A Representation of a Compiler in PDDL

Claudio Scheer
Master's Degree in Computer Science
Pontifical Catholic University of Rio Grande do Sul - PUCRS
Porto Alegre - RS, Brazil

Abstract

Testing whether a program can be run in parallel is not an easy task. This process involves tasks that demand a lot of time and, consequently, finantial resources. As an approach to this problem, this paper proposes a formalization of a compiler with PDDL. With this formalization, it will be possible to reduce the identification of a parallel code to a conventional search problem.

There are still many programs that do not exploit the parallel capacity of today's processors. The cost to rewrite this program is high. Among the steps involved in the process of migrating a program to a parallel approach, the most expensive is to identify the regions of the program than can be executed in parallel.

According to (del Rio Astorga et al. 2018), loops detection, variable dependencies, identifying whether the arguments are read or written, among other analyzes, are the main patterns that identify a region parallel in a program.

Instead of using a static analysis of the source code, my approach will be to reduce the problem of identifying a region that can be executed in parallel to a search problem. The domain will be formalized using PDDL. In a nutshell, the domain will function as a compiler. Section discusses in more detail how the PDDL domain will work.

A PDDL domain describe actions that can be performed in an initial state, to achieve a set of goals. The initial state of the proposed approach will be the source code mapped to a set of predicates. This step is the bottleneck of that approach. However, this approach can be easily automated (I was thinking of creating this tool, but first I will implement the compiler). The expected set of goals will be the sequential execution of all the instruction mapped in the predicates.

Technical Approach

Compilers, in general, are complex programs. They need to translate the written source code into an executable code. In a rough overview, the compiler needs to break the source code into tokens. A token can be a keyword, a variable name, an operator, etc. The tokens is generally represented in a tree. In this tree, the compiler run some optimizations, such as removing unused variables and unreachable codes. After all that, the compiler can convert the tree to machine instructions.

In addition, the compiler must support all instructions provided by the programming language. As you can see, it is difficult to develop a compiler. Therefore, in this paper, I propose a compiler formalized as a PDDL domain, with support for only a few intructions (I do not know which instructions. I was thinking in arithmetic instructions, functions and loops). In theory, this approach would work for all programming languages. However, I will create the actions and the predicates from the perspective of a source code written in C.

- It would be cool to use fluents and test whether the output is correct or not
- cite planners to be tested
- include an example

Results Evaluation

As previously described, the goal of the planner will be to execute all the instructions of the program in the correct order, and find for instructions that can be executed in parallel. Therefore, to evaluate the correct output of the planner, I will map the predicates back to the source code and validate the parallel execution proposed by the planner.

Project Management

Task	Start	End
Understand better compilers	06-01-2020	06-03-2020
Support sum instruction	06-03-2020	06-07-2020
Support proposed instructions	06-08-2020	06-18-2020
Write paper	06-20-2020	06-25-2020

References

del Rio Astorga, D.; Dolz, M. F.; Sánchez, L. M.; García, J. D.; Danelutto, M.; and Torquati, M. 2018. Finding parallel patterns through static analysis in c++ applications. *The International Journal of High Performance Computing Applications* 32(6):779–788.