

ASD: The Action Semantic Description Tools

Arie van Deursen¹ and Peter D. Mosses²

¹ CWI, P. O. Box 94079, 1090 GB Amsterdam, The Netherlands, arie@cwi.nl

² BRICS (Basic Research in Computer Science, Centre of the Danish National Research Foundation), Computer Science Department, Aarhus University, Ny Munkegade Bldg. 540, DK-8000 Aarhus C, Denmark, pdmosses@brics.dk

Introduction *Action Semantics* is a framework for describing the semantics of programming languages [6, 9]. One of the main advantages of Action Semantics over other frameworks is that it scales up smoothly to the description of larger practical languages, such as Standard Pascal [7]. An increasing number of researchers and practitioners are starting to use action semantics in preference to other frameworks, e.g., [8].

The ASD tools include facilities for parsing, syntax-directed (and textual) editing, checking, and interpretation of action semantic descriptions. Such facilities significantly enhance accuracy and productivity when writing large specifications, and are also particularly useful for students learning about the framework. The notation supported by the ASD tools is a direct ASCII representation of the standard notation used for action semantic descriptions in the literature, as defined in [6, Appendices B–F].

Action Semantic Descriptions The notation used in action semantic descriptions can be divided into four kinds:

Meta-Notation, used for introducing and specifying the other notations;

Action Notation, a fixed notation used for expressing so-called *actions*, which represent the semantics of programming constructs;

Data Notation, a fixed notation used for expressing the data processed by actions; and

Specific Notation, introduced in particular action semantic descriptions to specify the abstract syntax of the programming language, the semantic functions that map abstract syntax to semantic entities, and the semantic entities themselves (extending the fixed action and data notation with new sorts and operations).

Compared with conventional frameworks for algebraic specification, the Meta-Notation is unusual in that it allows operations on sorts, not only on individual values. Its foundations are given by the framework of Unified Algebras [5]. Moreover, so-called *mix-fix* notation for operations is allowed, thus there is no fixed grammar for terms. This is a crucial feature, because action notation includes many infix combinators (e.g., A_1 and then A_2 , which expresses sequencing of the actions A_1, A_2) and mix-fix primitive actions (e.g., bind I to D). The specific notation introduced by users tends to follow the same style.

The Meta-Notation's features include constructs for modularization, introduction of symbols, expressing functionalities, equations between sorts or individuals, and the definition of abstract syntax. These constructs can be given in arbitrary order; one module can both introduce mix-fix symbols and use them in equations.

The Platform The ASD tools are implemented using the ASF+SDF system [1, 4, 3]. In the ASF+SDF approach to tool generation, the syntax of a language is described using the Syntax Definition Formalism SDF, which defines context-free syntax and signature at the same time. Functions operating on terms over such a signature are defined using (conditional) equations in the algebraic specification formalism ASF. Typical functions describe type checking, interpreting, compiling, etc., of programs. These functions are executed by interpreting the algebraic specifications as term rewriting systems. Moreover, from SDF definitions, parsers can be generated, which in turn are used for the generation of syntax-directed editors. ASF+SDF modules allow hiding and mutual dependence. (The demonstration will start by explaining the basic features of ASF+SDF).

The ASF+SDF system currently runs on, e.g., Sparc (Solaris or SunOS) and Silicon Graphics workstations, and uses X-Windows. It is based on the Centaur system (developed by, amongst others, INRIA). Once one has installed the ASF+SDF system, all that is needed before using the ASD tools is to get a copy of the ASF+SDF modules implementing ASD and the user guide, together with a configuration file that specifies the effects of the various buttons in the ASD interface; these items are freely available by FTP.

The Implementation ASD modules written in the Meta-Notation are translated to ASF+SDF modules, using the ASF+SDF system itself. Concerning the unusual features of the Meta-Notation: The arbitrary mix-fix operations are catered for by a two-phase generation scheme. First, a very basic grammar (essentially recognizing key-words, lists of tokens, and well-balanced brackets) is used to parse Meta-Notation modules. Using this first parse, the Meta-Notation constructs introducing symbols, functionalities, and grammars are further processed and translated to ASF+SDF modules. These generated ASF+SDF modules are then used to analyse the complete ASD module again. Sort operations in the Meta-Notation are dealt with by generating (in some cases) extra sorts in the ASF+SDF module.

Main Features The main features of ASD are centered around a syntax-directed editor containing an ASD module. A user writing an ASD module can use the buttons in this editor to check this module or test the grammar and semantic functions contained in it. The features include:

SYMBOL Checking: By starting the SYMBOLS phase, it is checked whether all symbols used in the ASD module were indeed introduced, and whether their use conforms to the arity specified. In this SYMBOLS phase, an

ASF+SDF module is generated the signature of which contains all symbols introduced.

SORT Checking: By starting the SORTS phase, the functionalities — which specify the sort of the operands and the result — are taken into account as well. Since sort checking in Unified Algebras is undecidable, several heuristics are used. In this SORTS phase, an ASF+SDF module is generated containing all (auxiliary) sorts and function declarations.

Grammar Generation: ASD grammars can be used for two purposes: first to produce a context-free grammar that can be used to generate a parser recognizing programs in concrete syntax (“real” programs); and second to write abstract syntax trees in the format used in action semantic descriptions. These grammar declarations are translated to (1) an ASF+SDF module called ABSTRACT covering the ASD abstract syntax notation; and (2) an ASF+SDF module called CONCRETE containing the concrete notation, as well as a function (signature and defining equations) mapping the CONCRETE representation to the ABSTRACT one.

Rewriting: The semantic equations given in an action semantic description specify the mapping between ABSTRACT trees and Action Notation (generally extended with specific notation). The semantic function definitions are directly translated to ASF+SDF rewrite rules. Together with the ABSTRACT and CONCRETE steps, these can be used to translate actual program sources automatically to their action semantic denotation.

Further features (e.g., “pretty-printing”) are currently being implemented, using the generator technology as described in [2].

The demonstration will be based on the newest version of the ASD Tools (2.5) which is not only approximately 10 times as fast as the version showed during AMAST’93, but also capable of handling much larger action semantic descriptions. Most likely, example semantic definitions will be shown of “Ad” (a large subset of Ada) and full ISO Pascal.

References

1. J.A. Bergstra, J. Heering, and P. Klint, editors. *Algebraic Specification*. ACM Press Frontier Series. The ACM Press in co-operation with Addison-Wesley, 1989.
2. M. van den Brand and E. Visser. Generation of formatters for context-free languages. Technical Report P9506, Programming Research Group, University of Amsterdam, 1995. To appear in *ACM Transactions on Software Engineering Methodology*.
3. A. van Deursen, J. Heering, and P. Klint (eds.). *Language Prototyping, An Algebraic Specification Approach*, volume 5 of *AMAST Series in Computing*. World Scientific Publishing Co., 1996. To Appear.
4. P. Klint. A meta-environment for generating programming environments. *ACM Transactions on Software Engineering and Methodology*, 2(2):176–201, 1993.
5. P.D. Mosses. Unified algebras and institutions. In *LICS’89, Proceedings 4th Annual Symposium on Logic in Computer Science*, pages 304–312. IEEE, 1989.

6. P.D. Mosses. *Action Semantics*, volume 26 of *Cambridge Tracts in Theoretical Computer Science*. Cambridge University Press, 1992.
7. P.D. Mosses and D.A. Watt. Pascal action semantics. Technical report, Aarhus University, 1993. Draft, version 0.6. Available by *ftp* from ftp.daimi.aau.dk: pub/action/pascal.
8. J. P. Nielsen and J. U. Toft. Formal specification of ANDF, existing subset. Technical Report 202104/RPT/19, issue 2, DDC International A/S, Lundtoftevej 1C, DK-2800 Lyngby, Denmark, 1994.
9. D.A. Watt. *Programming Language Syntax and Semantics*. Prentice Hall, 1991.

Hardware and Software Requirements

For installing and demonstrating the ASD system:

Disk Space

- 250 Mbytes needed to install and store ASD.

Workstation

- Minimum 64 Mbytes main memory needed for running the demonstration.
- Preferably Sparc Station running Solaris.
- Standard Unix software needed: X-Windows, twm.
- Colour screen desirable, but not essential.