The ELAN Environment:  
a Rewriting Logic Environment based on  
ASF+SDF Technology  
— System Demonstration —

M.G.J. van den Brand

Centrum voor Wiskunde en Informatica (CWI), Kruislaan 413, NL-1098 SJ Amsterdam,  
The Netherlands  
LORIA-INRIA, 615 rue du Jardin Botanique, BP 101, F-54602 Villers-lès-Nancy Cedex,  
France  

P.-E. Moreau

LORIA-INRIA, 615 rue du Jardin Botanique, BP 101, F-54602 Villers-lès-Nancy Cedex,  
France  

C. Ringeissen

LORIA-INRIA, 615 rue du Jardin Botanique, BP 101, F-54602 Villers-lès-Nancy Cedex,  
France

1 Introduction

ELAN [10,3] is a specification language based on rewriting logic [9,12]. Some  
of the characteristic features of ELAN are rewriting, AC-matching, and strategies  
to control the non-determinism induced by non-confluent rewrite systems. Hence,  
AC-matching and strategies are two sources of non-determinism. The specificity  
of ELAN consists of integrating the two forms of non-determinism plus deter-  
ministic rule-based computations in the same environment. The development of  
ELAN specifications is supported by an environment which contains, among oth-  
ers, a parser, an interpreter [10], and a compiler [14,11]. This environment can be  
characterized as command line based. ELAN specifications were developed using  
standard editors and given to the environment for compilation or interpretation via  
the command line. The current version of the ELAN compiler [11] is independent

1 Email: Mark.van.den.Brand@cwi.nl, vandenbr@loria.fr  
2 Email: moreau@loria.fr  
3 Email: ringeiss@loria.fr  

© 2002 Published by Elsevier Science B. V. Open access under CC BY-NC-ND license.
of the rest of the environment and data exchange is based on REF (Reduced ELAN Format) [2].

ELAN can be used for the formal specification of a wide variety of problems, especially in the fields of automated deduction and constraint solving. ELAN provides:

- A general-purpose algebraic specification formalism based on (conditional) term rewriting.
- Modular structuring of specifications.
- Associative commutative rewriting.
- Strategies to guide rule application.

The design goals of the new ELAN environment to be demonstrated include: openness, adaptability, extensibility, and reusability of components.

The old ELAN environment

The old ELAN environment can be considered as a monolithic piece of software which is hard to maintain and not really open. The ELAN syntax, for instance, is “hard-wired” in the current implementation of the parser. Language modifications and extensions were hampered by this “hard-wiredness”. Furthermore, ELAN provided pre-processor language constructs to generate automatically pieces of ELAN code given a specification file. This pre-processor mechanism can be considered as a macro facility, it uses the ELAN interpreter to evaluate pre-processor expressions and thus led to a subtle interaction between parser and interpreter. A few years ago it was decided to use more generic language technologies for the design and implementation of a new ELAN environment. The idea was to use the ASF+SDF parsing technology to build a new ELAN parser, given the ELAN syntax without the pre-processor language constructs, which are not viewed as part of the ELAN syntax [6].

ASF+SDF technology

The technology applied in the ASF+SDF Meta-Environment [4] was also considered as a possible solution to improve the structure and maintainability of the ELAN environment and to make adaptations of the syntax for language extensions easier. ASF+SDF [8] is an algebraic specification formalism designed for the definition of the syntax and semantics of (programming) languages. The ASF+SDF Meta-Environment is an integrated programming environment to develop these language definitions and to generate a programming environment given a language definition. Three technical developments of ASF+SDF proved to be very useful for the development of an ELAN environment, namely ATerms [5], ToolBus [11], and the generic parsing technology [7]. The ATerms format is a generic formalism for the representation of structured information, like (abstract) syntax tree, parse tables, environments, etc. The ToolBus is a software coordination architecture, which
Fig. 1. Primitive strategy operators

takes care of the coordination of software components. The generic parsing technology consists of a parse table generator and a parser. The parser is a scannerless generalized LR parser (SGLR) [7].

2 ELAN4.0

The design of this new ELAN environment has also led to a complete redesign of the ELAN formalism itself. The syntax of this new ELAN version has completely changed. The syntax modifications can be summarized as follows:

- The signature definition is replaced by SDF.
- The ELAN pre-processor has been removed, and a fixed abstract syntax and a predefined ELAN library to manipulate abstract syntax trees has been introduced. Via this library it is possible to develop ELAN specifications which create and manipulate ELAN modules on the abstract syntax tree level.
- The families of rules per sort is replaced by a single list of rules.
- Variables are locally defined per module or globally per specification and no longer locally per family of rules.
- Builtin strategy primitives have no longer a fixed syntax.
- Strategy operators are defined in the same way as ordinary operators in the signature. Also the distinction between the definition of strategy rules and ordinary rules has disappeared.

An important difference between old and new ELAN is the way the strategies are defined. Instead of having builtin strategy primitives with a fixed syntax, we have opted for a more flexible approach where the strategies are built-in primitives, but the exact syntax of the strategy primitives is defined in a module. A part of the module defining the strategies is shown in Figure [1]. Figure [2] shows the use of this strategy module.
module Nqueens
imports BuiltinInt
   BasicStrategies[(BuiltinInt->BuiltinInt)]
...

Fig. 2. Importing and instantiating the strategy operators

User Interface

TOOLBUS

Text Editor Structure Editor Parser Parsetable Generator

ELAN Compiler

Interpreter

Unparser

Tree Repository

Fig. 3. Architecture of the ELAN4.0 environment

3 Architecture of ELAN4.0 environment

The architecture of the ELAN4.0 environment is shown in Figure 3. It consists of a ToolBus that interconnects the following components:

- **User interface**: the top level user interface of the system. It consists primarily of a graph browser for the import graph of the current specification.
- **Text Editor**: a XEmacs for text editing.
- **Structure Editor**: a syntax-directed editor which cooperates with the Text Editor.
- **Parser**: SGLR parser that is parameterized with a parse table.
- **Parsetable generator**: takes an SDF definition as input and generates a parse table for SGLR.
- **Tree Repository**: stores all terms corresponding to modules, parse tables, etc.
- **Compiler**: generates C code for a specification.
- **Interpreter**: executes specifications via interpretation.
- **Unparser**: pretty prints terms.

4 Lessons Learned

The design and implementation of the ELAN environment based on ASF+SDF technology has revealed a number of points where the ASF+SDF Meta-Environment could become more generic.
module BuiltinInt
...
  BuiltinInt "==" BuiltinInt
  -> BuiltinBool {builtin("eq")}
...
context-free syntax
"eq_int"(BuiltinInt, BuiltinInt) -> BuiltinBool
{alias(BuiltinInt "==" BuiltinInt
  -> BuiltinBool {builtin("eq")})}
...

Fig. 4. The use of aliases

4.1 SDF extensions

First of all, in order to express all ELAN specific syntax features, SDF had to be extended. The three main extensions of SDF to deal with ELAN specific features are:

- Parameterized sorts, e.g. List[[BuiltinInt]] or List[[X]], where X is a parameter of the module in which the parameterized sort List[[X]] is used. Instantiating the parameter X leads to an instantiation of List[[X]] as well.

- Strategy sorts, e.g. (BuiltinInt->BuiltinInt), they are used in combination with strategy operators.

- Alias attributes, they are used to define an alternative for a production rule.

SDF provides “symbol aliases” which are a kind of abbreviation mechanism when writing SDF specifications. The idea of the “ELAN aliases” is that the alias defines an alternative for a production rule. In the rules and terms both alternatives may be used, but internally only the aliased production rule is used. Figure 4 shows the use of the alias attribute for the definition of a prefix equality operator for BuiltinInts as an alternative for the infix equality operator.

4.2 Format differences

Both the ASF+SDF Meta-Environment and the ELAN environment use ATerms to represent parse tables and syntax trees. However, in the ASF+SDF Meta-Environment only parse trees are exchanged between components, whereas the ELAN environment uses both parse trees as well as abstract syntax trees. Components, like interpreter, compiler, and structure editors, of the ASF+SDF Meta-Environment use parse trees. The ELAN interpreter and compiler, however, use abstract syntax trees, this makes it necessary to define filters to map the parse trees into abstract syntax trees and vice versa. The mapping from parse trees to abstract syntax trees is in fact simple and can be very well localized, but the mapping back from abstract syntax trees is essential in order to visualize the normal forms via (structure) editors.
4.3 Conclusions

The ASF+SDF Meta-Environment is very generic. However, when developing this ELAN environment, it turned out that at some places, e.g. user interface and tree repository, ASF specific information was hard coded. These shortcomings were easy to correct.

Replacing the current SDF definition by an extended one or switching from parse trees to abstract syntax trees asked for more fundamental modifications of component interfaces and of the components themselves.

We will continue our experiments in the direction of developing an SDF Meta-Environment, which can easily be parameterized with formalisms like ELAN, ASF, Stratego [13], etc.

Acknowledgements

We would like to thank Paul Klint and Jurgen Vinju for their suggestions with respect to improvements of the generic nature of the Meta-Environment.

References


