

Paper:

Design of Nondeterministic Program Termination Based on the Equivalent Transformation Computation Model

Itaru Takarajima*, Kiyoshi Akama**, Ikumi Imani***, and Hiroshi Mabuchi****

*Department of Commerce, Nagoya Gakuin University
1350 Kami-Shinano, Seto, Aichi 480-1298, Japan
E-mail: takaraji@ngu.ac.jp

**Information Initiative Center, Hokkaido University, Sapporo, Japan
E-mail: akama@cims.hokudai.ac.jp

***Department of Foreign Studies, Nagoya Gakuin University, Seto, Japan
E-mail: imani@ngu.ac.jp

****Faculty of Software and Information Science, Iwate Prefectural University, Iwate, Japan
E-mail: mabu@soft.iwate-pu.ac.jp

[Received February 22, 2005; accepted December 21, 2005]

We studied the termination of nondeterministic programs, which play an important, basic role in program synthesis, and provide a foundation for proving program termination. The class of programs we consider here is based on the *equivalent transformation (ET)* computation model. Computation with this model involves the successive application of rules to sets of clauses.

Since the ET model has more expressive terms and rules than other programming languages, program termination is difficult to prove: we must take into account all possible changes of terms made by various rules. We propose a sufficient condition of ET program termination that proves termination by checking each rule in a given program. We also provide an algorithm for this check.

Keywords: termination, nondeterministic program, equivalent transformation

1. Introduction

We studied termination of *equivalent transformation (ET)* programs. An ET program is defined as a set of ET rules and an ET computation as a successive transformation of definite clauses by these rules.

We confine ourselves to problems represented by definite clauses and query atoms, which are well-known in logic programming (LP) [15]¹. Note that although this class of problems is often treated in LP, the ET computation model differs from logic-based declarative computation models. The correctness of computations is guaranteed by the principle that computations are equal to equivalent transformations in the ET model, while logic-based models consider computations as inferences as far as SLD

resolution is concerned.

ET programs allow various rules, including single-head rules, multihead rules, single-body rules, multibody rules, and rules having extra-logical atoms in their condition parts. This does not mean that we allow any kind of rules to be introduced freely. In ET programs, rules are restricted to those that “cause equivalent transformations,” guaranteeing the correctness of computations. The principle that “guarantees correctness by equivalent transformation” is the most general principle, and properly includes other principles guaranteeing correctness, e.g., resolution. In this sense, the ET computation model is more powerful than many other computation models.

ET programs include as subclasses most of the rule-based programs proposed insofar as their basic structures are concerned: examples include Petri nets [17], term rewriting systems [13], pure Prolog, functional languages and constraint logic programming (CLP) programs [9, 11, 12]. This strong expressiveness makes proving ET program termination more difficult, which creates new problems not solvable by existing termination theories.

ET programs make a highly effective use of nondeterminacy. A nondeterministic program is a program that may have more than one possible computation for a single input. Like Petri nets and term rewriting systems, which are well-known nondeterministic programs, logic-based programs based on the resolution principle have nondeterminacy. The notion of nondeterministic programs is important in program synthesis. Especially important are classes of nondeterministic programs that have expressiveness dealing with various computations and realize correct computations, of which the class of ET programs offers the most desirable framework among existing frameworks. We deal with nondeterministic programs in more detail in Section 2.

This paper is organized as follows: Section 2 details nondeterministic programs and their termination. Section 3 explains the ET framework. Section 4 demonstrates how ET rules work and describes program termination.

1. Problems that the ET computation model deals with are not restricted to those defined by logical formulas. See [14].

Section 5 presents a design for terminating nondeterministic programs.

2. Nondeterministic Programs and Their Termination

2.1. Nondeterministic Programs

By a *nondeterministic program* we mean a program that may have more than one possible computation path for a single input: like the case of Petri nets or term rewriting systems, the path to be selected cannot be determined beforehand.

Nondeterminacy is often seen in rule-based frameworks. In systems whose programs consist of several rewriting rules, e.g., term rewriting systems, a choice of rules is made at each computation step. Several rules are usually applicable at several positions. Since different choices result in different computation paths, confluency is assumed to guarantee the uniqueness of the result. Such nondeterminacy is also seen in functional languages and lambda calculi.

In a Petri net, transition is nondeterministically selected when more than one transition is possible. This demonstrates the typical nondeterminacy of parallel computation systems.

Nondeterminacy is also seen in basic parts of logic-based languages. In execution by SLD-resolution with a given set of definite clauses and a goal, the choice of a goal atom is made by a selection function. Since a selection function is arbitrarily chosen, logic-based languages may have many possible computations (SLD trees) for a pair of a logic program and a query.

Note that the term *nondeterminacy* in automata theory is used differently from that in this paper and the theories of term rewriting systems, Petri nets, and logic programming. In our paper, any single computation path selected nondeterministically gives all necessary results, while in a nondeterministic automaton, all paths should be searched until acceptance or rejection of an input word is determined. Nondeterminacy of an automaton means possible or-branches, none of which should be neglected from the logical viewpoint.

2.2. Developing a Theory of Nondeterministic Programs

Although nondeterminacy is seen in various computation paradigms, as stated above, it has not been thoroughly studied: the general definition of nondeterministic programs has not been made precise enough and the relationship between correctness and program synthesis of nondeterministic programs has not been fully clarified. This negligence results from the traditional belief that programs should be written to provide deterministic procedures in sequential computation, and from the fact that program synthesis theories have not been developed fully.

The deterministic language, Prolog, for example, is mainly used in logic-based languages, although such lan-

guages are expected to make good use of nondeterminacy. Parallel logic-based languages are supposed to contain nondeterminacy, but no general theory of nondeterminacy has been established, and correctness has not been sufficiently discussed in parallel computation paradigms.

Following attempts to overcome this situation have been proposed:

- (1) a general definition of nondeterministic programs and a definition of their correctness [3],
- (2) necessary and sufficient conditions of correctness of nondeterministic programs in the state-transition computation model [5],
- (3) the proposal and formalization of a large class of nondeterministic programs [4],
- (4) a theory of program synthesis for this class of programs [2],
- (5) systematic production of correct parallel programs from nondeterministic programs [16].

2.3. Nondeterministic Programs and Program Synthesis

A deterministic program has a single computation path for each input, and hence is a special case of nondeterministic programs, i.e., nondeterministic programs are an extension of deterministic programs.

When extending the range of programs from deterministic to nondeterministic, there are advantages in synthesizing programs, e.g., we may save description costs when we do not have to determine one computation path.

The most important advantage in using nondeterministic programs is that there are cases in which we get nondeterministic programs directly from problem descriptions, such as constraint satisfaction problems [18, 19] represented by definite clauses and queries. Starting with SLD resolution in logic paradigms, for example, a set of definite clauses is regarded as a nondeterministic program without any cost. In this case, we first get a nondeterministic program, then transform it to another nondeterministic program or a deterministic program.

2.4. ET Programs

Here we focus on the ET computation framework, which is one of the most general nondeterministic computation frameworks. The class of nondeterministic programs used in (3)-(5) of Section 2.2 is the class of ET programs.

The ET model provides a large class of representations and computations together with rigorously guaranteed correctness of computation. An ET program consists of ET rules applied to sets of definite clauses. A computation in the ET model is a finite or infinite sequence of sets of definite clauses, which are transformed repeatedly by ET rules.

An ET rule is mostly applied to a set of definite clauses with respect to an atom selected among body atoms of the clauses, which results in a nondeterministic computation.