Semantics of Programming Languages (18)

Christian Urban

http://www4.in.tum.de/~urbanc/Teaching/semantics08.html
Compilers Are in the TCB

The **Trusted Code Base** is the code that, if compromised, causes all of your security to fail.
Compilers Are in the TCB

- The **Trusted Code Base** is the code that, if compromised, causes all of your security to fail.
Theorem: For all programs $S$, if the compiler generates machine code $C$ from $S$ (without reporting an error), and if $S$ satisfies property $P$, then also $C$ satisfies $P$.

$P$ might be
- observable behaviour
- type safety
- memory safety
Typical Language Definition

if (Expression) Statement

An if-then statement is executed by first evaluating the Expression. If the result is of type Boolean, it is subject to unboxing conversion (5.1.8). If evaluation of the Expression or the subsequent unboxing conversion (if any) completes abruptly for some reason, the if-then statement completes abruptly for the same reason. Otherwise, execution continues by making a choice based on the resulting value:

- If the value is true, then the contained Statement is executed; the if-then statement completes normally if and only if execution of the Statement completes normally.
- If the value is false, no further action is taken and the if-then statement completes normally.
If you write a compiler, you like to do transformations like:

\[
\text{if TRUE then } P \text{ else } Q \approx P
\]
Our Work on LF

Logical Framework is a system for specifying and reasoning about formal systems (type systems, deduction systems, etc).

```plaintext
nominal_datatype kind =
  Type
  | KPi "ty" "«var» kind"
and ty =
  TConst "id"
  | TApp "ty" "trm"
  | TPi "ty" "«var»ty"
and trm =
  Const "id"
  | Var "var"
  | App "trm" "trm"
  | Lam "ty" "«var»trm"
```
Correctness in LF depends on type-checking, which in turn depends on type-equivalence checking.

Harper and Pfenning describe in a 41-pages (refereed) journal paper an algorithm that decides type-equivalence.
Type-Checking in LF

Correctness in LF depends on type-checking, which in turn depends on type-equivalence checking.

Harper and Pfenning describe in a 41-pages (refereed) journal paper an algorithm that decides type-equivalence.

We found and fixed a gap in their proof.
Type-Checking in LF

- Correctness in LF depends on type-checking, which in turn depends on type-equivalence checking.

Harper and Pfenning describe in a 41-pages (refereed) journal paper an algorithm that decides type-equivalence.

- We found and fixed a gap in their proof.
- I will present this next week at LICS (Pittsburgh). Dr Stefan Berghofer will give two guest lectures.
In my PhD, I proved a property about classical logic:

```plaintext
nominal_datatype trm =
    Ax  "name" "coname"
  | Cut "<coname>trm" "<name>trm"
  | NotR "<name>trm" "coname"
  | NotL "<coname>trm" "name"
  | AndR "<coname>trm" "<coname>trm" "coname"
  | AndL_1 "<name>trm" "name"
  | AndL_2 "<name>trm" "name"
  | OrR_1 "<coname>trm" "coname"
  | OrR_2 "<coname>trm" "coname"
  | OrL "<name>trm" "<name>trm" "name"
  | ImpR "<name>(<coname>trm)" "coname"
  | ImpL "<coname>trm" "<name>trm" "name"
```

Me, my supervisor, my referees (Henk Barendregt and Andrew Pitts), the referees of a conference and a journal paper all overlooked a few gaps in a complicated proof.
In my PhD, I proved a property about classical logic:

```plaintext
definition nominal_datatype trm =
    Ax "name" "coname"
    Cut "<coname>trm" "<name>trm"
    NotR "<name>trm" "coname"
    NotL "<coname>trm" "name"
    AndR "<coname>trm" "<coname>trm" "coname"
    AndL₁ "<name>trm" "name"
    AndL₂ "<name>trm" "name"
    OrR₁ "<coname>trm" "coname"
    OrR₂ "<coname>trm" "coname"
    OrL "<name>trm" "<name>trm" "name"
    ImpR "<name>(<coname>trm)" "coname"
    ImpL "<coname>trm" "<name>trm" "name"
```

Me, my supervisor, my referees (Henk Barendregt and Andrew Pitts), the referees of a conference and a journal paper all overlooked a few gaps in a complicated proof.

Stefan will also jump in on 15.+16. July
nominal_datatype lam =
  VAR "name"
  APP "lam" "lam"
  LAM "«name»lam" ("LAM [__].__")
  TRUE | FALSE
  IF "lam" "lam" "lam"
  FIX "«name»lam" ("FIX [__].__")

datatype ctx =
  Hole ("□")
  CLAM "name" "ctx"
  CAPPL "ctx" "lam"
  CAPPR "lam" "ctx"
  CIF1 "ctx" "lam" "lam"
  CIF2 "lam" "ctx" "lam"
  CIF3 "lam" "lam" "ctx"
  CFIX "name" "ctx"
Congruences

A congruence \( R \) is an equivalence relation such that for all contexts \( C \)

\[
t_1 \ R \ t_2 \text{ implies } C[t_1] \ R \ C[t_2]
\]

definition

\[
\text{EQUIV :: "(lam } \Rightarrow \text{ lam } \Rightarrow \text{ bool) } \Rightarrow \text{ bool"
}
\]

where

"\text{EQUIV } R \equiv \text{REFL } R \land \text{SYM } R \land \text{TRANS } R"

definition

\[
\text{CONGR :: "(lam } \Rightarrow \text{ lam } \Rightarrow \text{ bool) } \Rightarrow \text{ bool"
}
\]

where

"\text{CONGR } R \equiv \text{EQUIV } R \land (\forall C \ t_1 \ t_2. \ R \ t_1 \ t_2 \rightarrow R \ C[t_1] \ C[t_2])"
Compatibility

A congruence $R$ is compatible if $\rightarrow \subseteq R$.

definition
$COMP :: "(lam \Rightarrow lam \Rightarrow bool) \Rightarrow bool"
where
"COMP R \equiv CONGR R \land (\forall t_1 t_2. t_1 \rightarrow cbv t_2 \rightarrow R t_1 t_2)"

lemma
assumes $a: "COMB R"
shows "t_1 \rightarrow cbv t_2 \rightarrow R t_1 t_2"
and "t \downarrow v \rightarrow R t v"
Convergence and Context Equivalence

A term \( t \) converges, written \( t \downarrow \), iff there exists a \( v \) such that \( t \downarrow v \).

definition
converges :: "lam \Rightarrow bool" ("_ \downarrow")
where
"t \downarrow \equiv \exists v. t \downarrow v"

Two terms, \( t_1 \) and \( t_2 \), are contextual equivalent iff for all \( C \), \( C[t_1] \downarrow \) iff \( C[t_2] \downarrow \).

definition
ctx_equiv :: "lam \Rightarrow lam \Rightarrow bool" ("_ \approx _")
where
"t_1 \approx t_2 \equiv \forall C. (C[t_1] \downarrow) \leftrightarrow (C[t_2] \downarrow)"
\[ \approx \text{ is Compatible} \]

**lemma**

"COMP ctx\_equiv"

- TRUE \( \not\approx \) FALSE
- if \( t \) diverges then \( t \approx \text{LOOP} \) where \( \text{LOOP} \) is \( \text{FIX} \ [f].\text{VAR} \ f \)
\( \equiv \) is Compatible

**Lemma**

"COMP ctx_equiv"

- TRUE \( \not\equiv \) FALSE
- if \( t \) diverges then \( t \equiv \textit{LOOP} \) where \( \textit{LOOP} \) is \( \text{FIX} \ [f]. \text{VAR} \ f \)

- Contextual equivalence talks about all contexts. CIU theorems prove that one can restrict this to evaluation contexts.
The purpose of theory is not just correctness; it provides a much greater understanding of the issues. Correctness is a nice by-product.