A Component Concept for Abstract State Machines

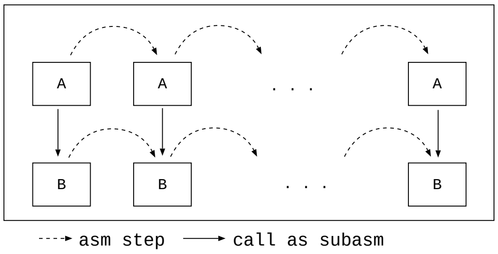
Matthias Anlauff

GMD FIRST, Kekuléstr. 7
D-12489 Berlin, Germany
ma@first.gmd.de

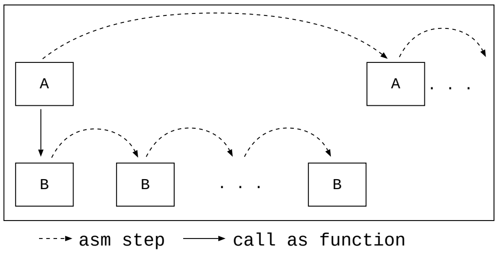
Abstract. The Abstract State Machine (ASM) [9] approach has already proven to be suitable for large-scale specifications of realistic systems [7, 6, 11, 13]. Due to the fact that the ASM approach defines a notion of *executing* specifications, it provides a perfect basis for a language, which can be used as a specification language as well as a high-level programming language. However, in order to upgrade to a realistic programming language, such a language must – besides other features – add a modularization concept to the core ASM constructs in order to provide the possibility to structure large-scale ASM-formalizations and to flexibly define reusable specification units. The language XASM[1] – which stands for *Extensible ASM* – realizes a component-based modularization concept based on the notion of external functions as defined in ASMs. In the presentation, the component-based modularization concept of XASM will be briefly described.

XASM formalizations are structured using "**asm**. . . **endasm**" units each of which can be seen as a component containing a certain part of an overall ASM specification. In order to correctly integrate the component concept into the ASM framework, there exists two kinds of relationships between these units; these relationships will be refered to as "use modes" in the sequel. An **asm** can be accessed by other **asms** in either of the following two ways:

If an asm A uses B as **sub-asm**, it means that B – possibly together with arguments, if the arity of $B > 0$ – is used as a *rule* in the body of A. If this rule fires, the rules of **asm** B fire, which may result in updating locations of functions declared in A. The sub-asm-use relation between **asms** may contain cycles; lazy evaluation techniques are used to avoid an infinite number of rules. The call as subasm is illustrated in the figure. The sub-asm B and its parent asm A step simultaneously; formally they can be seen as one single ASM.



Asm A uses B as a **function**, if B is defined as **external function** in A. In this case, B – possibly together with arguments, if the arity of $B > 0$ – is used as a *term* in the body of A. Recursion is allowed, so that the function-use relation between **asms** may contain cycles. The call as function is illustrated in the figure. During the run of the function-asm B, its parent A doesn't make any step; from A's point of view B's run happens in zero time. As depicted in the figure,



B behaves like a "normal" asm, the iterations shown here are caused by the steps of the B-asm itself.

In each of the above cases, we call A the *parent-asm* of B, if A uses B as sub-asm or as function. In any case, the **asm** must be declared in the parent **asm**. As part of its meta information, an **asm** can be marked as a function or as a sub-asm, so that it can only be used by other **asms** in the specified way.

The XASM language is fully implemented. The system is used as the basis for the Montages/Gem-Mex, where generated XASM code is translated into an interpreter for the language specified using Montages, a semi-visual method for specifying the syntax and semantics of programming languages [2–5, 12, 8, 10].

References

1. M. Anlauff. XASM – An Extensible, Component-Based Abstract State Machines Language. In Y. Gurevich, P. Kutter, M. Odersky, and L. Thiele, editors, *Proceedings of the International Workshop on Abstract State Machines (ASM2000)*, LNCS. Springer, 2000. to appear.
2. M. Anlauff, P. Kutter, and A. Pierantonio. Formal Aspects of and Development Environments for Montages. In M. Sellink, editor, *2nd International Workshop on the Theory and Practice of Algebraic Specifications*, Workshops in Computing, Amsterdam, 1997. Springer.
3. M. Anlauff, P. Kutter, and A. Pierantonio. Enhanced control flow graphs in Montages. In A. D.Bjoerner, M.Broy, editor, *Perspective of System Informatics*, LNCS, 1999. to appear.
4. M. Anlauff, P. Kutter, and A. Pierantonio. Tool Support for Language Design and Prototyping with Montages. In *Proceedings of Compiler Construction (CC'99)*, Lecture Notes in Computer Science. Springer, 1999.
5. M. Anlauff, P. Kutter, A. Pierantonio, and L. Thiele. Generating an Action Notation Environment from Montages Descriptions. In P. Mosses and D. Watt, editors, *Proceedings of the 2nd International Workshop on Action Semantics (AS'99)*, number NS-99-3 in BRICS Notes Series, pages 1–42. University of Aarhus, Department of Computer Science, Mar. 1999.
6. E. Börger and W. Schulte. A Modular Design for the Java VM architecture. In E. Börger, editor, *Architecture Design and Validation Methods*. Springer, 1998.
7. E. Börger and W. Schulte. Programmer Friendly Modular Definition of the Semantics of Java. In J. Alves-Foss, editor, *Formal Syntax and Semantics of Java*, LNCS. Springer, 1998.
8. G.Goos, A. Heberle, W. Löwe, and W. Zimmermann. On modular definitions and implementations of programming languages using order-sorted partial abstract state machines. In *Proceedings of the ASM2000 Conference*, 2000.
9. Y. Gurevich. Evolving Algebras 1993: Lipari Guide. In E. Börger, editor, *Specification and Validation Methods*, pages 9–36. Oxford University Press, 1995.
10. J. Huggins and W. Shen. The static and dynamic semantics of C. In *Proceedings of the ASM2000 Conference*, 2000.
11. J. K. Huggins and W. Shen. The static and dynamic semantics of C: Preliminary version. Technical Report Technical Report CPSC-1999-1, Computer Science Program, Kettering University, 1999.
12. P. Kutter and A. Pierantonio. Montages: Specifications of Realistic Programming Languages. *Journal of Universal Computer Science*, 3(5):416–442, 1997.
13. P. Kutter and A. Pierantonio. The Formal Specification of Oberon. *Journal of Universal Computer Science*, 3(5):443–503, 1997.