CS6848 - Principles of Programming Languages

Principles of Programming Languages

V. Krishna Nandivada

IIT Madras

Typed Assembly Language

What is TAL?

- A type system for assembly language(s)
  - Has built-in abstractions (tuple, code)
  - operators to build new abstraction ($\forall, \exists, \lambda$).
  - annotations on assembly code.
  - an abstraction checker (= type checker)
  - We will present a quick intro to TAL - details self study.
- Theorem: Well annotated code cannot violate abstractions.

Why Typed Assembly

- **Theory**
  - Simplifies proofs of compiler correctness
  - Helps in deeper understanding of compilation
- **Practice**
  - Helps in compiler debugging.
  - Software based protection (code over the wire).
- Difference between JVM and TAL?

Goal

- Control flow safety (TAL-0):
  - Cannot jump to arbitrary points.
  - Has to be well defined.
  - If it is a call - the arguments must have the ‘right’ properties.
  - Otherwise?
- Memory Safety (TAL-1):
  - No memory access should read or write data object at a given location, unless the program has been granted access to that location.

TAL description

$$
\begin{array}{lcll}
v & ::= & c|l & (values)\\
t & ::= & \text{Int} \mid \Gamma & (types)\\
e & ::= & \text{mov } r_d, r_s; e | \text{set } r_d, v; e | \text{inc } r; e | \text{jmp } r & (code)\\
R & ::= & [r \mapsto v, \cdots] & (registerfile)\\
\Gamma & ::= & [r : t, \cdots] & (regfiletype)\\
H & ::= & [l \mapsto (\Gamma, e), \cdots] & (codeheap)\\
A & ::= & [l : \Gamma, \cdots] & (heaptype)\\
s & ::= & (H, R, e) & (programState)
\end{array}
$$

c ranges over constants, l ranges over labels (or addresses).

Semantics

A small step operational semantics is given by the reflexive, transitive closure of the relation $\rightarrow_V$.

$$\rightarrow_V \subseteq programState \times programState$$

(1) $(H, R, \text{mov } r_d, r_s; e) \rightarrow_V (H, R[r_d \mapsto R(r_s)], e)$

(2) $(H, R, \text{set } r_d, v; e) \rightarrow_V (H, R[r_d \mapsto v], e)$

(3) $(H, R, \text{inc } r; e) \rightarrow_V (H, R[r \mapsto \lceil R(r) + 1 \rceil], e)$

(4) $(H, R, \text{jmp } r; e) \rightarrow_V (H, R, e)$, provided $R(r) = l$ and $H(l) = (\Gamma, e)$

Soundness

- A program is stuck if there is no program state s' such that $s \rightarrow_V s'$.
- A program state s goes wrong if $\exists s' : s \rightarrow_V^* s'$ and s' is stuck.
- We want to provide type rules so that we can claim that
  - Well typed programs cannot go wrong.

Type rules

Type rules for Values:

(5) $$A \vdash c : \text{Int}$$

(6) $$A \vdash l : \Gamma (A(l) = \Gamma)$$

Types rules for register files:

(7) $$\frac{A \vdash v : t \ \cdots \qquad \Gamma(r) = t, \cdots}{A, \Gamma \vdash [r \mapsto v, \cdots]}$$

Ordering of register file types:

(8) $[r_1 : t_1, \cdots r_n : t_n, \cdots r_{n+m} : t_{n+m}] \leq [r_1 : t_1, \cdots r_n : t_n]$ where $m \geq 0$

Type rules(cont.)

Type rules for Code:

(9) $$\frac{A, \Gamma[r_d : \Gamma(r_s)] \vdash e}{A, \Gamma \vdash \text{mov } r_d, r_s; e}$$

(10) $$\frac{A \vdash v : t \qquad A, \Gamma[r_d : t] \vdash e}{A, \Gamma \vdash \text{set } r_d, v; e}$$

(11) $$\frac{\Gamma(r) = \text{Int} \qquad A, \Gamma \vdash e}{A, \Gamma \vdash \text{inc } r; e}$$

(12) $$\frac{\Gamma(r) = \Gamma' \qquad \Gamma \leq \Gamma'}{A, \Gamma \vdash \text{jmp } r}$$

Type rules for code heaps:

(13) $$\frac{A, \Gamma \vdash e \qquad A(l) = \Gamma}{A \vdash [l \mapsto (\Gamma, e)]}$$

Type rules(cont.)

Type rules for the program states:

(14) $$\frac{A \vdash H \qquad A, \Gamma \vdash R \qquad A, \Gamma \vdash e}{\vdash (H, R, e)}$$

A program state s is well typed if and only if $\vdash s$.
Reading exercise: well-typed program state cannot go wrong.

Acknowledgements

- Lecture notes from Jens Palsberg
- Slides from David Walker
- Advanced Topics in Types in Programming Languages - Benjamin Pierce.