Certification of parameter size with dependent ML

Martin Hofmann, Munich

15th April 2004

Work partially funded by the IST-FET Project
Mobile Resource Guarantees No. IST-2001-33149

Motivation

Let `f:int*int->unit` be a system function (native method).

We have a Java Bytecode `P` program using `f`.

We want to certify that whenever `f(x,y)` is called within `P` the parameters `x,y` satisfy some constraint `phi(x,y)`.

"Certify" = provide a formal, independently checkable proof about the operational semantics of `P`.

Possible Applications

Provider of 3rd party software updates for appliances, phones, set top boxes etc., can endow their programs which such certificates to convince users of their safety (not usefulness!).

Possible system functions `f`:

- controlling a brake
- billing a credit card
- spinning a washing mashine drum
- . . .

Approach

Write programs in high-level functional language

Use type system to describe parameter size bounds

Compile HLL program into bytecode

Translate type derivation into proof in bytecode logic.

NB: In MRG project we have developed all the infrastructure (HLL, Bytecode logic) and carried out the above for *heap size certification.*

Tasks

- Find type system that can describe bounds on parameter size
- Express such bounds in bytecode logic while maintaining useful properties of the logic (VCG, modularity, "recursive proof").
- Make it happen

Expressing bounds in program logic

Use instrumented operational semantics: $E, h \vdash e \Downarrow v, h', \rho$. Here ρ is a "resource record" representing accrued resource usage.

In our case ρ will just be a boolean which represents that the parameter bounds have been satisfied.

Evaluation rule for f: $E, h \vdash f(x, y) \Downarrow (), h, \texttt{phi}(E[x], E[y])$.

Use $\wedge$ on ρ for sequential composition.

VDM-style bytecode logic:

$$e : \Phi \Longleftrightarrow \forall E, h, h', v, \rho. E, h \vdash e \Downarrow v, h', \rho \Rightarrow (E, h, h', v, \rho) \in \Phi$$

Prove in logic $\texttt{main} : \{(E, h, h', v, \rho) \mid \rho = \texttt{true}\}$

Using Types

Assume that constraint `phi(x,y)` is a conjunction of linear inequalities.

Assign `f` the "type"

$$\texttt{f} : \{(x, y) : \texttt{int*int} \mid \texttt{phi}(x, y)\} \to \texttt{unit}$$

If you succeed to type "`main`" using `f` with this type only then the desired parameter bound will be guaranteed.

Using Dependent ML

Want a system that can express such types yet has desirable algorithmic properties.

Proposal: Xi's "Dependent ML"

Alternative: Constraint type systems.

Dependent ML

Index sorts: `int`,`rational`,`...` + closed under product.

Types: may depend on index sorts (type families) but not on other types.

In this way, type checking and inference are decidable and may be reduced to constraint solving over the index sorts.

Signature

Use type families `Int, Bool : int -> Type` where `Int(x)` contains just the integer `x` and `Bool(x)` contains `true` if $1 \leq x$ and `false` otherwise.

```
0 : Int(0)
1 : Int(1)
plus : Pi x,y:int.Pi xx:Int(x).Pi yy:Int(y).Int(x+y)
times_q : Pi x:int.Pi xx.Int(x).Int(q.x)
true : Pi x:int|1<=x.Bool(x)
false : Pi x:int|x<= 0.Bool(x)
leq : Pi x,y:int.Int(x) -> Int(y) -> Bool(1+y-x)
```

Typing rule for if-then-else:

$$\frac{\Gamma \vdash t_1 : \mathtt{Bool}(i) \qquad \Gamma, 1 \leq i \vdash t_2 : T \qquad \Gamma, i \leq 0 \vdash t_3 : T}{\Gamma \vdash \mathtt{if}\ t_1\ \mathtt{then}\ t_2\ \mathtt{else}\ t_3 : T} \qquad (\textsc{If})$$

Example

```
f : Pi x,y:int|x+y<=10.Int(x) -> Int(y) -> Unit

main : Pi x,y:int.Int(x) -> Int(y) -> Unit
main-body = lambda x,y:int.lambda xx:Int(x).lambda yy:Int(y).
                  if leq[x+y,10](plus[x,y](xx,yy))
                        then f[x,y](xx,yy)
                        else f[0,0](0,0)
```

The easiest way to maintain a bound is by explicitly checking it.