Agent programming in Dribble: from beliefs to goals with plans

Birna van Riemsdijk
birna@myrealbox.com

Institute of Information and Computing Sciences
Utrecht University, The Netherlands

Supervisors:
prof. dr. J.-J. Ch. Meyer
prof. dr. W. van der Hoek

28th June 2002

Contents

Abstract

To support the practical development of intelligent agents, several programming languages have been introduced that incorporate concepts from agent logics: the language 3APL for example incorporates beliefs and plans, whereas the language GOAL implements the concepts of beliefs and goals. The language 3APL offers the means for *creating and modifying plans* during the execution of the agent. GOAL agents on the other hand do not have planning features, but they do offer the possibility to use goals to select actions.

My contribution now is the definition of the language Dribble, in which these features of both languages are combined. The language Dribble thus incorporates beliefs and goals as well as planning features. The idea is, that a Dribble agent should be able to select a plan to reach a goal from where it is at a certain point in time. In order to do that, the agent has beliefs, goals and rules to select plans and to create and modify plans.

The language Dribble has a formally defined operational semantics, defined by means of a transition system. On top of this semantics, a dynamic logic is constructed that can be used to specify and verify properties of Dribble agents. Furthermore, the correspondence between the logic and the operational semantics is proven.

Chapter 1

Introduction

The term agent has been used for a wide variety of applications. There is no real agreement on the definition of what an agent is. Often however [12], an agent is considered to be a computer system that displays a certain degree of *autonomy*. An agent should be able to act without the direct intervention of humans and should have control over its own actions. Furthermore, an intelligent agent should respond to changes in its environment (*reactive behaviour*). Moreover, it should not only act in response to its environment, but it should also be able to exhibit *pro-active* (goal-directed) behaviour and take the initiative where appropriate.

In research on agents, besides architectures, the areas of agent theories and agent programming languages are distinguished. Theories concern descriptions of (the behaviour of) agents. Agents are often described using logic [1, 10, 11]. Concepts that are commonly incorporated in such logics are for instance knowledge, beliefs, desires, intentions, commitments, goals and plans. To support the practical development of intelligent agents, several programming languages have been introduced that incorporate concepts from agent logics: the language 3APL [4] for example incorporates beliefs and plans, whereas the language GOAL [5] implements the concepts of beliefs and goals. The language 3APL offers the means for *creating and modifying plans* during the execution of the agent. GOAL agents on the other hand do not have planning features, but they do offer the possibility to *use goals* to select actions.

Beliefs describe the situation in which the agent is whereas goals describe the situation in which the agent wants to be. A goal is thus a *declarative* concept (also called goal-to-be). Plans are sequences of actions and are the means to get from a belief state to a goal state.

My contribution now is the definition of the language Dribble, in which the features of the languages GOAL and 3APL are combined. The language Dribble thus incorporates beliefs and goals as well as planning features. The idea is, that a Dribble agent should be able to select a plan to reach a goal from where it is at a certain point in time. In order to do that, the agent has beliefs, goals and rules to select plans and to create and modify plans. Using these beliefs, goals and rules, the agent can be programmed to exhibit autonomous, reactive and pro-active behaviour.

In chapter 2, the language Dribble is introduced and the meaning of constructs in the language is explained informally. The language has a formally defined operational semantics, which is described in chapter 3. The semantics is defined by means of a transition system. In chapter

4, an example Dribble program is given. Using this example, the languages GOAL, 3APL and Dribble are compared. On top of the operational semantics of Dribble, a dynamic logic is constructed that can be used to specify and verify properties of Dribble agents (chapter 5). Furthermore, the correspondence between the logic and the operational semantics is proven in section 6. Because of this correspondence, statements proven in the logic actually concern properties of the agent. I conclude the paper with a summary and an indication of future research topics concerning Dribble in chapter 7.

Chapter 2

The programming language

In this section, I introduce the programming language Dribble. As was stated in the introduction, the language is inspired by two agent programming languages: GOAL and 3APL. The language GOAL incorporates as a distinct feature the notion of a *declarative goal*. GOAL agents use these declarative goals to select actions. GOAL does not have planning features. The actions that an agent can select are basic and are executed immediately and completely when selected for execution (if they are enabled). The language 3APL on the other hand offers the means for *creating and modifying plans* during the execution of the agent. The language Dribble is an attempt to combine these features of both languages, thus creating an agent programming language which incorporates declarative goals as well as planning features.

A Dribble agent should be able to select a plan to reach a goal from where it is at a certain point in time. In order to do that, the agent has beliefs, goals and rules to select plans and to create and modify plans. The beliefs, goals and plan of the agent change during the execution of the agent. Together, they form the so called mental state of the agent. In the following sections, I will define these notions and explain the meaning informally. In chapter 3 the formal meaning will be discussed. In the sequel, sometimes typical elements will be used to denote the membership of some set without stating that set explicitly.

2.1 Beliefs and goals

The beliefs and goals of a Dribble agent are drawn from the same logical language $\mathcal{L}$, which is a propositional language. As a consequence, the use of variables or parameter mechanisms is not discussed. The reason is, that more research is needed to extend the programming language with a parameter passing mechanism and to extend the programming logic for Dribble with first order features. There was not enough time in this project to do this research. The agent keeps two databases: a *belief base* and a *goal base* which are both sets of formulas from $\mathcal{L}$. The difference between the two databases originates from the different meaning assigned to sentences from the belief base and goal base. This is explained further in section 2.3.

In the rules to select, create or modify plans, one should be able to refer to sentences from the belief base and/or goal base. Therefore, the language $\mathcal{L}$ is extended to the languages $\mathcal{L}_B$, $\mathcal{L}_G$ and $\mathcal{L}_{BG}$ to denote that reference is made to the belief base, to the goal base or to the belief and/or goal base respectively. These languages are defined below.

Definition (*belief and goal formulas*)
The language $\mathcal{L}$ is a propositional language with typical formula ϕ and the connectives $\wedge$ and $\neg$ with the usual meaning. The set of belief formulas $\mathcal{L}_B$ with typical formula β is defined by:

- if $\phi \in \mathcal{L}$, then $\mathbf{B}\phi \in \mathcal{L}_B$,
- if $\beta_1, \beta_2 \in \mathcal{L}_B$, then $\neg\beta_1, \beta_1 \wedge \beta_2 \in \mathcal{L}_B$.

The set of goal formulas $\mathcal{L}_G$ with typical formula κ is defined by:

- if $\phi \in \mathcal{L}$, then $\mathbf{G}\phi \in \mathcal{L}_G$,
- if $\kappa_1, \kappa_2 \in \mathcal{L}_G$, then $\neg\kappa_1, \kappa_1 \wedge \kappa_2 \in \mathcal{L}_G$.

The set of belief and goal formulas $\mathcal{L}_{BG}$ with typical formula φ is defined by:

- $\mathcal{L}_B \subseteq \mathcal{L}_{BG}$,
- $\mathcal{L}_G \subseteq \mathcal{L}_{BG}$,
- if $\varphi_1, \varphi_2 \in \mathcal{L}_{BG}$, then $\neg\varphi_1, \varphi_1 \wedge \varphi_2 \in \mathcal{L}_{BG}$.

The usual abbreviatons for the propositional operators $\vee$, $\rightarrow$ and $\leftrightarrow$ are used for $\mathcal{L}$, $\mathcal{L}_B$, $\mathcal{L}_G$ and $\mathcal{L}_{BG}$.

2.2 Plans

In order to reach its goals, a Dribble agent adopts plans. A plan is a sequence of *basic elements* of a plan. There are three types of basic elements: basic actions, abstract plans and if-then-else constructs. In the following sections, the basic elements and the construction of a sequence of basic elements will be discussed.

2.2.1 Basic actions

One type of basic element of a plan is a *basic action.* As in the languages GOAL and 3APL, basic actions specify the capabilities with which an agent has to achieve a certain state of affairs. The effect of execution of a basic action is not a change in the world, but a change in the belief base of the agent. The state of affairs that can be realised through execution of basic actions, is therefore a state of affairs of the belief base. One would hope that the belief base of the agent corresponds to a state of affairs of the external world and that the action therefore not only changes what the agent *believes* to be the case, but also what *is* the case in the world. This connection with the external world is however not incorporated in the programming language. A Dribble agent is a mental entity with capabilities to change its mental state. If execution of basic actions should not only affect the beliefs of the agent but should also affect the external world, this would have to be done through some interface with the programming language. If, for instance, the programming language is used to control a robot, an action *moveLeft* should make the robot believe that it has moved left and should actually make him move left through some interface with the programming language. The programming language however does not require such an interface with an external world.

It may seem unnatural that the effect of execution of a basic action is first of all a change in the beliefs of the agent and that changes in the external world are only possible "side effects" (realised through some interface with the programming language). Maybe it would seem more natural to have an agent execute an action and then have him observe the effects of this action. These observations could then be used to update the belief base of the agent. In this way, the agent would have a direct connection with the external world it acts in. Its beliefs would reflect its observations.

There are however two problems with this approach. First of all, not all actions an agent should be able to execute, influence the external world in a way which can be observed. For instance, if you throw a coin off a very tall building, you would expect the coin to end up on the streets at the foot of the building. There is however no way to observe this, standing on top of the building. In this case, the agent would not believe anything about the coin until it finds the coin. This does not seem to be a desirable situation. It seems more natural to have the agent *believe* the coin to be on the streets, after the throw. The beliefs of the agent are then influenced by its actions, or more specificly, by what the agent believes to be the result of the action. If the agent later observes the coin not to be on the streets, its belief could be changed according to this observation.

The second problem of defining actions only as changes to some external world, is that not all actions influence the world. Some actions should influence only the beliefs of the agent (for instance if the agent wants to memorise something). It is for these reasons that basic actions update the beliefs of an agent.

2.2.2 Abstract plans

The second basic element of a plan is the *abstract plan*. An abstract plan cannot be executed directly in the sense that it updates the belief base of an agent. Abstract plans serve as an abstraction mechanism like procedures in imperative programming. If a plan consists of an abstract plan, this abstract plan could be translated into basic actions through reasoning rules (see section 2.4). These basic actions could then be executed.

In 3APL, abstract plans are called achievement goals. The reason for this is that, apart from the procedural reading of these plans, an assertional reading is also possible. This is the case if the abstract plan is the same as some atomic proposition from the language $\mathcal{L}$. If the plans for achieving this achievement goal actually realise a state of affairs in which the proposition is true, this reading seems to be valid. The plans for realising an achievement goal are written by the programmer. Whether the assertional reading of an abstract plan is valid, is thus determined by the programmer.

2.2.3 If-then-else

The last basic element of a plan is the $\texttt{if}\ \beta\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2\ \texttt{fi}$ construct. The informal reading of this construct is as follows: if the agent believes β to be the case, adopt the plan π_1, else adopt the plan π_2. This construct does not look like a basic element of a plan, because it is composed of plans. The meaning however is not defined in terms of the meaning of these plans. The construct is treated as a basic element and the meaning is defined as such. This will be explained further in sections 3.2.4 and 5.2.3.

If the construct is executed, the plan of the agent is updated. The plan either becomes π_1 or π_2, depending on the truth of the belief formula β.

2.2.4 Sequential composition

In the previous sections, the basic elements of a plan were discussed. A plan can be empty, it can be a basic element or a sequence of multiple basic elements. A sequence of basic elements can be constructed using the programming construct sequential composition (denoted by a semicolon).

2.2.5 Definition of plans

Below, the set of plans is defined. In this definition, the set of executable actions is defined, consisting of basic actions and if-then-else constructs. Later on, it will be convenient to have this set.

Definition (*plans*)
Assume that a set $\mathsf{BasicAction}$ of basic actions is given and has typical element a. Assume that a set $\mathsf{AbstractPlan}$ of abstract plans is given, with typical element p. Let the set $\mathsf{ExecutableAction}$ with typical element b be $\mathsf{BasicAction} \cup \{\texttt{if}\ \beta\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2\ \texttt{fi}\}$ ($\beta \in \mathcal{L}_B$ and $\pi_1, \pi_2 \in \mathsf{Plan}$). The symbol E denotes the empty plan. $E;\pi$ is identified with π. The set of plans Plan with typical element π is defined by:

- $E \in \mathsf{Plan}$,
- $\mathsf{BasicAction} \subseteq \mathsf{Plan}$,
- $\mathsf{AbstractPlan} \subseteq \mathsf{Plan}$,
- if $\pi_1, \pi_2 \in \mathsf{Plan}$ and $\beta \in \mathcal{L}_B$ then $\texttt{if}\ \beta\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2\ \texttt{fi} \in \mathsf{Plan}$,
- if $c \in \mathsf{ExecutableAction} \cup \mathsf{AbstractPlan} \cup \{E\}$ and $\pi \in \mathsf{Plan}$ then $c\,;\pi \in \mathsf{Plan}$.

Plans in the language Dribble are defined different from plans in 3APL. The reason for this choice will be explained in section 2.5.

2.3 Mental state

In sections 2.1 and 2.2 I have defined the beliefs, goals and plans of a Dribble agent. These are the elements that constitute the mental state of an agent. As was stated, the beliefs in the belief base and the goals in the goal base are drawn from the same propositional language $\mathcal{L}$. The beliefs of the agent describe the state (of the world) the agent is in (or believes to be in) and the goals describe the state (of the world) the agent wants to realise (or wants to believe to have realised). As was explained, basic actions change the beliefs of the agent. It has however not yet been explained how the goals of the agent are changed and how the plan is changed. The plan of the agent is changed through execution of actions from the plan or through application of rules (see section 2.4). I will now explain how the goals of an agent are updated.

The question is: when should agents adopt or drop goals? The way in which an agent does this, is called a *commitment strategy* [5]. An agent that is very lazy could for instance drop a goal as soon as it is adopted (thinking, if I do not have a goal, I will not have to do anything to reach it). An agent could also drop a goal if it believes that that goal has been achieved. This seems a reasonable commitment strategy. A third way to handle the dropping of goals could be, that an agent drops a goal if it believes that it has been achieved or if it believes that it will never be achieved. This seems reasonable too but it is difficult to implement, for how would the agent come to believe that a goal will never be achieved?

The strategy that was chosen for Dribble agents is the second one: an agent drops a goal if and only if it believes that that goal has been achieved. The goals of the agent can thus only be updated through belief updates. They cannot be updated directly through execution of some action. In this way, a meaningful relation between beliefs and goals is established. It becomes clear moreover, that beliefs and goals will have to be drawn from the same language, because otherwise it will be difficult to establish whether the agent believes the goal to have been achieved. The way in which an agent adopts goals, is that the goals are given to it at startup (see section 2.6). It cannot adopt goals during execution.

If an agent behaves according to this commitment strategy, the mental state of the agent will have to meet the requirement that an agent does not believe that ϕ is the case if it has a goal to achieve ϕ. If it believes ϕ, it cannot have the goal to achieve ϕ, because the state of affairs ϕ is already reached. Another requirement is that a particular goal of an agent is consistent. It cannot desire a state of affairs which cannot be reached by definition. The last requirement on mental states of Dribble agents is that their belief bases are consistent. An agent with an inconsistent belief base would believe everything which is not a desirable situation. It could hardly function, because all requirements on beliefs in rules would be met, but none of the requirements on goals would be: the agent would believe everything and therefore would not have any goals.

The goal base of an agent does not have to be consistent, because two inconsistent goals could both be realised, although not at the same time. An agent may for instance have the goals to be at the bakery and be at gym which cannot be realised at the same time (they are inconsistent). They can however both be reached, but only in succession. A consequence of this is, that the agent cannot adopt the conjunction of two separate goals in the goal base as a new goal, for this could lead to inconsistent goals. The mental state of a Dribble agent is defined below.

Definition (*mental state of a Dribble agent*)
A mental state of a Dribble agent is a triple $\langle \sigma, \gamma, \pi \rangle$ where $\sigma \subseteq \mathcal{L}$ are the agent's beliefs and $\gamma \subseteq \mathcal{L}$ are the agent's goals and $\pi \in \mathsf{Plan}$ is the agent's plan. σ and γ are such that for any $\psi \in \gamma$ we have:

- ψ is not entailed by the agent's beliefs ($\sigma \not\models \psi$), and
- ψ is consistent ($\psi \not\models \perp$), and
- σ is consistent ($\sigma \not\models \perp$).

Σ with typical element s is the set of all possible mental states.

2.4 Rules

As was described in the previous sections, the plan of an agent can be changed through the application of rules or execution of executable actions (basic actions or if-then-else constructs). The execution of actions will be described in chapter 3. In this section I will describe the application of rules in more detail. There are two kinds of rules: *goal rules* and *Practical Reasoning (PR) rules*. Goal rules are taken from the language GOAL and PR rules are taken from 3APL. Through a goal rule, a new plan can be adopted, depending on the beliefs and goals of the agent. With PR rules, an adopted plan can be modified.

2.4.1 Goal rules

Definition (*goal rule*)
A goal rule g is a pair $\varphi \rightarrow \pi$ such that $\varphi \in \mathcal{L}_{BG}$ and $\pi \in \mathsf{Plan}$.

Informally, a goal rule means that the plan π can be adopted, if the mental condition φ holds. The formula φ is a condition on the beliefs and/or goals of the agent (called the antecedent of the rule). Goal rules are used to *select* plans to reach a goal from a certain state. The conditions in the goal rule on the beliefs of an agent are used to describe the situation in which it could be a good idea to execute the plan. The conditions on goals in φ specify what the plan is good for. It may be the case that the specified goal is not actually reached through execution of the plan, but the idea is that execution of the plan will at least help in bringing about the goal condition.

A goal rule can only be applied if the current plan of the agent is empty. This is done because an agent can only execute one plan at the time. If it would be possible to select new plans during the execution of another plan, it is not clear what to do with the newly adopted plan. Should it be executed before or after the old plan? If it would be executed *after* the old plan, the agent could just as well first execute its old plan and then apply the goal rule to adopt the new plan. If the goal rule is not applicable in that situation anymore, it would not have been a good idea to execute the new plan after the old plan anyway.

If the newly adopted plan is executed *before* the old plan, this would result in an agent that is not persistent. After adopting a plan to reach a goal, at the next point in time the agent could just as well adopt some other plan to go after another goal in an entirely different direction. These are the reasons why I chose for a goal rule to be applicable only if the agent has an empty plan. In this way goal rules are used for adoption of new plans only, while the modifying of plans is left to the PR rules.

2.4.2 PR rules

Definition (*PR rule*)
A PR rule ρ is a triple $\pi_h \mid \beta \to \pi_b$ such that $\beta \in \mathcal{L}_B$ and $\pi_h, \pi_b \in \mathsf{Plan}$.

The informal reading of a PR rule $\pi_h \mid \beta \to \pi_b$ is the following: if the agent has adopted the plan π_h and if it believes the belief formula β to be the case, the plan π_h can be replaced by the plan π_b. π_h is called the head of the rule, β is called the guard and π_b is called the body. I will explain the use of PR rules by demonstrating three ways in which they can be applied: to *create* plans (often from abstract plans), to *modify* plans and to model *reactive behaviour*.

Suppose p is an abstract plan. The rule $p \mid \beta \to a; b; c$ can then be used to create a more concrete plan $a; b; c$ from the abstract plan p. This type of rule is similar to a procedure in imperative programming. The name of the procedure is p, which can be called. If it is called, the body of the procedure is executed which can in turn contain procedure calls. The difference between a procedure and this kind of PR rule is, that the body of a procedure is usually executed completely if the procedure is called. The body of a PR rule on the contrary can be modified by the application of a PR rule.

Suppose $\pi_h \mid \beta \to \pi_b$ is a PR rule. Suppose β describes a situation in which the plan π_h would fail. This rule could then be used to modify the plan π_h by instead adopting the plan π_b which should be the more appropriate plan for situation β. The plan π_b could be another way to realise the state of affairs that π_h was supposed to realise, or could for instance be empty if that state of affairs could not be realised at all.

In a PR rule $\pi_h \mid \beta \to \pi_b$ the head of the rule can be empty, resulting in a rule of the form $E \mid \beta \to \pi_b$. This rule is not used to modify plans, but to model reactive behaviour. In any situation in which the agent believes β to be the case, the plan π_b can be adopted regardless of the current plan of the agent. The reason will be explained in section 3.2.5. The idea is that the agent should act immediately if for instance some hazardous situation occurs, regardless of what it is doing at the time. This rule has a close resemblance with the goal rule $\beta \to \pi_b$. The difference however is, that goal rules are used as a means to adopt new plans only, after the old plan has been executed completely (the current plan of the agent has to be empty). The PR rule on the contrary can also be applied if the current plan is not empty.

PR rules and the fact that plans are part of the mental state of an agent, provide the agent with reflective capabilities concerning plans. The agent can monitor its plans and adapt them if necessary. Note that these rules could not simply be replaced, for examply by the if-then-else construct. It would for instance not be possible to simulate the rule $a; b \mid \beta \to b; a$ in this way. With this rule, the agent can really adapt its plan, which is fundamentaly different from executing some procedure under some condition or executing a procedure call.

2.5 A choice made in defining plans

The set of plans of the language Dribble is defined a little bit different from the way it is done for the language 3APL. In both languages, a plan is a sequence of basic elements of length zero or more. The difference is, that the sequences are constructed in different ways. In Dribble, a plan can be a sequence of a basic element and a plan: $c;\pi$. A 3APL plan can be a sequence of two plans: $\pi_1;\pi_2$. The way in which a sequence is constructed, should not affect the execution of the agent. A PR rule $c_1;c_2;c_3 \mid \beta \rightarrow \pi_b$ for instance should be applicable to the sequence of basic elements $c_1;c_2;c_3$, regardless of whether this sequence was constructed as a concatenation of c_1 and $c_2;c_3$ or of $c_1;c_2$ and c_3.

I will now explain the difference between plans in Dribble and plans in 3APL, using grammars that abstract from the concrete syntax of plans. These grammars omit the fact that plans can be empty, because it is not relevant for the discussion.

Plan	::=	BasicElement |
		BasicElement Plan

Plan	::=	BasicElement |
		Plan Plan

The first grammar defines plans of Dribble. The second one defines plans of 3APL. The basic elements are the same for both grammars.

BasicElement ::= $c_1 \mid c_2 \mid c_3$

With these grammars, abstract syntax trees can be constructed for some plan $c_1;c_2;c_3$. I will describe these trees using a predicate notation. The predicate Plan(BasicElement(c_1), Plan(BasicElement(c_2), BasicElement(c_3))) refers to the tree of which a Plan is the root. The first child of the root is a BasicElement, which has a child c_1. The second child of the root is another Plan, which has two children BasicElement that each have a child: c_2 and c_3 respectively. Thus the leaves of the tree are c_1, c_2 and c_3. From now on I will omit the nodes BasicElement for the sake of clarity.

The trees that can be constructed for the plan $c_1;c_2;c_3$ with the second grammar are: Plan(Plan(c_1), Plan(c_2, c_3)) and Plan(Plan(c_1, c_2), Plan(c_3)). With the first grammar, only one tree can be constructed: Plan(c_1, Plan(c_2, c_3)). The second grammar is ambiguous: for one plan, multiple different abstract syntax trees can be constructed. The first grammar is unambiguous: only one abstract syntax tree can be constructed for some plan. The question is, which grammar do we want and/or need?

I will now try to answer this question, using the example of application of a PR rule $c_1;c_2;c_3 \mid \beta \rightarrow \pi_b$ to the plan $c_1;c_2;c_3$ of the agent. The PR rule can be applied if the head of it is equal to the plan of the agent. The rule should thus be applicable. Now suppose that the second grammar is used to define plans. The abstract syntax tree of the head of the PR rule can now differ from the tree of the plan of the agent. This difference should however result in the same execution of the agent, namely that the PR rule can be applied. What matters is the order of the basic elements of the plan, which are the leaves of the tree. So if the second grammar would be used, there would have to be some arrangement to make sure that different trees with the same leaves in the same order, are interpreted in the same way.

They would have to be transformed into a format in which they could be compared. This format could be the format of the first grammar, which I will call the list format. Having said all this, using the second grammar seems to complicate matters. Therefore, using the first grammar is preferable.

Not only however is it *preferable* to use the first grammar, we in fact do not even *need* the second grammar. As was stated above, what matters is the order of the leaves of the tree. Fortunately, the first grammar is enough to express this. The first grammar defines lists of basic elements and the order in which the elements are arranged in the list, defines the execution of the program. This is the reason that I chose to use the first grammar.

In the sequel, it will turn out to be useful to be able to express that a certain sequence of basic elements (which is a plan), is equal to the prefix of some other plan. If one wants to express that a plan π_1 is the prefix of some plan (of which π_2 is the rest), this cannot be done using the construct for sequential composition ($\pi_1;\pi_2$). This construct can only be used to express that some basic element is the prefix of some plan ($c\,;\pi$).

To express that a plan is equal to the prefix of another plan, I use the operator $\circ$ ($\pi_1 \circ \pi_2$). This operator is a function which concatenates two plans. The result is a new plan with π_1 as prefix of that plan and π_2 as the rest. The operator $\circ$ is of type : $\mathsf{Plan} \times \mathsf{Plan} \to \mathsf{Plan}$. It is defined as follows (the symbol E denotes the empty plan, which is an empty list):

- $E \circ \pi \;=\; \pi \circ E \;=\; \pi$,
- $(c\,;\pi_1) \circ \pi_2 = c\,;(\pi_1 \circ \pi_2)$.

2.6 A Dribble agent

In the previous sections, the beliefs, goals and plans of Dribble agents were described. These form the dynamic components of a Dribble agent which together constitute the mental state of the agent. Furthermore, goal rules and PR rules were introduced. To program an agent means to specify its initial beliefs and goals and to write sets of goal rules and PR rules. This is formalised in the specification of a Dribble agent below.

Definition (*a Dribble agent*)
A Dribble agent is a quadruple $\langle \sigma_0, \gamma_0, \Gamma, \Delta \rangle$ where $\langle \sigma_0, \gamma_0, E \rangle$ is the initial mental state, Γ is a set of goal rules and Δ is a set of PR rules.

A choice was made to let the plan of the agent be empty in the initial mental state. This was done because I think the agent should start to act because it has certain goals. A Dribble agent should adopt plans to reach these goals with the goal rules. Giving the agent a plan at start up is in contradiction with this idea.

2.7 Searching agents

From the previous sections, it can be concluded that Dribble agents do not use backward search. In backward search, the agent has a specification of the goal state and it has specifications of the preconditions and results of actions. With these action specifications, the agent constructs a plan to reach the goal state from the starting state. It constructs this plan, starting from the goal state (regression).

A Dribble agent does have action specifications and it is possible to define a goal state and a starting state, but it does not use these to do backward search. It selects its actions (or plans) in a forward manner. It however does not use forward search, because in that case it would use the action specifications to construct a plan. A Dribble agent uses its rules to select a plan. In these goal rules, the programmer can specify the goal state. The execution of the plan of the rule is supposed to help in bringing about this goal state. It is thus the programmer that should envision the result of a plan and the programmer should specify this in the rules. The agent cannot reason about the results of its actions, thus selecting actions that would result in realisation of the goal state. This is the job of the programmer.

The advantage of letting the programmer do this job is, that the agent does not have to search through a huge search space. It could then end up doing nothing, because all its time is consumed by searching for an appropriate plan. A Dribble agent is more flexible than a searching agent, because the agent can easily adapt its plan with its PR rules during execution. It does not have to start searching all over if the world changes in some way during the execution of the plan.

Thus a goal agent acts goal directed in the sense that it selects rules that are supposed to help in bringing about its goal state. The specification of these rules however is left to the programmer and the agent cannot reason about its goal state and the actions that should reach it.

Chapter 3

Operational semantics

The mental state of an agent consists of beliefs, goals and a plan. These elements change during execution of the agent and they are thus the dynamic parts of the agent. The mental state changes as a consequence of the execution of actions and the application of rules. In the previous chapter these changes were described informally. In this chapter I will give a formal semantics of the possible mental state changes. The semantics I use is an *operational semantics* defined by means of a transition system ([8], see section 3.2).

First I will however define the meaning of a statement about the beliefs or goals of the agent (for instance $\mathbf{B}\phi$). This is necessary because these statements can be used in rules. Depending on the truth of these statements, the rules can or cannot be applied.

3.1 Semantics of belief and goal formulas

In section 2.1, the belief and goal formulas were defined. In this section the meaning of these formulas will be defined. Belief and goal formulas are evaluated in a mental state. A formula $\mathbf{B}\phi$ is true in a mental state, if and only if the belief base σ models ϕ. In this way, the agent believes all logical consequences of its beliefs. If the agent for instance believes two formulas ϕ_1 and ϕ_2, it also believes $\phi_1 \wedge \phi_2$.

The definition of when a formula $\mathbf{G}\psi$ is true, is a little bit more complicated. In section 2.3 it was explained that the goal base of an agent can be inconsistent, because two separate goals may be inconsistent. It was also explained that because of this, two separate goals should not be combined into one goal. The definition of the semantics of $\mathbf{G}\psi$ will therefore have to be defined in terms of separate goals, as opposed to defining it in terms of the entire goal base. The idea is, that all logical consequences of a particular goal are also goals, but only if they are not believed. This is expressed with the relation $\leadsto_\sigma$ [5].

Definition ($\leadsto_\sigma$)
Let $\langle \sigma, \gamma, \pi \rangle$ be a mental state. Let $\psi_1, \psi_2 \in \mathcal{L}$ be propositional formulas.

$\psi_2 \ \leadsto_\sigma \ \gamma$ iff for some $\psi_1 \in \gamma : \psi_1 \models \psi_2$ and $\sigma \not\models \psi_2$

Below, the semantics of belief and goal formulas ($\mathcal{L}_{BG}$) is defined. The belief formulas $\mathcal{L}_B$ are a subset of $\mathcal{L}_{BG}$. The semantics of the belief formulas is thus also defined by the definition below. The semantics of $\mathbf{G}\psi$ is defined in terms of the relation $\rightsquigarrow_\sigma$.

Definition (*semantics of belief and goal formulas*)
Let $\langle\sigma,\gamma,\pi\rangle$ be a mental state. Let ϕ, $\psi \in \mathcal{L}$ be propositional formulas and φ, φ_1, $\varphi_2 \in \mathcal{L}_{BG}$ be belief and goal formulas.

- $\langle\sigma,\gamma,\pi\rangle \models \mathbf{B}\phi$ iff $\sigma \models \phi$
- $\langle\sigma,\gamma,\pi\rangle \models \mathbf{G}\psi$ iff $\psi \rightsquigarrow_\sigma \gamma$
- $\langle\sigma,\gamma,\pi\rangle \models \neg\varphi$ iff $\langle\sigma,\gamma,\pi\rangle \not\models \varphi$
- $\langle\sigma,\gamma,\pi\rangle \models \varphi_1 \wedge \varphi_2$ iff $\langle\sigma,\gamma,\pi\rangle \models \varphi_1$ and $\langle\sigma,\gamma,\pi\rangle \models \varphi_2$

Suppose that $p \wedge q$ is a goal in the goal base of the agent in a mental state and that $\mathbf{G}p \to \pi$ is a goal rule. This rule could be applied in that mental state, because p is a logical consequence of $p \wedge q$ (assuming p is not believed by the agent). It turns out to be very convenient to be able to formulate a goal as a conjunction of other goals and to have rules that apply to the individual conjuncts (see chapter 4). Moreover, it doesn't seem counterintuitive that an agent executes a plan that will help in bringing about p, if it has $p \wedge q$ as a goal. The execution of the plan will at least help to realise the state of affairs $p \wedge q$. The plan will probably not completely realise $p \wedge q$. If this is the case however, the goal $p \wedge q$ will remain present in the goal base.

3.1.1 Possible problems

In the definition *semantics of belief and goal formulas* defined above, a formula $\neg\varphi$ is true in a mental state if and only if that mental state does not model φ. It is not immediately clear why this definition does not cause problems. I will first explain which problems could arise and then I will explain why these problems will not occur. I will explain this in terms of the belief base, but the same kind of argument can be constructed for the goal base.

The key point to make is, that formulas from the belief base are "shielded" with the belief operator $\mathbf{B}$. Formulas that can be evaluated in a mental state are always formulas formed with the belief operator (belief formulas). A formula ϕ from the belief base cannot be evaluated in a mental state. The consequence of this is the following. If φ is equal to $\mathbf{B}\phi$ and if $\mathbf{B}\phi$ is not true in a certain mental state, the negation of $\mathbf{B}\phi$ becomes true in that mental state, as opposed to the negation of ϕ becoming true.

If one would evaluate formulas ϕ from the belief base in a mental state (ϕ is true if it follows from the belief base), it would cause problems to state that a formula $\neg\phi$ is true if and only if it is not the case that ϕ is true. This problematic definition of the meaning of formulas from the belief base would be the following:

$$\begin{aligned}\langle\sigma,\gamma,\pi\rangle \models \phi &\quad\Leftrightarrow\quad \sigma \models \phi,\\ \langle\sigma,\gamma,\pi\rangle \models \neg\phi &\quad\Leftrightarrow\quad \langle\sigma,\gamma,\pi\rangle \not\models \phi.\end{aligned}$$

This definition would give rise to the following equivalences:

$$\begin{aligned}\sigma \not\models \phi &\Leftrightarrow \langle\sigma,\gamma,\pi\rangle \not\models \phi\\ &\Leftrightarrow \langle\sigma,\gamma,\pi\rangle \models \neg\phi\\ &\Leftrightarrow \sigma \models \neg\phi\end{aligned}$$

The equivalence $\sigma \not\models \phi \Leftrightarrow \sigma \models \neg\phi$ causes problems if for instance ϕ is equal to $p \vee q$:

$$\begin{aligned}p \vee q \not\models p &\Leftrightarrow p \vee q \models \neg p\\ p \vee q \not\models q &\Leftrightarrow p \vee q \models \neg q\\ \\ &\Rightarrow p \vee q \models \neg p \wedge \neg q\\ &\Rightarrow p \vee q \models \neg(p \vee q)\end{aligned}$$

This last statement is simply untrue. The problematic equivalence will not occur if formulas evaluated in a mental state are belief formulas (formulas shielded with the **B** operator). The following equivalence can then be derived: $\sigma \not\models \phi \Leftrightarrow \langle\sigma,\gamma,\pi\rangle \not\models \mathbf{B}\phi \Leftrightarrow \langle\sigma,\gamma,\pi\rangle \models \neg\mathbf{B}\phi$. The problematic definition implements a closed world assumption: if ϕ cannot be derived from the belief base, assume that ϕ is false. The definition with the **B** operator does not implement a closed world assumption. A formula $\mathbf{B}\phi$ can be read as "ϕ is derivable from the belief base σ", whereas $\neg\mathbf{B}\phi$ can be read as "ϕ is not derivable from the belief base σ". One cannot conclude from the fact that a formula is not derivable, that the negation of this formula *is* derivable. There is a difference between $\neg\mathbf{B}\phi$ and $\mathbf{B}\neg\phi$. The last formula states that $\neg\phi$ is derivable.

Thus formulas of the form $\mathbf{B}\phi$ can be seen as atoms in the logic of belief formulas. They are evaluated in a mental state and the truth of these atoms is determined by the derivability of ϕ from the belief base in that mental state. A formula ϕ is always either derivable or not derivable. Therefore it does not cause problems to state that if it is *not* the case that a formula *is* derivable, this formula is *not* derivable.

3.2 Transition system

Transition systems are a means to define the operational semantics of a programming language. A transition system consists of a set of derivation rules for deriving transitions for an agent. A transition is a transformation of one mental state into another and it corresponds to a single computation step. A set of transition rules can be viewed as an inductive definition of a transition relation $\rightarrow$. The relation $\rightarrow$ defined by a transition system, is the smallest relation which contains all the axioms of the system and all conclusions which are derivable by using these axioms. This transition relation specifies the possible computation steps of an agent. In this section, the transition rules of a Dribble agent with goal rules Γ and PR rules Δ are specified.

3.2.1 Application of a goal rule

A goal rule is applicable in a mental state if the antecedent of the goal rule is true in that mental state. It can only be applied if the plan of the agent is empty. In the mental state resulting from application of a goal rule, the plan becomes equal to the consequent of the goal rule. The beliefs and the goals are not influenced by application of a goal rule.

Definition (*application of a goal rule*)

$$\frac{\langle\sigma,\gamma,E\rangle \models \varphi}{\langle\sigma,\gamma,E\rangle \rightarrow_{\texttt{applyRule}(g)} \langle\sigma,\gamma,\pi\rangle}$$

with $g : \varphi \rightarrow \pi \in \Gamma$ a goal rule.

For each transition, the kind of transition (for instance application of a goal rule) is specified as a subscript of the transition relation $\rightarrow$. This is necessary because the transition rule for sequential composition should not be applied to transitions derived with the transition rule for application of a goal rule. This will be explained in section 3.2.5.

The transition rule for application of a goal rule only specifies *how* to apply a rule in a mental state. It however does not specify *which* applicable rule should be applied, in case more than one rule is applicable.

3.2.2 Application of a PR rule

A PR rule is applicable in a mental state, if the plan of the agent is equal to the head of the rule and if the guard of the rule is true in that mental state. The result of the application of a PR rule is that the plan in the body of the rule is adopted, replacing the plan which was equal to the head of the rule. The beliefs and goals are not changed when a PR rule is applied. As for goal rules, the transition rule for application of a PR rule does not specify which PR rule should be applied in case more rules are applicable.

Definition (*application of a PR rule*)

$$\frac{\langle\sigma,\gamma,\pi_h\rangle \models \beta}{\langle\sigma,\gamma,\pi_h\rangle \rightarrow_{\texttt{applyRule}(\rho)} \langle\sigma,\gamma,\pi_b\rangle}$$

with $\rho : \pi_h \mid \beta \rightarrow \pi_b \in \Delta$ a PR rule.

The guard of a PR rule is a condition on the beliefs of the agent. This is in contrast with goal rules, in which the antecedent can be a condition on beliefs and/or goals. I will try to explain the reason for the decision to let the guard of PR rules be a condition on beliefs only.

The idea is, that a goal rule $g : \mathbf{B}(\phi) \wedge \mathbf{G}(\psi) \rightarrow \pi$ is used to *select* a plan π in a mental state in which the plan of the agent is empty. The plan π should help in bringing about the goal ψ from the belief state ϕ. PR rules can be used during execution of π, for example to modify it if the agent encounters some obstacle. Once the goal rule g is applied, ψ is chosen as the goal to be realised. The plan π should result in a mental state in which ψ holds (or at least in a mental state closer to ψ than ϕ was). Any change of the plan during execution should thus be directed towards realising a state of affairs in which ψ holds. For this it is not necessary to check the goals of the agent. Moreover, using belief and goal formulas as the guard of PR rule, obscures the idea of separation of plan selection and plan execution.

There is another disadvantage of letting the guard of PR rules be a condition on beliefs and/or goals. If this would be possible, it would be possible to specify the following PR rule: $\rho : E \mid \varphi \rightarrow \pi$. This rule has a close resemblance with goal rule $\varphi \rightarrow \pi$. The difference is, that the goal rule can only be applied if the plan of the agent is empty, whereas the PR rule can also be applied if the plan is not empty. The goal rule is applicable in a subset of the mental states in which the PR rule would be applicable. The reason for the restriction on mental states in which a goal rule is applicable, was explained in section 2.4.1. For this reason, it is not desirable that the PR rule ρ can be specified in the language Dribble.

An advantage of letting the guard of PR rules be a condition on beliefs and/or goals is the following. The plan of the agent could be modified, depending on the goals of the agent. I will explain this using an example. Suppose that the agent has the goal to be at the gym. In order to satisfy this goal, the agent adopts the plan to get downtown by bus. It however turns out that the employers of the buscompany have declared a strike. It is thus not possible to get downtown by bus and the agent will have to modify its plan. It can either drop the plan to get downtown or adopt an alternative plan for example to go by bike. If it would be possible to use goals in PR rules to modify plans, the agent could modify its plan depending on the goal for which it was selected. If the agent selected the plan to go downtown because it wants to be at the gym, the agent could for example drop this plan: it could try again tomorrow. If however the plan to go downtown would have been selected because the agent needs to go to work there, the agent should not drop this plan. It should try to get to work for example by bike. Despite this advantage, I chose to let the guard of PR rules be a condition on beliefs only because of the disadvantages described above. As demonstrated by the example, it would perhaps still be useful to establish some connection between the plan of the agent and the goal for which it was selected. I think that this connection should however not be realised by introducing goals in the guard of the PR rules because of the described disadvantages. More research is needed to develop a method for realising this connection without introducing goals in the guard of the PR rules.

3.2.3 Basic action execution

As was explained, basic actions update the belief base of an agent if they are executed. These belief updates are formally represented by a partial function $\mathcal{T}$ of type : $\mathsf{BasicAction} \times \wp(\mathcal{L}) \to \wp(\mathcal{L})$. $\mathcal{T}(a, \sigma)$ returns the result of updating belief base σ by performing action a. The fact that $\mathcal{T}$ is a partial function represents the fact that an action may not be executable in some belief states.

A basic action can be executed in a mental state if the function $\mathcal{T}$ is defined (and thus has some result σ') and if the plan of the agent is equal to the action in that mental state. If $\mathcal{T}(a, \sigma) = \sigma'$, execution of basic action a in a mental state with belief base σ, results in a mental state with belief base σ'. As was explained, the goals of an agent are updated through belief updates. Goals that are reached through execution of an action, should be removed from the goal base. This is specified in the definition below. After execution of an action, the action is removed from the plan.

Definition (*action execution(1)*)

$$\frac{\mathcal{T}(a, \sigma) = \sigma'}{\langle \sigma, \gamma, a \rangle \to_{\texttt{execute(a)}} \langle \sigma', \gamma', E \rangle}$$

with $\gamma' = \gamma \setminus \{\psi \in \gamma \mid \sigma' \models \psi\}$.

3.2.4 Execution of if-then-else constructs

An if-then-else construct can always be executed, as long as the plan of the agent is equal to this construct. The result of execution of the construct depends on the truth of the belief formula after the `if`. If the belief formula is true, the plan of the agent becomes equal to the then-part of the construct and if it is false, the plan becomes equal to the else-part. The beliefs and goals are not influenced by executing an if-then-else construct.

The truth of a belief formula in a mental state can always be established: it is either true or false. This is a consequence of the definition of the meaning of belief formulas (see section 3.1): $\langle \sigma, \gamma, \pi \rangle \models \neg\beta \Leftrightarrow \langle \sigma, \gamma, \pi \rangle \not\models \beta$. If it is not the case that a belief formula is true, it is false. The premise of the second transition below could therefore be replaced by stating the following: $\langle \sigma, \gamma, \texttt{if}\ \beta\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2\ \texttt{fi} \rangle \not\models \beta$.

Definition (*action execution(2)*)

$$\frac{\langle \sigma, \gamma, \texttt{if}\ \beta\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2\ \texttt{fi} \rangle \models \beta}{\langle \sigma, \gamma, \texttt{if}\ \beta\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2\ \texttt{fi} \rangle \rightarrow_{\texttt{execute}(ite)} \langle \sigma, \gamma, \pi_1 \rangle}$$

$$\frac{\langle \sigma, \gamma, \texttt{if}\ \beta\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2\ \texttt{fi} \rangle \models \neg\beta}{\langle \sigma, \gamma, \texttt{if}\ \beta\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2\ \texttt{fi} \rangle \rightarrow_{\texttt{execute}(ite)} \langle \sigma, \gamma, \pi_2 \rangle}$$

with $ite = \texttt{if}\ \beta\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2\ \texttt{fi}$.

The if-then-else construct is defined as a basic element of a plan, which can be executed in one step. It could instead have been defined in terms of a non-deterministic choice operator, sequential composition and a test ($\beta?; \pi_1 + \neg\beta?; \pi_2$). The meaning would be the same (if the test for β succeeds, the non-deterministic choice is replaced by π_1 and if the test for $\neg\beta$ succeeds, it is replaced by π_2). The reason that I do not use the non-deterministic choice operator is, that in section 5.2.3 it will prove to be convenient to be able to define the if-then-else construct as an executable basic element of a plan. Furthermore, for the language Dribble I could not think of a situation in which the non-deterministic choice operator would be used, other than to define the meaning of the if-then-else construct.

The if-condition of the if-then-else construct is a condition on the beliefs of the agent. The reason that it is not a condition on beliefs and/or goals is, that the construct is part of the plan. In section 3.2.2 it was explained that the execution of plans should not depend on the goals of the agent. This is the reason that the if-condition is a condition on beliefs only.

3.2.5 Execution of sequential composition

As was defined in section 2.2.5, a plan can be either a basic element or a sequence of basic elements. The meaning of executing a basic element (for instance a basic action) or a certain fixed sequence of basic elements (the head of a PR rule), was defined in the previous transition rules. In the transition rule for execution of sequential composition, it is defined what it means to execute an arbitrary sequence of basic elements (a plan). The first part of the plan is executed first, any changes to the belief base and goal base are recorded and the agent continues executing the remainder of the plan.

Definition (*execution of sequential composition*)

$$\frac{\langle \sigma, \gamma, \pi_1 \rangle \rightarrow_x \langle \sigma', \gamma', \pi_1' \rangle}{\langle \sigma, \gamma, \pi_1 \circ \pi_2 \rangle \rightarrow_x \langle \sigma', \gamma', \pi_1' \circ \pi_2 \rangle}$$

with $x \in \{\texttt{applyRule}(\rho), \texttt{execute}(a), \texttt{execute}(\texttt{if}\ \beta\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2\ \texttt{fi})\}$.

In this rule, the operator $\circ$ which was defined in section 2.5 is used. It is used to denote that some plan is a sequence of basic elements of which π_1 (or π_1') is the first part and π_2 is the second part: π_1 (or π_1') is the prefix of this plan. This prefix can be executed and the rest of the plan remains to be executed.

The transition rule specifies that it can only be used if the transition that is used as the premise of the rule, was not derived with the transition rule for application of a goal rule: $x \neq \texttt{applyRule}(g)$. The reason is, that goal rules should only be applied if the plan of the agent is empty. It should therefore not be possible to derive some transition $\langle \sigma, \gamma, E \circ \pi' \rangle \rightarrow_{\texttt{applyRule}(g)} \langle \sigma, \gamma, \pi \circ \pi' \rangle$ with $g : \varphi \rightarrow \pi \in \Gamma$ a goal rule. If this transition could be derived, the goal rule would be applicable in a mental state with a plan that is not empty. This transition could be derived through application of the transition rule for sequential composition, after deriving the transition $\langle \sigma, \gamma, E \rangle \rightarrow_{\texttt{applyRule(g)}} \langle \sigma, \gamma, \pi \rangle$ with the rule for application of a goal rule.

The restriction that the transition rule for sequential composition cannot be applied to a transition $s_i \rightarrow_{\texttt{applyRule}(g)} s_{i+1}$, is thus necessary to prevent the derivation of the transition $\langle \sigma, \gamma, E \circ \pi' \rangle \rightarrow_{\texttt{applyRule}(g)} \langle \sigma, \gamma, \pi \circ \pi' \rangle$ from the transition $\langle \sigma, \gamma, E \rangle \rightarrow_{\texttt{applyRule(g)}} \langle \sigma, \gamma, \pi \rangle$.

I will explain now that a PR rule of the form $E \mid \beta \rightarrow \pi_b$ can always be applied in a mental state, regardless of the plan of the agent. The reason is, that the transition $\langle \sigma, \gamma, E \rangle \rightarrow_x \langle \sigma, \gamma, \pi_b \rangle$ can always be derived (provided that the condition specified by the guard of the rule is met by the belief base). Using the transition rule for sequential composition, the transition $\langle \sigma, \gamma, E \circ \pi \rangle \rightarrow_x \langle \sigma, \gamma, \pi_b \circ \pi \rangle$ can be derived. In a mental state $\langle \sigma, \gamma, \pi \rangle$, the rule can thus be applied.

3.3 Semantics of a Dribble agent

The semantics of a Dribble agent is derived directly from the transition relation $\rightarrow$. The meaning of a Dribble agent consists of a set of so called computation runs. Computation runs are defined below.

Definition (*computation run*)
A computation run $\texttt{CR}(s_0)$ for a Dribble agent with goal rules Γ and PR rules Δ is a finite or infinite sequence $s_0, \ldots, s_n$ or $s_0, \ldots$ where:

- $s_i \in \Sigma$ are mental states,
- $\forall_{i>0} : \ s_{i-1} \rightarrow_x s_i$ is a transition in the transition system for the Dribble agent.

The meaning of a Dribble agent can now be defined in terms of a set of computation runs.

Definition (*semantics of a Dribble agent*)
The meaning of a Dribble agent $\langle \sigma_0, \gamma_0, \Gamma, \Delta \rangle$ is the set of computation runs $\texttt{CR}(\langle \sigma_0, \gamma_0, E \rangle)$.

Note that the first state of the computation runs is the initial mental state of the Dribble agent as defined in section 2.6. As was explained in sections 3.2.1 and 3.2.2, the transition rules for goal rule application and PR rule application do not specify which rule to apply, in case more than one rule is applicable in a mental state. The transition rules neither specify whether to apply a rule or to execute an action, if both are possible in a mental state. Any implementation of the language Dribble has to deal with how to reduce this non-determinism in the semantics of the language. Control structures are needed to determine whether an action should be executed or whether a rule should be applied and possibly which rule should be applied. I will however not discuss these control structures in this paper.

Chapter 4

Example

In this chapter, I give an example to illustrate how the programming language Dribble can be used to program agents. The example agent has to solve the problem of building a tower of blocks. The blocks have to be stacked in a certain order. In the example, I use variables as a means for abbreviation. Variables should be thought of as being instantiated with the relevant arguments in such a way that predicates with variables reduce to propositions.

I will first show how the tower building agent can be programmed in GOAL and then how it can be programmed in Dribble. I will compare the languages GOAL and Dribble with respect to the example. Then I will show how the tower building agent can be programmed in 3APL, followed by a comparison of the languages Dribble and 3APL. At the end of this section I will give a second way of programming a tower building agent in Dribble.

4.1 Introduction

The only action an agent can take, is to move a block x from some block y to another block z ($move(x, y, z)$). The action is enabled only if the block to be moved (x) and the block to which x is moved (z) are clear. The floor (Fl) is treated as a block that is always clear. The predicate $on(x, y)$ is used to denote that a block x is on block y. The predicate $clear(x)$ is used to denote that block x is clear.

I assume that the function $\mathcal{T}(move(x, y, z), \sigma)$ which specifies the changes to belief base σ if action $move(x, y, z)$ is executed, has the following properties. The function is defined only if σ implies $clear(x) \wedge clear(z) \wedge on(x, y)$. The result is a belief base that implies $on(x, z) \wedge clear(y) \wedge \neg on(x, y) \wedge \neg clear(z)$ if $z \neq Fl$. If $z = Fl$, the resulting belief base implies $on(x, z) \wedge clear(y) \wedge \neg on(x, y)$.

$$\mathcal{T}(move(x, y, z), \sigma) = \sigma' \qquad \Leftrightarrow \quad \sigma \models clear(x) \wedge clear(z) \wedge on(x, y)$$

$$\begin{array}{lll} \mathcal{T}(move(x, y, z), \sigma) = \sigma' \ \wedge \ z \neq Fl & \Rightarrow & \sigma' \models on(x, z) \wedge clear(y) \wedge \neg on(x, y) \wedge \neg clear(z) \\ \mathcal{T}(move(x, y, z), \sigma) = \sigma' \ \wedge \ z = Fl & \Rightarrow & \sigma' \models on(x, z) \wedge clear(y) \wedge \neg on(x, y) \end{array}$$

Above, the changes to the predicates $on(x, y), on(x, z), clear(y)$ and $clear(z)$ which are caused by the action $move(x, y, z)$, are specified. Now, I will specify which predicates are not changed by the action $move(x,y,z)$, assuming that the function $\mathcal{T}(move(x,y,z),\sigma)$ is defined: $\mathcal{T}(move(x,y,z),\sigma) = \sigma'$.

$$\begin{array}{llll}
\sigma \models on(u,v) & \wedge \quad u \neq x & \Rightarrow & \sigma' \models on(u,v) \\
\sigma \models \neg on(u,v) & \wedge \quad ((u \neq x) \vee (v \neq z)) & \Rightarrow & \sigma' \models \neg on(u,v) \\
\\
\sigma \models clear(w) & \wedge \quad w \neq z & \Rightarrow & \sigma' \models clear(w) \\
\sigma \models \neg clear(w) & \wedge \quad w \neq y & \Rightarrow & \sigma' \models \neg clear(w)
\end{array}$$

For the example programs in the different languages, the following initial belief base and goal base are used (see figure 4.1).

$$\begin{array}{lll}
\sigma_0 & = & \{on(C,A) \wedge on(A,Fl) \wedge on(B,Fl) \wedge clear(B) \wedge clear(C) \wedge clear(Fl), \\
 & & on(x,y) \rightarrow \neg clear(y)\} \\
\gamma_0 & = & \{on(A,B) \wedge on(B,C) \wedge on(C,Fl)\}
\end{array}$$

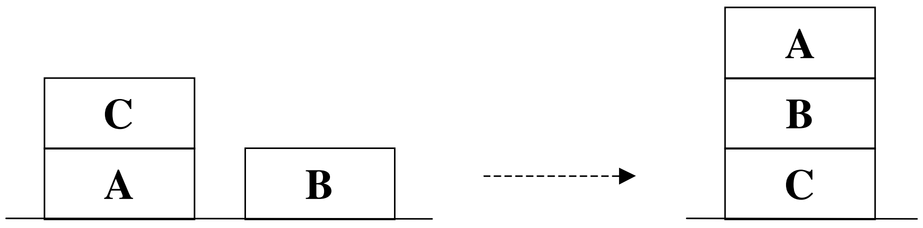

Figure 4.1: Initial beliefs and goal

4.2 GOAL

4.2.1 The rules

In GOAL, the tower building agent can be programmed with the following goal rules.

$$\begin{array}{llll}
g_1 : \mathbf{B}(on(x,y) \wedge clear(x) \wedge clear(z)) & \wedge \quad \mathbf{G}(on(x,z)) & \rightarrow & move(x,y,z) \\
g_2 : \mathbf{B}(on(x,y) \wedge \neg clear(x)) & \wedge \quad \mathbf{G}(on(x,z)) & \rightarrow & adopt(\mathbf{G}(clear(x))) \\
g_3 : \mathbf{B}(on(x,y) \wedge \neg clear(z)) & \wedge \quad \mathbf{G}(on(x,z)) & \rightarrow & adopt(\mathbf{G}(clear(z))) \\
\\
g_4 : \mathbf{B}(on(x,y) \wedge clear(x)) & \wedge \quad \mathbf{G}(clear(y)) & \rightarrow & move(x,y,Fl) \\
g_5 : \mathbf{B}(on(x,y) \wedge \neg clear(x)) & \wedge \quad \mathbf{G}(clear(y)) & \rightarrow & adopt(\mathbf{G}(clear(x)))
\end{array}$$

The consequent of a goal rule in GOAL is a basic action. In contrast with Dribble, plans cannot be used as the consequent of a goal rule. Furthermore, the special actions $adopt(\mathbf{G}(\psi))$ and $drop(\mathbf{G}(\psi))$ can be used in GOAL. These actions respectively add or delete the goal ψ from the goal base. Rule g_1 specifies that a $move(x,y,z)$ action to fulfil a goal $on(x,z)$ can be executed if the preconditions for the action are satisfied. If the precondition that x is clear or z is clear is not satisfied, rules g_2 or g_3 can be used to adopt the new goal that these blocks should become clear. A goal $clear(y)$ can be satisfied by executing rule g_4, which specifies that the block that is on top of block y is moved to the floor. This move action can only be executed if the block on top of y is clear. If it is not clear, rule g_5 can be used to adopt the goal to clear block x that is on top of block y.

Suppose for instance that the agent believes that blocks A, B, C and D are stacked in that order with A on top and D at the bottom (see figure 4.2). Suppose the agent has a goal to clear block D. Rule g_4 cannot be used to move C to the floor, because C is not clear. The agent could then adopt the goal to clear C with rule g_5. Block B that is on top of C cannot yet be moved to the floor to clear C, so the goal to clear B can be adopted with rule g_5. Now block A *can* be moved to the floor to clear B, using rule g_4. After moving A to the floor, B and C can be moved to the floor successively using rule g_4, satisfying the initial goal to clear block D.

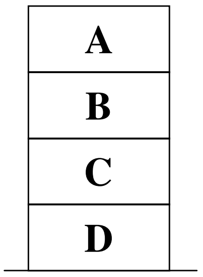

Figure 4.2: Tower

4.2.2 Example execution

Suppose the move action and the initial belief base and goal base are as specified in section 4.1. Note that the goal $on(A,B)$ for example is a logical consequence of the goal in the goal base. If a goal rule specifies a condition $\mathbf{G}(on(A,B))$, this rule is applicable in the initial mental state: the goal $\mathbf{G}(on(A,B))$ is implied by a goal in the goal base and it is not implied by the belief base (see section 3.1). In the initial mental state of the agent $\langle\sigma_0, \gamma_0\rangle$, the instantiation of rule g_1 with $x = B,\ y = Fl,\ z = C$ could be applied (yielding $move(B, Fl, C)$). Another option is applying an instantiation of rule g_1 with $x = C,\ y = A,\ z = Fl$ (yielding $move(C, A, Fl)$). Yet another rule that can be applied in the initial mental state is the instantiation of rule g_2 with $x = A,\ y = Fl,\ z = B$ ($adopt(\mathbf{G}(clear(A)))$). Suppose the second instantiation of rule g_1 is executed to satisfy the goal $on(C, Fl)$. The new mental state of the agent satisfies the following specifications (see figure 4.3):

$$\begin{aligned}\sigma_1 &\models on(A,Fl) \wedge on(B,Fl) \wedge on(C,Fl) \wedge clear(A) \wedge clear(B) \wedge clear(C) \wedge clear(Fl),\\ \gamma_1 &= \{on(A,B) \wedge on(B,C) \wedge on(C,Fl)\}.\end{aligned}$$

In this mental state, the instantiation of rule g_1 with $x = B,\ y = Fl,\ z = C$ is a rule that could be applied (yielding $move(B, Fl, C)$). The mental state resulting from application of this rule satisfies the following:

$$\begin{aligned}\sigma_2 &\models on(A,Fl) \wedge on(B,C) \wedge on(C,Fl) \wedge clear(A) \wedge clear(B) \wedge clear(Fl),\\ \gamma_2 &= \{on(A,B) \wedge on(B,C) \wedge on(C,Fl)\}.\end{aligned}$$

The only rule that can be applied in this mental state, is the instantiation of rule g_1 with $x = A,\ y = Fl,\ z = B$ (yielding $move(A, Fl, B)$):

$$\begin{aligned}\sigma_3 &\models on(A,B) \wedge on(B,C) \wedge on(C,Fl) \wedge clear(A) \wedge clear(Fl),\\ \gamma_3 &= \emptyset.\end{aligned}$$

This example execution shows, that it is possible for the specified GOAL agent to reach its initial goal.

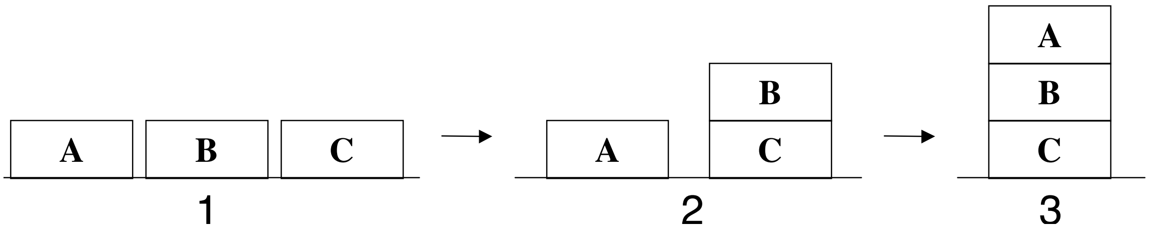

Figure 4.3: Example execution

4.2.3 A problem: cycles

Although the agent *could* reach its goal if the specified instantiations of rules are executed, it is unfortunately *not* the case that it *will* always reach its goal if other instantiations are applied. The problem is, that the agent could get into a cycle.

Suppose that all blocks are on the floor (mental state 1 of section 4.2.2). In section 4.2.2 block B is moved onto C. The agent could however instead execute the action $move(A, Fl, B)$ to satisfy the goal $on(A, B)$ (see figure 4.4). In the mental state in which A is on B and C is on the floor, the only part of the initial goal that is not satisfied is $on(B, C)$. To satisfy this goal, rule g_2 is applied to adopt the goal $clear(B)$. Then rule g_4 is applied and A is moved to the floor (yielding $move(A, B, Fl)$). In this mental state, all blocks are on the floor again and the only goal in the goal base is the initial goal. The mental state is thus equal to mental state number 1 (see section 4.2.2). The agent now has two options: it can move A onto B again or it can move B onto C. If the agent decides somehow to move block A, the agent enters a cycle. The only way to escape from this cycle is to move B onto C in mental state number 1. This shows that the agent will not always reach its initial goal. If it chooses to move A onto B every time it is in mental state 1, it will keep moving around in circles, not realising its goal.

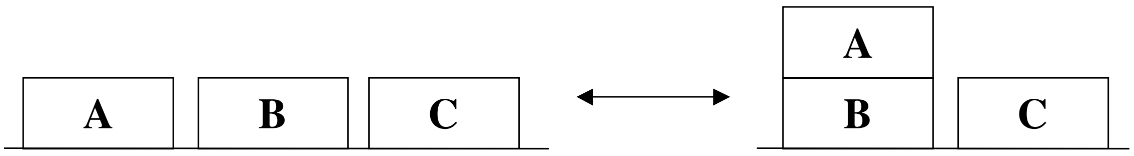

Figure 4.4: Cycle

This problem could possibly be solved by letting the agent keep track of which rules it executes in which mental state. It would record a graph as shown in figure 4.5. The nodes in the figure indicate the different mental states and the arrows denote mental state transitions. Mental state number 1 is the initial mental state and number 9 is the goal state (these numbers do not correspond with the numbers of the example execution above). If more than one arrow leaves from a node, multiple different rules are applicable in that state. Mental states 7, 10 and 11 are an example of a cycle. The agent could apply a rule in state number 1, resulting in state number 7. In this state, it could apply a rule resulting in state number 10. Then it will have to go to 11 and back to 7 again. The agent should now apply a different rule, resulting in state number 8 instead of 10. It should thus if possible apply a rule it has not applied yet. In this way, the agent will not move around in circles. The problem would however be to determine whether two mental states are equal.

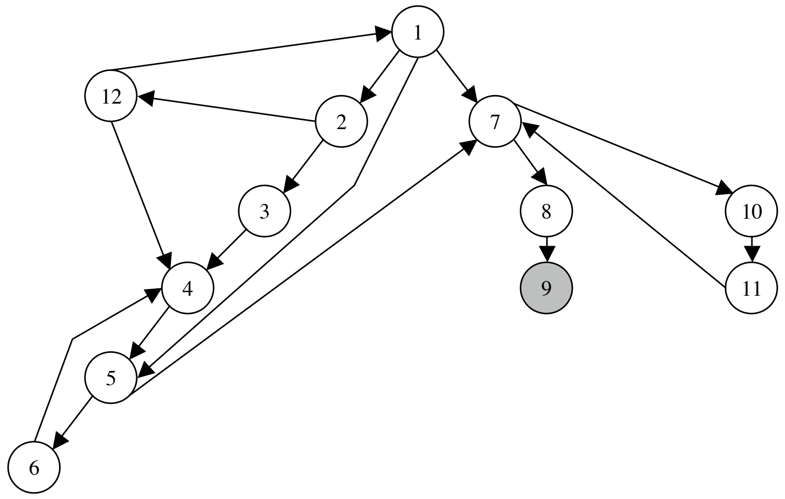

Figure 4.5: State diagram GOAL

4.3 Dribble

4.3.1 The rules

A Dribble agent can solve the tower building problem with the following rules.

$$\Gamma = \{ \; g_1 : \mathbf{B}(on(x,y)) \wedge \mathbf{G}(on(x,z)) \rightarrow move(x,y,z) \; \}$$

$$\begin{aligned}\Delta = \{ \; & \rho_1 : move(x,y,z) \mid \mathbf{B}(\neg clear(x) \wedge on(a,x)) \rightarrow move(a,x,Fl); move(x,y,z), \\ & \rho_2 : move(x,y,z) \mid \mathbf{B}(\neg clear(z) \wedge on(a,z)) \rightarrow move(a,z,Fl); move(x,y,z) \; \}\end{aligned}$$

The goal rule is used to derive the $move(x,y,z)$ action that should be executed to fulfil a goal $on(x,z)$. The preconditions of the move action are not checked in this rule, so it is possible that the derived action cannot be executed in a particular mental state. The PR rules can then be used to create a mental state in which this action *can* be executed.

PR rule ρ_1 can be applied if the condition that x is clear is not satisfied. Rule ρ_2 can be applied if z is not clear. The PR rules with head $move(x,y,z)$ construct a plan to create a mental state in which this action can be executed. Rule ρ_1 for example specifies that if x is not clear because $on(a,x)$, a $move(x,y,z)$ action should be replaced by the plan

$move(a, x, Fl); move(x, y, z)$: first clear x, then move x. If in turn the action $move(a, x, Fl)$ cannot be executed because there is some block b on top of x, the plan could be changed to $move(b, a, Fl); move(a, x, Fl); move(x, y, z)$: clear a, clear x and then move x.

4.3.2 Example execution

Suppose the move action and the initial belief base and goal base are as specified in section 4.1.

In the initial mental state of the agent $\langle\sigma_0, \gamma_0, E\rangle$, three possible instantiations of goal rule g_1 could be applied: $x = A,\ y = Fl,\ z = B$ or $x = B,\ y = Fl,\ z = C$ or $x = C,\ y = A,\ z = Fl$ (yielding $move(A, Fl, B), move(B, Fl, C)$ or $move(C, A, Fl)$). Suppose the first instantiation is chosen. After application of this goal rule, the plan of the agent becomes equal to the plan in the consequent of the rule, resulting in the following mental state (see figure 4.6):

$$\begin{array}{lcl} \sigma_1 & = & \{on(A, Fl) \wedge on(B, Fl) \wedge on(C, A) \wedge clear(B) \wedge clear(C) \wedge clear(Fl)\}, \\ \gamma_1 & = & \{on(A, B) \wedge on(B, C) \wedge on(C, Fl)\}, \\ \pi_1 & = & move(A, Fl, B). \end{array}$$

The plan cannot be executed because the preconditions of the action are not satisfied in this mental state. The goal rule cannot be applied because the plan of the agent is not empty. The only applicable rule is the PR rule ρ_1:

$$\begin{array}{lcl} \sigma_2 & = & \{on(A, Fl) \wedge on(B, Fl) \wedge on(C, A) \wedge clear(B) \wedge clear(C) \wedge clear(Fl)\}, \\ \gamma_2 & = & \{on(A, B) \wedge on(B, C) \wedge on(C, Fl)\}, \\ \pi_2 & = & move(C, A, Fl); move(A, Fl, B). \end{array}$$

The only option is to execute the first action of the plan:

$$\begin{array}{lcl} \sigma_3 & \models & on(A, Fl) \wedge on(B, Fl) \wedge on(C, Fl) \wedge clear(A) \wedge clear(B) \wedge clear(C) \wedge clear(Fl), \\ \gamma_3 & = & \{on(A, B) \wedge on(B, C) \wedge on(C, Fl)\}, \\ \pi_3 & = & move(A, Fl, B). \end{array}$$

The only option is to execute π_3:

$$\begin{array}{lcl} \sigma_4 & \models & on(A, B) \wedge on(B, Fl) \wedge on(C, Fl) \wedge clear(A) \wedge clear(C) \wedge clear(Fl), \\ \gamma_4 & = & \{on(A, B) \wedge on(B, C) \wedge on(C, Fl)\}, \\ \pi_4 & = & E. \end{array}$$

The only applicable rule instantiation is g_1 with $x = B,\ y = Fl,\ z = C$, because the goal $on(B, C)$ is the only (logical consequence of the) goal that is not satisfied:

$$\begin{array}{lcl} \sigma_5 & \models & on(A, B) \wedge on(B, Fl) \wedge on(C, Fl) \wedge clear(A) \wedge clear(C) \wedge clear(Fl), \\ \gamma_5 & = & \{on(A, B) \wedge on(B, C) \wedge on(C, Fl)\}, \\ \pi_5 & = & move(B, Fl, C). \end{array}$$

In this mental state, a plan is constructed using rule ρ_1 to move B onto C: $move(A, B, Fl); move(B, Fl, C)$. The previous move $move(A, Fl, B)$ is undone. The plan is executed, resulting in the following mental state:

$$\begin{array}{lcl} \sigma_7 & \models & on(A, Fl) \wedge on(B, C) \wedge on(C, Fl) \wedge clear(A) \wedge clear(B) \wedge clear(Fl), \\ \gamma_7 & = & \{on(A, B) \wedge on(B, C) \wedge on(C, Fl)\}, \\ \pi_7 & = & E. \end{array}$$

The only applicable rule is g_1, resulting in adoption of the plan $move(A, Fl, B)$. This plan is executed:

$$\begin{array}{lcl} \sigma_8 & \models & on(A,B) \wedge on(B,C) \wedge on(C,Fl) \wedge clear(A) \wedge clear(Fl), \\ \gamma_8 & = & \emptyset, \\ \pi_8 & = & E. \end{array}$$

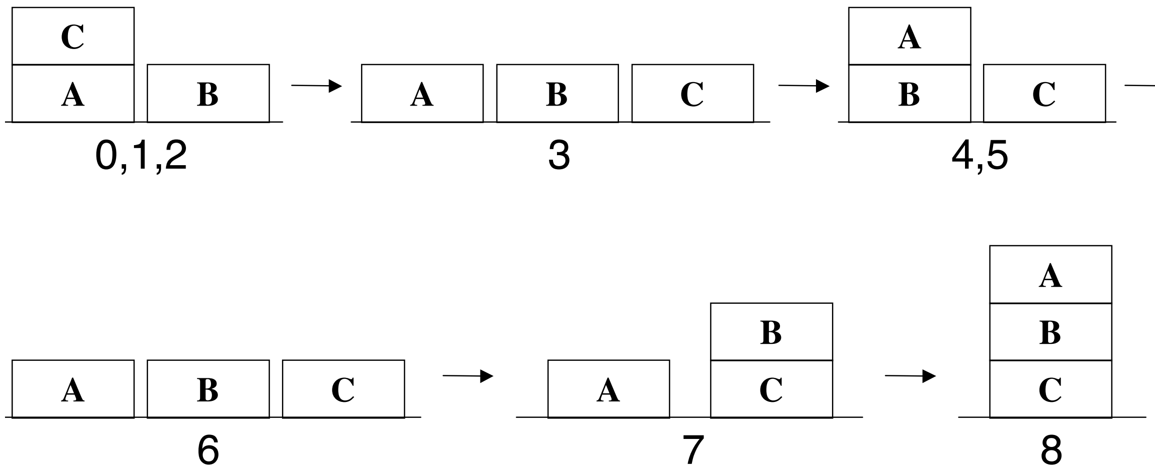

Figure 4.6: Example execution

This example execution shows that it is possible for the Dribble agent to reach its initial goal. Reaching the goal is however not only a possibility, but the example Dribble agent *will always* reach its initial goal. This is illustrated in figure 4.7, which shows the transition diagram for the example Dribble agent. Mental state number 1 is the initial mental state and number 11 is the goal state (these numbers do not correspond with the numbers of the example execution above). As is shown by the transition diagram, it does not matter which instantiation of a rule is chosen in a mental state. The agent will eventually reach its goal (although some paths to the goal state are shorter than others). This is partially due to the fact that the diagram does not contain cycles.

I will now explain why the goal specified in the goal rule is $on(x, y)$, instead of for instance $on(x_1, y_1) \wedge on(x_2, y_2)$. The reason is that a goal rule with only a goal $on(x, y)$ is more general. It can be used to help in satisfying any goal $on(x_1, y_1) \wedge \ldots \wedge on(x_n, y_n)$. Furthermore, it is not clear what the consequent of a goal rule with goal $on(x_1, y_1) \wedge on(x_2, y_2)$ should be. The consequent could be two move actions, but the order in which the move actions should be executed cannot be determined in general. The second move can undo the first move, so the result of executing these move actions will not always be that both goals are fulfilled. Thus this example shows that it is convenient that the meaning of goal formulas is defined the way it is (see section 3.1).

The example also illustrates how PR rules can be used to modify plans. The guard of the rule describes a situation in which the move action cannot be executed (or would fail). The PR rules are used to replace the move action by a plan that eventually makes it possible to execute the move action that could not be executed at first. In summary, the goal rule in the example is used to select a plan to satisfy a goal. The PR rules are used to modify this plan if the initally selected plan cannot be executed yet.

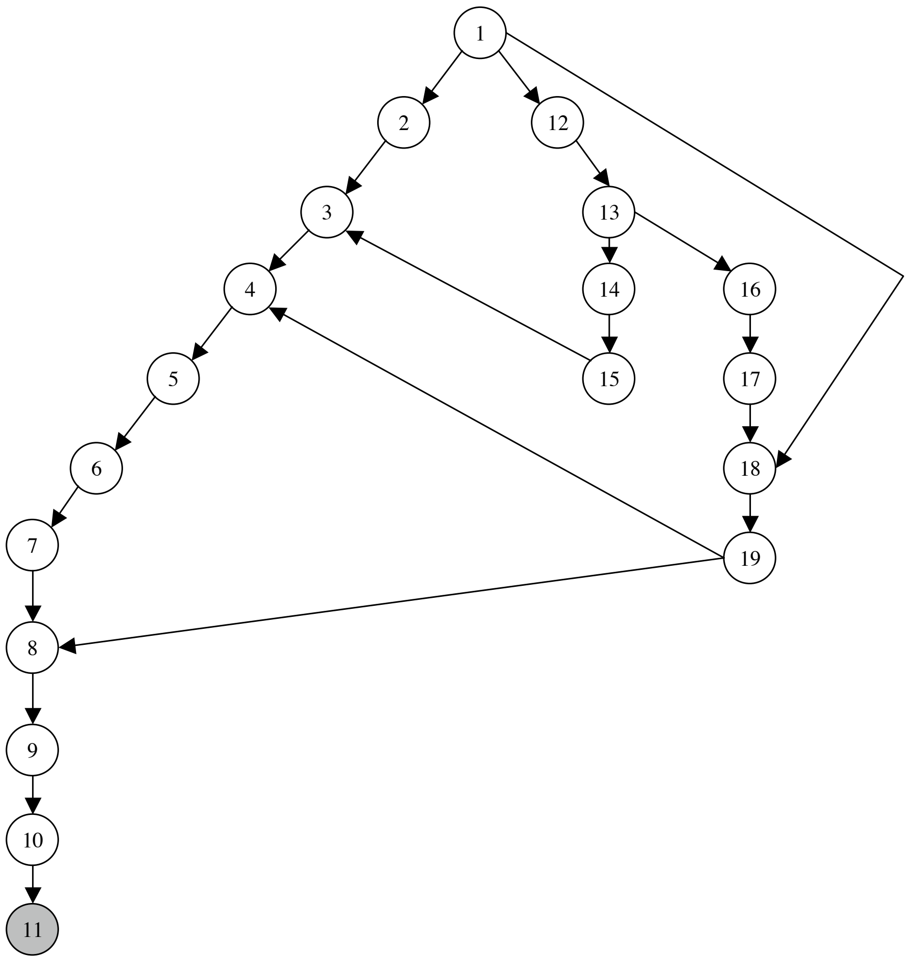

Figure 4.7: State diagram Dribble

4.4 GOAL versus Dribble

The main difference between GOAL and Dribble is that the mental state of a GOAL agent only contains beliefs and goals, whereas the mental state of a Dribble agent also contains a plan component. The idea of the way in which the agents are programmed in the languages GOAL and Dribble to solve the tower building problem is the same for both languages. Move actions are derived to realise parts (logical consequences) of the initial goal of the agent. If a move action that directly realises a goal cannot be executed, the conditions under which the action can be executed have to be created. In GOAL this is done by means of deriving new goals. These goals have to be satisfied before the action to realise a part of the initial goal can be executed. In Dribble it is done by constructing a plan to create the conditions for the action. The plan is constructed using PR rules.

The biggest difference between the two example agents is, that the Dribble agent will always reach its goal, but for the GOAL agent this is not guaranteed. The GOAL agent could stay in a cycle not realising its initial goal, but the Dribble agent will always eventually end up on the path leading to the goal state.

The reason that the Dribble agent will not get into cycles is that it will always eventually execute a derived move action. If the Dribble agent decides to try to realise the goal $on(A, B)$, the action $move(A, x, B)$ is recorded as the plan of the agent (applying goal rule g_1 of section 4.3.1). The PR rules can modify this plan, but the action $move(A, x, B)$ will remain the last action of the plan until it is executed. The way in which the PR rules are programmed, ensures this. If $move(A, x, B)$ is adopted as the plan, the Dribble agent will eventually execute this action.

A GOAL agent does not have a plan component in its mental state. In contrast with a Dribble agent, a GOAL agent therefore cannot create and modify a plan. The GOAL agent executes the action in the consequent of a rule as soon as the rule is applied. The conditions in the antecedent of the rule should make sure that the rule is only selected if the preconditions for the action are satisfied. The rules that adopt new goals and the rules that achieve these goals, make sure that these preconditions are realised. Nothing however guarantees that the rule whose preconditions have just been satisfied, will be executed: in that state another rule could be applicable as well.

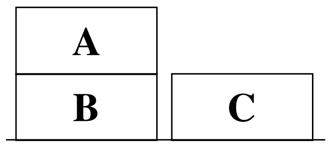

Figure 4.8: Example

Suppose for instance that block A is on B and block B and C are on the floor (see figure 4.8). The only part of the initial goal that is not satisfied is $on(B, C)$. The idea of for instance rule g_2 of section 4.2.1 is the following: if the agent has a goal $on(B, C)$ but B is not clear, it should adopt the goal to clear B. This block has to be cleared, because then the goal $on(B, C)$ can be realised through application of an instantiation of rule g_1. It is however not recorded that the situation in which B is clear has to be realised *because* then the goal $on(B, C)$ can be realised. If the goal $clear(B)$ is realised, all blocks are on the floor and the only goal of the agent is the initial goal. In this situation, the agent should execute the instantiation of g_1 which moves B onto C, because B was cleared to satisfy the preconditions for this action. The agent could however instead execute the action $move(A, Fl, B)$, because the preconditions for this action are also realised. If it does this, it will get into a state it was in before. If the agent keeps on doing this, it will keep moving around in circles and it will not reach its goal.

In summary, the problem in GOAL is that there is no direct relation between the newly adopted goals and the initial goals. The state that is described by a new goal is supposed to be a state (sub-state) on the way to a specific end-goal. If this sub-state is realised however, nothing guarantees that the specific end-goal for which it was created, is realised next. Instead of applying the specific rule realising the end-goal, just as well another applicable rule could be applied. In Dribble the end-goal (which is a part of the initial goal) will always be reached. The reason is, that the move action realising this goal is recorded and will be executed eventually. The agent does not reach some sub-state in which it could go into a direction away from the end-state for which it was created.

One reason that the Dribble agent will always reach its initial goal is that it cannot get into a cycle. The other reason is that all paths lead to the goal (see figure 4.7). There is not a path with a dead end from which the initial goal cannot be reached anymore. This is due to the fact that actions can always be reversed. This is a characteristic of the problem domain. Suppose for example that the domain is such, that the agent can fall into a deep pit from which it cannot get out. Entering the branch of the state diagram leading to this situation would be fatal. The agent would not be able to get onto the path leading to the goal state (assuming that falling into the pit is not a goal of the agent....). In the domain of the pit, choosing the right applicable rule will not only speed up the process of reaching the goal, but it is also crucial for the agent to reach the goal at all.

A last remark has to be made about the GOAL agent. Although the state diagram (figure 4.5) contains cycles, there is always the possibility of getting out of a cycle onto the path leading to the goal. The agent just has to choose the right rule for application.

4.5 3APL

In 3APL one cannot use declarative goals. The only available rules are the PR rules. With the following PR rules, the tower building problem can be solved.

$$\begin{array}{llll}
\rho_1 : on(x,z) & \mid \ \mathbf{B}(on(x,y)) & \rightarrow & move(x,y,z) \\
\rho_2 : move(x,y,z) & \mid \ \mathbf{B}(\neg clear(x) \wedge on(a,x)) & \rightarrow & move(a,x,Fl); move(x,y,z) \\
\rho_3 : move(x,y,z) & \mid \ \mathbf{B}(\neg clear(z) \wedge on(a,z)) & \rightarrow & move(a,z,Fl); move(x,y,z)
\end{array}$$

A 3APL agent gets an initial plan, comparable to the initial goal of a Dribble agent. The initial plan would have to be $on(C,Fl); on(B,C); on(A,B)$. Suppose that the move action and the initial belief base of the agent are as specified in section 4.1. With rule ρ_1 the action $move(C,A,Fl)$ is derived. The plan becomes $move(C,A,Fl); on(B,C); on(A,B)$. The first action of the plan is executed, after which it is removed from the plan. Then rule ρ_1 is applied again, resulting in the following plan: $move(B,Fl,C); on(A,B)$. The move action at the head of the plan is executed. Rule ρ_1 has to be applied one more time and the plan of the agent becomes the following: $move(A,Fl,B)$. This action is executed after which the plan is empty.

The plan $on(C,Fl); on(B,C); on(A,B)$ is a sequence of abstract plans. After execution of for instance the abstract plan $on(C,Fl)$, the agent believes that C is on the floor: $\mathbf{B}$(on(C,Fl)). In this case, the assertional reading of an abstract plan that was referred to in section 2.2.2, is valid. After execution of the abstract plan or achievement goal as it is called in 3APL, the agent believes the goal to be true.

In the following section, the differences between 3APL and Dribble are discussed using the example programs of the tower building agent in both languages.

4.6 3APL versus Dribble

The main difference between a 3APL agent and a Dribble agent is, that the mental state of the 3APL agent has a belief component and a plan component, whereas the mental state of the Dribble agent also contains goals. The question is, whether these goals really add something. I will try to answer this question by trying to *implement* goals in the language 3APL. If this could be done, we would not need Dribble and 3APL would be enough.

Implementing goals in 3APL could be done in two ways: using the plan structures of 3APL or using the belief structures. In the example program in section 4.5, the plan structures are used. Achievement goals as they are called in 3APL (abstract plans as they are called in this paper) fulfil the role of declarative goals. Implementing goals in the belief base could be done by letting the agent believe a predicate $goal(x)$.

Goals in Dribble have several properties:

- one goal can be a logical combination of propositions,
- all logical consequences of a goal in the goal base are also goals,
- goals are removed if the agent believes the goal situation to be the case.

These properties would have to be implemented in 3APL. I will explain in the following two sections whether this can or cannot be done using plan structures or belief structures respectively.

4.6.1 Using plan structures to implement goals in 3APL

PR rule ρ_1 of section 4.5 has a close resemblance with the goal rule of section 4.3.1:

$$\begin{array}{llll}\rho_1 : on(x,z) & \mid \ \mathbf{B}(on(x,y)) & \rightarrow & move(x,y,z), \\ g_1 : \mathbf{B}(on(x,y)) & \wedge \ \mathbf{G}(on(x,z)) & \rightarrow & move(x,y,z).\end{array}$$

In 3APL, the achievement goal $on(x,z)$ is used as a declarative goal. The properties of this achievement goal in 3APL are however different from those of the declarative goal $on(x,z)$ in Dribble.

- A goal $on(A,B) \wedge on(B,C)$ cannot be expressed in 3APL because logical operators are not allowed in achievement goals.
- Because a goal $on(A,B) \wedge on(B,C)$ cannot be expressed in 3APL, this would have to be translated into two separate goals $on(A,B)$ and $on(B,C)$. If one wants to program a 3APL agent that realises these goals, the order in which these goals have to be realised has to be specified by the programmer. In the example 3APL program in section 4.5 for instance, the initial plan of the agent is $on(C,Fl);on(B,C);on(A,B)$. This plan however has a different meaning than the goal $on(A,B) \wedge on(B,C) \wedge on(C,Fl)$. The three conjuncts of the goal have to be realised *at the same time*. The three achievement goals of the plan can only be realised *in sequence*.
- A goal of a Dribble agent is removed from its goal base only if it is achieved. An achievement goal is removed from the plan if a rule is applied to translate the goal into an action. This action should achieve the goal. There is however no guarantee that the agent will actually have reached the goal after execution of the action, yet the goal will be gone anyway. This shows an advantage of the clear separation between beliefs,

goals and plans which is realised in Dribble. The relation between beliefs and goals in Dribble provides for a kind of maintenance goal. If one conjunct of a goal is reached in a mental state, it is not a goal in that state. If in the next mental state the conjunct is not believed by the agent anymore, it becomes a goal again. This is useful in the domain of building a tower: actions to realise one part of the initial goal, often undo already realised other parts. Only if the entire goal is believed by the agent, it is removed from the goal base. In 3APL, achievement goals are part of the plan. If they are translated into actions and if these actions are executed, the goals are gone. If after execution the achievement goal is not reached, it will have to be explicitly added to the plan again.

- In 3APL, the programmer specifies the order in which the achievement goals should be reached. This order cannot always be any order. In the example of tower building, the agent should start at the bottom of the tower and end at the top. In 3APL the programmer of the agent would therefore have to think about the way a certain tower is built for every tower the agent should build. In Dribble the programmer can specify the goal state and the agent has to figure out how to reach this state. In 3APL, the programmer would either need to have knowledge about the initial beliefs of the agent, or a rule would have to be specified that removes an achievement goal from the plan if it is believed by the agent.

  Suppose for example that the initial mental state of the agent is, that block C is on the floor, B is on C and A is on the floor (figure 4.9). The goal is as specified in section 4.1. The agent only has to execute the action $move(A, Fl, B)$ to reach its goal. The programmer should now either specify the initial plan of the agent to be $on(A, B)$, or specify a rule that would drop the achievement goal $on(x, y)$ if the agent believes this to be the case.

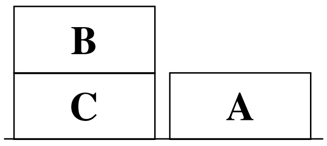

Figure 4.9: Example

- In the previous item I explained the disadvantage of letting the programmer specify the order in which the achievement goals should be reached. There can however also be an advantage to this. There is no guarantee that the Dribble agent will choose the shortest path to the goal state. Some actions may be superfluous. The advantage of letting the programmer specify the order now is, that the agent will follow the shortest path to the goal state (assuming the programmer did a good job, see also section 4.8).

In summary, being able to specify declarative goals in an agent programming language can be an advantage. These goals and the meaningful relation between beliefs and goals cannot be replaced by achievement goals in 3APL. This is due to the different meaning of declarative goals which are a separate component of the agent and achievement goals, which are part of the plan of the agent.

The example of the tower building agent is a good example of an agent for which it is intuitive and useful to specify a goal state. A goal has convenient properties. Implementing the agent without using declarative goals leads to a less intuitive specification and extra rules have to be specified to realise the desired behaviour. For other agents it may be more natural to specify an initial plan instead of an initial goal and think of the agent in terms of actions it should execute instead of goals it should reach. This can be the case if it is difficult to describe the desired goal state.

4.6.2 Using belief structures to implement goals in 3APL

Instead of letting achievement goals fulfil the role of declarative goals, one could try to use the beliefs of the agent to implement the concept of declarative goals. An example of the use of beliefs as goals in a PR rule ρ is demonstrated below. The PR rule has a close resemblance with goal rule g_1 of section 4.3.1:

$$\begin{array}{llll} \rho: & \mid \mathbf{B}(on(x,y)) \;\wedge & \mathbf{B}(goal(on(x,z))) & \rightarrow \; move(x,y,z), \\ g_1: & \mathbf{B}(on(x,y)) \;\wedge & \mathbf{G}(on(x,z)) & \rightarrow \; move(x,y,z). \end{array}$$

The properties of the belief-goal $\mathbf{B}(goal(on(x,z)))$ are however different from the goal $\mathbf{G}(on(x,z))$ in Dribble.

- A belief-goal cannot include logical operators. One could use predicates to describe a logical combination of propositions, for instance $goal(and(on(A,B),on(B,C)))$. The meaning of these predicates however would have to be defined. This is not trivial.
- A goal in Dribble is removed if the agent believes that the goal is achieved. In 3APL this removal would have to be done explicitly in definitions of actions.
- Because a belief-goal cannot include logical operators, the mechanism of maintenance goals that was explained in the previous section, is not supported by belief-goals. If a belief-goal is removed from the belief base because it is achieved, it will not return. One could try to use action specifications to let maintenance goals return if necessary. It would however be very difficult to specify the conditions under which they should return and it would make the specifications of actions very counterintuitive.

Concluding, belief-goals cannot replace the declarative goals of Dribble easily.

4.7 The benefit of being able to adopt goals

In section 4.2.3 it was explained that programming the tower building agent in GOAL with goal rules that adopt new goals, can lead to problems. The agent could stay in a cycle, not reaching its initial goal. Being able to adopt goals in the goal rules can however be beneficial. That will be explained in this section. The special actions to adopt and drop goals in the consequent of goal rules are not part of the language Dribble. More research is needed to extend the language with these actions. In this section I will nevertheless show that it can be beneficial to use goal adoption in goal rules and that this research could thus be useful.

4.7.1 Sub-goals

An agent realises a goal in a sequence of steps. After each step, the world is in a certain state (the agent believes the world to be in a certain state). As in section 4.4, I call these states on the way to the end-goal, sub-states. If these sub-states are adopted as goals, I call them sub-goals. An agent can execute a plan, while these sub-states are passed during execution of the plan. This was done in the example Dribble agent. The agent could however instead be programmed with goal rules that adopt these sub-states as goals if these rules are executed. Other goal rules should specify the plan (or basic action in case of GOAL) to reach these sub-goals. This was done in the example GOAL agent. The difference is, that the GOAL agent can decide once more what to do if it is in some sub-state. The Dribble agent already has a plan to be executed in a certain sub-state.

For the example domain of building towers, being able to decide once more what to do in a realised sub-state does not lead to desirable behaviour. In this domain, usually multiple actions can be executed in a sub-state. Executing the wrong one again and again results in cyclic behaviour. In other domains however in which these cycles are not likely to occur, adoption of new goals can be beneficial. Using the adoption of goals, the goal rules can be specified more *general* than would be possible without adoption of goals. Specification of an agent with these rules also leads to an agent that is more *autonomous* than a 3APL agent.

4.7.2 The rules with adoption of goals

I will try to explain the benefit of being able to adopt goals, using the following goal rules.

$$\begin{array}{llcllcl}
g_1: & \mathbf{B}(\phi_1) & \wedge & \mathbf{G}(\phi_2) & \rightarrow & \pi_1 \\
\\
g_2: & \mathbf{B}(\phi_2) & \wedge & \mathbf{G}(\phi_3) & \rightarrow & \pi_2 \\
g_2': & \neg\mathbf{B}(\phi_2) & \wedge & \mathbf{G}(\phi_3) & \rightarrow & \mathit{adopt}(\mathbf{G}(\phi_2)) \\
\\
g_3: & \mathbf{B}(\phi_3) & \wedge & \mathbf{G}(\phi_4) & \rightarrow & \pi_3 \\
g_3': & \neg\mathbf{B}(\phi_3) & \wedge & \mathbf{G}(\phi_4) & \rightarrow & \mathit{adopt}(\mathbf{G}(\phi_3)) \\
\\
g_4: & \mathbf{B}(\phi_4) & \wedge & \mathbf{G}(\phi_5) & \rightarrow & \pi_4 \\
g_4': & \neg\mathbf{B}(\phi_4) & \wedge & \mathbf{G}(\phi_5) & \rightarrow & \mathit{adopt}(\mathbf{G}(\phi_4))
\end{array}$$

The idea is that a plan π_i reaches the goal ϕ_{i+1} if it is executed (see figure 4.10). I assume that this is indeed the case. A plan π_i to reach a goal ϕ_{i+1} can be executed in a state ϕ_i. This state can be considered a precondition for the plan.

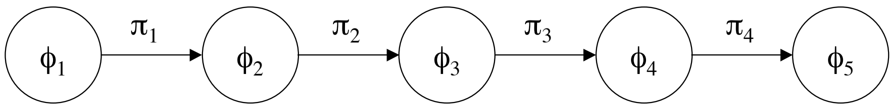

Figure 4.10: Subgoals

Rule g_2 for example specifies that the plan π_2 can be adopted if the agent believes to be in state ϕ_2 and has as a goal ϕ_3. This rule can only be used to go from ϕ_2 to ϕ_3. Now suppose that the agent is not in ϕ_2 but does have ϕ_3 as a goal. Rule g_2' can then be used to adopt ϕ_2 as a goal. This goal now is a sub-goal on the way from where the agent is to the goal ϕ_3.

These rules are very general, because they can be used for an agent who believes any state from ϕ_1 to ϕ_4 and has as a goal any state from ϕ_2 to ϕ_5 (as long as $i < j$ for $\mathbf{B}\phi_i$ and $\mathbf{G}\phi_j$). The agent cannot get from a state ϕ_i to ϕ_j where $i > j$ because it does not have plans achieving this.

4.7.3 The rules without adoption of goals

Suppose that the adoption of goals would not be possible. Suppose that an agent believes ϕ_5 and has ϕ_2 as a goal. Rule g_3 of section 4.7.2 cannot be applied because the agent is not (yet) in state ϕ_3. Rule g_1 cannot be applied because the agent only has the end-goal ϕ_2 and cannot derive the sub-goal ϕ_4. The rules would have to be changed to ones using only ϕ_2 as goal condition.

$g_{1.1}:\quad \mathbf{B}(\phi_1) \wedge \mathbf{G}(\phi_4) \rightarrow \pi_1$
$g_{2.1}:\quad \mathbf{B}(\phi_2) \wedge \mathbf{G}(\phi_4) \rightarrow \pi_2$
$g_{3.1}:\quad \mathbf{B}(\phi_3) \wedge \mathbf{G}(\phi_4) \rightarrow \pi_3$

These rules can however *only* be used for an agent with the goal ϕ_4 and they are thus less general. If the agent should also be able to handle goals ϕ_2, ϕ_3 and ϕ_5, rules would have to be added to deal with these goals. An agent with the same behaviour as the agent of the previous section could be created in this way. This agent would however have far more rules than the agent of the previous section.

Being able to adopt goals is thus useful, but I have not shown that it is necessary to model certain behaviour. More research is needed to proof or refute this.

4.7.4 An autonomous agent

The goal rules of the agent using the adoption of goals are more general than rules of the agent not using adoption of goals. I will now compare a Dribble agent using the rules with adoption of goals of section 4.7.2 to a 3APL agent that can execute plans π_i.

The rules of the Dribble agent can be used for an agent believing any state from ϕ_1 to ϕ_4 and having as a goal any state from ϕ_2 to ϕ_5 (as long as $i < j$ for $\mathbf{B}\phi_i$ and $\mathbf{G}\phi_j$). For each of these mental states with $\mathbf{B}\phi_i$ and $\mathbf{G}\phi_j$, a plan can be derived to reach the goal from the belief state. There are 10 different possible initial mental states of the agent. For each mental state, there is a different plan to reach the goal from the initial belief state (for instance $\pi_1; \pi_2; \pi_3; \pi_4$ if the agent believes ϕ_1 and has ϕ_5 as the initial goal, or $\pi_2; \pi_3$ if the agent initially believes ϕ_2 and has ϕ_4 as a goal).

In Dribble, the programmer needs to specify the initial goal (assuming the agent already has beliefs about its current situation). In 3APL, the programmer would have to specify the plan the agent should execute, to reach a certain implicit goal. The programmer would have to know the initial beliefs of the agent, because the plan differs if the initial beliefs differ. The programmer would also have to think of the goal the agent should reach, because the plan differs if the (implicit) goal differs. Thus, the programmer is doing a job the agent could do. The Dribble agent is more autonomous than the 3APL agent because it receives less guidance from its programmer.

4.8 An alternative tower building agent

In the previous sections, I compared the languages GOAL, Dribble and 3APL, using example implementations of a tower building agent in the different languages. The GOAL and Dribble tower building agents that were described, may not follow the shortest path to the goal (or may not even reach the goal at all, in case of the GOAL agent). In this section, I will give an implementation of a Dribble agent that *will* follow the shortest path to the goal. This agent will use the special actions to adopt or drop goals, although these actions are not yet part of the language Dribble.

I will first explain why the agents of the previous sections may not follow the shortest path to the goal. Suppose that the agent believes that block A and B are on the floor and C is on A. Suppose the goal is that C is on the floor, B is on C and A is on B as in section 4.1 (see figure 4.11).

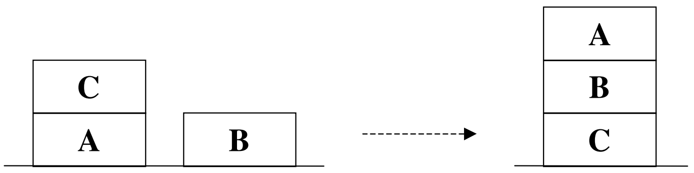

Figure 4.11: Initial beliefs and goal

The shortest path to the goal is moving C onto the floor, moving B onto C and then A onto B. The agent could however move B onto C first, satisfying the part of the goal that B should be on C. It does not know that this move will have to be undone later on. The example agents only have rules to satisfy parts of the goal. Satisfying some part of the goal may undo an earlier satisfied part. The agent has no way of knowing which part to satisfy first in order to follow the shortest path to the goal. It does not know that it should start at the bottom of the tower and work its way to the top. All parts of the goal have an equal status and it is not specified in which order they should be satisfied. For a 3APL tower building agent, the order *is* specified, but this also has disadvantages as was explained (see section 4.6.1).

4.8.1 The rules

It seems there should be a way of programming an agent that knows it should build a tower bottom-up. A possibility could be to change the way in which the goal of the agent is described. Instead of the predicate $on(x, y)$, a predicate $tower(x, T)$ could be used. This predicate means that the agent has the goal to build a tower of which x is the bottom block. T is a variable denoting the tower that is supposed to be built on top of block x. The goal of the agent to have for example a tower A on B on C on the floor, could be described by the following predicate: $tower(Fl, tower(C, tower(B, tower(A, nil))))$.

The following goal rules could be used to program the Dribble agent using this tower predicate.

$$\begin{array}{lll}
\mathbf{G}(\mathit{tower}(z,T)) \wedge \mathbf{B}(\mathit{clear}(z)) \wedge \mathbf{B}(T = \mathit{tower}(x,\mathit{nil}) \wedge \mathit{on}(x,y)) & \rightarrow & \mathit{move}(x,y,z); \\
 & & \mathit{drop}(\mathbf{G}(\mathit{tower}(z,T))) \\
\mathbf{G}(\mathit{tower}(z,T)) \wedge \mathbf{B}(\mathit{clear}(z)) \wedge \mathbf{B}(T = \mathit{tower}(x,T') \wedge \mathit{on}(x,y)) & \rightarrow & \mathit{move}(x,y,z); \\
 & & \mathit{adopt}(\mathbf{G}(\mathit{tower}(x,T'))); \\
 & & \mathit{drop}(\mathbf{G}(\mathit{tower}(z,T)))
\end{array}$$

PR rules should be added to deal with blocks that need to be cleared, but they are omitted here. The second rule can be applied if the agent has a goal that a tower is built of which z is the bottom block and of which T is the tower to be built on top of z. This bottom block has to be clear if a tower is to be built on top of it. The bottom block of the tower to be built on top of z, is x. The block z is clear and x should be on top of z. Therefore the action $\mathit{move}(x, y, z)$ is derived. The part of the goal that is left to be satisfied is $\mathit{tower}(x,T')$, which is adopted. The goal of which a part has already been satisfied ($\mathit{tower}(z,T)$), is dropped.

The first rule is used to stop the adoption of goals if the tower is finished. If the tower is finished except for the top block, the goal of the agent satisfies $T = \mathit{tower}(x, \mathit{nil})$. In that case, x should be moved onto z and no new goal should be adopted. The goal ($\mathit{tower}(z,T)$) can be dropped, because it is realised.

4.8.2 Example execution

Suppose the beliefs and goal of the agent are as described above (figure 4.11). The second rule can be applied. The goal of the agent is $\mathit{tower}(Fl,T)$. The floor is always clear, so the second belief condition of this rule is satisfied. The bottom block of the tower T to be built on the floor, is C ($T = \mathit{tower}(C,T')$). Therefore the action $\mathit{move}(C, A, Fl)$ is derived. Furthermore, the goal $\mathit{tower}(C, \mathit{tower}(B, \mathit{tower}(A, \mathit{nil})))$ is adopted and the goal $\mathit{tower}(Fl, \mathit{tower}(C, \mathit{tower}(B, \mathit{tower}(A, \mathit{nil}))))$ is dropped. The second rule can be applied again, but now with the goal $\mathit{tower}(C,T)$ where $T = \mathit{tower}(B,T')$. The action $\mathit{move}(B, Fl, C)$ is derived, the goal $\mathit{tower}(B, \mathit{tower}(A, \mathit{nil}))$ is adopted and the goal $\mathit{tower}(C, \mathit{tower}(B, \mathit{tower}(A, \mathit{nil})))$ is dropped. Now the first rule can be applied, because the goal is $\mathit{tower}(B,T)$ where $T = \mathit{tower}(A, \mathit{nil})$. The action $\mathit{move}(A, Fl, B)$ is derived, the goal $\mathit{tower}(B, \mathit{tower}(A, \mathit{nil}))$ is dropped after which the goals of the agent are empty and the tower is finished.

4.8.3 Discussion

The benefit of programming the tower building agent in this way, is that it will follow the shortest path to the goal. It will always start building the tower at the bottom. If it is for some reason crucial that the agent operates fast, this can be a good approach.

This approach has a procedural flavour to it. In the tower predicate the order in which blocks have to be stacked, is specified. This is exactly what we wanted, but it has some disadvantages. It is perhaps not surprising that these disadvantages correspond to the disadvantages of the 3APL tower building agent (see section 4.5).

Because of the specification of the goal with the tower predicate, the connection between beliefs and goals is lost. The tower predicate denoting the goal, will not at some point be believed by the agent after which it is removed from the goal base. The manipulation of goals is done manually through the *adopt* and *drop* actions.

The problem now is, that the programmer in fact specifies what the agent should *do* instead of what situation the agent should achieve. The programmer will need to have knowledge about the initial beliefs of the agent. If the agent already believes the tower to have been built except for the top block, the goal should be $tower(B, tower(A, nil))$. The goal specifies what the agent should do and if the situation is such that it should do less, the goal needs to specify this. In the case of the example Dribble agent, the connection between beliefs and goals solves this problem. The goal of the agent can be $on(A, B) \wedge on(B, C) \wedge on(C, Fl)$. If the last two conjuncts now are true in a mental state, these are not goals in that mental state by definition. The agent will therefore only apply rules to satisfy the goal $on(A, B)$.

In a world in which other agents are present that can also move blocks, the lack of connection between beliefs and goals again causes problems. The goal rules of section 4.8.1 assume, that once a block is in the right position at the bottom of the tower, this block will remain in that position. Assume for example that the agent has already moved the bottom two blocks into their right position (C on the floor and B on C). The goal of the agent now is $tower(B, tower(A, nil))$, because only block A needs to be moved. Suppose now that a hostile agent moves block B onto the floor again. The first rule of section 4.8.1 can still be applied, because B is still clear. A is moved onto B and the agent assumes that the goal is reached. The removal of goals is done manually and there is no guarantee that the goal is actually reached after the removal. The example Dribble agent on the other hand, would believe that B is not on C and would therefore not drop its initial goal $on(A, B) \wedge on(B, C) \wedge on(C, Fl)$. This is due to the connection between beliefs and goals.

Using the tower predicate to describe the goal causes a lack of connection between beliefs and goals. The Dribble agent described in this section does not use the commitment strategy connecting beliefs and goals. Using the tower predicate as an achievement goal in 3APL could therefore result in an agent with the same behaviour as the agent of this section.

$$\begin{array}{lcl} tower(z, T) \mid \mathbf{B}(clear(z)) \wedge \mathbf{B}(T = tower(x, nil) \wedge on(x, y)) & \rightarrow & move(x, y, z) \\ tower(z, T) \mid \mathbf{B}(clear(z)) \wedge \mathbf{B}(T = tower(x, T') \wedge on(x, y)) & \rightarrow & move(x, y, z); tower(x, T') \end{array}$$

The achievement goal $tower(z, T)$ does indeed have a close resemblance with the goal $tower(z, T)$. The goal is dropped manually after the move action and the new goal goal is adopted manually. The achievement goal is automatically dropped once it is replaced through application of a PR rule and the new achievement goal is adopted by adding it to the plan.

Chapter 5

Logic

On top of the language Dribble and its semantics, I now construct a dynamic logic to prove properties of Dribble agents.

5.1 The logical language

In this section, the logical language is defined that is used to specify and verify properties of Dribble agents. The meaning of the language is defined in section 5.2. The language consists of *mental state formulas* and of formulas denoting the results of actions.

5.1.1 Mental state formulas

The mental state formulas are used to describe properties of the mental state of a Dribble agent. They are an extension of the belief and goal formulas of section 2.1. With mental state formulas it is not only possible to formulate statements about the beliefs and goals of the agent, but also about the plan.

Definition (*mental state formulas*)
The set of mental state formulas $\mathcal{L}_M$ with typical formula μ is defined by:

- $\mathcal{L}_{BG} \subseteq \mathcal{L}_M$,
- if $\pi \in \mathsf{Plan}$ then $\mathbf{Com}(\pi) \in \mathcal{L}_M$,
- if $\mu_1, \mu_2 \in \mathcal{L}_M$, then $\neg\mu_1, \mu_1 \wedge \mu_2 \in \mathcal{L}_M$.

The usual abbreviatons for the propositional operators $\vee$, $\rightarrow$ and $\leftrightarrow$ are used for $\mathcal{L}_M$.

5.1.2 The language $\mathcal{L}_D$

The language $\mathcal{L}_D$ is a language of *dynamic logic* [6, 3, 2, 7, 9]. Dynamic logic can be used to reason about computer programs. These programs are explicit syntactic constructs in the logic. To be able to discuss the effect of the execution of a program p on the truth of a formula ϕ, the modal construct $\langle p\rangle\phi$ is used. This construct intuitively states that it is possible to execute p and to halt in a state satisfying ϕ. The program p can be a compound program, built inductively from primitive programs using a small set of program operators. The state resulting from execution of p is determined by the states resulting from execution of parts of p (compositionality). In conventional procedural programming languages, the program p is a transformation function on states: p can be executed in a state, resulting in a new state.

A Dribble agent is a quadruple $\langle\sigma_0, \gamma_0, \Gamma, \Delta\rangle$. A mental state consists of beliefs, goals and a plan. A transformation of one mental state into another can be established by rule application or execution of an action from the plan. This is in contrast with conventional procedural programs, where some fixed program is the transformation function on states. In a dynamic logic for procedural programs, it is this procedural program that can be reasoned about. The question is, what the program is that should be reasoned about in the dynamic logic for Dribble agents. A possibility is to reason about the plan component of the mental state of the agent.

The plan component of the mental state of a Dribble agent has a resemblance with a procedural program. There is however a difference. In contrast with the meaning of a procedural program, the meaning of the execution of a plan of an agent cannot be defined in terms of the meaning of its parts. If the plan is for instance a sequence of basic actions, this does not necessarily mean that these actions will all be executed in sequence. It is possible that a PR rule is applied to change the sequence of actions. The plan of an agent can be modified during execution of the agent because it is part of the mental state of the agent. A conventional procedural program is on the contrary usually fixed. It is not part of a state, but it is a transformation function *on* states. The meaning of the formula $\langle p\rangle\phi$ is usually defined in terms of the mental state resulting from execution of p, which is defined in terms of execution of the parts of p. If the program p in the formula $\langle p\rangle\phi$ would be the plan of the agent, the definition of the meaning of this formula would be very difficult: the meaning of a plan cannot be defined in terms of the meaning of its parts.

Thus the mental state transitions of a Dribble agent are not solely defined by the (parts of the) plan of the agent. The transformation of one mental state into another however depends on whether an applicable rule is applied (and which, if more than one is applicable) or whether an action from the plan is executed. It is these mental state transitions that I want to reason about in the logic, because these transitions define the execution of the agent. I refer to these transitions using so called *meta-actions*. In the dynamic logic for Dribble, a sequence of meta-actions is used as the program p in the formula $\langle p\rangle\phi$.

In summary, the execution of a program is a series of state transitions. In dynamic logic we can reason about the execution of a program. For a conventional procedural program, the program defines the transformations on states. It is thus this program that is reasoned about in the logic. For Dribble agents, mental state transitions are either caused by application of a rule or execution of an action. In the logic for Dribble it is thus these actions that are reasoned about. Below, the set of meta-actions is defined.

Definition (*meta-actions*)
The set $\mathsf{MetaAction}$ with typical element m is defined by:

- if $\pi \in \mathsf{Plan}$ then $\mathtt{commit}(\pi) \in \mathsf{MetaAction}$ and $\mathtt{uncommit}(\pi) \in \mathsf{MetaAction}$,
- if $g \in \Gamma$ then $\mathtt{applyRule}(g) \in \mathsf{MetaAction}$,
- if $\rho \in \Delta$ then $\mathtt{applyRule}(\rho) \in \mathsf{MetaAction}$,
- if $b \in \mathsf{ExecutableAction}$ then $\mathtt{execute}(b) \in \mathsf{MetaAction}$.

The meta-actions $\mathtt{applyRule}(g)$ and $\mathtt{applyRule}(\rho)$ are used to specify that a goal rule g and a PR rule ρ are applied respectively. The meta-actions $\mathtt{commit}(\pi)$ and $\mathtt{uncommit}(\pi)$ are used in defining the semantics of the other meta-actions (see section 5.2.3). The result of a commitment to the plan π is that the agent has π as its plan. The result of an uncommitment to π is that the plan π of the agent is dropped. The meta-action $\mathtt{execute}(b)$ is used to describe that a basic action or an if-then-else construct is executed.

As I explained above, with these meta-actions the language of dynamic logic for Dribble can be defined. The meta-actions are used as the program p in the formula $\langle p \rangle \phi$. This formula can be used to express the result of executing a sequence of meta-actions. For reasons that will be explained in section 5.2.3, the sequence of actions is in fact a sequence of meta-actions and basic actions, instead of just a sequence of meta-actions.

Below, the set of sequences of basic actions and meta-actions is defined.

Definition (*basic and meta-actions*)
The set $\mathsf{BasicAndMetaAction}$ with typical element α is defined by:

- $\mathsf{BasicAction} \subseteq \mathsf{BasicAndMetaAction}$,
- $\mathsf{MetaAction} \subseteq \mathsf{BasicAndMetaAction}$,
- if $\alpha \in \mathsf{BasicAndMetaAction}$ and $am \in \mathsf{BasicAction} \cup \mathsf{MetaAction}$
  then $am; \alpha \in \mathsf{BasicAndMetaAction}$.

The language $\mathcal{L}_D$ consists of mental state formulas and of formulas that can be used to reason about the result of execution of a sequence of meta-actions and basic actions. The result of this execution is expressed in terms of mental state formulas. Below, the language $\mathcal{L}_D$ is defined.

Definition (*language of dynamic logic for Dribble: $\mathcal{L}_D$*)
The dynamic logic language $\mathcal{L}_D$ with typical formula δ is defined by:

- $\mathcal{L}_M \subseteq \mathcal{L}_D$,
- if $\alpha \in \mathsf{BasicAndMetaAction}$ and $\mu \in \mathcal{L}_M$ then $\langle \alpha \rangle \mu \in \mathcal{L}_D$,
- if $\delta_1, \delta_2 \in \mathcal{L}_D$, then $\neg\delta_1, \delta_1 \wedge \delta_2 \in \mathcal{L}_D$.

The formula $[\alpha]\mu$ is shorthand for $\neg\langle\alpha\rangle\neg\mu$. The usual abbreviatons for the propositional operators $\vee$, $\rightarrow$ and $\leftrightarrow$ are used for $\mathcal{L}_D$.

5.2 The semantics

5.2.1 Semantics of mental state formulas

Mental state formulas are evaluated in a mental state. As was stated, mental state formulas are an extension of the belief and goal formulas. The meaning of the belief and goal formulas was already defined in section 3.1. A formula $\mathbf{Com}(\pi')$ is true in a mental state, if and only if the agent is committed to the plan π' in that state. It is committed to this plan if the plan component of the mental state is equal to π'. Note that the agent only has one commitment, which is the commitment to its current plan. The agent is not committed to prefixes of its plan. The reason is, that we do not need commitment to prefixes when defining the meaning of the meta-actions (see section 5.2.3).

Definition (*semantics of mental state formulas*)
Let $\langle\sigma,\gamma,\pi\rangle$ be a mental state. Let ϕ, $\psi \in \mathcal{L}$ be propositional formulas, let π, $\pi' \in \mathsf{Plan}$ be plans and μ, μ_1, $\mu_2 \in \mathcal{L}_M$ be mental state formulas.

- $\langle\sigma,\gamma,\pi\rangle \models \mathbf{B}\phi$ iff $\sigma \models \phi$
- $\langle\sigma,\gamma,\pi\rangle \models \mathbf{G}\psi$ iff $\psi \rightsquigarrow_\sigma \gamma$
- $\langle\sigma,\gamma,\pi\rangle \models \mathbf{Com}(\pi')$ iff $\pi = \pi'$
- $\langle\sigma,\gamma,\pi\rangle \models \neg\mu$ iff $\langle\sigma,\gamma,\pi\rangle \not\models \mu$
- $\langle\sigma,\gamma,\pi\rangle \models \mu_1 \wedge \mu_2$ iff $\langle\sigma,\gamma,\pi\rangle \models \mu_1$ and $\langle\sigma,\gamma,\pi\rangle \models \mu_2$

5.2.2 Semantics of $\mathcal{L}_D$

Formulas from the language $\mathcal{L}_D$ are evaluated in a mental state. The semantics of mental state formulas ($\mathcal{L}_M$) in $\mathcal{L}_D$ was defined in the previous section (5.2.1). The formula $\langle\alpha\rangle\mu$ is true in a mental state s, if it is possible to reach a mental state in which μ holds, through execution of the sequence α of meta-actions and basic actions in the state s. The mental state reached through execution of α should not be $\mathcal{E}$. $\mathcal{E}$ is a state to which all impossible basic actions and meta-actions lead: from there, no other actions can be performed anymore. $\mathcal{E}$ is not an element of the set Σ of possible mental states.

Execution of a basic action or meta-action changes the mental state of the agent. This change is specified with the *result function* $\mathtt{r}^*(\alpha)\langle\sigma,\gamma,\pi\rangle$. This function yields the mental states resulting from executing α in $\langle\sigma,\gamma,\pi\rangle$. The function is defined in section 5.2.3.

Definition (*semantics of formulas in $\mathcal{L}_D$*)
The semantics of formulas in $\mathcal{L}_D$ is defined as follows with $\alpha \in \mathsf{BasicAndMetaAction}$, $\mu \in \mathcal{L}_M$ and $\delta, \delta_1, \delta_2 \in \mathcal{L}_D$:

- $\langle\sigma,\gamma,\pi\rangle \models \mu$ see section 5.2.1,
- $\langle\sigma,\gamma,\pi\rangle \models \langle\alpha\rangle\mu$ iff exists a mental state $\langle\sigma',\gamma',\pi'\rangle \neq \mathcal{E} \in \mathtt{r}^*(\alpha)\langle\sigma,\gamma,\pi\rangle$ with $\langle\sigma',\gamma',\pi'\rangle \models \mu$,

- $\langle\sigma,\gamma,\pi\rangle \models \neg\delta$ iff $\langle\sigma,\gamma,\pi\rangle \not\models \delta$,
- $\langle\sigma,\gamma,\pi\rangle \models \delta_1 \wedge \delta_2$ iff $\langle\sigma,\gamma,\pi\rangle \models \delta_1$ and $\langle\sigma,\gamma,\pi\rangle \models \delta_2$.

5.2.3 Semantics of meta-actions and basic actions

The function $\mathtt{r}^*$ is a transformation function on mental states through sequences of meta-actions and basic actions. It is defined in terms of the function $\mathtt{r}$, which is a transformation function on mental states through *atomic actions*. Atomic actions are actions that are not defined in terms of a sequence of other actions. Basic actions, the `commit` and `uncommit` actions and the `applyRule`(g) action are atomic actions as will become clear later on, whereas for instance `applyRule`(ρ) is not atomic. The set of atomic actions is defined below.

Definition (*atomic actions*)
The set $\mathsf{AtomicAction}$ with typical element aa is defined by:

- $\mathsf{BasicAction} \subseteq \mathsf{AtomicAction}$,
- if $\pi \in \mathsf{Plan}$ then $\mathtt{commit}(\pi) \in \mathsf{AtomicAction}$ and $\mathtt{uncommit}(\pi) \in \mathsf{AtomicAction}$,
- if $g \in \Gamma$ then $\mathtt{applyRule}(g) \in \mathsf{AtomicAction}$.

Now, the semantics of $\mathtt{r}^*$ can be defined. $\mathtt{r}^*$ is defined in terms of $\mathtt{r}$, which is a function of type $: \mathsf{AtomicAction} \times (\Sigma \cup \{\mathcal{E}\}) \to \wp(\Sigma \cup \{\mathcal{E}\})$. The function $\mathtt{r}^*$ is of type $: \mathsf{BasicAndMetaAction} \times (\Sigma \cup \{\mathcal{E}\}) \to \wp(\Sigma \cup \{\mathcal{E}\})$. The result of both functions is thus a set of mental states. The meaning of actions is however in this paper defined such, that they are deterministic: the resulting set contains only one element. For convenience, the result functions will therefore in the sequel be defined as if the result type was $\Sigma \cup \{\mathcal{E}\}$, instead of $\wp(\Sigma \cup \{\mathcal{E}\})$.

Definition (*semantics of* $\mathtt{r}^*$)
The semantics of $\mathtt{r}^*$ is defined as follows, with $aa \in \mathsf{AtomicAction}$, $am \in \mathsf{BasicAction} \cup \mathsf{MetaAction}$ and $\alpha \in \mathsf{BasicAndMetaAction}$:

$$\begin{array}{lcl} \mathtt{r}^*(aa)\langle\sigma,\gamma,\pi\rangle & = & \mathtt{r}(aa)\langle\sigma,\gamma,\pi\rangle, \\ \mathtt{r}^*(am;\alpha)\langle\sigma,\gamma,\pi\rangle & = & \mathtt{r}^*(\alpha)(\mathtt{r}^*(am)\langle\sigma,\gamma,\pi\rangle), \\ \mathtt{r}^*(\alpha)\ \mathcal{E} & = & \mathcal{E}. \end{array}$$

In the definition above I have defined what it means to execute a sequence of meta-actions and basic actions. Now I will define the semantics of executing specific single actions.

I will first define what it means to execute the meta-action `commit`(π). This action is used in defining the meaning of the meta-actions for rule application and action execution. Executing a `commit`(π) action leads to the adoption of the plan π. It can only be executed in a mental state if the plan of the agent is empty in that mental state. The reason for this is, that in the definitions of meta-actions using the `commit` action, this action is executed in a state with an empty plan. It is thus only necessary to have a commitment action for the case of an agent with an empty plan.

Definition (*semantics of commit*)

$\texttt{r(commit(}\pi'\texttt{))}\langle\sigma,\gamma,E\rangle \;=\; \langle\sigma,\gamma,\pi'\rangle$
$\texttt{r(commit(}\pi'\texttt{))}\langle\sigma,\gamma,\pi\rangle \;=\; \mathcal{E}\ \textit{with}\ \pi \neq E$

The meta-action `uncommit`(π) can only be executed if the agent is committed to π. If this is the case, the plan of the agent becomes empty in the mental state resulting from executing the `uncommit` action. The agent only has one commitment in a mental state, which is equal to the plan in that mental state. It does not have commitments to prefixes of the plan and it can thus not uncommit to a prefix. This is the reason why the result of an uncommitment is always an empty plan (assuming the uncommitment can be executed succesfully).

Definition (*semantics of uncommit*)

$\texttt{r(uncommit(}\pi'\texttt{))}\langle\sigma,\gamma,\pi\rangle \;=\; \langle\sigma,\gamma,E\rangle\ \textit{if}\ \pi = \pi'$
$\texttt{r(uncommit(}\pi'\texttt{))}\langle\sigma,\gamma,\pi\rangle \;=\; \mathcal{E}\ \textit{otherwise}$

The actions `commit` and `uncommit` are defined because it makes the definition of the other meta-actions easier. A Dribble agent should not be thought of as having the capability to commit or uncommit to plans. A Dribble agent can only manipulate its plan through rule application or action execution, as was defined in chapter 3.

The meta-action `applyRule`(g) where g is $\varphi \rightarrow \pi'$ can be executed in a mental state if the plan of the agent is empty and if φ is true in that mental state. In the mental state resulting from the application of g, π' is the plan of the agent.

Definition (*semantics of goal rule application*)
Let $g : \varphi \rightarrow \pi'$ be a goal rule.

$\texttt{r(applyRule(}g\texttt{))}\langle\sigma,\gamma,\pi\rangle \;=\; \texttt{r(commit(}\pi'\texttt{))}\langle\sigma,\gamma,\pi\rangle\ \textit{if}\ \pi = E\ \textit{and}\ \langle\sigma,\gamma,\pi\rangle \models \varphi$
$\texttt{r(applyRule(}g\texttt{))}\langle\sigma,\gamma,\pi\rangle \;=\; \mathcal{E}\ \textit{otherwise}$

The meta-action `applyRule`(ρ) where ρ is $\pi_h \mid \beta \rightarrow \pi_b$ can be executed in a mental state if the head of the PR rule ρ is equal to a prefix of the plan of the agent and if the guard of the rule is true in that mental state. In the mental state resulting from application of ρ, the prefix of the plan of the agent is equal to the body of the PR rule. The semantics of the `applyRule`(ρ) action is defined in terms of the `commit` and `uncommit` actions. First the plan of the agent is dropped, after which the plan is empty. Then the plan that has as a prefix the body of the PR rule instead of the head, is adopted through the `commit` action.

Definition (*semantics of PR rule application*)
Let $\rho : \pi_h \mid \beta \rightarrow \pi_b$ be a PR rule.

$\texttt{r}^*\texttt{(applyRule(}\rho\texttt{))}\langle\sigma,\gamma,\pi\rangle \;=\; \texttt{r}^*\texttt{(uncommit(}\pi_h \circ \pi'\texttt{);commit(}\pi_b \circ \pi'\texttt{))}\langle\sigma,\gamma,\pi\rangle$
$\quad \textit{if}\ \pi = \pi_h \circ \pi'\ \textit{and}\ \langle\sigma,\gamma,\pi\rangle \models \beta$
$\texttt{r}^*\texttt{(applyRule(}\rho\texttt{))}\langle\sigma,\gamma,\pi\rangle \;=\; \mathcal{E}\ \textit{otherwise}$

The meta-action `execute`(a) where a is a basic action, can be executed if a is a prefix of the plan of the agent. The meta-action is defined in terms of the basic action and the `commit` and `uncommit` meta-actions. First a is executed which possibly changes the belief base and goal base of the agent (see definition of *semantics of actions*). Then the plan of the agent is updated. There is thus a difference between executing the *meta-action* `execute`(a) and executing the *basic action* a. The first also results in an update of the plan of the agent, whereas the second one only updates the beliefs and goals.

Definition (*semantics of action execution(1)*)

$$\mathtt{r}^*(\mathtt{execute}(a))\langle\sigma,\gamma,\pi\rangle \;=\; \mathtt{r}^*(a;\mathtt{uncommit}(a;\pi');\mathtt{commit}(\pi'))\langle\sigma,\gamma,\pi\rangle$$
$$\quad \textit{if } \pi = a;\pi'$$
$$\mathtt{r}^*(\mathtt{execute}(a))\langle\sigma,\gamma,\pi\rangle \;=\; \mathcal{E} \textit{ otherwise}$$

The meaning of the meta-action $\mathtt{execute}(a)$ could have been defined without defining the meaning of the execution of a separately. I however thought that separating the plan updating part from the belief updating part, results in a clearer specification of the semantics of $\mathtt{execute}(a)$. Because I define the meaning of $\mathtt{execute}(a)$ in terms of the meaning of a, I use sequences of meta-actions *and* basic actions to reason about in the logic. In this way, one can reason about a as well as about $\mathtt{execute}(a)$.

The meta-action $\mathtt{execute}(\mathtt{if}\ \beta\ \mathtt{then}\ \pi_1\ \mathtt{else}\ \pi_2\ \mathtt{fi})$ can be executed in a mental state if the if-then-else construct is a prefix of the plan of the agent in that mental state. The plan of the agent in the mental state resulting from executing this meta-action, depends on the truth of the if-condition. If the condition is true, the if-then-else construct is replaced by the plan in the then-part of the construct. If the condition is false (or if it is not the case that the condition is true, which is equivalent), the construct is replaced by the plan in the else-part.

Definition (*semantics of action execution(2)*)

$$\mathtt{r}^*(\mathtt{execute}(\mathtt{if}\ \beta\ \mathtt{then}\ \pi_1\ \mathtt{else}\ \pi_2\ \mathtt{fi}))\langle\sigma,\gamma,\pi\rangle \;=$$
$$\quad \mathtt{r}^*\Big(\mathtt{uncommit}(\mathtt{if}\ \beta\ \mathtt{then}\ \pi_1\ \mathtt{else}\ \pi_2\ \mathtt{fi};\pi');\mathtt{commit}(\pi_1\circ\pi')\Big)\langle\sigma,\gamma,\pi\rangle$$
$$\quad \textit{if } \pi = \mathtt{if}\ \beta\ \mathtt{then}\ \pi_1\ \mathtt{else}\ \pi_2\ \mathtt{fi};\pi' \textit{ and } \langle\sigma,\gamma,\pi\rangle \models \beta$$
$$\mathtt{r}^*(\mathtt{execute}(\mathtt{if}\ \beta\ \mathtt{then}\ \pi_1\ \mathtt{else}\ \pi_2\ \mathtt{fi}))\langle\sigma,\gamma,\pi\rangle \;=$$
$$\quad \mathtt{r}^*\Big(\mathtt{uncommit}(\mathtt{if}\ \beta\ \mathtt{then}\ \pi_1\ \mathtt{else}\ \pi_2\ \mathtt{fi};\pi');\mathtt{commit}(\pi_2\circ\pi')\Big)\langle\sigma,\gamma,\pi\rangle$$
$$\quad \textit{if } \pi = \mathtt{if}\ \beta\ \mathtt{then}\ \pi_1\ \mathtt{else}\ \pi_2\ \mathtt{fi};\pi' \textit{ and } \langle\sigma,\gamma,\pi\rangle \models \neg\beta$$
$$\mathtt{r}^*(\mathtt{execute}(\mathtt{if}\ \beta\ \mathtt{then}\ \pi_1\ \mathtt{else}\ \pi_2\ \mathtt{fi}))\langle\sigma,\gamma,\pi\rangle \;=\; \mathcal{E} \textit{ otherwise}$$

The if-then-else construct can only be executed through the meta-action `execute`. The meaning of the formula $\langle\mathtt{if}\ \beta\ \mathtt{then}\ \pi_1\ \mathtt{else}\ \pi_2\ \mathtt{fi}\rangle\mu$ is not defined. The reason is, that one does not reason about plans in the logic, but about rule application and action execution. This was explained in section 5.1.2. Basic actions are the exception to this rule, as was explained above.

In section 3.2.4 it was stated that it is convenient to define the if-then-else construct as a basic element of a plan, instead of in terms of a non-deterministic choice operator. If the construct would have been defined in terms of this operator, the meaning of the execution of the operator would have to be defined in the logic. This would lead to a complicated definition, whereas the meaning of the construct can be defined in a simple manner using the `commit` and `uncommit` actions. Furthermore, the execution of a non-deterministic choice operator would not be deterministic by definition. The definition of $\mathtt{r}^*$ would therefore have to be changed.

If *basic actions* are executed, the beliefs and goals of the agent are updated. As in section 3.2.3, the belief update is formally represented by a partial transformation function $\mathcal{T}$ of type $: \mathsf{BasicAction} \times \wp(\mathcal{L}) \to \wp(\mathcal{L})$. If this function is not defined, the basic action cannot be executed.

Definition (*semantics of actions*)

$\mathtt{r}(a)\langle\sigma,\gamma,\pi\rangle = \langle\mathcal{T}(a,\sigma),\gamma',\pi\rangle$ *with* $\gamma' = \gamma \setminus \{\psi \in \gamma \mid \mathcal{T}(a,\sigma) \models \psi\}$ *if* $\mathcal{T}(a,\sigma)$ is defined
$\mathtt{r}(a)\langle\sigma,\gamma,\pi\rangle = \mathcal{E}$ *otherwise*

The meaning of basic action execution is defined through the function $\mathcal{T}$. One typically thinks of actions in terms of preconditions and postconditions. The action can only be executed in a mental state if the preconditions are satisfied in that mental state. The state resulting from executing the basic action should satisfy the postconditions. In section 4.1, I described the preconditions and postconditions of the action $move(x,y,z)$. I will repeat this definition here. The function $\mathcal{T}(move(x,y,z),\sigma)$ is defined, only if σ satisfies the preconditions for the action $move(x,y,z)$ (`Preconditions`). Some predicates are changed by the action (`Postconditions`). Other predicates are not changed by the action (`Frame conditions`).

Definition (*semantics of basic action move(x,y,z)*)

`Preconditions`

$$\mathcal{T}(move(x,y,z),\sigma) = \sigma' \quad \Leftrightarrow \quad \sigma \models clear(x) \wedge clear(z) \wedge on(x,y)$$

`Postconditions`

$$\begin{array}{lll}
\mathcal{T}(move(x,y,z),\sigma) = \sigma' \;\wedge\; z \neq Fl & \Rightarrow & \sigma' \models on(x,z) \wedge clear(y) \wedge \neg on(x,y) \wedge \neg clear(z) \\
\mathcal{T}(move(x,y,z),\sigma) = \sigma' \;\wedge\; z = Fl & \Rightarrow & \sigma' \models on(x,z) \wedge clear(y) \wedge \neg on(x,y)
\end{array}$$

`Frame conditions`

$$\begin{array}{llllllll}
\mathcal{T}(move(x,y,z),\sigma) = \sigma' & \wedge & \sigma \models on(u,v) & \wedge & u \neq x & \Rightarrow & \sigma' \models on(u,v) \\
\mathcal{T}(move(x,y,z),\sigma) = \sigma' & \wedge & \sigma \models \neg on(u,v) & \wedge & ((u \neq x) \vee (v \neq z)) & \Rightarrow & \sigma' \models \neg on(u,v) \\
 & & & & & & \\
\mathcal{T}(move(x,y,z),\sigma) = \sigma' & \wedge & \sigma \models clear(w) & \wedge & w \neq z & \Rightarrow & \sigma' \models clear(w) \\
\mathcal{T}(move(x,y,z),\sigma) = \sigma' & \wedge & \sigma \models \neg clear(w) & \wedge & w \neq y & \Rightarrow & \sigma' \models \neg clear(w)
\end{array}$$

5.3 Characteristics of actions in the logic

Meta-actions and basic actions are mental state transformers. The mental states resulting from execution of these actions were defined in the previous section using the function $\mathtt{r}^{(*)}$. If some action cannot be executed, the mental state resulting from executing the action anyway, is $\mathcal{E}$. In this section, I will define characteristics of these mental state transformer actions in the logic. I will split the characteristics into two parts: the preconditions or *realizability* and the postconditions or *results* of actions. The realizability specifies when an action can be executed ($\langle am\rangle\top$ where $am \in$ BasicAction $\cup$ MetaAction). The results specify the properties of the mental state resulting from executing an action ($[am]\mu$ where $am \in$ BasicAction$\cup$MetaAction).

In the following, ρ is a PR rule $\pi_h \mid \beta \to \pi_b$ and g is a goal rule $\varphi \to \pi$.

Proposition (*realizability of meta actions*)

1	$\models \mathbf{Com}(E)$	$\leftrightarrow$	$\langle \texttt{commit}(\pi)\rangle\top$
2	$\models \mathbf{Com}(\pi)$	$\leftrightarrow$	$\langle \texttt{uncommit}(\pi)\rangle\top$
3	$\models \mathbf{Com}(E) \wedge \varphi$	$\leftrightarrow$	$\langle \texttt{applyRule}(g)\rangle\top$
4	$\models \mathbf{Com}(\pi_h \circ \pi) \wedge \beta$	$\rightarrow$	$\langle \texttt{applyRule}(\rho)\rangle\top$
5	$\models \mathbf{Com}(a;\pi) \wedge \langle a\rangle\top$	$\rightarrow$	$\langle \texttt{execute}(a)\rangle\top$
6	$\models \mathbf{Com}(\texttt{if } \beta \texttt{ then } \pi_1 \texttt{ else } \pi_2 \texttt{ fi};\pi)$	$\rightarrow$	$\langle \texttt{execute}(\texttt{if } \beta \texttt{ then } \pi_1 \texttt{ else } \pi_2 \texttt{ fi})\rangle\top$

All actions require a commitment to some plan to be executed succesfully. The first three properties specify an equivalence whereas the last three specify an implication. The last three properties do not specify an equivalence because no conclusion can be drawn about the plan of the agent, assuming that one of these meta-actions is executable. Assuming that the meta-action `execute`(a) is executable, the agent could have a commitment to $a;\pi_1$, to $a;\pi_2$ and so on and so forth. The third and fourth properties do not only have a requirement on the plan, but also on the beliefs and/or goals and the beliefs respectively. The fifth property has the requirement that basic action a should be realizable. Only then can the meta-action `execute`(a) be executed succesfully. An if-then-else construct can always be executed as long as a prefix of the plan is equal to this construct.

Proposition (*results of meta actions*)

1 $\models [\texttt{commit}(\pi)]\mathbf{Com}(\pi)$
2 $\models [\texttt{uncommit}(\pi)]\mathbf{Com}(E)$
3 $\models [\texttt{applyRule}(g)]\mathbf{Com}(\pi)$

4 $\models \mathbf{Com}(\pi_h \circ \pi) \rightarrow$
$\quad [\texttt{applyRule}(\rho)]\mathbf{Com}(\pi_b \circ \pi)$
5 $\models \mathbf{Com}(a;\pi) \wedge [a]\beta \rightarrow$
$\quad [\texttt{execute}(a)]\Big(\mathbf{Com}(\pi) \wedge \beta\Big)$
6 $\models \mathbf{Com}(\texttt{if } \beta \texttt{ then } \pi_1 \texttt{ else } \pi_2 \texttt{ fi};\pi) \wedge \beta \rightarrow$
$\quad [\texttt{execute}(\texttt{if } \beta \texttt{ then } \pi_1 \texttt{ else } \pi_2 \texttt{ fi})]\mathbf{Com}(\pi_1 \circ \pi)$
7 $\models \mathbf{Com}(\texttt{if } \beta \texttt{ then } \pi_1 \texttt{ else } \pi_2 \texttt{ fi};\pi) \wedge \neg\beta \rightarrow$
$\quad [\texttt{execute}(\texttt{if } \beta \texttt{ then } \pi_1 \texttt{ else } \pi_2 \texttt{ fi})]\mathbf{Com}(\pi_2 \circ \pi)$

The first, second and third properties state that if the specified atomic actions can be executed, they have the specified commitments as a result. The fourth action needs a condition on the plan of the agent ($\mathbf{Com}(\pi_h \circ \pi)$). Without this condition, no statement can be made about the resulting plan. The same holds for the rest of the meta-actions. The fifth property specifies the result of the meta-action `execute`(a) if the result on the belief base of executing the basic action a is known. The result of execution of the meta-action `execute`(`if` β `then` π_1 `else` π_2 `fi`) depends on the truth of β.

The last three properties of the first proposition specify implications and not equivalences, as was explained. If an assumption is made about the plan after execution of the meta-actions in question, the following equivalences can be specified.

Proposition (*equivalences*)

4	$\models \mathbf{Com}(\pi_h \circ \pi) \wedge \beta$	$\leftrightarrow$	$\langle \texttt{applyRule}(\rho)\rangle \mathbf{Com}(\pi_b \circ \pi)$
5	$\models \mathbf{Com}(a;\pi) \wedge \langle a\rangle\top$	$\leftrightarrow$	$\langle \texttt{execute}(a)\rangle \mathbf{Com}(\pi)$
6	$\models \mathbf{Com}(\texttt{if } \beta \texttt{ then } \pi_1 \texttt{ else } \pi_2 \texttt{ fi};\pi)$	$\leftrightarrow$	$\langle \texttt{execute}(\texttt{if } \beta \texttt{ then } \pi_1 \texttt{ else } \pi_2 \texttt{ fi})\rangle \mathbf{Com}(\pi)$

The realizability (1) and result (2 - 7) of the basic action $move(x, y, z)$ which was defined in the previous chapter are specified below.

Proposition (*realizability and result of basic action* $move(x, y, z)$)

1	$\models \mathbf{B}(clear(x) \wedge clear(z) \wedge on(x, y))$	$\leftrightarrow$	$\langle move(x, y, z)\rangle\top$
2	$\models$		$[move(x, y, z)]\mathbf{B}(on(x, z) \wedge clear(y) \wedge \neg on(x, y))$
3	$\models$		$[move(x, y, z)]\mathbf{B}(\neg clear(z))$ where $z \neq Fl$
4	$\models \mathbf{B}(on(u, v))$	$\rightarrow$	$[move(x, y, z)]\mathbf{B}(on(u, v))$ where $u \neq x$
5	$\models \mathbf{B}(\neg on(u, v))$	$\rightarrow$	$[move(x, y, z)]\mathbf{B}(\neg on(u, v))$ where $(u \neq x) \vee (v \neq z)$
6	$\models \mathbf{B}(clear(w))$	$\rightarrow$	$[move(x, y, z)]\mathbf{B}(clear(w))$ where $w \neq z$
7	$\models \mathbf{B}(\neg clear(w))$	$\rightarrow$	$[move(x, y, z)]\mathbf{B}(\neg clear(w))$ where $w \neq y$

Variables are used as a means for abbreviation. Variables should be thought of as being instantiated with the relevant arguments in such a way that predicates with variables reduce to propositions. If a condition on variables is specified, these variables can only be instantiated with arguments if the instantiation satisfies the condition.

As was explained in the previous section, actions are deterministic. A consequence of this is, that the formula $\langle\alpha\rangle\mu$ is stronger than the formula $[\alpha]\mu$. The diamond formula expresses that μ holds in the state resulting from executing α. The box formula only states that *if* α can be executed, μ will hold in the resulting state. The following equivalence holds: $\langle\alpha\rangle\mu \leftrightarrow (\langle\alpha\rangle\top \wedge [\alpha]\mu)$. The conditions on realizability $\langle\alpha\rangle\top$ and results $[\alpha]\mu$ of actions can thus be combined, deriving the formula $\langle\alpha\rangle\mu$.

5.4 Example proof

In this section I will give an example of a proof of a property of the Dribble agent of section 4.3. The property is that it is possible to realise the goal from the initial belief state with a certain sequence of meta-actions (where the goal and initial belief state are as specified in section 4.1). I will need the following instantiations of the goal rule of section 4.3.1.

$$\begin{array}{llll}
g_1 : \mathbf{B}(on(C, A)) & \wedge & \mathbf{G}(on(C, Fl)) & \rightarrow & move(C, A, Fl) \\
g_2 : \mathbf{B}(on(B, Fl)) & \wedge & \mathbf{G}(on(B, C)) & \rightarrow & move(B, Fl, C) \\
g_3 : \mathbf{B}(on(A, Fl)) & \wedge & \mathbf{G}(on(A, B)) & \rightarrow & move(A, Fl, B)
\end{array}$$

The initial mental state s_0 of the agent is the following:

$$\begin{array}{lcl}\sigma_0 & = & \{on(A,Fl) \wedge on(B,Fl) \wedge on(C,A) \wedge clear(B) \wedge clear(C) \wedge clear(Fl)\},\\ \gamma_0 & = & \{on(A,B) \wedge on(B,C) \wedge on(C,Fl)\},\\ \pi_0 & = & E.\end{array}$$

The following formula is true in that mental state:

$$\begin{array}{lcl}s_0 & \models & \langle \texttt{applyRule}(g_1); \texttt{execute}(move(C,A,Fl)); \texttt{applyRule}(g_2); \texttt{execute}(move(B,Fl,C));\\ & & \texttt{applyRule}(g_3); \texttt{execute}(move(A,Fl,B))\rangle\ \mathbf{B}(on(A,B) \wedge on(B,C) \wedge on(C,Fl)).\end{array}$$

The formula specifies that in state s_0, the following is true: the specified sequence of rule applications and action executions results in a state in which the agent believes that the goal tower is built. The mental state resulting from execution of the specified sequence of meta-actions satisfies the following:

$$\begin{array}{lcl}\sigma_6 & \models & \{on(A,B) \wedge on(B,C) \wedge on(C,Fl) \wedge clear(A) \wedge clear(Fl)\},\\ \gamma_6 & = & \emptyset,\\ \pi_6 & = & E.\end{array}$$

This mental state is obtained by applying the definitions of the meta-actions $\texttt{applyRule}(g)$ and $\texttt{execute}(a)$ and the definition of a sequence of meta-actions $\texttt{r}^*$ (see section 5.2.3). First the mental state s_1 resulting from execution of the meta-action $\texttt{applyRule}(g_1)$ in s_0 is derived. Then the mental state s_2 resulting from executing the meta-action $\texttt{execute}(move(C,A,Fl))$ in s_1 is derived. The mental state resulting from execution of the complete sequence of meta-actions is s_6 and is specified above. In this mental state, the belief formula $\mathbf{B}(on(A,B) \wedge on(B,C) \wedge on(C,Fl))$ is true and thus the property of the example Dribble agent is proven.

Chapter 6

Correspondence between logic and operational semantics

In chapter 3, the meaning of a Dribble agent was defined as the set of computation runs, starting in the initial mental state of the agent. A computation run is a series of mental state transitions, where each transition is a transition in the transition system for the Dribble agent. In chapter 5, a dynamic logic was defined in which one can reason about actions defined in that logic. These actions transform the mental state of the agent, expressed with the function $\mathtt{r}^*$. The idea is, that mental state transitions defined by actions in the logic, correspond to the mental state transitions defined by the transition system. If this were the case, properties derived for actions in the logic are actually properties of the Dribble agent. This is because the meaning of a Dribble agent is defined in terms of these transitions in the transition system. In this chapter I will prove this correspondence between logic and operational semantics.

6.1 Computation run of θ

The actions that can be reasoned about in the logic are basic actions and meta-actions. These do not all correspond to a mental state transition in the transition system. The meta-actions `commit` and `uncommit` for instance were only defined because it made the definition of other meta-actions easier. They do not have a counterpart in the transition system. The same holds for the basic actions: in the logic one can reason about the result of a basic action with respect to the beliefs and goals of the agent, regardless of whether the basic action is part of the plan of the agent. The only transition defined in the transition system with respect to a basic action, is however the execution of it if it is a prefix of the plan.

The meta-actions that do have a counterpart in the transition system are the actions $\mathtt{applyRule}(g), \mathtt{applyRule}(\rho), \mathtt{execute}(a)$ and $\mathtt{execute}(\mathtt{if}\ \beta\ \mathtt{then}\ \pi_1\ \mathtt{else}\ \pi_2\ \mathtt{fi})$. I call these actions *transition actions.* The mental state transitions defined by transition actions, correspond to the mental state transitions defined by the transition system. This will be proven in section 6.2. First I will however define sequences of transitions actions.

Definition (*sequences of transition actions*)
The set $\mathsf{TransitionAction}$ with typical element θ is defined below, where $\texttt{applyRule}(g)$, $\texttt{applyRule}(\rho), \texttt{execute}(b) \in \mathsf{MetaAction}$:

- if $t \in \{\texttt{applyRule}(g), \texttt{applyRule}(\rho), \texttt{execute}(b)\}$ then $t \in \mathsf{TransitionAction}$,
- if $t \in \{\texttt{applyRule}(g), \texttt{applyRule}(\rho), \texttt{execute}(b)\}$ and $\theta \in \mathsf{TransitionAction}$ then $t\,;\theta \in \mathsf{TransitionAction}$.

In the logic, one can reason about sequences of transition actions θ $(= t_1; \ldots ; t_n)$. What I want to prove is, that the mental state resulting from execution of θ, corresponds to the mental state resulting from a sequence of transitions in the transition system. This sequence of transitions is not just any sequence. The elements of the sequence of transitions have a one to one correspondence with the elements of the sequence of transition actions θ: each transition $s_i \rightarrow_{t_i} s_{i+1}$ corresponds to the transition action t_i in the sequence θ. I express this correspondence using the concept of a *computation run of* θ.

Definition (*computation run of* θ)
A computation run of θ $\texttt{CR}_\theta(s_0)$ for a Dribble agent with goal rules Γ and PR rules Δ is a finite sequence $s_0, t_1, s_1, \ldots, t_n, s_n$ where:

- $s_i \in \Sigma$ are mental states,
- $\forall_{i>0} : \; s_{i-1} \rightarrow_{t_i} s_i$ is a transition in the transition system for the Dribble agent,
- $\theta = t_1; \ldots ; t_n$.

6.2 Correspondence

6.2.1 Theorem

In the following theorem I express the correspondence between logic and operational semantics. The function `last` yields the last state of the finite computation run. In the next section I will proof this theorem.

Theorem (*equivalence of operational semantics and actions in the logic*)
For $\theta \in \mathsf{TransitionAction}$ and $s_0, s_n \in \Sigma$, the following holds for a Dribble agent with goal rules Γ and PR rules Δ :

$$\texttt{r}^*(\theta)\ s_0 = s_n \Leftrightarrow \texttt{last}(\texttt{CR}_\theta(s_0)) = s_n$$

6.2.2 Proof

The inductive proof of the theorem above is constructed by proving the theorem for every transition action t and then proving that it also holds for sequences of transition actions $t\,;\theta'$. The proofs of the various cases should be read as follows: statements derived in a certain step, can be derived from results of all previous derivation steps. First the theorem is proven from left to right, followed by a proof from right to left.

Left to right

The first four cases below which prove the theorem for a specific transition action, all have the same structure. First the left part of the theorem is instantiated for a specific transition action (t). A mental state not equal to $\mathcal{E}$ thus results from execution of t. From this and from the definition of $\mathtt{r}^*(t)$, conclusions can be drawn about the mental states s_i and s_{i+1} if $\mathtt{r}^*(t)s_i = s_{i+1}$. With these conclusions, a transition $s_i \rightarrow_t s_{i+1}$ can be derived. This is the computation run $\mathtt{CR}_t(s_i)$. The last mental state of this computation run is s_{i+1}. End of proof.

Case 1 ($\theta = \mathtt{applyRule}(g)$)
$g : \varphi \rightarrow \pi_g \in \Gamma$

$$\begin{aligned}
\mathtt{r}^*(\mathtt{applyRule}(g))\langle\sigma,\gamma,\pi\rangle = \langle\sigma',\gamma',\pi'\rangle \;&\Rightarrow\; \sigma = \sigma',\ \gamma = \gamma',\ \pi = E,\ \langle\sigma,\gamma,\pi\rangle \models \varphi,\ \pi' = \pi_g \\
&\Rightarrow\; \langle\sigma,\gamma,\pi\rangle \rightarrow_{\mathtt{applyRule}(g)} \langle\sigma',\gamma',\pi'\rangle \\
&\Rightarrow\; \mathtt{last}\Big(\mathtt{CR}_{\mathtt{applyRule}(g)}(\langle\sigma,\gamma,\pi\rangle)\Big) = \langle\sigma',\gamma',\pi'\rangle
\end{aligned}$$

Case 2 ($\theta = \mathtt{applyRule}(\rho)$)
$\rho : \pi_h \mid \beta \rightarrow \pi_b \in \Delta$

$$\begin{aligned}
\mathtt{r}^*(\mathtt{applyRule}(\rho))\langle\sigma,\gamma,\pi\rangle = \langle\sigma',\gamma',\pi'\rangle \;&\Rightarrow\; \sigma = \sigma',\ \gamma = \gamma',\ \pi = \pi_h \circ \pi_r,\ \pi' = \pi_b \circ \pi_r, \\
&\qquad \langle\sigma,\gamma,\pi\rangle \models \beta \\
&\Rightarrow\; \langle\sigma,\gamma,\pi_h\rangle \models \beta \\
&\Rightarrow\; \langle\sigma,\gamma,\pi_h\rangle \rightarrow_{\mathtt{applyRule}(\rho)} \langle\sigma',\gamma',\pi_b\rangle \\
&\Rightarrow\; \langle\sigma,\gamma,\pi_h \circ \pi_r\rangle \rightarrow_{\mathtt{applyRule}(\rho)} \langle\sigma',\gamma',\pi_b \circ \pi_r\rangle \\
&\Rightarrow\; \langle\sigma,\gamma,\pi\rangle \rightarrow_{\mathtt{applyRule}(\rho)} \langle\sigma',\gamma',\pi'\rangle \\
&\Rightarrow\; \mathtt{last}\Big(\mathtt{CR}_{\mathtt{applyRule}(\rho)}(\langle\sigma,\gamma,\pi\rangle)\Big) = \langle\sigma',\gamma',\pi'\rangle
\end{aligned}$$

In cases two, three and four the transition rule *execution of sequential composition* of section 3.2.5 is used to derive a transition $\langle\sigma,\gamma,\pi \circ \pi_r\rangle \rightarrow_t \langle\sigma',\gamma',\pi' \circ \pi_r\rangle$ from $\langle\sigma,\gamma,\pi\rangle \rightarrow_t \langle\sigma',\gamma',\pi'\rangle$.

Furthermore, the second derivation step of case 2 needs some explaining. The truth of a belief formula β in a mental state depends on the belief base σ in that state. It does not depend on the plan. If a belief formula is true in a mental state, it is therefore true in any state with the same belief base, regardless of the plan.

Case 3 ($\theta = \mathtt{execute}(a)$)

$$\begin{aligned}
\mathtt{r}^*(\mathtt{execute}(a))\langle\sigma,\gamma,\pi\rangle = \langle\sigma',\gamma',\pi'\rangle \;&\Rightarrow\; \mathcal{T}(a,\sigma) = \sigma',\ \gamma \setminus \{\psi \in \gamma \mid \mathcal{T}(a,\sigma) \models \psi\} = \gamma', \\
&\qquad \pi = a \circ \pi_r,\ \pi' = \pi_r \\
&\Rightarrow\; \langle\sigma,\gamma,a\rangle \rightarrow_{\mathtt{execute}(a)} \langle\sigma',\gamma',E\rangle \\
&\Rightarrow\; \langle\sigma,\gamma,a \circ \pi_r\rangle \rightarrow_{\mathtt{execute}(a)} \langle\sigma',\gamma',\pi_r\rangle \\
&\Rightarrow\; \langle\sigma,\gamma,\pi\rangle \rightarrow_{\mathtt{execute}(a)} \langle\sigma',\gamma',\pi'\rangle \\
&\Rightarrow\; \mathtt{last}\Big(\mathtt{CR}_{\mathtt{execute}(a)}(\langle\sigma,\gamma,\pi\rangle)\Big) = \langle\sigma',\gamma',\pi'\rangle
\end{aligned}$$

Case 4 ($\theta = \texttt{execute(if}\ \beta\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2\ \texttt{fi)}$)

Let *ite* be $\texttt{if}\ \beta\ \texttt{then}\ \pi_1\ \texttt{else}\ \pi_2\ \texttt{fi}$.

$$\begin{array}{lll}
\texttt{r}^*(\texttt{execute}(ite))\langle\sigma,\gamma,\pi\rangle = \langle\sigma',\gamma',\pi'\rangle, & & \\
\langle\sigma,\gamma,\pi\rangle \models \beta & \Rightarrow & \sigma = \sigma',\ \gamma = \gamma',\ \pi = ite \circ \pi_r,\ \pi' = \pi_1 \circ \pi_r, \\
 & & \langle\sigma,\gamma,ite\rangle \models \beta \\
 & \Rightarrow & \langle\sigma,\gamma,ite\rangle \rightarrow_{\texttt{execute}(ite)} \langle\sigma',\gamma',\pi_1\rangle \\
 & \Rightarrow & \langle\sigma,\gamma,ite \circ \pi_r\rangle \rightarrow_{\texttt{execute}(ite)} \langle\sigma',\gamma',\pi_1 \circ \pi_r\rangle \\
 & \Rightarrow & \langle\sigma,\gamma,\pi\rangle \rightarrow_{\texttt{execute}(ite)} \langle\sigma',\gamma',\pi'\rangle \\
 & \Rightarrow & \texttt{last}\Big(\texttt{CR}_{\texttt{execute}(ite)}(\langle\sigma,\gamma,\pi\rangle)\Big) = \langle\sigma',\gamma',\pi'\rangle \\
\texttt{r}^*(\texttt{execute}(ite))\langle\sigma,\gamma,\pi\rangle = \langle\sigma',\gamma',\pi'\rangle, & & \\
\langle\sigma,\gamma,\pi\rangle \models \neg\beta & \Rightarrow & \sigma = \sigma',\ \gamma = \gamma',\ \pi = ite \circ \pi_r,\ \pi' = \pi_2 \circ \pi_r, \\
 & & \langle\sigma,\gamma,ite\rangle \models \neg\beta \\
 & \Rightarrow & \langle\sigma,\gamma,ite\rangle \rightarrow_{\texttt{execute}(ite)} \langle\sigma',\gamma',\pi_2\rangle \\
 & \Rightarrow & \langle\sigma,\gamma,ite \circ \pi_r\rangle \rightarrow_{\texttt{execute}(ite)} \langle\sigma',\gamma',\pi_2 \circ \pi_r\rangle \\
 & \Rightarrow & \langle\sigma,\gamma,\pi\rangle \rightarrow_{\texttt{execute}(ite)} \langle\sigma',\gamma',\pi'\rangle \\
 & \Rightarrow & \texttt{last}\Big(\texttt{CR}_{\texttt{execute}(ite)}(\langle\sigma,\gamma,\pi\rangle)\Big) = \langle\sigma',\gamma',\pi'\rangle
\end{array}$$

Case 4 is split up into two parts: one for the case that the belief condition of the if-then-else construct is true and one for the case that it is false.

Case 5 ($\theta = t\,;\theta'$)

$$\begin{array}{lll}
\texttt{r}^*(t)\ s_0 = s_1 & \Rightarrow & \texttt{last}(\texttt{CR}_t(s_0)) = s_1\ \ \text{(induction hypothesis)} \\
\texttt{r}^*(\theta')\ s_1 = s_n & \Rightarrow & \texttt{last}(\texttt{CR}_{\theta'}(s_1)) = s_n \\
 & & \\
\texttt{r}^*(t\,;\theta')\ s_0 = s_n & \Rightarrow & \texttt{r}^*(\theta')(\texttt{r}^*(t)\ s_0) = s_n \\
 & \Rightarrow & \texttt{r}^*(\theta')\ s_1 = s_n \text{ and } s_1 = \texttt{r}^*(t)\ s_0 \\
 & \Rightarrow & \texttt{last}(\texttt{CR}_{\theta'}(s_1)) = s_n \text{ and } \texttt{last}(\texttt{CR}_t(s_0)) = s_1 \\
 & \Rightarrow & \texttt{CR}_t(s_0) = s_0, t, s_1 \text{ and } \texttt{CR}_{\theta'}(s_1) = s_1, t_2, s_2, \ldots, t_n, s_n \\
 & & \text{where } \theta' = t_2; \ldots; t_n \\
 & \Rightarrow & \texttt{CR}_{t;\theta'}(s_0) = s_0, t, s_1, t_2, s_2, \ldots, t_n, s_n \\
 & \Rightarrow & \texttt{last}(\texttt{CR}_{t;\theta'}(s_0)) = s_n
\end{array}$$

In case 5, the theorem is proven for arbitrary sequences of transition actions. First, the induction hypothesis is stated. Then the left part of the theorem is instantiated for sequences of transition actions $t\,;\theta'$. This is rewritten in the first derivation step using the definition of $\texttt{r}^*$ of section 5.2.3. This is rewritten again in the second derivation step, introducing the state s_1. In the next step, the induction hypothesis is used. The computation runs of t and θ' introduced in this step, are specified in the fourth step. In the fifth step, these computation runs are combined, yielding the computation run of $t\,;\theta'$. The last mental state of this computation run is specified in the last derivation step. End of proof.

Right to left

Case 1 ($\theta = \texttt{applyRule}(g)$)
$g : \varphi \to \pi_g \in \Gamma$

$$\begin{aligned}
\texttt{last}\Big(\texttt{CR}_{\texttt{applyRule}(g)}(\langle\sigma,\gamma,\pi\rangle)\Big) = \langle\sigma',\gamma',\pi'\rangle \quad &\Rightarrow \quad \langle\sigma,\gamma,\pi\rangle \to_{\texttt{applyRule}(g)} \langle\sigma',\gamma',\pi'\rangle \\
&\Rightarrow \quad \sigma = \sigma',\ \gamma = \gamma',\ \pi = E,\ \langle\sigma,\gamma,\pi\rangle \models \varphi, \\
&\qquad\ \pi' = \pi_g \\
&\Rightarrow \quad \texttt{r}^*(\texttt{applyRule}(g))\langle\sigma,\gamma,\pi\rangle = \langle\sigma',\gamma',\pi'\rangle
\end{aligned}$$

The proof of the theorem from left to right for application of a goal rule can be reversed, yielding the proof from right to left. First the right part of the theorem is instantiated for the transition action $\texttt{applyRule}(g)$. From this, a transition can be derived in the first step, using the definition of *computation run of* $\texttt{applyRule}(g)$. In section 3.2, it was stated that the relation $\to_t$ defined by a transition system, is the smallest relation which contains all the axioms of the system and all conclusions which are derivable by using these axioms. Thus a transition $s_i \to_{\texttt{applyRule}(g)} s_{i+1}$ must have been derived with the transition rule for *application of a goal rule* of section 3.2.1. Using this transition rule reversed, conclusions can be drawn about the belief base, goal base and plans of mental states s_i and s_{i+1}. With these conclusions and the definition of $\texttt{r}^*(\texttt{applyRule}(g))(s_i)$ of section 5.2.3, the last derivation step can be made.

Case 2 ($\theta = \texttt{applyRule}(\rho)$)
$\rho : \pi_h \mid \beta \to \pi_b \in \Delta$

$$\begin{aligned}
\texttt{last}\Big(\texttt{CR}_{\texttt{applyRule}(\rho)}(\langle\sigma,\gamma,\pi\rangle)\Big) = \langle\sigma',\gamma',\pi'\rangle \quad &\Rightarrow \quad \langle\sigma,\gamma,\pi\rangle \to_{\texttt{applyRule}(\rho)} \langle\sigma',\gamma',\pi'\rangle \\
&\Rightarrow \quad \ldots \text{ (see text)} \\
&\Rightarrow \quad \langle\sigma,\gamma,\pi_h\rangle \to_{\texttt{applyRule}(\rho)} \langle\sigma',\gamma',\pi_b\rangle, \\
&\qquad\ \pi = \pi_h \circ \pi_r,\ \pi' = \pi_b \circ \pi_r \\
&\Rightarrow \quad \sigma = \sigma',\ \gamma = \gamma',\ \langle\sigma,\gamma,\pi_h\rangle \models \beta \\
&\Rightarrow \quad \langle\sigma,\gamma,\pi\rangle \models \beta \\
&\Rightarrow \quad \texttt{r}^*(\texttt{applyRule}(\rho))\langle\sigma,\gamma,\pi\rangle = \langle\sigma',\gamma',\pi'\rangle
\end{aligned}$$

The proof of the theorem from left to right for cases two, three and four cannot be reversed. I will explain the proof from right to left of case two, but the same kind of argument can be constructed for cases three and four.

First, the right part of the theorem is instantiated with the meta-action $\texttt{applyRule}(\rho)$. From this, the transition $\langle\sigma,\gamma,\pi\rangle \to_{\texttt{applyRule}(\rho)} \langle\sigma',\gamma',\pi'\rangle$ can be derived in the first step, using the definition of *computation run of* $\texttt{applyRule}(\rho)$. From this transition we want to derive the transition $\langle\sigma,\gamma,\pi_h\rangle \to_{\texttt{applyRule}(\rho)} \langle\sigma',\gamma',\pi_b\rangle$. Using this transition and the transition rule for *application of a PR rule* of section 3.2.2, we could then conclude that $\langle\sigma,\gamma,\pi\rangle \models \beta$. If we can also conclude that $\pi = \pi_h \circ \pi_r$ and $\pi' = \pi_b \circ \pi_r$, we can derive $\texttt{r}^*(\texttt{applyRule}(\rho))\langle\sigma,\gamma,\pi\rangle = \langle\sigma',\gamma',\pi'\rangle$.

I will now explain why it can be concluded that the transition $\langle\sigma,\gamma,\pi_h\rangle \to_{\texttt{applyRule}(\rho)} \langle\sigma',\gamma',\pi_b\rangle$ (named 1) must be a transition in the transition system if the transition $\langle\sigma,\gamma,\pi\rangle \to_{\texttt{applyRule}(\rho)} \langle\sigma',\gamma',\pi'\rangle$ (named n) is a transition. The reason is, that transition

n can only be derived in a transition system if transition 1 can be derived. Transition n is a transition describing a PR rule application (specified by the subscript of the transition). Such a transition can only be derived in the transition system by applying the transition rule for *application of a PR rule* or by applying the transition rule for *execution of sequential composition*. This last transition rule can be applied to a PR rule transition and the result is a new PR rule transition. This rule thus can only be applied if a PR rule transition has been derived using the transition rule for application of a PR rule. Now the assumption is that PR rule transition n can be derived in the transition system. This transition must have been derived using one of the two transition rules that were mentioned above. Regardless however of whether transition n was derived with one or the other transition rule, we can conclude that transition 1 must have been derived in the first derivation step of the derivation of n: without transition 1, the transition rule for sequential composition cannot be applied to yield a PR rule transition. Thus, from the assumption that transition n is a transition in the transition system, we can conclude that transition 1 must be a transition. A derivation of n from 1 exists in the transition system and this derivation can be reversed if we assume that n is a transition. This can be done because the relation $\rightarrow_t$ defined by a transition system, is the smallest relation which contains all the axioms of the system and all conclusions which are derivable by using these axioms.

If n is derived from 1, π must be of the form $\pi_h \circ \pi_r$ and π' must be of the form $\pi_b \circ \pi_r$. The maximum length of the derivation of n from 1 depends on the length of π_r. If $\pi_r = E$, then $n = 1$ and the length of the derivation is zero. If the length of π_r is k, the maximum length of the derivation is k: the rule for sequential composition can be applied k times, eventually yielding transition n. Instead of applying the rule for sequential composition k times, it could be applied only once. In that case, transition n is derived directly from 1. As was explained above, the derivation of n from 1 can be reversed, deriving 1 from n. This is indicated by the dots (...) in case 2 above: a derivation of 1 from n exists, but the length of the derivation is unknown.

In summary, the transition $\langle \sigma, \gamma, \pi_h \rangle \rightarrow_{\texttt{applyRule}(\rho)} \langle \sigma', \gamma', \pi_b \rangle$ can be derived from $\langle \sigma, \gamma, \pi \rangle \rightarrow_{\texttt{applyRule}(\rho)} \langle \sigma', \gamma', \pi' \rangle$. π must be of the form $\pi_h \circ \pi_r$ and π' must be of the form $\pi_b \circ \pi_r$. The maximum length of the derivation depends on the length of π_r.

Case 3 ($\theta = \texttt{execute}(a)$)

$$\begin{array}{rcl}
\texttt{last}\Big(\texttt{CR}_{\texttt{execute}(a)}(\langle \sigma, \gamma, \pi \rangle)\Big) = \langle \sigma', \gamma', \pi' \rangle & \Rightarrow & \langle \sigma, \gamma, \pi \rangle \rightarrow_{\texttt{execute}(a)} \langle \sigma', \gamma', \pi' \rangle \\
& \Rightarrow & \ldots \text{ (see text)} \\
& \Rightarrow & \langle \sigma, \gamma, a \rangle \rightarrow_{\texttt{execute}(a)} \langle \sigma', \gamma', E \rangle, \\
& & \pi = a \circ \pi_r,\ \pi' = E \circ \pi_r = \pi_r \\
& \Rightarrow & \mathcal{T}(a, \sigma) = \sigma',\ \gamma \setminus \{\psi \in \gamma \mid \mathcal{T}(a, \sigma) \models \psi\} = \gamma' \\
& \Rightarrow & \texttt{r}^*(\texttt{execute}(a))\langle \sigma, \gamma, \pi \rangle = \langle \sigma', \gamma', \pi' \rangle
\end{array}$$

Case 4 ($\theta = \texttt{execute(if } \beta \texttt{ then } \pi_1 \texttt{ else } \pi_2 \texttt{ fi)}$)

Let *ite* be $\texttt{if } \beta \texttt{ then } \pi_1 \texttt{ else } \pi_2 \texttt{ fi}$.

$$\begin{array}{rll}
\texttt{last}\Big(\texttt{CR}_{\texttt{execute}(ite)}(\langle\sigma,\gamma,\pi\rangle)\Big) = \langle\sigma',\gamma',\pi'\rangle & \Rightarrow & \langle\sigma,\gamma,\pi\rangle \rightarrow_{\texttt{execute}(ite)} \langle\sigma',\gamma',\pi'\rangle \\
& \Rightarrow & \ldots \text{ (see text)} \\
& \Rightarrow & \langle\sigma,\gamma,ite\rangle \rightarrow_{\texttt{execute}(ite)} \langle\sigma',\gamma',\pi_1\rangle, \\
& & \pi = ite \circ \pi_r,\ \pi' = \pi_1 \circ \pi_r \\
& \Rightarrow & \sigma = \sigma',\ \gamma = \gamma',\ \langle\sigma,\gamma,ite\rangle \models \beta \\
& \Rightarrow & \langle\sigma,\gamma,\pi\rangle \models \beta \\
& \Rightarrow & \texttt{r}^*(\texttt{execute}(ite))\langle\sigma,\gamma,\pi\rangle = \langle\sigma',\gamma',\pi'\rangle
\end{array}$$

$$\begin{array}{rll}
\texttt{last}\Big(\texttt{CR}_{\texttt{execute}(ite)}(\langle\sigma,\gamma,\pi\rangle)\Big) = \langle\sigma',\gamma',\pi'\rangle & \Rightarrow & \langle\sigma,\gamma,\pi\rangle \rightarrow_{\texttt{execute}(ite)} \langle\sigma',\gamma',\pi'\rangle \\
& \Rightarrow & \ldots \text{ (see text)} \\
& \Rightarrow & \langle\sigma,\gamma,ite\rangle \rightarrow_{\texttt{execute}(ite)} \langle\sigma',\gamma',\pi_2\rangle, \\
& & \pi = ite \circ \pi_r,\ \pi' = \pi_2 \circ \pi_r \\
& \Rightarrow & \sigma = \sigma',\ \gamma = \gamma',\ \langle\sigma,\gamma,ite\rangle \models \neg\beta \\
& \Rightarrow & \langle\sigma,\gamma,\pi\rangle \models \neg\beta \\
& \Rightarrow & \texttt{r}^*(\texttt{execute}(ite))\langle\sigma,\gamma,\pi\rangle = \langle\sigma',\gamma',\pi'\rangle
\end{array}$$

Case 4 is split up into two parts: one for the case that the belief condition of the if-then-else construct is true and one for the case that it is false.

Case 5 ($\theta = t\,;\theta'$)

$$\begin{array}{rll}
\texttt{last}(\texttt{CR}_t(s_0)) = s_1 & \Rightarrow & \texttt{r}^*(t)\ s_0 = s_1 \ \text{ (induction hypothesis)} \\
\texttt{last}(\texttt{CR}_{\theta'}(s_1)) = s_n & \Rightarrow & \texttt{r}^*(\theta')\ s_1 = s_n
\end{array}$$

$$\begin{array}{rll}
\texttt{last}(\texttt{CR}_{t;\theta'}(s_0)) = s_n & \Rightarrow & \texttt{CR}_{t;\theta'}(s_0) = s_0, t, s_1, t_2, s_2, \ldots, t_n, s_n \\
& \Rightarrow & \texttt{CR}_t(s_0) = s_0, t, s_1 \text{ and } \texttt{CR}_{\theta'}(s_1) = s_1, t_2, s_2, \ldots, t_n, s_n \\
& & \text{where } \theta' = t_2; \ldots ; t_n \\
& \Rightarrow & \texttt{last}(\texttt{CR}_{\theta'}(s_1)) = s_n \text{ and } \texttt{last}(\texttt{CR}_t(s_0)) = s_1 \\
& \Rightarrow & \texttt{r}^*(\theta')\ s_1 = s_n \text{ and } s_1 = \texttt{r}^*(t)\ s_0 \\
& \Rightarrow & \texttt{r}^*(\theta')(\texttt{r}^*(t)\ s_0) = s_n \\
& \Rightarrow & \texttt{r}^*(t\,;\theta')\ s_0 = s_n
\end{array}$$

The proof of the theorem from left to right for arbitrary sequences of transition actions can be reversed, yielding the proof from right to left.

Chapter 7

Conclusion

In this paper, I defined the agent programming language Dribble. The language combines features from the languages GOAL and 3APL. It incorporates beliefs and goals, as well as planning features. A Dribble agent selects a plan to get to a goal with its goal rules. PR rules are used to create or modify the plan of the agent. The operational semantics of Dribble was defined using a transition system. Furthermore, I defined a dynamic logic to specify and verify properties of Dribble agents. The logic uses meta-actions to reason about rule application and action execution. Finally, it was proven that mental state transitions defined by the transition system, correspond to transitions defined by meta-actions in the logic. Properties of actions in the logic are thus properties of a Dribble agent.

In the introduction, I mentioned three characteristics of agent behaviour: autonomy, reactivity and pro-activity. A Dribble agent can exhibit each of these kinds of behaviour. A Dribble agent is autonomous because it determines what actions it is going to take. The programmer does not tell the agent what to do; he tells the agent what its goals are and the agent figures out how to get there. The programmer does not specify the order in which the goals of the agent should be reached. A Dribble agent thus has control over its own actions. This is in contrast with a 3APL agent, whom the programmer tells which plans it should execute and in which order. Reactive behaviour can be modelled using reactive PR rules. A Dribble agent can also exhibit pro-active behaviour because it can be programmed to go after its goals. The agent takes the initiative to do something, because it has certain goals. A 3APL agent on the contrary does not have goals and therefore cannot exhibit truly pro-active behaviour. The main difference between a GOAL agent and a Dribble agent is that the GOAL agent cannot adopt and modify plans. As was shown in the example in this paper, this lack of planning features can lead to undesirable behaviour.

One line of research to extend the language Dribble concerns incorporating first order languages. As was shown in the example in chapter 4, this extension is necessary to make the language suitable for practical use. Another extension that has already been mentioned in chapter 4, is the addition of actions to adopt and drop goals. The benefit has been explained in the chapter mentioned. Yet another extension could be the selection of a goal from a wish base. Instead of reasoning with the entire goal base to select a new plan for execution, the agent would reason only with the selected goal. Selection of plans with goal rules would only concern plans aimed at reaching this goal. One could argue that this would lead to more persistent behaviour. The reason is, that it may take more than one goal rule application to reach a certain goal. If the agent would have only one goal to go after, it could only apply goal rules to select a plan to get towards this specific goal. In the current version of Dribble, the agent could first apply a goal rule to get towards goal A and, assuming that this goal is not reached after execution of the adopted plan, apply another rule to go after goal B. This behaviour does not seem very persistent. A third possibility to extend Dribble concerns the commitment strategy. In the current version of Dribble, the agent drops a goal only if it believes it has achieved this goal. A goal that is unachievable will thus remain in the goal base forever. One could try to extend the commitment strategy in such a way that the agent would also drop a goal if it believes that the goal will never be reached. Determining whether a goal is unachievable is however not trivial. One way to determine this, could be to keep track of all rule applications for this goal. If all possiblities for rule application have been tried and the goal is not reached, one could argue that the goal is not achievable, or at least not in those circumstances. If the environment is somehow changed by for instance another agent, it could be that the goal becomes achievable. A second way to modify the commitment strategy is to introduce the concept of maintenance goals. A maintenance goal describes a state of the world which should remain that way after it is achieved. These goals should not be removed from the goal base if achieved. Extending Dribble with these features would result in a programming language with which agents can be programmed that even more closely approximate the intuitive concept of an intelligent agent.

Bibliography

[1] P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990.

[2] D. Harel. *First-Order Dynamic Logic.* Lectures Notes in Computer Science 68. Springer, Berlin, 1979.

[3] D. Harel. Dynamic logic. In D. Gabbay and F. Guenthner, editors, *Handbook of Philosophical Logic*, volume II, pages 497–604. Reidel, Dordrecht/Boston, 1984.

[4] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. Agent programming in 3APL. *Int. J. of Autonomous Agents and Multi-Agent Systems*, 2(4):357–401, 1999.

[5] K. V. Hindriks, F. S. de Boer, W. van der Hoek, and J.-J. Ch. Meyer. Agent programming with declarative goals. In *Intelligent Agents VI - Proceedings of the 7th International Workshop on Agent Theories, Architectures, and Languages (ATAL'2000)*, Lecture Notes in AI. Springer, Berlin, 2001.

[6] D. Kozen and J. Tiuryn. Logics of programs. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 789–840. Elsevier, Amsterdam, 1990.

[7] R. Moore. A formal theory of knowledge and action. In J. Hobbs and R. Moore, editors, *Formal Theories of the Commonsense World*, pages 319–358. Ablex, Norwood NJ, 1985.

[8] G. Plotkin. A structural approach to operational semantics. Technical report, Aarhus University, Computer Science Department, 1981.

[9] V. Pratt. Semantical considerations on floyd-hoare logic. In *Proceedings of the 17th IEEE Symp. on Foundations of Computer Science*, pages 109–121, 1976.

[10] A. S. Rao and M. P. Georgeff. Modeling rational agents within a BDI-architecture. In J. Allen, R. Fikes, and E. Sandewall, editors, *Proceedings of the Second International Conference on Principles of Knowledge Representation and Reasoning (KR'91)*, pages 473–484. Morgan Kaufmann, 1991.

[11] W. van der Hoek, B. van Linder, and J.-J. Ch. Meyer. An integrated modal approach to rational agents. In M. Wooldridge and A. S. Rao, editors, *Foundations of Rational Agency*, Applied Logic Series 14, pages 133–168. Kluwer, Dordrecht, 1998.

[12] M. Wooldridge. Intelligent agents. In G. Weiss, editor, *Multi-Agent Systems*, pages 27–77. The MIT Press, 1999.