A Type System Equivalent to a Model Checker

Mayur Naik, Stanford University

Jens Palsberg, UCLA

Type Systems and Model Checking

- Prevalent approaches to program verification
- Essentially abstract interpretations
  - Types as Abstract Interpretations (Cousot [POPL 97])
  - Temporal Abstract Interpretation (Cousot & Cousot [POPL 00])
- Significant differences:

Type Systems	Model Checking
syntactic	semantic
modular	whole-program

Implications of Differences

- Type systems are good at explaining why a program was accepted
  - annotate program with *types* (keywords: syntactic, modular)
- Large body of research on explaining why a program was rejected by a type system
- Model checking is good at explaining why a program was rejected
  - provides a *counterexample* (keywords: semantic, whole-program)
- Large body of research on explaining why a program was accepted by a model checker

Motivation

- Exploring the relationship between type systems and model checking

- Developing a methodology for studying their relative expressiveness

- Sharing results between them

  - types as models (Chaki, Rehof, Rajamani [POPL 02])

  - models as types (this paper)

- Devising synergistic program analyses involving interplay between a type system and a model checker

Our Result

- A type system equivalent to a model checker for verifying temporal safety properties of imperative WHILE programs

- Model checker is conventional and may be instantiated with any finite-state abstraction scheme (e.g., predicate abstraction)

- Type system is also parametric but unconventional:
  - encodes state-transition relation of the model checker in a syntactic and modular style
  - uses function types and intersection/union types for flow-, context-, and path-sensitivity
  - uses $\top$ and $\bot$ types for checking dead code

Relationship between Type Systems and Model Checking

- Model Checker: $\{ \langle \omega_i, \omega_j \rangle \mid \omega_j \in \delta_s(\omega_i) \}$

  where ω ranges over a *finite* set of abstract contexts Ω and $\delta_s : \Omega \rightharpoonup 2^\Omega$ is abstract transfer function of s

- Type System: $s \ : \ \bigwedge_{i \in A} (\omega_i \rightarrow \bigvee_{j \in B_i} \omega_j)$

  where A and $\forall i \in A : B_i$ *finite*

- Most straightforward form of model checking corresponds to most complex form of typing

- Conventional type systems use restricted cases of this form of typing: $|A| = 1$ (no intersection types) or $\forall i \in A : |B_i| = 1$ (no union types)

Relationship between Type Systems and Model Checking

Type Checking: Is $\langle s, \omega \rangle$ well-typed?

Type Soundness: If $\langle s, \omega \rangle$ is well-typed and $\omega \cong \rho$, then $\langle s, \rho \rangle$ does not go wrong in the concrete semantics.

Model Checking: Does $\langle s, \omega \rangle$ go wrong (in the abstract semantics)?

Model Checking Soundness: If $\langle s, \omega \rangle$ does not go wrong and $\omega \cong \rho$, then $\langle s, \rho \rangle$ does not go wrong in the concrete semantics.

Equivalence Theorem: $\langle s, \omega \rangle$ is well-typed if and only if $\langle s, \omega \rangle$ does not go wrong (in the abstract semantics).

WHILE Language: Abstract Syntax

$$
\begin{array}{llll}
\text{(program)} & s & ::= & p \\
 & & | & \texttt{assume}(e) \\
 & & | & \texttt{assert}(e) \\
 & & | & s_1;\ s_2 \\
 & & | & \texttt{if}\ (*)\ \texttt{then}\ s_1\ \texttt{else}\ s_2 \\
 & & | & \texttt{while}\ (*)\ \texttt{do}\ s'
\end{array}
$$

p is an uninterpreted primitive statement

e is an uninterpreted boolean expression

$*$ denotes non-deterministic choice

Preserving Path Sensititivity

$$\texttt{if}\ (e)\ \texttt{then}\ s_1\ \texttt{else}\ s_2 \quad \equiv \quad \begin{array}{l} \texttt{if}\ (*)\ \texttt{then} \\ \qquad \texttt{assume}(e);\ s_1 \\ \texttt{else} \\ \qquad \texttt{assume}(\overline{e});\ s_2 \end{array}$$

$$\texttt{while}\ (e)\ \texttt{do}\ s' \quad \equiv \quad \begin{array}{l} \texttt{while}\ (*)\ \texttt{do} \\ \qquad \texttt{assume}(e);\ s' \\ \texttt{assume}(\overline{e}) \end{array}$$

Parameters of Model Checker

Model checker is parameterized by:

1. Finite set of abstract contexts Ω

2. For each primitive statement p: Abstract transfer function $\delta_p \in \Omega \rightarrow 2^{\Omega}$ (describing effect of p on abstract contexts)

   - δ_p is total

   - $\forall i \in \Omega : \ \delta_p(i) \neq \emptyset$

3. For each boolean expression e: Predicate $\delta_e \subseteq \Omega$ (denoting set of abstract contexts in which e is true)

Abstract Semantics of Model Checker

$$\text{(abstract state)} \quad a \quad ::= \quad \omega \mid \text{error} \mid \langle s, \omega \rangle$$

$$\begin{array}{rcll}
\langle p, \omega_k \rangle & \hookrightarrow & \omega_l & \text{if } l \in \delta_p(k) \\
\langle \texttt{assume}(e), \omega_k \rangle & \hookrightarrow & \omega_k & \text{if } k \in \delta_e \\
\langle \texttt{assume}(e), \omega_k \rangle & \hookrightarrow & \text{error} & \text{if } k \notin \delta_e \\
\langle \texttt{assert}(e), \omega_k \rangle & \hookrightarrow & \omega_k & \text{if } k \in \delta_e
\end{array}$$

$$\frac{\langle s_1, \omega \rangle \hookrightarrow \omega'}{\langle s_1; s_2, \omega \rangle \hookrightarrow \langle s_2, \omega' \rangle} \qquad \frac{\langle s_1, \omega \rangle \hookrightarrow \text{error}}{\langle s_1; s_2, \omega \rangle \hookrightarrow \text{error}} \qquad \frac{\langle s_1, \omega \rangle \hookrightarrow \langle s_1', \omega' \rangle}{\langle s_1; s_2, \omega \rangle \hookrightarrow \langle s_1'; s_2, \omega' \rangle}$$

$$\begin{array}{rcl}
\langle \texttt{if } (*) \texttt{ then } s_1 \texttt{ else } s_2, \omega \rangle & \hookrightarrow & \langle s_1, \omega \rangle \\
\langle \texttt{if } (*) \texttt{ then } s_1 \texttt{ else } s_2, \omega \rangle & \hookrightarrow & \langle s_2, \omega \rangle \\
\langle \texttt{while } (*) \texttt{ do } s', \omega \rangle & \hookrightarrow & \langle s'; \texttt{ while } (*) \texttt{ do } s', \omega \rangle \\
\langle \texttt{while } (*) \texttt{ do } s', \omega \rangle & \hookrightarrow & \omega
\end{array}$$

Model Checking

State $\langle s, \omega \rangle$ is *stuck* if $\nexists a : \langle s, \omega \rangle \hookrightarrow a$

State $\langle s, \omega \rangle$ *goes wrong* if $\exists \langle s', \omega' \rangle : (\langle s, \omega \rangle \hookrightarrow^* \langle s', \omega' \rangle$ and $\langle s', \omega' \rangle$ is stuck)

Model Checking: Given program s and abstract context ω:

Does $\langle s, \omega \rangle$ go wrong?

Type System

Syntax of Types:

$$\tau \ ::= \ \bigwedge_{i \in A}(\omega_i \rightarrow \bigvee_{j \in B_i} \omega_j)$$

$A \subseteq \Omega$ and $\forall i \in A : B_i \subseteq \Omega$ (recall that Ω is finite)

$$\top \triangleq \bigwedge \emptyset$$
$$\bot \triangleq \bigvee \emptyset$$

Type Judgment: $s \ : \ \tau$

Type Rules: Simple Statements

$$p \; : \; \bigwedge_{i \in A}(\omega_i \rightarrow \bigvee_{j \in \delta_p(i)} \omega_j) \qquad [A \subseteq \Omega]$$

$$\texttt{assume}(e) \; : \; \bigwedge_{i \in A}(\omega_i \rightarrow \omega_i) \;\wedge\; \bigwedge_{i \in B}(\omega_i \rightarrow \bot) \qquad [A \subseteq \delta_e \text{ and } B \subseteq \Omega \setminus \delta_e]$$

$$\texttt{assert}(e) \; : \; \bigwedge_{i \in A}(\omega_i \rightarrow \omega_i) \qquad [A \subseteq \delta_e]$$

Type Rules: Compound Statements

$$\frac{\begin{array}{rcl} s_1 & : & \bigwedge_{i \in A_1} (\omega_i \rightarrow \bigvee_{j \in B_i} \omega_j) \\ s_2 & : & \bigwedge_{i \in A_2} (\omega_i \rightarrow \bigvee_{j \in B'_i} \omega_j) \end{array}}{s_1 ; s_2 \ : \ \bigwedge_{i \in A} (\omega_i \rightarrow \bigvee_{k \in \bigcup \{ B'_j \mid j \in B_i \}} \omega_k)} \qquad [A \subseteq A_1 \text{ and } \bigcup_{i \in A} B_i \subseteq A_2]$$

$$\frac{\begin{array}{rcl} s_1 & : & \bigwedge_{i \in A_1} (\omega_i \rightarrow \bigvee_{j \in B_i} \omega_j) \\ s_2 & : & \bigwedge_{i \in A_2} (\omega_i \rightarrow \bigvee_{j \in B'_i} \omega_j) \end{array}}{\texttt{if } (*) \texttt{ then } s_1 \texttt{ else } s_2 \ : \ \bigwedge_{i \in A} (\omega_i \rightarrow \bigvee_{j \in B_i \cup B'_i} \omega_j)} \qquad [A \subseteq A_1 \text{ and } A \subseteq A_2]$$

$$\frac{s' \ : \ \bigwedge_{i \in A'} (\omega_i \rightarrow \bigvee_{j \in B_i} \omega_j)}{\texttt{while } (*) \texttt{ do } s' \ : \ \bigwedge_{i \in A} (\omega_i \rightarrow \bigvee_{k \in \mu X.(\{i\} \cup \{B_j \mid j \in X\})} \omega_k)} \qquad [A \subseteq A' \text{ and } \bigcup_{i \in A} B_i \subseteq A]$$

Type Checking

$$\frac{s \ : \ \bigwedge_{i \in A}(\omega_i \rightarrow \bigvee_{j \in B_i} \omega_j)}{\langle s, \omega_k \rangle \text{ is well-typed}} \qquad [k \in A]$$

Type Checking: Given program s and abstract context ω:

Is $\langle s, \omega \rangle$ well-typed?

Equivalence Result

Equivalence Theorem: $\langle s, \omega \rangle$ is well-typed if and only if $\langle s, \omega \rangle$ does not go wrong

From Type Checking to Model Checking:

Type soundness (progress + type preservation) with respect to the abstract semantics of the model checker

From Model Checking to Type Checking:

Building a type derivation from the model constructed by the model checker

Example 1

$$s \triangleq \{\,\text{lock}_1();\text{lock}_2()\,\} \quad \text{where} \quad \text{lock}() \triangleq \{\,\mathtt{assert}(v=\mathtt{U});v:=\mathtt{L}\,\}$$

Suppose $\Omega = \{\mathsf{v}=\mathsf{U},\mathsf{v}=\mathsf{L}\}$

- $\langle s,\mathsf{v}=\mathsf{U}\rangle$ goes wrong in the model checker's abstract semantics
- $\langle s,\mathsf{v}=\mathsf{U}\rangle$ is not well-typed in the type system

Example 2

$$s \triangleq \mathsf{lock}_1(); \texttt{assume}(false); \mathsf{lock}_2()$$

Suppose $\Omega = \{\mathsf{v}{=}\mathsf{U}, \mathsf{v}{=}\mathsf{L}\}$

- $\langle s, \mathsf{v}{=}\mathsf{U}\rangle$ does not go wrong in the model checker's abstract semantics

- $\langle s, \mathsf{v}{=}\mathsf{U}\rangle$ is well-typed in the type system:

$$\dfrac{\dfrac{\mathsf{lock}_1() \ : \ \mathsf{v}{=}\mathsf{U} \rightarrow \mathsf{v}{=}\mathsf{L} \quad \texttt{assume}(false) \ : \ \mathsf{v}{=}\mathsf{L} \rightarrow \bot}{\mathsf{lock}_1(); \texttt{assume}(false) \ : \ \mathsf{v}{=}\mathsf{U} \rightarrow \bot} \quad \mathsf{lock}_2() \ : \ \top}{s \ : \ \mathsf{v}{=}\mathsf{U} \rightarrow \bot}$$

Example 3

$$s \triangleq \{\texttt{while}\ (*)\ \texttt{do}\ \{\texttt{assume}(i \neq 2); i := i+1\}\}; \texttt{assume}(i = 2)$$

Suppose $\Omega = \{\mathsf{i}=0,\ \mathsf{i}=1, \mathsf{i}=2\}$

- $\langle s, \mathsf{i}=0\rangle$ does not go wrong in the model checker's abstract semantics
- $\langle s, \mathsf{i}=0\rangle$ is well-typed in the type system:

$$\dfrac{\dfrac{\dfrac{\begin{array}{rcl}\texttt{assume}(i \neq 2) & : & \mathsf{i}=0 \to \mathsf{i}=0 \ \wedge\ \mathsf{i}=1 \to \mathsf{i}=1 \ \wedge\ \mathsf{i}=2 \to \bot \\ i := i+1 & : & \mathsf{i}=0 \to \mathsf{i}=1 \ \wedge\ \mathsf{i}=1 \to \mathsf{i}=2\end{array}}{\begin{array}{c}\texttt{assume}(i \neq 2);\ i := i+1\ : \\ \mathsf{i}=0 \to \mathsf{i}=1 \ \wedge\ \mathsf{i}=1 \to \mathsf{i}=2 \ \wedge\ \mathsf{i}=2 \to \bot\end{array}}}{\begin{array}{c}\texttt{while}\ (*)\ \texttt{do}\ \{\ \texttt{assume}(i \neq 2);\ i := i+1\ \}\ : \\ \mathsf{i}=0 \to (\mathsf{i}=0 \ \vee\ \mathsf{i}=1 \ \vee\ \mathsf{i}=2)\end{array}} \quad \begin{array}{c}\texttt{assume}(i = 2)\ : \\ \mathsf{i}=0 \to \bot \ \wedge \\ \mathsf{i}=1 \to \bot \ \wedge \\ \mathsf{i}=2 \to \mathsf{i}=2\end{array}}{s\ :\ \mathsf{i}=0 \to \mathsf{i}=2}$$

Related Work: Type Systems for Temporal Safety Properties

- CQual (Foster, Terauchi, Aiken [PLDI 02])
- Refinement Types (Mandelbaum, Walker, Harper [ICFP 03])
- Resource Usage Analysis (Igarashi & Kobayashi [POPL 02])
- Vault (DeLine & Faehndrich [PLDI 01])
- Xanadu (Xi [LICS 00])

In our type system: $s \;:\; \mathsf{i}=0 \rightarrow \mathsf{i}=2 \;\wedge\; \mathsf{i}=1 \rightarrow \mathsf{i}=2 \;\wedge\; \mathsf{i}=2 \rightarrow \mathsf{i}=2$

In CQual:

$$s \;:\; \forall c, c'.\; (ref(l), [l \mapsto int(c)]) \rightarrow (ref(l), [l \mapsto int(c')]) \;/\; \{(c=0 \Rightarrow c'=2), (c=1 \Rightarrow c'=2), (c=2 \Rightarrow c'=2)\}$$

Related Work: Equivalence Results

- Type systems and control-flow analysis for functional languages
  - Amadio-Cardelli type system $\equiv$ 0-CFA-based safety analysis (Palsberg & O'Keefe [POPL 95])
  - Amadio-Cardelli type system $\equiv$ a form of constrained types (Palsberg & Smith [TOPLAS 96])
  - 0-CFAs $\equiv$ type systems with recursive types and subtyping (Heintze [SAS 95])
  - finitary polyvariant CFA $\equiv$ type system with finitary polymorphism (Amtoft & Turbak [ESOP 00], Palsberg & Pavlopoulou [POPL 00])

- Data-flow analysis and model checking for imperative languages (Steffan [TACS 91], Schmidt & Steffan [SAS 98], Schmidt [POPL 98])

Conclusions and Future Work

- Equivalence result highlights essence of relationship between type systems and model checking

- Limitations:
  - Lacks support for higher-order functions, objects, and concurrency
  - Type system not suitable for human reasoning

Explore these issues in the context of specific verification problems, e.g., infer Abadi-Flanagan-style types from models of concurrent Java programs in the context of verifying race-freedom (Agarwal & Stoller [VMCAI 04]).