The Golog Programming Language and Agency

Sergio Alejandro Gómez

Laboratorio de Investigación y Desarrollo en Inteligencia Artificial
Departamento de Ciencias e Ingeniería de la Computación
Universidad Nacional del Sur
Av. Alem 1253 – B8000CPB Bahía Blanca – República Argentina
Tel/Fax: (+54) (291) 459 5135/5136 – Email: sag@cs.uns.edu.ar

Keywords: Situation calculus, Golog programming language, software agents.

Abstract

In this paper, we present our initial experiences modeling agency in the situation calculus and the Golog programming language. We describe the problems we have faced and discuss some research issues that remain to be addressed in the future.

As an application of this research line, we propose an axiomatization of the robot Charles and the Fantastic City in the situation calculus as well as a controller written in Golog.

1 Introduction and motivations

Software development requires of methods for writing correct programs. Historically, several approaches have implemented this view, for instance Hoare's logics in concurrent systems [And91], functional programming [BW88], transformational programming [BD77], algebraic programming [BdM97], fixed-point semantics in logic programming [Llo87], for mentioning just a few. Furthermore, it requires of systems with the ability of explaining the results obtained. Thus, the problem specification should be embedded in the application as opposed to the traditional programming paradigm where the code only abstracts the underlying hardware architecture and the model of the problem domain exists only in the developer and end-user minds.

The *situation calculus* —a first-order language (with some second-order features) specifically designed for representing dynamically changing worlds— provides a framework for doing this by means of the *Golog* programming language [LPR98, LRL+94]. Golog and the situation calculus appear well suited for applications in high level control of robots and industrial processes, intelligent software agents, discrete event simulation, etc. Golog provides macros for representing primitive actions, test actions, sequence, nondeterministic choice of two actions, nondeterministic choice of action arguments and nondeterministic iteration.

In this paper we describe our brief experience modeling discrete agent simulations in the Golog programming language. The paper is structured as follows. First, we outline the problems we have faced and some research issues. Second, we describe a prototypical implementation in Golog of the robot Charles. Finally, we summarize the work and look at where this research can be aimed for in the future.

2 Problems faced and research issues

Reiter et al. explain the possible uses of the Golog programming language [LRL+94]. The Golog interpreter automatically maintains an explicit representation of the dynamic world

being modeled, on the basis of user supplied axioms about the preconditions and effects of actions and the initial state of the world. This allows programs to reason about the state of the world and consider the effects of various possible courses of action before committing to a particular behavior. The net effect is that programs may be written at a much higher level of abstraction than is usually possible. The language appears well suited for applications in high level control of robots and industrial processes, intelligent software agents, discrete event simulation, etc. It is based on a formal theory of action specified in an extended version of the situation calculus. A prototype implementation in Prolog has been developed by Reiter et al. as well as a Prolog flavor called *Eclipse Prolog*.

The simulation of an agent in Golog implies facing the following problems:

- **Choosing a suitable repertoire for modeling the agent's primitive actions**: All changes to the world are the result of named actions. This actions have unique names and can be parameterized.

- **Choosing a suitable set of fluents**: Relations whose truth value vary from situation to situation, called relational fluents, are denoted by predicate symbols taking a situation term as their last argument. Functions whose denotations vary from situation to situation are called functional fluents.

- **Specifying primitive action preconditions**: Actions have preconditions, that is, necessary and sufficient conditions that characterize when the action is physically possible.

- **Specifying successor state axioms for primitive fluents**: A possible world history, which is a sequence of actions, is represented by a first order term called a *situation*. There is a distinguished binary function symbol do; $do(\alpha, s)$ denotes the successor situation to s resulting from performing the action α. World dynamics are specified by *effect axioms*. These describe the effects of a given action on the fluents, i.e. the casual laws of the domain. Besides the so-called *frame actions* are also necessary, these specify the action *invariants* of the domain, namely, those fluents which remain unaffected by a given action.

  If the axiomatization is done carelessly, the number of needed axioms grows quadratically respect to the number of primitive actions. So we solve the frame problem as suggested by Reiter et al. in [LRL$^+$94]. The result is a set of situation calculus formulae which are maybe hard to read but whose variable bindings can be efficiently computed.

- **Defining Golog procedures for ultimately controlling the agent**: Golog provides a formalism for expressing primitive actions, test actions, sequence, nondeterministic choice of two actions, nondeterministic choice of action arguments, nondeterministic iteration, and procedure call; all of this facilities macro expand to situation calculus formulae.

- **Finding formal proofs of the correctness of the proposed implementation**: Due to the fact that Golog is based on the situation calculus, which in turn is an extension of first-order logic (with some second-order features), this task implies proving logical entailments.

- **Finding plans for solving problems associated to the agent environment**: This is done automatically by the Golog interpreter. Running the program implies proving

the following entailment [LRL+94]:

$$Axioms \models (\exists s) Do(\Pi; application1, S_0, s). \quad (1)$$

where $Axioms$ are the foundational axioms of the situation calculus, S_0 is a constant regarding as the initial situation (that is, the situation where no actions have yet occurred), and Π the sequence of procedure declarations describing the agent controller.

A successful execution of the program, i.e. a successful proof of Eq. 1, will result in a binding for the final situation s. This final situation s will have as its value the plan for achieving the desired goal and thus finding a plan for the problem.

The version of Golog we are currently using omits some important considerations mentioned in the literature [LRL+94]:

- **Sensing and knowledge**: When modeling an autonomous agent, it is necessary to consider the agent's perceptual actions, e.g. acts of seeing, hearing, etc. Unlike ordinary actions that affect the environment, perceptual actions affect an agent's mental state, i.e. its state of knowledge.
- **Sensing and knowing how**: In the presence of sensing actions, the method we are currently using for executing Golog program is no longer adequate.
- **Exogenous actions**: We assume that all events of importance are under the agent control. Actually, exogenous actions can occur at any time and are not in control of the agent. One example of exogenous actions are actions under nature control (like starting to rain) or other agents's behaviours. In our simulations, we do not include exogenous actions as part of the program, because the agent is in no position to cause such actions to happen.
- **Concurrency and reactivity**: Once that exogenous events are allowed, it becomes very useful to write programs which monitor certain conditions, and take appropriate actions when they become true. It is desirable that a number of complex actions of this sort can be executed concurrently. This form of concurrency allows a much more natural specification of controllers that need to quickly react to their environment while following predetermined plans. The modeling of these issues are addressed in an extension to Golog named *ConGolog* [DGLL00] that we plan to use in the future.

3 The robot Charles: a case study

Charles is a robot methafor used in the University of La Plata to teach elementary programming [DGMB+98, P. 6 and Appendix A]. The robot Charles walks in the Fantastic City which is a rectangular environment made up of avenues (rows) and streets (columns). A corner is the intersection of an avenue and a street. Charles has basic capabilities, such as moving, turning to right, possessing visual sensors for both acknowledging obstacles on its way and identifying some specified simple form, such as candies, flowers, and papers (that can be picked one at a time only in corners and stored in a bag).

We have proposed an axiomatization of the robot Charles and the Fantastic City in the situation calculus and have also codified a controller for it in the Golog programming language. Thus we have given a formal specification of how the robot actions affect its environment and it is used to obtain a plan for solving problems associated to the robot domain. These results will be presented in subsequent publications.

4 Work in progress

We have presented an overview of our research concerning the application of the Golog programming language for modeling agency. In particular we propose the use of Golog for modeling the robot Charles and its environment (the Fantastic City) and solving problems associated to it. This initial work addresses several issues in the area of cognitive robotics. This issues will be presented in subsequent publications. While there are some issues that have not been addressed, they are actively pursued and are part of our current research work.

References

[And91] Andrews, G. R. *Concurrent Programming: Principles and Practice.* Benjamin-Cummings Publishing Co, Inc. Redwood City, CA, USA, 1991.

[BD77] Burstall, R. M., and Darlington, J. A transformation system for developing recursive programs. *Journal of the ACM (JACM), Vol. 24 No. 1* (Jan. 1977), Pp. 44–67.

[BdM97] Bird, R., and de Moor, O. *Algebra of Programming.* Prentice Hall, 1997.

[BW88] Bird, R. S., and Wadler, P. *Functional Programming.* Prentice-Hall, 1988.

[DGLL00] De Giacomo, G., Lespérance, Y., and Levesque, H. ConGolog, a concurrent programming language based on the situation calculus. *Artificial Intelligence 121*, 1–2 (2000), 109–169.

[DGMB+98] De Giusti, A., Madoz, M. C., Bertone, R., Naiouf, M., Lanzarini, L., Gorga, G., and Russo, C. *Algoritmos, Datos y Programas. Conceptos Básicos.* Editorial Exacta, 1998.

[Llo87] Lloyd, J. *Foundations of Logic Programming.* Springer-Verlag, 1987.

[LPR98] Levesque, H., Pirri, F., and Reiter, R. Foundations for the situation calculus. *Linköping Electronic Articles in Computer and Information Science, Vol. 3 (1998): nr 18. December 22, 1998.* (1998).

[LRL+94] Levesque, H. J., Reiter, R., Lespérance, Y., Fangzhen, L., and Scherl, R. B. Golog: A logic programming language for dynamic domains. *J. Logic Programming* (1994). Available at http://www.stud.uni-hannover.de/~gerasch/download/GOLOGlang.pdf.