The functions ϕ and $\hat{\phi}$ now allow us to define a *goal template*. The intuition is that each type of goal, such as *secureArea* or *evacuateForest*, has an associated template. On the basis of a goal template, goal instances can be created.

Definition 1 (goal template). *Consider a multi-agent system (MAS) for which the set of possible states is defined as S. Let A be a nonempty set with a partial ordering $\leq$ (the* progress metric*), and let M be a set representing means that can be used for achieving a goal. A* goal template *T is then defined as a tuple $\langle A, M, \phi : S \to A, \hat{\phi} : S \times M \to A \rangle$, where ϕ is the* progress appraisal function*, and $\hat{\phi}$ is the* progress upper bound function*.*

This notion of goal template may be simplified to consist of only A and ϕ, if $\hat{\phi}$ cannot be provided in a certain case, i.e., when no sensible upper bound can be specified for a goal. Alternatively, it may be extended in various ways. First, the goal template itself may be parameterized to account for variants of the template. For example, depending on the area that has to be secured, the number of road blocks that have to be set up will differ, and this may influence the definition of ϕ and $\hat{\phi}$. Second, one may want to define a goal template for a single goal based on different progress metrics, allowing the agent to choose a progress metric depending on circumstances. We can capture this most simply by having two separate goal templates. Formally relating these templates (for instance by making them siblings in a hierarchy of goal types) is an extension of our basic framework. For reasons of simplicity and space, we leave the pursuit of these extensions for future work.

As noted, in order to simplify definition and computation of ϕ and $\hat{\phi}$, these functions may yield *estimated* values for progress appraisal and the upper bound. In environments that are not fully observable or that are open or dynamic, the agent may not be able to compute precisely the functions. However, an agent must be mindful of the potential adverse effects of estimation. In over-estimation of $\hat{\phi}$ or under-estimation of ϕ, the agent would try to achieve a goal even though it may be impossible to fully satisfy it, or it is already completely satisfied. On the other hand, in under-estimation of $\hat{\phi}$ or over-estimation of ϕ the agent would stop too soon. Thus, while ϕ and $\hat{\phi}$ may yield estimated values, intuitively the agent should estimate the progress upper bound in a state $s \in S$ to be at least the current progress in that state. We call this *coherency* of a goal template, and formally define it as $\forall s \in S, m \in M : \hat{\phi}(s, m) \geq \phi(s)$.

To illustrate Def. 1, consider the goal *secureArea* of the example scenario. In the scenario, the main resource (leaving aside time) is the number of police officers $P = \{0, \ldots, 10\}$. We base the progress metric for the goal on the NUMBER OF SUBGOALS ACHIEVED.

Example 1. The goal template for *secureArea* is: $T_{sa} = \langle \mathbb{R}, P, \phi_{sa}, \hat{\phi}_{sa} \rangle$. Thus, the progress metric is $A = \mathbb{R}$ with its standard $\leq$ ordering. Arbitrarily, we define $\phi_{sa}(s)$ to be 20 if all subgoals have been fully achieved in s (assuming the agent can determine this in each state s), which means that road blocks have been set up and at least one police officer guards each road block, 10 if all road blocks have been set up but not all of them have at least one officer, and 0

otherwise. Let the means include p, the number of officers allocated. We define $\hat{\phi}_{sa}(s, p)$ to be 20 iff the plan *EstablishRoadblocks* can be executed in s and it is executed with at least 6 police officers, i.e., $p \geq 6$, 10 if $1 \leq p < 6$ and the plan can be executed successfully, and 0 otherwise. Computation of the upper bound thus requires determining whether *EstablishRoadblocks* can be executed successfully. This may be done by checking simply the precondition of the plan, or by performing planning or lookahead (compare [3, 12]).

A goal template specifies the progress appraisal and progress upper bound functions. As already addressed above, we need to specify the *completion value* for a goal to specify when it is completely satisfied. In addition, the agent should determine the means that will be allocated for pursuing the goal. The completion value and means together form a goal instance.

Definition 2 (goal instance). *Let $T = \langle A, M, \phi : S \to A, \hat{\phi} : S \times M \to A \rangle$ be a goal template. A* goal instance *of T is specified as $(a_{min}, m) : T$, where $a_{min} \in A$ is the completion value, and $m \in M$ specifies the means that will be used for achieving the instance.*

Example 2. In the scenario, one goal instance of the goal template T_{sa} for *secureArea* is $g_{sa} = (20, \{0, \ldots, 6\}) : T_{sa}$, expressing that the commander would like to achieve a progress metric value of 20 with no more than six police officers.

Achievement. Using this notion of goal instance, we can easily define when a goal is *achieved* (completely satisfied) in a certain state $s \in S$, namely, when the appraised value of the progress metric in s is at least the completion value.

We will assume that each progress metric $(A, \leq)$ has a (totally ordered) bottom element $\bot_a \in A$ for which $\forall a \in A$ with $a \neq \bot_a$, we have $\bot_a < a$. The bottom element represents a 'zero' achievement level. When a goal instance g is created, it may start partially completed, i.e., $\phi(s) > \bot_A$ where s is the state in which g is created. For example, the road block on road 1 may already be in place, when an instance of *secureArea* is created, because the road was closed for construction.

We can now formally define goal achievement. In addition, we use the progress upper bound function and the means of the goal instance to define a kind of *goal consistency*. In logic-based frameworks for goals [30, 31, 10], an inconsistent goal is not reachable by definition. Achievement in our framework for partial goal satisfaction is similar. We say that a goal instance is *achieved* in a state s if the maximum attainable value of the progress metric from s, given the means of the goal instance, is at least the completion value.

Definition 3 (goal achievement and achievability). *Let T be a goal template, let $(a_{min}, m) : T$ be an instance of T, and let $s \in S$ be the current state. The goal instance $(a_{min}, m) : T$ is* completely unachieved *iff $\phi(s) = \bot_A$,* (completely) achieved *(or* satisfied*) iff $\phi(s) \geq a_{min}$, and* partially achieved *otherwise, i.e., iff $\bot_A < \phi(s) < a_{min}$. The goal instance is* achievable w.r.t. m *(or simply* achievable*, where the context is clear) iff $\hat{\phi}(s, m) \geq a_{min}$.*

For example, the goal instance g_{sa} of Example 2 above is achieved if all road blocks have been set up and each remains guarded by at least one police officer (since in that case the achieved ϕ_{sa} value is 20). It is achievable in any state $s \in S$ since six police officers are allocated for achieving the instance, whence the progress upper bound is 20, equalling the completion value. If less than six officers were allocated, the goal instance would not be achievable since then the agent could maximally attain a ϕ_{sa} value of 10.

4.1 Binary Goal Achievement

We now discuss how our framework relates to logic-based frameworks for (achievement) goals. In the latter, as noted in Sect. 2, the success condition of a goal is usually defined as a logical formula s, which is achieved in a state $s \in S$ if the agent believes s to hold in that state. We show how our definition for partial goal achievement can be instantiated such that it yields the usual binary definition of goal. We abstract from means M.

Definition 4 (binary goal instance). *Let ψ be a logical formula, for which the truth value can be determined in the current MAS state s (denoted as $s \models \psi$ iff ψ is entailed). Let $A = \{\text{false}, \text{true}\}$ with* true $>$ false. *Let $M = \{\epsilon\}$ where ϵ is a dummy element. Let $\phi(s) =$* true *if $s \models \psi$ and* false *otherwise, and let $\hat{\phi}(s, \epsilon) =$* true *if $\psi \not\models \bot$ and* false *otherwise. Let $T_{bin(\psi)} = \langle A, M, \phi, \hat{\phi} \rangle$. Then we define a* binary goal instance *$\psi = (\text{true}, \epsilon) : T_{bin(\psi)}$.*

Proposition 1 (correspondence). *The instantiation of the partial goal framework as specified in Def. 4, corresponds to the binary definition of goal (Sect. 2) with respect to achievement and consistency.*

Proof. We have to show that achievement and consistency hold in the binary definition of goal, iff achievement and achievability hold in the instantiated partial definition of goal. The goal ψ is achieved in the partial case in some state s iff $\phi(s) \geq a_{min}$, i.e., iff $\phi(s) \geq$ true, i.e., iff $\phi(s) =$ true, i.e., if $s \models \psi$. This is exactly the definition of achievement in the binary case. The goal ψ is achievable in the partial case iff $\hat{\phi}(s, \epsilon) \geq a_{min}$, i.e., iff $\hat{\phi}(s, \epsilon) =$ true, which is precisely the case iff ψ is consistent. □

We envisage an instantiation of our framework with the logic-based characterization of partiality of Zhou et al. [32], where in particular the ordering relation of the progress metric will have to be defined. That is, consider a semantics of partial implication and an alphabet of atoms. Intuitively, we must specify a metric on a set such as propositions over the alphabet, that gives rise to a partial order of the propositions w.r.t. the semantics of implication. Making this instantiation precise will be future research. Indeed, the role of partial implication in connection with subgoals and plans—which we account for in our framework through the computation of metrics in the GPT—has already been noted as a research topic [32].

The instantiation of our framework with a binary goal definition emphasizes that progress metrics need not be numeric. However, if ϕ_g is numeric, the agent

can compute how far it is in achieving a goal as a ratio with the completion value. That is, if T is a goal template with progress appraisal function $\phi : S \to A$ and $g = (a_{min}, m) : T$ is a goal instance of T, and if quotients in A are defined (e.g., if $A = \mathbb{R}$), then a measure of progress of goal instance g when the agent is in state s is the ratio $\frac{\phi(s)}{a_{min}}$. This metric of % COMPLETE corresponds to the intuitive notion of progress as the percentage of the completion value attained.

5 Goal Adaptation

The previous section outlined an abstract framework for partial goal satisfaction. We have taken progress appraisal as the most basic form of reasoning that such a framework should support. In the motivating scenario, we argued that the framework should support more advanced kinds of reasoning, such as goal negotiation. In this section, we highlight a type of reasoning that we suggest underlies many of these more advanced kinds of reasoning, namely *reasoning about goal adaptation*. Given a goal instance $g = (a_{min}, m) : T$ where T is a goal template, we define *goal adaptation* as modifying a_{min} or m (or both). Note that modifying the plan for g is included in the scope of modifying m.

The reasoning question is how to determine which goals to adapt and how to adapt them. While this is a question that we cannot fully answer here, we analyze the kinds of adaptation and possible reasons for adapting. One important factor that may influence the decision on how, and particularly when, to adapt is the evolution of the agent's beliefs. This aspect is a focus of prior works [22, 32]. Another important factor is the consideration of a *cost/benefit analysis*. We develop our basic framework to support this kind of reasoning.

5.1 Reasons for and Uses of Adaptation

We begin by distinguishing *internal* and *external* reasons for goal adaptation. By internal reasons for we mean those that arise from issues with respect to the goal itself, while external reasons are those that arise from other factors.

More specifically, we see a lack of achievability as a main internal reason for goal adaptation. If a goal instance g is not achievable, it means that its completion value cannot be attained from the current state with the means that are currently allocated. The options without a concept of partial satisfaction are to drop/abort g, to attempt a different plan for g (if possible), to suspend g until it becomes achievable (for example, waiting for more officers to arrive), or to abort or suspend another goal in favour of g. In our framework, the goal instance can be *adapted* to make it achievable by lowering the completion value, which we call *goal weakening*, as well as by the alternative of choosing different means that allows the achievement of the current completion value, e.g., by investing additional resources. Depending on the circumstances, the latter may not always be possible. For example, if the goal is to evacuate people from their houses but it is physically not possible to get to these houses, e.g., because of flooding, it does not matter whether the officers devote more time or personnel.

Several external reasons may lead to goal adaptation. First, a goal instance g may in itself be achievable, but (collective) unachievability of other goal instances may be a reason for adapting g. That is, in practice an agent has only limited resources and it has to choose how it will invest them to achieve a set of current and future goal instances [1, 27]. For example, the agent may decide that another goal instance is more important and needs resources, leading to adaptation of the means of g. In our framework, goal adaptation provides the agent with the option of *partially* suspending, replanning, or abandoning goals. Moreover, progress appraisal helps the agent determine which goals to adapt. For example, it does not seem sensible to drop a goal instance that has a plan that is almost completed and that yields zero utility unless it is completely satisfied.

There are further external reasons for goal adaptation. Second, a particular case is consideration of a new candidate goal instance g': the question of *goal adoption*. Partial satisfaction allows an agent to consider adapting an existing goal instance, or adopting the new instance g' in a weakened form. Third, an agent might be requested by another agent to increase the completion value of a goal instance, which we call *goal strengthening*. For example, the team leader may decide that more time should be spent searching the forest.

Together, progress appraisal and goal adaptation form a basis for higher-level reasoning tasks. We have already discussed goal negotiation (Sect. 3), goal adoption, and avoiding and resolving goal achievement inconsistencies. We now briefly discuss several other kinds of reasoning. First, in order to coordinate their actions, agents should *communicate* about how far they are in achieving certain goals [7, 17, 15]. Progress appraisal provides a principled approach. Second, an agent might realize it cannot achieve a goal completely. Allowing itself to weaken a goal, it can *delegate* part of the goal to other agents. Similarly, delegation may be another option for an agent finding it has achievement difficulties. Related, third, is *reasoning about other agents* and their ability to complete tasks. For example, one agent realizing that another agent is unlikely to fully complete its task(s), irrespective of whether the other agent has acknowledged this.

5.2 Cost/Benefit Analysis

When deciding which goals to adapt and how, we suggest that a cost/benefit analysis can be an important consideration (see also, e.g., [1, 21]). We have already noted the example of an agent pursuing a goal that yields zero utility unless completely satisfied, for which only a small additional amount of effort is required. On the other hand, if an agent has obtained much utility from a goal instance g, compared to that expected when the progress metric of g reaches the completion value, and if much more effort would have to be invested to fully achieve g, it may be sensible to stop pursuit of the goal if resources are needed elsewhere. These kinds of cost/benefit analyses to obtain an optimal division of resources over goals essentially form an optimization problem. While it is beyond the scope of this paper to investigate how optimization techniques can be applied in this context, we do analyze how our framework supports it.

In order to weight up costs and benefits, one needs to know how much it would cost to achieve a certain benefit. The benefit obtained through progress on a goal can be derived in our framework by means of a UTILITY metric $u_g : S \to U$, where U is a set.[4] Note that the progress metric of a goal template might be defined in terms of u_g (as is the case for *publicSafety*), in which case $\phi_g \equiv u_g$ and $A \equiv U$; or u_g might be a different metric (as in the case of *secureArea*).

The incremental benefit obtained when achieving a goal completely is the difference between the benefit at a state s^* for which $\phi(s^*) \geq a_{min}$ (i.e., upon completion of the goal) and the benefit in the current state s_{now}. That is, for a goal instance $(a_{min}, m) : T$, the incremental benefit is $\Delta u = u(s^*) - u(s_{now})$. Note that Δu can only be calculated in this way if differences are defined on U, which will be the case if U is numeric.

The cost associated with obtaining a_{min} can be computed by a COST function $\kappa : S \times M \times S \to C$, where C is a set and $\kappa(s, m, s') = c$ implies that the cost of going from state s with means m to state s' is estimated to be c. Then we can calculate the estimated minimal incremental cost to move from the current state s_{now} to a completion state s^* with means m as $\min_{s':\phi(s')\geq a_{min}} \kappa(s_{now}, m, s')$.

Supposing that the set that measures benefit, U, and the set that measures cost, C, are mutually comparable—for instance, if both are subsets of $\mathbb{R}$—then the estimates for utility achieved so far, utility expected upon completion, cost so far, and cost to completion can be compared.

6 Towards an Embedding within a Goal Framework

In this section, we illustrate how our metric-based framework for partial goal satisfaction can be applied to a concrete goal representation framework, namely the GPT as introduced earlier. This is a step towards rendering the capabilities within a cognitive agent programming framework. An attraction of the GPT is its representation of goals, subgoals, and plans—which is pertinent for reasoning about the means and the progress in execution of a goal—combined with the annotation of and aggregation of quantities on the tree nodes—which we will use for computation of metrics. Fig. 1 depicted a goal-plan tree for the evacuation scenario. The goal and action nodes correspond to goal instances in our framework; the tree structure gives the plan aspect of their means.

For the reasons just given, we posit that the concept of partially satisfied goals fits naturally into this kind of representation framework for goals. We augment annotations of tree nodes to include metrics about goal (and, where relevant, plan) satisfaction. In the simplest case, this comprises annotating each goal node with values from its progress metric A, as we will explain. The % COMPLETE metric allows normalization of the values.

Progress appraisal. Inference over the tree structure computes and updates metrics by propagation upwards from descendant nodes, in a similar fashion as resource estimates and other information are propagated [3, 24]. For example, the current value of % COMPLETE of a parent plan node may be aggregated

from the values of its child goal nodes. Metrics are aggregated according to their nature and the type of the node. For example, by default, a conjunctive plan node will aggregate % COMPLETE as the arithmetic mean of the children's values, while a disjunctive plan node will aggregate it as the maximum of their values. Mechanisms for aggregation have been explored in the cited literature. Since the algorithms are already parameterizable according to the nature of the quantity (in our case, the metric) and the type of the node, we need not repeat them.

The computation is to be made *dynamically* as the current situation evolves [18, 15]. We assume agents can assess the progress of leaf nodes. For instance, the police officers should believe they know when they have finished clearing a house (and so achieve the utility depicted on each leaf node). Hence, there are two types of metric values attributed onto nodes. The first type are *static*, initial, *a priori* values before execution (as depicted in Fig. 1). These correspond to expected, estimated, or required values, such as the utility expected upon full satisfaction of a goal (i.e., $u(s^*)$), and the resources expected to achieve this. The second type of metric values are *dynamic* estimates computed during execution, such as the utility achieved so far from a goal. For the progress metric of each goal instance g, the static value corresponds to the completion value a_{min} of g, while the dynamic value corresponds to the appraised value $\phi_g(s_{now})$.[5]

6.1 Reasoning in the Example Scenario

The response team commander is given the goal *publicSafety*. The doctrinal plan, *SecureAndClearArea*, involves the two subgoals, *secureArea* and *evacuatePeople*; the two may be achieved concurrently, although the team must be mindful that the public may (re-)enter the incident area until it is secured.

Goal templates, metrics, and goal instances. Recall from Example 1 that the goal template for *secureArea* is $T_{sa} = \langle \mathbb{R}, P, \phi_{sa}, \hat{\phi_{sa}} \rangle$, where the progress metric for T_{sa} is the achievement of its subgoals. The UTILITY metric of T_{sa} can be seen from Fig. 1 to be $u_{sa} = 5 * (\#$ achieved subgoals). u_{sa} may be of interest as a measure of progress, even though this metric does not *define* the progress (according to police doctrine) nor therefore the completion of the goal.

By contrast to *secureArea*, the progress metric of the initial goal *publicSafety* in the scenario is defined in terms of utility. Its goal template is $T_{ps} = \langle \mathbb{R}, P, u_\Sigma, \hat{u_\Sigma} \rangle$ where $\phi_{ps} \equiv u_\Sigma$ specifies the cumulative utility from the subgoals in the current plan for a goal instance of T_{ps}. This progress metric is computed in the obvious manner by recursively transversing the subtree below the goal instance, summing up the current utility estimates for each goal node. Likewise, the progress upper bound function, $\hat{u_\Sigma}$, can be computed by a recursive descent through the GPT. An *a priori* estimate can be computed, based on the upper bounds of the static, *a priori* utility attributions on leaf nodes [27, 3,

[5] An agent may be capable of directly computing the value of a metric at a (non-leaf) node. In that case, if the reasoning is consistent and the static values on leaf notes are reliable estimates, then the directly-computed and aggregated values should agree. Where they do not, the agent may resolve the conflict according to which of the two computations it believes is most reliable.

24]. For example, an *a priori* upper bound on *EstablishRoadblocks*, relaxing resource considerations, is $4+4+\max(2,5) = 13$. Tighter bounds can be obtained by considering resource limitations and the resulting goal interaction and plan scheduling [27, 24].

Goal adoption. The police commander and her team are tasked with the initial goal *publicSafety*; its goal instance is $(40, 10) : T_{ps}$. The team of 10, including the commander, has too few officers to meet the expected requirements for the full completion of the three roadblock actions (rb_i) and the two house-clearance actions (h_i), let alone the forest. That is, the goal instance is unachievable (i.e., $\hat{\phi_{ps}} < a_{min}$), as can be seen to be the case by examination of the GPT.

Negotiation, delegation, and requesting help. At first, the commander considers allocating six officers for *secureArea* and weakening the *evacuatePeople* goal by omitting the *evacuateForest* subgoal. This is unacceptable to incident control. After further negotiation, control agrees to send urgently a second team to perform *rb1*. The commander thus allocates four officers for *secureArea*. Hence, the goal instances are $(20, 4) : T_{sa}$ and $(25, 6) : T_{ep}$. Two officers will search each house; when done, they will join the forest search.

Appraisal and sharing information. As execution proceeds, updated metric values are computed on the leaf nodes of the GPT and aggregated to parent nodes. This provides a situational assessment for the commander. Searching house 2 is taking longer than anticipated. Should the two officers continue with *h2*, or join those searching the forest? Utility of 4 is estimated achieved from *h2* after 25 minutes have elapsed. The original estimate of UTILITY for completion of the goal was 7; but this was only an *a priori* estimate based on typical experience. The commander appraises that the rate of achieving utility is outweighed by the resources employed, and so calls off the officers from house 2.

This extract from the scenario illustrates the more sophisticated reasoning enabled by and founded on a metric-based notion of partial goal satisfaction that is embedded into a concrete computational framework for the metrics.

7 Conclusion and Next Steps

The contribution of this line of work stems from the recognition of the need for a concept of partial goal satisfaction in cognitive agent frameworks, manifest in terms of the proposal of an abstract framework for partial goal satisfaction that identifies the main necessary ingredients for reasoning based on partial goal satisfaction. Our objective is a representation of partial satisfaction integrated into a reasoning framework, and allowing for a quantitative instantiation, in order that cognitive agent programming frameworks might be enhanced. The benefit of the topic and our approach is more sophisticated reasoning about goals, impacting reasoning about selection, adoption, and pursuit; goal progress appraisal; goal interaction; and inter-agent communication and collaboration.

Although we have indicated how our framework may be concretized in the context of GPTs, more work is needed to flesh out the details and investigate how advanced types of reasoning can be built on top of this basis and integrated into a programming framework. The modifications necessary to the semantics of a language such as GOAL [10] must be established and their correctness proved. To be investigated is how the various functions of our framework can be defined in concrete settings, and how existing work on, e.g., reasoning about resources can be used in this context. Also, while our framework provides the basis for reasoning about goal adaptation, it does not provide algorithms that allow the agent to decide how to adapt, weighing costs and benefits. This is an important area for future research, with just one relevant aspect being how to estimate cost and benefit projection into the future. Lastly, possible extensions are ripe for investigation, such as a logical instantiation with reasoning between goal outcomes, following Zhou et al. [32], inclusion of parameters in goal templates, and relation of templates in a hierarchy.

Acknowledgments

We thank David Martin for discussions and the reviewers for their helpful comments.

References

1. M. E. Bratman, D. J. Israel, and M. E. Pollack. Plans and resource-bounded practical reasoning. *Computational Intelligence*, 14:349–355, 1988.
2. L. Braubach and A. Pokahr. Representing long-term and interest BDI goals. In *Proc. of ProMAS'09*, 2009.
3. B. J. Clement, E. H. Durfee, and A. C. Barrett. Abstract reasoning for planning and coordination. *JAIR*, 28:453–515, 2007.
4. M. Dastani. 2APL: A practical agent programming language. *JAAMAS*, 16(3):214–248, 2008.
5. M. B. Do, J. Benton, M. van den Briel, and S. Kambhampati. Planning with goal utility dependencies. In *Proc. of IJCAI'07*, 2007.
6. P. J. Feltovich, J. M. Bradshaw, W. J. Clancey, M. Johnson, and L. Bunch. Progress appraisal as a challenging element of coordination in human and machine joint activity. In *Proc. of ESAW'07*, 2007.
7. B. Grosz and S. Kraus. Collaborative plans for complex group action. *Artificial Intelligence*, 86(2):269–357, 1996.
8. B. J. Grosz and L. Hunsberger. The dynamics of intention in collaborative activity. *Cognitive Systems Research*, 7(2–3):259–272, 2006.
9. P. Haddawy and S. Hanks. Representations for decision theoretic planning: Utility functions for deadline goals. In *Proc. of KR'92*, 1992.
10. K. V. Hindriks. Programming rational agents in GOAL. In *Multi-Agent Programming: Languages, Tools and Applications*. Springer, Berlin, 2009.
11. K. V. Hindriks, C. Jonker, and W. Pasman. Exploring heuristic action selection in agent programming. In *Proc. of ProMAS'08*, 2008.

12. K. V. Hindriks, W. van der Hoek, and M. B. van Riemsdijk. Agent programming with temporally extended goals. In *Proc. of AAMAS'09*, 2009.
13. R. Holton. Partial belief, partial intention. *Mind*, 117:27–58, 2008.
14. Z. Huang and J. Bell. Dynamic goal hierarchies. In *Proc. of the 1997 AAAI Spring Symp. on Qualitative Preferences in Deliberation and Practical Reasoning*, 1997.
15. E. Kamar, Y. Gal, and B. J. Grosz. Incorporating helpful behavior into collaborative planning. In *Proc. of AAMAS'09*, 2009.
16. G. Klein, D. D. Woods, J. M. Bradshaw, R. R. Hoffman, and P. J. Feltovich. Ten challenges for making automation a "team player" in joint human-agent activity. *IEEE Intelligent Systems*, 19(6):91–95, 2004.
17. V. Lesser and et al. Evolution of the GPGP/TAEMS Domain-Independent Coordination Framework. *JAAMAS*, 9(1):87–143, 2004.
18. D. Morley, K. L. Myers, and N. Yorke-Smith. Continuous refinement of agent resource estimates. In *Proc. of AAMAS'06*, 2006.
19. B. Nebel and J. Koehler. Plan reuse versus plan generation: A theoretical and empirical analysis. *Artificial Intelligence*, 76(1–2):427–454, 1995.
20. A. S. Rao and M. P. Georgeff. Modeling agents within a BDI-architecture. In *Proc. of KR'91*, 1991.
21. M. Schut, M. Wooldridge, and S. Parsons. The theory and practice of intention reconsideration. *JETAI*, 16(4):261–293, 2004.
22. S. Shapiro and G. Brewka. Dynamic interactions between goals and beliefs. In *Proc. of IJCAI'07*, 2007.
23. S. Shapiro, Y. Lespérance, and H. J. Levesque. Goal change. In *Proc. of IJCAI'05*.
24. P. H. Shaw, B. Farwer, and R. H. Bordini. Theoretical and experimental results on the goal-plan tree problem. In *Proc. of AAMAS'08*, 2008.
25. M. P. Singh. A critical examination of use Cohen-Levesque theory of intentions. In *Proc. of ECAI-92*.
26. D. E. Smith. Choosing objectives in over-subscription planning. In *Proc. of ICAPS'04*, 2004.
27. J. Thangarajah, M. Winikoff, L. Padgham, and K. Fischer. Avoiding resource conflicts in intelligent agents. In *Proc. of ECAI-02*, 2002.
28. W. van der Hoek, W. Jamroga, and M. Wooldridge. Towards a theory of intention revision. *Synthese*, 155(2):265–290, 2007.
29. M. B. van Riemsdijk, M. Dastani, and M. Winikoff. Goals in agent systems: A unifying framework. In *Proc. of AAMAS'08*, 2008.
30. M. Winikoff, L. Padgham, J. Harland, and J. Thangarajah. Declarative and procedural goals in intelligent agent systems. In *Proc. of KR'02*, 2002.
31. Y. Zhou and X. Chen. Partial implication semantics for desirable propositions. In *Proc. of KR'04*, 2004.
32. Y. Zhou, L. van der Torre, and Y. Zhang. Partial goal satisfaction and goal change: Weak and strong partial implication, logical properties, complexity. In *Proc. of AAMAS'08*, 2008.

Reinforcement Learning as Heuristic for Action-Rule Preferences

Joost Broekens, Koen Hindriks, Pascal Wiggers

Man-Machine Interaction department (MMI)
Delft University of Technology

Abstract. A common action selection mechanism used in agent-oriented programming is to base action selection on a set of rules. Since rules need not be mutually exclusive, agents are often underspecified. This means that the decision-making of such agents leaves room for multiple choices of actions. Underspecification implies there is potential for improvement or optimalization of the agent's behavior. Such optimalization, however, is not always naturally coded using BDI-like agent concepts. In this paper, we propose an approach to exploit this potential for improvement using reinforcement learning. This approach is based on learning rule priorities to solve the rule-selection problem, and we show that using this approach the behavior of an agent is significantly improved. Key here is the use of a state representation that combines the set of rules of the agent with a domain-independent heuristic based on the number of active goals. Our experiments show that this provides a useful generic base for learning while avoiding the state-explosion problem or overfitting.

Categories and subject descriptors: I.2.5 [**Artificial Intelligence**]: Programming Languages and Software; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*Intelligent Agents*

General terms: Agent programming languages; Robotics; AI; Methodologies and Languages

Keywords: Agent-oriented programming, rule preferences, reinforcement learning

1 Introduction

Agent platforms, whether agent programming languages or architectures, that are rule-based and use rules to generate the actions that an agent performs introduce the problem of how to select rules that generate the most effective choice of action. Such agent programming languages and architectures are based on concepts such as rules, beliefs, and goals to generate agent behavior. Here, rules specify the agent's behavior. A planning or reasoning engine tries to resolve all rules by matching the conditions and actions with the current mental state of the agent. Multiple instantiations of each rule can therefore be possible. An agent can select any one of these instantiations, resulting in a particular action. So,

the rule-selection problem is analogous to but different from the action-selection problem [13]. Rule selection is about which uninstantiated rule to chose; action selection, in the context of rule-based agent frameworks, is about which instantiated rule to chose. In this paper, when we refer to rule we mean uninstantiated rule, i.e, rules that still contain free variables.

Rule-based agent languages or architectures typically underspecify the behavior of an agent, leaving room for multiple choices of actions. The reason is that multiple rules are applicable in a particular situation and, as a result, multiple actions may be selected by the agent to perform next. In practice, it is often hard to specify rule conditions that are mutually exclusive. Moreover, doing so is undesirable as the BDI concepts used to develop agents often are not the most suitable for optimizing agent behavior. An alternative approach is to optimize agent behavior based on learning techniques.

In this paper we address the following question: how to automatically prioritize rules in such a way that the prioritization reflects the utility of a rule given a certain goal. Our aim is a generic approach to learning such preferences, that can be integrated in rule-based agent languages or architectures. The overall goal is to optimize the agent's behavior given a predefined set of rules by an agent programmer, but our approach can also be used by agent programmers to gain insight into the rule preferences and use these to further specify the agent program. As such, we focus on a useful heuristic for rule preferences. We have chosen reinforcement learning (RL) as heuristic as it can cope with delayed rewards and state dependency. These are important aspects in agent behavior as getting to a goal state typically involves a chain of multiple actions, and rules can have different utility depending on the state of the agent/environment.

We present experimental evidence that reinforcement learning can be used to learn rule priorities that can subsequently be used for rule selection. This heuristic for rule priorities works very well, and results in sometimes optimal agent behavior. We demonstrate this with a set of experiments using the Goal agent programming language [5]. Key in our approach is that the RL mechanism uses a state representation based on a combination of the set of rules of the agent and the number of active goals. Our state representation is as abstract as possible while still being a useful base for learning. We take this approach for two main reasons: (1) we aim for a generic learning mechanism; RL should be a useful addition to all programs, and the programmer should not be bothered by the state representation or state-space explosions; (2) an abstract state helps generalization of the learning result as a concrete state representation runs the risk of over fitting on a particular problem instance.

It is important to immediately explain one aspect of our approach that is different from the usual setup for reinforcement learning. In reinforcement learning it is common to learn the *action* that has to be selected from a set of possible actions. In our approach, however, we will apply reinforcement learning to select an *uninstantiated rule* from a set of rules in an agent program. An uninstantiated rule (called *action rule* in Goal, see also Listing 1) is a generic rule defined by the agent programmer.

```
if goal(tower([X|T])), not(bel(T=[])) then move(X,table)
```

is an example of such a rule. We refer to an instantiated rule as a completely resolved (grounded) version of an action rule generated by the reasoning engine responsible for matching rules to the agent's current state. This also means that the action in an instantiated rule may be selected for execution, as the conditions of the rule have been verified by the engine.

```
if goal(tower([a,b])), not(bel([b]=[])) then move(a,table)
```

is an example of an instantiated rule and the action `move(a,table)` is the corresponding action that may be selected. One instantiated rule thus is the equivalent of one action (as it is completely filled in). Many different instantiated rules may be derived from one and the same program rule, depending on the state. An uninstantiated rule is more generic as it defines many possible actions. We focus on learning preferences for uninstantiated rules.

The paper is organized as follows. Section 2 discusses some related work and discusses how our approach differs from earlier work. In Section 3 we briefly introduce the agent language GOAL and use it to illustrate the rule selection problem. Section 4 presents our approach to this problem based on reinforcement learning and presents an extension of GOAL with a reinforcement learning mechanism. In Section 5 experimental results are presented that show the effectiveness of this mechanism. Finally, Section 6 concludes the paper and discusses future work.

2 Related Work

There is almost no work on incorporating learning mechanisms into agent programming. More generally, BDI agents typically lack learning capabilities to modify their behavior [1], although several related approaches do exist.

With regards to related work, several studies attempt to learn rule sets that produce a policy for solving a problem in a particular domain. Key in these approaches is that the rules themselves are learned, or more specific, rule instantiations are generated and evaluated with respect to a utility function. The best performing rule instantiations are kept and result in a policy for the agent. The evaluation mechanism can be different, for example genetic programming [9] or supervised machine learning [7]. In any case, the main difference is that our approach tries to learn rule preferences, i.e., a priority for pre-existing rules given that multiple rules can be active, while the previously mentioned approaches try to learn rule instantiations that solve a problem.

Other studies attempt to learn rule preferences like we do. However, these approaches are based on learning preferences for instantiated rules [10][4], not preferences for the uninstantiated, generic, rules. Further, the state used for learning is often represented in a much more detailed way [10][4]. Finally, as the state representation strongly depends on the environment, the use of learning mechanisms often involves effort and understanding of the programmer [10].

Reinforcement learning has recently been added to cognitive architectures such as Soar [8] and Act-R [2]. In various respects these cognitive architectures are related to agent programming and architectures. They use similar concepts to generate behavior, using mental constructs such as knowledge, beliefs and goals, are also based on an sense-plan-act cycle, and generate behavior using these mental constructs as input for a reasoning- or planning-based interpreter. Most importantly, cognitive architectures typically are rule-based, and therefore also need to solve the rule (and action) selection problem. For example, Soar-RL has been explicitly used to study action selection in the context of RL [6]. Soar-RL [10] is the approach that comes closest to ours in the sense that it uses a similar reinforcement learning mechanism (Sarsa) to learn rule preferences. As explained above, the key difference is that we attempt to learn uninstantiated rule preferences, while Soar-RL learns preferences for instantiated rules [10]. Another key difference is that we use an abstract rule-activity based state representation complemented with a 'goals left to fulfill' counter, as explained in section 4.2.

Finally, [1] present a learning technique based on decision trees to learn the context conditions of plan rules. The focus of their work is to make agents adaptive in order to avoid failures. Learning a context condition refers to learning when to select a particular plan/action, while learning a rule preference refers to attaching a value to a particular plan/action. Our work is thus complementary in the sense that we do not learn context conditions, but instead propose a learning mechanism that is able to guide the rule selection mechanism itself.

3 The Agent Language GOAL

In this Section we briefly present the agent programming language GOAL and use it to illustrate the rule selection problem in agent languages and architectures. For a more extensive discussion of GOAL we refer the reader to [5]. The approach to the rule selection problem introduced in this paper is not specific to GOAL and may be applied to other similar BDI-based platforms. As our approach involves a domain-independent heuristic based on counting the number of goals that need to be achieved, the language GOAL is however particularly suitable to illustrate the approach as declarative goals are a key concept in the language.

GOAL, for Goal-Oriented Agent Language, is a programming language for programming *rational agents*. GOAL agents derive their choice of action from their beliefs and goals. A GOAL agent program consists of five sections: (1) a knowledge section, called the *knowledge base*, (2) a set of beliefs, collectively called the *belief base*, (3) a set of *declarative* goals, called the *goal base*, (4) a *program section* which consists of a set of *action rules*, and (5) an *action specification section* that consists of a specification of the pre- and postconditions of actions of the agent. Listing 1 presents an example GOAL agent that manipulates blocks on a table.

The knowledge, beliefs and goals of a *GOAL* agent are represented using a knowledge representation language. Together, these make up the mental state of

```
main stackBuilder {
 knowledge{
   block(a), block(b), block(c).
   clear(table).
   clear(X) :- block(X), not(on(Y,X)).
   tower([X]) :- on(X,table).
   tower([X,Y|T]) :- on(X,Y), tower([Y|T]).
 }
 beliefs{
   on(a,table), on(b,table), on(c,a), on(d,c).
 }
 goals{
   on(a,d), on(b,c), on(c,table), on(d,b).
 }
 program{
   if goal(tower([X|T])),
     bel((T=[Y|T1], tower(T)); (T=[], Y=table))
     then move(X,Y).
   if goal(tower([X|T])), not(bel(T=[]))
     then move(X,table).
 }
 actionspec{
   move(X,Y) {
     pre{ clear(X), clear(Y), on(X,Z) }
     post{ not(on(X,Z)), on(X,Y) }
   }
 }
}
```

Table 1. Agent for Solving a Blocks World Problem

an agent. Here, we use *Prolog* to represent mental states. An agent's knowledge represents general conceptual and domain knowledge, and does not change. An example is the definition of the concept `tower` in Listing 1. In contrast, the beliefs of an agent represent the *current state of affairs* in the environment of an agent. By performing actions and possibly by events in the environment, the environment changes, and it is up to the agent to make sure its beliefs stay up to date. Finally, the goals of an agent represent *what* the agent wants the environment to be like. For example, the agent of Listing 1 wants to realise a state where block `a` is on top of block `b`. Goals are to be interpreted as *achievement goals*, that is as a goal the agent wants to achieve at some future moment in time and does not believe to be the case yet. This requirement is implemented by imposing a rationality constraint such that any goal in the goal base must not believed to be the case. Upon achieving the *complete* goal, an agent will drop the goal. The agent in Listing 1 will drop the goal `on(a,b), on(b,c), on(c,table)` if this configuration of blocks has been achieved, and only if the *complete* configuration has been achieved.

As GOAL agents derive their choice of action from their knowledge, beliefs and goals, they need a way to inspect their mental state. GOAL agents do so by means of *mental state conditions*. Mental state conditions are Boolean combinations of so-called basic *mental atoms* of the form **bel**(ϕ) or **goal**(ϕ). For example, **bel**`(tower([c,a]))` is a mental state condition which is true in the initial mental state specified in the agent program of Listing 1.

A GOAL agent uses so-called *action rules* to generate possible actions it may select for execution. This provides for a rule-based action selection mechanism, where rules are of the form **if** ψ **then** $a(\boldsymbol{t})$ with ψ a mental state condition and $a(\boldsymbol{t})$ an action. A mental state condition part of an action rule thus determines the states in which the action $a(\boldsymbol{t})$ may be executed. Action rules are located in the program section of a GOAL agent. The first action rule in this section of our example agent generates so-called *constructive moves*, whereas the second rule generates actions to move a *misplaced* block to the table. Informally, the first rule reads as follows: if the agent wants to construct a tower with `X` on top of a tower that has `Y` on top and the agent believes that the tower with `Y` on top already exists, or believes `Y` should be equal to the table, then it may consider moving `X` on top of `Y`; in this case the move would put the block `Y` *in position*, and it will never have to be moved again. The second rule reads as follows: if the agent finds that a block is misplaced, i.e. believes it to be in a position that does not match the (achievement) goal condition, then it may consider moving the block to the table. These rules code a strategy for solving blocks world problems that can be proven to always achieve a goal configuration. As such, they already specify a *correct* strategy for solving blocks world problems. However, they do not necessarily determine a unique choice of action. For example, the agent in Listing 1 may either move block `d` on top of block `b` using the first action rule, or move the same block to the table using the second action rule. In such a case, a GOAL agent will nondeterministically select either of these actions. It is important for our purposes to note here that the choice of rule is at stake here, and not a particular *instantiation* of a rule. Moreover, as in the blocks world it is a good strategy to prefer making constructive moves rather than other types of moves, the behavior of the agent can be improved by preferring the application of the first rule over the second whenever both are applicable. It is exactly this type of preference that we aim to learn automatically.

Finally, to complete our discussion of GOAL agents, actions are specified in the action specification section of such an agent using a STRIPS-like specification. When the preconditions of the action are true, the action is executed and the agent updates its beliefs (and subsequently its goals) based on the postcondition. Details can be found in [5].

As illustrated by our simple example agent for the blocks world, rule-based agent programs or architectures may leave room for applying multiple rules, and, as a consequence, for selecting multiple actions for execution. Rule-based agents thus typically are *underspecified*. Such underspecification is perfectly fine, as long as the agent achieves its goals, but may also indicate there is room for improvement of the agent's behavior (though not necessarily so). The problem of optimizing the behavior of a rule-based agent thus can be summarized as follows, and consists of two components: First, solving a particular task efficiently depends on using the appropriate rule to produce actions (the rule selection problem) and, second, to select one of these actions for execution (the action selection problem). The latter problem is actually identical to selecting an *instantiated* rule where all variables have been grounded, as instantiated rules