Automated Micro-analysis of Haskell

Adam Crume
adamcrume@soe.ucsc.edu

Review Buddy: Dimitris Skourtis

University of California, Santa Cruz

March 12, 2010

Overview

1. Background
2. Assumptions
3. Process
   1. Generate recurrence equations from code
   2. Substitute polynomial for time function
   3. Create linear system of polynomial coefficients
   4. Solve linear system
4. Issues
5. Conclusion

Background

Andrew Shewmaker worked on a project performing micro-analysis of C[1]. Approach was probabilistic, recommended solving recurrence equations.

Not much research on static execution time analysis of Haskell.

Assumptions

- Simplified syntax
- Time functions are exact polynomials
- Function arguments are integers or lists
- Time function only depends on list length
- Compiler is non-optimizing
- Arguments and results are fully evaluated

Generating Recurrence Equations

For example, consider a factorial function:

$$\begin{aligned}
&\texttt{fac 0} = 1 \\
&\texttt{fac x} = \underbrace{\ }_{1}\texttt{x}\underbrace{\ *\ }_{3}\underbrace{\texttt{fac(x}\underbrace{\ -\ }_{3}\texttt{1)}}_{T_{fac}(x-1)}
\end{aligned}$$

The recurrence equations would be:

$$\begin{aligned}
T_{fac}(0) &= 1 \\
T_{fac}(x) &= 7 + T_{fac}(x-1)
\end{aligned}$$

Substitute Polynomial for Time Function

Assuming $T_{fac}(x) = ax^2 + bx + c$, we get:

$$a0^2 + b0 + c = 1$$
$$ax^2 + bx + c = 7 + a(x-1)^2 + b(x-1) + c$$

Create Linear System of Polynomial Coefficients

The previous equations simplify to:

$$c = 1$$
$$0 = 7 - 2ax + a - b$$

The linear system extracted from this is:

$$c = 1$$
$$0 = -2a$$
$$0 = 7 + a - b$$

Solve Linear System

The linear system can be expressed as:

$$\begin{bmatrix} 0 & 0 & 1 \\ 2 & 0 & 0 \\ -1 & 1 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 7 \end{bmatrix}$$

Solving this yields $a = 0$, $b = 7$, and $c = 1$, so the final solution is $T_{fac}(x) = 7x + 1$.

More Examples

```
-- Should be 5/2*x^2 + 15/2*x + 5y + 2
f5 0 0 = 0
f5 x 0 = f5 (x-1) x
f5 x y = f5 x (y-1)

-- Should be 7x + 1
f6 x =
    case x of
      0 -> 0
      y -> 1 + f6 (y - 1)

-- Should be 4x + 1
f7 :: [Int] -> Int
f7 [] = 0
f7 (x:xs) = 1 + f7 xs
```

Under-constrained System

The linear system may be under-constrained (e.g. $a - b = 0$). This may be caused by lack of a base case.

$$\texttt{fac x} = \texttt{x} \ * \ \texttt{fac}(\texttt{x} - 1)$$

$$T_{fac}(x) = 7 + T_{fac}(x - 1)$$

$$ax^2 + bx + c = 7 + a(x - 1)^2 + b(x - 1) + c$$

In this case, c is undetermined.

Over-constrained System

The linear system may be over-constrained (e.g. $a = 0$ and $a = 1$)

- Time function may not be a polynomial
- Time function may have multiple cases
- Polynomial may have a higher degree than what was used

Conclusion

- Porting code from Java to Haskell is hard
- Monads are useful
- Haskell's unique features make it interesting to analyze
  - Easy - immutability, no side effects
  - Hard - lazy evaluation
- Difficult problem
- Lots of room for improvement
  - More supported syntax
  - Asymptotic time (esp. for partially solved systems)
  - Non-polynomial time functions
  - Verification; execution time as API

References

Andrew Shewmaker.
Micro-analysis of C project report.
http://users.soe.ucsc.edu/~cormac/wiki/lib/exe/fetch.php?id=projects&cache=cache&media=micro-analysis-of-c-report.pdf, Dec 2007.