types of agents to be encountered. These bring along a diversity of behaviors and tactics, which cannot be foreseen by the programmer. Because of this, demands on each agent's ability and flexibility to interact with the other agents rise, especially if the other agents have adversarial goals.
In order to act optimally it is reasonable to try predicting the other's behavior. New theories about the human brain claim that a high percentage of its capacity deals with predicting the future, including the behavior of other humans. Similarly, experiments in game theory showed that anticipating the opponent's future moves leads to better results than just reacting to its recent move [9]. A lot of work in MAS concentrates on finding out about the opponent's plans [44, 42].
To predict the opponent's behavior, some kind of model of the opponent is needed. Additionally, using models of collaborating agents can in fact turn out to be useful, too, in order to increase team cooperation while saving communication bandwidth. Both cases of models will be referred to as `opponent models` in this thesis meaning that the model refers to other agents. This

way, conflicts with the notion `multi-agent-modelling` are avoided which is used to describe the implementation process of agents.

There has been a lot of research on opponent-modelling, from general game-theoretic approaches over plan recognition and domain-dependent operator hierarchies. Yet, most approaches cannot be applied in dynamic, non-discrete multi-agent-systems.
In this thesis a method for representing opponent models is introduced which is feature-based and works in a wide range of domains, even in those in which no accurate evaluation functions exist. Instead of describing the complete opponent behavior like other approaches do, feature-based declarative opponent-modelling (FBDOM) focusses on typical and thus identifying tactical moves of the opponent. Just like humans compare their contrahent to former ones based on the most typical or striking properties, the observed agent behavior is matched onto the most appropriate opponent model. After this classification a corresponding counter-strategy is selected.
While this thesis introduces feature-based opponent-modelling as a general framework that can be used in different MAS, this method is also implemented and empirically evaluated in the RoboCup domain. Apart from the proof-of-concept, this evaluation also points out how parameters in the matching and classification process influence the results. Several experiments were run to measure the method's performance.

In chapter 2 an overview over game-theoretic approaches of opponent-modelling is given in order to show that information about the opponent's behavior is useful, but cannot trivially be used without spending some consideration. Chapter 3 describes instances of the abductive approach for acquiring models of the opponent. It is shown that feature-based approaches have been succesfully employed in areas like image recognition and intelligent tutoring. As well the shortcomings of plan-recognition and other opponent-modelling methods are discussed. Since RoboCup was selected as a testbed for the introduced framework, chapter 4 motivates why RoboCup is an appropriate domain for this, gives a short overview over the domain characteristics and copes with applications of automated analysis and modelling opponent behavior in the RoboCup domain. In chapter 5 feature-based declarative opponent-modelling is introduced and in chapter 6 an implementation for the RoboCup domain is presented and evaluated in a series of experiments. Finally, chapter 7 concludes.

Chapter 2

Using opponent models in Game-theoretic approaches

> An interesting game - the only winning move is not to play.
>
> 'Joshua' in the movie 'War Games'

2.1 Introduction

Game-theory is the basis of opponent modelling. It has always worked on the question how one can play optimally against an opponent. First approaches focussed on the game mechanics without considering the other player. Later it was realized that knowledge about the opponent's strategy can enhance the game results. Due to this realization opponent modelling was investigated.
Applications of this have not only been successful in classical games like chess, but also in other domains like economy, negotiations and warfare. Many of these aspects can be transferred to MAS. This chapter focusses on those game-theoretic approaches that form the origins of opponent models and are relevant for this paper. After describing these approaches their flaws will be discussed and it will be pointed out why they cannot be used in complex and dynamic MAS yet.
This chapter starts with an algorithm which does not model the opponent at all, but assumes that it acts optimally. The flaws and disadvantages will be pointed out. Based on this, another approach is described, which introduces

the concept of opponent classes and models of the opponent. The idea of opponent classes heavily influenced FBDOM. Finally, a technique to acquire opponent models is outlined. It will be shown why it is not applicable in certain domains, in order to motivate FBDOM.

2.2 Classical game theory

In a finite game there are a number of players who have available a set of discreet actions or moves. At each play of the game the players have to simultaneously select one action from their sets. In classical game theory it is assumed that each player knows not only its own set of actions, but also the opponent's. The same is true for the evaluation functions, which return the payoff of a play for the player. It is also assumed that both players are rational agents, i. e. they try to maximize their expected utility. The rules that the player uses to choose from among its available actions form the player's `strategy`.

In round-based games the minimax algorithm [38] is a method for perfect play in deterministic, perfect-information games, provided the opponent plays optimally. Yet, in complex games the opponent often plays suboptimally [18], because it is hard or impossible to find the optimal strategy. This can result in so-called swindle- or trap-situations, in which the opponent under- or overestimates a move, respectively. On the one hand, these flaws in the opponent's play can be exploited by actively creating situations in which he will make suboptimal decisions. On the other hand, the assumption that the opponent plays optimally can even lead to disastruous results. In the example in figure 2.1 player MAX assumes that MIN will play optimally. The game ends with a negative score for MAX, because on the top node he decides to execute the left action which promises a score of -2. But if he had known that MIN will overestimate its rightmost action, MAX would have chosen the right action, yielding a score of +2.

This reveals that knowledge about the opponent can be useful or is even necessary in order to get the best results. Yet, this knowledge has to be acquired first, e. g. by observing him during a number of games. Instead of observing, the approach, which is described in the next section, makes assumptions about the opponent's membership in certain opponent classes.

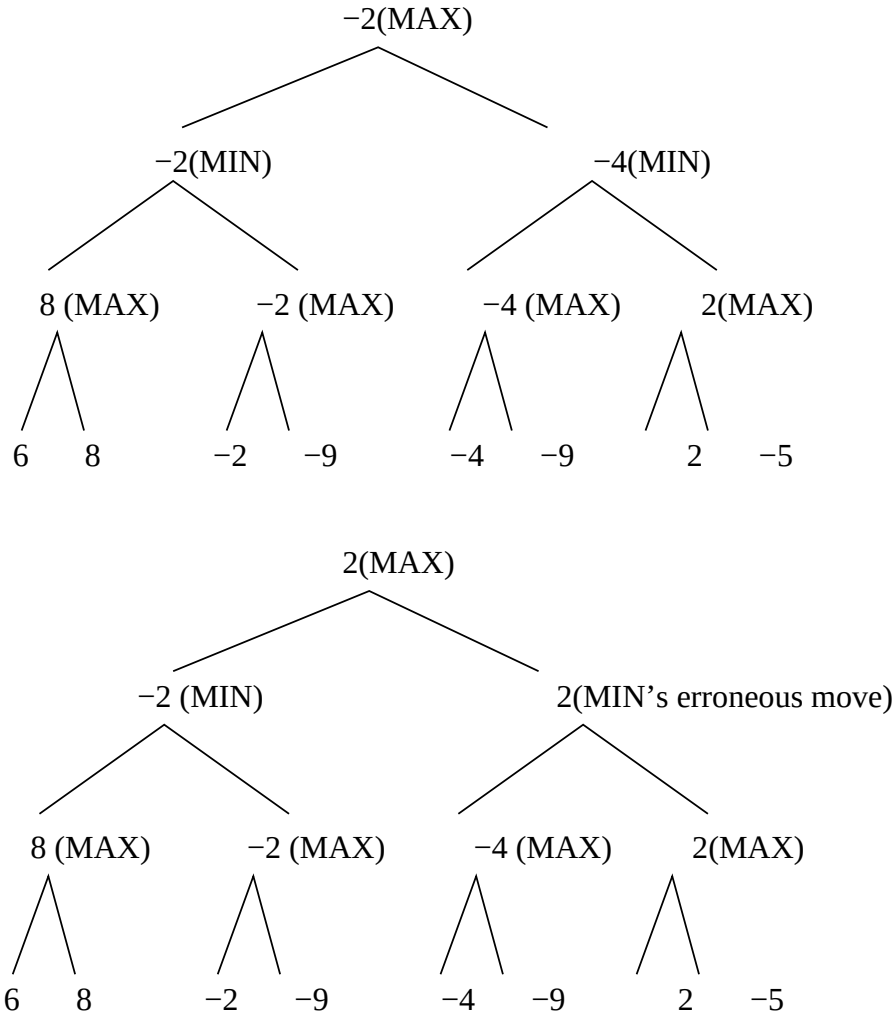


Figure 2.1: The top tree describes a play with the minimax-algorithm assuming that the opponent plays optimally. The bottom tree describes the game as it would have turned out if player MAX had known MIN's real suboptimal strategy. Evaluation values at the leaf nodes depict the utility which the perfect evaluation function would return. MIN's calculated values differ from these perfect values and are not shown. Values at the parent nodes describe which action the player chooses.

2.3 Opponent models in search

While the minimax-algorithm uses no knowledge about the opponent's decision procedure, extensions of it do. One such algorithm is described in this section.
Carmel and Markovitch [9] introduce a generalization of the minimax-algorithm that uses a model of the opponent to benefit from its flaws. This is the beginning of opponent modelling and introduces classes of opponents. The models are not rebuilt for each new game. Instead it is assumed that there is a fixed set of models which models all possible opponents. Based on this classification the player is then able to select the appropriate action. The basic idea is very similar to feature-based opponent modelling. Yet, the models themselves are quite different.
Carmel and Markovitch define a recursive opponent model which consists of an evaluation function and a player model that the opponent holds. This way it is possible to reason about how the opponent reasons about other players (including oneself).
Let S be the set of possible game states. Let $\rho : S \to 2^S$ be the successor function, which returns the set of successor states. Let $\phi : S \to S$ be an opponent model that specifies the successor function that the opponent uses. Note that this does not return a set, because the evaluation function determines the opponent's optimal move. Now, the M function takes a position $s \in S$, a depth limit d, a static evaluation function $f : S \to \mathbb{R}$, an arbitrary opponent model ϕ. It can be defined to return the value of the best applicable move or the move itself:

$$M(s, d, f, \phi) = \begin{cases} f(s) & : \quad d \leq 0 \\ \max_{s' \in \rho(s)}(f(s')) & : \quad d = 1 \\ \max_{s' \in \rho(s)}(M(\phi(s'), d-2, f, \phi)) & : \quad d > 1 \end{cases}$$

M selects the best move for the player by generating the successors of the current state. By applying the opponent model ϕ it obtains the opponent's moves for each of its own actions and evaluates them recursively, reducing the depth limit by 2 plies (half-rounds). This specific reduction of the depth limit is due to the fact that the algorithm returns values for the player's own ply at depth 0 and the opponent's next move at depth 1 directly. The player's next move is 2 plies away, which is one round. Finally, when the depth limit is zero at the end of the recursion, the static evaluation function is used. Note that the classic minimax-algorithm with a depth limit of d uses

the negated evaluation function of the player and assumes that the opponent searches down to $d-1$. So it is just a special case of M. If the player is using an evaluation function f_0, the minimax algorithm can be written as a special form of M as

$$M^0_{(<f_0>,d)}(s) = M(s, d, f_0, M^0_{(<-f_0>,d-1)})$$

To illustrate how this opponent modelling works, let us assume that the player uses another evaluation function f_1 and knows that the opponent uses f_0. This allows us to define another special case of M:

$$M^1_{(<f_1,f_0>,d)}(s) = M(s, d, f_1, M^0_{(<f_0>,d-1)})$$

The M^1 algorithm simulates the opponent's reasoning using f_0 down to depth $d-1$ to find the opponent's selected move and evaluates those by calling itself recursively to depth $d-2$. Note that the depth is counted in terms of plies and not rounds.
In the same way, one can define the M^n algorithm which assumes that the opponent uses the M^{n-1} algorithm:

$$M^1_{(<f_n,\ldots,f_0>,d)}(s) = M(s, d, f_n, M^{n-1}_{(<f_{n-1},\ldots,f_0>,d-1)})$$

So, if there is a correct model of the opponent, the player can benefit from this knowledge.
Several problems are still inherent in these algorithms. First of all, these algorithms work in round-based games and it is not clear how they can be generalized to parallel multi-agent domains. Also, in domains that allow non-discrete actions (that e. g. use real numbers as parameters), the sequential checking of all possible successor states is infeasible. This would also make it hard to create an efficient and accurate evaluation function. So the aim of this thesis is to develop a framework which can handle opponent-models in MAS with noisy and non-discrete actions, without using an evaluation function.
Finally, the opponent model needs to be acquired. This last point is dealt with in the next section.

2.4 Learning opponent models

Throughout the following chapters different methods to learn or infer opponent models will be mentioned. In this chapter a game-theoretic approach is described which forms the basis of most work in this area.

Carmel and Markovitch [10] suggest a method to find an opponent model that is consistent with the observed input and ouput of that opponent. They consider only repeated two-player games and strategies that can be modeled as finite automatons (DFA), though. Given a DFA opponent model, there exists a best response DFA that can be found in polynomial time [28]. Therefore, learning such a DFA opponent model leads to good play.
While finding the minimal DFA has been proved to be NP-complete, Carmel and Markovitch show a poly-nomial method for finding some DFA that is consistent with the observed data. The model is learned in an unsupervised way, because there is no teacher available, and experiments with the opponent might be too costly or impossible at all.
The algorithm works by holding a consistent model throughout encounters with the opponent. If an observation is made that is inconsistent with the model, it has to be modified. In such a case the observation table of the DFA has two entries that are in the same equivalence class, but their outputs are not. So these two entries have to be extended so that they are not equivalent anymore, while keeping the whole table consistent and closed.
The idea of maintaining consistent models instead of one optimal model is used in feature-based declarative opponent modelling.
Carmel and Markovich applied their DFA-learning algorithm only to repeated two-player games. Since they use an observation table, it is restricted to discrete domains and the approach is very sensitive to noise. If an output observation is blurred by noise, it will not belong to the same equivalence class as another observation that describes the same situation. It remains unclear, how this algorithm can be scaled up to multi-agent-systems. In the two-player-games considered by Carmel and Markovich, each action and round can be completely described by one action per player. If and how the automatons can be extended to work with more complex situations, and thus more complex input alphabets, is not tackled in their papers.
Creating the opponent models automatically is out of the scope of this thesis. Yet, since FBDOM also uses models of other agents, some general ideas for acquiring them are outlined later. These models have to be able to express the interactions of multiple agents in noisy and dynamic environments. In order to evaluate FBDOM, several opponent models were created manually.

Chapter 3

Abductive approaches

3.1 Introduction

Most of the work this paper builds upon, is basically concerned with finding the best explanation for observed data, which is according to [11] just a form of abduction. According to the classic form ($a \rightarrow b$ and b lead to a), the idea is that an agent employing a certain model will show certain obervations. From these observations its model can be inferred.
Since this is a very common method, this chapter gives an overview over different approaches like plan recognition, intelligent tutoring, image recognition and multi-agent modelling, upon which the methods in this paper are built. All of the following approaches in this chapter are basically about mapping observations into some kind of model.
It is shown how Intelligent Tutoring successfully models students by means of so-called features. This notion of feature will be used in FBDOM, too. Also image recognition, of which an overview will be given, contributes to FBDOM, by introducing certain properties that the features in a model have to satisfy. Finally, the standard methods for finding out plans and actions of agents will be examined for their applicability in general multi-agent systems.

3.2 Intelligent Tutoring

Interestingly, intelligent tutoring is related to opponent modeling, too. The basic idea in this area of research is to infer a model about the cognitive abilities of the student based on the input and output of the cognitive system

[2]. Such models are used to predict the answers of students to questions, thus making it possible to support learning by providing appropriate problem tasks. They can also be used to explain mistakes that were made. Yet, trying to model the internal processes of the cognitive system is most likely to fail or at least face many difficulties because of the problems that are related to introspection and deriving internal mechanisms from external observations.
Webb introduces a feature-based modeling paradigm that avoids assumptions about cognitive processes [47]. He uses attribute-value machine learning techniques where the attributes are features of the actions and the context of the tasks. So all of these features are observable externally and do not try to model the internal processes.
The model that is learned contains associations of the form $X \rightarrow a$, where X is a set of context features, and a is a single action feature that applies to the student's action in context X. For example, in a szenario in which the student is supposed to learn substraction, an association might look like

$$\{subtrahend\ is\ zero\} \rightarrow result\ equals\ the\ minuend$$

These associations are generalized to achieve compact models. So, for example the above association can be derived from

$$\{subtrahend\ is\ zero,\ minuent\ is\ positive\} \rightarrow result\ equals\ the\ minuend$$

and

$$\{subtrahend\ is\ zero,\ minuent\ is\ negative\} \rightarrow result\ equals\ the\ minuend$$

Because of the static and finite nature of the domain, it is possible to learn models of the student. It should be noted that the learned associations only have the expressiveness of propositional logic. Not only because of this, just like in section 2.4, these methods cannot be transferred to dynamic, non-discrete domains, in which multiple agents act and have to be modelled. Yet, the idea of using features to build a model of an agent is important for FBDOM.

3.3 Image recognition

Image recognition is concerned with determining the object which is most likely to have generated a given image. The general insight is that images

can be modeled as conjunctions of local features. That is, the observation of features in the image needs to be mapped onto models of objects [6]. In most implementations these features are very simple, like edges, that are supposed to capture the most important characteristics of the image, independent of lighting and hue variations. Models of an image are very compact, because they are just a list of the features and their positions in the image. This idea is important for feature-based opponent-modelling. Features do not describe the whole image, but only identifying pieces.
The problem that emerges in image recognition is called the correspondence problem. The predefined features have to be identified in the image. Since the features are very simple, they tend to be found in many different images. For example, a vertical edge of a certain length can be found nearly in all images. So identifying such a simple feature does not carry much class information and is not very expressive for identification.
Instead, complex feature recognition [45] does not use predefined features, but learns complex features that contain more class information. Instead of bitmaps of single features like vertical or horizontal edges, a complex feature is represented by a series of templates. These templates show a part of the object (e. g. the nose in a face) in different poses. In order to increase independence from lightning conditions, these templates contain not explicit color values, but intensity values relative to the neighbors. Complex features need to be distinct, i. e. they appear only in the class of images that they are supposed to code, and to be stable, i. e. they appear rather constant in different poses and lighting conditions. These two terms are crucial for selecting the appropriate features for feature-based declarative opponent models.
While a large set of complex features is to be maintained by the image recognition system, each object is modeled by only a handful of complex features. This facilitates the correspondence problem and yields computationally effective recognizing algorithms. It is hoped that this success can also be transferred to feature-based opponent modelling.

3.4 Plan recognition

Plan recognition has developed parallel to research in planning, and is in a way the reversed process of plan execution. While planning deals with finding a sequence of actions that fulfill a given goal and plan execution is the sequential handling of the planned actions, plan recognition tries to infer

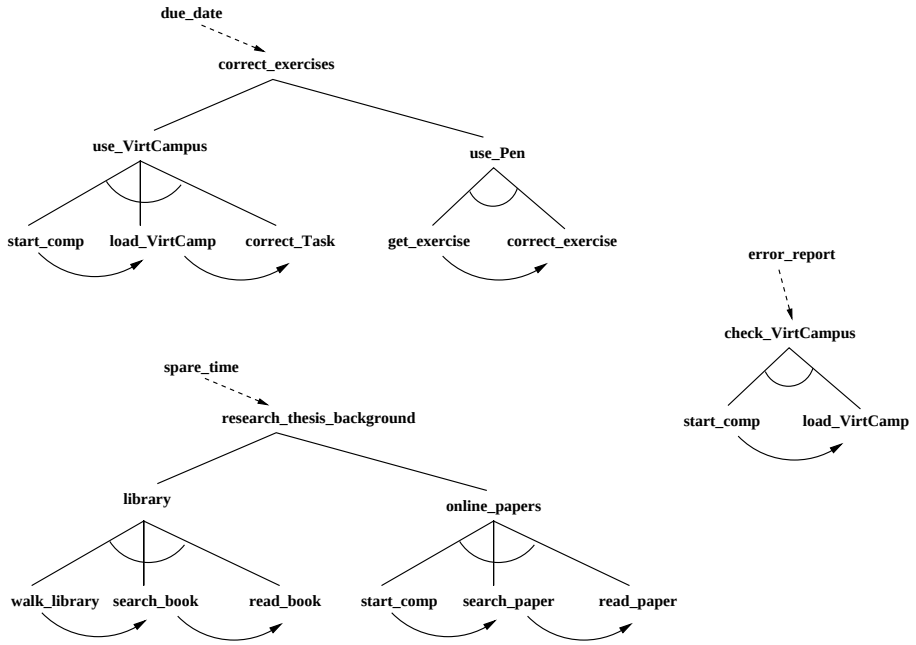


Figure 3.1: A plan library containing three plans.

the plan that an agent executes from observing actions. Owing to that, it is focussed on deliberative agent architectures.
More specifically, Kautz and Allen [20] defined the problem of plan recognition as the problem of identifying a minimal set of top-level actions that are sufficient to explain the observed actions. Their work is the basis for most plan recognition approaches [15]. In their representation, plans are graphs with the top-level actions as root nodes. Other actions, decompositions or specializations, depend as nodes from the top-level actions. Plan recognition is then a form of graph covering.
Agents that try to recognize plans of other agents first of all need a plan library, that contains plans that the observed agent is likely to execute. Most work uses hierarchical plans, i. e. plans that decompose tasks on different levels [15, 29, 30]. A plan library might look like the one in figure 3.4.
Note that the depicted plan library can be used by the plan recognizing and the plan executing agent. A plan is triggered by an event (depicted by the dashed line). These are not observable by the recognizing agent. In our example, there are three plans for three goals: `correct_exercises`, `research_thesis_background` and `check_VirtCampus`. A goal can have sev-

eral methods to achieve it. For example, `research_thesis_background` can be achieved by the methods `library` or `online_papers`. Methods can be decomposed into several steps. E.g. `library` requires the agent to sequentially execute the steps `walk_library`, `search_book` and `read_book`. The temporal orderings are depicted as arrows. The method- and step- graph can be viewed as an AND/OR-graph, in which the methods are connected by OR and the steps by AND.
This view can be used for plan-recognition. Suppose that the recognizing agent observes the action `start_comp`. It can infer

$$(correct\_exercises \wedge use\_VirtCampus) \vee$$
$$(research\_thesis\_background \wedge online\_papers) \vee$$
$$check\_VirtCampus$$

In order to disambiguate the plans, it needs more observations. So, if `load_VirtCampus` is observed, the minimal explanation that covers these actions is

$$(correct\_exercises \wedge use\_VirtCampus) \vee$$
$$check\_VirtCampus$$

Interestingly, most plan recognition systems cannot represent the fact that an event has not been observed yet. Suppose that after `load_VirtCampus` no further action is observed. While intuitively the `check_VirtCampus`-plan becomes more and more probable, most plan recognition systems fail to use the information of not-observed actions. This is adressed in [15, 29]. Viewing the problem from a different perspective, Tambe and Kaminka proposed to use knowledge about social roles to detect if missions have to be aborted due to failures [19].

As mentioned above, plan recognition focusses on deliberative agent architectures. So it deals quite well with the temporal ordering of actions and hierarchical plans. Yet, reactive behaviors or plans that change rapidly in dynamic domains cannot be modelled.

3.5 Opponent modelling in MAS

Opponent modelling in MAS builds upon the work described above, yet it has to cope with difficulties that are not encountered in game-theory or in classic plan recognition. For example, in some MAS models should be

able to express mutual goals and actions of several agents. In other domains it is important to reason about reactive rather than deliberative agents or to act simultaneously with the opponent while at the same time modeling it. Only recently has attention in plan-recognition shifted to dynamic, real-time environments, and modeling of multi-agent teamwork. Several of these approaches will be described in this section.

In an information economy domain Vidal and Durfee [44] show in which cases it is advantageous to model the other agents. They suggest nested agent models similar to the M*-algorithm mentioned in section 2.3: For example, a 0-level modeler does not model other agents at all. A 1-level modeler assumes that the other agents are 0-level modelers, and tries to learn their decision functions. In their work, Vidal and Durfee also run tests with 2-level modelers that assume that the other agents are 1-level modelers. These experiments show that higher-level modelers can exploit lower-level modelers. It was stated briefly that in some cases the assumption of a wrong modelling-level leads to suboptimal behavior. While this intuitively makes sense, such observations would need more systematic evaluations.

Vidal and Durfee's approach needs an utility function, and assumes that the agent interaction is one cycle of bidding and buying/selling. Thus, the world offers perfect information, because the bidding and the actual used price are always available. For domains other than market-simulations, other approaches have to be found. While recursive modelling is an interesting subject, it is not tackled in this thesis.

In domains that are only partially accessible, agents also need to infer unobserved actions from those observations that they were able to make in order to reason successfully about the plans and goals of the other agents. Tambe et al. provide a framework for event-tracking [42] that proves successful in military simulations. In such environments it is important to infer unobservable actions like the remote launching of a missile by observable events like turning and radar guidance maneuvers. Tambe uses an operator hierarchy that is implemented in, but not specific to, the Soar architecture [25]. This operator hierarchy is similar to hierarchical plans and decomposes goals step by step into actions. By matching observed events into the hierarchy one can infer the goals in the higher levels.

Such domains require the ability to model reactive or rapidly changing deliberative behaviors. To achieve this, a recursive agent-tracking technique is proposed that uses the agent's own decision architecture to model the opponent's decision process. The agent assumes that the opponent uses a similar

operator hierarchy and just executes the hierarchy out of the opponent's view. This of course may also include modeling the agent out of the opponent's perspective and recursively ad infinitum. Tambe et al. reason that this can be done computationally by so-called model-sharing. Since the model of the agent and the opponent are always almost the same within the chain of recursion, they don't have to be executed all over again. Yet, Tambe's agent-tracking relies heavily on the assumptions that the modelling and modelled agent use the same operators and problem spaces. It might be the case that in military scenarios standard maneuvers exist which the opponent also knows. But in order to track an opponent, Tambe also makes the simplifying assumption that the modeller knows the opponent's exact state. And in the general case it is impossible to estimate or infer the exact state of another agent from observation, especially if several actions of the other are impossible to detect [41]. Particularly, the recursive agent-tracking does not appear in Tambe's later papers about agent modelling.

Intille and Bobick [17] present a framework for recognizing multi-agent actions from visual evidence and evaluate it by analyzing taped games of American football. Similar to the image recognition work in section 3.3 they use models that consist of local events. Additionally they define temporal relationships between these events, for example `before` and `after`. For example, a `run-with-ball` action might have the prerequisite that a `catch` action occurs `before` it. This is inspired by plan recognition and works well for American football which has a lot of typical `plays`, i. e. constant tactical moves. Each primitive action is encoded in a belief network, a method shown to be able to represent agent goals and plans so that they are suitable for plan recognition [16]. If these actions are detected by the networks, then the temporal relationships between them are tested. The networks consist of two kinds of nodes (see fig. 3.2): The first are called unobservable belief nodes that can be either true or false and contain internal states of the modeled agent that cannot be observed. The other are observable evidence nodes and represent the observed actions. For example, it cannot be observed directly, if a player is `readyToCatch`. Such an unobservable event has to be inferred from observable facts like if it is `facing` the object and close enough. So, unobservable internal states are a source of ambiguity. It will be described later, how FBDOM handles these unobservable states.

Observable events are n-ary, so for example the result of a `distant` detector can be `inContact`, `nextTo`, `near`, `far` and so on. The belief nodes can also be filled with other networks. This violates the assumption that all dependencies

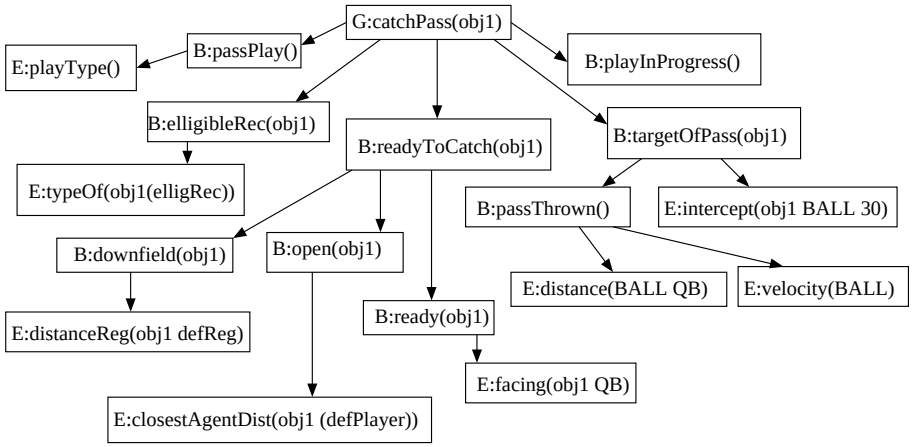


Figure 3.2: A belief network, representing the an agent's plans in American football. Nodes marked with B are unobservable states, nodes with E are observable events.

are represented in the given network, but Intille and Bobick argue that this approximation works because the whole network is just an approximation of the agent's goals.

This shows another aspect of Intille and Bobick's approach: The use of fuzzy descriptions. These fit well into the probabilistic nature of Bayesian networks.

Such networks can then be combined to `multi-agent networks` in which a multi-agent action like a `play` is decomposed into the actions of the individual agents. Between these actions exist fuzzy temporal relationships like `before`, `around` and so on. The recognizer then checks if the predicted events can be observed at all and if so, whether they happen in the correct temporal order.

An extension of belief networks is also the basis of Suryadi and Gmytrasiewicz's work [40]. While Intille and Bobick's approach focussed on matching observations to predefined networks, Suryadi's work is about learning these networks. Since the correct model is usually not known with certainty, a probability distribution over the different models is kept. If the observed behavior of an agent is not likely to match any of the models, then one of them is modified to better account for the observations.

3.6 Conclusion

As was described above, previous abductive approaches which map observations to models span a variety of applications and domains. While several ideas like using features will prove useful for an opponent modelling framework in MAS, none of the mentioned approaches is suited to be applied in general MAS. Some are from totally different domains like image recognition, others are from single-agent-domains. Those that are MAS approaches, make restricting assumptions about internal states, constant moves/plays or are not well suited for noisy and non-discrete applications.
Some of these issues are tackled by work which will be described in the next chapter. Yet, as will be seen, these methods are domain-specific to simulated-soccer. A great deal of attention is currently focussed on opponent-modelling in RoboCup[37, 35, 32, 46]. Unfortunately, it is doubtful if they can be transferred to other MAS domains.
The aim of this thesis is to provide a framework which can represent and handle opponent models in general MAS. It will incorporate ideas raised by the outlined approaches, like the use of distinct and stable features or focussing on externally observable events while avoiding assumptions about internal processes.

Chapter 4

Opponent modelling in RoboCup

> I came to the conclusion, that the whole purpose of the simulation league was to come up with coaching.
>
> *Gal A. Kaminka*

The Robot World Cup Initiative (RoboCup) is an attempt to encourage AI and intelligent robotics research by providing a standard problem in which a large variety of technologies can be integrated and examined [22]. Because for a robot team to fare well in a game of soccer it needs to employ many intelligent techniques, RoboCup is well suited to evaluate different approaches and measure the progress of technology [21]. Just like computer chess was the standard challenge problem for AI before Deep Blue beat the human World Champion, RoboCup was proposed to be the new standard challenge problem with the goal to beat the human Soccer World Champion with a team of robots by the year 2050.
The challenges for the near future also include opponent modelling [21], because it is not only a key technology for RoboCup, but also for general multi-agent interaction. Because robotic soccer is such a complex domain, i. e. real-time, non-deterministic, dynamic, the opponent-modelling techniques in the previous chapters cannot be used as off-the-shelf solutions. While they are a good basis to build upon, there is more research needed for applying them in this and similar domains. This chapter will give an overview of the domain

and document the state of the art of opponent-modelling in RoboCup.

4.1 The environment: RoboCup simulation league

RoboCup has different leagues, with different sizes of robots and different key challenges. There are the humanoid, middle-size, SONY-legged, small-size and simulation league. While the research focus of the first three leagues up to now is on hardware issues like vision and mechanics, only the latter two and particularly the simulation league are already prone to use opponent modeling. Since for the evaluation experiments in this paper the simulation league is used and the fundamental techniques were developed in it, it will be described in this section.

4.1.1 General remarks

The RoboCup simulation league uses the Soccer Server System [12] to simulate the field and the objects (see figure 4.1). Each player has to be a unique process that connects via a standard network protocol to the server. The players receive visual and audio information every 150 msec over the network and can issue primitive actions like kick, dash, turn, turn-neck and say every 100 msec. The server processes the actions of the players and generates new visual information. The later consists basically of the distances and angles to other players, the ball and landmarks. The players can only perceive objects that are in their field of vision and both the visual information and the execution of the actions are noisy. Additionally, the accuracy and amount of sensory information decreases with the distance of an object. Communication between the clients is only allowed if it passes via the server, and the bandwidth and hear range are limited.

4.1.2 Coaching

In addition to the players, each team may connect a priviledged client to the server, the so-called online-coach. It resembles the human trainer or coach in real soccer and may not execute actions on the field. But it receives visual information, which is global and not noisy. That is, at any time step the coach receives the correct absolute positions and velocities of all players and the

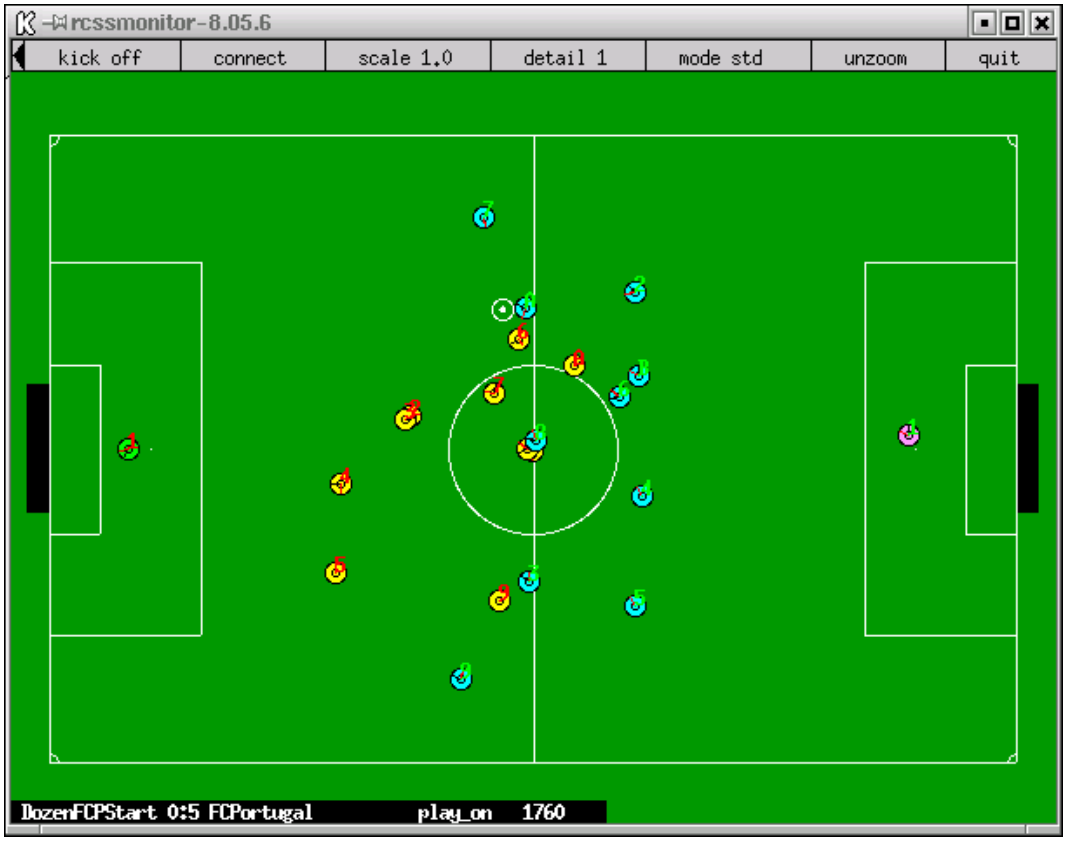


Figure 4.1: The playing field, as simulated by the soccerserver.

ball. It is also allowed to communicate arbitrary messages to the players while the game is stopped, and messages in a standard language at certain intervals. This makes the coach a valuable tool for tactical analysis and opponent modelling [7].
The same information that is sent to the coach during the game can be recorded into a logfile, so that more time-consuming analysis methods can be executed offline.

4.1.3 The standard coach language

The standard language that the coach can use to communicate with its players at certain intervals is CLang [12]. It is basically a production system that maps situations to actions. Depending on the type of speech act [3], the rules are interpreted differently. If uttered as assertive, the rules express the belief that they are likely to describe the behavior of a player or a set of players. As a directive, the rules are an advice of the coach to act in a certain way in the specified condition.
With this language it is not only possible to work on opponent-modelling, but also to compare different approaches by testing them with different teams and coaches.

4.2 Opponent-modeling approaches in RoboCup

To promote research on opponent-modelling, the influence of the online coach has been fostered and a coach competition has been established. This section outlines the complexity of the coaching problem and gives an overview of the used approaches.

4.2.1 The coaching problem

Riley et al. define the coaching problem as the task to provide advice to a number of distributed agents [37].
The coaching problem subsumes opponent-modelling, so many issues that arise within coaching also have to be adressed in opponent-modelling in complex domains like RoboCup.

In this section, this thesis identifies several new issues which have to be dealt with. The first is the representation and reasoning about strategies. One major issue is the modularity of strategies, meaning that strategies are composed of different substrategies or tactics that refer to different situations of the game. E.g. in ball-sports domains a complete team strategy has to comprise tactics for offensive and defensive situations. While these components refer to different situations, they nevertheless have to interact with each other. If the defensive behavior results in a spatial distribution of players that is suboptimal for offensive actions, the acquisition of the ball leads to time-consuming repositioning. This means that the coach cannot arbitrarily change parts of the strategy without considering the effects on the other parts.
Also in MAS the coordination of cooperative players must be addressed in coaching and in opponent-modeling. The representation of strategies for example has to consider that the execution of a pass not only requires the ballowner to shoot the ball, but also a teammate to receive it. Automated analysis thus has to keep track of the intentions of several players and needs to disambiguate events like player movements which can be part of the goal to mark or attack an opponent, to occupy a defensive position or to run free for being a potential pass-receiver.
Another issue is the changing of the observed tactics. This is similar to the problem that user-modeling faces when trying to adapt to shift or drift in users' interests[23]. Most analysis methods in user- and opponent-modelling require observations during a certain time-span. If the behavior changes during this time, the result might be inaccurate and biased. Also it is hard to specify thresholds that determine which changes are due to a change in behavior and which are due to the natural fluctuations and noise.
As mentioned in section 2.4, it is possible to calculate the best response to each opponent's move only for strategies that can be modeled as finite automatons. Yet, to the author's knowledge in complex domains like RoboCup, that even lack general evaluation functions for states, there are no general approaches for deriving counter-strategies. The response to an opponent tactic is either hard-coded or obtained in time- and ressource-consuming experiments. Even if the model can be used to predict the behavior of the opponent, the lack of an evaluation function for successor states makes it hard to select the best action. So, even if an agent acquired an accurate model of the opponent, there is still a long way to go before playing optimally against it.
The following sections describe different approaches of opponent-modeling

and coaching in the RoboCup domain.

4.2.2 External opponent-modelling

While in other MAS it is possible that the agents are deployed and then have to act autonomously for a long time without interference of their designers [19], in RoboCup the execution of the agent teams is limited to a game. This leaves their designers with the possibility to change the behavior between games by manually changing parameters or pieces of code. Thus they adapt the team to the future opponent by watching logfiles of previous games. While this leads to good results [31], the approach does not satisfy the RoboCup Challenge, because human interference is to be minimized. So, automated opponent-modelling should be preferred.

4.2.3 Using ideal opponent models

The first step into automated opponent-modelling is to assume that the opponent plays optimally [39]. For example, if a player wants to intercept the ball before a certain opponent player does, it assumes that the opponent has a correct world model, runs to the ball using the optimal interception route and does not miss server cycles. This way, the agent can reason about the needed speed and power that it has to spend in order to get to the ball first. Or, if the needed power is not available, it can give up trying.
This resembles the approach of the classic minimax-algorithm (described in section 2.3) and does not use knowledge about the opponent at all. So just like in the M* argumentation, idealized opponent models can lead to overestimating the opponent and thus missing chances or generating wrong predictions. On a theoretic and general view, two situations are described in [18], where it is important to consider the opponent's strategy. In the so-called swindle position, the player has good reason to assume that the opponent will underestimate a good or the best move and will execute a poorer move instead. The other situation is called the trap position, where the player expects the opponent to overestimate and execute a bad move. Building on this, a player can use opponent models if it is in a losing position [4], where all moves would lead to worse situations. But by using an opponent model, the player might favor a move which leads to a situation in which the opponent is likely to be in a swindle situation.

There are concrete examples for swindle situations in RoboCup. In some teams it was often observed that players do not shoot towards the goal, although neither goalie nor defenders were between the ball and the goal-line. It is not clear why this happens. It might be the case that the agents assume that their world model is incorrect and turn around endlessly in order to update it with visual information. Using the argumentation in [4], players who have good reason to believe that they will lose the ball, might choose to steer the ball to the opponent who is in the described swindle situation. Yet, this requires a realistic and not an optimal opponent model.

4.2.4 Formation recognition

Very early work on the online coach includes the recognition of formations. Formations are deemed to be an important part of a team strategy, so adapting them to the opponent's formation promises better scores.
One approach feeds the observed player positions into a neural network and tries to classify them into a predefined set of formations [13]. If a classification can be done, the appropriate counter-formation is looked up and communicated to the players.
The recognition of the formation is a classic abductive approach, that is, finding the best explanation for the observed events. The appropriate counter-formations cannot be inferred automatically, but have to be evaluated empirically by testing them against each other before-hand.
Although formations are an important part of soccer tactics, there are many more concepts of a team strategy which have to be represented in a different way.

4.2.5 Methods used during the first coach competition

Several techniques of opponent-modelling were used during the first coach competition at RoboCup 2001. One method was to not model the opponent at all, but to send hard-coded advice that was deemed to be helpful. Another approach was to statistically identify dangerous zones and advice the players to avoid those [1]. Similarly based on statistical observations was to advice marking assignments which depended on the modeled opponent and team formation in order to match defenders to close forwards [7]. As of now there are no experimental results that provide information about the impact on team performance.

A learning approach was demonstrated that recognized formations from observed player positions. Such a formation consists of one rectangle per player and the learning algorithm returns the smallest rectangle that covers the majority of player positions while removing bias by ball-oriented shifting [37]. The effect on the teams was very small.
The OWL coach had learnt rules from previous games that described the passing behavior of different teams [37]. First the coordinates of the receiver and the shooter are clustered. After this, rules are learnt that include the name of the receiver and the shooter cluster, and the distances and angles of all players to the ball. These pass rules were used to mimic the passing behavior of good teams, thus improving the passing quality of the coached team, or to predict passes of those opponents that were analysed. However, experiments revealed that these rules did not improve the performance of the team.
An important observation is that the players of no team processed so-called info-messages (except for identification of heterogenouos player types). Info-Messages are an essential part of CLang and are used to provide opponent modelling information to the players. They contain descriptions about the behavior of the opponent. Although the players did not process them and only made use of advice-messages, which contain explicit directives how to act in certain situations, several coaches sent info-messages. This gives rise to the hypothesis that it is easy for the coach to come up with descriptions of the opponent's behavior, but very hard for the players to infer appropriate actions based on such knowledge.
Several implementation bugs and lacking autonomy in the player clients made the clients misinterprete coach advice or follow erroneous advice without consideration. So the coach competition 2001 could not be used to evaluate and compare the different approaches. The probably most advanced approach which was demonstrated at the coach competition and which this work builds upon will be described in the next subsection.

4.2.6 Probabilistic positional opponent models

The assumption of Riley's work, which is important for FBDOM, is that human players observe the opponent's behavior during a game and compare it to that of previously played opponents. Based on these observations they try to identify the style of strategy and then adopt the strategy that was effective against the previous adversary [34].

This assumption is also shared by the approach of this thesis. It is reasonable to imagine that the advantage of more experienced players is (apart from physical training) partly due to the fact that they have more knowledge about what to do in certain situations. This knowledge has to be acquired by playing many games or by tactical training. In both cases an experienced player can compare the actual situation to a broader range of previous experiences. If he recognises a situation, he will know which actions he has to execute.
Riley's work will be looked at in more detail. He decomposes the process into several steps:

- 1. Abstracting features and extracting them
- 2. Building so-called "adversary classes" based on the extracted features
- 3. Matching the observed events into the adversary classes
- 4. Changing the behavior based on the selected classification

Abstracting features

As mentioned above the coach receives the positions and velocities of all moveable objects at each time step. From this raw visual information the following features are extracted: the positions of the ball, opponent players, opponent pass lanes (that is, each position that the pass goes through is counted), opponent dribbling and opponent shooting.
Obviously it is not possible to deduce the opponent's strategy from one time-step, so the features are cumulated. The field is discretized into small squares, which together form a data structure called RectGrid. For each feature there is one RectGrid, and each observed position is counted in the corresponding small square. This will lead to a characteristic distribution throughout the RectGrids. Note that for example the temporal information is lost, that is, after counting a pass it is impossible to decide towards which of the two directions it went, because the starting and the end point are treated identically.
This is one of the shortcomings of this representation for opponent models. It cannot differentiate between a tactic which uses passes from the middle to the wings from a tactic which makes passes from the wings into the middle. Another disadvantage is that the opponent model holds only positional information. There is no way to explicitly represent pass-chains or any other

mutual interaction between team-mates. Coordination between players is an essential part of strategies, so there has to be a way to represent this.

Building adversary classes

According to Riley, the duration of a game is too short to create a model of the opponent. Thus, it is more promising to create the adversary classes beforehand and then to just match the observation into those. The claim is that a small set of models should be sufficient to represent a wide range of opponent strategies and decision mechanisms accurately enough [36]. I think that this is the major insight and the whole approach is dependent on the question if the models will indeed generalize on new teams. This thesis also relies on this claim, because it also tries to classify opponents into pre-defined adversary classes. It will be evaluated empirically, if the models cover indeed new teams.
An adversary class as Riley uses it is a set of some RectGrids, called target configuration, e. g. a set that contains a RectGrid for the opponent passing positions could denote an opponent that passes from the center to the wings. These classes were constructed manually.

Selecting the adversary class

A first approach was to build RectGrids for the target configurations during the game and then measuring the similarity to the appropriate target configurations in the opponent models [33]. Unfortunately, experiments revealed that the identification of the correct model was only a little above random guessing. It was suggested that this was due to the fact that the similarities between the different models were very small and that for example the opponent player positions depend highly on the ball movement and marked players, so that these vary too much between different games.
The next technique was to use decision trees to disassociate the target configurations with a specific adversary class [34]. The idea behind this is that for distinguishing between classes C_1 and C_3 the best feature might be the similarity to a target configuration from class C_3. But for distinguishing between C_1 and C_2, C_3 is of no use. This feature selection can be done by decision trees. While this approach performed around 40 % of accuracy and thus is not yet useful for incorporating into a team, it performed one order of magnitude better than guessing, which was around 3 % due to the large

number of adversary classes.
Another approach is to use a naive Bayesian classifier [36]. Initially there is a probabilistic distribution over the models. Each observation is used to recalculate the distribution by

$$P(M_i|\alpha) = \frac{P(\alpha|M_i)P(M_i)}{P(\alpha)}$$

where M_i are the models and α is the observation.

I think that the use of decision trees in order to disassociate the target configurations is a real innovation. Unfortunately its performance of 40 % accuracy makes it of no or rather low use in an application compared to the effort of learning and building the decision trees. So in the approach which will be described in this paper, the more traditional Bayesian classifier will be used.

Adapting the behavior

The first of Riley's works skipped this part totally [33, 34]. Later research used the opponent models to predict probabilities for player positions [35]. Upon these predictions, a plan for setplays, i. e. standard situations like free-kicks, is created so that the probability of an opponent player intercepting a pass chain is minimized.
Although it was pointed out that the team benefitted from these adaptive setplays [35] in some way, to my knowledge there exist no significant experiments that show these benefits in terms of score differences.

Chapter 5

Feature-based opponent modelling

A commander who approaches a battle with a picture before him of how such and such a fight went on such and such occasion, will find, two minutes after the forces have joined, that something has gone awry. Then his picture is destroyed. He has nothing in reserve except another individual picture, and this also will not serve him for long. Or it may be that when his first pictured forecast is found to be inapplicable, he has so multifarious and pressing a collection of pictures that equally he is at loss what practical adjustment to make. Too great individuality of past reference may be very nearly as embarassing as no individuality of past reference at all. To serve adequately the demands of a constantly changing environment, we have not only to pick items out of their general setting, but we must know what parts of them may flow and alter without disturbing their general siginificance and functions.

F. C. Bartlett
REMEMBERING, 1967

This chapter introduces a representation for opponent models, called feature-based declarative opponent modelling (FBDOM). Since this is a general approach, several possible methods are discussed to build such models and to select the appropriate one based on observations. Additionally, methods are discussed to benefit from classifying the opponent's behavior. Several of these discussed methods are evaluated and compared in the following chapter.
It will also be pointed out which are the demands that an opponent model representation in complex domains like RoboCup should satisfy and how FBDOM handles these demands.

5.1 Features and feature-based declarative opponent models

This section introduces the technique of feature-based declarative opponent modelling.
The general idea is the same as in Riley's work described in section 4.2.6: Humans try to match the playing style of the opponent to previously encountered ones and then select the strategy that performed best against the previous opponent. But considering Riley's approach it is surely cognitively inadequate to assume that human players store every single observation to compare the cumulated observations with previously acquired models that again contain all observations (although defendants of the exemplar theory might think differently[26]. Although Riley abstracts features and removes the temporal ordering, e. g. in the target configuration `player positions`, it is unlikely that humans consider every position that they observed (apart from the fact that the real world is not discretized into time-steps).
The method proposed in this paper relies on the assumption that humans identify certain properties of the opponent. These properties are hence-forth called features. Conceptually, features have to describe typical parts of the behavior. For example, in RoboCup [12], a simulated soccer domain, one feature might be that the forwards of FCPortugal [31] pass to each other in front of the goal until the goalie makes a mistake and leaves enough free space for the forwards to score. Such behavior can be formalized as mapping from situations to actions, just like in CLang mentioned in section 4.1.3.
As a rough example, the aforementioned passing behavior can be imagined as a mapping from situations to actions in the following way. Note that two

features are needed, because there are two different situations to consider.

- `If the goalies blocks the space between ball and goal, pass to the other forward.`
- `If the goalie is too far away, shoot the ball into the goal.`

In another domain like MODSAF [8], a military air-combat simulator, a feature might look like this:

- `If the aircraft comes into radar range, it turns onto collision course.`

These features can then be compared to concrete observations. If in the first example a player dribbles with the ball instead of passing, whenever the way to the goal is blocked, then his behavior is not covered by the above features. Analogously, an observation from the MODSAF simulator which shows that an aircraft indeed goes onto collision course instead of circling around, matches the described feature.

There are two aspects of features that need to be satisfied in order for a human to deem them typical for a strategy. These are distinctness and stableness (cf. section 3.3). So, regarding distinctness these features have to be encountered in the particular strategy, but much less often in other strategies. Regarding stableness they have to appear with a high probability in the particular strategy.
In order to clarify the use of the notion `feature` in this paper and to make a distinction between the general use of it in every day's language, its definition is as follows:

Definition 5.1.1 *Let S be the set of all (eventually incomplete) situation descriptions. Let H be the set of all possible actions. Let $H*$ be the powerset of H. A* `feature` $< s, A >, s \in S, A \in H*$ *is an externally observable mapping from a situation description s to a set of actions A.*

An incomplete situation description focusses on some part of the situation, for example only on the position of the ball. This helps to keep the features concise, and reflects the fact that an offender is likely to ignore the position of its own defenders if he stands in front of the opponent goal.
The actions in H are assumed to carry also information on which of the agents executes it. So, a pass from player 6 to player 11 is different from a

pass from player 10 to 11. Note that in Riley's approach these actions would be identical if the players are in the same region/RectGrid.

The definition of a feature-based declarative opponent model is also straightforward:

Definition 5.1.2 *A* `feature-based declarative opponent model` ω *is a (unordered and thus declarative) set of features* $< s_i, A_i, p_i >, s_i \in S, A_i \in H*, p_i \in \mathbb{R}$*, where* p_i *denotes the probability of the feature to occur in the strategy which is specified by the model. Also the constraint* $s_i = s_j \wedge A_i = A_j \rightarrow p_i = p_j$ *has to be satisfied.*

The last constraint guarantees that there are no features which are equal but have different probabilities. It should be noted that there is no need to satisfy a constraint like $s_i = s_j \rightarrow A_i = A_j$. Instead, it is indeed reasonable to enter multiple features which share the same situation but contain different actions. This reflects the fact that agents have different (perhaps even indeterministic) options to execute in a situation.

5.2 An architecture for a FBDOM system

This section outlines an architecture for an agent applying FBDOM.
As shown in figure 5.1, the basis are the opponent models. These can contain tatical descriptions on different levels. For example, some might specify the complete strategy of a certain team, others might only specify the marking-assignment of the left-wing defender.
In order to detect the features in the opponent models, the observations which come in as raw sensor data have to be processed by the action- and situation-detectors. They try to match the observations into the features of the models. The information if and how the observations match will be passed on to the counting component. It handles for example partial mismatches, and implements one of the possible counting methods, e. g. a Bayesian classifier. The opponent model with the best value will be chosen and then a knowledge base will decide which counter-strategy is applicable. Just like the opponent models, the counter-strategies can vary in size. They can either contain full team strategies or just specifications how a forward should shoot.
This architectures allows usage as an online- or offline-method. It depends on the domain though, if the method needs only so much data as is provided

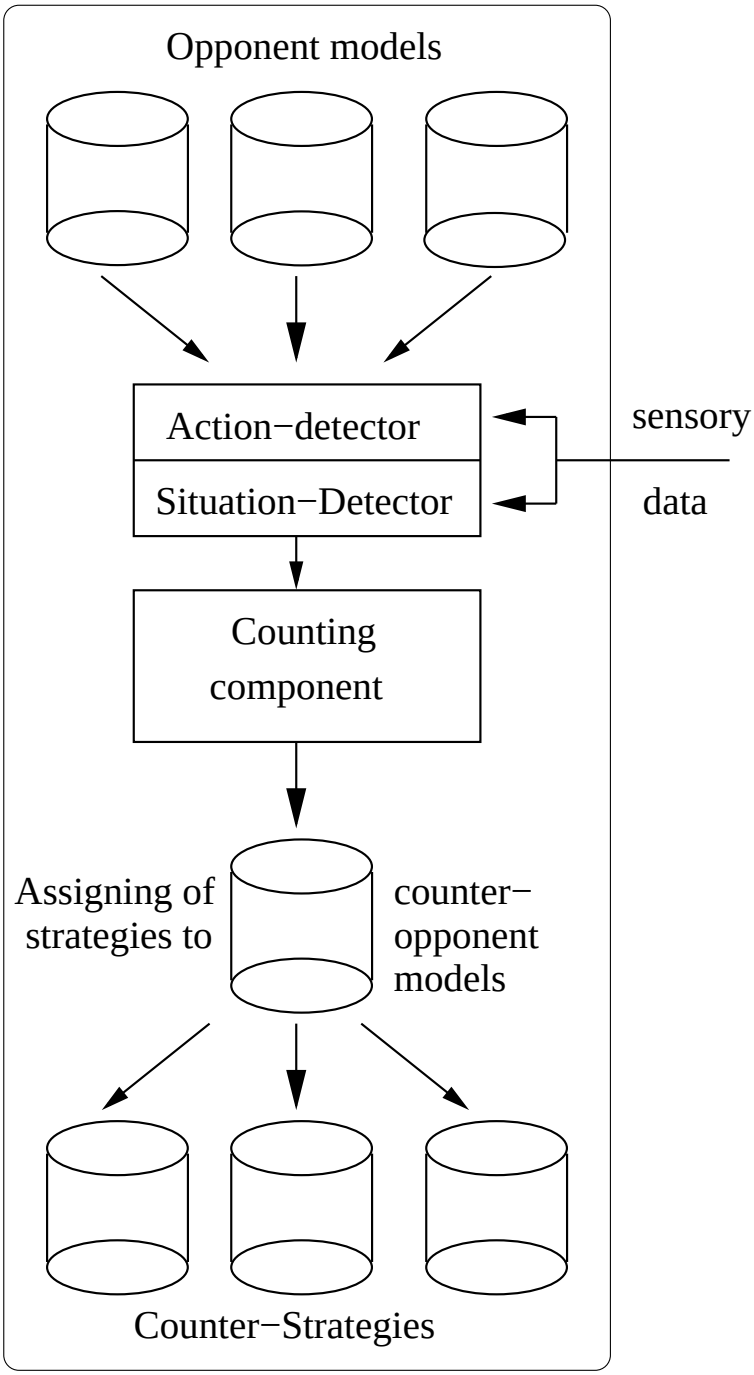


Figure 5.1: Components of a FBDOM system.

during the encounter with other agents, or if the histories or logfiles of previous encounters have to be analyzed in order to select the counter-strategy beforehand.

5.3 Demands on an opponent model representation

There are several methods to represent opponent models. One method is to use a domain-specific ad-hoc model [36]. Other approaches were derived from more general representation techniques like Bayesian networks [17] or STEAM [19] (see section 3.5 for an outline of these approaches). Yet, to our knowledge there is no systematic framework for evaluating which representation fits the specific needs.
In this section several general demands are proposed that a representation mechanism for opponent models has to satisfy in order to be useful in complex domains like RoboCup. The focus of this paper is on opponent models that are used to classify strategies, rather than to predict the opponent's behavior. Nevertheless, both kinds of usage demand the following aspects. It should be pointed out that these demands can be employed on all scientific work and thus should be seen as ideal goals that cannot be satisfied in general.
First the demands are proposed, then their interactions will be looked at.

- Granularity:
  It should be possible to express opponent models on different levels of granularity. This way a hierarchy of classes can be specified, with top-level classes denoting very broad ranges of strategies, and the leaves denoting very specific strategies. Building models is facilitated by this, because the first step is to create very general classes, and then if need arises, these can be refined further by decomposing or spezialising them.

  Concrete example from soccer: A very general class would consist of those teams that play with two forwards. These teams can then be partitioned by more detailed models. Some teams might position their forwards in the center of the field, others on the wings. The latter can be differentiated even more, by describing if the ball is dribbled towards the goal, or passed to the other forward.

- Modularity:
  As mentioned before, in realistic domains strategies are a cumulation of different tactics or substrategies. For example, in sports different forms of offensive strategies can be coupled with different forms of defensive strategies. A strategy has to denote in which situations which tactic should be applied. So, a representation mechanism should provide the ability to specify these modules separately, thus facilitating combinations of them. It has to provide means to specify the interactions of modules: If for example a module for the midfielders and the forwards exists, there also has to be a way to describe how the ball is transferred from the midfielders to the forwards.

  By dividing opponent models into modules, the identification can be done separately. In RoboCup for example this will be helpful if during a period of the game a team is forced into defense and provides no observations for offense at all. Monolithic models would either fail to recognize the complete strategy or would have to wait until enough offensive observations can be gathered.

  Again a concrete example from soccer: One module can describe the behavior of the opponent when it does the kick-off by specifiying the player positions and the pass chains. Another module can model the offensive tactics in front of the goal. These modules have a minimum of interaction between them, whereas modules for the goalie and the defenders will interact much more with each other.

- Need for data/Scalability:
  An opponent model should be easy to recognize even with very sparse data. This shortens the identification time and thus maximizes the time in which the team can try to benefit from the classification. An any-time-method would be optimal, i. e. a method that does not have to rely on a fixed amount of data or a fixed observation window, but which can return good (preliminary) classifications at any point of the observation process with a high confidence.

- Externalization:
  As in section 3.2, the models should make as little assumptions about the internal decision processes of the modelled agents as possible. This is due to the fact that there is no way to observe these internal processes or its internal states. If for example the model relies on the assumption

that the modelled agent uses a deterministic finite automaton, although it uses a more expressive method, then the model might be unapt to predict the agent's behavior.

A way to guarantee assumption-free models is to use only externally accessible observations for the recognizing parts.

- Redundancy:
  The redundancy in opponent models should also be as little as possible, so that they can be stored compactly and modified easily without risk of causing inconsistencies. Apart from that, the representation should allow to infer more observable events out of the represented actions without the need to specify them explicitly. For example, the fact that a player receives a pass at a certain position also implicates that the player had to run there in the first place.

- Completeness:
  The notion of completeness used here is, that all relevant aspects and kinds of a strategy should be expressable in the model. If in the MODSAF domain a representation formalism misses a maneuver like collision-course, then certain classes of opponents cannot be modelled and the modelling agent will not be able to take appropriate actions if it encounters such a maneuver.

  Another interpretation of completeness is that the complete behavior should be described in the model. Yet, the argument put forward and evaluated in this thesis is that it is actually sufficient to focus on the most typical behavior parts in order to identify a strategy.

- Consistency:
  The representation should not permit the existence of inconsistencies. If avoiding them is not possible, the representation should provide means to identify inconsistencies.

  For example such an inconsistency would be the prediction of two different positions for an agent, or inherently unsatisfiable preconditions for an action.

These demands interact with each other. For example, if the granularity needs to be more fine-grained in order to focus on details, the need for data is higher. This leads to the idea, that in the beginning of an observation

session, the opponent will be classified into a rather general class. As more data is acquired, the classification can become more detailed.
Also, modularity is a form of granularity: with large granularity, an opponent model might contain a complete strategy. But with smaller granularity, strategies can be splitted into more detailed modules, so that each opponent model only describes certain parts of the behavior. Completeness is influenced by granularity, too. The more fine-grained a model is, the more details need to be expressed, which puts higher demands on the representation formalism. In the worst case this can lead to conflicts, if the demanded granularity cannot be satisfied by the representation mechanism. E.g. if the granularity needs to model the sensomotorical skill of the pilot in MODSAF, but the representation mechanism does not provide such detailled concepts, then it is impossible to satisfy both granularity and completeness. So there is a trade-off.
Based on these thoughts, it is also obvious, that the need for data rises with the degree of modularity: If identification works separately, for each module data is needed. Though, on another note, modularity can also help to reduce the need for data. If for example the opponent never has a chance to become offensive, then there is no need to gather data about its offensive moves, because the agent does not need to choose a defensive strategy.
Another interaction between demands is about the need for data and redundancy. If a representation does not need to explicitly express each action, then the need for data is reduced. In a sports domain this can be illustrated like this: If the model contains a feature which describes that an agent is likely to shoot on the goal from a certain position, then it suffices to observe if the agent is waiting for the ball there, or dribbling there. It is not necessary to gather data about his movement to that position. So a low redundancy also reduces the need for data.
Obviously it is impossible to satisfy all of these demands. One has to find trade-offs after examining the concrete requirements of the domain or application.

Now it will be examined how FBDOM handles these demands.

- Granularity:
  This demand is satisfied to the degree that the underlying language which specifies the situation-action mapping provides different levels of granularity. One such language is described in section 6.1.

- Modularity:
  Since each feature contains its own situation in which it is applicable, several features can be put together to form a module. For example it is possible to create a module which specifies how the defenders mark the forwards, by combining several features which describe the marking assignment of different defenders in different situations.

  If the situation descriptions are crisp and not fuzzy, then the modules can be differentiated very clearly.

  To a certain extent this also depends on the expressivity of the underlying language. It has to provide sufficient means in order to create modular features.

- Need for data / scalability:
  Each observation adds to the information available for selecting the proper class. There are no fixed observation windows or fixed amounts of data that are needed. The claim is that feature-recognition is an anytime-method. This will be experimentally evaluated in section 6.7.2.

- Externalization:
  Again this depends on the underlying language. This demand is satisfied to the degree that it contains only externally observable concepts.

- Redundancy:
  Redundancy is a problem that has to be coped with in this approach. Basically it is possible to include highly redundant features into the set. So this has to be taken care of when constructing the models.

- Completeness:
  The underlying language is also the keypoint for completeness. It has to provide concepts for all necessary actions and situations.

- Consistency:
  Within a feature-based declarative opponent model the only inconsistencies that can arise are of the type that several different actions are mapped to the same situation. But if such different mappings exist, then this reflects only the fact that the opponent uses a variety of tactical moves.

  If for example in soccer the opponent uses the right wing as often as the left wing, then there can be two features in the model with the situation

that the ball is in the middle of the field. One of these features maps a right-sweep on the situation and the other one a left-sweep.

Regarding the externalization, obviously the uncertainty of Intille's unobservable internal states [17] can be avoided, because feature-based declarative opponent modelling makes no assumption about the internal state of agents. Interestingly, the uncertainty still remains, but was shifted into the ambiguity of actions. While in Intille's belief network it is for example uncertain whether a specific player is ready to catch the ball, in FBDOM the uncertainty is whether a shot is a pass or a goal shot. The problems are equivalent, if one defines the choice of an action (e. g. pass or goal shot) to be dependent on the player's internal state.
Approaches like plan-recognition (see section 3.4) or Intille's framework (section 3.5) focus very much on the hierarchical or temporal structure of actions, respectively. Plan-recognition is of restricted use in a dynamic and fast-changing environment, because the plans will be aborted or changed rapidly. The only plans which plan further into the future than five cycles are to the author's knowledge restricted to setplays [35], standard situations in breaks like free- or corner-kicks. The focus of feature-based declarative opponent modelling is different from the hierarchical temporal one. It uses similar techniques to infer actions, but it rather copes with the question in which situations these actions will be executed, and which actions are used by the opponent at all. Based on these observations, the opponent is classified into an opponent class.
Temporal structures can only be used if a certain amount of constant "moves" or "plays" exist like in American football. Other domains, like RoboCup, are very dynamic, and not all of the tactics can be represented as constant "plays", and those that can, may be aborted and changed very rapidly. So in the general framework of feature-based opponent modelling, there is no temporal structuring. But if it is needed in a domain, it can be incorporated into the underlying language.

5.4 Building models

Having accurate models for different opponent classes is crucial. The models of feature-based declarative opponent modelling are more complicated to build than those of Riley's approach. While Riley's models can be built by

just counting positional observations of the modelled teams [33], the models here need distinct and stable features. That is, building such a model is basically about finding features that describe the behavior of opponents that belong into the class. Several methods for achieving this can be applied and are described rather generally. That building models can indeed be done is shown in the following chapter.

First of all this can be done by domain experts who have an understanding about the different strategies that can be used. The descriptions of "plays" in Intille's framework [17] for example are very likely to be entered by domain experts, too. Also the plan libraries used in plan recognition are created manually.
This method might also lead to models at different granularity and specificity levels. So a domain expert might want to create rather general models first which describe the overall different strategies. In soccer strategy books often very general strategies like the "English" or "Brasilian" playing style are compared [5]. After that he might want to construct more specific models in order to differentiate between teams that e. g. adopted the "English" playing style.

Another approach is to acquire the features by automatic clustering of observations from opponents. In the general case there have to be two kinds of clustering. First of all the different opponent classes need to be clustered. Yet, it might be the case that the opponent classes are already identified and only need to be filled with features, if for example models shall be created for single teams. In these cases there still remains the task of clustering the observations. In a soccer domain clustering could be done by self-organizing maps [24]. For each class observations will be processed. The input samples would be vectors with he following components: two components per player specifying its 2D-coordinates, two components for the coordinates of the ball, and a series of components for the actions of each opponent player. Per player only one action segment can be active, the others will be filled with zeros. A pass to a player will take one component, a pass to a point two components, marking a player one component, dribbling to a point one component and so on.
If there are repeated situations in which similar actions are executed, these will form a cluster. From the vector representation, the feature representation can be extracted. These are stable, because they were frequent enough to

form a cluster. In order to also guarantee distinctness, cross-tests have to be run with the other classes in order to check if the features only exist in the particular class. Features that are not distinct enough will be removed.
It has already been shown that self-organizing maps can be used to learn clusters of agent behavior [48]. For example, clusters differentiated between regions of passing and dribbling.

The second clustering task might also be replaced by concept-learning methods. Rules that predict when and where players pass have been learned already [37]. Such rules can be used as features, after additional cross-tests in order to estimate their probability. Similar methods seem promising to learn rules even for goal shots or marking behavior. The idea is to collect observations in which an action was executed or was avoided. These observations form the positive and negative samples which are needed for concept-learning. The learning algorithm outputs generalizations which can be used as the situation-component of features for the given action.

Genetic algorithms might also be applied to find appropriate features. The population would consist of features and the fitness function would be a combination of the stableness and distinctness of a feature.

5.5 Selecting the model

Like in any abductive approach the model that provides the best explanation for the observed events is to be found. Or in other words, the model that is most likely to match the observations should be the appropriate opponent class. It has been noted, that selecting the model which best describes the opponent, is a challenge [36]. While in plan-recognition the appropriate plan follows logically from the action hierarchy (cf. section 3.4), the model selection here is numerical instead of logical. Other than in Riley's approach, no auxiliary model is built from the observations, so that no similarity metric is necessary. Instead, the observations are matched to features, and the number of matches determines the appropriate opponent model. But the way how these matches should be counted is not straight-forward. Intille can use the straight-forward classic inference techniques of belief networks [17]. In feature-based approaches the Bayesian theorem is such a classic approach. E.g. in image recognition for each model the probability is updated after an

observation using Bayes' theorem [45]. It has to be evaluated if it also leads to good results in feature-based declarative opponent modelling. Several other methods for counting and calculating observations are possible, and later experiments showed, that the parameters for counting must be carefully chosen because they heavily influence the accuracy.
Feature-based models do not explain every observation, but just a subset. This has to be taken into account when determining the best model. Several methods are possible and need to be evaluated.
First it must be noted that in MAS there exist pairs of actions that are mutually exclusive, or mutex. This depends on the domain, for example in RoboCup an agent cannot turn and kick at the same time. While in simple one-agent-domains all pairs of action may be mutex, in MAS most of the action pairs that involve two distinct agents are not mutex.

Definition 5.5.1 *Two actions a and a' are* `mutually exclusive` *or short* `mutex` *to each other, if they cannot be executed simultaneously. For two mutex actions a and a' we write $a \div a'$.*

Definition 5.5.2 *An action a is* `mutex` *to a set of actions A iff*

$$\exists a' : a' \in A \ \wedge \ a \div a'$$

We write $a \div A$.

Definition 5.5.3 *A situation s is* `subsumed` *by situation s', iff s is true whenever s' is true. We write $s \subset s'$.*

Definition 5.5.4 *Let S be the set of all situation descriptions. Let H be the set of all possible actions. Let $H*$ be the powerset of H, consisting of all possible subsets of H. An* `observation` *is a tupel $< s, A >, s \in S, A \in H*$, where s is the denotation of an observed situation and A is the set of observed actions of all agents in that situation.*

An observation $< s, A >$ can be evaluated into the following primitive results wrt. a model ω. For simplicity's sake, in the following list of match-types the probability parameter of the features in the model is abandoned, because it has no influence on the matching process. Because two actions are either mutex or not, this list is complete:

- `No-match`:

$$\neg\ \exists s' :\ s' \subset s\ \wedge\ < s', A' > \in \omega$$

  No situation of any feature in the model matches the observation.

- `Partial Match`:

$$\exists s', a :\ s' \subset s\ \wedge\ a \in A\ \wedge\ a \in A' \wedge\ < s', A' > \in \omega$$

  At least one feature in the model matches both the situation and at least one action of the observation.

- `Full Match`:

$$\forall a :\ a \in A\ \rightarrow \exists s' : s' \subset s\ \wedge\ a \in A' \wedge\ < s', A' > \in \omega$$

  One or more features in the model match the situation of the observation, and all of the observed actions are in the action sets of these features.

- `Partial Mismatch`:

$$\exists s', a :\ s' \subset s\ \wedge\ a \in A \wedge\ < s', A' > \in \omega\ \wedge\ a \div A'$$

  There is a feature in the model so that it matches the observed situation but includes an action that is mutex to the observed action.

- `Full Mismatch`:

$$\forall a :\ a \in A \rightarrow\ \exists s' :\ s' \subset s \wedge\ < s', A' > \in \omega\ \wedge\ a \div A'$$

  For every observed action there is a feature in the model which predicts a mutex action.

Several of these primitive results can be combined. For example a partial match can be encountered simultaneously with any kind of mismatch, if two features exist that share the same situation but have contradictory actions. This has to be taken into account when the number of matches is compared between the opponent models.

The simplest way to select a model based on the observations is to just count the observations that result in full matches. Yet, this method misses some

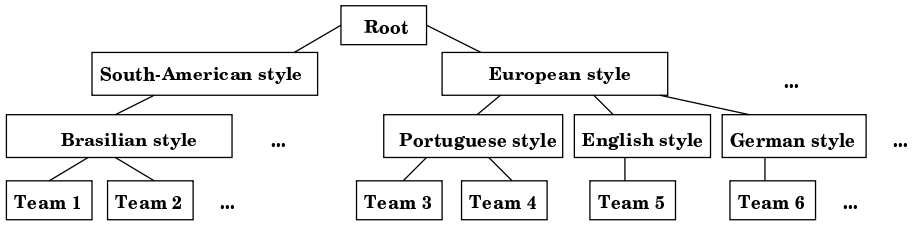


Figure 5.2: Tree of models.

things. Since there is no fixed number of features that are in a model, the selection might be biased towards models with large sets of features, because the probability of matching should be higher for these.
To overcome this bias, one can consider the fraction of full matches and full mismatches. Still, there can be observations that are full matches and full mismatches. Consider the following example.
Suppose the model has the two features

- `If the ball is controlled by an opponent in the left half, player 5 attacks the ballowner and player 4 marks opponent 10.`
- `If opponent 10 is in its own half, player 4 returns to its home position.`

Now, an observation

`The ball is controlled by an opponent in the left half and opponent 10 is in its own half, player 5 attacks the ballowner and player 4 marks opponent 10.`

is a full match with the first feature and a full mismatch with the second.
An intuitive way to overcome this problem is the following: For each observation it is tested if it can be explained by the model. That is, if it is a full match, the eventual full mismatches are ignored. Also, if several features are full matches for the observation, only one is counted. Analogously, if several features are full mismatches.
If there are a lot of models, checking each of them might prove infeasible. For such cases this paper proposes the use of hierarchical models. That is, the models are nodes of a tree with different granularity levels.
This way, not all models have to be checked. Instead, according to the matching top-level nodes, only the according subtrees are checked.

At a first glance it might look as if the models cannot be arranged in a tree, because they can share features. This could result in cross-inheritances. But feature-based declarative opponent models do not contain all features that describe the complete behavior. Instead, only those features are contained, which are distinctive enough to identify the model. So, sharing features between subtrees will be avoided and the tree-structure can be guaranteed under optimal circumstances. It has to be evaluated if this can also be applied in the worst case, i. e. in realistic applications. The main question here is, which thresholds for the distinctness value can be used in reality.

Of course also the Bayesian approach is applicable. Each feature in the model is assigned a prior probability. This can either be set manually by a domain expert or empirically acquired by automated observation sessions (as part of building of the model itself). A prior probability distribution over the models is needed as well. Again this can be done by a domain expert or during the building of the model. It might also work to assign a uniform distribution to the features and models.
Then, each observation α changes the probability distribution of the models M_i according to Bayes' theorem:

$$P(M_i|\alpha) = \frac{P(\alpha|M_i)\ P(M_i)}{P(\alpha)}$$

$P(M_i|\alpha)$ is the new posterior for the model, $P(\alpha|M_i)$ is the probability of the matching feature in the model (which is part of the feature-tupel), $P(M_i)$ is the assigned probability of the model. This leaves $P(\alpha)$, the probability of the observation. In discrete domains this can be set by domain experts or empirically acquired by running experiments. However, in non-discrete domains this is not feasible. In such a case the following procedure can be applied: Since $P(\alpha)$ does not depend on the model, it is constant and does not need to be computed explicitly. Instead, it can be ignored, and the calculated $P(M_i|\alpha)$ are then normalized to a probability distribution.

5.6 Benefitting from the knowledge about the opponent

Basically opponent models can be used to predict the opponent's behavior. Yet, as mentioned earlier, often it is not obvious how to compute the best

response to an opponent's actions. Also, predicting something does not mean that one can actually influence the outcoming. For example, in RoboCup the prediction that the opponent will execute a successful shot on the goal provides no advantage if no player is able to stop the opponent from doing so.
The other way to use opponent models is to classify the opponent and change the strategy based on this classification. So the agents need an entry for each model which contains the appropriate counter-strategy. These entries may be acquired by previous encounters with the opponent, as proposed by Riley. But since feature-based declarative opponent modelling offers the opportunity to decompose the opponent model into several modules, it might be easier for the agent's designer or a domain expert to set these entries manually. Say, if a module was recognized that states that the opponent often scores from the center of the penalty area, the appropriate counter-strategy might be to position more defenders there. This is on a more abstract level than predicting the opponent action and calculating the best response, because this uses larger chunks of strategies.
The next chapter evaluates the techniques proposed in this chapter.

Chapter 6

Applying feature-based declarative opponent modelling to RoboCup

> The strongest arguments prove nothing so long as the conclusions are not verified by experience. Experimental science is the queen of sciences and the goal of all speculation.
>
> *Roger Bacon (1214?-94?), philosopher and scientist.*

This chapter provides a proof of concept for feature-based declarative opponent modelling by implementing it in the RoboCup domain, particularly the simulation league (see section 4.1 for a description). The advantages and shortcomings of this approach are highlighted. Additionally, several of the possible methods as discussed in the previous chapter are compared by running experiments.

This chapter is organized as follows: section 1 introduces a language for representing the features, section 2 describes how the opponent models were built. Section 3 and 4 outline how the situations and actions respectively are matched to observations. In section 5 several methods for selecting the opponent model are described and in section 6 the results of the experiments are reported.

6.1 Representing features for RoboCup

As shown in section 5.1 the language for mapping situations to actions is very important. Several of the proposed demands depend heavily on the language. This section describes a domain-specific language for RoboCup, called SFL. It is based on the standard coach language (see 4.1.3) and was extended for facilitating the specification of a team's behavior. The differences to CLang and the complete grammar can be found in [27]. SFL is preferred over Clang here, because it is more expressive.
To illustrate the languages, some examples are given. The first one specifies that the game is to be opened by a pass from player 11 to player 10:

```
1   (advice
2      (200
3         (playm ko_our)
4         (do our {11} (bto {10} {p}))
5      )
6   )
```

Line 1 denotes that the following rule is meant as a directive and not as an info-message, which only describes a behavior (cf. section 4.2.5). Line 2 states that the rule will be valid for 200 cycles. After that period it should not be executed anymore. Line 3 contains the condition of the rule, stating that the playmode must be "our kickoff". Finally, line 4 specifies the action which should be taken when the condition is true. Player 11 passes to player 10. *bto* is the ball-to-directive which contains parameters how the ball should be handled. Another parameter could be a region, and the set p which now only contains the directive to pass, can also contain directives to dribble, shoot or clear the ball.
SFL rules look very similar, but it was extended to allow flexible and concise specifications of a team's behavior. E. g. in CLang it is not possible to denote the closest player to the ball explicitly. So situation-specific symbols were introduced into SFL. This is important if one wants to describe opponent models, because teams exchange player positions and roles in order to save stamina. Constant player denotations would not be capable to describe tactics independent of such exchanges.
A rule which advises a player to intercept the ball if he is the closest, looks like:

```
1   (advice
2      (6000
3         (and
4            (bowner opp {0})
5            (playm play_on)
6         )
7         (do our {(closestPlayerToBall our)}
7a                        (interceptball 100))
8      )
9   )
```

Line 4 states the condition that any player of the opponent team controls the ball. In line 7 the use of the situation-specific symbol is demonstrated, stating that the closest player to the ball should intercept it with maximal speed.

While building the models it became obvious that SFL is even better suited for building opponent models than for implementing a team behavior, because of the possibility to express general rules and the fact that the situation and action concepts in SFL are externally observable. That is, in order to describe a behavior only the observable events have to be described. So the specific decision process of the team implementation can be ignored. While SFL cannot model all details on a low level like individual skills (e. g. kicking or dribbling), it is detailed enough to make distinction between teams on the tactical level. Regarding abstractness, even very general rules can be specified, e. g. the fact that each team intents to get the ball into the opponent's goal. This most general rule in soccer is formalized in SFL (which in this case is identical to CLang) below. So, the granularity and externalization demands are satisfied to a very large degree.

```
(advice
   (6000
      (true)
      (do our {0} (bto (rec (pt 52 -7) (pt 52.5 7)) {pdcs}))
   )
)
```

The above rule says that in all situations the team is trying to get the ball

by passing, dribbling, clearing or shooting into the rectangle which specifies the opponent goal.
SFL is also expressive regarding the situations. Situations can be described on different granularity levels, by using score differences, time, ball positions, player positions, ball ownership, etc. This makes it possible to create different modules for offense and defense, for player roles like forwards or midfielders, for the beginning of a game and the end-game, for tactics in the own and the opponent's half. This fulfills the modularity demand to a satisfying degree.
Since SFL does not contain primitive actions like turning and dashing, but more complex skills like passing and scoring, it is not required to specify the agent's actions explicitly. For example, the passing-concept in SFL covers the agent's need to approach the ball, turn into the correct direction, and kick the ball with the appropriate speed. Thus, also the "need for explicitness"-demand is satisfied.
Yet, the fact that primitive actions are not contained also means that SFL, just like CLang, is not complete. It neither possess interference methods nor iteration or recursion. The focus of both languages is to describe strategies and not complete behaviors. For example, a rule which specifies that the ball should be passed to a teammate does not describe with which speed the ball should be accelerated or if the pass should be avoided if an opponent player is too close. This is due to the fact, that CLang was created as a coaching language which cannot cope with all possibilities, because the coach can only send messages in certain intervalls. Also, SFL was created as a tool to specify the tactics of a team. The low-level skills are assumed to be hard-coded into the agent. However, since all of the other demands that rely on the language are satisfied by SFL, it is used for representing the opponent models in this chapter.

6.2 Building the models

The opponent models used in the experiments were created manually. The focus was on offensive strategies, where a situation was deemed as offensive when the opponent was in the team's half and had possession of the ball. This is still a large part of a team's strategy, but focussing on one part of the behavior facilitated building the models.
Models were built for several teams whose binaries were publicly available. This was chosen instead of using logfiles of competitions to ensure that the

teams were likely to use the same strategy in the different games. That is, the configuration files during all of the games were exactly the same as provided by the research groups. It should be noted that the opponent models are not intended to model different teams, but different strategies. The assumption is that a set of models can describe a wide range of teams [36]. With the same assumption, Riley states that his team benefitted from his approach [36]. Yet, he does not comment on measurable or quantitative values. The experiments in the course of this thesis will have to evaluate this assumption. Although the models were created for single teams, it is hoped that they are general enough to describe new teams. A trade-off between team-specific and general descriptions has to be found.
By watching each team play some games (in most cases one or two games were sufficient), striking offensive behaviors that happened several times were described in SFL. These SFL rules formed the features of the opponent model. In order to make sure that these features are stable and not specific reactions to the other teams, the games were played against changing teams.

Models were built for the following teams: ATTCMUUnited 2000, FCPortugal 2000, Virtual Werder W2000, Yow AI 2000, and AT Humbold 2000. The number of features in the models varies from 2 to 15 (refer to the appendix for some example models). This means that the models are no complete specification of the behavior, but reflect only typical tactics. So many observations will not be matched on features in the models. This is consistent with the idea that human players cannot identify the opponent's tactic from arbitrary situations. Many situations contain only few or no information about the underlying tactic. These situations will be ignored in FBDOM, as compared to the RectGrid approach (see section 4.2.6), where every single position is counted.
Additionally, the variation in the number of features has to be considered in the selection process. To ensure stableness and distinctness as mentioned in 3.3, a script was written that ran all features through cross-tests with all modelled teams. So each feature was tested on all teams and its team-specific number of matches was determined. The following cases are possible:

- The feature has less matches on its own team than on average on the other teams. In this case the feature was removed.
- The feature matches well not only on its own team, but also on a small number of other teams. The models of these teams were then extended

with the feature. While this on the first glance violates the distinctness aspect, it increases the distinctness between those models that include the feature and those that do not include it. It should be remembered that models can share features (see 5.5), if the granularity is high.

- In all other cases, the feature was left unchanged in the model.

The manually created models form the leaves of the afore-mentioned model-hierarchy-tree (section 5.5). It is not a trivial task to build the tree upon these models. Two general ways of creating parent-nodes exist. If complete features are shared between models, then the features of these models can be combined to create a super-model. Unfortunately, in the general case this can lead to cross-inheritances. If model A and B share a certain set S_1 of features, and model A and C share another set S_2 of features disjunct to S_1, then the tree-structure is violated. So, in general, it should be avoided to let models share features (just like the distinctness factor demands anyway). The other way to create super-nodes is to infer completely new features from the existing ones. Let there be two features $< c_1, a_1 >, < c_2, a_2 >$[1]. In some cases a new feature $< c_3, a_3 >$ can be formed, where $c_1 \wedge c_2 \rightarrow c_3$ and $a_1 \wedge a_2 \rightarrow a_3$. If one or more of these more general features can be formed, the original features can be replaced by it, so that a more general model is created which subsumes the former models.
An example from the RoboCup domain. Features 1 and 2 can be generalized into feature 3.

```
1) If the ball is five or less metres in front of the
   center-line, pass the ball to the left forward.
2) If the ball is five or less metres behind the center-line,
   pass the ball to the right forward.
3) If the ball is five or less metres within the center-line,
   pass the ball to one of the forwards.
```

Since the features in the models might describe totally different situations, it cannot be guaranteed that a parent-node and thus a tree-structure can be found in the general case. While a tree-structure is nice for efficiency aspects, the classification will also work without it.

[1]For simplicity's sake, it is assumed that the features contain single actions instead of sets. Features with sets of actions can easily be transferred into a list of features which contain only single actions.

6.3 Situation-matching

One part of matching the observations to features in the model is to match the situations. The condition part of the features has to be true for the observation. The implementation of this is based on the matcher-module of the Dirty-Dozen team, but since it was very specific to the player's world-model, it had to be rewritten completely, so that it could be used in an online-coach or logfile-analyser.
The condition concepts in SFL are not fuzzy, so a strict boolean evaluation of the `and`-, `or`- and `not`-combined conditions was applicable. For a detailed description of the matching process see [27].

6.4 Action detection

The other part of matching observations to features is to analyse the actions of the players. This algorithm had to be built from scratch, because it was not neccessary in the Dirty-Dozen team.
The first step was to operationalize the actions that can be expressed in SFL:

- Positioning in a region:
  Positioning in CLang and SFL means that a player either moves into the given region or if it already is there, positions itself somewhere within the region. Since the velocity of the players can be observed as a vector, the positioning action can be checked easily. If the vector of the observed player points to the target region or the player is already in the target region in the feature, this can be counted as a match. However, the real target position of the player might be anywhere along the vector, so movement is ambiguous to a certain degree. Of course it can be disambiguated by tracking the player over a certain time interval, but this misses the point that the player might change its intentions in the mean-time.

  To illustrate this more, assume that the observer tries to match the action of an agent at time t into the features. Assume further that the agent dribbles from the left wing into the direction of the center. Now it is ambiguous, if his target region is just a bit more to the center, exactly in the center, somewhere on the right wing or somwhere in between. The observer might be able to detect the target region by waiting some

more cycles until the dribbling is finished. But it still remains uncertain if the detected target region was the same that the agent selected with his information at time t or if he changed the target region since then because of new information.

For the experiments, an action like dribbling matches a feature if the target region of the feature is in the direction of the dribbling action.

- Marking a player:
  To mark a player means to position oneself on a line between the ball and the player. So detecting this action requires to check if a player aims to a position on this line, and if it already is there, remains on the line and moves according to the ball and the marked opponent. If its movement angle differs from the line's angle more than a certain threshold, it can be deemed as leaving the marking position. Again, marking detection is ambiguous, if the player's movement crosses the particular line by chance.

- Marking a region:
  Marking a region is more general than marking an opponent and requires the player to position itself between the opponent and the given region. This can be detected analogously to the player-marking action.

- Dribbling to a region:
  Dribbling is a series of dashes and kicks. A recognition algorithms has to check if the vector of the player and the vector of the ball are parallel and aimed at the given region. It also has to distinguish between the case of dribbling and the case of passing and following the ball. This can be checked by the power used to accelerate the ball. Dribbling uses very small amounts of power for kicking.

- Passing to a player:
  Identifying a pass is to check if the vector of the moving ball points to the given teammate. Additionally, if the pass is aimed ahead of the receiver's movement, it has to be checked if the receiver's and the ball's vectors intersect and that both will be at the intersection point at the same time.

- Passing to a region:
  Passing to a region is a generalization of passing to a player. It has to

be checked if the pass is targeted at one of the players in the region. So for each of the teammates in the region the above testing algorithm must be applied.

- Clearing the ball:
  Clearing the ball means to shoot the ball into a free area. The implemented algorithm checks if the kick is harder than a dribbling kick and cannot be interpreted as a pass.

- Scoring:
  The scoring identification test checks if the ball moves into the direction of the goal and has enough power to reach it.

Other definitions for detecting complex actions have been proposed before. In [43] dribbling and passing are defined as so-called genuine kick sequences. Dribbling is a series of kicks by the same player that are more than 0.5 seconds and more than 0.5 meters apart. In the case of passing, two players have to be involved. In [33] dribbling is defined as continuous possession of the ball while moving at least 3m. Passing is operationalized as two players having control of the ball within a time window of 50 cycles.
While these definitions work fine if one wants only to count the number of passes and dribblings, it is not sufficient, if one is interested in the tactics to which they belong. For this reason, in this thesis the actions had to be separated, e. g. into passes to players and regions. Additionally the target players and target regions have to be considered for tactical aspects. In feature-based opponent-modelling it is important to discriminate between passes back to the defenders or agressive passes to the forwards.
Some words about the observation intervals are needed: Passing and dribbling for example differ in the time span that a player executes them. Dribbling involves a series of consequent dashes and kicks, while passing in an extreme view is only one kick. So a feature that describes a dribbling action would match more observations than a pass feature. On the one hand, because of this the observations that match a pass feature should be weighted higher or the observations during the ball's movement should be counted as well. On the other hand, the passing player has no chance to change its decision after the kick, so from this point of view it is correct that the pass feature is only matched once. It is not clear which view is the correct one, so this is a point that needs experimental evaluation.

In general, whenever an observed action was ambiguous and one of its interpretations matched a feature, it was counted as a match. This is justified, because features are defined as the observable external behavior. Whatever the player's intention is, the effect in the observation is the same. From the analogy to human players it is also obvious that humans cannot explicitly observe the internal states of an opponent and their judgement is only affected by the observations.

6.5 Determining the best matching opponent model

As mentioned in section 5.5 several methods exist to select a model based on the observations. Some of these were implemented and evaluated.
Apart from the Bayes method, the counting methods were specified by three binary parameters:

- The first parameter (Once vs. All) determines how many matches will be counted for each situation. In case of "Once", the matching process aborts after the first successful match. In the other case several features may be matched in the same situation. "Once" also means that a match overrules a mismatch of another feature.

- The second parameter (Most vs. Ratio) triggers if the model that has the highest number of matches or the one that has the best match-mismatch ratio will be selected.

- The third parameter (Increasing vs. Normalized) specifies if the number of matches and mismatches will be divided by the number of features in the model. This way a normalization is done to overcome the variability in the number of features (cf. section 6.2).

Combining these parameters results in eight different selection methods which were evaluated.
Additionally a Bayes classifier was implemented and tested. The classic Bayesian update function was used:

$$P(M_i|\alpha) = \frac{P(\alpha|M_i)P(M_i)}{P(\alpha)}$$

where M_i are the models and α is the observation.
In order to calculate $P(M_i|\alpha)$ correctly, values for $P(\alpha)$ and $P(\alpha|M_i)$ had to be estimated. This was done empirically by analysing logfiles. For $P(\alpha)$ the number of matches in various logfiles was divided by the overall number of time-steps in all games. $P(\alpha|M_i)$ was estimated analogously, with the exception that for each value only games were used in which model M_i was employed. $P(\alpha|M_i)$ is the third component in the feature-tupels, so it had been obtained before.

6.6 Benefitting from the classification

Of course a classification of the opponent alone does not improve the team's performance. The knowledge about the opponent's behavior must be used. So for each model a counter-strategy has been created manually. The counter-strategies were built depending on the characteristics of the model.

To illustrate this, some examples are listed:
If the model specified a fixed formation, a counter-formation was used. I.e. in the defense the players pool around the forwards and offensive midfielders, and in the offense the forwards are located in the free spaces of the opponent's defenders.
If the positions of the forwards were variable, but the forwards kept their role throughout the game, then the defenders were assigned marking assignments.
If a model used fixed setplays (positions and/or pass chains), the counter-strategy incorporates marking assignments or positions for kick-offs etc.

A first series of experiments has to evaluate if the counter-strategies are significantly better than the unadapted baseline strategy. This needs to be done in order to ensure that the manually created counter-strategies are in fact appropriate counter-measures. Table 6.1 shows the goal means for the counter and baseline strategies. As table 6.1 shows, all counter-strategy means are significantly higher than the baseline games. The significance was determined with a one-tailed t-test with $\alpha = 0.9$. This α-value is not very strict, but should suffice, because these experiments are not supposed to be the basis of any theory. The number of games run for the teams varies. As many games as necessary to achieve significance were run. It should be noted, that the counter strategy often is only slightly better than the baseline.

Team	Co.-Strat. mean	N_1	Basel.-Strat. mean	N_2
Virtual Werder 00	-11.52	22	-16.70	22
ATHumboldt 00	-1.66	87	-2.04	87
Yow AI 00	-6.0	120	-6.91	120
ATTCMU 00	-3.70	22	-5.13	22
FC Portugal 00	-25.64	16	-36.82	16

Table 6.1: Goal mean for the baseline and the counter strategy against the modelled teams. N_1 and N_2 denote the number of games.

This is due to the fact, that the Dirty Dozen team, which was used to run the different strategies, has very weak low-level skills. Strategical changes will have a lesser impact on teams which have weak low-level skills, because adapted pass-rules or dribblings are likely to fail, no matter what the tactic is.
This experiment only shows that the counter-strategies play better against the opponent model than the baseline does. It does not cover the question, whether the counter-strategies also play better against other opponent models, for which they were not intended. While this can be easily tested by running cross-tests, the number of games necessary for significant results is too large to be feasible, because every team has to be paired with every other opponent model (and these games cannot be recycled for other experiments). So, some random tests have to suffice for now:
The counter-strategy which was intended for AT Humboldt was used to play against ATTCMU. Although it performed worse (goal mean -4.46) than the counter-strategy for ATTCMU, it also performed better than the baseline. It is very likely that this is the case for several more pairings. So this has to be taken into account when one examines the further experiments. Yet, this cannot be avoided, because building a strategy which performs well against one model, but bad against four other models is hard because of the many degrees of freedom. Because of this, special care has to be used when designing subsequent experiments.

The most important point is to evaluate if the classification and counter-strategies also work for new teams, i. e. teams for which no model was created. Experiments with new teams have to be run. It must be evaluated if the classification of new teams is stable and yields better counter-strategies than random selection of counter-strategies.

6.7 Experiments

6.7.1 Experiment design

This section will outline which hypotheses have to be tested and which questions have to be answered by the experiments. These considerations result in three experiment designs.
Conceptually the first two experiments should show that matching the behavior of an opponent team into one of several opponent models and selecting the corresponding counter-strategy increases the performance of one's own team. Also the different counting methods mentioned in section 6.5 should be compared. So a first step is to test the selection methods, which will also determine if the identification process is better than selecting randomly. To test this, games must be run, in which the modelled teams have to be correctly classified. This experiment is called experiment 1.

Experiment 1 will also be useful to find out about the following points:
As mentioned in 6.2 the models were built after watching one or two games. So it is possible that the features in the model describe the particular games (by singular events) and not the tactics. SFL and CLang are expressive enough to describe single time steps of a game, thereby representing single situations that might never occur in other games again. The experiments have to show that the models are general enough so that the identification works in many different games.
It might be possible that behavior is highly dependent on the specific opponent, i. e. certain behaviors occur only when playing against one specific opponent. The experiments have to test if the identification is constant over different opponents.

For practical reasons the number of necessary games should be as low as possible.
Based on the above points, the design of experiment 1 is as follows:
There are five opponent models OM_1 - OM_5, that were built based on observing games by team T_1 - T_5. Each team plays several games against all teams (including itself). The logfiles will be analyzed once for each counting method. There are nine counting methods: The Bayesian classifier and the eight combinations of the parameters described in section 6.5. If the classification is OM_i for team T_i, the identification will be counted as correct.

To fare better than random, more than 20 % of the identifications have to be correct.

In order to test if the team can benefit from the models even if it plays against a totally new opponent, experiment 2 has to be executed. It should be tested, if classifying a new opponent and then playing with the appropriate counter strategy yields better goal-differences than playing with the baseline strategy. The baseline team is the Dirty Dozen (DD) with the binaries and behavior specification as it competed at RoboCup 2001.
Several aspects have to be considered: It must be avoided that eventual performance boosts are only due to the fact that all counter-strategies are better than the baseline strategy. This is possible, because as mentioned in section 6.6 the counter-strategies sometimes play well against other opponent models, for which they were not built. If all counter-strategies were better than using the baseline, classifying the opponent would be almost useless, since randomly selecting from the strategies would fare better than the baseline.
Additionally, it can be assumed, that the five created opponent models do not cover all possible strategies which new teams might employ. So, depending on which new teams are selected, the results could be very good or very bad. To accomplish valid results, the new counter-strategies will not only be compared to the baseline, but also to randomly selected counter-strategies.
Another interesting point which is worth observing is the selection constancy of new teams. Will new teams always be classified into the same opponent model or will the classification vary? This is especially interesting, since there are no experimentally evaluated hints how similar the opponent models are.

The design for experiment 2 is as follows:
There are six new teams N_1 - N_6, the baseline team DD, the opponent models O_1 - O_5 and the counter strategies CS_1 - CS_5. For each new team N_j a baseline goal-difference will be found by running several games against DD. In order to use the logfiles effectively, these games will also be used to classify the new teams. Subsequently, according to each single classification OM_i, the new teams will play against the counter strategy CS_i. This will result in two sets of games per new team with exactly the same number of instances, the baseline games (set 1) and the counter strategy games (set 2).
Another set of games will be created, in which each new team will play against randomly selected counter strategies. It is assumed that all counter strategies are equally probable, so that the new teams play the same number

of games against all counter strategies. These games form set 3.
If the team benefits from classifying the opponents, the goal differences in set 2 must be better than in set 1 and 3.

In order to test the any-time performance of FBDOM, experiment 3 will evaluate the identification accuracy during different intervalls. The best selection parameter combination (Once, Most, Norm) will be tested on 20 games between the five modelled teams, yielding 40 test instances. The window in which observations are made will vary.

6.7.2 Experimental results

Experiment 1

20 games were run and recorded for experiment 1. In these games only teams competed for which an opponent model existed. This way, 40 test instances were created, namely two teams in each game. The team pairings were assigned randomly.
So each of the aforementioned nine counting methods was tested with 40 test cases. Apart from the Bayes classifier, each counting method was specified by three boolean parameters (see tables 6.2, 6.3, 6.4). The experiments revealed that the most important parameter was the normalization-flag. All methods with the positive normalization-flag fared better than those with the negative flag. So it is obvious that normalizing the matches depending on the number of features in the model is advisable. In each identification process there were five models to choose from, so classifiying randomly would perform correctly in only 20 %. It should be pointed out, that all counting methods performed better than random.
However, the Bayesian classifier performed worse than one would expect after reading the literature. It was only insignificantly better than doing a random classification. It is possible that the independence assumption among the different features, which is a prequisite for using Bayes' classifier, was violated. Certain features are more likely to follow each other than others. For example, a feature that specifies that a forward moves along the right wing will often be followed by a feature which describes that the forward shoots on the goal from the right side. Such correlations could explain the bad performance. Unfortunately, determining the correlation between features is out of the scope of this paper.

	Count matches: Once	**Count matches: All**
Win-value: Most	82.5 %	80 %
Win-value: Ratio	75 %	65 %

Table 6.2: Parameters for counting methods and their identification success. Normalization turned ON.

	Count matches: Once	**Count matches: All**
Win-value: Most	25 %	35 %
Win-value: Ratio	32.5 %	47.5 %

Table 6.3: Parameters for counting methods and their identification success. Normalization turned OFF.

Bayes
26 %

Table 6.4: The identification accuracy for Bayes' classifier.

On the other hand, one configuration was correct in 82.5 % of the test cases, and two other methods were correct in 80 % and 75 % respectively. These three methods perform significantly better than random.

Obviously using the proper counting methods is important in feature-based declarative opponent modelling. The range from an accuracy of 25 % to 82.5 % is huge and determines if the whole mechanism will be succesful or not.

This experiment also shows that these opponent models were general enough, although they were created by observing only one or two games. Features can obviously be used to generalize over different occasions. So it is possible to extract and represent features which describe a strategy that is constant throughout many different games. Additionally, it is proven that the behavior does not depend too heavily on the opponent. The identification accuracy is constant over the different opponents. This observation will be important when it comes to creating opponent models. It is not necessary to base the features on games against many different opponents. Obviously strategies reveal themselves independent of the opponent. Features are a proper way to represent such strategies.

Experiment 1a

After executing experiment 1 as planned, the realization that there are several counting methods with very good performance lead to the idea to combine the top three methods. Using more than one counting method costs only little additional computation time. The intention behind this was that deciding by a majority vote might even improve the excellent accuracy.
Yet, as it turned out, the combination of Once-Most-Norm, All-Most-Norm, and Once-Ratio-Norm did not perform better than Once-Most-Norm alone. This was due to the fact that all methods failed on the same test cases, selecting the same wrong model.
It is possible that some of the five teams for which opponent models were created are very similar to each other. In such a case it would be obvious that all counting methods have the same problems differentiating between such similar pairs. So the misidentifications of the top three methods were examined:
25 misclassifications were done by the three methods altogether. 14 of these were cases in which the model of ATTCMU2000 was selected for team FC-Portugal2000. A first assumption was that some rules in the model were not distinctive enough. But even before the experiments, tests for distinctness were run and two rules were removed from the ATTCMU2000-model because they were too common in the FCPortugal2000 team. From the remaining features none was more indistinct than the others. Instead a majority of the features had the tendency to be found in both teams. This is consistent with the intuitive observation of a human that the teams ATTCMU2000 and FCPortugal2000 are more similar to each other than to the other modelled teams.
With such a result, it is possible to create a super-class as mentioned in section 5.5 by collecting those features that can be found in both team behaviors. It looks like the tactics of ATTCMU2000 and FCPortugal have similarities. It remains to be evaluated, if misclassifications in such cases might lead to the same performance. Maybe both counter-strategies lead to better results than the baseline strategy.

Experiment 2

For experiment 2 each new team played several games against the baseline strategy of the Dirty Dozen team. Two of the new teams were the successors

Team	Exp. mean	N_1	Basel. mean	N_2	α
Cyberoos 2000	-4.694	172	-5.125	183	/
FC DrWeb 2001	-5.46	126	-6.89	128	0.05
Helli Respina 2001	-13.84	72	-18.19	83	0.05
Virtual Werder 01	-0.839	30	-1.375	31	0.05
Mainz Rolling Brains	-13.21	23	-21.08	23	0.025
FC Portugal 2001	-34.27	21	-44.58	23	0.005

Table 6.5: Goal mean and number of games of the new teams against the baseline and the classification. If the differences are significant in a one-tailed t-test, also α is depicted.

of teams for which opponent models were created, namely Virtual Werder 2001 and FC Portugal 2001. An examination of statistics revealed that teams in general changed their behavior and playing style from 1997 to 1998 [43]. Unfortunately, these experiments gathered global data and did not evaluate how much teams by the same research group changed over the years. So it was unclear, if the successor teams should be classified into the opponent model of their predecessor. However, these teams were selected for the experiments as if they were completely new.
Table 6.5 shows the mean of the baseline games (new teams vs. the baseline strategy) and the means of the games in which the selected counter-strategy was used. Note that in some cases the number of games differ between the two sets. This is due to the fact that some counter-strategy games had to be taken out of consideration because of crashing clients. The classification of the opponent and using the counter strategy yields significantly better goal differences against five of the six new teams. For now, it is assumed that there was no suitable opponent model for the unsuccesful case, considering, that five opponent models cannot be expected to handle all possible new teams.
The natural variance in the goal-difference against certain teams was rather great, so that many games had to be run in order to get significant results. The variance in the outcome is typical for noisy domains. Interestingly, in five of the six cases, the deviation was reduced in the counter-strategy games (see table 6.6). Obviously the outcome is more stable and easier to predict in these games.

The classification was stable, too. Table 6.7 shows how often which team was classified into which opponent model. It is not clear, whether the small num-

Team	Var. in counter-strat. games	Var. in basel.-games
Cyberoos 2000	5.97	7.33
FC DrWeb 2001	6.31	7.24
Helli Respina 2001	12.41	18.73
Virtual Werder 01	1.01	1.27
Mainz Rolling Brains	8.52	18.17
FC Portugal 2001	16.02	6.60

Table 6.6: Continuation of the previous table. Variances of the goal-differences in the two sets of games.

Team	ATTCMU	VW00	ATH	YowAI00	FCP00
Cyberoos 2000	0	0	180	0	3
FC DrWeb 2001	123	0	0	0	3
Helli Respina 2001	11	0	0	0	72
Virtual Werder 01	0	4	0	0	27
Mainz Rolling Brains 01	0	2	0	0	21
FC Portugal 2001	18	0	0	0	5

Table 6.7: Classification results.

ber of classifications that are different from the majority should be counted as misclassifications or whether they signal similarity between the strategies. It would be interesting to hear a domain expert's opinion about the strategic similarity between the used teams. In all rows there is one opponent model with a clear majority of classifications. The number of different classifications is in most cases insignificant or at least within the bounds of the classifications accuracy.

Obviously some opponent models were selected more often than others. In experiment 1 there was no tendency to some team in the misclassifications, so it is reasonable to assume that these unbalanced classifications are not an artifact of the method, but reflect the frequency of strategies. Interestingly, the models built for the three world-champions ATHumboldt ('97), ATTCMU ('98 and '99) and FCP ('00) are among those models that were selected frequently. This might be a hint that other teams imitate some of the behaviours of successful teams.

While the significant better goal-differences against the classified counter-

Team	random	N_1	classif.	N_2	α
Cyberoos 2000	-4.77	126	-4.69	172	/
FC DrWeb 2001	-5.50	168	-5.85	163	/
Helli Respina 2001	-16.68	173	-14.25	173	0.05
Virtual Werder 01	-1.352	54	-0.8197	61	0.025
Mainz Rolling Brains	-15.65	336	-14.11	343	0.1
FC Portugal 2001	-32.50	42	-33.96	45	/

Table 6.8: Goaldifference mean of games in which the strategy was selected randomly and of the games, in which the classified strategy was used. Some means of the classified strategies differ from the previous tables, because some more games had to be run in order to get significant results.

strategies are a necessary condition for proving that the method is successful, they are not a sufficient condition. It might still be the case that all counter-strategies are better in general than the baseline strategy (as the results in section 6.6 suggest). In the latter case, the classification would be obsolete, because any choice between the counter-strategies would yield better results than using the baseline strategy. So another test was needed, in which several games were run against the new teams in which the strategy was randomly selected. If the means of these games are less than the means in which the classified counter-strategy was used, the optimal strategy was used in the classification & selection-runs.
The outcome of this experiment was non-uniform (see table 6.8). In three cases there was no significant difference between the random selection and the selection based on classification. One of those teams had not yielded significant better results in the baseline-vs-counter-strategy experiments before, so this result was not surprising. At least in the three other cases, the selection based on classification performed significantly better than the random selection. This means that in these cases the most suitable opponent model and the corresponding counter-strategies were selected. It also means that the opponent-models and counter-strategies did indeed generalize over the new teams, making the FBDOM approach successful.
However, the three cases in which the approach was not better than random selection have to be discussed. One possible explanation is that the strategies of the involved teams are so similar, that the related counter-strategies perform similarly well. Interestingly, two of these three cases were classified as ATTCMU or FCPortugal most of the times (see table 6.7), which are very

Intervall length	accuracy
100	42.5 %
250	57.5 %
500	62.5 %
1000	62.5 %
3000	67.5 %
6000	82.5 %

Table 6.9: Accuracy of the selection mechanism for different lengths of the observation window.

similar to each other anyway, even for human observers. From this it can be concluded, that also the counter-strategies for ATTCMU and FCPortugal might perform similarly, which would contribute to the lack of significant difference. Another reason for the outcome that several counter-strategies performed similar might be that they are only different in the defensive parts, because the opponent models focussed on offensive behaviors. So, whenever the team was in a defense situation, there was no difference between any of the counter-strategies.
Anyway, in the three cases which did not achieve significant improvements, the counter-strategy that was selected by the classification was not better than the others. This might also be due to the fact that the five created opponent-model/counter-strategy pairs cannot be assumed to cover all existing teams. As of now there is no evidence how well the six new teams are covered by the opponent-models. Based on this last thought it is strong evidence for the quality of FBDOM, that three cases were nevertheless significantly better than the baselines.

Experiment 3

In order to test the any-time-behaviour of the classification, six observation lengths were tested. A complete RoboCup match lasts 6000 cycles. In order to illustrate how short an observation window of 100 cycles is, it should be noted that some teams wait about 150 cycles before they do free-kicks or kick-offs. The classification performs very well even for very short observation windows. After 100 cycles the classification is much better than random selection (which is 20%). After 250 cycles the classification is already correct in more than 50% of the cases. The accuracy gets better the more data is

acquired.
This means that the classification cannot only be done offline by analysing logfiles, but also online. This also renders the idea applicable to select rather general models in the beginning of the game, and select more detailed models when more data is acquired.

Chapter 7

Conclusion

A method for representing opponent models in multi-agent-systems was introduced and its performance was experimentally evaluated in the RoboCup domain. It was claimed that for classifying an opponent it is sufficient to focus on distinct and stable features instead of processing the complete behavior for all situations. The assumption was that a set of opponent models covers a great amount of existing opponent behaviors. The experiments showed that the identification accuracy was high for the modelled teams, so the claim can be supported that features are a well-suited method to describe opponent behaviors.
Regarding the coverage of new teams, the experiments were non-uniform, but hint in a promising direction. There are some methodological difficulties to measure the impact of counter-strategies. In a perfect experimentation setting, each counter-strategy would perform well against only one opponent model, and bad against all other models. Yet, this can only be achieved in restricted toy-domains or against manually created opponents, but not under realistic conditions with using real teams. Obviously in the experiments the five created opponent models were not enough to cover all new teams. In five of six cases, the selected counter-strategy performed better than the baseline, and three of these five cases were also better than random selection. More work is needed to verify that the cause for the unsuccessful cases was the similarity between the opponent models and the small number of models which cannot be expected to generalize over all new teams. Creating more elaborate models that also contain information about defensive situations or in-depth analysis of the existing offensive behaviors could be helpful for this further work.

However, features form compact opponent models which successfully generalized over several new teams, so that the related counter-strategies were effective against previously unknown opponents. This also revealed that tactics can be identified by certain typical features, which are independent of the opponent. Because of this, opponent models can easily be created by observing arbitrary agent-opponent pairs. The identified features are very likely to be encountered against different opponents.
FBDOM is not restricted to any specific domain. In MAS in which during time many new agents are entered, FBDOM may help to prepare agents to handle new opponents if these base their behaviour on previously encountered ones. Good performance against other agents can be expected to appear very fast, because experiments also revealed that good results will be achieved in short observation windows. This makes FBDOM an any-time-method. An interesting matter would be to contemplate the risk of misclassifications in short observation windows. Will switching to a counter-strategy with low probability be more dangerous than keeping the baseline? Unlike in the mentioned experiments, there might be counter-strategies that perform only well against their related opponent-model, but very bad against others. In such a case, misclassification would be dangerous and costly.

Although the experiments showed that the used opponent-models covered some of the new teams, the question of how many opponent models are necessary to cover the majority of all existing teams remains open. Not all potential aspects of FBDOM have been used yet to examine this question. Several other aspects need to be implemented and evaluated. The automatic extraction of features by clustering or inductive learning was outlined, which can help to create enough models to have enough knowledge to cope with the majority of new teams. At that time, it will certainly be necessary to create the mentioned model-hierarchy-trees, in order to handle the large amount of opponent models efficiently. While the modularization of tactics promises interesting results for improving the separability of opponent models, it can also help with handling these large amounts of opponent models, because observations need only be matched to certain modules and not all existing features.
The performance of the approach is still very dependent on the quality of the manually created opponent-models and counter-strategies. In order to increase the system's autonomy, their creation has to become automated. Because of this the next step is to extract features from games by clustering or

inductive logic programming approaches, and to arrange them into opponent models by calculating their frequencies in different teams. The automatic creation of these opponent modules leaves only the assigning of counter-strategy to the domain expert. This should facilitate his work and if the modularization of strategies is enforced, counter-strategies may be combined from already existing modules. This is where the future work is aimed at.

List of Figures

List of Tables

An example model for ATTCMU 2000
Comments begin with #
The lines of numbers are the a prioris and frequencys of the feature in the other models which are only used by the Bayesian classifier.

```
# Am rechten Rand bis neben den Strafraum dribbeln:
<0.00083121657064, 0.0041922227041, 0.00084214958682, 0.0030527922522,
0.0054739723143, 0.017421969577, 0.00021053739671, 0.0031580609506,
0.00047370914259, 0.0046318227275, 0.000052634349176, 0.002684351808>
(advice
  (9000
    (and
      (bpos (quad (pt 15 17) (pt 38 20) (pt 38 34) (pt 15 34)))
      (bowner our {0})
      (ppos our {(closestPlayerToBall our)} 1 11
  (arc (pt ball) 0 1.3 0 360))
    )
    (do our {(closestPlayerToBall our)}
    (bto (quad (pt 38.5 20.5) (pt 50 20.5)
     (pt 50 34) (pt 38.5 34)) {dc} ))
) )

# Am linken Rand bis neben den Strafraum dribbeln:
<0.00066600582368, 0.0044452016604, 0.00084214958682, 0.0021053739671,
0.004789725775, 0.025369756303, 0.00010526869835, 0.0066319279962,
0.0, 0.0015790304753, 0.00010526869835, 0.0025790831096>
(advice
  (9000
    (and
      (bpos (quad (pt 15 -34) (pt 38 -34) (pt 38 -20) (pt 15 -20)))
      (bowner our {0})
      (ppos our {(closestPlayerToBall our)} 1 11
   (arc (pt ball) 0 1.3 0 360))
    )
    (do our {(closestPlayerToBall our)}
(bto (quad (pt 38.5 -34) (pt 50 -34)
   (pt 50 -20.5) (pt 38.5 -20.5)) {dc} ))
```

```
) )

# In der linken Mitte zum Fluegel nach vorn passen oder forwarden
<0.00026846746381, 0.0029789562811, 0.00036844044423, 0.0019474709195,
0.0010000526343, 0.0072109058371, 0.000052634349176, 0.0032633296489,
0.00021053739671, 0.0028422548555, 0.000052634349176, 0.0030527922522>
(advice
  (9000
    (and
       (bpos (quad (pt 8 -20) (pt 20 -20) (pt 20 0) (pt 8 0)))
       (bowner our {0})
       (ppos our {(closestPlayerToBall our)} 1 11
    (arc (pt ball) 0 1.3 0 360))
     )
     (do our {(closestPlayerToBall our)}
  (bto (quad (pt 24 -34) (pt 52 -34)
   (pt 52 -19) (pt 24 -19) ) {pc}))
) )

# In der rechten Mitte zum Fluegel nach vorn passen oder forwarden
<0.00022716477707, 0.0036759391198, 0.00026317174588, 0.0025264487605,
0.0012105900311, 0.0096320858993, 0.00010526869835, 0.0052634349176,
0.00010526869835, 0.0015263961261, 0.00010526869835, 0.0034212326965>
(advice
  (9000
    (and
      (bpos (quad (pt 8 0) (pt 20 0) (pt 20 20) (pt 8 20)))
      (bowner our {0})
      (ppos our {(closestPlayerToBall our)} 1 11
   (arc (pt ball) 0 1.3 0 360))
    )
    (do our {(closestPlayerToBall our)}
(bto (quad (pt 24 19) (pt 52 19)
       (pt 52 34) (pt 24 34) ) {pc}))
) )

# Am linken Fluegel, unmotiviert nach aussen und vorn schiessen
#(ueberdeckt sich mit anderen Situationen):
```

```
<0.00016004791112, 0.0050699047973, 0.00036844044423, 0.0034212326965,
0.00094741828517, 0.022474867098, 0.0,0.0076319806306,0.00010526869835,
0.0019474709195, 0.00010526869835, 0.0044212853308>
(advice
  (9000
    (and
      (bpos (quad (pt 9 -34) (pt 35 -34) (pt 35 -18) (pt 9 -18)) )
      (bowner our {0})
      (ppos our {(closestPlayerToBall our)} 1 11
    (arc (pt ball) 0 1.3 0 360))
    )
    (do our {(closestPlayerToBall our)}
  (bto (quad (pt 37 -34) (pt 52.5 -34)
   (pt 52.5 -31) (pt 37 -31)) {dc} ))
) )

# Am rechten Fluegel, unmotiviert nach aussen und vorn schiessen
# (ueberdeckt sich mit anderen Situationen):
<0.0001703735828, 0.0054261404704, 0.00026317174588, 0.0031580609506,
0.000894783936, 0.024159166272, 0.00010526869835, 0.0037370387915,
0.000052634349176, 0.0061055845044, 0.0, 0.0035265013948>
(advice
  (9000
    (and
      (bpos (quad (pt 9 18) (pt 35 18) (pt 35 34) (pt 9 34)) )
      (bowner our {0})
      (ppos our {(closestPlayerToBall our)} 1 11
    (arc (pt ball) 0 1.3 0 360))
    )
    (do our {(closestPlayerToBall our)}
  (bto (quad (pt 37 31) (pt 52.5 31) (pt 52.5 34)
   (pt 37 34)) {dc} ))
) )
```

An example model for FCPortugal 2000

```
#
# 0: In der Mitte, kein Gegner oder Mate weiter vorn, also nach vorn
# dribbeln (sehr VAGE)
# "Weiter vorn" modellieren als Rechteck, das 5 m vor dem Ballbesitzer
# beginnt und 15 m hoeher und tiefer ist
<0.00002581, 0.00016004791112, 0.00010526869835, 0.0012105900311,
0.001789567872, 0.014632349071, 0.00052634349176, 0.0011579556819,
0.000052634349176, 0.0016842991736, 0.00015790304753, 0.0021053739671>
(advice
    (9000
        (and
            (bpos (quad (pt 12 -20)(pt 32 -20)(pt 32 20)(pt 12 20)))
            (bowner our {0})
            (ppos our {(closestPlayerToBall our)} 1 11
(arc (pt ball) 0 1.3 0 360))
            (ppos our {0} 0 0
                (quad
                    (plus (pt our {(closestPlayerToBall our)})
(pt 4 -15) )
                    (pt 52.5 -34)
                    (pt 52.5 34)
                    (plus (pt our {(closestPlayerToBall our)})
(pt 4 15) )
                )
            )
            (ppos opp {0} 1 1
                (quad
                    (plus (pt our {(closestPlayerToBall our)})
(pt 4 -15) )
                    (pt 52.5 -34)
                    (pt 52.5 34)
                    (plus (pt our {(closestPlayerToBall our)})
(pt 4 15) )
                )
            )
```

```
        )
        (do our {(closestPlayerToBall our)} (bto (quad (pt 52 -20)
(pt 52.5 -20) (pt 52.5 20) (pt 52 20)) {d}))
    )
)
# 1: Wenn im Torwartraum, dann Schuss (auch durch Gegner hindurch)
# mismatch 0.00037172418066
<0.00012390806022, 0.00037172418066, 0.000052634, 0.000052634,
0.000052634, 0.000052634, 0.0, 0.0, 0.00010526869835,
0.00036844044423, 0.000052634349176, 0.00047370914259>
(advice
    (9000
        (and
            (bpos (quad (pt 45 -10) (pt 52.5 -10) (pt 52.5 10)
(pt 45 10)))
            (bowner our {0})
            (ppos our {(closestPlayerToBall our)} 1 11
(arc (pt ball) 0 1.3 0 360))
        )
        (do our {(closestPlayerToBall our)} (bto (quad (pt 52 -7)
(pt 53 -7) (pt 53 7) (pt 51 7) ) {s} ))
    )
)
#
#
# 2: Allein seitlich links vor dem Tor, zwei Gegner nah, Querpass
# (nach rechts und nach vorn)
<0.00011874522438, 0.00044400388245, 0.000052634, 0.00010526869835,
0.00084214958682, 0.0021580083162,0.000052634349176,0.00031580609506,
0.00010526869835, 0.00042107479341,0.00010526869835,0.00021053739671>
(advice
    (9000
        (and
            (bpos (quad (pt 22 10) (pt 42 10) (pt 42 21)(pt 22 21)))
            (bowner our {0})
            (ppos our {(closestPlayerToBall our)} 1 11
(arc (pt ball) 0 1.3 0 360))
            (ppos opp {0} 2 2 (arc (pt ball) 1.5 6 0 360))
```

```
        )
        (do our {(closestPlayerToBall our)} (bto
            (quad
                (plus (pt ball) (pt 3 -32))
                (pt 52.5 -34)
                (plus (pt ball) (pt 32 -3))
                (plus (pt ball) (pt 1 -1))
            )
        {p}))
    )
)
#
# 3: Allein seitlich rechts vor dem Tor, zwei Gegner nah, Querpass
# (nach links und nach vorn):
<0.000005163, 0.00044400388245, 0.000052634, 0.00015790304753,
0.00052634349176, 0.002684351808, 0.0, 0.0, 0.000052634349176,
0.00015790304753, 0.00031580609506, 0.00057897784094>
(advice
    (9000
        (and
            (bpos (quad    (pt 22 -26) (pt 42 -26)(pt 42 -12)
(pt 22 -12)))
            (bowner our {0})
            (ppos our {(closestPlayerToBall our)} 1 11
(arc (pt ball) 0 1.3 0 360))
            (ppos opp {0} 2 2 (arc (pt ball) 0 6 0 360))
        )
        (do our {(closestPlayerToBall our)} (bto
            (quad
                (plus (pt ball) (pt 1 1))
                (plus (pt ball) (pt 32 3))
                (pt 52.5 34)
                (plus (pt ball) (pt 3 32))
            )
        {p}))
    )
)
# 4: Noch vor der linken Strafraumecke, Gegner vor ihm, dann clear
```

```
# links zur Grundlinie
<0.000015489, 0.00055242343514, 0.0, 0.00047370914259, 0.000052634,
0.0015263961261, 0.0, 0.0, 0.0, 0.00021053739671, 0.0,
0.00068424653929>
(advice
    (9000
        (and
            (bpos (quad (pt 29 -22) (pt 36.5 -22) (pt 36.5 -7.5)
(pt 29 -7.5 )))
            (bowner our {0})
            (ppos our {(closestPlayerToBall our)} 1 11
(arc (pt ball) 0 1.3 0 360))
            (ppos opp {0} 1 11
                (quad
                    (plus (pt ball) (pt 0 -5))
                    (plus (pt ball) (pt 5 -5))
                    (plus (pt ball) (pt 5 5))
                    (plus (pt ball) (pt 0 5))
                )
            )
        )
        (do our {(closestPlayerToBall our)}
            (bto
                (quad
                    (pt 46 -34)
                    (pt 52 -34)
                    (pt 52 -20)
                    (pt 46 -20)
                )
                {c}
            )
        )
    )
)
# 5: Noch vor der rechten Strafraumecke, Gegner vor ihm, dann clear
# rechts zur Grundlinie
<0.00021683910538, 0.0038256613593, 0.000052634, 0.00015790304753,
0.000052634, 0.002684351808, 0.0, 0.000052634349176, 0.0,
```

```
0.0010000526343, 0.0, 0.00047370914259>
(advice
    (9000
        (and
            (bpos (quad (pt 29 7.5 )(pt 36.5 7.5)(pt 36.5 22)
(pt 29 22)))
            (bowner our {0})
            (ppos our {(closestPlayerToBall our)} 1 11
(arc (pt ball) 0 1.3 0 360))
            (ppos opp {0} 1 11
                (quad
                    (plus (pt ball) (pt 0 -5))
                    (plus (pt ball) (pt 5 -5))
                    (plus (pt ball) (pt 5 5))
                    (plus (pt ball) (pt 0 5))
                )
            )
        )
        (do our {(closestPlayerToBall our)}
            (bto
                (quad
                    (pt 46 20)
                    (pt 52 20)
                    (pt 52 34)
                    (pt 46 34)
                )
                {c}
            )
        )
    )
)
#
# 6: Ball im linken Korridor entlang der Strafraumlinie,Pass entlang
# der Linie nach rechts
<0.0002426532846, 0.0032319352374, 0.00010526869835, 0.0026317174588,
0.0015790304753, 0.021527448813, 0.0, 0.00052634349176,
0.000052634349176, 0.00063161219012, 0.00015790304753,
0.0031580609506>
```

```
(advice
    (9000
        (and
            (bpos (quad (pt 31 -21) (pt 41 -21) (pt 41 -1)
(pt 31 -1)))
            (bowner our {0})
            (ppos our {(closestPlayerToBall our)} 1 11
(arc (pt ball) 0 1.3 0 360))
        )
        (do our {(closestPlayerToBall our)} (bto (quad (pt 31 1)
(pt 41 1) (pt 41 21) (pt 31 21)) {p} ))
    )
)
# 7: Ball im rechten Korridor entlang der Strafraumlinie, Pass
# entlang der Linie nach links
<0.00023232761291, 0.00064019164447, 0.000052634, 0.0021053739671,
0.0016842991736, 0.016264013895, 0.0, 0.00021053739671,
0.00052634349176, 0.0024738144113, 0.00021053739671,
0.0023159113638>
(advice
    (9000
        (and
            (bpos (quad (pt 31 1) (pt 41 1) (pt 41 21)
(pt 31 21)))
            (bowner our {0})
            (ppos our {(closestPlayerToBall our)} 1 11
(arc (pt ball) 0 1.3 0 360))
        )
        (do our {(closestPlayerToBall our)} (bto (quad (pt 31 -21)
(pt 41 -21) (pt 41 -1) (pt 31 -1)) {p}))
    )
)
#
# 8: Im Strafraum, zwischen den Torpfosten, kein Mate weiter vorn,
# ab 1.5m niemand zwischen Ball und Tor:
<0.00065051731615, 0.0075583916734, 0.000052634, 0.00015790304753,
0.0016316648245, 0.0019474709195, 0.0, 0.00021053739671,
0.00015790304753, 0.00073688088847, 0.000052634349176,
```

```
0.00094741828517 >
(advice
    (9000
        (and
            (bpos (quad (pt 36 -7) (pt 52.5 -7) (pt 52.5 7)
(pt 36 7)))
            (ppos our {0} 0 0
                (quad
                    (plus (mult (pt ball) (pt 1 0)) (pt 1.5 -20))
                    (pt 52.5 -20)
                    (pt 52.5 20)
                    (plus (mult (pt ball) (pt 1 0)) (pt 1.5 20))
                )
            )
            (ppos opp {0} 0 0
                (quad
                    (plus (pt ball) (pt 1.5 -2))
                    (plus (mult (pt ball) (pt 0 1)) (pt 52.5 -2))
                    (plus (mult (pt ball) (pt 0 1)) (pt 52.5 2))
                    (plus (pt ball) (pt 1.5 2))
                )
            )
            (bowner our {0})
            (ppos our {(closestPlayerToBall our)} 1 11
(arc (pt ball) 0 1.3 0 360))
        )
        (do our {(closestPlayerToBall our)} (bto (quad (pt 52 -7)
(pt 52.5 -7) (pt 52.5 7) (pt 52 7)) {s}  ))
    )
)
# 9: 4 Sturmspitzen in der Mitte. Gegner nah, dann zu anderer
# Sturmspitze passen
<0.00031493298639, 0.003211283894, 0.00036844044423, 0.0037370387915,
0.0040528448866, 0.037791462709, 0.000052634349176, 0.010000526343,
0.0, 0.00021053739671, 0.00021053739671, 0.0034738670456>
(advice
    (9000
        (and
```

```
            (bpos (quad (pt 16 -34) (pt 36 -34) (pt 36 34)
(pt 16 34)))
            (ppos our {X} 4 7
                (quad
                    (plus (pt ball) (pt -6 -68))
                    (plus (pt ball) (pt 6 -68))
                    (plus (pt ball) (pt 6 68))
                    (plus (pt ball) (pt -6 68))
                )
            )
            (bowner our {0})
            (ppos our {(closestPlayerToBall our)} 1 11
(arc (pt ball) 0 1.3 0 360))
        )
        (do our {(closestPlayerToBall our)} (bto {X}))
    )
)
# 10: Am rechten Fluegel, kein Mate weiter vorn, in die Region
# neben dem Strafraum dribbeln:
<0.00027363029965, 0.003572682403, 0.0, 0.00021053739671,
0.0025264487605, 0.017790410022, 0.00010526869835,
0.0020001052687, 0.00010526869835,
0.0025790831096, 0.000052634349176, 0.0017369335228>
(advice
    (9000
        (and
            (bpos (quad (pt 10 19) (pt 37 19) (pt 37 34)
(pt 10 34)))
            (ppos our {0} 0 0
                (quad
                    (plus (mult (pt ball) (pt 1 0)) (pt 1.5 0))
                    (pt 52.5 0)
                    (pt 52.5 34)
                    (plus (mult (pt ball) (pt 1 0)) (pt 1.5 34))
                )
            )
            (bowner our {0})
            (ppos our {(closestPlayerToBall our)} 1 11
```

```
(arc (pt ball) 0 1.3 0 360))
        )
        (do our {(closestPlayerToBall our)} (bto (quad (pt 38 20)
(pt 52.5 20) (pt 52.5 34) (pt 38 34)) {d} ))
    )
)
#
# 11: Am linken Fluegel, kein Mate weiter vorn, in die Region neben
# dem Strafraum dribbeln:
<0.00012390806022, 0.0028860252359, 0.00021053739671, 0.000894783936,
0.0022106426654, 0.025632928049, 0.0, 0.00057897784094, 0.0,
0.0011053213327, 0.0, 0.0020527396179>
(advice
    (9000
        (and
            (bpos (quad (pt 10 -34)(pt 37 -34)(pt 37 -19)(pt 10 -19)))
            (ppos our {0} 0 0
                (quad
                    (plus (mult (pt ball) (pt 1 0)) (pt 1.5 -34))
                    (pt 52.5 -34)
                    (pt 52.5 0)
                    (plus (mult (pt ball) (pt 1 0)) (pt 1.5 0))
                )
            )
            (bowner our {0})
            (ppos our {(closestPlayerToBall our)} 1 11
(arc (pt ball) 0 1.3 0 360))
        )
        (do our {(closestPlayerToBall our)} (bto (quad  (pt 38 -34)
(pt 52.5 -34) (pt 52.5 -20)(pt 38 -20) ) {d} ))
    )
)
#
# 12: Wenn an der rechten Seite des Strafraums (innen oder aussen),
# dann zu Mate im Bereich des Strafraums passen
<0.00012390806022, 0.0028860252359, 0.00010526869835, 0.0048423601242,
0.00084214958682, 0.0082635928207, 0.0, 0.0, 0.00010526869835,
0.001789567872, 0.0, 0.00057897784094>
```

```
(advice
    (9000
        (and
            (bpos (quad (pt 35 17) (pt 52.5 17) (pt 52.5 34)
(pt 35 34)))
            (ppos our {X} 1 11 (quad (pt 33 -18) (pt 52.5 -18)
(pt 52.5 18) (pt 33 18)))
            (bowner our {0})
            (ppos our {(closestPlayerToBall our)} 1 11
(arc (pt ball) 0 1.3 0 360))
        )
        (do our {(closestPlayerToBall our)} (bto {X}) )
    )
)
# 13: Wenn an der linken Seite des Strafraums (innen oder aussen),
# dann zu Mate im Bereich des Strafraums passen
<0.0034023088202, 0.008193420482, 0.00010526869835, 0.0048423601242,
0.00084214958682, 0.0082635928207, 0.0, 0, 0.00010526869835,
0.001789567872, 0.0, 0.00057897784094>
(advice
    (9000
        (and
            (bpos (quad (pt 35 17)(pt 52.5 17)(pt 52.5 34)(pt 35 34)))
            (ppos our {X} 1 11 (quad (pt 33 -18) (pt 52.5 -18)
(pt 52.5 18) (pt 33 18)))
            (bowner our {0})
            (ppos our {(closestPlayerToBall our)} 1 11
(arc (pt ball) 0 1.3 0 360))
        )
        (do our {(closestPlayerToBall our)} (bto {X}) )
    )
)
#
# 14: Von ATTCMU2000.sfl hierherverschoben:
<0.00009293, 0.0017863412015, 0.003000157903, 0.0055792410127,
0.016737723038, 0.019106268751, 0.0020527396179, 0.009263645455,
0.0014737617769, 0.0063687562503, 0.0029475235539, 0.0091583767567>
(advice
```

```
    (9000
        (and
            (bpos (quad (pt 8 -20) (pt 28 -20) (pt 28 20) (pt 8 20)))
            (bowner our {0})
            (ppos our {(closestPlayerToBall our)} 1 11
(arc (pt ball) 0 1.3 0 360))
        )
        (do our {(closestPlayerToBall our)} (bto
            (reg
                (quad (pt 8 20) (pt 28 20) (pt 28 34) (pt 8 34))
# querschuss, relativ zum Ball:
                (quad
                    (plus (pt ball) (pt -10 -20))
                    (plus (pt ball) (pt 10 -20))
                    (plus (pt ball) (pt 10 -5))
                    (plus (pt ball) (pt -10 -5))
                )
                (quad
                    (plus (pt ball) (pt -10 5))
                    (plus (pt ball) (pt 10 5))
                    (plus (pt ball) (pt 10 20))
                    (plus (pt ball) (pt -10 20))
                )
                (quad (pt 8 -34) (pt 28 -34) (pt 28 -20) (pt 8 -20))
            )
            {c})
        )
    )
)
```

Bibliography

[1] Omid Aladini, Bahador Nooraei, and Siavash Rahbar. The Helli Resoina Team and Coach Description. In Andreas Birk, Silvia Coradeschi, and Satoshi Tadokoro, editors, *RoboCup-2001: Robot Soccer World Cup V.* Springer, Berlin, 2002. (to appear).

[2] J. Anderson, C. Boyle, A. Corbett, and M. Lewis. Cognitive modeling and intelligent tutoring. In *Proceedings of the First International Conference on Artificial Intelligence and Law*, pages 7–49, 1990.

[3] J. Austin. *How to do things with words.* Clarendon Press, Oxford, UK, 1962.

[4] H. Berliner. Search and knowledge. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 975–979, 1977.

[5] Christoph Biermann and Ulrich Fuchs. *Der Ball ist rund, damit das Spiel die Richtung aendern kann - Wie moderner Fussball funktioniert.* Kiepenheuer & Witsch, Koeln, 1999.

[6] R. Bolles and R. Cain. Recognizing and locating partially visible objects: the local feature-focus method. *International Journal of Robotic Research*, 1(3):57–82, 1982.

[7] Sean Buttinger, Marco Diedrich, Leonhard Hennig, Angelika Hoenemann, Philipp Huegelmeyer, Andreas Nie, Andres Pegam, Collin Rogowski, Claus Rollinger, Timo Steffens, and Wilfried Teiken. The Dirty Dozen Team and Coach Description. In Andreas Birk, Silvia Coradeschi, and Satoshi Tadokoro, editors, *RoboCup-2001: Robot Soccer World Cup V.* Springer, Berlin, 2002. (to appear).

[8] R. B. Calder, J. E. Smith, A.J. Courtemarche, J. M. F. Mar, and A. Z. Ceranowicz. Modsaf behavior simulation and control. In *Proceedings of the Second Conference on Computer Generated Forces and Behavioral Representation, STRICOM-DMSO*, July 1993.

[9] David Carmel and Shaul Markovitch. Incorporating opponent models into adversary search. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI)*, Portland, Oregon, 1996. AAAI Press.

[10] David Carmel and Shaul Markovitch. Learning models of intelligent agents. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI)*, Portland, Oregon, 1996. AAAI Press.

[11] E. Charniak and P. McDermott. *Introduction to Articial Intelligence.* Addison Wesley, Menlo Park, California, 1985.

[12] Mao Chen, Ehsan Foroughi, Fredrik Heintz, ZhanXiang Huang, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Itsuki Noda, Oliver Obst, Patrick Riley, Timo Steffens, Yi Wang, and Xiang Yin. *Soccerserver Manual v7.* The RoboCup Federation, 2001. http://sf.net/projects/sserver.

[13] Christian Druecker, Christian Duddeck, Sebastian Huebner, Holger Neumann, Esko Schmidt, Ubbo Visser, and Hans-Georg Weland. Virtual werder: Using the online-coach to change team formations. Technical report, TZI - Center for Computing Technologies, University of Bremen, 2000.

[14] Editor Gerhard Weiss. *Multiagent Systems - A Modern Approach to Distributed Artificial Intelligence.* The MIT Press, Cambridge, Massachusetts, 1999.

[15] Robert P. Goldman, Christopher W. Geib, and Christopher A. Miller. A new model of plan recognition. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, pages 245–254, 1999.

[16] Marcus J. Huber, Edmund H. Durfee, and Michael P. Wellman. The automated mapping of plans for plan recognition. In Ramon Lopez de Mantaras and David Poole, editors, *Proceedings of the 10th Conference on*

Uncertainty in Artificial Intelligence, pages 344–351, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers.

[17] Stephen S. Intille and Aaron F. Bobick. A framework for recognizing multi-agent action from visual evidence. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 518–525, Orlando, Florida, 1999. AAAI Press.

[18] P. Jansen. Problematic positions and speculative play. In T. A. Marsland and J. Schaeffer, editors, *Computers, Chess and Cognition*, pages 169–182. Springer, New York, 1990.

[19] Gal Kaminka, D. V. Pynadath, and Milind Tambe. Monitoring deployed agent teams. In *Proceedings of the Fifth International Conference on Autonomous Agents (Agents-2001)*, Montreal, Canada, 2001.

[20] A. Kautz and J. F. Allen. Generalized plan recognition. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, pages 32–37, Menlo Park, CA, 1986. AAAI Press.

[21] H. Kitano, M. Tambe, P. Stone, M. Veloso, S. Coradeschi, E. Osawa, H. Matsubara, I. Noda, and M. Asada. The robocup synthetic agent challenge. In *International Joint Conference on Artificial Intelligence (IJCAI97)*, 1997.

[22] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. RoboCup: The robot world cup initiative. In W. Lewis Johnson and Barbara Hayes-Roth, editors, *Proceedings of the First International Conference on Autonomous Agents (Agents'97)*, pages 340–347, New York, 5–8, 1997. ACM Press.

[23] R. Klinkenberg and I. Renz. Adaptive information filtering: Learning in the presence of concept drifts. In *Learning for Text Categorization*, pages 33–40. AAAI Press, Menlo Park, California, 1998.

[24] Teuvo Kohonen. *Self-organizing maps*. Springer, New York, 1995.

[25] John E. Laird, Allen Newell, and Paul S. Rosenbloom. SOAR: An architecture for general intelligence. *Artificial Intelligence*, 33(1):1–64, 1987.

[26] D. Medin, M. Altom, and T. Murphy. Given versus induced category representations: Use of prototype and exemplar information in classification. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 10:333–352, 1984.

[27] Andreas G. Nie, Angelika Honemann, Andres Pegam, Collin Rogowski, Leonhard Hennig, Marco Diedrich, Philipp Hugelmeyer, Sean Buttinger, and Timo Steffens. the osnabrueck robocup agents project. Technical report, Institute of Cognitive Science, Osnabrueck, 2001.

[28] Christos H. Papadimitriou and John N. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

[29] Alex Quilici, Qiang Yang, and Steven Woods. Applying plan recognition algorithms to program understanding. *Automated Software Engineering: An International Journal*, 5(3):347–372, July 1998.

[30] Anand S. Rao. Means-end plan recognition : Towards a theory of reactive recognition. In Pietro Torasso, Jon Doyle, and Erik Sandewall, editors, *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning*, pages 497–508, Bonn, Germany, 1994.

[31] Luis Paulo Reis and Nuno Lau. FC Portugal Team Description: Robocup 2000 Simulation League Champion. In Peter Stone, Tucker Balch, and Gerhard Kraetzschmar, editors, *RoboCup-2000: Robot Soccer World Cup IV*. Springer, Berlin, 2001.

[32] Luis Paulo Reis and Nuno Lau. Coach unilang - a standard language for coaching a (robo)soccer team. In Andreas Birk, Silvia Coradeshi, and Satoshi Tadokoro, editors, *RoboCup-2001: Robot Soccer World Cup V*. Springer, Berlin, 2002. (to appear).

[33] Patrick Riley. Classifying adversarial behaviors in a dynamic, inaccessible, multi-agent environment. Technical Report CMU-CS-99-175, Carnegie Mellon University, 1999.

[34] Patrick Riley and Manuela Veloso. On behavior classification in adversarial environments. In Lynne E. Parker, George Bekey, and Jacob

Barhen, editors, *Distributed Autonomous Robotic Systems 4*, pages 371–380. Springer-Verlag, 2000.

[35] Patrick Riley and Manuela Veloso. Planning for distributed execution through use of probabilistic opponent models. In *Proceedings of the IJCAI-2001 Workshop PRO-2: Planning under Uncertainty and Incomplete Information*, 2001.

[36] Patrick Riley and Manuela Veloso. Recognizing probabilistic opponent movement models. In A. Birk, S. Coradeschi, and S. Tadokoro, editors, *RoboCup-2001: The Fifth RoboCup Competitions and Conferences*. Springer Verlag, Berlin, 2002. (extended abstract).

[37] Patrick Riley, Manuela Veloso, and Gal Kaminka. Towards any-team coaching in adversarial domains. In Onn Shehory, Thomas R. Ioerger, Julita Vassileva, and John Yen, editors, *Proceedings of the First International Joint Conference on Autonomous Agents and Multi-Agent Sytemsi (AAMAS)*, Bologna, Italy, 2002. (to appear).

[38] Claude E. Shannon. Programming a computer for playing chess. *Philosophical Magazine*, 41:256–275, 1950.

[39] Peter Stone, Patrick Riley, and Manuela M. Veloso. Defining and using ideal teammate and opponent agent models. In *Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI)*, pages 1040–1045. AAAI Press/MIT Press, 2000.

[40] D. Suryadi and P. Gmytrasiewicz. Learning models of other agents using influence diagrams. In *Proceedings of the Internationale Conference on User Modeling*, pages 223–232, Banf, California, 1999.

[41] M. Tambe and P. S. Rosenbloom. Event tracking in a dynamic multi-agent environment. *Computational Intelligence*, 12(3), 1995.

[42] Milind Tambe and Paul S. Rosenbloom. Architectures for agents that track other agents in multi-agent worlds. In Michael Wooldridge, Jörg P. Müller, and Milind Tambe, editors, *Proceedings on the IJCAI Workshop on Intelligent Agents II : Agent Theories, Architectures, and Languages*, pages 156–170, Heidelberg, Germany, 1996. Springer.

[43] Kumiko Tanaka-Ishii, Ian Frank, Itsuki Noda, and Hitoshi Matsubara. A statistical perspective on the robocup simulator league: Progress and prospects. In *RoboCup-99: Robot Soccer World Cup III*, pages 114–127, Berlin, 1999. Springer Verlag.

[44] Jose M. Vidal and Edmund H. Durfee. Learning nested agent models in an information economy. *Journal of Experimental and Theoretical Artificial Intelligence*, 10(3):291–308, 1998.

[45] Paul Viola. Complex feature recognition: A bayesian approach for learning to recognize objects. Technical Report AIM-1591, AI Lab, Massachusetts Institute of Technology, 11, 1996.

[46] Ubbo Visser and Hans-Georg Weland. Using online learning to analyze the opponents behavior. In Gal A. Kaminka, Pedro U. Lima, and Raul Rojas, editors, *RoboCup-2002: Robot Soccer World Cup VI*. Springer, Berlin, 2003. (to appear).

[47] Geoffrey I. Webb and Mark Kuzmycz. Feature based modelling: A methodology for producing coherent, consistent, dynamically changing models of agent's competencies. *User Modeling and User-Adapted Interaction*, 5(2):117–150, 1996.

[48] M. Wunstel, D. Polani, T. Uthmann, and J. Perl. Behavior classification with self-organizing maps. In *The Fourth International Workshop on RoboCup*, pages 179–188, 2000.

Index

Erklärung
Hiermit versichere ich, dass ich die vorliegende Arbeit selbst verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Timo Steffens