Satisfying Maintenance Goals

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

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

Abstract. A rational agent derives its choice of action from its beliefs and goals. Goals can be distinguished into achievement goals and maintenance goals. The aim of this paper is to define a mechanism which ensures the satisfaction of maintenance goals. We argue that such a mechanism requires the agent to look ahead, in order to make sure that the execution of actions does not lead to a violation of a maintenance goal. That is, maintenance goals may constrain the agent in choosing its actions. We propose a formal semantics of maintenance goals based on the notion of lookahead, and analyze the semantics by proving some properties. Additionally, we discuss the issue of achievement goal revision, in case the maintenance goals are so restrictive that all courses of action for satisfying achievement goals will lead to a violation of maintenance goals.

1 Introduction

The research presented in this paper concerns the role of *maintenance goals* in the selection of actions by a rational agent. A rational agent aims at satisfying its goals, which may include both achievement goals as well as maintenance goals. Achievement goals define states that are to be achieved, whereas maintenance goals define states that must remain true.

The distinction between achievement and maintenance goals is common in the literature about rational agents. However, whereas various proposals for computational semantics and programming frameworks that include achievement goals are available [3, 5, 11, 14, 17, 18], maintenance goals have received less attention [3, 4, 6]. In this paper we investigate a semantics for maintenance goals. Our aim is to define a mechanism which ensures the satisfaction of maintenance goals that can be integrated into various agent programming languages.

Achievement goals in agent programming frameworks are typically used in combination with rules that express which action or plan an agent may execute in certain circumstances in order to achieve a particular achievement goal. In such a setting, achievement goals thus trigger the execution of a course of action. A maintenance goal can have a similar role in agent programming frameworks, in the sense that it can trigger an agent to perform actions in order to ensure that a maintenance goal is not violated, or to take action to reestablish the maintenance goal if it is violated.

Implementing maintenance goals using conditions to trigger the execution of actions, however, is not sufficient to guarantee that maintenance goals are not violated. In order to prevent the violation of a maintenance goal, an agent may sometimes have to *refrain* from performing an action that the agent would otherwise have selected, e.g., to satisfy one of its achievement goals [6]. A comprehensive framework for maintenance goals should thus not only incorporate an action selection mechanism based on triggering conditions, but should also take into account the *constraining* role of maintenance goals. As we will show, a selection mechanism that is based on this constraining role can also be used to actively ensure that a maintenance goal is not violated.

We argue that taking into account the constraining role of maintenance goals requires some kind of *lookahead* mechanism, which allows the agent to determine whether certain actions or plans it would like to perform might lead to the violation of one of its maintenance goals. The main aim of this paper is to investigate how the semantics of maintenance goals can be formally defined through such a lookahead mechanism. We analyze the semantics formally by proving several properties. It is important to note that one advantage of giving a formal semantics, is the fact that one can formally prove certain properties. This is important in order to get a clear understanding of the phenomenon under investigation. Besides providing a formal semantics and analysis of maintenance goals, we discuss the issue of achievement goal revision, in case the maintenance goals are so restrictive that all plans for satisfying achievement goals will lead to a violation of maintenance goals.

The paper is organized as follows. In Section 2, a motivating example is introduced to illustrate the main ideas throughout the paper. Our investigations are carried out in the context of the agent programming language GOAL [5], which is briefly introduced in Section 3. The results presented, however, are general and can be integrated into any agent framework. Section 4 formally defines a lookahead mechanism that ensures selected actions do not violate the agent's maintenance goals. The look-ahead mechanism introduced, however, may overconstrain an agent's decision procedure. In Section 5 this problem is discussed and a revision procedure is suggested to resolve it. Section 6 concludes the paper, outlines some directions for future work, and discusses related work.

2 Motivating Example: A Carrier Agent

In this section, a simple scenario involving a carrier agent is presented in order to illustrate the role of maintenance goals in action selection and the reasoning we believe is involved in avoiding violation of maintenance goals.

2.1 The Basic Scenario

The setting is as follows. Consider an agent who wants to bring parcels from some location A to a location B, using its truck. The distance between A and B is too large to make it without refueling, and so, in order not to end up without gas, the agent needs to stop every once in a while to refuel. The fact that the agent does not want to end up without gas, can be modeled as a maintenance goal.[3] This maintenance goal *constrains* the actions of the agent, as it is not supposed to drive on in order to fulfil its goal of delivering the parcels, if driving on would cause it to run out of gas.

The action of driving is an action that the agent can take in order to fulfil its achievement goal of delivering the parcels. Other actions that the agent has at its disposal, may be used to *actively* ensure that the agent's maintenance goals are not violated. In the example scenario, the action of refueling can be viewed as an action of this kind (although, in this example, not violating the maintenance goal is also instrumental for achieving the achievement goal). Maintenance goals thus on the one hand constrain the agent's actions, but may also induce the agent to take preventive actions to make sure maintenance goals are not violated.

An essential reasoning mechanism in order to ensure that the agent does not take actions that would violate the agent's maintenance goals is a *lookahead* mechanism. In the example scenario, the agent should reason about the distance to the next gas station and the amount of fuel it has left, in order to make sure it does not end up without fuel between two gas stations. That is, it should in one way or another, reason about the consequences of possible future sequences of actions in order to be able to choose those actions that will not lead to a violation of maintenance goals at some point in the future.

2.2 Conflicts between Achievement and Maintenance Goals

In this simple scenario so far, there is no conflict between the agent's maintenance goals and achievement goals. It is perfectly possible for the agent to deliver its parcels without running out of gas, as long as it refuels in time. It may, however, sometimes be the case that conflicts between achievement goals and maintenance goals arise, in the sense that in order to achieve an achievement goal, the agent will have to violate a maintenance goal.

In the example scenario, such a conflict may arise if the agent has the additional maintenance goal of making sure that the weight of truck load stays below a certain threshold. Assuming that the total weight of the parcels exceeds this threshold, and assuming that the agent cannot drive back and forth between A and B (e.g., because the agent has loaned the truck and has to return it after arriving at location B), there is a conflict between the achievement goal of bringing all the parcels from A to B, and not overloading the truck.

Such a situation of conflict may result in the agent not doing anything anymore at a certain point. That is, it may be the case that any action the agent is able to do to achieve its achievement goal is not allowed because this would lead to a violation of the agent's maintenance goal, and, moreover, there is no possibility to actively ensure that the maintenance goal is not violated. In general, there are several possibilities of dealing with such a situation.

The first option is not to do anything about it. The intuition here is that the agent should never violate its maintenance goals, i.e., maintenance goals are *hard constraints*, and the agent wants "all or nothing" when it comes to

[3] Other papers [3, 4, 6] have used a similar maintenance goal in some of their examples.

its achievement goals. In the example scenario, it may be the case that it is of utmost importance that the truck is not overloaded, e.g., because the truck has a device with which the weight of the freight is measured, and if the weight exceeds the threshold the truck cannot start. Moreover, it may be the case that bringing the parcels only makes sense if all parcels are brought, e.g., because the parcels contain parts of a closet and there is no use for bringing only part of the closet. Put differently, the utility of delivering only part of the parcels is zero.

A second option is to allow the agent to *violate its maintenance goals*, if this is absolutely necessary in order to achieve an achievement goal. An intuitive implementation of such a mechanism would have to make sure that the agent really only violates maintenance goals if there is no way around it, and if this is necessary, it should try to "minimize" the violation, e.g., by trying to make sure that the maintenance goal is satisfied again as soon as possible after the achievement goal that was the reason to violate the maintenance goal has been satisified. In the example scenario, it may be the case that overloading the truck does not do too much harm, as long as this does not happen too often. It is then important that the truck is unloaded as soon as the destination is reached.

The third option is to *modify the achievement goal*, such that the modified achievement goal does not conflict anymore with the agent's maintenance goals. The idea here is that there might be achievement goals that can be achieved "to a certain degree", i.e., it might be possible to "weaken" the achievement goal, in case it would conflict with a maintenance goal. In the example scenario, the conflict between the achievement goal of getting all parcels at location A, and the maintenance goal of not overloading the truck, could be resolved by modifying the achievement goal such that the agent settles on bringing only *part* of the parcels to location B. The decision of which parcels to leave behind can be based on the weight of the parcels, i.e., the weight of the parcels to be taken along should not exceed the threshold, and on the utility of getting certain parcels at the destination, i.e., some parcels may be more important than others.

Of course, combinations of these possibilities of dealing with conflicts are also possible. Such combinations might define certain maintenance goals as hard constraints and certain achievement goals as "all or nothing" goals, while other maintenance goals and achievement goals may be violated or modified, respectively. In this paper, however, we focus on the third option, i.e., we view maintenance goals as hard constraints, and opt for the modification or weakening of achievement goals in case a conflict with a maintenance goal arises. In domains in which maintenance goals relate, e.g., to the limited availability of resources and time which cannot easily be lifted the third strategy will typically be valid.

3 The GOAL Language

In this section, the GOAL programming language [5, 10] is briefly introduced and a GOAL agent that implements a simplified version of the carrier agent of Section 2.1 is presented. A GOAL agent selects actions on the basis of its beliefs and achievement goals, i.e., maintenance goals were not investigated in the original GOAL language. Whenever goals are mentioned in this section, this should thus be interpreted as meaning achievement goals. The definitions we provide in this section are used to make the notion of an agent computation precise, which we use in Section 4 to define the semantics for maintenance goals.

A GOAL program for the carrier agent is specified in Table 1. The program consists of four sections: (1) a set of initial beliefs, collectively called the (initial) *belief base* of the agent, (2) a set of initial achievement goals, called the (initial) *goal base*, (3) a *program section* which consists of a set of conditional actions, and (4) an *action specification* that consists of a specification of the pre- and post-conditions of *basic actions* of the agent. In the example, variables are used as a means for abbreviation; variables should be thought of as being instantiated with the relevant arguments to yield propositions. The constants used in the example denote locations (`a`, `ab1`, `ab2`, `b`, assumed to be spatially positioned in this order), parcels (`p1`,`p2`) and a truck `truck`. The order of the locations means that if the agent wants to get from `a` to `b`, it first has to pass `ab1`, and then `ab2`. We use the comma to denote conjunction.

```
:beliefs{ loc(p1,a). loc(p2,a). loc(truck,a). loc(gasstation,ab1).
          fuel(2). next(a,ab1). next(ab1,ab2). next(ab2,b). }
:a-goals{ loc(p1,b), loc(p2,b). }
:program{
    if B(loc(truck,X), loc(P,X), X≠Y), G(loc(P,Y)) then load(P).
    if B(loc(truck,a)), ~(B(loc(P,a)), G(loc(P,b))), G(loc(R,b))
        then adopt(loc(truck,b)).
    if G(loc(truck,b)) then move.
    if B(loc(gasstation,X)) then tank.
    if B(loc(truck,X), in(P,truck)), G(loc(P,X)) then unload(P). }
:action-spec{
    move     { :pre{loc(truck,X), next(X,Y), fuel(Z), Z > 0}
               :post{loc(truck,Y), not loc(truck,X), fuel(Z-1), not fuel(Z)} }
    load(P)  { :pre{loc(P,X), loc(truck,X)}     :post{in(P,truck), not loc(P,X)} }
    unload(P){ :pre{in(P,truck), loc(truck,X)}  :post{loc(P,X), not in(P,truck)} }
    tank     { :pre{loc(truck,X), loc(gasstation,X), fuel(Y), Y<3}
               :post{fuel(3), not fuel(Y)}} }
```

Table 1. GOAL Carrier Agent

The belief base, typically denoted by Σ, and the goal base, typically denoted by A, together define the *mental state* of a GOAL agent. Mental states should satisfy a number of rationality constraints, which are introduced next.

Definition 1 *(Mental States)*
Assume a language of propositional logic $\mathcal{L}_0$ with the standard entailment relation $\models$ and typical element ϕ. A mental state of a GOAL agent, typically denoted by s, is a pair $\langle \Sigma, A \rangle$ with $\Sigma, A \subseteq \mathcal{L}_0$ where Σ is the belief base, and A with typical element α 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)	Individual goals are consistent[4]:	for all $\alpha \in A$: $\not\models \neg\alpha$,
(iii)	Goals are not believed to be achieved:	for all $\alpha \in A$: $\Sigma \not\models \alpha$.

In the example carrier agent, the two parcels and the truck are initially believed to be at location `a`, represented by `loc(p1,a)`, `loc(p2,a)`, and `loc(truck,a)`. The agent also believes it has two units of fuel, and that the gas station is at location `ab1`. The initial achievement goal of the agent is to have both parcels at location `b`, represented by `loc(p1,b), loc(p2,b).` Note that the carrier agent satisfies the rationality constraints on mental states.

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 2 *(Mental State Conditions)*
The language $\mathcal{L}_M$ of mental state conditions, typically denoted by ψ, is inductively defined by the two clauses:

- if $\phi \in \mathcal{L}_0$, then $\mathbf{B}\phi, \mathbf{G}\phi \in \mathcal{L}_M$,
- if $\psi_1, \psi_2 \in \mathcal{L}_M$, then $\neg\psi_1, \psi_1 \wedge \psi_2 \in \mathcal{L}_M$.

The truth conditions of mental state conditions ψ, relative to a mental state $s = \langle \Sigma, A \rangle$, are defined by the following four clauses:

$$
\begin{array}{lll}
s \models_m \mathbf{B}\phi & \text{iff} & \Sigma \models \phi, \\
s \models_m \mathbf{G}\phi & \text{iff} & \text{there is } \alpha \in A \text{ such that } \alpha \models \phi \text{ and } \Sigma \not\models \phi, \\
s \models_m \neg\psi & \text{iff} & s \not\models_m \psi, \\
s \models_m \psi_1 \wedge \psi_2 & \text{iff} & s \models_m \psi_1 \text{ and } s \models_m \psi_2.
\end{array}
$$

The semantics of $\mathbf{B}\phi$ defines that this holds iff ϕ follows from the belief base under a standard proposition logic entailment relation. The definition of the semantics of $\mathbf{G}\phi$ is somewhat more involved. It specifies that $\mathbf{G}\phi$ holds, iff ϕ is not already believed by the agent, and there is a formula in the goal base from which ϕ follows. Also multiple goals are not required to be consistent which reflects the fact that each goal may be realized at a different moment in time.

In GOAL, two types of actions are distinguished: basic actions and goal update actions. The execution of basic actions updates and modifies the agent's beliefs, apart from changing the agent's environment. Indirectly, a basic action may also affect the goal base of an agent. That is, in case a goal is believed to be achieved after action execution the goal is dropped by the agent and may be removed from the agent's goal base.

In the example program, the way in which the execution of basic actions changes the beliefs of the agent is specified using pre- and post-conditions. The example agent has four basic actions at its disposal, i.e., the actions `move`, `load(P)`, `unload(P)`, and `tank`. Through the action `move`, it can move one position towards location `b`. Using `unload(P)` and `load(P)`, it can unload and load the parcel `P`, respectively, if the agent is at the same location as the parcel. The action `tank` can be executed if the agent is at location `ab1`, resulting in the amount of fuel becoming `3`.

In the formal definition of GOAL, we use a *transition function* $\mathcal{T}$ to model the effects of basic actions. This function maps a basic action `a` and a belief base Σ to an updated belief base $\mathcal{T}(\mathsf{a}, \Sigma) = \Sigma'$. The transition function is undefined if an action is not enabled in a mental state. In a GOAL agent, the action specification section of that agent specifies this transition function. In the example agent in Table 1 a STRIPS-like notation is used, where positive literals define the add list and negative literals define the delete list (cf. [12]). (Other, extended action formalisms could be used but for the purpose of this paper a more extended formalism is not needed.) GOAL has two built-in goal update actions: the **adopt**(ϕ) action to adopt a goal, and the **drop**(ϕ) to drop goals from the agent's goal base. An **adopt**(ϕ) action has to satisfy the rationality constraints on mental states, i.e. ϕ must be consistent and not believed by the agent. The **drop**(ϕ) action removes all goals from the goal base that imply ϕ.

Definition 3 *(Mental State Transformer $\mathcal{M}$)*
Let 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:

$$\begin{array}{ll}
\mathcal{M}(\mathsf{a}, \langle \Sigma, A \rangle) & = \begin{cases} \langle \Sigma', A \setminus \{\psi \mid \Sigma' \models \psi\}\rangle & \text{if } \mathcal{T}(\mathsf{a}, \Sigma) = \Sigma' \\ \text{undefined} & \text{otherwise} \end{cases} \\
\mathcal{M}(\mathbf{adopt}(\phi), \langle \Sigma, A \rangle) & = \begin{cases} \langle \Sigma, A \cup \{\phi\}\rangle & \text{if } \not\models \neg\phi \text{ and } \Sigma \not\models \phi \\ \text{undefined} & \text{otherwise} \end{cases} \\
\mathcal{M}(\mathbf{drop}(\phi), \langle \Sigma, A \rangle) & = \langle \Sigma, A \setminus \{\psi \in A \mid \psi \models \phi\}\rangle
\end{array}$$

In order to select the appropriate actions to achieve the goal of having the two parcels at location `b`, our example carrier agent has five conditional actions as listed in the program section of Table 1. A conditional action c has the form **if** ψ **then** a, with a either a basic action or a goal update action. This conditional action specifies that a may be performed if the mental state condition ψ and the preconditions of a hold. In that case we say that conditional action c is *enabled*.

During execution, a GOAL agent selects non-deterministically any of its enabled conditional actions. This is expressed in the following transition rule, describing how an agent gets from one mental state to another.

Definition 4 *(Conditional Action Semantics)*
Let s be a mental state, and $c =$ **if** ψ **then** a be a conditional action. The transition relation $\xrightarrow{c}$ is the smallest relation induced by the following transition rule.

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

The execution of a GOAL agent results in a *computation*. We define a computation as a sequence of mental states, such that each mental state can be obtained from the previous by applying the transition rule of Definition 4. 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 5 *(Agent Computation)*
A computation, typically denoted by t, is an infinite sequence of mental states $s_0, s_1, s_2, \ldots$ such that for each i there is an action c_i and $s_i \xrightarrow{c_i} s_{i+1}$ can be derived using the transition rule of Definition 4, or $s_i \overset{c_i}{\not\longrightarrow}$ and for all $j > i$, $s_j = s_i$. The meaning $S_\mathcal{A}$ of a GOAL agent named $\mathcal{A}$ with initial mental state $\langle \Sigma_0, A_0 \rangle$ 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 an agent computation is a general notion in program semantics that is not particular to GOAL. The notion of a computation can be defined for any agent programming language that is provided with a well-defined operational semantics. For such languages, it is possible to transfer the analysis of maintenance goals in this paper that is based on the notion of a computation and to incorporate the proposed maintenance goal semantics.

Our example carrier agent may execute the following computations. In the initial mental state, the conditional action for loading a parcel is executed, and the agent non-deterministically picks up one of the parcels, followed by another execution of this conditional action to load the other parcel. Consecutively, the only enabled conditional action is the one for adopting the goal `loc(truck,b)`, by which the example agent adopts the goal to be at location `b`. As the agent now has the goal to be at location `b` it will execute the enabled action `move`. After executing the `move` action, the agent is at location `ab1`, and has one unit of fuel left.

In this situation, there are two possibilities. The agent can execute another `move` action, after which the agent will be at location `ab2` without any fuel. The other option is that the agent executes the `tank` action, after which the agent will have three units of fuel while still being at location `ab1`. If the agent chooses the first option, it will get stuck at `ab2`, as it has no fuel and there is no possibility to tank. If the agent chooses the second option, it can execute two `move` actions after tanking and get to location `b`. Then the only option is to execute the conditional action for unloading parcels two times, after which the achievement goal of having the parcels at location `b` is reached.

4 Semantics of Maintenance Goals

In this section, we define the semantics of a GOAL agent if this agent is given a set of maintenance goals to satisfy. In defining the operational semantics for maintenance goals, the idea is that agents reason about the result of the execution of their actions, in order to make sure that only those actions are chosen that do not violate the agent's maintenance goals. That is, agents *look ahead* in order to foresee the consequences of their actions. Adding maintenance goals that may have a constraining role makes sense only if the original agent is underspecified, that is, if alternative courses of action are available, as in the case of GOAL agents. Only then can the agent actually *choose* to take actions that do not violate maintenance goals. Intuitively, the idea is thus that the incorporation of maintenance goals leads to the exclusion of (parts of) computations that were allowed in the agent semantics of Definition 5 without maintenance goals.

In the example program of Section 3, we have seen that the carrier agent gets stuck at location `ab2` if it does not tank at location `ab1`. The idea is that such behavior can be prevented by introducing a maintenance goal that expresses that the agent should not be in a situation where it has no fuel left (Table 2).

```
:m-goals{ fuel(X), X > 0. }
```

Table 2. Extension With Maintenance Goals

Syntactically, the introduction of maintenance goals thus poses no problems. Incorporating maintenance goals in the semantics, however, is more involved and is the subject of the remainder of this section. In Section 5 we look at the case that maintenance and achievement goals cannot be satisfied simultaneously.

4.1 Operational Semantics of Maintenance Goals

Ideally, an agent should look ahead infinitely far into the future, in order to be absolutely sure that it does not choose a path that will lead to the violation of a maintenance goal. In practice, however, infinite lookahead cannot be implemented, and presumably it will neither be necessary. We propose a general definition of lookahead, that takes the *number of steps* that an agent may look ahead as a parameter. This parameter is called the *lookahead range.*

In the following, $\Omega \subseteq \mathcal{L}_0$ will denote a set of maintenance goals. A set of maintenance goals will be assumed to be consistent, i.e., $\Omega \not\models \bot$. If maintenance goals are hard constraints, it is not rational to have two maintenance goals that are inconsistent, as it will never be possible to satisfy both maintenance goals. Moreover, we assume that maintenance goals are satisfied initially, i.e., it should be the case that for the initial belief base Σ_0 we have $\Sigma_0 \models \Omega$ (where $\Sigma_0 \models \Omega$ abbreviates $\forall \omega \in \Omega : \Sigma \models \omega$). Also, we take the set of maintenance goals as being static. That is, an agent cannot drop or adopt new maintenance goals. Although there might be situations where one would want to consider dropping or adopting maintenance goals, we think that maintenance goals are intuitively more stable than achievement goals as the former express a kind of background requirements that an agent should always fulfill.

In order to provide a formal definition of the effect of n-step lookahead on the computations of an agent, we first introduce some additional terminology and notation. 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. We write $p \preccurlyeq p'$ to denote that p is a prefix of p'.

The order $\preccurlyeq$ is lifted to sets as follows: $S \preccurlyeq S'$ iff each $p \in S$ is a prefix of some $p' \in S'$. A set S of sequences is called a *chain* if for all $p, p' \in S$ we have either $p \preccurlyeq p'$ or $p' \preccurlyeq p$. The *least upper bound* of a chain S is denoted by $\sqcup S$. In case of a set S of prefixes of a computation t, $\sqcup S$ is either a maximal element in S (i.e. a prefix that has the greatest finite length), or the computation t itself (which need not be in S); moreover, $\sqcup\emptyset = \epsilon$ with ϵ the empty sequence. Finally, $s \in p$ for s a mental state and p a prefix of a computation abbreviates that s is a state on the prefix p; sometimes s_i is used to denote the i^{th} state in the sequence.

Now we are in a position to formally define how maintenance goals, given an n-step lookahead operator $\upharpoonleft_n$, restrict the possible computations of an agent $\mathcal{A}$. First, we define the notion of a *safe prefix* of a computation t, given a set of maintenance goals Ω and the capability to do a lookahead of n steps. The predicate $safe_n(p, \Omega)$, with $n \in \mathbb{N} \cup \{\infty\}$, is true if all states of the prefix p of computation t satisfy the maintenance goals Ω and, in the next n steps of computation t no violation of such a goal will occur, except possibly for the last state. (Note that we leave the computation t implicit in $safe_n(p, \Omega)$.) This corresponds with the behavior of a very cautious agent that will avoid to go in a direction that may lead towards a violation of a maintenance goal. Formally, we define $safe_n(\epsilon, \Omega)$ to be false for technical reasons, and we define $safe_n(t^{\langle k\rangle}, \Omega)$ for prefixes of non-zero length $k > 0$ as follows:

$$safe_n(t^{\langle k\rangle}, \Omega) \text{ iff } \forall s \in t^{\langle k+n-1\rangle}(\Sigma_s \models \Omega).$$

When the set of maintenance goals Ω is clear from the context, we also simply write $safe_n(t^{\langle k\rangle})$. All states on a safe prefix of a computation t based on n-step lookahead have the property that lookahead does not predict any violations of a maintenance goal in Ω in less then n steps. Note that there is at least one non-empty safe prefix including the initial state using 0-step lookahead since a goal agent initially must believe that its maintenance goals are satisfied. The set of all safe prefixes of computation t is denoted by $Safe_n(t, \Omega)$. Note that the set $Safe_n(t, \Omega)$ is a chain and has a least upper bound, which is the computation t itself when all prefixes of t are safe.

The n-step lookahead operator $\upharpoonleft_n$ applied to a computation t and a set of maintenance goals Ω can now be defined in terms of safe prefixes. Using this operator it is easy to define the effect of maintenance goals as hard constraints on the behavior of an agent with an n-step lookahead capability: The semantics $S_{\mathcal{A}}$ of an agent without such goals, i.e. its associated set of computations, is restricted by applying the lookahead operator to each computation in $S_{\mathcal{A}}$ to ensure that an agent with such lookahead capabilities will act cautiously and will never head towards a predicted violation of one of its maintenance goals.

Definition 6 (Lookahead Operator and Semantics of Maintenance Goals)
The n-step lookahead operator $\upharpoonleft_n$, applied to a computation t and a set of maintenance goals Ω, is defined as the least upper bound of the set of safe prefixes of t with respect to Ω, and is also lifted to sets of computations.

– The n-step lookahead operator $\upharpoonleft_n$ is defined as: $t\upharpoonleft_n\Omega = \sqcup Safe_n(t, \Omega)$.

– The lift of $\upharpoonright_n$ to a set S is defined by:

$$S\upharpoonright_n \Omega = \bigcup_{t \in S} \{t\upharpoonright_n \Omega \,|\forall t' \in S : t\upharpoonright_n \Omega \preccurlyeq t'\upharpoonright_n \Omega \Rightarrow t\upharpoonright_n \Omega = t'\upharpoonright_n \Omega\}$$

– Let $\mathcal{A}$ be an agent with an n-step lookahead capability. Then the semantics of $\mathcal{A}$ with a set of maintenance goals Ω is defined as: $S_{\mathcal{A}}\upharpoonright_n \Omega$.

The lift of $\upharpoonright_n$ to a set S is the set of all maximal elements of the set $\bigcup_{t\in S} t\upharpoonright_n \Omega$. Only the maximal elements are taken in order to exclude prefixes p that are a strict prefix of another prefix p' in this set, i.e., $p \prec p'$. The semantics $S_{\mathcal{A}}\upharpoonright_n \Omega$ for an agent $\mathcal{A}$ with maintenance goals Ω thus specifies that the agent continues until all further action would lead to a violation within n steps. Note that the set $S_{\mathcal{A}}\upharpoonright_n \Omega$ may be empty when the set of maintenance goals Ω is so restrictive that each computation would violate a maintenance goal within n steps.

4.2 Properties

The following proposition says that a lookahead capability with a bigger lookahead range than another one is more restrictive than the latter. Since the semantics implements a cautious strategy towards possible violations of maintenance goals, an agent that detects such potential violations sooner, will act cautiously and will not follow a course of action that may lead to this violation.

Proposition 1. *If $n > m$, then $S_{\mathcal{A}}\upharpoonright_n \Omega \preccurlyeq S_{\mathcal{A}}\upharpoonright_m \Omega$.*

The proposition suggests that agents with a more powerful lookahead capability, i.e. with a greater lookahead range, possibly are able to satisfy fewer achievement goals than they would be able to satisfy with a less powerful lookahead capability. That is, an agent that does everything to avoid maintenance goal violation will not allow itself to achieve a highly valued goal on a path that will lead to such a violation. Such computation paths may be excluded by the the more powerful lookahead capability while still being allowed by the weaker one.

For the idealized situation where an agent has infinite lookahead, we have the following proposition.

Proposition 2. (Infinite Lookahead Maintenance Goal Semantics)

$$S_{\mathcal{A}}\upharpoonright_\infty \Omega = \{t \in S_{\mathcal{A}} \mid \forall s \in t : \Sigma_s \models \Omega\}$$

This proposition states that an agent with infinite lookahead will only execute a computation that is completely free of maintenance goal violations. For the example carrier agent, if we assume infinite lookahead, any computation where the agent does not tank at location `ab1` are excluded from the semantics. The reason is that in these computations the agent will violate its maintenance goal as it will be at location `ab2` without any fuel.

Although the infinite lookahead semantics is elegant and captures intuitions in a simple manner, such lookahead cannot be implemented. In the next proposition we look at *bounded lookahead* where lookahead ranges are less than ∞.

Proposition 3. (Bounded Lookahead Maintenance Goal Semantics)
Let $n \in \mathbb{N}$. The n-step lookahead semantics $S_{\mathcal{A}}\lceil_n \Omega$ is equal to:

$$\bigcup_{t \in S_{\mathcal{A}}} \{p \prec t \mid safe_n(p) \ \& \ (\forall p', t' : p \preccurlyeq p' \prec t' \ \& \ safe_n(p') \Rightarrow p = p')\} \cup S_{\mathcal{A}}\lceil_\infty \Omega$$

Corollary 1. (One-Step Lookahead Maintenance Goal Semantics)
The one-step lookahead semantics $S_{\mathcal{A}}\lceil_1 \Omega$ of an agent $\mathcal{A}$ is equal to:

$$\bigcup_{t \in S_{\mathcal{A}}} \{p \prec t \mid (\forall s \in p : \Sigma_s \models \Omega) \ \& \ (\forall t' : p \prec t' \ \& \ s_{k+1} \in t' \Rightarrow \Sigma_{s_{k+1}} \not\models \Omega)\} \cup \ S_{\mathcal{A}}\lceil_\infty \Omega$$

Bounded lookahead implies that the agent may choose a path which inevitably will violate a maintenance goal because potential violations of the maintenance goal lie outside of the agent's lookahead range. As discussed above, it might be the case that on such a path an achievement goal is achieved that would never have been achieved if the agent would have had a greater lookahead range that would have predicted these violations. Note, however, that the fact that an agent takes a path on which it would violate a maintenance goal if it would continue still does *not* lead to violation of a maintenance goal. The reason is that the agent will be required to stop acting as soon as there are only actions enabled that would lead to a violation of a maintenance goal. This is in line with our assumption that maintenance goals are hard constraints.

In our example carrier agent it is sufficient to have a lookahead of one. As stated in Corollary 1, an agent with a lookahead range of one continues acting until it recognizes that by doing so at all possible next states it violates a maintenance goal. The carrier agent with a lookahead of one will be able to detect that if it executes a `move` action at location `ab1` before tanking, it will immediately violate its maintenance goal and will select the alternative action of tanking as a result. This illustrates that the lookahead mechanism, which primarily *constrains* the actions of the agent, may also induce the agent to *actively* prevent the violation of maintenance goals (in the example realized through tanking). To be more accurate, our mechanism does not distinguish between preventive actions that should prevent the violation of an achievement goal, and actions that are executed to fulfill achievement goals. As we can see in this example, in practice a very limited lookahead range may already be sufficient to prevent the agent from taking a path that would lead to violation of maintenance goals. To be more specific, the semantics of the example agent with lookahead range of one is equal to the semantics with lookahead range ∞. In general, the minimally needed lookahead range should be derived from available domain knowledge.

In this simple example, it is not difficult to modify the GOAL program in such a way that the desired behavior is obtained without explicitly incorporating maintenance goals. One could, e.g., add a condition to the conditional action for moving, specifying that if the agent is at location `ab1`, it may not move unless its tank is full. We argue, however, that the explicit incorporation of maintenance goals in the GOAL program provides a separation of concerns, and thereby potentially yields more transparent and easier to verify agent programs.

It is interesting to investigate under what circumstances bounded lookahead is guaranteed to be sufficient to avoid violation of maintenance goals. One particular such case is the case that an agent can *undo* actions, that is, if it has a *rollback mechanism* to go back to a previous state. In the presence of such a rollback mechanism, a bounded lookahead of 1 is sufficient to satisfy all maintenance goals. Obviously, the ability to rollback combined with 1 step lookahead will not be sufficient in all cases to realize the agent's achievement goals. The combination does allow the agent, however, to continue any computation given that at least one action is enabled. For our purposes, we model such a rollback mechanism simply by adding for each transition $s \rightarrow s'$ the inverse transition $s' \rightarrow s$ to the agent semantics.

Theorem 1. (Lookahead of One Sufficient with Rollback Mechanism)
For agents that can do at least one action initially without violating a maintenance goal, and that have a rollback mechanism to undo arbitrary actions, that is, are able to reverse a computation step $s \rightarrow s'$ by performing the step $s' \rightarrow s$, we have the following:

$$S_{\mathcal{A}} \upharpoonright_1 \Omega = S_{\mathcal{A}} \upharpoonright_\infty \Omega$$

Proof. The main observation needed in the proof is that any finite, safe prefix can be continued without violating a maintenance goal by doing either a "regular" action or otherwise by doing an "undo" action. By assumption, the agent can at least do one action initially, and so any finite safe prefix can be extended to a complete computation that does not violate a maintenance goal.

Although Theorem 1 shows that an agent will always be able to continue pursuing its goals, it does not state that it will also achieve these goals if possible. In the presence of a rollback mechanism, computations that make no progress but instead repeatedly have to recover from performing an action that leads to a violation of a maintenance goal are included in the set $S_{\mathcal{A}} \upharpoonright_\infty \Omega$. What is missing is a notion of fairness that would prevent such repeated execution of a part of the computation (cf. [7]). Fairness is included in the original GOAL semantics but is not discussed further in this paper (cf. [5]). Intuitively, moreover, by using lookahead of more than one step computations that require rollback can be detected sooner which will reduce the need for such rollbacks.

5 Detecting and Revising Goal Conflicts

In this section an algorithm is presented that implements the maintenance goal semantics and, additionally, it includes an extension that provides the agent with the option to revise its achievement goals in case no achievement goal is reachable without choosing a path that would lead to violation of a maintenance goal. As discussed in Section 2.2, revising achievement goals is a way of dealing with conflicts between maintenance goals and achievement goals, if maintenance goals are taken as hard constraints. Revision of achievement goals is not the main subject of this paper (see e.g. [9]), but we will illustrate the main ideas using the carrier agent example.

```
Function SelectAction($E, s, n$)
Input: A set of enabled conditional actions $E$, a state $s$, a lookahead range $n$
Output: A selected conditional action $c$, or skip
1.  $actionOkSet \leftarrow \emptyset$
2.  for each $c \in E$
3.      do $conflict[c] \leftarrow$ ConflictSets($c, s, n$)
4.          if $\emptyset \in conflict[c]$ then $actionOkSet \leftarrow actionOkSet \cup \{c\}$
5.  if $actionOkSet \neq \emptyset$
6.      then return ChooseAction($actionOkSet$)
7.      else $c' \leftarrow$ SelectActionWithMinimalConficts($E, conflict$)
8.              ReviseConflictingAchievementGoals($conflict[c']$)
9.              (* do nothing and recompute enabled actions using revised achievement goal(s) *)
10.             return skip

Function ConflictSets($c, s, n$)
Input: A conditional action $c$, a state $s$, and a lookahead range $n$
Output: The conflict sets of $c$
1.  if $n \leq 0$
2.      then return $\{\emptyset\}$ (* Indicates that at least one path is ok. *)
3.      else $S \leftarrow$ SuccessorStates($c, s$)
4.              for each $s' \in S$
5.                  do $cset \leftarrow \emptyset$ (* Conflict set *)
6.                      if $\Sigma_{s'} \not\models \Omega$
7.                          then $cset \leftarrow cset \cup \{$ReasonConflict($c$)$\}$
8.                          else $E \leftarrow$ ComputeEnabledActions($s'$)
9.                                  for each $c' \in E$
10.                                     do $cset \leftarrow cset \cup$ ConflictSets($c', s', n-1$)
11.             return $cset$
```

Table 3. Action Selection Algorithm Including Maintenance Goals

The first step to implement the semantics for maintenance goals based on lookahead is to define an algorithm which is able to *detect* potential future maintenance goal violations. The algorithm depicted in Table 3 implements the detection of such violations as well as the cautious strategy of an agent that avoids taking a path that would lead to violation of a maintenance goal. The function SelectAction computes for each enabled conditional action whether it might result in any conflicts with or violations of maintenance goals for a given lookahead range n. In case executing an action does not inevitably lead to such a conflict, it is added to the set of actions that are *ok* to select for execution. Only if there are no actions that are "safe" in this sense, the action selection algorithm will select an achievement goal in order to revise it. The detection of these conflicts is done through the function ConflictSets. This function recursively computes the so-called conflict sets, which will be explained in more detail below. An empty conflict set indicates that no future violation of a maintenance goal within lookahead range is detected.

As discussed in Section 2.2, detected conflicts between achievement goals and maintenance goals may cause the agent not to do anything at a certain point, as it might be the case that any action would lead to a future violation of a maintenance goal. In the example scenario, adding a weight constraint that expresses that the truck cannot carry a load that weighs more than a certain threshold, has this effect if the sum of the weight of the two parcels is higher than the threshold (see Table 4, where `weightTotal(N)` computes the total weight of the parcels in the truck).

```
:beliefs{ ... weight(p1,3). weight(p2,2). threshold(4). weightTotal(N) :- ... }
:a-goals{ loc(p1,b), loc(p2,b). }
:m-goals{ fuel(X), X > 0. weightTotal(T), threshold(W), T<W. }
```

Table 4. GOAL Carrier Agent

If at least a lookahead of two is used, the agent will not be able to execute any action in the initial mental state. After loading either one of the parcels, loading the other one would lead to a violation of the weight maintenance goal. With the cautious strategy, taking a path on which the violation of a maintenance goal is foreseen within two steps, is not an option (note that the agent can only unload parcels at location `b`).

In this case where the agent cannot execute any action as this would lead to violation of maintenance goals, the algorithm of Table 3 allows the revision of achievement goals by means of lowering ones ambitions. The idea here is that actions are induced by achievement goals and these actions thus may be prevented from being taken by revising those goals (we disregard the possibility of incorrect beliefs, which might instead require an agent to revise its beliefs). In order to revise its achievement goals the agent needs more information to base the revision on and to this end the notion of a *conflict set* is introduced. A *conflict set* is an achievement goal α which has been identified as a potential reason for the violation of a maintenance goal. In general, identifying such a reason may involve complicated diagnostic reasoning, but in GOAL a more pragmatic solution is available. In GOAL, goal conditions are typically associated with the selection of actions and we can simply take these conditions as the reason why a maintenance goal is violated. In our example agent, the function ReasonConflict(c) extracts *an instance* of the goal condition `loc(P,b)` as a reason for the violation of the maximum weight loaded. The function Revise-ConflictingAchievementGoals then may revise the achievement goal in the goal base and drop one of the conjuncts to avoid the violation. Consecutively, the agent verifies again if the maintenance goal violation has been eliminated. If no reason can be identified in this way, # is returned to indicate a violation of a maintenance goal.

6 Conclusion and Related Work

In this paper, we have looked at a mechanism for agents to handle maintenance goals. In particular, we have proposed a formal semantics of maintenance goals based on the notion of lookahead, and we have analyzed the semantics by proving some properties, in order to gain a better understanding of the role of maintenance goals in action selection. We presented an algorithm for detecting maintenance goal violation, parametrized by a variable lookahead range in order to be able to control computational costs. Additionally, we have discussed the issue of achievement goal revision, in case the maintenance goals are so restrictive that all courses of action for satisfying achievement goals will lead to a violation of maintenance goals.

There are several interesting directions for future research. Regarding the revision of achievement goals, several issues have remained unexplored. For example, we have suggested one possible way of determining that an achievement goal conflicts with a maintenance goal. In future research, we plan to investigate this approach and possible alternatives in more detail. One research direction in this respect is the investigation of existing techniques for determining whether achievement goals conflict with each other [16, 15, 13]. It will need to be investigated whether the issue of conflicts between maintenance goals and achievement goals is the same as or similar to the issue of conflicts between achievement goals.

Existing approaches for defining preferences over goals, such as in utility theory [1], may be useful to refine the strategy for revising achievement goals. Intuitively, an agent should revise its achievement goals in such a way that they are reachable without violating maintenance goals, and the revision should maximize the agents expected utility. Moreover, in this paper we have taken maintenance goals as hard constraints, and have suggested to revise achievement goals in case they conflict with the agent's maintenance goals. Alternatively, it could be allowed to violate maintenance goals under certain circumstances. Again utility theory could be useful here, in order to weigh the violation of a maintenance goal against the realization of an achievement goal. For example, negative utility could be associated with the violation of a maintenance goal assigning a maintenance goal that defines a hard constraint e.g. as having infinitely negative utility. The work in [8] on qualitative preferences in agent programming could also be relevant here. There are also some similarities with the planning literature on oversubscription (e.g. [2]), but as with planning approaches in general the main difference is that GOAL agents check violations of maintenance goals while executing actions.

Regarding related work on maintenance goals, we discuss the approach followed in the Jadex framework [3], the language presented by Dastani et al. [4], and the work of Duff et al. [6]. These approaches can be categorized into approaches that use maintenance goals as a *trigger* for the execution of actions, and approaches that use some mechanism for *reasoning* about the result of action execution in order to prevent maintenance goals from being violated. Jadex uses maintenance goals to trigger the execution of actions in case the maintenance goal is violated. In the framework of Dastani et al., a trigger condition is used to determine when action is needed to prevent the violation of maintenance goals. In our approach and in the framework of Duff et al., a reasoning mechanism is used in order to prevent maintenance goals from being violated.

One of the main differences between the work of Duff et al. and our work is that in Duff et al. it is determined *before* an achievement goal is pursued whether the plans for achieving this achievement goal may conflict with one of the agent's maintenance goals. In our work, by contrast, we propose to use a lookahead mechanism for keeping maintenance goals from being violated *during* pursuit of achievement goals. We also suggested the possibility to revise achievement goals when they cannot be realized without violating maintenance goals, while Duff et al. propose to not adopt such achievement goals to avoid the risk of violating maintenance goals. The approaches also differ in that in this paper a mechanism to ensure satisfaction of maintenance goals is based on a semantic analysis and Duff et al. validate their work using an experimental approach.

Finally, an advantage of doing lookahead during achievement goal pursuit, we believe, is that it may provide for more flexible agent behavior. An approach based on executing a preventive plan that is associated with the maintenance goal in case an achievement goal might conflict with a maintenance goal, as proposed in Duff et al., does not seem to leave the agent with as many options as are possible. Moreover, such an approach still does not guarantee that the consecutive pursuit of the achievement goal will not violate the maintenance goal. The approach of Duff et al. can be compared with planning approaches, in the sense that reasoning takes place before execution. If something is about to go wrong during execution, this is not detected. In our approach, the agent pursues achievement goals, but takes any measures that it has at its disposal if this is necessary to prevent a maintenance goal from being violated.

References

1. Craig Boutilier, Thomas Dean, and Steve Hanks. Decision-theoretic planning: Structural assumptions and computational leverage. *Journal of AI Research*, 11:1–94, 1999.
2. Ronen I. Brafman and Yuri Chernyavsky. Planning with goal preferences and constraints. In *Proceedings of ICAPS'05*, 2006.
3. Lars Braubach, Alexander Pokahr, Daniel Moldt, and Winfried Lamersdorf. Goal representation for BDI agent systems. In *Programming multiagent systems, second international workshop (ProMAS'04)*, volume 3346 of *LNAI*, pages 44–65. Springer, Berlin, 2005.
4. Mehdi Dastani, M. Birna van Riemsdijk, and John-Jules Ch Meyer. Goal types in agent programming. In *Proceedings of the 17th European Conference on Artifical Intelligence 2006 (ECAI'06)*, volume 141 of *Frontiers in Artificial Intelligence and Applications*, pages 220–224. IOS Press, 2006.
5. F.S. de Boer, K.V. Hindriks, W. van der Hoek, and J.-J.Ch. Meyer. A Verification Framework for Agent Programming with Declarative Goals. *Journal of Applied Logic*, 2006. To appear.
6. Simon Duff, James Harland, and John Thangarajah. On Proactivity and Maintenance Goals. In *Proceedings of the fifth international joint conference on autonomous agents and multiagent systems (AAMAS'06)*, pages 1033–1040, Hakodate, 2006.
7. N. Francez. *Fairness.* Springer, 1986.
8. Christian Fritz and Sheila A. McIlraith. Decision-theoretic golog with qualitative preferences. In *KR*, pages 153–163, 2006.
9. P. Gardenfors. *Belief Revision.* Cambridge Computer Tracts. Cambridge University Press, 1992.
10. 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.

11. Jomi Fred Hübner, Rafael H. Bordini, and Michael Wooldridge. Declarative goal patterns for AgentSpeak. In *Proceedings of the fourth International Workshop on Declarative Agent Languages and Technologies (DALT'06)*, 2006.
12. V. Lifschitz. On the semantics of strips. In M.P. Georgeff and A.L. Lansky, editors, *Reasoning about Actions and Plans*, pages 1–9. Morgan Kaufman, 1986.
13. Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. A goal deliberation strategy for BDI agent systems. In *MATES 2005*, volume 3550 of *LNAI*, pages 82–93. Springer-Verlag, 2005.
14. Sebastian Sardina and Steven Shapiro. Rational action in agent programs with prioritized goals. In *Proceedings of the second international joint conference on autonomous agents and multiagent systems (AAMAS'03)*, pages 417–424, Melbourne, 2003.
15. J. Thangarajah, L. Padgham, and M. Winikoff. Detecting and avoiding interference between goals in intelligent agents. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI 2003)*, 2003.
16. J. Thangarajah, M. Winikoff, L. Padgham, and K. Fischer. Avoiding resource conflicts in intelligent agents. In F. van Harmelen, editor, *Proceedings of the 15th European Conference on Artifical Intelligence 2002 (ECAI 2002)*, Lyon, France, 2002.
17. M. Birna van Riemsdijk, Mehdi Dastani, John-Jules Ch Meyer, and Frank S. de Boer. Goal-oriented modularity in agent programming. In *Proceedings of the fifth international joint conference on autonomous agents and multiagent systems (AAMAS'06)*, pages 1271–1278, Hakodate, 2006.
18. Micheal Winikoff, Lin Padgham, James Harland, and John Thangarajah. Declarative and procedural goals in intelligent agent systems. In *Proceedings of the eighth international conference on principles of knowledge respresentation and reasoning (KR2002)*, Toulouse, 2002.