Soundness is not Sufficient*

Fritz Henglein
DIKU

DANSAS, 2013-08-23
Odense

Disclaimer

• This is an idea (= rambling) talk.

• The ideas and the intentions behind them are important, I believe.

• The technical definitions may not be the definite ones (yet).

• I have likely overlooked some results of yours -- tell me.

• Ask, comment, interrupt any time.

• I’ll skip slides
Goals

• Propose informal criteria for what a static analysis should satisfy to warrant being called a “good” static analysis.

• Propose technical criteria for capturing some aspects of the informal criteria

• Identify questions for further work, both conceptual and technical.
A **program property** is a predicate on programs.

A program property $P$ is **semantic** (**extensional**) if

$$ p \equiv q \Rightarrow (P(p) \iff P(q)) $$

A program property $P$ is **trivial** if

$P(p)$ for all $p$, or $\neg P(p)$ for all $p$. 

Behavioral equivalence
Rice’s Curse

Theorem:
Let \( L \) be a Turing-complete programming language, \( P \) a nontrivial semantic program property. Then \( P \) is undecidable.

Rice, Classes of recursively enumerable sets and their decision problems, Trans. AMS 1953
Rice’s Curse, pictorially

$P$ holds if $p \equiv q \neq p' \equiv q'$

$P$ does not hold

$P$ is not decidable!
Rice’s Curse: Example

Normalizing $\lambda$-terms ($N$)

Corollary: $N$ is not decidable!

Can we approximate it?

semantic and nontrivial
Static analysis

- Given:
  - $P$: Extensional program property
  - $(S, S')$: Static analysis for $P$

- We want of $(S, S')$:
  - **Soundness**: $S \subseteq P$, $S' \subseteq \neg P$

  Is that sufficient? No, we also want...

- **Goodness**

What does “good” mean??
Goodness characteristics

- **Usefulness:**
  - Has some effective use, fitness for a purpose

- **Declarative specification:**
  - Separation of **what** the analysis computes from **how** it computes it (the particular algorithm[s] used)
Goodness characteristics

• **Unimprovability:**
  • Can’t get **better** approximation at **lower** computational cost

• **Predictability:**
  • Predictability under certain, specified program transformations and changes
Goodness characteristics

• Compositional certification
  • Explicit, modular (syntax-oriented), efficiently checkable logical explanation of analysis results

• Constructive interpretation
  • Operational interpretation of certificate, not just of yes/no answer
Goodness characteristics

- **Adaptiveness:**
  - Easy instances are handled efficiently
  - Hard instances may take more time.

- **Parameter sensitivity**
  - Scale well with parameter, which captures expectations on input distribution.
Static Analysis for $N$

- Consider normalizability of $\lambda$-terms.
- Is System $F$ typability a “good” static analysis for $N$?
System F for N

• Sound? ✔
• Declarative? ✔
• Compositionally certified? ✔
• Useful? ✔
• Predictability properties? (✔)
• Unimprovability? Hmm...
Static Analysis for N

Theorem: F is undecidable

Nontrivial, not semantic (as we'd expect from an analysis)

Wells, Typability and Type Checking in the Second-Order λ-Calculus Are Equivalent and Undecidable, LICS 1994
System F for N: Improvability

• Acceptable for System F (as a static analysis for N) to be undecidable, as long as there is no better approximation of N that is decidable (more efficient).
Unimprovability via separability

• A static analysis \((S, S')\) for \(P\) is **improvable** if there exists \((T, T')\) such that:
  
  • \(S \subseteq T \subseteq P, S' \subseteq T' \subseteq \neg P, \text{ and} \)
  
  • \(T \text{ and } T'\) have “lower” (structural) complexity than \(S\) and \(S'\); e.g. \(S\) is NP-hard, but \(T\) is in \(P\) (with \(S' = T' = \emptyset\)).

**Property of analysis, not any particular algorithm**
Recursive inseparability

Definition:
Let \( A \subseteq P \). A is recursively inseparable from \( P \) if there is no \( B \) such that \( A \subseteq B \subseteq P \) and \( B \) is decidable ("recursive").

Is \( F \) recursively inseparable from \( N \)?
Is F recursively inseparable from N?

- The answer is...

- **We don’t know!**
  - Does not follow from Well’s proof
  - We don’t know whether F is improvable:
    - There may be a (type) system out there that extends System F, guarantees \( N \) and is decidable.

(I don’t believe it is improvable)
Another analysis for N

Theorem: $F_{\omega^I}$ is undecidable

Urzyczyn, Type reconstruction in $F_\omega$, MSCS 1997
Another analysis for $N$

**Theorem**: $F^{(l)}_{\omega}$ is recursively inseparable from $N$

Follows from proof method: TM simulation
Theorem: SCT is decidable.
(Complexity: PSCACE-complete)

Bohr, Jones, Termination analysis of the untyped lambda-calculus, RTA 2004
An analysis for type error freeness

System F(1) typable
ML goodness

- **Invariant** under let-reduction:
  \[ \text{ML}(\text{let } x = e \text{ in } e') \Leftrightarrow \text{ML}(e'[e/x]) \]

- **Preservation** under \( \beta \)-reduction:
  \[ \text{ML}((\lambda x.e)e') \Rightarrow \text{ML}(e[e'/x]) \]

- **Preservation** under eta-reduction:
  \[ \text{ML}(\lambda x.ex) \Rightarrow \text{ML}(e) \]

- ML is invariant under arbitrary unfolding (inlining) and folding (refactoring) of (nonrecursive) definitions
ML typability as static analysis for type error freeness

- Is ML typability improvable?
ML typability as static analysis for type error freeness

Theorem: Let $ML \subseteq B \subseteq T$. Then $B$ is \textit{DEXPTIME}-hard.

Henglein, A Lower Bound for Full Polymorphic Type Inference: Girard-Reynolds Typability is \textit{DEXPTIME}-hard, Utrecht U.TR RUU-CS-90-14, 1990
mVFA
(0CFA in direct style)

Build graph with flow and tree edges. One node per subexpression, plus some extra ones ($\lambda^-$).

1. Base flow rules, resulting in graph $G$: 
mVFA
0CFA in direct style

$O(n)$ nodes
$O(n)$ edges
mVFA
0CFA in direct style

2. Closure rule:
**mVFA**
0CFA in direct style

**Algorithm:**
Close base graph under closure rule, resulting in graph G.
mVFA
0CFA in direct style

**Theorem**: mVFA can be implemented in time $O(d m^* + p n + q)$, where
- $n$: number of nodes
- $d$: maximum outdegree of nodes in $G$,
- $m^*$: number of flow edges in $G^*$ (flow-transitive closure of $G$),
- $p$: number of closure rule applications.
- $q$: number of reachability queries

sVFA
Simple closure analysis

1. Base rules: As for mVFA
2. Closure rule:

- Top-level directional flow!
- Only difference!
- Undirected! (= both directions)
sVFA
Simple closure analysis

**Algorithm:**
Close base graph under closure rule by unification closure, using union/find data structure.
**sVFA**
Simple closure analysis

**Theorem:** sVFA can be implemented in time $O(n \alpha(n,n) + q n)$, where

- $\alpha(m,n)$: inverse Ackerman function
- $q$: number of reach set queries

Henglein, Simple Closure Analysis, TOPPS TR D-193, 1992
sVFA
Simple closure analysis

• Very fast in practice

• Applications:
  • Binding-time analysis
    Henglein, Efficient Type Inference for Higher-Order Binding-Time Analysis, FPCA 1991
  • Dynamic type inference for Scheme
    Henglein, Global tagging optimization by type inference, LFP 1992
  • Closure analysis in Similix

• No significant reduction in precision vis a vis mVFA observed

• Flows are not undirectional (“equational”)

Henglein, Efficient Analysis for Realistic Off-Line Partial Evaluation, JFP 1993
sVFA predictability

- sVFA is invariant under
  - linear beta-reduction
  - eta-reduction (for pure $\lambda$-terms)
sVFA predictability

Theorem:
sVFA-reachability is $P$-complete
Van Horn, Mairson, Flow Analysis, Linearity, and PTIME, SAS 2008

Not a corollary. Follows from proof method used: invariance under linear $\lambda$-term reduction

Theorem:
Let $B$ be such that $\text{sVFA} \subseteq B \subseteq R$, where $R$ is semantic (un)reachability. Then $B$ is $P$-hard.
Adaptiveness

• Assume $S_0 \subseteq S_1 \subseteq P$, with algorithms $A_0, A_1$ for $S_0, S_1$, respectively. ($S_0' = S_1' = \emptyset$)

• $A_1$ is naturally adaptive over $A_0$ if its (time) complexity is as good as the complexity of $A_0$ on instances from $S_0$, without explicitly invoking $A_0$

• $A_1$ is allowed to take substantially more time than $A_0$ on instances outside $S_0$. (where $A_0$ and $A_1$ give different results).
Adaptiveness

- **Intuition:** A static analysis algorithm should *not be slower* on instances where a less precise analysis algorithm manages to compute the semantically *correct* result (on “easy instances”).
Questions

• Are the various $k$CFA-algorithms adaptive over $sVFA$?

• Is (functional) $k$CFA improvable for $k \geq l$?

• Is SCT improvable? How predictable is it?

• ...
End of talk