Non-deterministic Computations in ELAN

Hélène Kirchner and Pierre-Etienne Moreau
LORIA-CNRS & INRIA
BP 239
54506 Vandoeuvre-lès-Nancy Cedex, France

Abstract. The ELAN system is an environment for specifying and prototyping constraint solvers, theorem provers and deduction systems in general. It also provides a framework for experimenting their combination. The ELAN language is based on rewriting logic and evaluation of labelled conditional rewrite rules. ELAN has two originalities with respect to several other algebraic languages, namely to handle non-deterministic computations and to provide a user-defined strategy language for controlling rule application. We focus in this paper on these two related aspects and explain how non-determinism is used in ELAN programs and handled in the ELAN compiler.

1 Introduction

The ELAN system [KKV95] provides an environment for specifying and prototyping deduction systems in a language based on rules controlled by strategies. Its purpose is to support the design of theorem provers, logic programming languages, constraint solvers and decision procedures and to offer a framework for studying their combination.

ELAN takes from functional programming the concept of abstract data types and the function evaluation principle based on rewriting. In ELAN a rewrite rule may be labelled, may have conditions and may introduce local variables. But rewriting is inherently non-deterministic since several rules can be applied simultaneously on a same term. So in ELAN, a computation may have several results. This aspect is taken into account through choice operations and a backtracking capability. One of the main originality of the language is to provide strategy constructors to specify whether a function call returns several, at least one or only one result. Non-determinism is handled with two operators: dc standing for dont-care-choose and dk standing for dont-know-choose. Determinism is enforced by the operator dc one standing for dont-care-choose one result. This declarative handling of non-determinism is part of a strategy language allowing the programmer to specify the control on rules application. This is in contrast to many existing rewriting-based languages where the term reduction strategy is hard-wired and not accessible to the designer of an application. The strategy language offers primitives for sequential composition, iteration, deterministic and non-deterministic choices of elementary strategies that are labelled rules. From these primitives, more complex strategies can be expressed. In addition the user
can introduce new strategy operators and define them by rewrite rules. Evaluation of strategy application is itself based on rewriting. Moreover it should be emphasised that ELAN has logical foundations based on rewriting logic [Mc92] and detailed in [BKK96,BKK98]. So the simple and well-known paradigm of rewriting provides both the logical framework in which deduction systems can be expressed and combined, and the evaluation mechanism of the language. The current version of ELAN includes an interpreter and a compiler written in C++ and Java, a library of standard ELAN modules, a user manual and examples of applications. Among those, let us mention for instance the design of rules and strategies for constraint satisfaction problems [Cas98], theorem proving tools in first-order logic with equality [KM95,CK97], the combination of unification algorithms and of decision procedures in various equational theories [Rin97,KR98]. More information on the system can be found on the WEB site\(^1\).

A first ELAN compiler was designed and presented in [Vit96]. Experiments made clear that a higher-level of programming is achieved when some functions may be declared as associative and commutative (AC for short). However rewriting in such theories is computationally difficult and the challenge is then to provide an efficient compiler for the language. The difficulty is both at the level of AC-matching and rewriting, addressed in [MK98], and at the level of non-deterministic computations, addressed in this paper. After a short presentation in Section 2 of ELAN programs and of the evaluation mechanism in ELAN, we explain in Section 3 the expressive power of non-deterministic features of the language and the related constructions of the strategy language. Then in Section 4, we detail how an analysis of non-determinism is performed and exploited in the ELAN compiler. We conclude in Section 5. We assume the reader familiar with basic definitions of term rewriting given for instance in [JK86].

2 Programs and evaluation mechanism of ELAN

An ELAN program is composed of a signature describing operators with their types, a list of rules and a list of strategies. A strategy provides a way to describe which computations the user is interested in, and specifies where a given rule should be applied in the term to be reduced. We describe informally here the evaluation mechanism and how it deals with rewrite rules and strategies.

In ELAN, rules are labelled rewrite rules with an optional sequence of conditions and/or local variable assignments:

\[ \text{[lab]} \quad l \Rightarrow r \quad \{ \text{if } v \quad \mid \text{where } y := (S)u \}^* \]

where lab is the label, \( l \) and \( r \) are terms respectively called left and right-hand sides, \( v \) is a boolean term called condition, and \( y := (S)u \) is a local assignment, giving to the local variable \( y \) the results of the strategy \( S \) applied to the term \( u \). Any sequence of where and if is allowed and their order is relevant for the evaluation. For applying such a rule on a term \( t \) at top position, first \( l \) is matched

\(^1\) http://www.loria.fr/ELAN.