Checking Correctness of Code Generator Architecture Specifications

APR 18TH, 2016

论文下载

Intro & Abstract

由于现代指令集架构的复杂化，导致编译器需要时常地去维护更新coder generator使用的architecture specifications，但是例如GCC这类的编译器在测试时并不会对编译器中某个单独的部分进行测试，而只是进行end-toend testing

本文目的就是要去检测编译器中coder generator所使用的architecture specifications的正确性。由于现代编译器普遍地利用一个特定语言的前端将源代码转化为中间语言表示，再将这个中间语言表示利用coder generator转化为汇编代码。因此本文检测code generator正确性的方法就是检测IR片段以及它对应生成的汇编代码是否等价。

本文的主要贡献就在于提供了一个GCC的RTL的解释器，以及一个有效的架构无关的test case生成策略。

Problem Definition

Problem：Assuming that the code generator maps an IR instruction I to an assembly instruction A for some target architecture, is the semantics represented by I same as that of A?

为了比较汇编指令以及IR直接的语义一致性，通过设定Input state然后逐一比较每一步的状态，来确定两段代码之间的等价性。因此文章给出了相关的定义：

- Assembly State;
- IR State;
- Processor state correspondance;
- Soundness of IR;
- Code generator testing.

这些定义与以前的代码段之间的语义检测中的定义都十分类似，此处略。

Approach Overview

下图说明了ArCheck的主要的三个步骤：

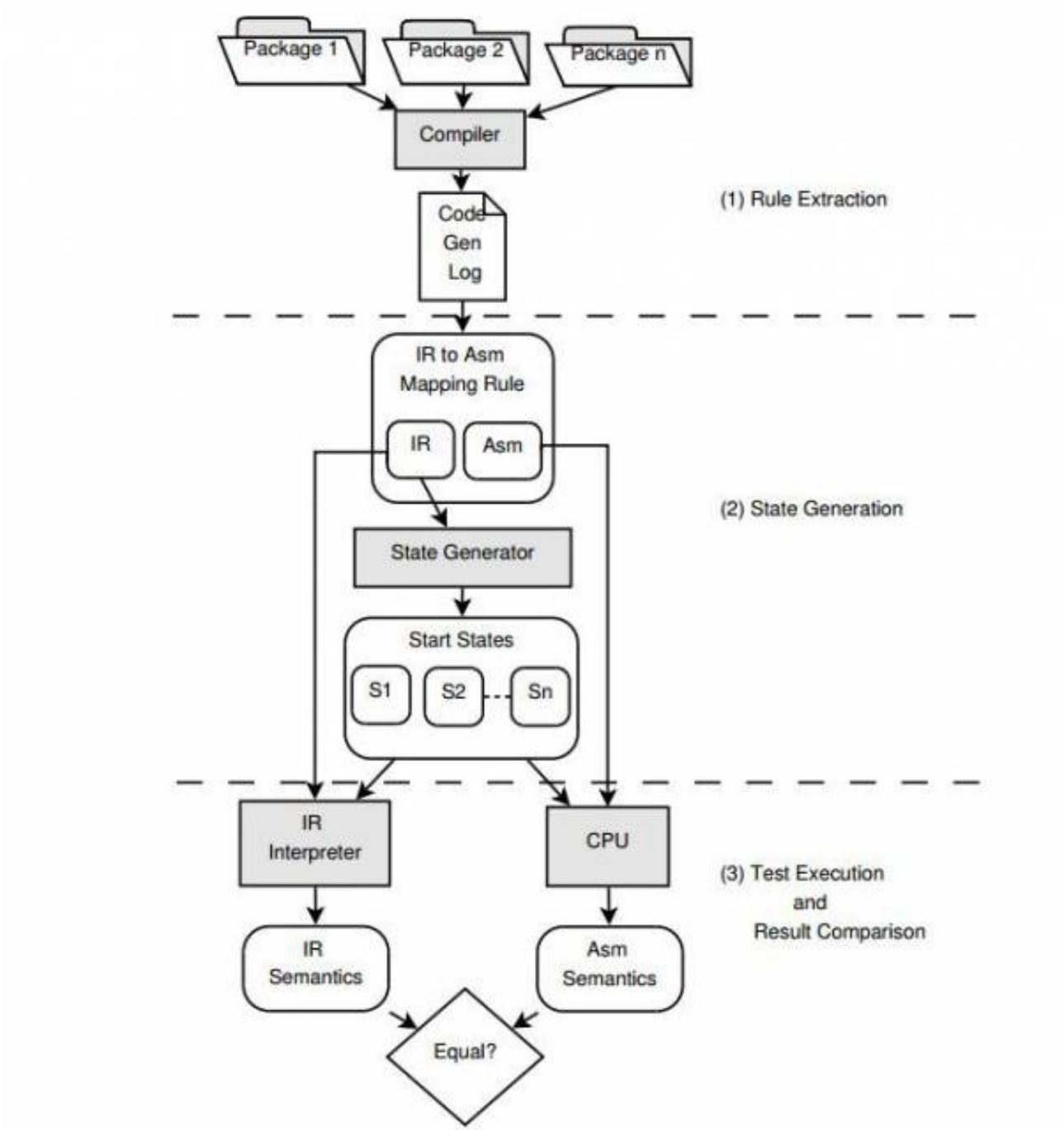

Figure 2. Design of *ArCheck*

Rule Extraction

这一步主要用于收集需要的RTL和汇编指令的配对用于测试，这一步和他们在后来的LISC（learning assembly-to-IL translators）中相同，将Code Generator当做一个黑盒编译各种程序，在编译过程中观察并收集这些<I, A>的配对。

Start State Generation

对某一对<I, A>需要生产starting states用于检测，本文基于对IR的白盒分析，利用constraint propagation，根据constraint solver获取初始状态中可能的取指范围，生成合适的starting state。

Test Exectuion and Result Comparison

主要就是根据start state分别计算执行RTL和汇编指令后的IR state以及Assembly state，比较是否对应

Evaluation

本文测试了GCC-4.5.1的x86 code generator for general-purpose and SSE x86 instructions.
主要是针对它的test case的生成策略相比于Random Testing的效率。
另外在检测中发现GCC的code generator中的一些bug。