Formalizing Complex Task Libraries in Golog

Alfredo Gabaldon[1]

Abstract. We present an approach to building libraries of *tasks* in complex action languages such as Golog, for query answering. Our formalization is based on a situation calculus framework that allows probabilistic, temporal actions. Once a knowledge base is built containing domain knowledge including type information and a library of tasks and the goals they can achieve, we are interested in queries about the achievability of goals. We consider cases where, using domain object type and goal information in the KB, a user is able to get specific answers to a query while leaving some of the specifics for the system to figure out. In some cases where the specifics are missing from the KB, the user is provided with the possible alternative answers that are compatible with the incomplete information in the KB. This approach is being explored in the context of a military operations planning domain for decision support.

1 Introduction

Complex activity planning as required in military operations, the construction industry, software development, etc., is usually carried out by expert hand from scratch or by reuse and adaptation of solutions to similar, previous problems. Computer assistance is used for solving the resulting scheduling problem, usually only after the set of activities necessary to achieve the desired goal has been chosen. The activity planning phase is thus time consuming because it is done by hand, and expensive because it requires domain expertise.

We are interested in building libraries of tasks in the situation calculus based action programming languages Golog [4] and its relatives. One motivation is the possibility of mitigating the time and cost overhead in complex activity planning by developing techniques that allow the reuse of libraries of tasks in an (ideally) automatic way. Although domain experts would still be required during the initial construction of such a library, once a library containing a substantial amount of knowledge in the form of domain-specific, detailed tasks for achieving a number of goals, the cost incurred in this initial phase would be amortized over the many problems solved by reusing that knowledge. In order to achieve this, the library has to be in a form that can be reused by users who are not necessarily domain experts. For example, we would like the library to allow planning for the construction of a building on a soft clay ground, without consulting with an expert on how to lay foundations on such a ground if the library contains tasks for doing that. In this case, it should be enough to simply specify the type of ground, the type of building, and so on, in order to obtain from the library a detailed construction plan. If some details are missing or underspecified, we ultimately would like the library to provide alternative plans, possibly including costs and risks.

Toward this end, we consider formalizing tasks and task execution in a stochastic, temporal version of Reiter's action theories in the situation calculus [6]. A task is defined by means of a set of axioms in the action theory, together with a Golog program. We also allow the user to specify properties (goals) that tasks are capable of bringing about, so that, given a goal, relevant tasks are executed without explicit invocation. This notion is useful for our purposes of designing a general task library that can be utilized by non expert users. Once this framework is in place, we then consider ways of generalizing goal achievement so that more of the burden required in retrieving information is shifted from the user to the knowledge base that contains the library. In particular, we consider the use of *generalized goals* so that tasks that achieve a very specific goal is also attempted on generalized versions of the specific goal. We also consider a form of generalized query based on *compatible types* of objects so that useful answers are obtained even in the case of incomplete information about domain objects' types. We have implemented this framework in Prolog building on the interpreters of the action programming languages mentioned above. The full version of this paper [3] contains a detailed description of this framework and sample runs obtained from the implementation.

2 Background

As a practical footing for our work, we consider a military operations planning problem described in [1]. Roughly, a problem instance in this domain consists of a description of a set of *tasks*, e.g. "anti-submarine operation", which are similar but more complex than actions in classical planning, together with a description of the initial state, including numerical quantities of available resources. A task description includes a duration, multiple possible outcomes with corresponding probabilities, resource requirements, and other preconditions. Also included are statements about goals that tasks can achieve, which are used to invoke relevant tasks as required by a goal. Moreover, some tasks require subgoals to hold prior or during their execution. These last two characteristics of tasks in this application resemble how Hierarchical Task Networks (HTNs) [2] and complex action languages like Golog operate. We appeal to the latter in this work, since these languages are based on a formal, logic foundation that allows stochastic, temporal actions, and is first-order [6].

One of the motivations for building a library of tasks is for using it in decision support. We are thus interested in query answering with respect to the library. For instance, we'd like to be able to issue queries such as: what is the prob. of achieving control of sea region $r2$?, and obtaining answers such as: A1: prob=0.8, executcing tasks: aswWsubs,... A2: prob=0.75, executing tasks: aswWfrigates,... corresponding to two available ways of achieving the goal in the query.

3 Executing Tasks and Achieving Goals

As part of the specification of a task as described above, a "body" is defined in the form of a Golog program δ to be executed when the

[1] National ICT Australia and U. of New South Wales, Sydney, Australia, alfredo@cse.unsw.edu.au

task is invoked. A task is executed by 1) checking the arguments are of the right types, 2) substituting the arguments in the body of the task, and 3) executing the body.We formalize the execution of a task as the following Golog procedure:

```
**proc** $execTsk(tsk, \vec{x}, t)$
    $typeCheck(tsk, \vec{x})?$ ;
    $(\exists\delta).task(tsk, \vec{x}, t, \delta)?$ ;
    $\delta$
**endProc**
```

The definition of a task also includes declarations of what goals it can achieve. This allows requests that a goal be achieved without referring to a particular task. This is important for our purpose of designing libraries that can be used by non experts once the background knowledge has been stored in the libraries. For making such requests of achieving a goal, we define a procedure $achieve(goal, \vec{x}, t)$. In general there may be multiple tasks capable of achieving the same goal. For instance, the two tasks mentioned above, $aswWsubs$ and $aswWfrigates$, represent two different ways of establishing control of a region of type "sea sub-surface". Thus the $achieve$ procedure is defined so that it non-deterministically chooses a task from among those that can achieve the goal, and executes it.

```
**proc** $achieve(goal, \vec{x}, t)$
  $goal(\vec{x})?$ |
  $(\pi\ tsk)[achievesG(tsk, \vec{x}, goal)?$ ;
          $execTsk(tsk, \vec{x}, t)]$
**endProc**
```

Notice that one of the choices for achieving the goal is to do nothing other than successfully test that the goal is already true. $(\pi\ x)$ means choose an x, and $\delta_1|\delta_2$ means choose between executing δ_1 and δ_2.

4 Generalized goals and achieve

A successful execution of a task, as defined above, requires that the parameters be of the correct type for the task. For instance, executing task $aswWsubs$ on a region reg will only be successful if the knowledge base contains the fact that reg is of type $seaSubSurf$. Since we are interested in query answering we consider generalizing our framework so that goals, and goal achievement can be organized into a hierarchy based on specificity.

Consider a scenario where a user issuing queries is interested in control of region reg, which he knows to be a sea region, but not whether it is a sea surface or a sea sub-surface. Suppose that the user does not know, but the info about the specific type of reg is in fact stored in the system. In such a situation, we would nevertheless like to be able to check if the goal $seaRegionCtrlEst$ can be achieved for reg. The library should be able to invoke an appropriate task for reg based on its type, even if the user is not aware of it. However, we need to endow the system with the ability to figure out that, for example, if the more specific goal $seaSubSurfCtrlEst$ is achieved, this implies that the more general goal $seaRegionCtrlEst$ has also been achieved. The fact that one goal is a generalized version of another cannot be derived from type information alone as it also depends on the nature of the goals, which in our case are all about establishing control of a region. Thus we will rely on the knowledge engineer to provide additional information by means of statements of the form: $generalizedG(generalGoal,\ specificGoal)$. Since tasks that achieve a specific goal also achieve the less specific version of the goal, we can then define a generalization of $achievesG$ capturing this relationship between goals by means of these statements and axiom:

$$achievesG(tsk, \vec{x}, g) \equiv achievesG_0(tsk, \vec{x}, g) \vee \\ generalizedG(g, g') \wedge achievesG_0(tsk, \vec{x}, g')$$

This results in the invocation of specific tasks, if necessary and available, for attempting to achieve a generalized goal. Type information plays a critical role in this.

Consider again the scenario mentioned above, but this time let us assume that the KB does not contain any info on whether reg is of type $seaSurf$ or $seaSubSurf$. In this case, none of the specific tasks will execute since their type-check tests fail. Nevertheless, if the system knows reg is a sea region, then for a query whether control of reg can be achieved, we would like to obtain an answer that is more useful than "don't know". For instance, if a sea region must be either a $seaSurf$ or a $seaSubSurf$, the system could give answers corresponding to these alternatives. For this, we introduce a test to check if the arguments passed to a task are "compatible" with the types required, where "compatible" is defined in terms of possible sub-types. Then we introduce execution of compatible tasks as a procedure $execCompTsk(tsk, \vec{x}, t)$ similar to $execTsk(tsk, \vec{x}, t)$, but with a type-compatibility test instead of the strict type-check. Finally, we modify procedure $achieve$ so that it tries compatible tasks when no specific task can execute:

```
**proc** $achieve(goal, \vec{x}, t)$
  $goal(\vec{x})?$ |
  $(\pi\ tsk)[achievesG(tsk, \vec{x}, goal)?$ ; $execTsk(tsk, \vec{x}, t)]$ |
  $[(\forall\ tsk')(achievesG(tsk', \vec{x}, goal) \supset \neg typeCheck(tsk', \vec{x}))?$ ;
     $(\pi\ tsk)[achievesG(tsk, \vec{x}, goal)?$ ;
          $execCompTsk(tsk, \vec{x}, t)]]$
**endProc**
```

5 Conclusions

In this paper we present preliminary ideas on building a knowledge base containing a library of tasks for query answering. We have shown an encoding of stochastic, temporal tasks, and their execution, as complex actions in programming language Golog and its relatives. We consider how such a library can be used together with information about a domain's object types to handle queries about the achievement of goals. We have also considered how to make the knowledge base more robust by allowing users to obtain useful answers even in cases where queries lack some degree of specificity, and where the knowledge base lacks detailed info about the types of certain domain objects. Much work remains to be done, including a proof of correctness of our implementation and extending the system to handle a larger class of queries.

REFERENCES

[1] Douglas Aberdeen, Sylvie Thiébaux, and Lin Zhang, 'Decision-theoretic military operations planning', in *Procs. of ICAPS'04*, (2004).

[2] Kutluhan Erol, James A. Hendler, and Dana S. Nau, 'Complexity results for hierarchical task-network planning', *Annals of Mathematics and Artificial Intelligence*, **18**, 69–93, (1996).

[3] Alfredo Gabaldon, 'Encoding Task Libraries in Complex Action Languages', Avail. at http://cse.unsw.edu.au/∼alfredo/papers.html (2006).

[4] H. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. Scherl, 'Golog: A logic programming language for dynamic domains', *Journal of Logic Programming*, **31**(1–3), 59–83, (1997).

[5] R. Reiter, 'The frame problem in the situation calculus: A simple solution (sometimes) and a completeness result for goal regression', in *Artificial Intelligence and Mathematical Theory of Computation*, ed., V. Lifschitz, 359–380, Academic Press, (1991).

[6] Ray Reiter, *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems*, MIT Press, (2001).