CORRECTNESS PROOFS FOR META IV WRITTEN CODE GENERATOR SPECIFICATIONS USING TERM REWRITING

Bettina and Karl-Heinz Buth
Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel
Olshausenstraße 40-60
D-2300 Kiel 1

Abstract:

In recent years, computer scientists have become more and more convinced that verification is an important part of software development. We give an example for formal verification of "realistic" software: we show how to prove correctness of code generators that are developed within the compiler generating system CAT. Such code generators are large and involved programs specified in META IV. The proofs that are necessary become even larger and more involved, and to carry them out by hand seems to be an unfeasible task. Therefore automatic proof support is needed. We demonstrate that it is possible to give this support; we have written a proof support system which does essential parts of the proof. It is based on term rewriting and on specification by pre- and postconditions.

1. INTRODUCTION

Since the 1960's it is undisputed that verification of software is an essential part of software development, not only for software of theoretical interest but also for software that is practically used for industrial purposes. In recent years the employment of term rewriting methods in this field has been considered (cf. e.g. [Thatcher/Wagner/Wright 1981]), but up to now it has not been tried to a greater extent. The main obstacles are, on the one hand, the very high number of rewrite rules necessary, and on the other hand, that software usually is not given in an appropriate form.

In this paper, we want to demonstrate that term rewriting systems can be an effective support for the verification of large software systems, as in this case for the verification of code generators.

In the compiler generating system CAT, the essential part of a code generator is defined by a specification written in META IV. In the
main, such a specification is a system of recursive function definitions with one main function which is responsible for the translation of single source instructions into target code. Each of these specification functions is equipped with a pre- and a postcondition describing its characteristics. Proving the correctness of a code generator means to prove its partial and total correctness. For partial correctness, we use fixpoint induction; it turns out that it is sufficient to show that each specification function is partially correct with respect to its pre- and postcondition. Total correctness in addition demands that the main function is defined for all wellformed source instructions.

For the proof of partial correctness, we have found it quite natural to employ term rewriting. We transform the postcondition according to certain laws (arithmetical and logical rules, semantics definitions for source and target language and META IV,...) using the information we have about the function.

The proofs that arise this way can be carried out very systematically, and we succeeded in automating them by developing a proof support system. This system makes use of special term rewriting systems for each code generator specification and organizes the proof according to the requirements of fixpoint induction. The term rewriting systems are conditional ones; furthermore, their rules are applied according to a fixed order ("priority").

As an example, we use a code generator for the MC 68000 microprocessor and a subset of PASCAL as target language (without functions and procedures, real arithmetic and sets).

This paper is organized as follows:

Section 2 briefly describes the CAT system.
Section 3 contains descriptions of the specifications we have to deal with, i.e. the language and machine specifications.
In section 4, we introduce our proof principles, and in section 5, term rewriting aspects are discussed.
Section 6 contains a larger example of a proof with our system, and in section 7, we briefly compare our system with other approaches.

2. THE CAT SYSTEM

The CAT compiler system has originally been designed to be a multi-language multi-target system and is now used for compiler generation. It has been developed by U. Schmidt and R. Völler at the University of