Program analysis

“process of automatically analyzing the behavior of computer programs”

- all kinds of beneficial effects...
Program analysis

“process of automatically analyzing the behavior of computer programs”

- all kinds of beneficial effects...

Rice’s theorem

All non-trivial, semantic properties of programs are undecidable.
The good, the bad, and the ones we can’t figure out
The good, the bad, and the ones we can’t figure out
Two out of three ain’t bad
Sacrifice soundness / underapproximation

- testing (and “testing can be formal too”)
- run-time verification
Sacrifice soundness / underapproximation

- testing (and “testing can be formal too”)
- run-time verification
Of course: one can do better than just that. . .

- avoiding redundant exploration (POR)
- prioritizing
  - preemption bounding
- symbolic execution
- combination with (other) static analysis
Giving up on completeness
Giving up on completeness

false positives
But how?

\[
\begin{align*}
\text{if } x=y & \text{ then } \text{skip} \quad \text{else } x:=x+1 \\
\text{Assume } (x, y) & \in \{(0, 0), (1, 1), (0, 1)\}
\end{align*}
\]
But how?

```plaintext
if x=y then skip else x:=x+1
```

Assume \((x, y) \in \{(0, 0), (1,1),(0,1)\}\)
Abstraction

- abstraction $\Rightarrow$ overapproximation $\Rightarrow$ false positives
- lumping different “elements” together (values, program points ...)
- symbolic representation
- abstract interpretation
- data flow analysis
Data flow analysis

- abstraction: “Ampere”
- circuit laws (Kirchhoff)
- stationary solution

- abstraction: sets of values...
- data flow constraints, transfer functions
- fixpoint ($\mu$ or $\nu$)

[Diagram of a circuit with labels 1mA, 4kΩ, 2V, t₀, v₁, i₁, 4V, 4nF]
What’s a type?
What’s a type?

A **union** is a special data type available in C that allows to store different data types in the same memory location. You can define a union with many members, but only one member can contain a value at any given time. Unions provide an efficient way of using the same memory location for multiple-purpose.
What’s a type?

- directive for memory allocation
- an object in a category? homotopy?
- an abstraction?
- formula in a HO constructive logic?
Types, flows, and effects

\[ t : T \]
Types, flows, and effects

\[ t : \text{Int} \]

What?

if \( t \) terminates, 
Int-value
Types, flows, and effects

\[ t : T^{++} \]

<table>
<thead>
<tr>
<th>What?</th>
<th>From where?</th>
</tr>
</thead>
<tbody>
<tr>
<td>if ( t ) terminates, Int-value</td>
<td>data flow</td>
</tr>
</tbody>
</table>
Types, flows, and effects

$t : T^++ :: \varphi$

<table>
<thead>
<tr>
<th>What?</th>
<th>From where?</th>
<th>During?</th>
</tr>
</thead>
<tbody>
<tr>
<td>if $t$ terminates, Int-value</td>
<td>data flow</td>
<td>effects, <em>while</em> executing</td>
</tr>
</tbody>
</table>

- all of it: more or less approximative
Type systems & type checking

\[ \Gamma \vdash t : T \]

- derivation systems

\[
\begin{align*}
A, B &\vdash A \\
A, B &\vdash B \\
\hline
A, B &\vdash A \land B \\
\hline
A &\vdash B \rightarrow A \land B \\
\hline
\vdash A \rightarrow B \rightarrow A \land B
\end{align*}
\]
Type systems & type checking

\[ \Gamma \vdash t : T \]

- derivation systems

\[
\begin{align*}
x_1 : T_1, x_2 : T_2 & \vdash x_1 : T_1 & x_1 : T_1, x_2 : T_2 & \vdash x_2 : T_2 \\
\hline
x_1 : T_1, x_2 : T_2 & \vdash (x_1, x_2) : T_1 \times T_2 \\
\hline
x_1 : T_1 & \vdash \lambda x_2 : T_2.(x_1, x_2) : T_2 \rightarrow T_1 \times T_2 \\
\hline
\vdash \lambda x_1 : T_1. \lambda x_2 : T_2.(x_1, x_2) : T_1 \rightarrow T_2 \rightarrow T_1 \times T_2
\end{align*}
\]
Type checking

\[ \Gamma \vdash t : T \]

\[ \Gamma \vdash t : T \]
Type inference

\[ \Gamma \vdash t : ? \]

\[ \Gamma \vdash t : ? \]
Type inference

\[ \Gamma \vdash t : ? \]

\[ \Gamma \vdash t : ? \]
Type inference

- unification
- decidability?
- cf. synthesized and inherited attributes
Type inference

- **unification**
- decidability?
- cf. synthesized and inherited attributes
And for flows and effects?

\[ \hat{\Gamma} \vdash t : \hat{T} :: \varphi \]
And for flows and effects?

\[ C; \hat{\Gamma} \vdash t : \hat{T} :: \varphi \]

- adding **constraints**
- for flows: *simple* constraints
Correctness

Milner’s dictum ("type safety" / "static typing")
Well-typed programs cannot go wrong!
Correctness

Milner’s dictum ("type safety" / "static typing")

Well-typed programs cannot go wrong!
Correctness

Milner’s dictum ("type safety" / "static typing")
Well-typed programs cannot go wrong!
Context-sensitivity

- treat function calls “properly” (= dependent on call-site)
Context-sensitivity

- treat function calls “properly” (= dependent on call-site)
Context-sensitivity

- treat function calls “properly” (= dependent on call-site)
Context-sensitivity

- treat function calls “properly” (= dependent on call-site)
Compositional accounts and “polymorphism”

- context-sensitive: function analysis distinguishing different call-sites

**Constrained universally quantified types**

\[ \forall Y . \hat{T} \]

cf.

- *type schemes* in ML-polymorphismism and
- bounded quantification \( \forall Y \leq T_1.T_2 \) in \( F_{\leq} \ldots \)
Compositional accounts and “polymorphism”

- context-sensitive: function analysis distinguishing different call-sites

**constrained universally quantified types**

\[
\forall Y : C.\hat{T}
\]

cf.

- *type schemes* in ML-polymorphismism and
- bounded quantification \( \forall Y \leq T_1.T_2 \) in \( F_\leq \) ...
Program analysis

- types: basically single threaded
- structural analysis of program code
Concurrency and analysis

New kinds of errors

- races
- deadlocks
- starvation
- ...

Analyses get more hairy

- reproducability ("Heisenbugs")
- interference vs. isolation
- state space explosion problem
Illustration: Deadlock analysis

- given: multi-threaded calculus
- types and effects for lock interaction

\[
\begin{align*}
Y & ::= \varrho \mid X \\
r & ::= \varrho \mid \{\pi\} \mid r \sqcup r \\
\hat{T} & ::= B \mid Lr \mid \hat{T} \overset{\varphi}{\rightarrow} \hat{T} \\
\hat{\sigma} & ::= \forall\vec{Y}:C. \hat{T} \overset{\varphi}{\rightarrow} \hat{T} \mid \hat{T} \\
C & ::= \emptyset \mid \varrho \sqsubseteq r, C \mid \epsilon \sqsubseteq \varphi, C
\end{align*}
\]

- type-level variables
- lock/label sets
- types
- type schemes
- simple constraints
Deadlocks (2)

- two level approach
  - local type and effect system
  - global state exploration (à la model checking)
- flows: tracing lock instances (rudimentary *alias* analysis)

### Processes as effects

Abstracting approximative lock interaction of one thread into an “abstract process” (as in process algebra)

- correctness: deadlock-sensitive *simulation*
- further abstractions: bounded call stack
Correctness, Compositionality, Concurrency
Facets of formal program analysis

Introduction
Types & more
Comp. & conc.
Sel. contributions

Type & effect systems

- Taint analysis for Go
- Safe nested transactions
- Resource analysis for join synchronization
- Poly. behavior inference for deadlocks/races

Automation
Soundness
Completeness
Proof systems for program verification

- Full abstraction
- Hybrid system verification
- Inheritance of proofs
- Lazy behavioral subtyping
- Comp. verification of multithreaded oo programs

Correctness, Compositionality, Concurrency
Facets of formal program analysis

Martin Steffen

Introduction
Types & more
Comp. & conc.
Sel. contributions
Proof systems for program verification

Correctness, Compositionality, Concurrency
Facets of formal program analysis

Martin Steffen

Introduction
Types & more
Comp. & conc.
Sel. contributions

Full abstraction
Hybrid system verification
Inheritance of proofs
Lazy behavioral subtyping
Comp. verification of multithreaded oo programs
Run-time verification, testing, BMC

- DTrace + LTL monitoring
- Model testing modulo congruence
- Test generation from traces
- Bounded MC for hybrid systems
railway design verification (with datalog)
full abstraction
protocol verification (model checking, SDL) and engineering
parametric model checking (rewriting theory, transducers)
timed ambients & $\pi$-calculus (semantics, assumption-commitment type system for resources)
semantics of weak memory models
Petri-net semantics for concurrent actor language
Kudos

Slides were done with emacs org-mode, \LaTeX, TikZ (and a bit Lilypond).
Introduction

Types & more

Comp. & conc.

Sel. contributions

References I

Bibliography


References II


References III


