Abstract

This paper describes the implementation of lazy pattern driven narrowing by extending an abstract machine for lazy rewriting to perform unification and backtracking.

The abstract machine LANAM is based on implementation methods for functional and logic languages like LML and PROLOG.

The approach leads to an efficient unification method for an equational theory and can also be used as a basis for efficient execution of functional logic programs.

We give an abstract definition of the implemented algorithm, describe the architecture of the abstract machine and discuss performance and implementation issues.

1 Introduction

Recently several approaches have been proposed to achieve an efficient implementation for functional logic languages [Red 87]. Most of them are based on Horn clauses with equality and use narrowing as operational semantics. Narrowing subsumes reduction and unification and can be implemented by extending the WAM [War 83], an abstract machine used to implement PROLOG [Bos 89, Bal 89, Han90, Mue 90].

We choose another approach, the extension of an abstract machine for lazy term rewriting LATERM [Wo 89] to handle unification and backtracking. The lazy narrowing machine LANAM uses an environment based architecture like the TIM machine [Fa 87].

For simplicity we restrict ourselves to first order conditional rewrite systems. An extension to arbitrary Horn clauses or higher order functions is straightforward.

We use no flattening of the input program as in [Bal 89, Mor90]. Flattening removes nested patterns and function calls by introducing auxiliary functions and simplifies the implementation. We avoid flattening because the additional functions introduce inefficiencies both at compile time and at run time.

Instead we transform the pattern into decision trees for efficient unification, similar to the technique used for the compilation of LML [Jos 87, Au 87]. There is an optimization of the WAM called full indexing in all arguments [III 89] achieving the same goal. Backtracking is restricted to rule selection by conditions, it is avoided for the selection by pattern.

We don't distinct explicitly between constructors and derived operations. If a function is not defined over all constructors it is possible, that the corresponding function symbol cannot be removed from a term by reduction. In this case no backtracking occurs. In a condition goal like $p = \text{true}$ the effect is the same as if we would perform backtracking. If $p$ contains a non reducible subterm, $p$ cannot be evaluated to $\text{true}$ and backtracking will occur because both sides of the goal are narrowed to different normal forms.

The paper is organized as follows. Section 2 gives a short introduction in conditional equations and narrowing, the lazy narrowing strategy is discussed in section 3. Section 4 presents the lazy narrowing algorithm. Section 5 explains the compilation of pattern into decision trees, section 6 contains a description of the LANAM architecture and the LANAM code generation. In section 7 implementation and performance issues are sketched briefly and finally section 8 includes some conclusions and remarks.
2 Conditional Equations

2.1 Notation

A signature \( \text{SIG} = (S, OP) \) consists of a set \( S \) of sorts and a \( S^+ \)-sorted set \( OP \) of operation symbols. \( X \) is a \( S \)-sorted set of variables, \( T_{OP,X} \) denotes the set of well sorted terms on \( OP \) and \( X \). For \( t \in T_{OP,X} \) \( V(t) \) denotes the set of variables in \( t \). We say a term \( t \) is ground if \( V(t) = \emptyset \) and \( T_{OP} \) denotes the set of ground terms.

Substitutions \( \sigma \) are defined as endomorphisms on \( T_{OP,X} \) that extend mappings from \( X \) to \( T_{OP,X} \) with a finite domain \( D(\sigma) \). A substitution is denoted by \( \sigma = [x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n] \). Applications of substitutions are written in postfix notation \( (t \sigma) \).

The composition of two substitutions \( \sigma \) and \( \sigma' \) is denoted by \( \sigma'' = \sigma \sigma' \). We call \( \sigma \) a prefix of \( \sigma'' \).

The subsumption quasi ordering \( \preceq \) on \( T_{OP,X} \) is defined by: \( t \preceq t' \) iff \( t = t \sigma \) for a substitution \( \sigma \). We call this situation a match from \( t \) to \( t' \), \( t' \) is an instance of \( t \) by \( \sigma \). \( \preceq \) can be extended to substitutions in a straightforward manner: \( \sigma \preceq \sigma' \) iff \( \sigma \) is a prefix of \( \sigma' \).

A substitution \( \sigma \) unifies two terms \( t \) and \( t' \) if \( t \sigma = t' \sigma \). A unifier \( \sigma \) is called most general if \( \sigma \) is a prefix of every unifier of \( t \) and \( t' \).

We call a set of equations \( P = \{ t_1 = t'_1, \ldots, t_n = t'_n \} \) a goal and \( P \rightarrow l = r \) a conditional equation (clause) which can be denoted \( \{ t_1 \rightarrow t'_1, \ldots, t_n \rightarrow t'_n \} \rightarrow l = r \). \( A \rightarrow l = r \) is the premise and the conclusion of the clause. Empty premises are omitted, we write \( l = r \) for \( \emptyset \rightarrow l = r \).

The concepts of substitution, variable set, subterm and subterm replacement are extended from terms to equations and goals in an obvious manner.

2.2 The conditional equational calculus

We derive the conditional equational calculus as a special case of the Horn clause calculus described in [Pad 87, Pad 88].

Let \( C \) be a set of clauses (conditional equations) and \( e \) a single clause. The conditional equational calculus consists of the following derivation rules where \( C \vdash c \) stands for \( c \) is derivable from \( C \):

Substitution Rule: For clauses \( c \) and substitutions \( \sigma \),

\[\{c\} \vdash c \sigma\]

Cut Rule: For equations \( p = p', q = q' \) and goals \( G, G' \),

\[\{G \cup \{q = q'\} \rightarrow p = p', G' \rightarrow q = q'\} \vdash G \cup G' \rightarrow p = p'\]

Composition Rule: \( C \vdash c \) and \( C' \vdash c' \) imply \( C \cup C' \vdash c' \).

The set \( \text{EAX} \) of equality axioms consists of all clauses of the form

- \( t = t' \)
- \( t = t', t' = t'' \)
- \( t_1 = t'_1, \ldots, t_n = t'_n \rightarrow op(t_1, \ldots, t_n) = op(t'_1, \ldots, t'_n) \)

The deductive theory of a specification \( (\text{SIG}, AX) \) (where \( AX \) is a set of clauses), \( \text{TH}(AX) \) is given by all clauses over \( \text{SIG} \) which are derivable from \( AX \cup \text{EAX} \) using the conditional equational calculus. Goals in \( \text{TH}(AX) \) are called \( (\text{SIG}, AX) \)-theorems. \( \text{TH}(AX) \) agrees with the congruence relation on \( T_{OP,X} \) generated by \( AX \). We call a substitution \( \sigma \) a solution of a goal \( G \) if \( \sigma \sigma \in \text{TH}(AX) \).

In general, for all goals \( g, g \in \text{TH}(AX) \) iff all models (algebras) of the specification \( (\text{SIG}, AX) \) satisfy \( g \) [Pad 88].

1^With "equation" we mean an unconditional equation. To achieve a clear distinction we adapt the notation "clause" from predicate logic and call conditional equations clauses.

2^We assume non-empty carrier sets for each sort.