Proof Carrying Code

George Ciprian Necula

BRICS, University of Aarhus
July 5, 2000
Motivation: The Problem

Often profitable:
▶ Execution of mobile code (MC) in a host system (H)
▶ Execution of programs with components written in different languages
▶ Execution of applications within the OS kernel space.

Often troublesome:
▶ Guarantee that MC will not abuse H.
▶ Guarantee that different components comply with execution policies.
▶ Guarantee that applications respect kernel’s internal invariants.
Motivations: Some Approaches

- Personal Authority.
- Hardware-Based Address Spaces.
- Trusted Interpretation.
- **Proof Carrying Code.**
Agenda

1. Proof Carrying Code Overview.

2. Case Study: Safe extension of TIL run-time system.

3. Discussion.

4. Conclusion.
Overview of PCC

Stages: Certification, Validation and Execution.
Case Study: Safe Extensions of the TIL Run-Time System

A Standard ML program.

```ml
datatype T = Int of int | Pair of int * int

fun sum (l : T list) =
  let
    fun foldr f nil a = a
    | foldr f (h::t) a = foldr f t (f(a, h))
  in
    foldr (fn (acc, Int i) => acc + i
       | (acc, Pair (i, j)) => acc + i + j)
      l 0
  end
```
Establishing a Safety Policy

Data Representation, Expressions and Memory States:

\[ \tau ::= \text{int} \mid \tau \times \tau \mid \tau + \tau \mid \tau \text{ list} \]

val r0 : int = 5
val r1 : int * int = (2, 3)
val r2 : T = Pair r1
val r3 : T = Int 6
val r4 : T list = [r3, r2]

\[ e ::= n \mid r_i \mid \text{sel}(m, e) \mid e_1 + e_2 \]

\[ m ::= r_m \mid \text{upd}(m, e_1, e_2) \]

Type Judgments \( m \vdash e : \tau \)
Establishing a Safety Policy

The Typing Rules:

\[
\begin{align*}
& m \vdash e : \tau_1 \times \tau_2 \\
\Rightarrow & m \vdash e : \text{addr} \land m \vdash e + 4 : \text{addr} \land m \vdash \text{sel}(m, e) : \tau_1 \land m \vdash \text{sel}(m, e + 4) : \tau_2 \\
\end{align*}
\]

\[
\begin{align*}
& m \vdash e : \tau_1 + \tau_2 \\
\Rightarrow & m \vdash e : \text{addr} \land m \vdash e + 4 : \text{addr} \land \text{sel}(m, e) = 0 \Im m \vdash \text{sel}(m, e + 4) : \tau_1 \land \text{sel}(m, e) \neq 0 \Im m \vdash \text{sel}(m, e + 4) : \tau_2 \\
\end{align*}
\]

\[
\begin{align*}
& m \vdash e : \tau \text{ list} \\
\Rightarrow & e \neq 0 \Im m \vdash e : \text{addr} \land m \vdash e + 4 : \text{addr} \land m \vdash \text{sel}(m, e) : \tau \land m \vdash \text{sel}(m, e + 4) : \tau \text{ list} \\
\end{align*}
\]

\[
\begin{align*}
& m \vdash e_1 : \text{int} \land m \vdash e_2 : \text{int} \\
\Rightarrow & m \vdash e_1 + e_2 : \text{int} \\
\end{align*}
\]

\[
\begin{align*}
& m \vdash 0 : \text{int} \\
\end{align*}
\]
An Assembly Implementation of sum

```
0  sum: INV r_m ≡ r_0 : T list
    %r_0 is 1
1    MOV r_1, 0       %r_1 is acc
2    L_2              %Initialize acc
2    INV r_m ≡ r_0 : T list ∧ r_m ≡ r_1 : int
    %Loop invariant
3    BEQ r_0, L_{14}  %Is list empty?
4    LD r_2, 0(r_0)    %Load head
5    LD r_0, 4(r_0)    %Load tail
6    LD r_3, 0(r_2)    %Load constructor
7    LD r_2, 4(r_2)    %Load data
8    BEQ r_3, L_{12}  %Is an integer?
9    LD r_3, 0(r_2)    %Load i
10   LD r_2, 4(r_2)    %Load j
11   ADD r_2, r_3, r_2 %Add i and j
12   L_{12}            %Do the addition
13   BR L_2            %Loop
14   L_{14}            %Copy result in r_0
15   MOV r_0, r_1     %Result is in r_0
15   RET
```
Computing the Verification Condition

\[
\begin{align*}
\text{Pre} & \equiv r_m \vdash r_0 : T\text{ list} \\
\text{Post} & \equiv r_m \vdash r_0 : \text{int} \\
VC_i = \begin{cases} \\
[r_s + op/r_d] VC_{i+1}, & \text{if } \Pi_i = \text{ADD } r_s, op, r_d \\
r_m \vdash r_s + n : \text{addr} \land [sel(r_m, r_s + n)/r_d] VC_{i+1}, & \text{if } \Pi_i = \text{LD } r_d, n(r_s) \\
(r_s = 0 \lor VC_{i+n+1}) \land (r_s \neq 0 \lor VC_{i+1}), & \text{if } \Pi_i = \text{BEQ } r_s, n \\
\text{Post}, & \text{if } \Pi_i = \text{RET} \\
\text{I}, & \text{if } \Pi_i = \text{INV } I \\
\end{cases} \\
\forall r_i. \bigwedge_{i \in \text{Inv}} Inv_i \supset VC_{i+1}
\end{align*}
\]
Soundness of VC-based Certification

Theorem 3.1 For any program $\Pi$, set of invariants $Inv$ and postcondition $Post$ such that $\Pi_0 = INV \ Pre$, if $\triangleright VC(\Pi, Inv, Post)$ and the initial state satisfies the precondition $Pre$, then the program reads only from valid memory locations as they are defined by the typing rules, and if it terminates, it does so in a state satisfying the postcondition.
Safety Proofs

▷ Predicates and Proofs represented in LF.
▷ Proof Checking Reduced to Type-Checking.

**Corollary 3.3** If $P$ is a closed predicate and $M$ is a canonical LF object such that $\vdash_{LF} M : \text{pf } P$, then there exists a derivation of $D :: \vDash P$, that is $P$ is valid. Furthermore, $M = \vdash D$. 
Discussion and Future Work

▷ Troublesome to find good Invariants.

▷ Certifying Compilers to scale up PCC.

▷ Choice of the Underlying Logic.
Conclusions

- Staging program verification into certification and proof validation.
- Checking type safety at assembly language level.
- Encoding of safety proofs as first-order logic derivations in LF.
- Combination of ideas from program verification, logic and type theory.