UNIVERSIDADE DE ÉVORA

Escola de Ciências e Tecnologias
Mestrado em Engenharia Informática

Tópicos Avançados de Compilação
2015/2016

Assignment 1: TACL Intermediate Representation Generation

Autor:
Ricardo Fusco
29263

Professor:
Vasco Pedro

Évora, 28 de Novembro de 2015

1 Introduction

This assignment consists in making an IR generator, given the language AST, for the TACL language created by the professor for this curricular unit, a language a bit similar to C or Java in some aspects but a more simple one. This Intermediate Representation is intended to be used later on in order to generate the assembly code specific to the processor with any of the different architectures available.

2 Tools used

Prolog language was used to make the IR generator due to the fact that the AST is given either in a linearised form like this - (name attribute attribute ...) - or in the form of prolog terms. Given those prolog terms it becomes much easier to generate the IR using Prolog taking away the need to have a parser. In this AST functions, procedures, arguments, expressions, statements are represented by nodes which make up the tree, each node is represented by a prolog term - fun (identier, formal-arguments, body(declarations, statement,expression)).

3 IR

To generate the IR there were some first issues that had to do with how to properly "parse"and traverse the tree in order to correctly print the IR representation for each node, how to manage the conditional jumps like if/elseif/else and whiles and how to properly track the temporaries and labels for each function or procedure. The labels and temporaries(integer and real) tracking issue was easily solved bye having dynamic predicates for each and incrementing each while generating the IR and at the end of the function reset the predicates to 0 again using retracts and asserts. For the generation of the IR for each of the nodes the gen_ir/2 predicate was created, it generates the IR for the node calling the appropriate print and returns the result in the cases it is needed to be used later for some other expression or another operation that might be needed. The function and body predicates match the terms from the AST not requiring to be called by the predicate used. In the case of conditional jumps, the IR for a logic expression is generated similarly to the one from within an if node. Besides all this there was a need for the adequate prints for each kind of node, term, temporaries and labels. The generation of IR for global variable declarations was suppressed by simply ignoring the case where we have a global variable decl term in the AST.

4 Instructions

The main file for running the program is the TAC.pl one and the program can be run in two ways, one can open Swi-Prolog (Swi-Prolog was the environment used for this) and consult the TAC.pl file by running "['TAC.pl']. "or "consult('TAC.pl'). "and then pasting the AST prolog terms for the TACL code in the terminal and the program will output the IR, or a command can be run while in the terminal redirecting the file to the standard input -¿ "swipl TAC.pl ¡ example.pl "and the IR will be printed to the standard output. There is also a bash file in the assignment folder called "run_tests.sh"with a small script to runs all the AST examples in .pl files within the inputs folder and outputs the IR to the outputs folder to .ir files.