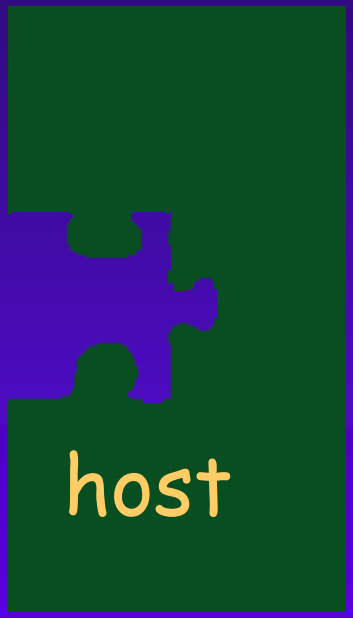
Construction of a Semantic Model for a Typed Assembly Language

Gang Tan, Andrew Appel, Kedar Swadi and Dinghao Wu
Princeton University
Jan 11, 2004

Extensible systems

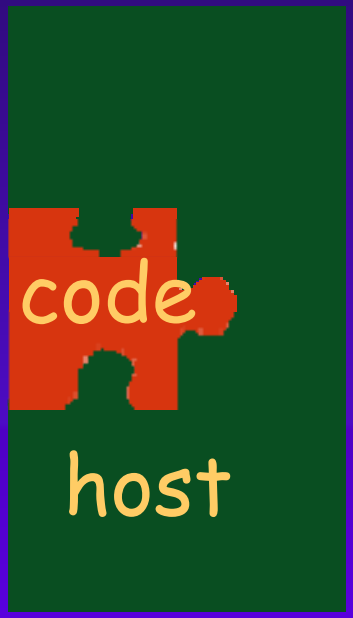
extensions



Applet, plug-in
Device drivers
Packet filters
VB Add-ins

Web browser
Operating system
Routers
PowerPoint

Security concerns

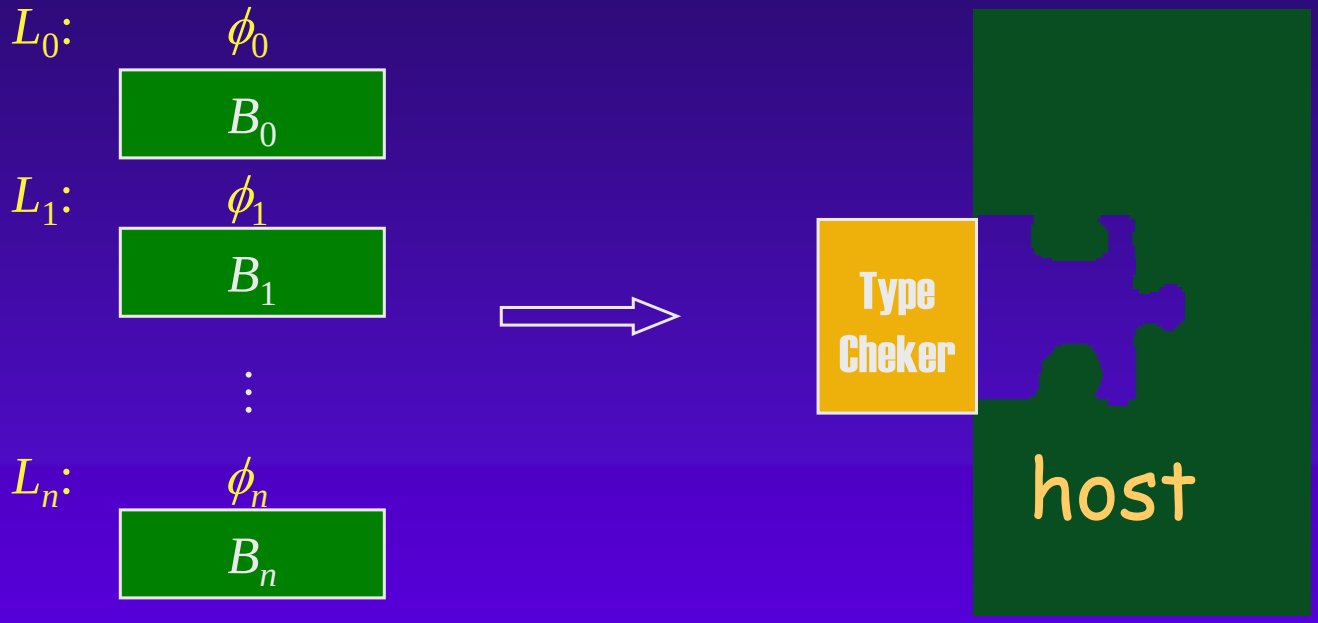


How to give this untrusted code direct access without violating host's safety policy?

We consider the case of machine/assembly code

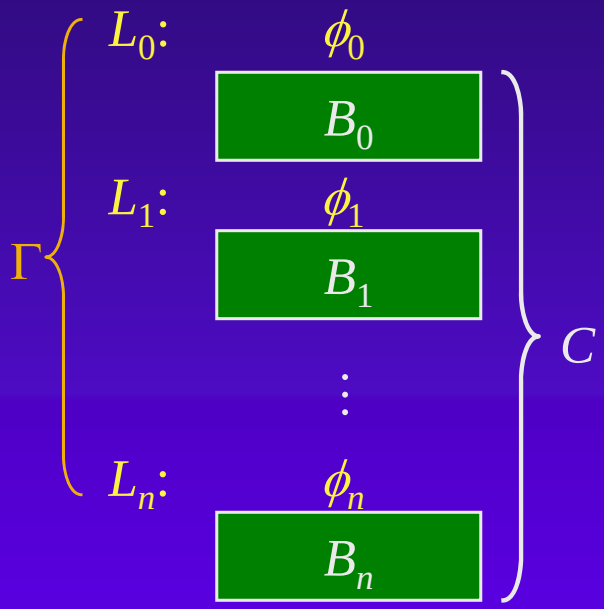
Typed Assembly Languages (TAL)

[Morrisett, Walker, Crary and Glew 1999]



- Loop invariants
  - in terms of types
  - generated by a compiler
- Type checks the assembly code

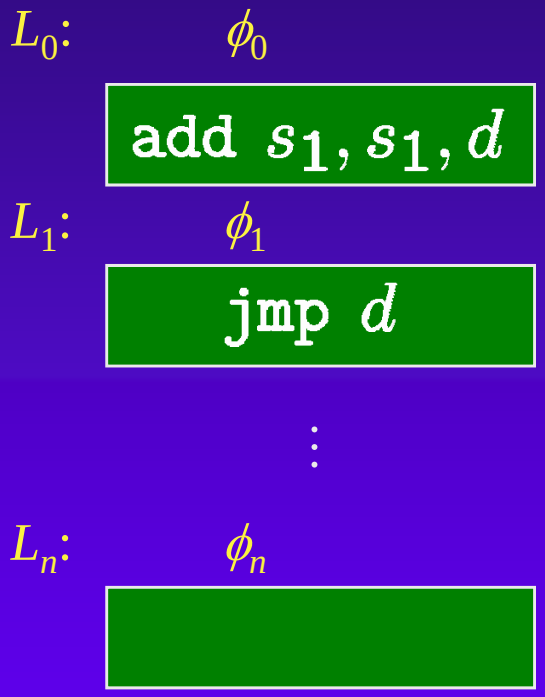
TAL type checking



$$\frac{\Gamma \vdash_b \{\phi_0\}\, B_0\, \{\phi_1\} \qquad \Gamma \vdash_b \{\phi_1\}\, B_1\, \{\phi_2\} \qquad \cdots}{\vdash_p C : \Gamma}$$

- Hoare-logic style checking: pre- and postconditions

Checking instructions



$$\frac{\phi \subset \{s_1 : \mathsf{int}\} \qquad \phi \subset \{s_2 : \mathsf{int}\} \qquad \phi' = \phi[d \mapsto \mathsf{int}]}{\Gamma \vdash_{\mathtt{i}} \{\phi\}\ \mathtt{add}\ s_1, s_2, d\ \{\phi'\}}$$

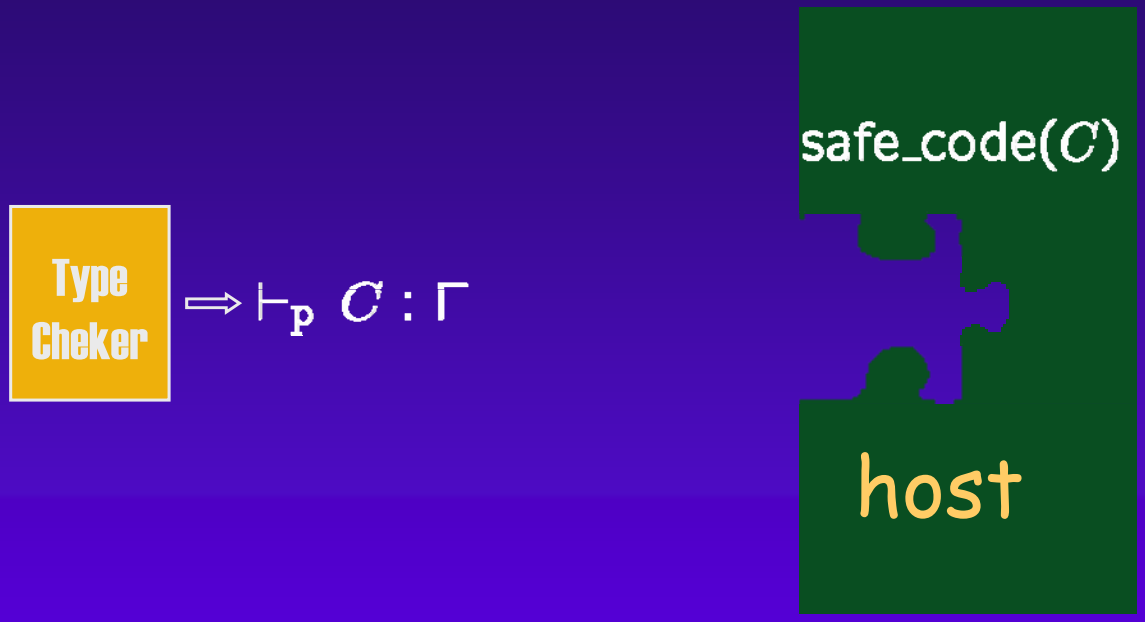
$$\frac{\Gamma(d) = \phi_d \qquad \phi \subset \phi_d}{\Gamma \vdash_{\mathtt{i}} \{\phi\}\ \mathtt{jmp}\ d\ \{\bot_\phi\}}$$

Can we trust these rules!

Can we trust these typing rules?

- For small systems, maybe yes.
- Production-scale low-level type systems
  - Huge: LTAL by Chen *et al.* has 1200 operators & rules!
  - Complex: because of intricate machine semantics
    - Think about condition code types
  - We routinely find and fix bugs in its early versions

The type-safety theorem



$$\frac{\vdash_{\mathbf{p}} C : \Gamma}{\text{safe\_code}(C)}$$

Semantic model approach

$$\frac{\vdash_{\mathtt{p}} C : \Gamma}{\mathsf{safe\_code}(C)}$$

$$\frac{\phi \subset \{s_1 : \mathsf{int}\} \qquad \phi \subset \{s_2 : \mathsf{int}\}}{\Gamma \vdash_{\mathtt{i}} \{\phi\}\ \mathbf{add}\ s_1, s_2, d\ \{\phi[d \mapsto \mathsf{int}]\}}$$

- A classic idea: give a model in some logic so that the rule can be proved as a lemma

What we need to model

Previous work
This talk

$$\frac{\vdash_{\mathtt{p}} C : \Gamma}{\mathsf{safe\_code}(C)}$$

$$\frac{\phi \subset \{s_1 : \mathsf{int}\} \qquad \phi \subset \{s_2 : \mathsf{int}\}}{\Gamma \vdash_{\mathtt{i}} \{\phi\}\,\mathtt{add}\ s_1, s_2, d\,\{\phi[d \mapsto \mathsf{int}]\}}$$

- Models for safety of code, instructions, types, …
  - [Appel & Felty 2000, Michael & Appel 2000]
- We also need models for typing judgments

$$\vdash_{\mathtt{p}} C : \Gamma \qquad\qquad \Gamma \vdash_{\mathtt{i}} \{\phi\}\,\mathtt{add}\ s_1, s_2, d\,\{\phi'\}$$

- Goal: give models to typing judgments
  - Prove all the typing rules as derived lemmas
  - Verify the type-safety theorem

Axiomatization of Sparc machine

$$\texttt{ld}\ s, d \equiv \lambda r, m, r', m'.\ r'(d) = m(r(s)) \ \wedge\ \left(\forall x \neq d.\ r'(x) = r(x)\right) \wedge\ m' = m\ \wedge \text{readable}(r(s))$$

$$(r, m) \mapsto (r', m') \equiv \ldots$$

- Our step relation is deliberately partial
  - Omit any steps that would violate the safety policy
- Mixing of machine semantics and safety policy is to follow standard practice in type theory

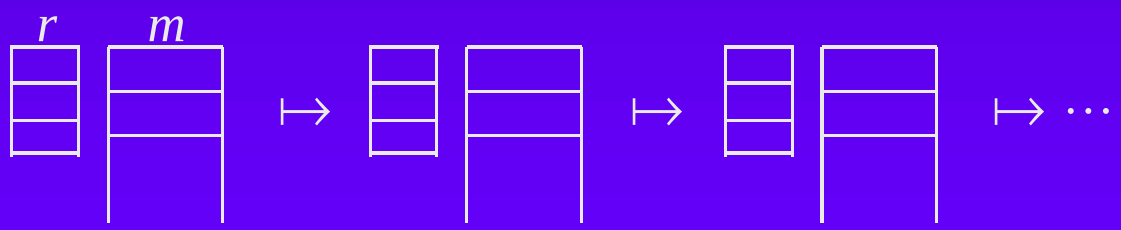
Safety definition

- A state is safe for k steps

$$\mathsf{safe\_state}(k, r, m) \equiv \forall r', m'.\ \forall j < k.\ (r, m) \mapsto^{j} (r', m') \Rightarrow \exists r'', m''.\ (r', m') \mapsto (r'', m'')$$

- Safe code

$$\mathsf{safe\_code}(C) \equiv \forall k, r, m.\ \Big(\mathsf{prog\_loaded}(m, C) \wedge r(\mathsf{pc}) = L_0 \wedge (r, m) : \phi_0\Big) \Rightarrow \mathsf{safe\_state}(k, r, m)$$



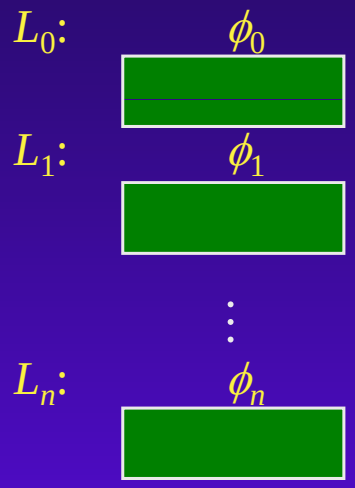
codeptr types

- Address l has type codeptr(ϕ) if it is safe to pass the control to l, provided that ϕ is satisfied

$$(m,l) :_k \mathsf{codeptr}\,(\phi) \equiv \forall r.\ r(\mathsf{pc}) = l \wedge (r,m) : \phi \Rightarrow \mathsf{safe\_state}(k,r,m)$$

  - Safety within k steps
- $(m,l) : \mathrm{codeptr}(\phi) \equiv \forall k.\ (m,l) :_k \mathrm{codeptr}(\phi)$
  - Safe for any number of steps

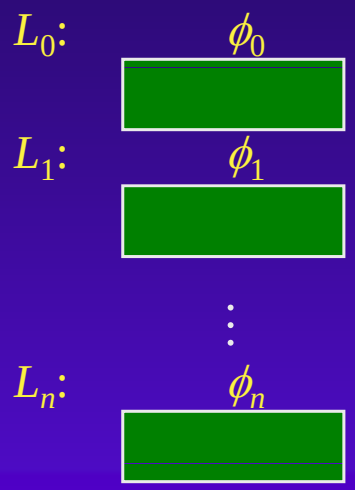
Constructing a safety proof



- The goal: safe_state(k,r,m) for any natural number k
  - $\forall\, k.\ (m,L_0) :_k \text{codeptr}(\phi_0)$
- Do it by induction
  - safe_state(0,r,m) is vacuously true
  - safe_state(k,r,m) $\stackrel{?}{\Rightarrow}$ safe_state(k+1,r,m)

    (need a stronger induction hypothesis!)

Simultaneous induction over all labels



- The goal $\forall k.\ (m, L_0) : \text{codeptr}(\phi_0)$
- Induction hypothesis: $\forall l, k.\ (m, l) :_k \text{codeptr}(\phi_l)$
  - simultaneously prove that all labels are safe for k steps

An example of the inductive case

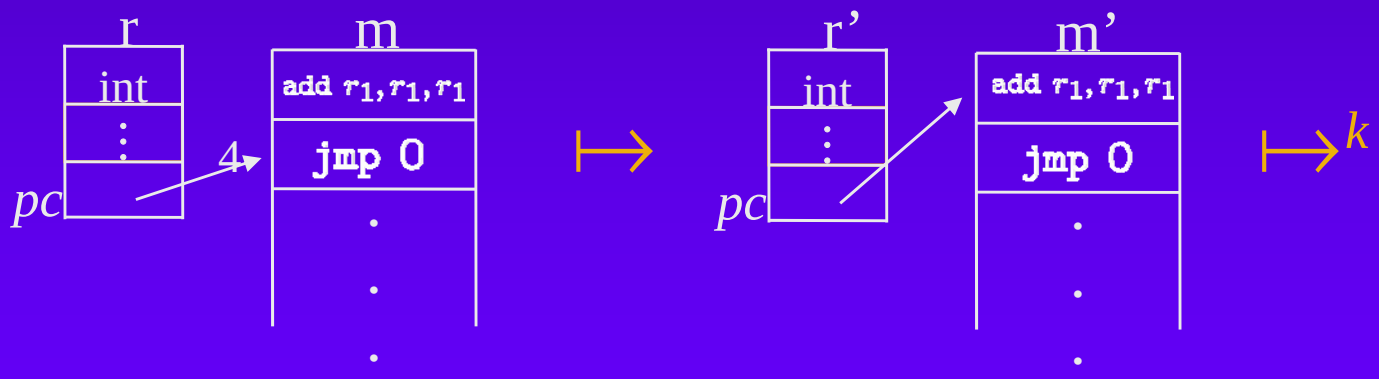
$\phi_0 = \{r_1 : \text{int}\}$

0 : `add` r_1, r_1, r_1

$\phi_4 = \{r_1 : \text{int}\}$

4 : `jmp 0`

- Prove $(m,4) :_{k+1} \text{codeptr}(\phi_4)$
- Induction hypothesis has
  - $(m,0) :_k \text{codeptr}(\phi_0)$



The model of instruction judgment

$$\Gamma \vdash_{\mathtt{i}} \{\phi\}\, i\, \{\phi'\}$$

- For any state (r,m) and k such that
  - $(r,m) : \phi$
  - Instruction i is at location l
  - $(m, l+4) :_k \text{codeptr}(\phi')$ and $m :_k \Gamma$
- Prove that $(m,l) :_{k+1} \text{codeptr}(\phi)$

$$\frac{\Gamma(d) = \phi_d \qquad \phi \subset \phi_d}{\Gamma \vdash_{\mathtt{i}} \{\phi\}\ \mathtt{jmp}\ d\ \{\bot_\phi\}}$$

The model of $\vdash_{\mathbf{p}} C : \Gamma$

$$\vdash_{\mathbf{p}} C : \Gamma \equiv \forall m. m : \Delta(C) \Rightarrow m : \Gamma$$

- $m : \Delta(C)$ is to describe the program in the memory
- $m : \Gamma$ means that the program respects all the loop invariants
- The model can be written as

$$\vdash_{\mathbf{p}} C : \Gamma \equiv \Delta(C) \subset \Gamma$$

  - If we define $\Delta(C) \subset \Gamma \equiv \forall k, m.\ m :_k \Delta(C) \Rightarrow m :_k \Gamma$

$\phi_0 = \{r_1 : \text{int}\}$

0 : add r_1, r_1, r_1

$\phi_4 = \{r_1 : \text{int}\}$

4 : jmp 0

$\{0 : \text{add } r_1, r_1 r_1,\ 4 : \text{jmp } 0\} \subset \{0 : \text{codeptr}(\phi_0),\ 4 : \text{codeptr}(\phi_4)\}$

$\Delta(C)$ $\qquad$ Γ

The type-safety theorem

$$\frac{\vdash_{\mathrm{p}} C : \Gamma \qquad \Gamma \subset \{L_0 : \mathsf{codeptr}\,(\phi_0)\}}{\mathsf{safe\_code}(C)}$$

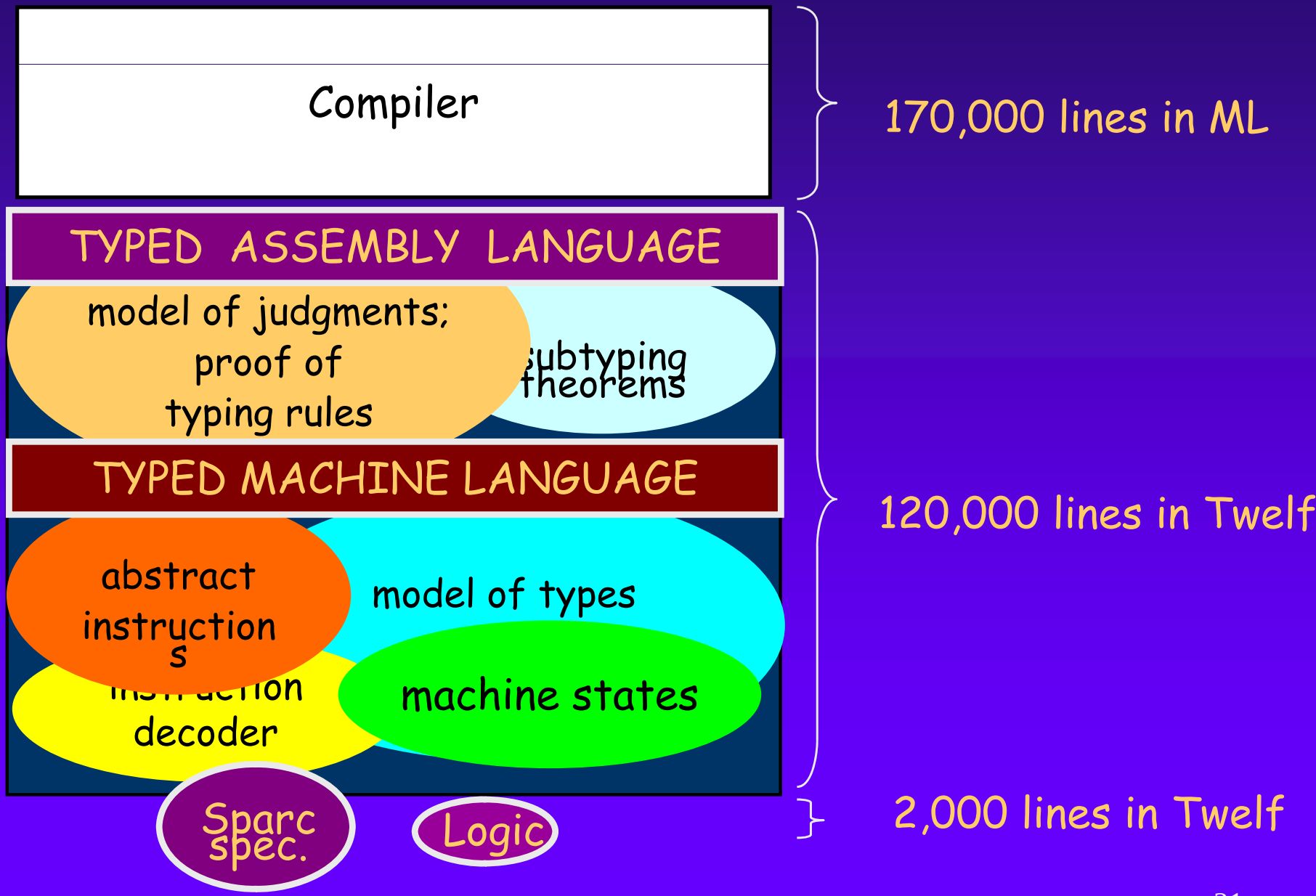
$$\mathsf{safe\_code}(C) \equiv \forall k, r, m.\ \Big(\mathsf{prog\_loaded}(m, C) \ \wedge\ r(\mathsf{pc}) = L_0 \ \wedge\ (r, m) : \phi_0\Big) \Rightarrow \mathsf{safe\_state}(k, r, m)$$

$$\dfrac{\dfrac{\dfrac{\mathsf{prog\_loaded}(m, C)}{m : \Delta(C)} \qquad \Delta(C) \subset \Gamma \qquad \Gamma \subset \{L_0 : \mathsf{codeptr}\,(\phi_0)\}}{(m, L_0) : \mathsf{codeptr}\,(\phi_0)}}{\vdots}}{\forall k.\ \mathsf{safe\_state}(k, r, m)}$$

Implementations

- Successfully defined the models of typing judgments in LTAL
- Proved the type safety theorem and the typing rules of instructions
- All the proofs are implemented in Twelf and machine checkable

FPCC system


Compiler
170,000 lines in ML
TYPED ASSEMBLY LANGUAGE
model of judgments;
proof of
typing rules
subtyping
theorems
TYPED MACHINE LANGUAGE
abstract
instruction
s
model of types
machine states
decoder
120,000 lines in Twelf
Sparc
spec.
Logic
2,000 lines in Twelf

Related work

- Proof of the type safety theorem
  - Necula had 12 pages in his thesis
  - The TAL paper by Morrisett *et al.* had 8 pages
  - Paper proofs; not machine checked
  - Not proofs about their implemented systems
- A syntactic approach to prove type soundness

  [Hamid *et al.* 2002, Crary 2003]
  - Type soundness theorem based on an abstract machine
  - A simulation relation between the abstract machine and the real machine
- Models for unstructured programs with goto statements and labels [de Bruin 1981]
  - Domain-theoretic models
  - *k*-th approximations of code behavior respects invariants