Bound Analysis for Whiley Programs

Inferring Integer Bounds for Efficient Code Generation

Min-Hsien Weng, Mark Utting and Bernhard Pfahringer

Department of Computer Science at the University of Waikato

mw169@students.waikato.ac.nz, marku@waikato.ac.nz, bernhard@waikato.ac.nz

THE UNIVERSITY OF WAIKATO

Te Whare Wānanga o Waikato

Introduction

The Whiley programming language employs extended static checking to eliminate errors at the compile time, and compiles the high-level Whiley program into different kinds of implementations.[3] But translating high-leveled Whiley programs into efficient implementations has some challenges. For example, the use of arbitrary-sized integers downgrades the performance of Whiley implementation. The bound analyzer aims to assist the compiler to determine the efficient integer data types.

Main Objectives

1. Analyze each bytecode of the Whiley program to produce the constraints (propagation rule).
2. Infer the bounds and keep track of all bounds to analyze the feasibility of fixed points.
3. Determine the efficient integer data types.

Methods

Each integer at bytecode level has its own domain, which contains lower and upper bounds. And those constraints, produced from each bytecode, could restrict the domain to a finite set. In that case, solving the constraints over the finite constraint set (a.k.a constraint satisfaction problem) can be tackled with bound consistency.[2]

Bound Consistency

`Bound consistency` technique can infer the bounds consistent with all the constraints by propagating the lower or upper bounds among the variables.

Widening Operator

$$
\begin{aligned}
\perp \nabla [l_0,\ u_0] &= [l_0,\ u_0] \\
[l_0,\ u_0] \nabla \perp &= [l_0,\ u_0] \\
[l_0,\ u_0] \nabla [l_1,\ u_1] &= [\text{if } (l_1 < l_0) \text{ then } l_1 \text{ else } l_0, \\
&\quad\ \text{if } (u_0 < u_1) \text{ then } u_1 \text{ else } u_0]
\end{aligned}
$$

The `widening operator` ∇ in abstract interpretation converges the time of reaching the fixed points by extrapolating the bounds to $\pm\,\mathrm{inf}$.[1] The modified widening operator can be applied on bound inference to propagate the wider bounds and take the union of other bounds.

Class Diagram

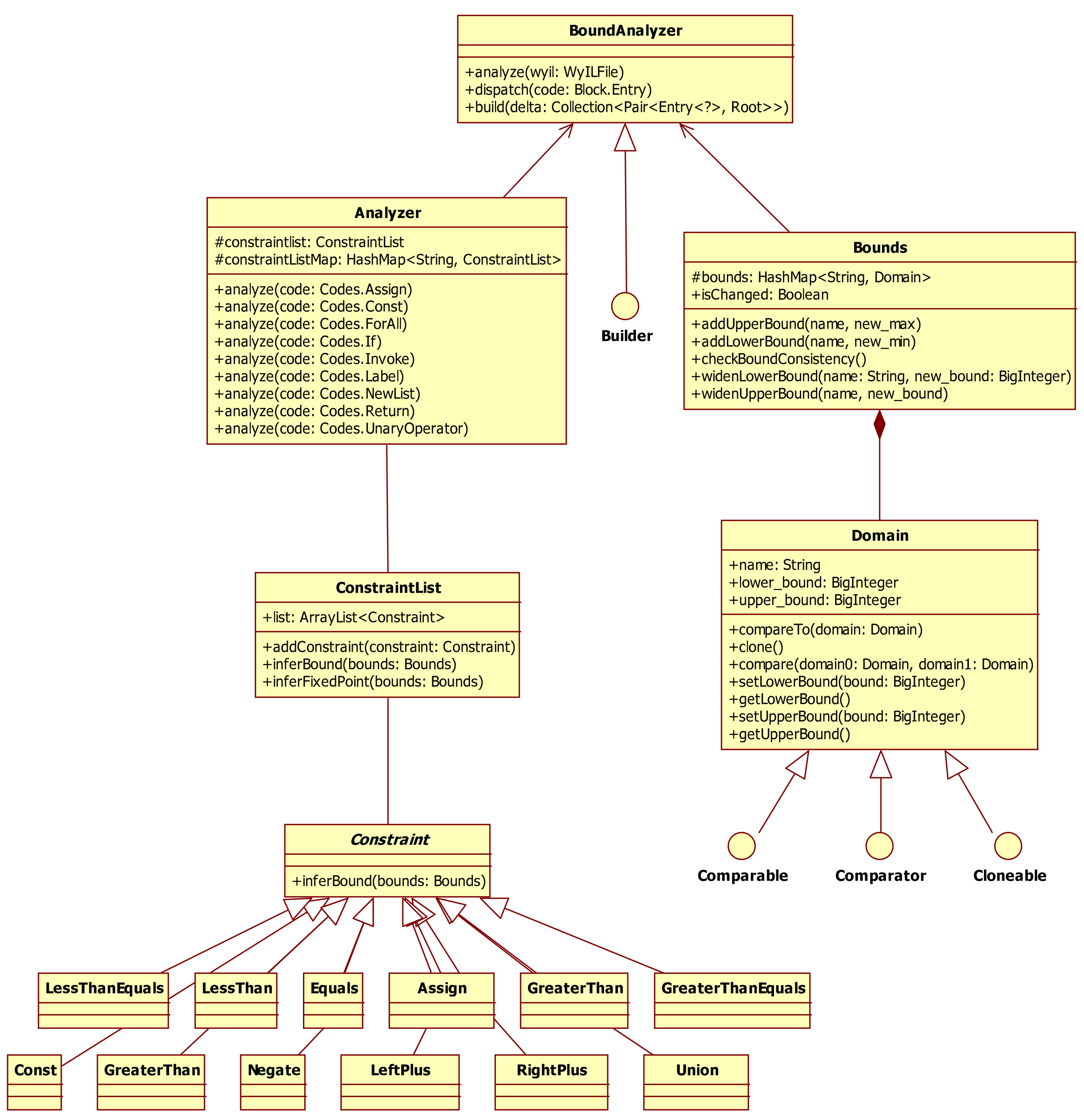

Figure 1: Class Diagram of Bound Analyzer

The bound analyzer is built on top of Whiley compiler project as a new module with extensive interfaces and classes in Figure1. It first uses the Whiley compiler to compile the Whiley program into the in-memory WyIL (Whiley Intermediate Language) code. Then it dispatches each byte-code to the specific analysis method accordingly for adding the constraints to the list or branching out the constraint list. After iterating over all the byte-code of a function, it infers the bounds of each list and take the union of the available bounds to produce the aggregated bound analysis results.

Analysis Result

The bound analyzer is used to analyze the below Whiley program:

```
function f(int x) => int:
    if x < 10:
        return 1
    else:
        if x > 10:
            return 2
    return 0
```

It branches the constraint list for if/else statement and outputs the below analysis results:

```
int f(int):
f.0 [   const %2 = 10 : int]
f.1 [   ifge %0, %2 goto blklab0 : int]
f.2 [   const %3 = 1 : int]
f.3 [   return %3 : int]
f.4 [.blklab0]
f.5 [   const %5 = 10 : int]
f.6 [   ifle %0, %5 goto blklab2 : int]
f.7 [   const %6 = 2 : int]
f.8 [   return %6 : int]
f.9 [.blklab2]
f.10 [.blklab1]
f.11 [  const %7 = 0 : int]
f.12 [  return %7 : int]

Union Bounds:
Bounds [
        D(%0) = [-infinity..9]
        D(%0_blklab0) = [10..infinity]
        D(%0_blklab2) = [-infinity..10]
        D(%2) = [10..10]
        D(%3) = [1..1]
        D(%5) = [10..10]
        D(%6) = [2..2]
        D(%7) = [0..0]
        D(return) = [0..2]
]
isBoundConsistency=true
```

The analysis results show that the input parameter (%0) has an infinite domain while the return value of $f(x)$ is restricted to a finite range ($[0..2]$), whose values can be stored with a `short` integer sufficiently.

Conclusions

The bound analyzer can

- add the constraints and infer the bounds for each function in Whiley.
- branch out the constraint list to produce the aggressive analysis results.
- provides an extensible architecture to include more constraints and analysis.

Forthcoming Research

The performance issues on the Whiley implementation include the unbounded integers, unbounded data structures and extra-value copying problems, etc. By solving those problems, I plan to develop a Whiley compiler to generate the efficient and reliable OpenCL code.

References

[1] Agostino Cortesi. Widening operators for abstract interpretation. In *Sixth IEEE International Conference on Software Engineering and Formal Methods, SEFM 2008, Cape Town, South Africa, 10-14 November 2008*, pages 31–40, 2008.

[2] K. Marriott and P.J. Stuckey. *Programming with Constraints: An Introduction*. Adaptive Computation and Machine. MIT Press, 1998.

[3] David J. Pearce. The Whiley Language Specification. Technical report, Victoria University of Wellington, 2014. Available at `http://whiley.org/download/WhileyLanguageSpec.pdf`.

Acknowledgements

Thank Dr. David J. Pearce (Whiley creator) for technique support on the Whiley compiler.