Soundness is not Sufficient*

Fritz Henglein
DIKU

WG2.8, 2014-08-13
Estes Park, CO

*Previously presented at Shonan Village Workshop on Automated Techniques for Higher-Order Program Verification, Sep. 2011, and DANSAS '13, Aug. 2013

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 (which ones I don't know yet)

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.

Program property

- A **program property** is a predicate on programs.
- A program property P is **semantic** (**extensional**) if

$$p \cong q \Rightarrow (P(p) \Leftrightarrow P(q))$$

Behavioral equivalence

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

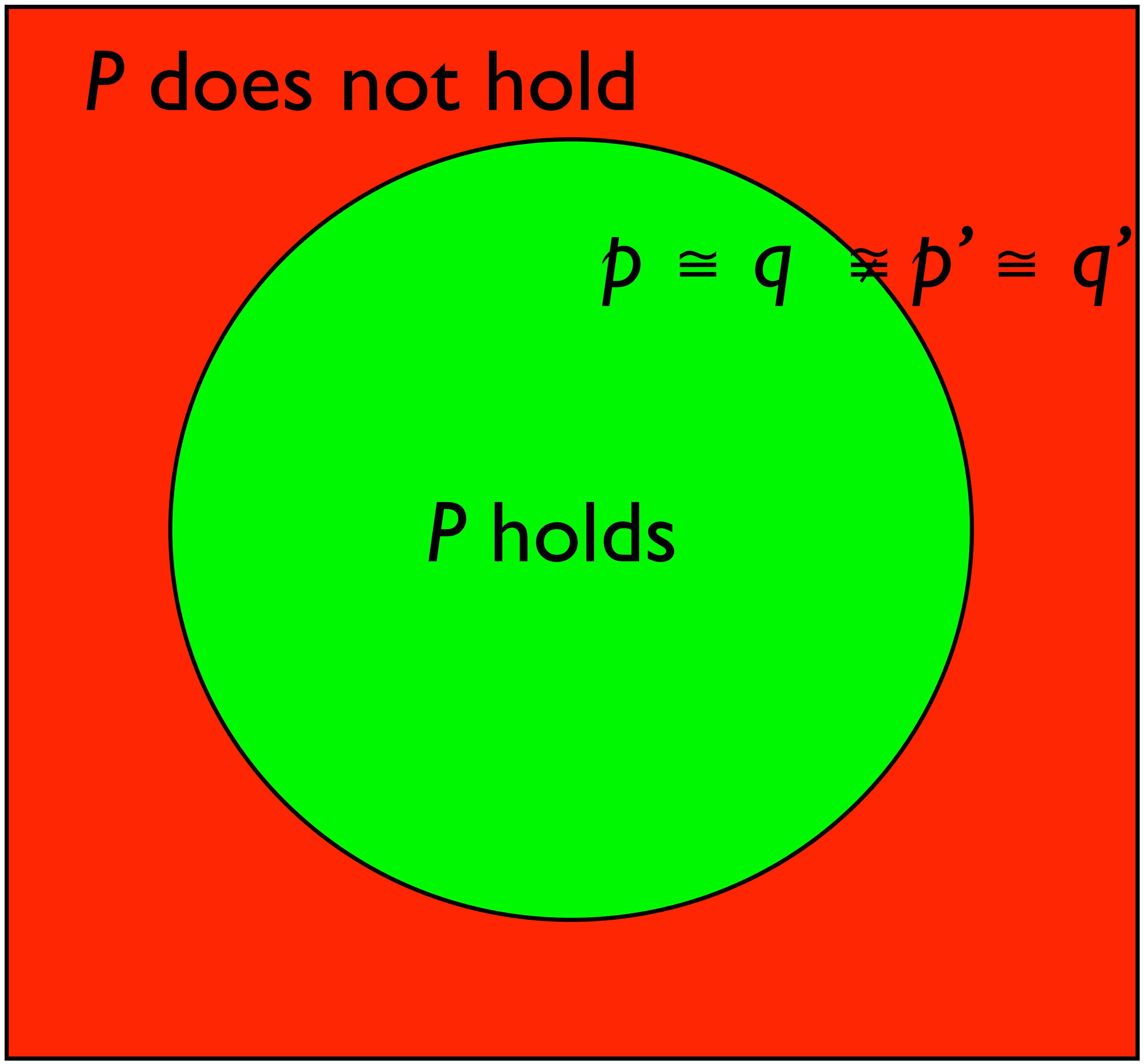
  $P(p)$ for all p, or $\neg P(p)$ for all p.

Rice's Curse

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

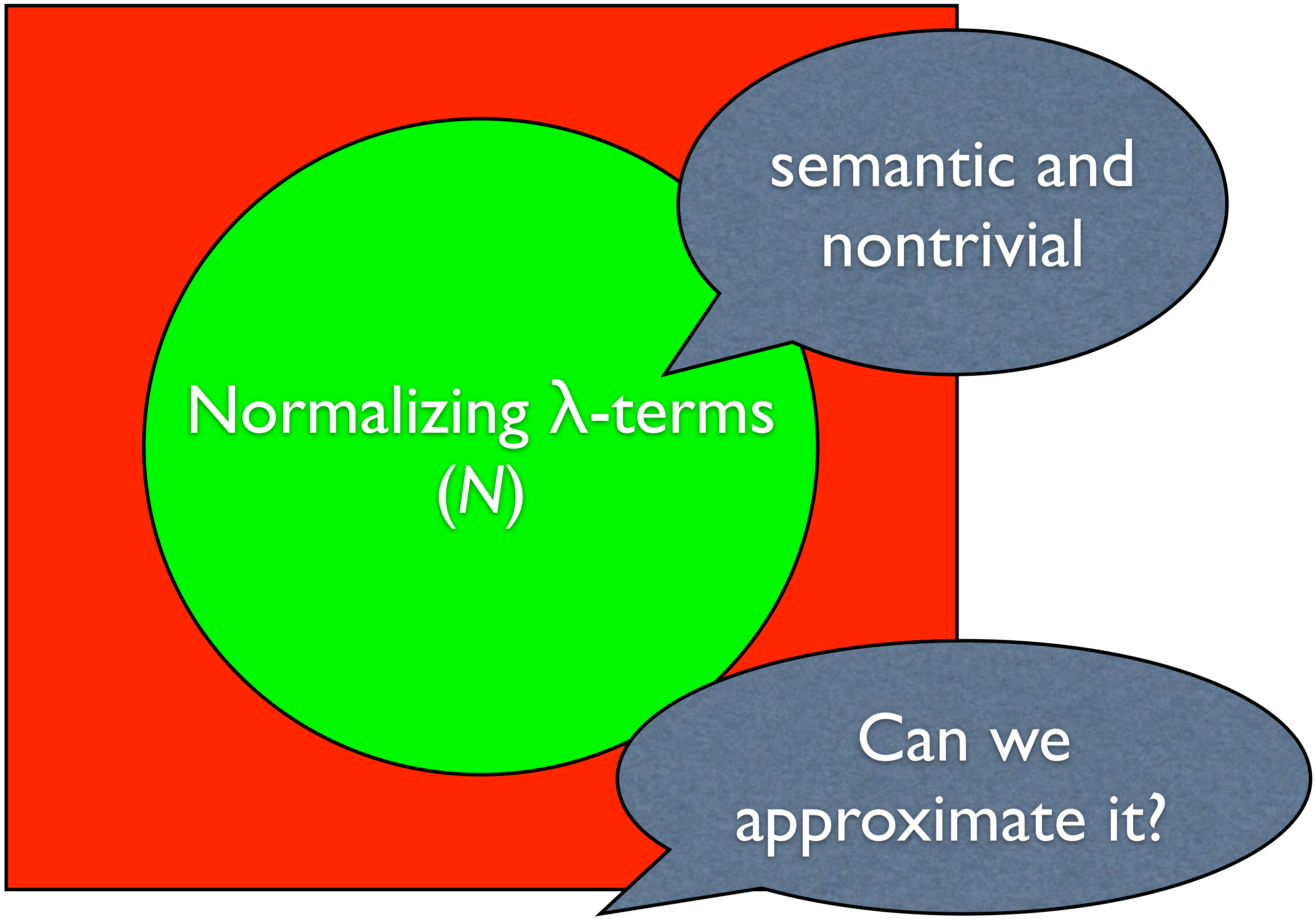
Rice, Classes of recursively enumerable sets and their decision problems, Trans. AMS 1953

Rice's Curse, pictorially



P is not decidable!

Rice's Curse: Example



Corollary: *N* is not decidable!

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

Goodne
character

Algorithm need not be compositional, only its result

- **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 characteris

Properties of particular **algorithm A** implementing an **analysis S**

- **Adaptiveness:**
  - Easy instances are handled efficiently
  - Hard instances may take more time.
- **Parameter sensitivity**
  - Scale well with parameter, which captures expectations on input distribution.

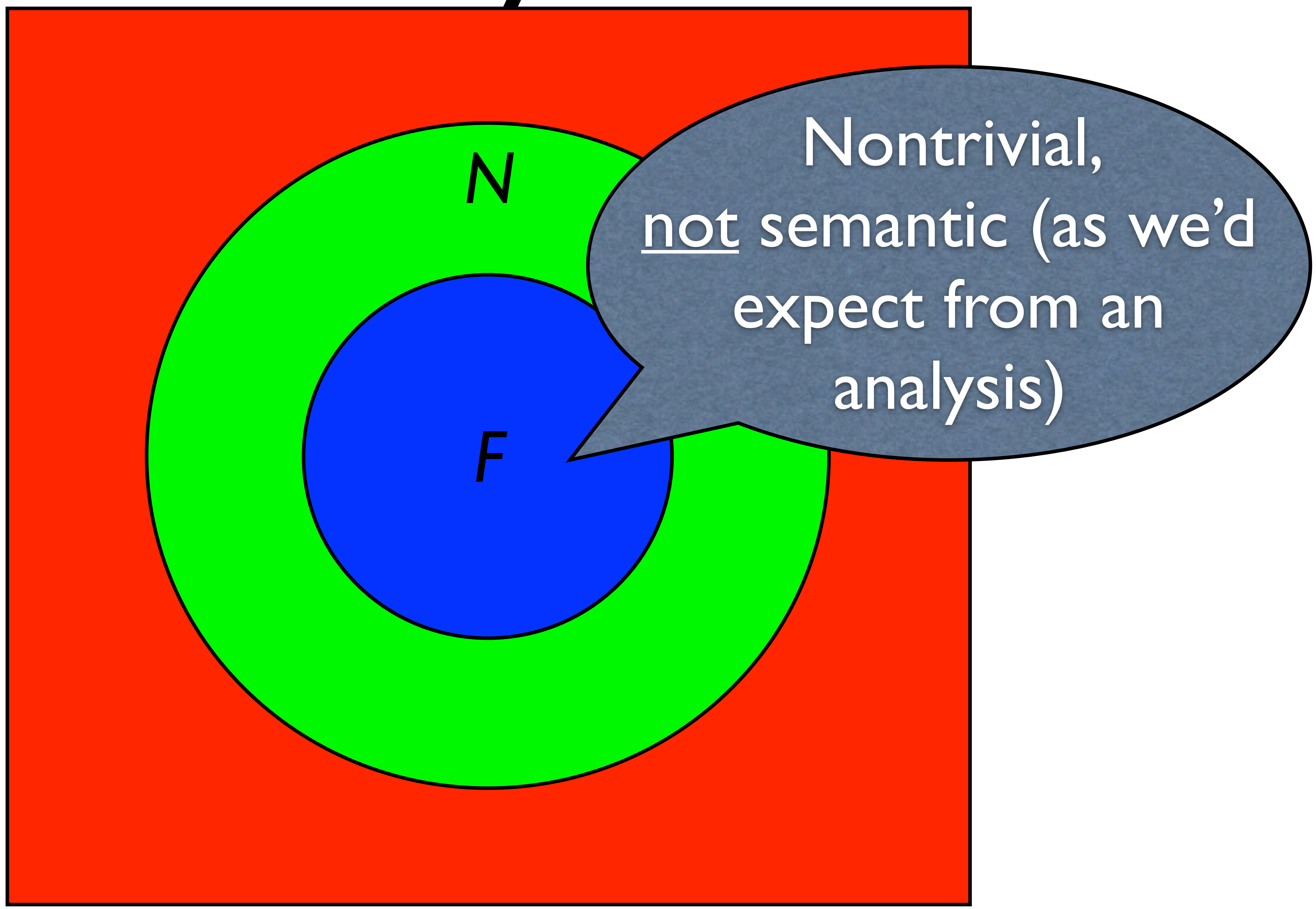
Static Analysis for *N*

- Consider normalizability of λ-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

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

Property of analysis, not any particular algorithm

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

Recursive inseparability

Classical definition in recursion theory: "... from *complement of* P if there is no B..."

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").

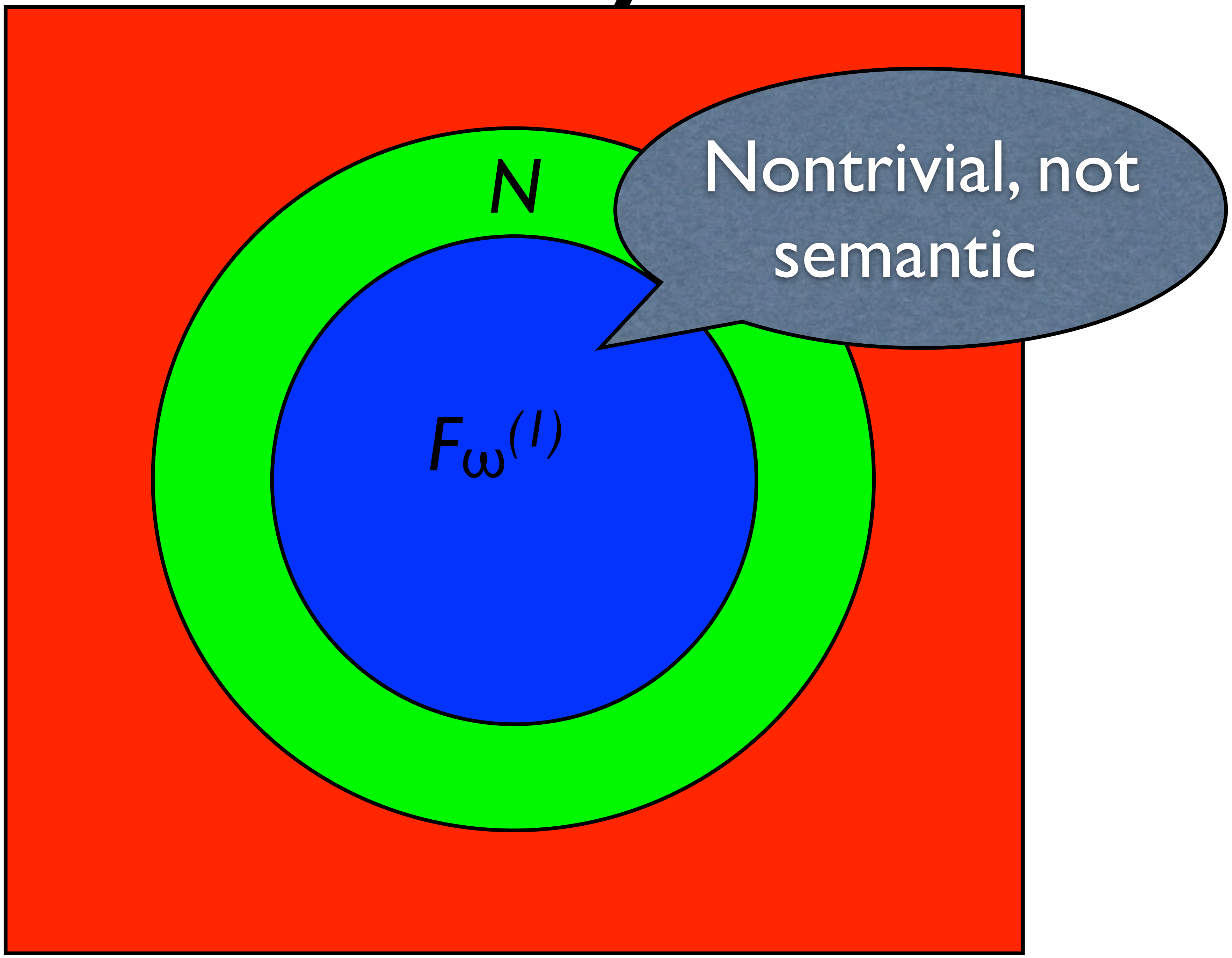
Is F recursively inseparable from N?

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)

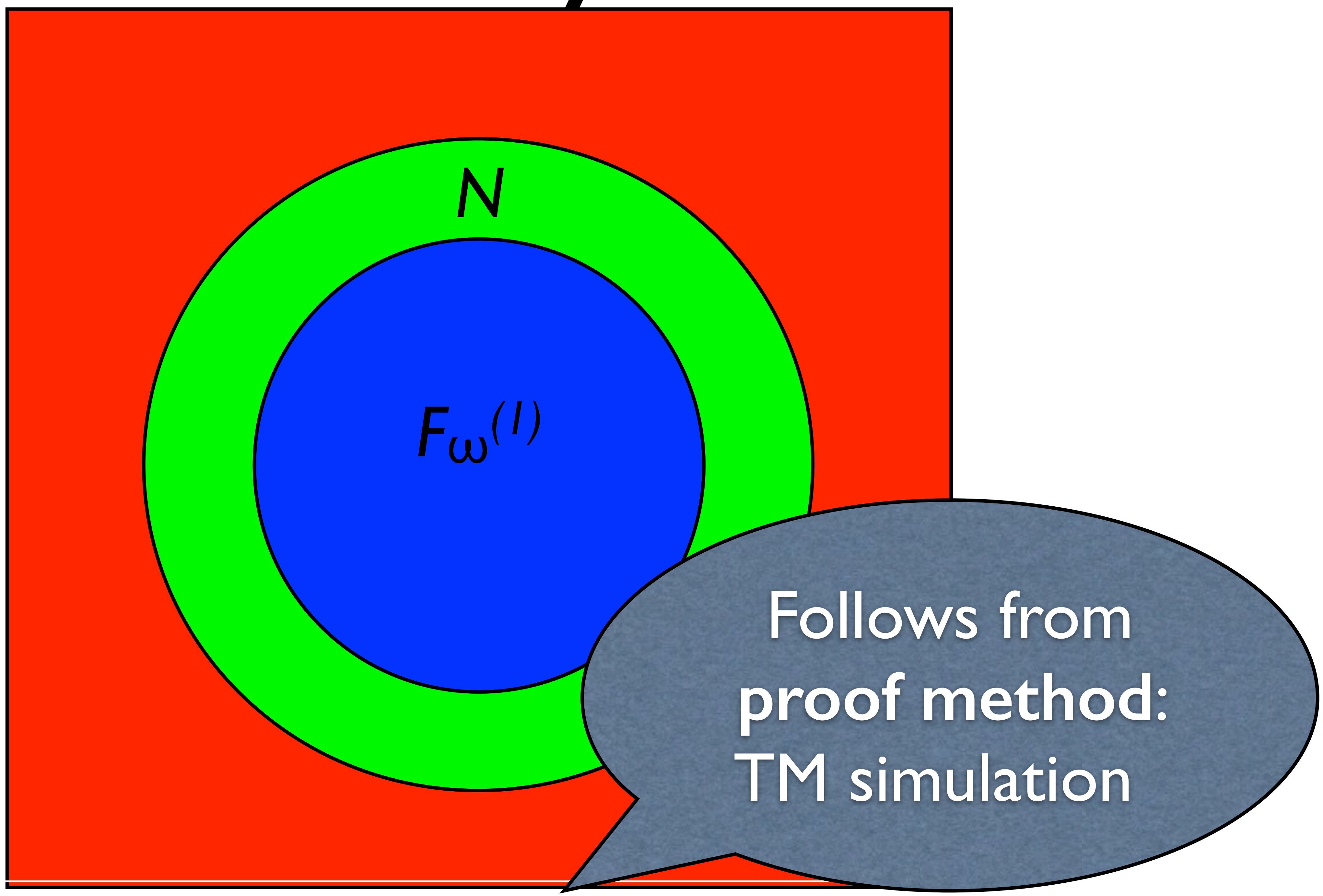
Another analysis for N



Theorem: $F_\omega^{(1)}$ is undecidable

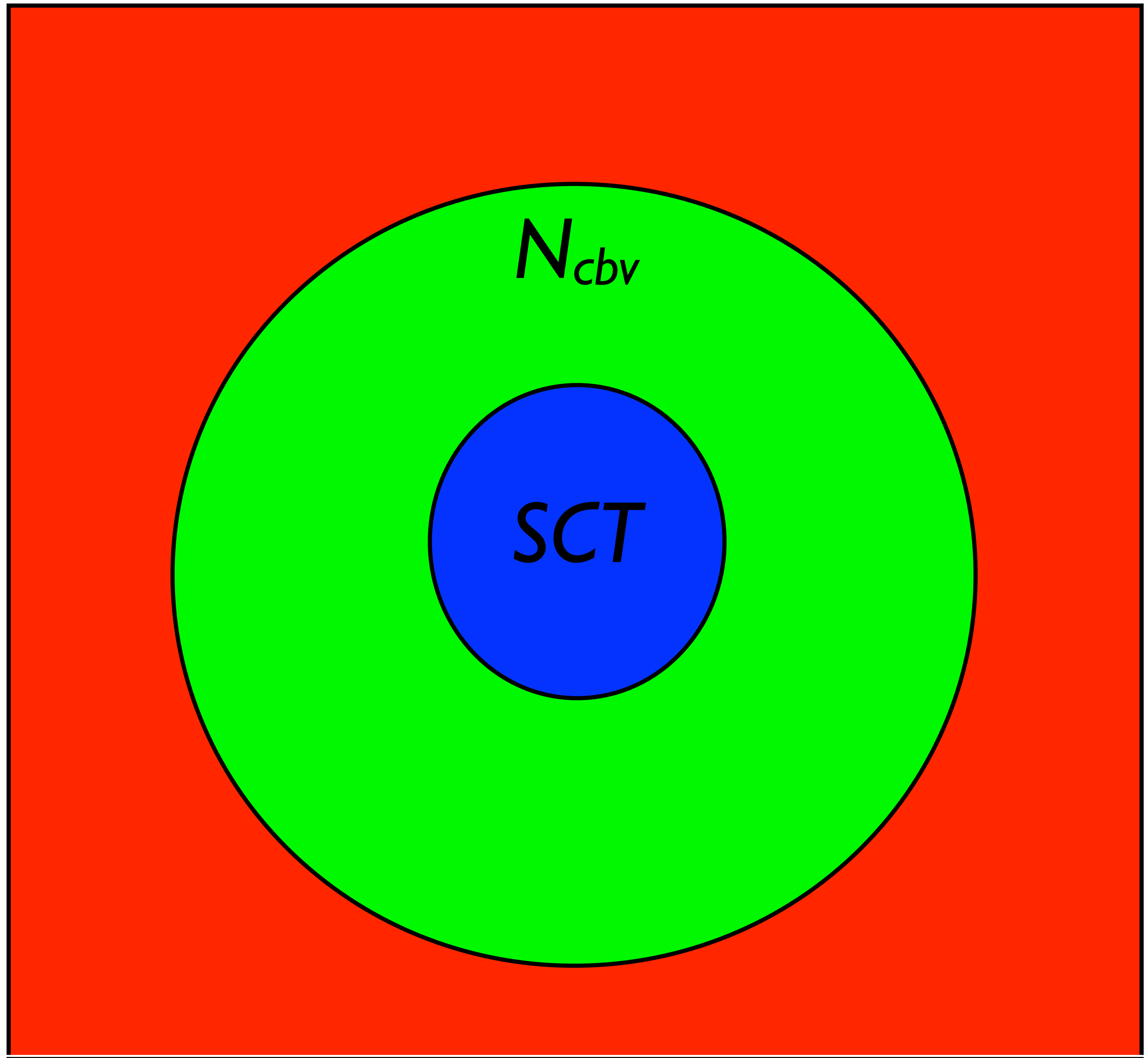
Urzyczyn, Type reconstruction in Fω, MSCS 1997

Another analysis for N



Theorem: $F_\omega^{(I)}$ is recursively inseparable from N

SCT for N

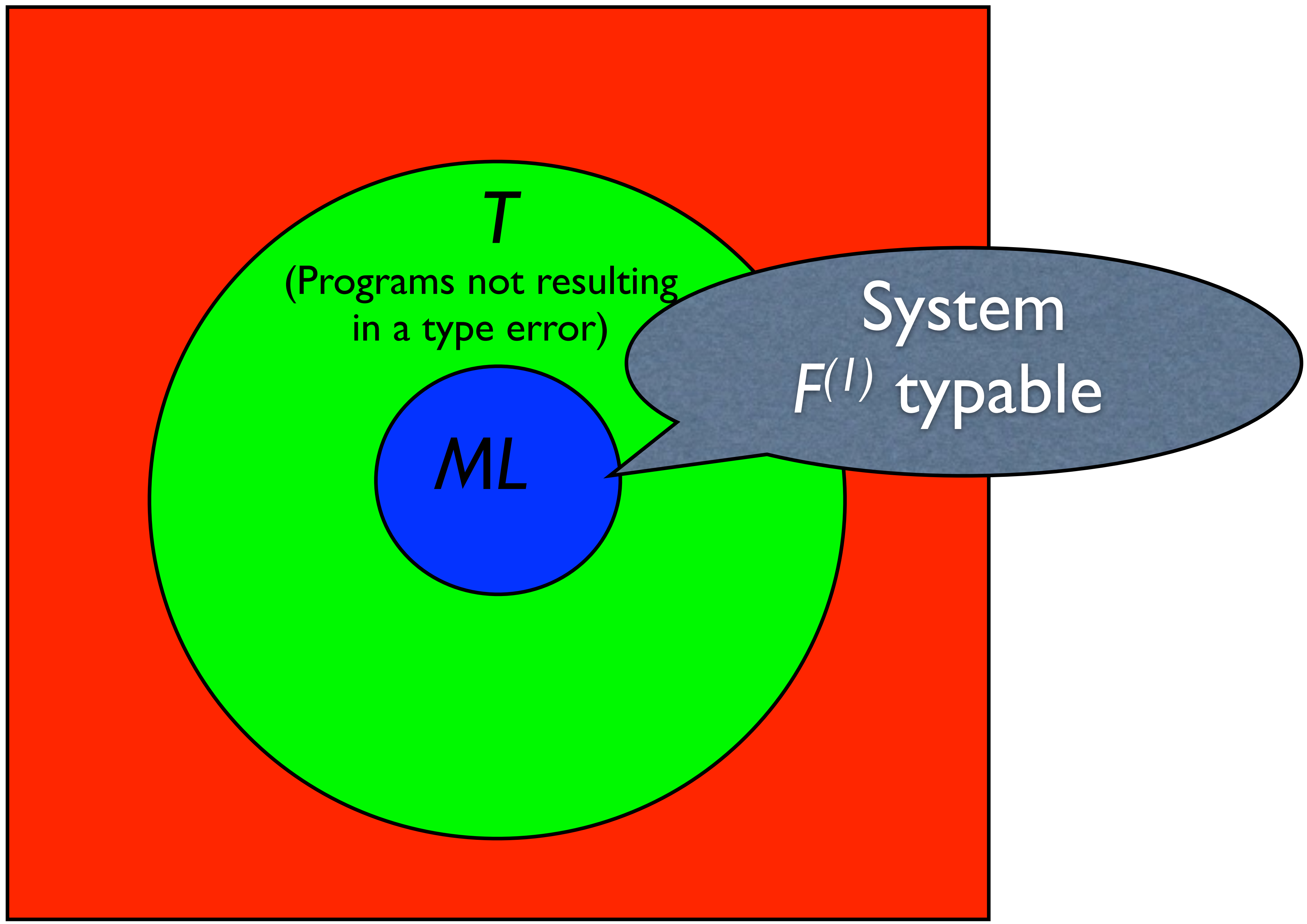


Theorem: *SCT* is decidable.

(Complexity: *PSCACE*-complete)

Bohr, Jones, Termination analysis of the untyped lambda-calculus, RTA 2004

An analysis for type error freeness



ML goodness

- **Invariant** under let-reduction:
  ML(let x = e in e') $\Leftrightarrow$ ML(e'$[e/x]$)
- **Preservation** under β-reduction:
  ML($(\lambda x.e)e'$) => ML(e$[e'/x]$)

  if x occurs in e
- **Preservation** under eta-reduction:
  ML($\lambda x.ex$) => 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 typ freen

No, ML is not improvable for type error detection

Theorem: Let $ML \subseteq B \subseteq T$.
Then B is *DEXPTIME*-hard.

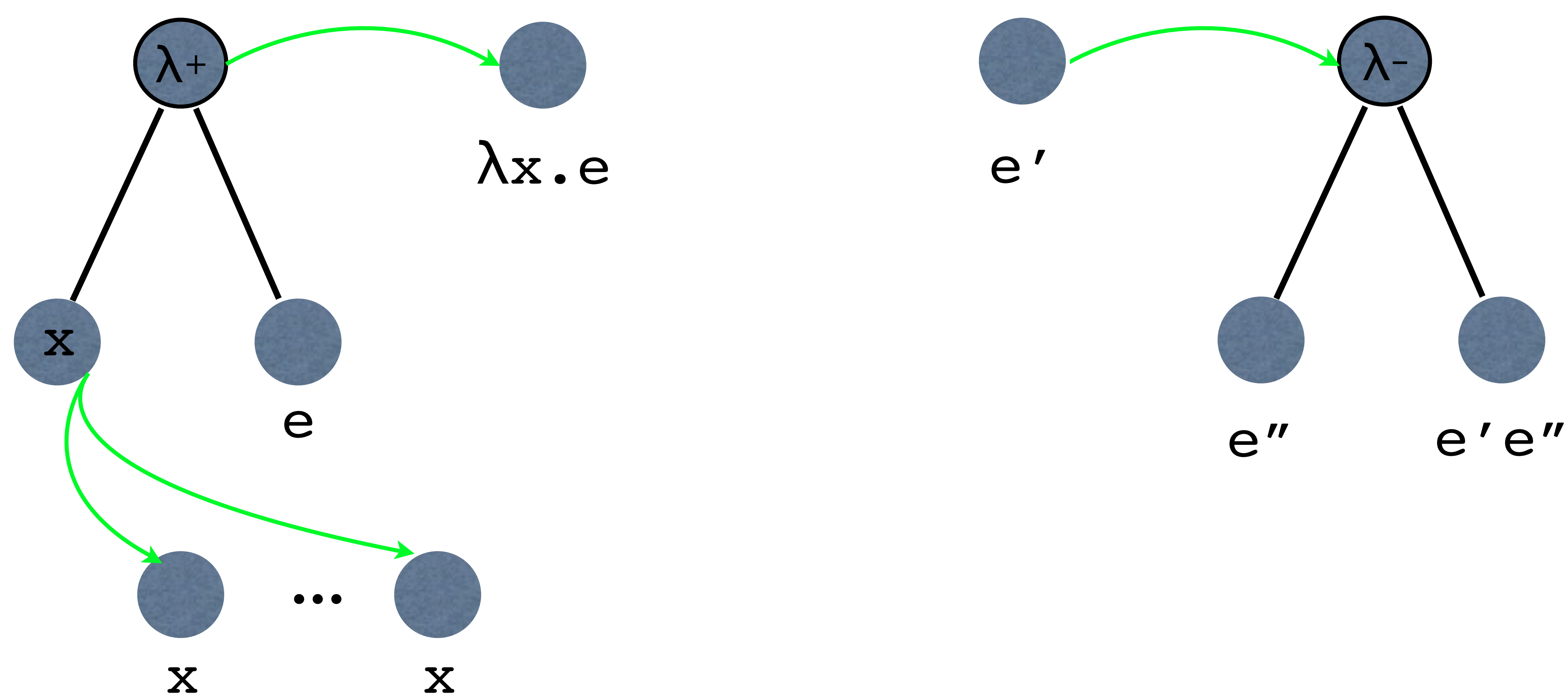
Henglein, A Lower Bound for Full Polymorphic Type Inference: Girard-Reynolds Typability is DEXPTIME-hard, Utrecht U. TR RUU-CS-90-14, 1990

mVFA

(0CFA in direct style)

Build **graph** with **flow** and **tree** edges. One node per subexpression, plus some extra ones (λ-).

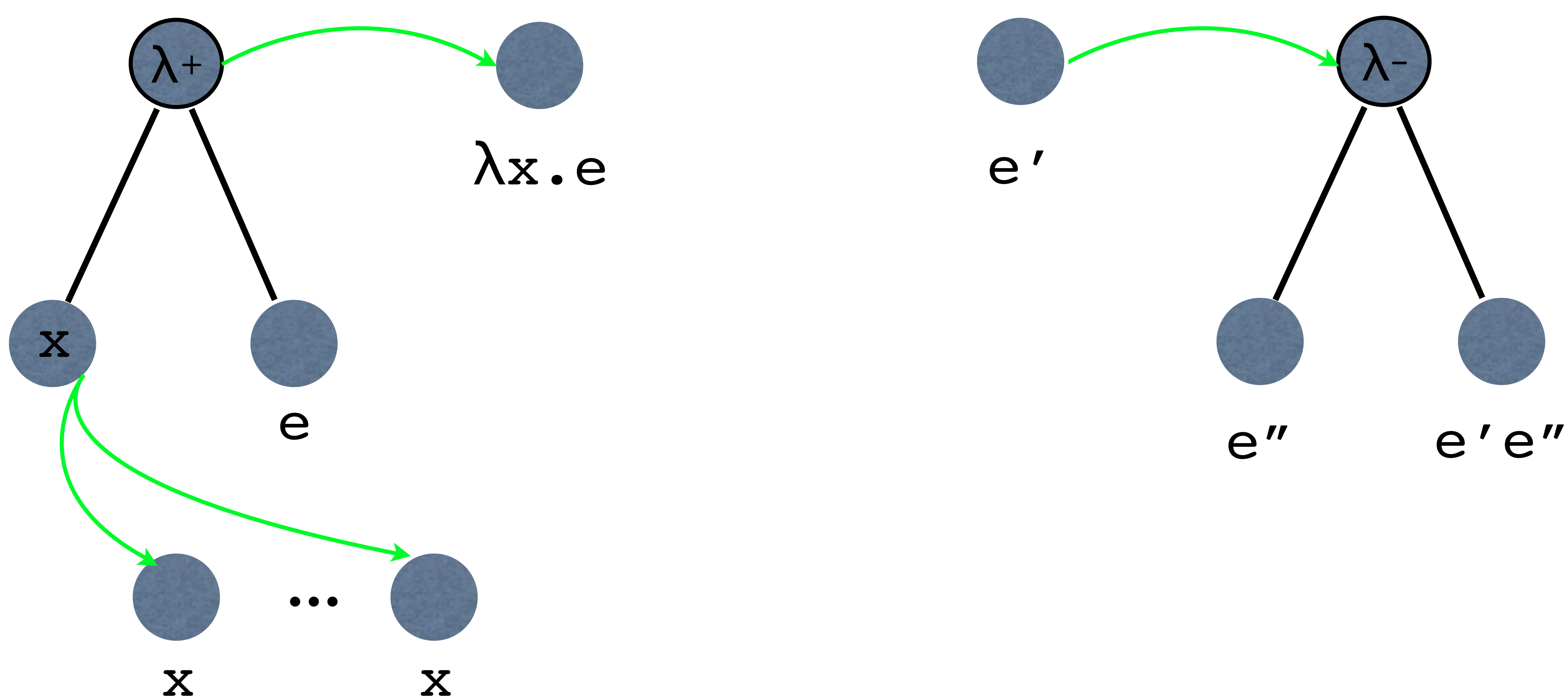
1. Base flow rules, resulting in graph *G*:



mVFA

0CFA in direct style

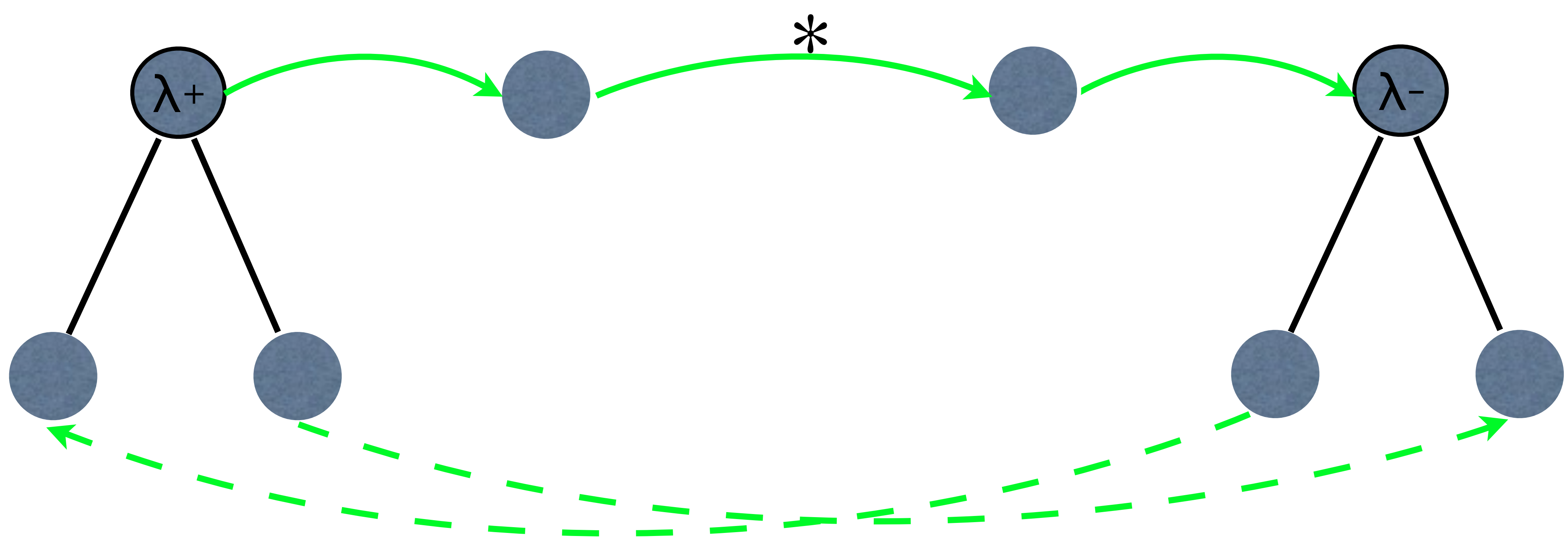
O(*n*) nodes
O(*n*) edges



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

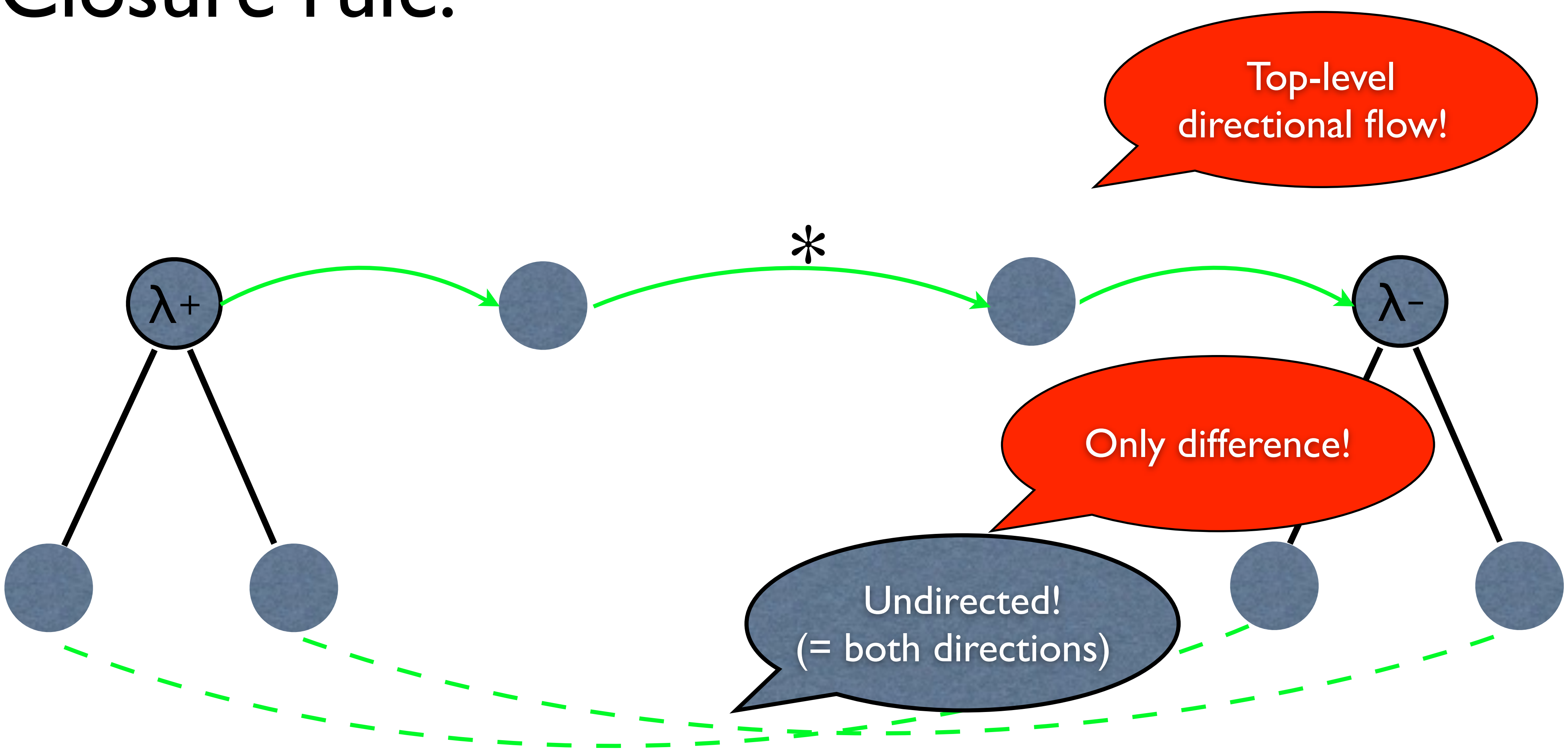
- n: number of nodes
- d: maximum outdegree of nodes in G,
- m^*: number of flow edges in G^*
  (flow-transitive closure of G),
- p: number of closure rule applications.
- q: number of reachability queries

Yellin, Speeding Up Dynamic Transitive Closure for Bounded Degree Graphs,
Acta Informatica 30, 369-384, 1993

sVFA

Simple closure analysis

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



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

    Bondorf, Jørgensen, Efficient Analysis for Realistic Off-Line Partial Evaluation, JFP 1993
- No significant reduction in precision vis a vis mVFA observed
  - Flows are *not* undirectional (“equational”)

sVFA predictability

- sVFA is invariant under
  - linear beta-reduction
  - eta-reduction (for pure λ-terms)

sVFA predictability

Theorem:
sVFA-reachability is *P*-complete

Van Horn, Mairson, Flow Analysis, Linearity, and PTIME, SAS 2008

Not a corollary. Follows from <u>proof method</u> used: invariance under linear λ-term reduction

Theorem:
Let *B* be such that *sVFA* ⊆ *B* ⊆ *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. ($S_0' = S_1' = \varnothing$)

Not a proper definition formal

- 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 $k \geq 1$?
- Is SCT improvable? How predictable is it?
- ...

End of talk