CALM: a Compiler for Modular Action Language ALM

Edward Wertz
Texas Tech University

Abstract

CALM is the first compiler for the modular action language ALM. ALM supports the modeling of larger dynamic domain problems through the use of modules, module hierarchy, and libraries. CALM translates an ALM system description P into a SPARC (an Answer Set Programming language) program which specifies the same state transition diagram as P. CALM also supports language for specifying temporal projection and planning problems. Given an ALM system description and a temporal projection or planning problem, CALM will translate them into a SPARC program whose answer sets contain solutions to these problems.

Introduction

A dynamic domain can be viewed as a state transition diagram whose nodes are possible states of the domain and whose arcs are actions in the domain. Action languages have been developed to effectively specify the diagram, but are restricted to small or medium sized domains (Gelfond and Kahl 2014). ALM is a recently developed modular action language that addresses larger domains (Inclezan and Gelfond 2016). It supports modeling by modules, module hierarchy and library.

CALM is the first compiler for ALM. It translates an ALM system description P into a SPARC (an Answer Set Programming language (Balai, Gelfond, and Zhang 2013)) program which specifies the same state transition diagram as P. It also supports language for specifying temporal projection and planning problems. Given an ALM system description and a temporal projection or planning problem, CALM will translate them into a SPARC program whose answer sets contain solutions to these problems.

In this presentation we first review the modeling capability of ALM. We then discuss how CALM translates ALM system descriptions to SPARC and how CALM can be used to specify temporal projection and planning problems. Finally we discuss the significance of CALM and future work for the compiler.

ALM Preliminaries

An ALM *system description* is composed of two parts, a *theory* and a *structure*. The *theory* contains a hierarchy of *modules* which specifies a *basic action theory* (BAT). The *structure* provides an interpretation of the sorts and static functions in the BAT. The structure and the BAT together define a state transition diagram.

The BAT specified by an ALM program has two parts, a *sorted signature* and a collection of *axioms*. The axioms are composed of *function definitions*, *state constraints*, *executability conditions* and *dynamic causal laws* of actions. The *sorted signature* is a 4-tuple $\langle C, O, H, F \rangle$ where C is a set of sort names, O is a set of object constant names, F is a set of function signatures $f : c_1, \ldots, c_n \rightarrow c_0$ where f is the name of the function and $c_0, \ldots, c_n \in C$, and H is a directed acyclic graph over elements in $C \cup O$.

Within H, if there is an arc from sort s_1 to sort s_2, then s_1 is called a *subsort* of s_2. The *subsort* relation is transitive in H. If there is an arc from object constant o to sort s, then o is called an *instance of* s and s is called the *sort of* o. There are no arcs that target object constants in H. If o is an instance of s_1 and s_1 is a subsort of s_2 then o is an instance of s_2. H contains two special sorts *universe* and *actions* such that *universe* is the only sink in H and there is an arc from *actions* to *universe*. All user defined actions must be instances of sorts that are subsorts of *actions*.

The BAT of an ALM program is defined by the modules within the *theory*. A module's *sort declarations* section adds sort names to C, defines arcs in H, and adds function signatures to F for the attribute functions of sorts. The *constant declarations* section adds object constant names to O and arcs in H. The *function declaration* section adds signatures to F. Functions whose range value changes over time are declared as *fluents*. Functions whose value does not change are declared as *statics*. Each module also provides *axioms* related to the locally declared functions. If module M_1 is dependent on module M_2 then the sorts of M_1 may be declared as subsorts of the sorts in M_2.

There are two notions of inheritance in ALM. If s_1 is a subsort of s_2 then s_1 inherits the attribute functions of s_2 and they are defined on the instances of s_1. If module M_1 is dependent on module M_2 then M_1 inherits the sorts, constants, functions and axioms of M_2. Through module inheritance, an ALM system description can be *flattened* into a single module that defines the BAT.

CALM - The ALM Compiler

When defining the semantics of an ALM program, the original ALM paper (Inclezan and Gelfond 2016) translates ALM to ASP{f} (Balduccini and Gelfond 2013), a variation of ASP allowing the use of functions in addition to relations. In our compiler, we chose to target SPARC for its support of sorted signatures. Since SPARC does not support functions directly, we employ predicates to represent them.

Before CALM flattens a multi-module system description into a single module, it checks for semantic errors and type checks the axioms following module dependency. If a module declares s_1 to be a subsort of s_2, but s_2 is not declared locally and not inherited through module dependency, a semantic error is produced. Similarly all sorts used in function signatures must exist in the sort hierarchy defined by the modules. For each axiom, the variables are assigned sorts that are inferred from their use in functions. If a variable occurs multiple times in the same axiom, the assigned sort is restricted to the greatest common subsort of all the inferred sorts. If no common subsort exists, an error is recorded.

If no syntax, semantic or type errors exist in the ALM Program, CALM translates the BAT and the *structure* to an output SPARC program with 3 sections: *sorts*, *predicates*, and *rules*. Since instances in the structure may have conditions placed on their definition, we use an intermediate SPARC program to calculate the exact instances of each sort. From the answer set of the sort-calculating program, we enumerate the instances of each sort in the *sorts* section of the output SPARC program. The *predicates* section contains the sorted signatures of the predicates used to model functions. The *rules* section of the output SPARC program contains the rules modeling the sort hierarchy and axioms of BAT and auxiliary rules needed for modeling functions as predicates.

A *temporal projection* problem is specified by providing a *history* of action occurrences and observed fluent values, including the values of fluents at the initial time step 0. A *planning problem* extends a *temporal projection* by adding a goal state to be achieved. CALM uses SPARC CR rules to generate actions required to reach the goal state. If CALM is given a temporal projection or planning problem to supplement the ALM system description, it will translate them to the SPARC program as discussed above and use the SPARC solver to find answer sets for the program. Each answer set contains a solution to the given problem.

Significance of CALM

Before CALM, ALM programs were manually translated to SPARC or other ASP languages. By automating the translation process, we free the domain modeler to focus on the development of ALM modules and libraries. CALM's ability to apply temporal projection and planning problems to the transition diagram allows the modeler to test and verify the correctness of their modules. CALM's syntax, semantic, and type checking capability help catch errors early instead of debugging complex ASP programs.

Now that CALM exists, programmers and researchers are able to use ALM as a target for reasoning about dynamic domains. One application of CALM would be as a component of a natural language question answering system. After parsing the text and identifying subject domains, the NLP system would build an ALM system description by including the modules related to each subject in the text. From the questions being asked, the NLP system would add a temporal projection or planning problem to derive the answers.

As for related work, another modular action language is MAD (Lifschitz and Ren 2006; Erdoğan 2008). A MAD compiler can translate a MAD program into a program in the language of the Causal Calculator (CCalc) such that CCalc can be used to carry out reasoning tasks.

Future Work

We are testing the CALM compiler on increasingly complex dynamic domains. We hope this presentation inspires collaboration to use and further develop CALM.

One priority for us is to extend CALM to support aggregates, which is already supported by SPARC. It is currently awkward to express state constraints related to aggregation (e.g. no more than two toasts should be cooked in the pan in the French Toast Problem (Lifschitz 2015)).

Acknowledgments

This presentation is based on a paper of the same title submitted to NMR-2018, co-authored by Edward Wertz, Anuradha Chandrasekaran and Yuanlin Zhang of Texas Tech University. We are grateful to Michael Gelfond for numerous discussions on ALM. We also thank Sonia Baee, Justin Lugo, Alexander Meyer, Christian Reotutar, Jason Yee, Levi Brown and Elias Marcopulos for their contribution to this project. We thank Daniela Inclezan for providing us ALM programs.

References

Balai, E.; Gelfond, M.; and Zhang, Y. 2013. Towards answer set programming with sorts. In *Logic Programming and Nonmonotonic Reasoning, 12th International Conference, LPNMR 2013, Corunna, Spain, September 15-19, 2013. Proceedings*, 135–147.

Balduccini, M., and Gelfond, M. 2013. Language asp {f} with arithmetic expressions and consistency-restoring rules. *arXiv preprint arXiv:1301.1387*.

Erdoğan, S. T. 2008. A library of general-purpose action descriptions.

Gelfond, M., and Kahl, Y. 2014. *Knowledge Representation, Reasoning, and the Design of Intelligent Agents*. Cambridge University Press.

Inclezan, D., and Gelfond, M. 2016. Modular action language $\mathscr{ALM}$. *Theory and Practice of Logic Programming* 16(2):189–235.

Lifschitz, V., and Ren, W. 2006. Toward a modular action description language. AAAI 2006 Spring Symposium Series. to appear.

Lifschitz, V. 2015. French toast discussion at tag, https://www.cs.utexas.edu/users/vl/tag/discussions.html, retrieved in August 2018.