Using temporal logic to integrate goals and qualitative preferences into agent programming*

Koen V. Hindriks[1] and M. Birna van Riemsdijk[2]

[1] EEMCS, Delft University of Technology, Delft, The Netherlands
[2] LMU, Munich, Germany

Abstract. The core capability of a rational agent is to choose its next action in a rational fashion, a capability that can be put to good use by a designer to satisfy the design objectives of an agent system. In agent programming languages for rational agents, such choices are derived from the agent's beliefs and goals. At any one time, an agent can typically choose from multiple actions, which may all lead to goal achievement. Existing approaches usually select one of those actions non-deterministically. In this paper, we propose the use of goals as *hard constraints* and qualitative preferences as *soft constraints* for choosing a most preferred action among the available ones. We use temporal logic for the representation of various kinds of goals and preferences, leading to a uniform framework for integrating goals and preferences into agent programming.

1 Introduction

The core component of a rational agent is its capability to make rational choices of action, in order to satisfy its design objectives. In agent programming languages for rational agents, such choices are derived from the agent's beliefs and goals. These goals are often achievement goals, i.e., goals that define states that are to be achieved. Typically, agent programs consist of a number of rules (usually called action selection or plan rules) that provide the agent with the means to choose actions that achieve its goals. Such rule sets, however, do not necessarily completely determine the choice of action, and, at any one time, an agent can typically choose from *multiple* actions, which may all lead to goal achievement. Existing approaches usually select one of those actions non-deterministically.

Recent research in both the planning and agent programming field [1, 2, 10, 12, 13, 15] has shown that it can be useful to have additional mechanisms for further *constraining* the choice of action in an agent program and thus to reduce the non-determinism that would be present in an agent program without such mechanisms. In [13], for example, we introduced an operational semantics for so-called *maintenance goals* which express conditions that must remain true throughout (some part of) the agent's lifetime. Adding maintenance goals to agent programs provides a programmer with a tool to restrict the options for action selection that an agent has, since the agent is programmed to avoid selecting actions that would violate maintenance goals.

* This work has been sponsored by the project SENSORIA, IST-2005-016004

In this paper, we investigate such mechanisms for further constraining an agent's choice of action in agent programming languages in a more general and uniform setting. We distinguish between *hard constraints* which *must* be satisfied, and *soft constraints* or preferences which allow an agent to distinguish preferred courses of action from those that are less preferred. The main contribution of this paper consists of the layered rational action selection architecture (*RASA* for short) that we propose as a conceptual framework for rational action selection in agent programming. The RASA architecture introduces a uniform framework based on temporal logic that integrates rule-based action selection (based on beliefs and goals), hard constraints such as maintenance goals, and soft constraints such as preferences.

We show how the proposed architecture can be operationalized using the GOAL agent programming language [7]. For this purpose, we introduce an extension of the GOAL language that incorporates temporal logic operators used to represent both goals as well as preferences. A key motivation for integrating temporal logic into agent programming languages is the potential that the additional expressiveness thus introduced provides for defining a uniform framework that naturally allows for the integration of hard and soft constraints, including such concepts as *achievement goals*, *maintenance goals*, as well as *temporally extended preferences*. The work reported is inspired among others by work in planning where temporal logic is used to "guide" planners through the search space [1] and to select preferred plans [2, 10, 15].

In Section 2, we discuss rational action selection in agent programming, and informally present the rational action selection architecture. In Section 3, we present the extension of GOAL with temporal logic, which forms the first layer of our action selection architecture. Section 4 defines some technical preliminaries with respect to temporal logic, which are needed in Sections 5 and 6 in which we introduce the second and third layer, respectively, of the action selection architecture. In Section 7 we conclude the paper.

2 Rational Action Selection Architecture

An agent program typically does not completely determine the behaviour of the agent in each state as it allows multiple actions to be executed in such a state. Agent programs thus typically *underspecify* an agent's behaviour.[3] For example, consider a carrier agent that needs to bring parcels to two different locations A and B. An agent program for such a carrier agent may instruct the agent to $goto(A)$ and $goto(B)$ but leave unspecified in which order this should be done.

The underspecification of agent behaviour can have benefits from a design point of view. If it does not matter whether the agent executes one action or another for reaching a goal, one can argue that it is more natural to let the agent

[3] It may be the case that interpreters for agent programs are implemented such that they produce the same sequence of actions each time the agent program is executed. However, when read as specifications they typically do not dictate such a unique course of action.

program reflect this by leaving these choices open. However, recent research in both the planning and agent programming field [1, 2, 10, 12, 13, 15] has shown that it can be useful to have additional mechanisms for finding courses of actions to further constrain the choice of action in an agent program. The idea is that *additional selection mechanisms can be introduced on top of an existing agent programming semantics* which can be used by the agent to further restrict the selection among actions. For example, one might wish to specify that going to location A first before going to location B is to be preferred by the carrier agent. Preferences such as these are difficult to code into an agent program, and, we argue, are more naturally introduced as an additional constraint in the agent program. In this particular example, an agent then should check whether a selected (course of) action satisfies such constraints to optimize its performance. In this section, we present a layered rational action selection architecture (RASA) for adding such additional selection mechanisms on top of agent programs.

2.1 An Architecture for Rational Action Selection

Regarding the kinds of additional constraints that may be applied to select among the possible courses of action, we distinguish between *hard constraints* which *must* be satisfied, and *soft constraints* or *preferences*, by means of which one can distinguish more preferred courses of action from less preferred ones (see also [15, 10, 13]).

What one takes as hard constraints varies across approaches. In [15, 10], and more generally in planning, the (achievement) goals themselves are considered to be hard constraints.[4] In [13], maintenance goals are taken as hard constraints, i.e., an agent may never violate a maintenance goal. Preferences can be used to distinguish between optimal and suboptimal courses of action, e.g., in terms of costs, but may also be used to distinguish courses of action with respect to which goals are reached (if goals are not considered as hard constraints and not all goals can be reached).

These considerations lead to the following layered rational action selection architecture.

- **Layer 1**: The RASA *generates* options for courses of action, typically on the basis of an agent's beliefs and (achievement) goals.
- **Layer 2**: The RASA verifies whether the courses of action from layer one satisfy *hard constraints* and discards those that do not.
- **Layer 3**: The RASA selects from the options remaining after the application of layer two those courses of action that maximize satisfaction of *soft constraints*.

In this paper, we show how this architecture can be made concrete in the context of the GOAL agent programming language.

[4] That is, in that approach the initial goals are the basis for plan generation, while at the same time being considered as hard constraints. The initial goals are thus not used as an *additional* action selection mechanism.

2.2 Rational Action Selection in Agent Programming

This rational action selection architecture is inspired among others by research on planning with preferences [10, 15, 4, 2]. The classical AI planning problem is to search for a plan (a sequence of actions) to get from the current state to a goal state, given a set of action specifications [9, 11]. In agent programming, on the other hand, the behaviour of the agent is specified by means of a program [5]. One of the main differences between planning and programming approaches, is that in planning one seeks a *complete plan*, the execution of which will result in the agent achieving its goal (given certain assumptions on the environment). An agent program, on the other hand, is executed *step by step*, typically without first checking whether the executed actions will eventually lead to the agent reaching its goal.[5] That is, in our approach we want the agent to execute an action at each step, even though it does not know for sure that the action is the best one according to the hard and soft constraints. Nevertheless, we *do* want the agent to take into account the constraints to select an action that is at least likely to be a good one.

The way in which we propose to do this, is partly based on our previous work on the incorporation of maintenance goals in agent programming languages [13]. The idea is that the agent has a fixed, usually finite, *lookahead horizon*, i.e., the agent can lookahead a certain number of execution steps. The agent then evaluates the possible courses of action or paths it can take (up to the given horizon) using its constraints, in order to determine which paths are the best. It then takes one step along one of these paths, and the process is repeated.

It is important to note that, when the agent takes a step in a particular direction, it does not have complete knowledge of the outcome of following that path. It can only look forward until its lookahead horizon, but does not know what happens beyond this horizon. This means that the agent may take a step that is suboptimal, i.e., the agent uses its constraints as a *heuristic* for action selection. Constraints are also used as a heuristic in some planning approaches [1, 2], in which they are used to guide the search for an optimal plan.

The technical tool we use for the representation of goals and preferences is linear temporal logic (LTL) [8]. The idea is that if the agent has a temporal formula as a goal, it should try to produce execution traces on which this temporal formula holds, and similarly for preferences. This idea is inspired by work on the representation of goals and qualitative preferences in the context of planning [1, 10, 15, 4, 2]. While LTL formulas are typically evaluated on infinite traces, we need to evaluate LTL formulas on finite traces, since we use a finite lookahead horizon. It turns out that the 3-valued semantics of LTL as proposed in [3] provides a natural solution to this problem, and we use it to incorporate temporal goals and preferences in GOAL.

[5] If the programmer has written a program that is correct (with respect to some specification), of course, the execution will indeed lead to goal achievement (cf. [7] for a verification framework of the agent programming language GOAL.). Verifying agent programs that are executed in dynamic, unpredictable environments, however, is an unsolved problem and a non-trivial undertaking in practice.

3 RASA Layer 1: Temporalized GOAL

In this section, we define the first layer of the RASA architecture in the context of the GOAL agent programming language [7, 14]. We extend the original GOAL language by allowing temporal formulas as goals, rather than only propositional formulas.

3.1 The General Idea

In the GOAL language, an agent selects actions on the basis of its beliefs and goals. A program consists of (1) a set of beliefs, collectively called the *belief base* of the agent, (2) a set of goals, called the *goal base*, (3) an *action specification* which consists of a specification of the pre- and post-conditions of *basic actions* of the agent, and (4) a program section which consists of a set of *actions rules*.

In the original GOAL language, the belief base and goal base are sets of propositional formulas. The goals are interpreted as achievement goals. That is, if a propositional formula ϕ is in the goal base, this informally means that the agent wants to reach a situation in which ϕ is (believed to be) the case. If a basic action is executed, the agent's beliefs change as specified in the pre- and postconditions of the action, and the achievement goals that are believed to be reached through the execution of the action are removed from the goal base.[6] As an example, an action $goto(A)$ would update the belief base to include $at(A)$. An action rule consists of a basic action and a condition on the agent's beliefs and goals. Such a rule expresses that the basic action may be executed, if the condition holds. An action rule might specify that the agent only goes to a location x if a parcel needs to be delivered to x (a goal) and the agent does not believe it is currently at x. During execution, a GOAL agent selects non-deterministically any of its enabled action rules, i.e., an action rule of which the condition holds, and then executes its corresponding basic action. At any one time, typically *multiple* action rules are enabled, i.e., a GOAL program underspecifies an agent's behaviour.

In [13], we have extended the GOAL language with maintenance goals. Just like achievement goals, maintenance goals were also expressed by propositional formulas. A maintenance goal ϕ expresses that the agent wants that ϕ holds continuously throughout the execution of the agent. Both achievement goals and maintenance goals express particular desired properties of the behaviour of the agent. These properties can be expressed conveniently in LTL. LTL has computation traces (sequences of states) as models. An achievement goal for ϕ can be represented by the LTL formula $\Diamond\phi$, specifying that ϕ should hold eventually, i.e., in some state on the computation trace. A maintenance goal for ϕ can be represented by $\Box\phi$, specifying that ϕ should always hold. This idea can be

[6] The idea is that the agent's beliefs also represent the environment, and that basic actions change this environment. However, the environment is not modeled in the formal specification of GOAL, and consequently basic actions update only the belief base.

generalized by realizing that in fact any LTL formula can be used for expressing goals. As an example, $\Diamond at(A)$ may be used to represent the achievement goal of being at location A and $\Box fuel(x) \wedge x > 30$ may be used to represent a maintenance goal of having always at least 30 units of fuel in the tank.

In this paper, we make this idea concrete in the context of the GOAL language by using LTL for the representation of goals. This increases the expressiveness of the language by allowing the representation of all kinds of goals, and allows for the representation of goals in a uniform way. For example, the goal $\phi U \phi'$, which expresses that the agent wants to ensure that ϕ while it is trying to achieve ϕ', can now be expressed easily.

3.2 Formalization

First, we define the LTL language that we use for representing goals. For reasons of simplicity we do not extend the beliefs of an agent in the programming language to temporal formulas. For example, it would be more involved to establish when an agent should drop one of its goals.

The states of the traces on which the LTL formulas are evaluated are belief bases, i.e., goals express how the belief base of the agent should evolve.[7] We assume a language $\mathcal{L}_0$ of propositional logic with typical element ϕ, and the standard entailment relation $\models$. A belief base $\Sigma \subseteq \mathcal{L}_0$ is a set of propositional formulas. Our LTL language contains the standard (temporal) operators of LTL. The difference between our LTL language and standard LTL is that the states of the traces are belief bases, rather than worlds as valuations of propositional atoms. The semantics of non-temporal propositional formulas is thus defined on belief bases. We specify that a propositional formula ϕ holds in a belief base Σ if $\Sigma \models \phi$.

Definition 1 *(linear temporal logic (LTL))*
Let $\phi \in \mathcal{L}_0$. The set of LTL formulas $\mathcal{L}_{LTL}$ with typical element χ is defined as follows.

$$\chi ::= \top \mid \phi \mid \neg\chi \mid \chi_1 \wedge \chi_2 \mid \Diamond\chi \mid \bigcirc\chi_1 \mid \chi U \chi_2$$

Let $t^b = \Sigma_0, \Sigma_1, \ldots$ be an infinite trace of belief bases and let $i \in \mathbb{N}$ be a position in a trace. The semantics of LTL formulas is defined on infinite traces t^b as follows.

$$\begin{array}{ll}
t^b, i \models_{LTL} \top & \\
t^b, i \models_{LTL} \phi & \Leftrightarrow \Sigma_i \models \phi \\
t^b, i \models_{LTL} \neg\chi & \Leftrightarrow t^b, i \not\models_{LTL} \chi \\
t^b, i \models_{LTL} \chi_1 \wedge \chi_2 & \Leftrightarrow t^b, i \models_{LTL} \chi_1 \text{ and } t^b, i \models_{LTL} \chi_1 \\
t^b, i \models_{LTL} \Diamond\chi & \Leftrightarrow \exists k \geq i : t^b, k \models_{LTL} \chi \\
t^b, i \models_{LTL} \bigcirc\chi & \Leftrightarrow t^b, i+1 \models_{LTL} \chi \\
t^b, i \models_{LTL} \chi_1 U \chi_2 & \Leftrightarrow \exists k \geq i : t^b, k \models_{LTL} \chi_2 \text{ and } \forall i \leq l < k : t^b, l \models_{LTL} \chi_1
\end{array}$$

[7] As the idea is that the beliefs also represent the environment of the agent, goals also express desired properties of the environment.

As usual, the always operator is defined in terms of the eventually operator by: $\Box\chi \equiv \neg\Diamond\neg\chi$.

The mental state of a GOAL agent consists of those components that change during execution. That is, a mental state consists of a belief base and a goal base. The goal base is typically denoted by Γ and in temporalized GOAL this is a set of LTL formulas. Mental states should satisfy a number of rationality constraints.

Definition 2 *(Mental States)*
A mental state of a GOAL agent, typically denoted by m, is a pair $\langle \Sigma, \Gamma \rangle$ with $\Sigma \subseteq \mathcal{L}_0$ and $\Gamma \subseteq \mathcal{L}_{LTL}$ where Σ is the belief base, and Γ with typical element χ is the goal base. Additionally, mental states need to satisfy the following *rationality constraints*:

(i) The belief base is consistent: $\Sigma \not\models \bot$,
(ii) The goal base is consistent: $\Gamma \not\models_{LTL} \bot$,
(iii) The goal base does not contain goals that have already been achieved.

The third rationality constraint is implemented by means of the progression operator which will be introduced below (Definition 4).

A GOAL agent derives its choice of action from its beliefs and goals. In order to do so, a GOAL agent inspects its mental state by evaluating so-called *mental state conditions*. The syntax and semantics of these conditions is defined next.

Definition 3 *(Mental State Conditions)*
Let $\phi \in \mathcal{L}_0$, $\chi \in \mathcal{L}_{LTL}$. The language $\mathcal{L}_M$ of mental state conditions, typically denoted by ψ, is defined as follows.

$$\psi ::= \mathbf{B}\phi \mid \mathbf{G}\chi \mid \neg\psi \mid \psi_1 \wedge \psi_2$$

The truth conditions of mental state conditions ψ, relative to a mental state $m = \langle \Sigma, \Gamma \rangle$, are defined as follows.

$$\begin{array}{lll} m \models_m \mathbf{B}\phi & \text{iff} & \Sigma \models \phi \\ m \models_m \mathbf{G}\chi & \text{iff} & \Gamma \models_{LTL} \chi \\ m \models_m \neg\psi & \text{iff} & m \not\models_m \psi \\ m \models_m \psi_1 \wedge \psi_2 & \text{iff} & m \models_m \psi_1 \text{ and } m \models_m \psi_2 \end{array}$$

The semantics of $\mathbf{B}\phi$ is defined relative to a given belief base: $\mathbf{B}\phi$ holds iff ϕ follows from the belief base under a standard propositional logic entailment relation. The semantics of the $\mathbf{G}$ operator is different from original GOAL, where the goal base is a set of propositional formulas. Here, we use the LTL entailment relation to generalize the operator to be able to express arbitrary types of goals. $\neg\mathbf{B}(at(x)) \wedge \mathbf{G}(\Diamond at(x))$ is a simple example of a mental state condition that expresses that the agent does not belief it is at location x although it wants to be. Such a condition can be used to determine whether to goto a location x (see below).

Before we can move on to defining how the execution of a basic action changes the agent's mental state, we need to explain how the goal base is updated. In the original GOAL language, achievement goals that are believed to be achieved after the execution of a basic action are removed from the goal base. Checking whether an achievement goal is achieved is simple if these are represented as propositional formulas: an achievement goal ϕ is achieved in a mental state if it follows from the belief base in that mental state. In temporalized GOAL, goals are temporal formulas. In order to be able to evaluate the achievement of temporal formulas in a mental state, we use a technique from [1] for "progressing" LTL formulas.

Progression of an LTL formula is a transformation of this formula, which should be performed at each execution step. The idea is that the transformation yields a new formula in which those "parts" of the formula which have already been achieved are set to $\top$, leaving an LTL formula which expresses what still has to be satisfied.

For example, if the agent has a goal $\Diamond\phi$ ($\phi \in \mathcal{L}_0$) in a particular mental state and ϕ is achieved in that mental state, i.e., follows from the belief base, then the progression of this formula is "$\top$", as the agent has produced an execution trace on which the formula holds. If ϕ does not hold, then the progression is the formula "$\Diamond\phi$" itself, since it still needs to be satisfied. If a formula has progressed to a formula equivalent to $\top$, it means the agent has produced an execution trace on which the formula holds. Note that formulas of the form $\Box\chi$ can never progress to $\top$, since it needs to be checked continuously whether χ holds.

The progression operator can be defined inductively for general LTL formulas, as specified in the next definition. We adapt the definition of [1] slightly, as we want a formula which is reached in a mental state to be true already in that state, rather than one mental state later. For this, we define the progression of $\bigcirc\chi'$ as $\mathit{Progress}(\chi')$, rather than as χ'.

Definition 4 *(progression of LTL formulas)*
Let Σ be a belief base, let $\chi \in \mathcal{L}_{LTL}$ be an LTL formula, and let $\phi \in \mathcal{L}_0$. The progression of χ in Σ, $\mathit{Progress}(\chi, \Sigma)$ is then defined as follows.

form of χ is	$\mathit{Progress}(\chi, \Sigma) =$
$\top$	$\top$
ϕ	$\top$ if $\Sigma \models \phi, \bot$ otherwise
$\neg\chi'$	$\neg\mathit{Progress}(\chi', \Sigma)$
$\chi_1 \wedge \chi_2$	$\mathit{Progress}(\chi_1, \Sigma) \wedge \mathit{Progress}(\chi_2, \Sigma)$
$\Diamond\chi'$	$\mathit{Progress}(\chi', \Sigma) \vee \chi$
$\bigcirc\chi'$	$\mathit{Progress}(\chi')$
$\chi_1 U \chi_2$	$\mathit{Progress}(\chi_2, \Sigma) \vee (\mathit{Progress}(\chi_1, \Sigma) \wedge \chi)$
$\Box\chi'$	$\mathit{Progress}(\chi', \Sigma) \wedge \chi$

We lift the progression function of Definition 4 to sets of LTL formulas $\Gamma \subseteq \mathcal{L}_{LTL}$ as follows: $\mathit{Progress}(\Gamma, \Sigma) = \bigcup_{\chi\in\Gamma} \mathit{Progress}(\chi, \Sigma)$.

The next definition specifies how the execution of a basic action changes an agent's mental state. In the formal definition of GOAL, we use a *transition*

function $\mathcal{T}$ to model the effects of basic actions for technical convenience, rather than a specification of pre- and postconditions. The function $\mathcal{T}$ maps a basic action $\mathtt{a}$ and a belief base Σ to an updated belief base $\mathcal{T}(\mathtt{a}, \Sigma) = \Sigma'$. The transition function is undefined if an action is not enabled in a mental state. For example, the transition function may specify that an atom $at(A)$ is added to the belief base, if a basic action $goto(A)$ is executed, while the action is undefined if the agent is already at location A. The GOAL language also includes special actions for adding and removing goals from the goal base, but we do not discuss these actions here (see e.g. [7]). The change of the goal base is defined by means of the progression operator. At each step, all goals of the goal base are progressed.

Definition 5 *(Mental State Transformer $\mathcal{M}$)*
Let $\mathtt{a}$ be a basic action, $\phi \in \mathcal{L}_0$ and $\mathcal{T}$ be a transition function for basic actions. Then the *mental state transformer function* $\mathcal{M}$ is defined as a mapping from actions and mental states to updated mental states as follows:

$$\mathcal{M}(\mathtt{a}, \langle \Sigma, \Gamma \rangle) = \begin{cases} \langle \Sigma', \mathit{Progress}(\Gamma, \Sigma') \rangle & \text{if } \mathcal{T}(\mathtt{a}, \Sigma) = \Sigma' \\ \text{undefined} & \text{otherwise} \end{cases}$$

As is noted in [1], the progression operator has to a certain extent the ability to model check formulas used here to express maintenance goals such as $\Box\chi$. In particular it is able to detect that a finite prefix of an execution path falsifies such goals. The progression operator however is not complete and will not always be able to detect for a particular goal that it can never be satisfied on extensions of all finite prefixes of all execution paths generated by the agent program. Similarly, it cannot be detected that an achievement goal $\Diamond\chi$ might never be achieved on an extension of a course of actions taken so far, nor can we use the operator to detect that a formula is unsatisfiable. The advantage, however, of giving up this component of completeness is computational efficiency; the progression of a formula can be computed in time linear in the size of the formula [1]. In addition, in the context of agent programming the expressivity gained by allowing temporal formulas arguably also provides for a more natural means of designing declarative agent programs. We see the investigation of the properties of the progression operator as an important issue for future research.

The specification of when a basic action may be executed, is done by means of action rules. An action rule c has the form **if** ψ **then** $\mathtt{a}$, with $\mathtt{a}$ a basic action. This action rule specifies that $\mathtt{a}$ may be performed if the mental state condition ψ holds and the transition function is defined for $\mathtt{a}$. In that case we say that action c is *enabled*. As an example, we can now formally write the action rule to goto a location as: **if** $\neg\mathbf{B}(at(x)) \wedge \mathbf{G}(\Diamond at(x))$ **then** $goto(x)$. Given that the agent believes it is still at its base location $at(Base)$ which implies $\neg at(A)$ and has a goal $\Diamond at(A)$ the agent can derive from this rule that $goto(A)$ is an action it might perform (is enabled). During execution, a GOAL agent selects non-deterministically any of its enabled actions. This is expressed in the following transition rule, describing how an agent gets from one mental state to another.

Definition 6 *(Action Semantics)*
Let m be a mental state, and $c =$ **if** ψ **then** $\mathtt{a}$ be an action. The transition

relation $\xrightarrow{c}$ is the smallest relation induced by the following transition rule.

$$\frac{m \models \psi \quad \mathcal{M}(\mathsf{a}, m) \text{ is defined}}{m \xrightarrow{c} \mathcal{M}(\mathsf{a}, m)}$$

The execution of a GOAL agent results in a *computation trace*. We define a trace as a sequence of mental states, such that each mental state can be obtained from the previous by applying the transition rule of Definition 6. As GOAL agents are non-deterministic, the semantics of a GOAL agent is defined as the *set* of possible computations of the GOAL agent, where all computations start in the initial mental state of the agent.

Definition 7 *(Agent Computation)* A computation trace, typically denoted by t, is an infinite sequence of mental states $m_0, m_1, m_2, \ldots$ such that for each i there is an action c_i and $m_i \xrightarrow{c_i} m_{i+1}$ can be derived using the transition rule of Definition 6, or $m_i \not\xrightarrow{c_i}$ and for all $j > i$, $m_j = m_i$. The meaning $R_{\mathcal{A}}(m_0)$ of a GOAL agent named $\mathcal{A}$ with initial mental state m_0 is the set of all computations starting in that state.

Observe that a computation is infinite by definition, even if the agent is not able to perform any action anymore from some point in time on. Also note that the concept of a computation trace is a general notion in program semantics that is not particular to GOAL. The notion of a computation trace can be defined for any agent programming language that is provided with a well-defined operational semantics. The semantics $R_{\mathcal{A}}(m_0)$ thus consists of all traces that may be generated by the agent program. These traces form the first layer of the RASA architecture.

4 Evaluating Temporal Formulas on Prefixes of Traces

In Sections 5 and 6, we will show how to define the second layer (hard constraints) and third layer (soft constraints) on top of the first layer as defined in the previous section. Both hard constraints and soft constraints will be represented using LTL. In standard LTL, formulas are evaluated on *infinite* traces. As explained in Section 2.2, however, in our setting an agent can lookahead a *finite* number of steps and we want to evaluate LTL formulas on such finite traces. We thus need to define the semantics of LTL formulas on such finite traces. This introduces several issues, one of which is the definition of the semantics of the next operator $\bigcirc$. It is not immediately clear what the semantics of a formula $\bigcirc\phi$ should be if it is evaluated in the last state of a finite trace.

For explaining our approach, it is important to realize that the finite trace on which we evaluate LTL formulas is only a *prefix* of a trace which will be continued beyond the lookahead horizon. Intuitively, the truth of a formula $\bigcirc\phi$ evaluated in the last state of such a finite prefix cannot be established. Its truth depends on how the trace continues beyond the finite prefix under consideration. A formula $\Diamond\phi$, on the other hand, clearly holds on a finite prefix if ϕ holds on

some state of this prefix. Similarly, a formula $\Box\phi$ is clearly false on a finite prefix if $\neg\phi$ holds in some state of this prefix.

From these examples, we can see that the truth of an LTL formula evaluated on a finite prefix of a trace depends on how this trace progresses beyond the finite prefix. If a formula is false on any progression, it is also false on the finite prefix. Similarly, if it is true on any progression, it is true on the finite prefix. In all other cases, we cannot determine the truth of the formula. This intuition is reflected by a 3-valued semantics of LTL as presented in [3] in the context of monitoring. In the 3-valued semantics, the truth value of an LTL formula evaluated on a finite trace is $\top$ (true), $\bot$ (false), or "?" (unknown). We use (a slightly adapted version of) the definition of [3] for evaluating LTL formulas on finite prefixes. We define the semantics of LTL formulas over traces of mental states in terms of the semantics over belief bases as follows, where t^b is derived from t by keeping only the belief bases of each mental state: $t, i \models_{LTL} \chi \Leftrightarrow t^b, i \models_{LTL} \chi$.

Definition 8 *(3-valued LTL semantics)*
Let M be a set of mental states. Finite traces over M are elements of M^* and infinite traces are elements of M^ω. Both are typically denoted by t. Let $t, t' \in M^* \cup M^\omega$ be (finite or infinite) traces. Concatenation of traces t, t' is defined as usual and simply denoted as tt'; we stipulate that if $t \in M^\omega$, then $tt' = t$. The truth value of an LTL_3 formula φ with respect to a trace $t \in M^* \cup M^\omega$, denoted by $[t \models \chi]$, is an element of $\{\top, \bot, ?\}$ and defined as follows.

$$[t \models \chi] = \begin{cases} \top \text{ if } \forall t' \in M^\omega : tt' \models_{LTL} \chi, \\ \bot \text{ if } \forall t' \in M^\omega : tt' \not\models_{LTL} \chi, \\ ? \text{ otherwise.} \end{cases}$$

Corollary 1. *If* $t = \epsilon$, $[t \models \varphi] = ?$ *if* $\varphi \not\equiv \bot$ *and* $\varphi \not\equiv \top$.

The only difference between our definition and the one of [3] is that in our definition t can also be infinite. This allows us to investigate our semantics also for an infinite lookahead horizon in a uniform way. In [3], a technique based on the construction of a finite state machine from an LTL formula and a finite trace is presented for determining the truth value of an LTL_3 formula at runtime in implemented systems, which provides a basis for implementing Layers 2 and 3 of our architecture explained in the next sections.

The need for a 3-valued semantics in our approach highlights an important difference with the use of LTL in planning approaches such as [10, 15]. In a planning context, the plans under consideration are always complete. That is, it is not taken into account that these plans might progress beyond their final state. This means that, e.g., a formula $\Diamond\phi$ in those approaches is considered to be false if ϕ does not hold in some state resulting from execution of the plan, and true otherwise. In our semantics, on the other hand, the truth value of the formula is unknown if ϕ does not hold on some state of the finite prefix.

5 RASA Layer 2: Goals as Hard Constraints

Goals of an agent are temporal formulas. The semantics of agent programs provided in Section 3 accounts for the role such goals have in selecting actions using action selection rules. Action selection rules allow an agent to derive actions from its beliefs and goals in a *reactive* manner, but we argue that this layer in the architecture does not yet account for the full role that such goals can have in the action selection mechanism of a rational agent.

If an agent has the ability to *lookahead* a (finite) number of steps, it can also use its goals to avoid selecting those actions that prevent the realization of (some of) the agent's goals. In this section we define the second layer of RASA which accounts for this role of goals. Goals thus viewed introduce additional constraints on action selection to the effect of excluding those actions of which the agent foresees that they will not satisfy its goals. Here we consider such constraints to be *hard* constraints, meaning that any foreseen violation of goals by performing an action will force a rational agent to choose an alternative action (if possible) or become inactive (in line with previous work, cf. [13]). That is, actions will only be selected if for all that is known the action may still eventually allow satisfaction of the goals of the agent.

In this view of goals as hard constraints, achievement goals of the form $\Diamond\phi$ per se do not add any additional constraints on action selection since an agent with a finite lookahead horizon will not be able to conclude that an action will prohibit realizing such a goal. Achievement goals thus may be said to have no "selective force" beyond the rule-based mechanism of the action selection architecture. The role of a goal such as being at location A, $\Diamond at(A)$, thus is mainly to guide this selection process and, in order to avoid wasting resources, to remove such goals as reasons for action when they have been achieved (see the *Progress* operator of Section 3.2). Other types of goals such as maintenance goals do have a selective force in this sense; for example, if $\Box\phi$ is a goal of the agent, an agent can conclude that it is no longer possible to satisfy this goal if ϕ does not hold in some state within the lookahead horizon. An agent thus can use such goals to filter certain options for action generated by the rule-based layer one.[8] For example, the maintenance goal of having a minimum level of fuel in the tank of an agent will prevent selection of an action $goto(A)$ in case this would drop fuel levels below this minimum, assuming that going somewhere consumes fuel.

Of course, the agent needs to consider possible future effects of a course of action allowed by the agent program to check such constraints, which is why a lookahead horizon needs to be defined to do so. The lookahead horizon is an integer specifying the length of a prefix of a possible trace of the agent program. Here the 3-valued semantics for LTL discussed in the previous section is particularly useful since it allows us to evaluate goals on finite prefixes of traces.

[8] One could argue that layer one should also make sure that such maintenance goals are not violated. However, trying to account for such general constraints on agent behaviour in the rules in an ad hoc manner will often lead to less understandable programs, which is why we argue for the separation of concerns provided by the proposed RASA.

The fact that an achievement goal $\Diamond\phi$ does not have selective force, is reflected by the fact that its truth value in the 3-valued LTL semantics will be "?" (the "unknown" value).

The filtering of action options by means of verifying whether finite prefixes of a trace of an agent program satisfy the goals of an agent changes the meaning of that agent program. It does not change the fact that an agent can continue performing actions infinitely. We next show how the semantics of an agent program can be defined to capture the effects of the second layer of the RASA formally in the context of GOAL. A *prefix* of a computation t is an initial finite sequence of t or t itself. A prefix of length n of a computation t is denoted by $t^{\langle n \rangle}$ with $n \in \mathbb{N} \cup \{\infty\}$, where $t^{\langle \infty \rangle}$ is defined as t. $\mathbb{N}$ is the set of natural numbers including 0, and ∞ is the first infinite ordinal. The lookahead horizon, i.e., the number of execution steps that an agent can lookahead, is denoted by h. The idea is that we take the set of possible execution traces $R_{\mathcal{A}}$ that may be selected according to the first layer, and filter out those traces that do not satisfy one or more of the agent's goals, when looking ahead h steps from some state on the trace.

We define a filter function $\sigma^h_{\mathcal{A}}$ inductively on the set of traces $R_{\mathcal{A}}$ and on the time point i on such a trace. An agent will use its goals to restrict selection of actions from the start, i.e. time 0, which explains why the base case of the inductive definition starts at -1. The base case defines the starting point of traces $R_{\mathcal{A}}$ that need to be filtered. For technical reasons the filter function is defined as two different components, and in addition to σ we define a function ς. The idea is that the filter function $\sigma^h(i)$ returns all (infinite) traces that do not violate the goals of the agent on a finite prefix of length h starting from state i, while the function $\varsigma^h(i)$ returns all (finite) prefixes of traces that do not violate the goals of an agent given a lookahead capability of h until the end of that prefix but that do violate some goal in any possible next state.

Definition 9 *(Goal Filter Functions σ and ς)*
Let $\mathcal{A}$ be some agent with meaning $R_{\mathcal{A}}$ and let $h \in \mathbb{N}$ be a horizon. Let $t[i]$ be the tail of trace t starting in the i-th state of t. Then the *goal filter functions* $\sigma^h_{\mathcal{A}}$ and $\varsigma^h_{\mathcal{A}}$ are defined by simultaneous induction as follows:

$$\begin{aligned} \sigma^h_A(-1) &= R_{\mathcal{A}}, \\ \sigma^h_A(i) &= \{t \in \sigma^h_A(i-1) \mid \forall \chi \in \Gamma^t_i : [t[i]^{\langle h \rangle} \models_{LTL} \chi] \neq \bot\} \end{aligned}$$

$$\begin{aligned} \varsigma^h_{\mathcal{A}}(-1) &= \emptyset, \\ \varsigma^h_{\mathcal{A}}(i) &= \varsigma^h_{\mathcal{A}}(i-1) \cup \{t^{\langle i \rangle} \mid t \in \sigma^h_A(i-1), \exists \chi \in \Gamma^t_i : [t[i]^{\langle h \rangle} \models \chi] = \bot\}, \text{ for } i \geq 0 \end{aligned}$$

To avoid complicating the definition of the function ς it is defined slightly too general and includes prefixes that do have continuations that satisfy all of an agent's goals. In order to eliminate these prefixes and keep only maximal prefixes of traces that cannot be extended given the agent's goals we additionally introduce a function $maxPrefix$. Let $t \sqsubset t'$ denote that t is a strict prefix of t' and T a set of (in)finite traces. Then $t \in maxPrefix(T)$ iff there is no $t' \in T$ such that $t \sqsubset t'$. Using the definitions of the goal filter function(s), we can now

provide a simple definition of the semantics of traces induced by the second layer in the action selection architecture. The meaning of an agent at this layer is denoted by $H_{\mathcal{A}}$ and defined as the maximal elements of the limit of the filter functions.

Definition 10 *(Semantics of GOAL Agent with Goals as Hard Constraints)* The meaning of a GOAL agent H_A that applies hard constraints in addition to its rule-based action selection is defined by:

$$H_{\mathcal{A}} = maxPrefix\big(\bigcap_{i=-1}^{\infty} \sigma_{\mathcal{A}}^{h}(i) \cup \bigcup_{i=-1}^{\infty} \varsigma_{\mathcal{A}}^{h}(i)\big)$$

It should be clear from the definition of the meaning of a GOAL agent that the semantics introduced clearly distinguishes between the different layers of RASA. Moreover, given the computational properties of LTL_3 the semantics of the second layer can be realized computationally if the initial prefixes of length h of the traces induced by the first layer can be efficiently generated. In the next section we introduce the third layer to complete our formal picture of the informal RASA discussed in Section 2.

6 RASA Layer 3: Preferences as Soft Constraints

In the third layer of RASA an agent aims to select those traces that maximize satisfaction of its preferences. Whereas the second layer *eliminates* any action options that would lead to violation of a goal, the third layer only eliminates action options if *more preferred alternatives* are available. An agent thus might prefer to have 60 units of fuel minimally in its tank but in case there are no actions enabled (as determined by the agent program) that would allow the agent to satisfy this preference it would still select one of these actions; contrast this with a maintenance goal which would prevent the agent from doing anything at all in this case (in case there would not be an option to refuel). Similarly to goals, preferences are expressed using LTL. For example, preferring to go first to location A before going to location B can be expressed by $\neg(\neg at(A) U at(B)) \wedge \Diamond at(B)$. Since LTL formulas express properties of traces, this allows an agent to express that it prefers one way of achieving a goal, i.e., one particular trace, over another, if multiple courses of action for realizing the goal are available. Such preferences are represented by a so-called *preference structure*. A preference structure consists of a sequence of LTL formulas.

Definition 11 *(Preference Structure)*
A *preference structure* Ψ is a sequence $(\chi_1, \ldots, \chi_n)$ of LTL formulas.

This preference structure expresses that traces on which some χ_i is satisfied are preferred over traces on which χ_i is not satisfied, and the satisfaction of χ_i is preferred over the satisfaction of χ_j for $j > i$. That is, if χ_1 is satisfied on a trace t but not on another trace t', t is preferred over t'. If both t and t' do not

satisfy χ_1, but t satisfies χ_2 and t' does not, t is again preferred over t', etc. This interpretation of the preference structure thus induces a *lexicographic* ordering on traces.

This ordering can formally be defined as follows. We use Ψ_t to denote the sequence $(b_1, \ldots, b_n)$, where $b_i = [t \models \chi_i]$ for $1 \leq i \leq n$, i.e., $b_i \in \{\top, \bot, ?\}$; we use Ψ_t^i to denote b_i. We now use the ordering $\bot < ? < \top$ to define a lexicographic preference ordering on traces on the basis of a preferences structure. That is, if for a χ_i of the preference structure we have $[t \models \chi_i] = \top$, this is better than when $[t \models \chi_i] = ?$, which is again better than when $[t \models \chi_i] = \bot$.

Definition 12 *(Lexicographic Preference Ordering)*
Let $\Psi = (\chi_1, \ldots, \chi_n)$ be a preference structure and $t, t' \in M^* \cup M^\omega$ be finite or infinite traces. Then we say that trace t is (lexicographically) preferred over t' with respect to Ψ, written $t \prec_\psi t'$, iff:

$$\exists 1 \leq j \leq n : \forall 1 \leq i < m : (l_t^i = l_{t'}^i \text{ and } l_t^j < l_{t'}^j)$$

We also write $t \preccurlyeq_\psi t'$ if $t \prec_\psi t'$ or $\psi_t = \psi_{t'}$.

The main advantage of using a lexicographic preference order $\prec_\psi$ is that such a preference order on traces is a total preorder, which means that any two traces are either equally good with respect to the preference order, or one is better than the other. Other kinds of preference orderings such as those discussed in [6] may be considered, but investigating this is left for future research.

As an example, given that the preference to go first to location A before goint to B is ranked higher than the preference to keep a minimum fuel level of 60, even if refuelling would be possible while going to B and not while going to A, the agent would choose to first go to A using the lexicographic preference ordering.

Our use of a lexicographic preference order is inspired by the work of [10, 15]. There are, however, many differences with [10, 15] and our approach. Instead of integrating preferences into the situation calculus as in [10] we propose a uniform framework for goals and preferences that is integrated into a programming framework for rational agents, and instead of compiling preferences into certain programs we have taken a more direct approach by defining a layered action selection architecture that is made precise by means of a formal semantics. Finally, we use three-valued LTL to evaluate goals and preferences on (prefixes of) traces.

Preferences are progressed or updated through time, just like goals. The idea is that a preference such as $p \wedge \bigcirc q \wedge \Diamond r$ has been satisfied when now p holds, in the next state q and possibly sometime thereafter r holds. Such a preference cannot be satisfied anymore when p is not true currently, and to express this we use the *Progression* operator of Section 3.2 to keep track of such facts. Since preferences are progressed during execution of the agent, we extend mental states of an agent with the agent's preference structure.

Definition 13 *(Mental State with Preferences)*
Let $m = \langle \Sigma, \Gamma \rangle$ be a mental state, and Ψ a preference structure. A mental state with preferences then simply is the tuple $\langle \Sigma, \Gamma, \Psi \rangle$.

Definition 14 *(Progression of Preference structure)*
The progression of a preference structure $\Psi = (\chi_1, \ldots, \chi_n)$ from a mental state $\langle \Sigma, \Gamma, \Psi \rangle$ to a new state $\langle \Sigma', \Gamma', \mathit{Progress}(\Psi, \Sigma') \rangle$ is simply defined as the progression of each of the individual preference in the structure, i.e.

$$\mathit{Progress}(\Psi, \Sigma') = (\mathit{Progress}(\chi_1, \Sigma'), \ldots, \mathit{Progress}(\chi_n, \Sigma'))$$

The semantics of action execution of Definition 6 is changed accordingly as follows, where $m = \langle \Sigma, \Gamma, \Psi \rangle$ and $c = \textbf{if } \psi \textbf{ then } \mathtt{a}$ is an action:

$$\frac{m \models \psi \quad \mathcal{M}(\mathtt{a}, m) = \langle \Sigma', \Gamma' \rangle}{\langle \Sigma, \Gamma, \Psi \rangle \xrightarrow{c} \langle \Sigma', \Gamma', \mathit{Progress}(\Psi, \Sigma') \rangle}$$

The main difference between goals and preferences in our approach is that goals are used as hard constraints to guide action selection whereas preferences are used as soft constraints. That is, an agent would like to satisfy all of its preferences but if this is not possible it will simply choose to satisfy those that are most preferred, if any can be satisfied at all. The preference order introduced above can be used for this purpose. In the third layer of RASA those traces are selected from the remaining ones (those that survived filtering by layer 2) that are maximal elements in this order. As with the second layer, the third layer of RASA modifies the meaning of an agent program. In order to specify the effects of this layer on the semantics of agent programs, we introduce some notation again. We use $\mathit{max}(T, \prec_\psi)$ to denote the maximal elements of a set of traces T under the lexicographic preference order $\prec_\psi$, induced by the preference structure Ψ. The preference filter function of layer 3 in our action selection architecture can then be defined similarly to that of the goal filter function.

Definition 15 *(Preference Filter Function ψ)*
Let $\mathcal{A}$ be some agent using layer 2 goal filtering with meaning $H_\mathcal{A}$ and let $h \in \mathbb{N}$ be a horizon. Then the *preference function* $\psi^h_\mathcal{A}$ is defined by induction as follows:

$$\begin{aligned} \psi^h_A(-1) &= H_\mathcal{A}, \\ \psi^h_A(i) &= \{t \mid t^{\langle i+h \rangle} \in \mathit{max}(\{t[i]^{\langle h \rangle} \mid t \in \psi^h_A(i-1)\}, \prec_\psi)\} \end{aligned}$$

Similar to Section 5, the semantics of an agent that uses all three layers of the action selection architecture is defined as the limit of the preference filter function over the traces obtained from layer 2. The definition is somewhat simpler since an agent can always continue on a given trace since it only needs to select a maximum continuation of its past action performance but does not have to stop acting altogether due to a potential violation of a hard constraint. This is one of the main differences between layer 2 and 3.

Definition 16 *(Semantics of GOAL with Preferences)*
The meaning of a GOAL agent P_A that applies soft constraints with horizon h in addition to the action selection mechanisms of layer 1 and 2 is defined by:

$$P_{\mathcal{A}} = \bigcap_{i=-1}^{\infty} \psi^h_{\mathcal{A}}(i)$$

Definition 16 completes the specification of each of the three layers of the rational action selection architecture. The informal discussion of the architecture in which we distinguished three layers of action selection is mirrored in respectively Definitions 7, 10, and 16. The formal approach has shown that it is feasible to define a transparent and rich RASA into an agent programming language, that integrates beliefs, goals and preferences as tools for choosing the right action. In particular, the approach offers a uniform framework based on temporal logic to express goals and preferences.

7 Conclusion and Related Work

In this paper, we have proposed a layered rational action selection architecture which specifies a rational action selection mechanism in which hard and soft constraints are integrated. We have shown how the proposed architecture can be formalized in the context of the GOAL agent programming language and how it modifies the semantics of agent programs. In order to obtain a uniform framework that naturally allows for the integration of hard and soft constraints including such concepts as achievement goals, maintenance goals, as well as (temporally extended) preferences, we have used linear temporal logic for the representation of goals and preferences.

The way in which we use temporal logic is inspired by work in planning where temporal logic is used to "guide" planners through the search space [1] and to select preferred plans [10, 2, 15]. One of the differences is that the RASA is defined for agent programming languages and, in contrast with planners that would compare *complete plans* against the available constraints, these constraints are taken into account continuously *during* execution of an agent program. Technically, this has resulted in the use of 3-valued LTL for the realization of RASA in GOAL.

The practical realizability of our approach is partly facilitated by the use of techniques, in particular the progression algorithm of [1] and the implementation of 3-valued LTL of [3], that have been shown to be implementable. The investigation of restrictions of the LTL language for representing goals to make checking of entailment of a goal from the goal base feasible, is left for future research.

Acknowledgements

We would like to thank Moritz Hammer for pointing us to [3].

References

1. F. Bacchus and F. Kabanza. Using temporal logics to express search control knowledge for planning. *Artificial Intelligence*, 166, 2000.
2. Jorge A. Baier, Fahiem Bacchus, and Sheila A. McIlraith. A heuristic search approach to planning with temporally extended preferences. In *IJCAI*, pages 1808–1815, 2007.
3. Andreas Bauer, Martin Leucker, and Christian Schallhart. Monitoring of real-time properties. In *Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *LNCS*, pages 260–272. Springer, 2006.
4. Meghyn Bienvenu, Christian Fritz, and Sheila A. McIlraith. Planning with qualitative temporal preferences. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR'06)*, pages 134–144, 2006.
5. Rafael H. Bordini, Mehdi Dastani, Jürgen Dix, and Amal El Fallah Seghrouchni. *Multi-Agent Programming: Languages, Platforms and Applications*. Springer, Berlin, 2005.
6. Gerhard Brewka. A rank based description language for qualitative preferences. In *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI'04)*, pages 303–307, 2004.
7. Frank de Boer, Koen Hindriks, Wiebe van der Hoek, and John-Jules Meyer. A Verification Framework for Agent Programming with Declarative Goals. *Journal of Applied Logic*, 5:277–302, 2007.
8. E.A. Emerson. Temporal and modal logic. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, volume B: Formal Models and Semantics, pages 996–1072. Elsevier, Amsterdam, 1990.
9. R.E. Fikes and N.J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.
10. Christian Fritz and Sheila A. McIlraith. Decision-theoretic golog with qualitative preferences. In *KR*, pages 153–163, 2006.
11. M. Ghallab, D. Nau, and P. Traverso. *Automated Planning: Theory and Practice*. Morgan Kaufmann, 2004.
12. Koen Hindriks. Modules as policy-based intentions: Modular agent programming in goal. In *Proceedings of the International Workshop on Programming Multi-Agent Systems (PROMAS'07)*, number 4908 in LNAI. Springer, 2008.
13. Koen Hindriks and Birna van Riemsdijk. Satisfying maintenance goals. In *Proceedings of the International Workshop on Declarative Agent Languages and Theory (DALT'07)*, number 4897 in LNAI. Springer, 2008.
14. Koen V. Hindriks, Frank S. de Boer, Wiebe van der Hoek, and John-Jules Ch. Meyer. Agent Programming with Declarative Goals. In *Proceedings of ATAL00*, volume 1986 of *LNCS*, pages 228–243, 2000.
15. Tran Cao Son and Enrico Pontelli. Planning with preferences using logic programming. *Theory and Practice of Logic Programming*, 6(5):559–607, 2006.