Soundness is not Sufficient

Fritz Henglein
DIKU

Shonan Village, 2011-09-25

Disclaimer

Disclaimer

- This is a rambling talk.

Disclaimer

- This is a rambling talk.
- The ideas and the intentions behind them are important, I believe.

Disclaimer

- This is a rambling talk.
- The ideas and the intentions behind them are important, I believe.
  - The technical definitions may not hold water (yet).

Disclaimer

- This is a rambling talk.
- The ideas and the intentions behind them are important, I believe.
  - The technical definitions may not hold water (yet).
- I have likely overlooked some results of yours -- tell me.

Disclaimer

- This is a rambling talk.
- The ideas and the intentions behind them are important, I believe.
  - The technical definitions may not hold water (yet).
- I have likely overlooked some results of yours -- tell me.
- Ask, comment, interrupt any time.

Goals

Goals

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

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

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

Program property

- A **program property** is a predicate on programs.

Program property

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

$$p \cong q => (P(p) \Leftrightarrow P(q))$$

Program property

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

$$\mathbf{p \cong q \Rightarrow (P(p) \Leftrightarrow P(q))}$$

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

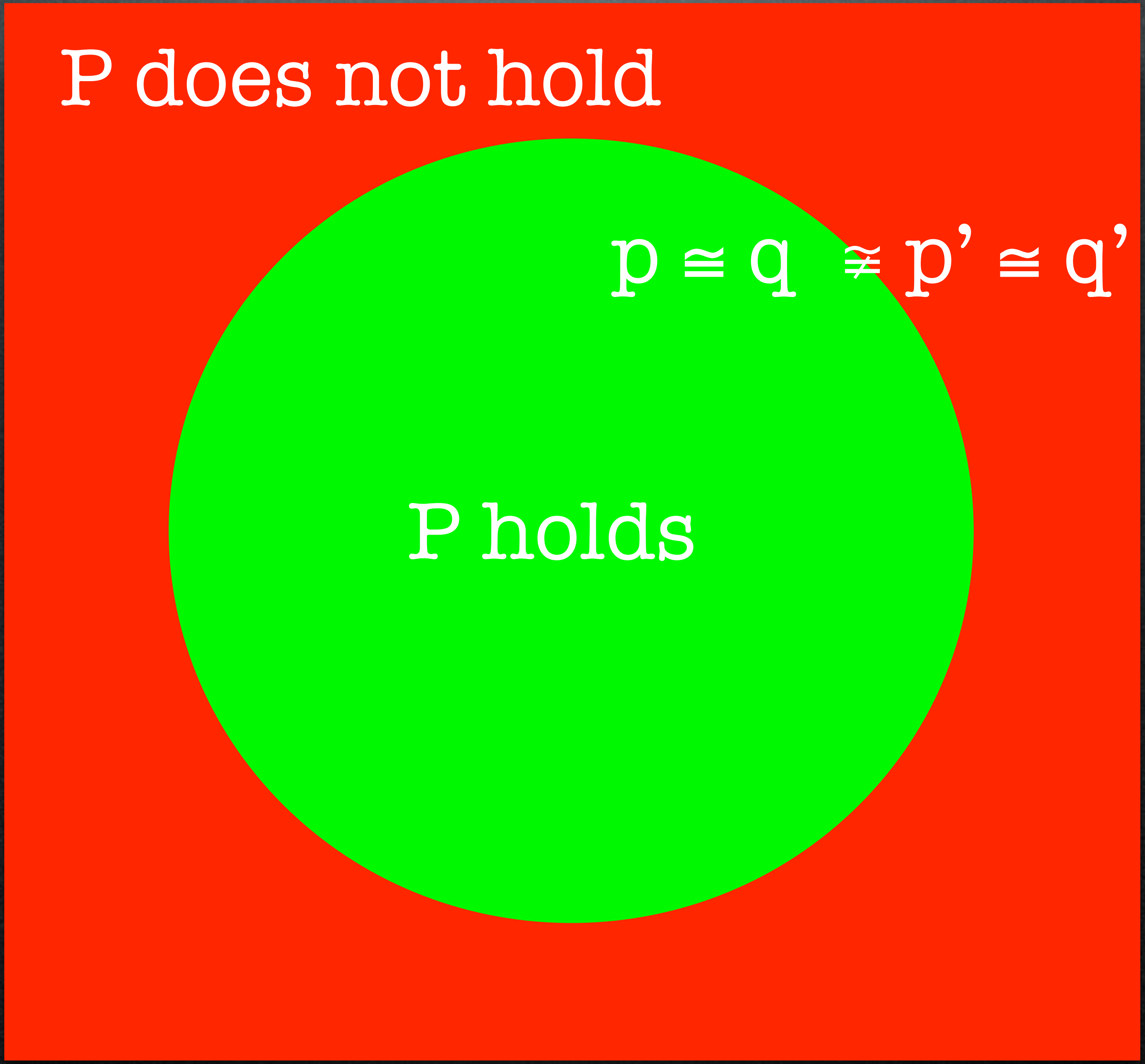
  P(p) for all p, or ¬P(p) for all p.

Rice’s Curse

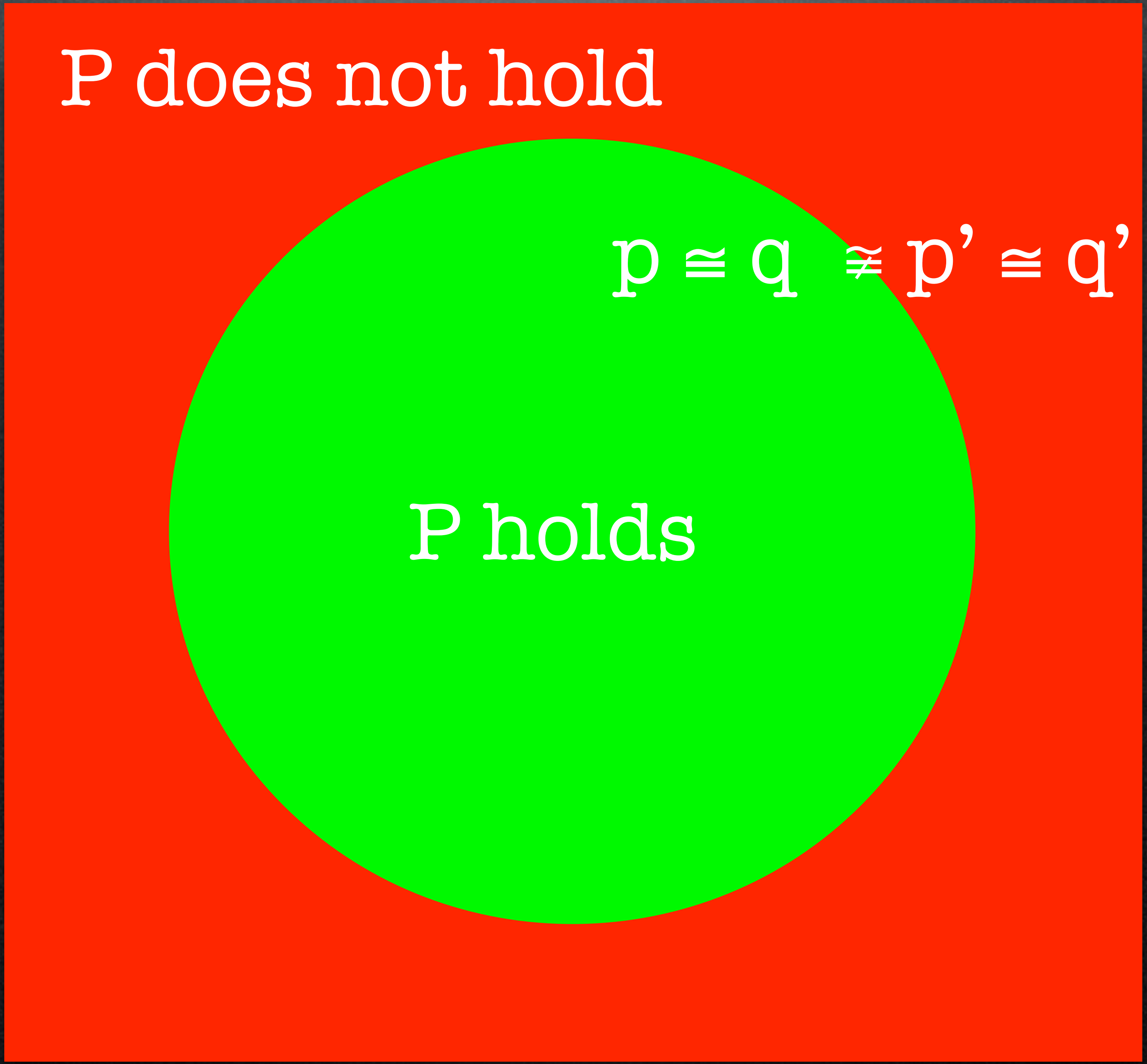
Theorem:
Let L be a Turing-complete programming language, P a nontrivial semantic program property.
Then P is undecidable.

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

Rice’s Curse, pictorially



Rice's Curse, pictorially



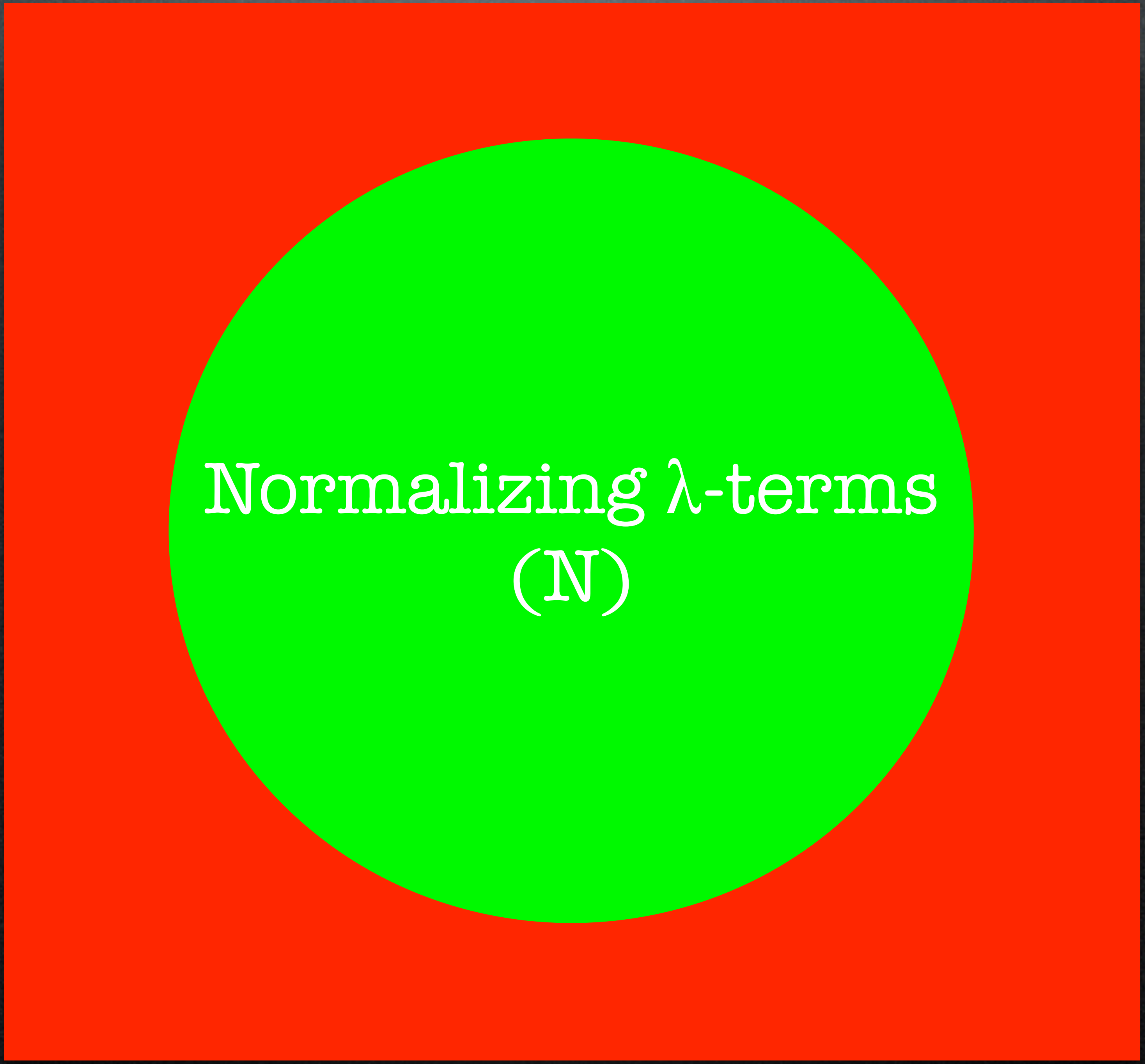
P does not hold

$p \cong q \ncong p' \cong q'$

P holds

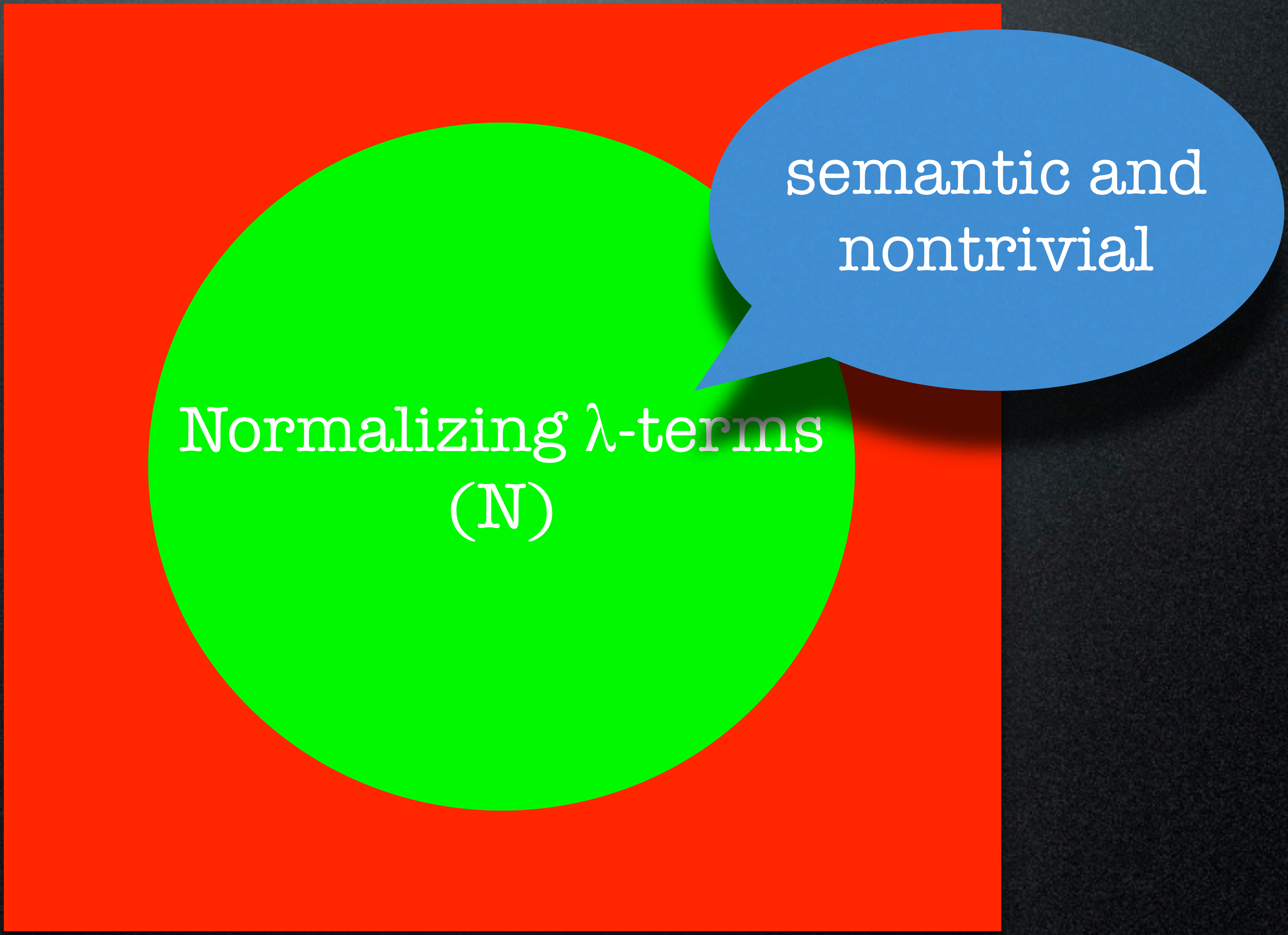
P is not decidable!

Rice’s Curse: Example



Normalizing λ-terms
(N)

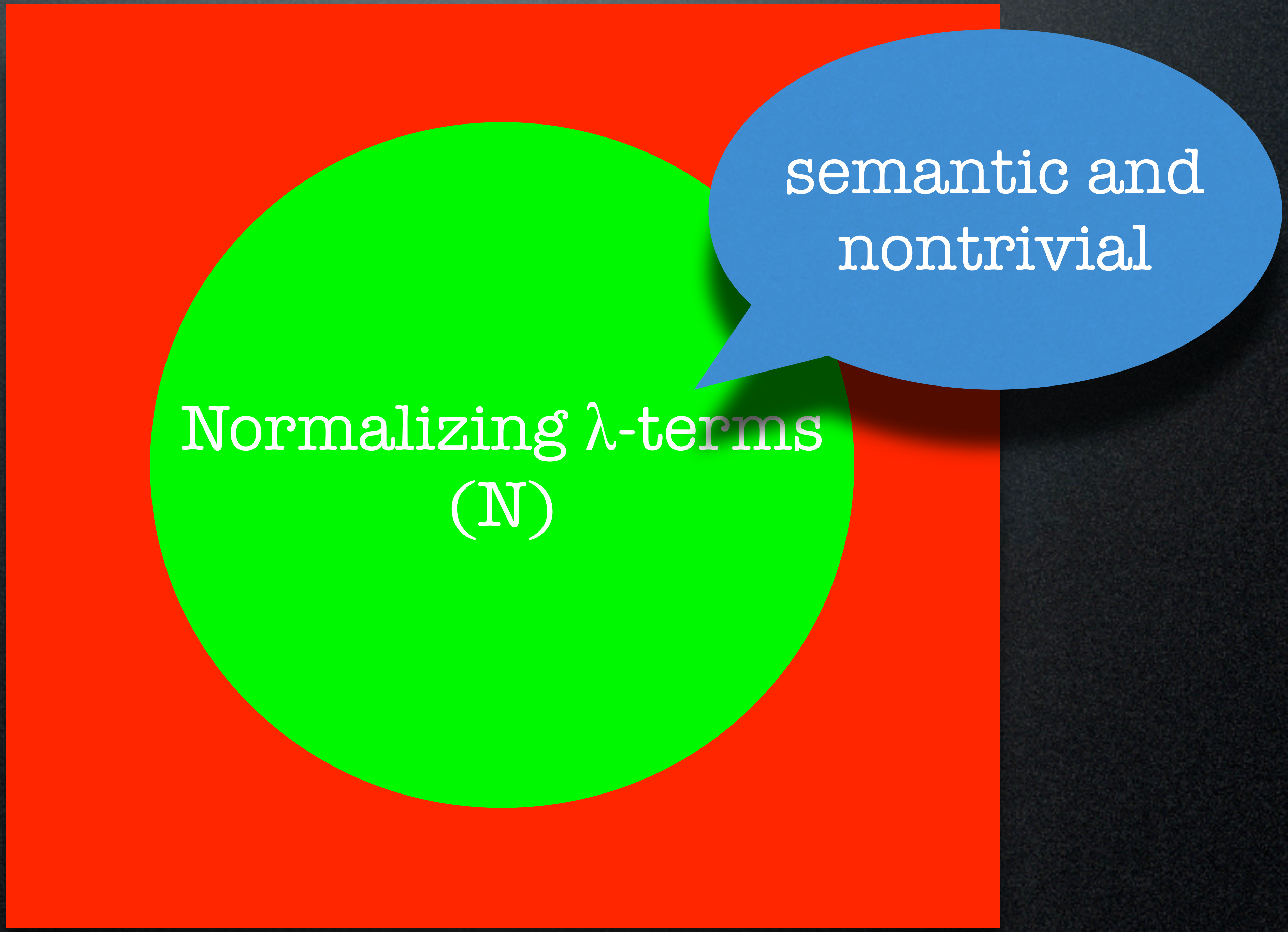
Rice’s Curse: Example



semantic and nontrivial

Normalizing λ-terms (N)

Rice’s Curse: Example

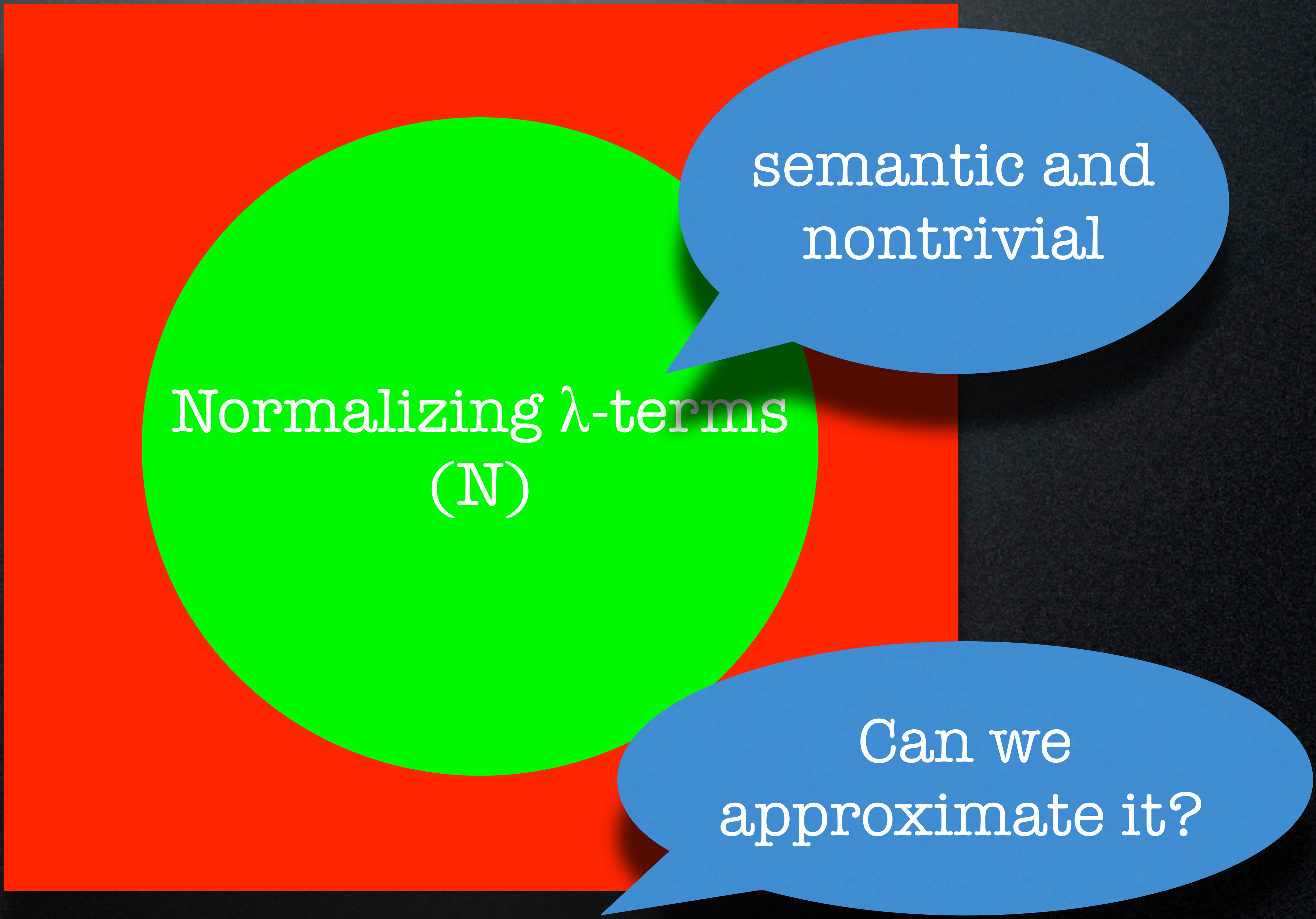


semantic and nontrivial

Normalizing λ-terms (N)

Corollary: N is not decidable!

Rice’s Curse: Example



Normalizing λ-terms (N)

semantic and nontrivial

Can we approximate it?

Corollary: N is not decidable!

Static analysis

Static analysis

- Given:

Static analysis

- Given:
    - P: Extensional program property

Static analysis

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

Static analysis

- Given:
    - P: Extensional program property
    - (S, S’): Static analysis for P
- We want of (S, S’):

Static analysis

- Given:
    - P: Extensional program property
    - (S, S’): Static analysis for P
- We want of (S, S’):
    - **Soundness**: $S \subseteq P$, $S' \subseteq \neg P$

Static analysis

- Given:
  - P: Extensional program property
  - (S, S’): Static analysis for P
- We want of (S, S’):
  - **Soundness**: S $\subseteq$ P, S’ $\subseteq$ ¬P

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

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**

Static analysis

- Given:
  - P: Extensional program property
  - (S, S’): Static analysis for P
- We want of (S, S’):
  - **Soundness**: ⊆ ¬P

    Is that suf e also want…
  - **Goodness**

What does “good” mean??

Goodness characteristics

Goodness characteristics

- **Usefulness**:
  - Has some effective use

Goodness characteristics

- **Usefulness**:
  - Has some effective use
- **Declarative specification**:
  - Separation of **what** the analysis computes from **how** it computes it (the particular algorithm[s] used)

Goodness characteristics

Goodness characteristics

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

Goodness characteristics

- **Unimprovability**:
  - Can’t get **better** approximation at **lower** computational cost
- **Predictability**:
  - Predictability under program transformations

Goodness tics

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 characteristics

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.

Goodnes

Property 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.

Goodnes s

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

- **Adaptiveness:**
  - Easy instances are handled efficiently
  - Hard instan time.

(Not developed here)

- **Parameter sensitivity**
  - Scale well with parameter, which captures expectations on input distribution.

Static Analysis for N

Static Analysis for N

- Imagine we want to analyze N

Static Analysis for N

- Imagine we want to analyze N
- Is System F typability a good static analysis for N?

System F for N

System F for N

- Sound? ✔

System F for N

- Sound? ✔
- Declarative? ✔

System F for N

- Sound? ✔
- Declarative? ✔
- Compositionally certified? ✔

System F for N

- Sound? ✔
- Declarative? ✔
- Compositionally certified? ✔
- Useful? ✔

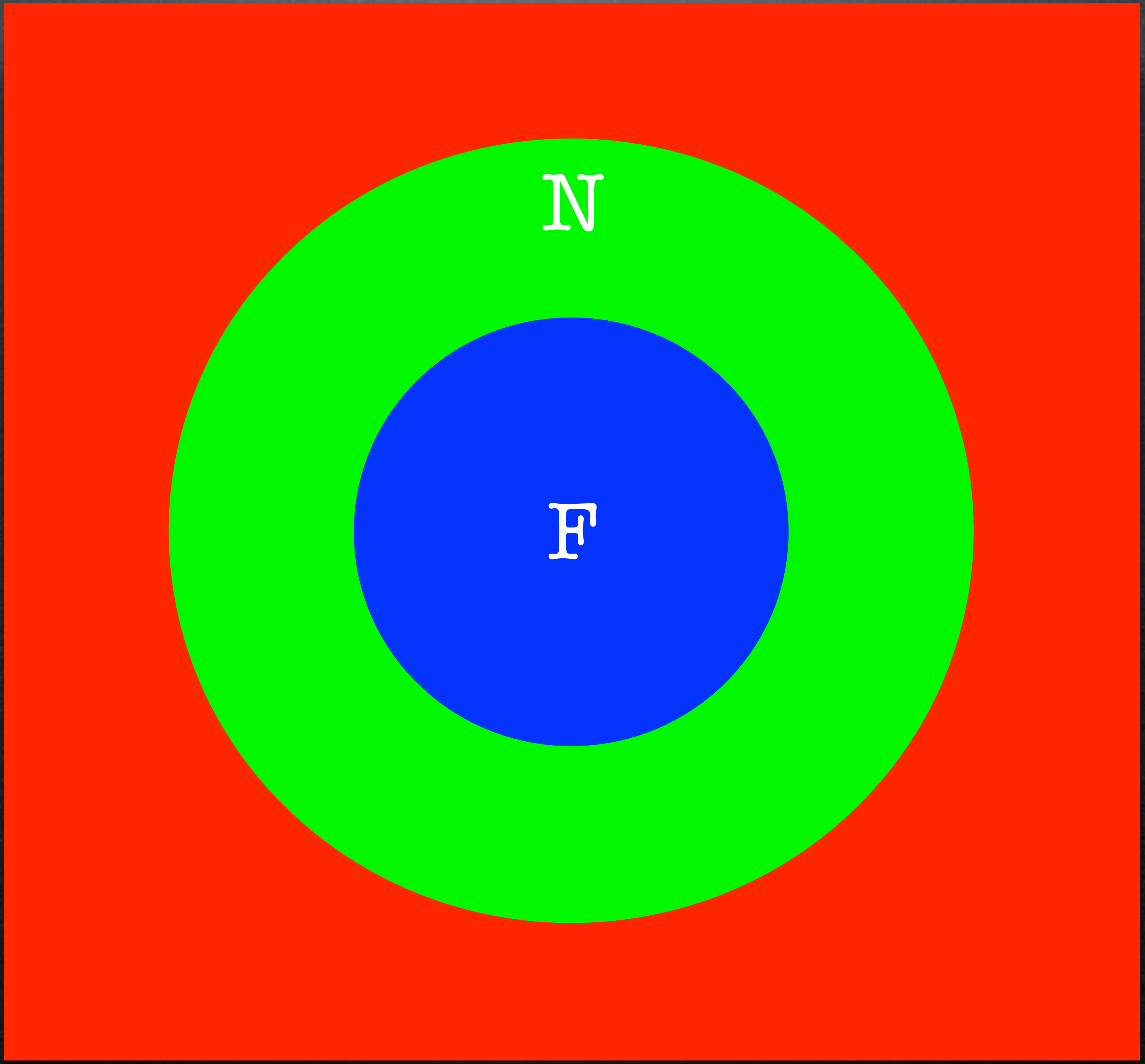
System F for N

- Sound? ✔
- Declarative? ✔
- Compositionally certified? ✔
- Useful? ✔
- Predictability properties? (✔)

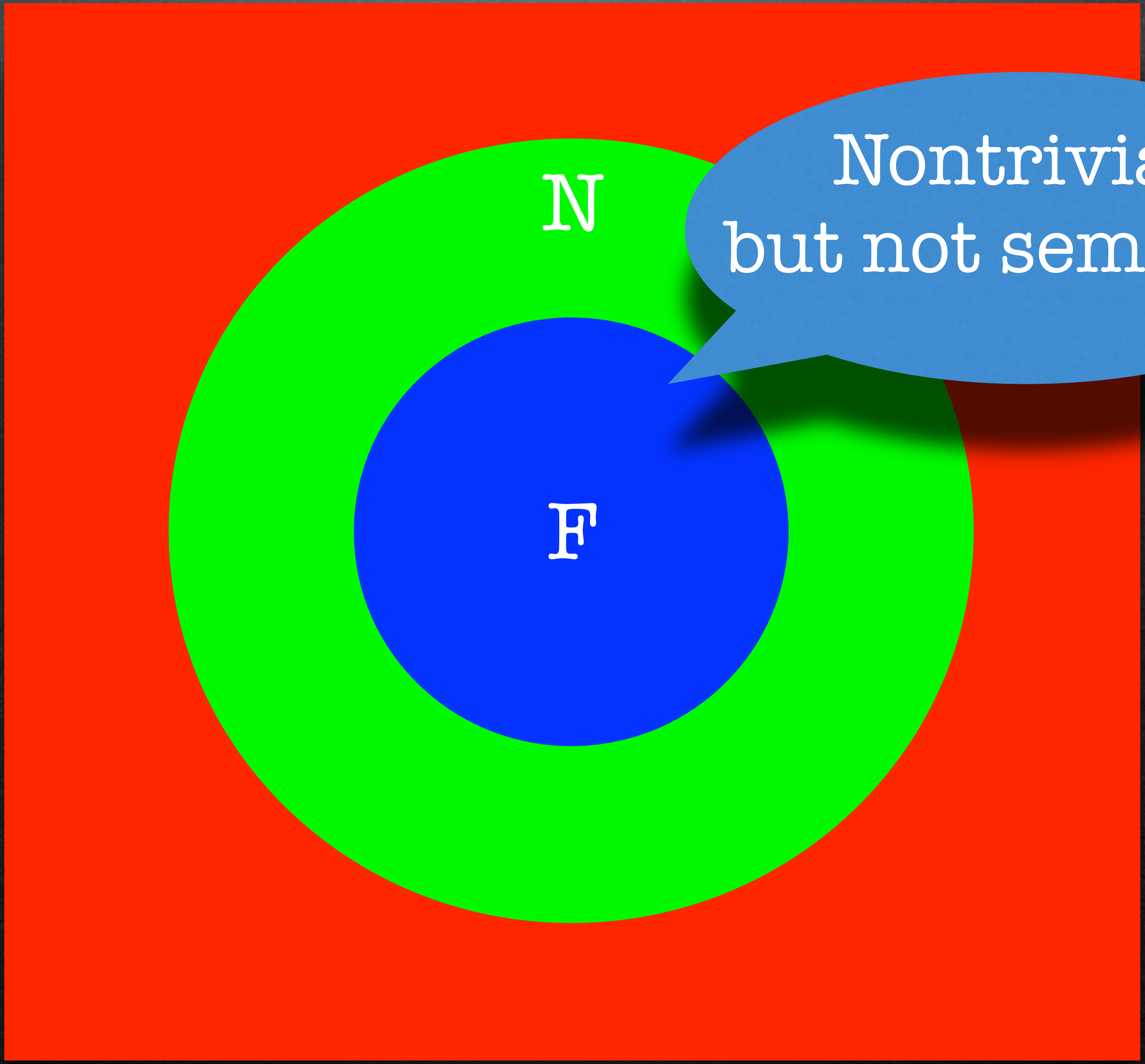
System F for N

- Sound? ✔
- Declarative? ✔
- Compositionally certified? ✔
- Useful? ✔
- Predictability properties? (✔)
- Unimprovability? Hmm...

Static Analysis for N

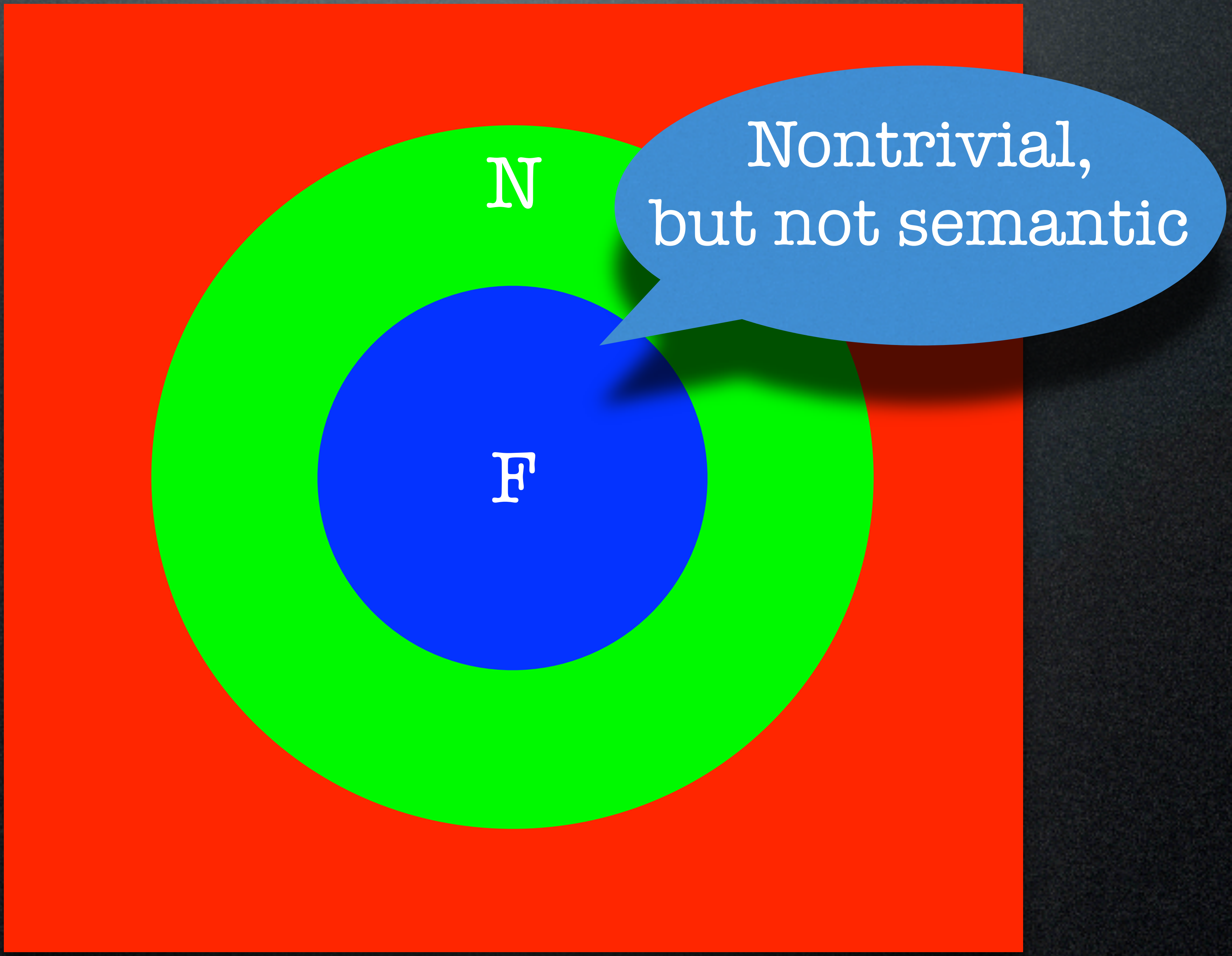


Static Analysis for N



Nontrivial, but not semantic

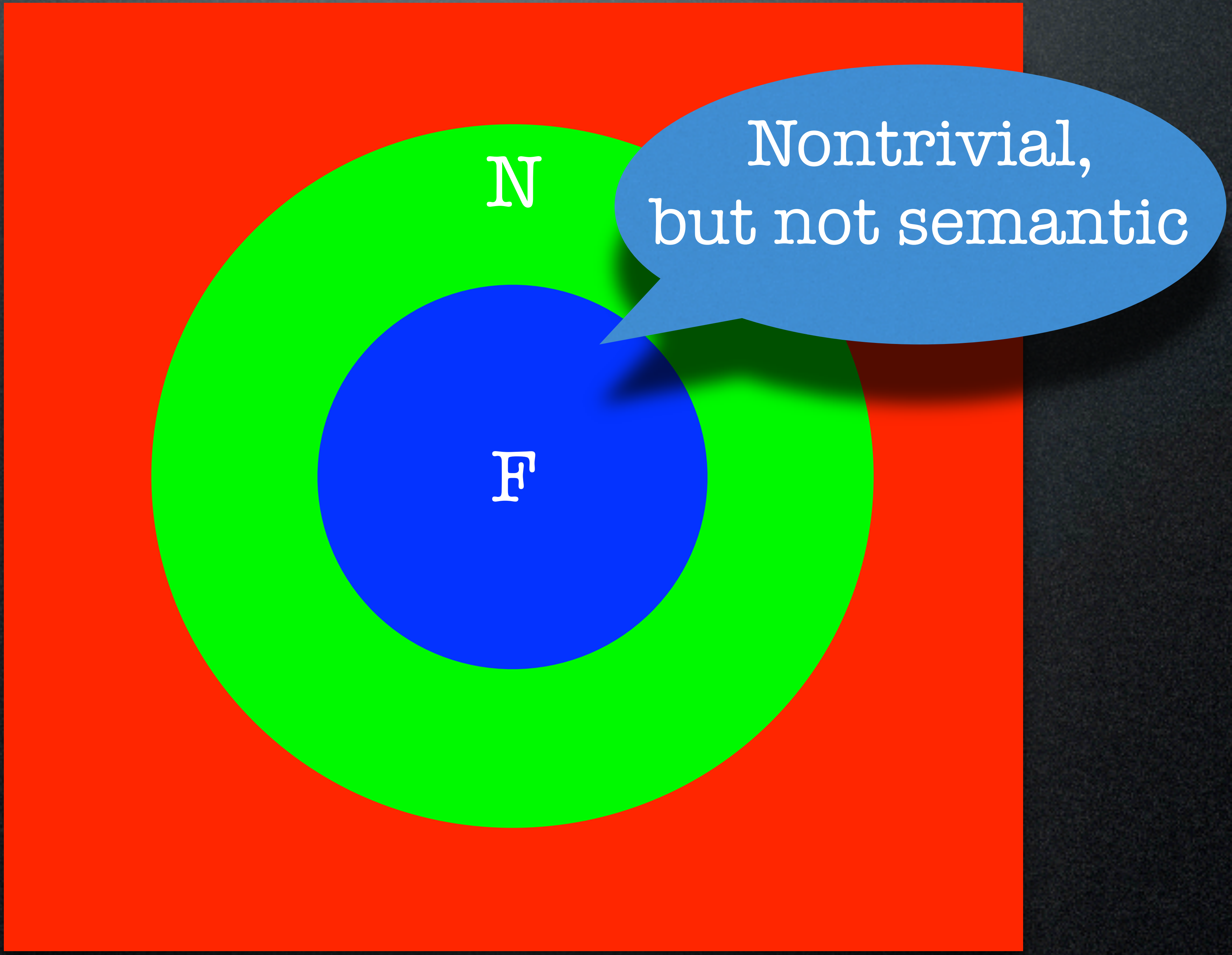
Static Analysis for N



Nontrivial, but not semantic

Theorem: F is undecidable

Static Analysis for N



Nontrivial,
but not semantic

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

- Okay for System F to be undecidable, as long as there is no **better** approximation of N that is decidable (**more** efficient).

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”).

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?

Is F recursively inseparable from N?

- The answer is...

Is F recursively inseparable from N?

- The answer is…
- **We don’t know!**

Is F recursively inseparable from N?

- The answer is...
- **We don’t know!**
  - Does not follow from Well’s proof

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

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.

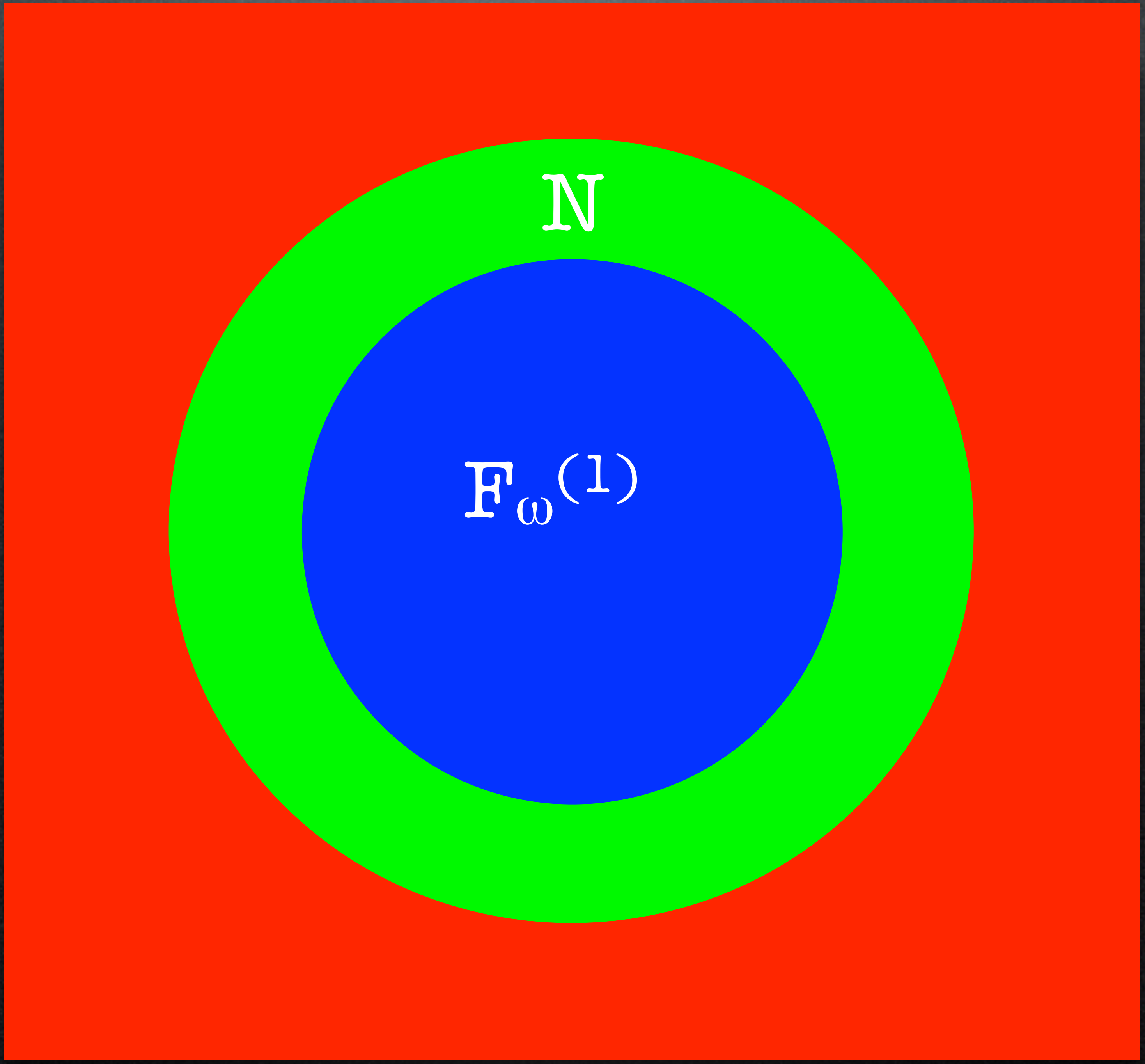
Is F recursively inseparable from N?

- The answer is...
- **We don't know!**
  - Does not fo roof

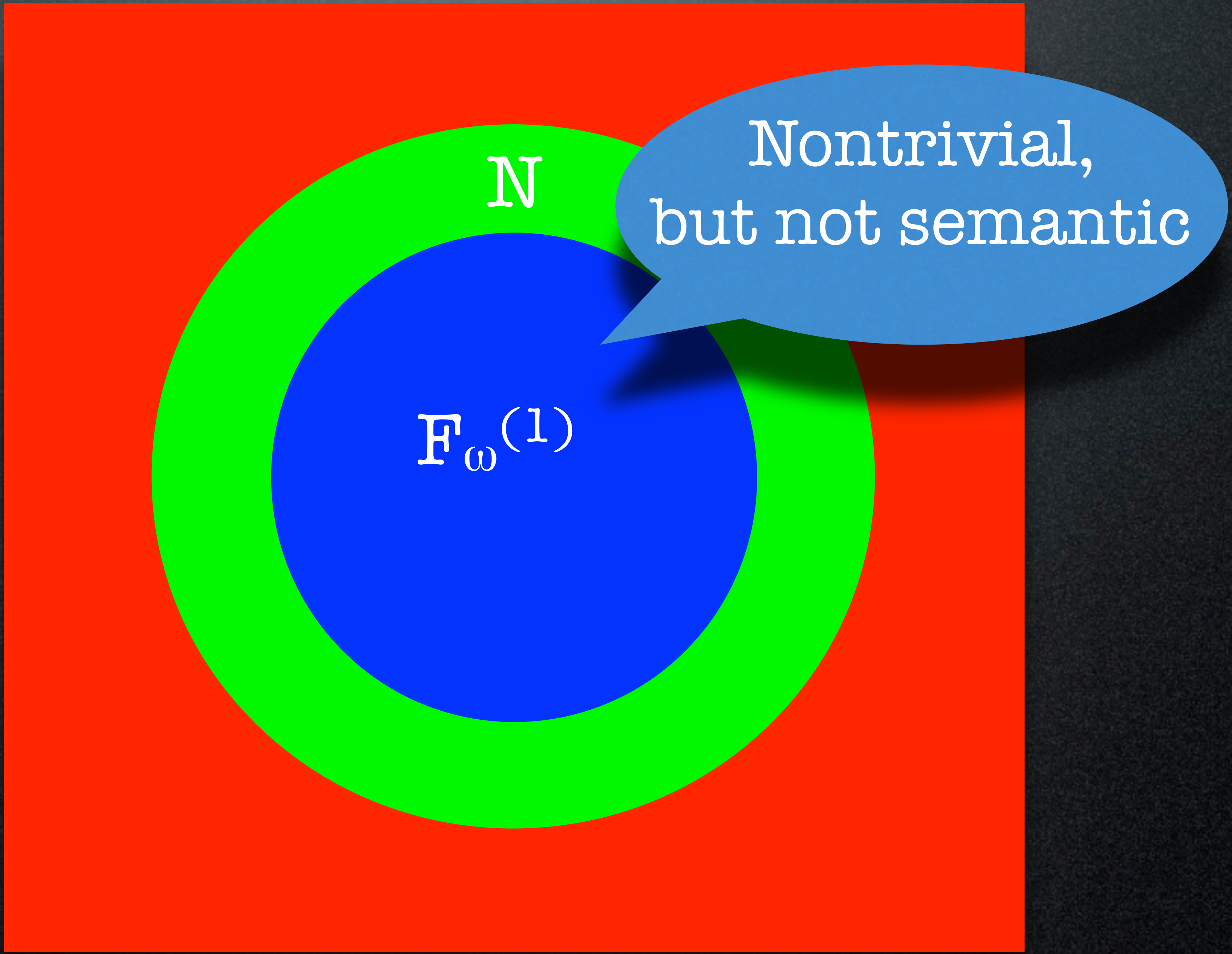
I don't believe it, though

- 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.

Another analysis for N

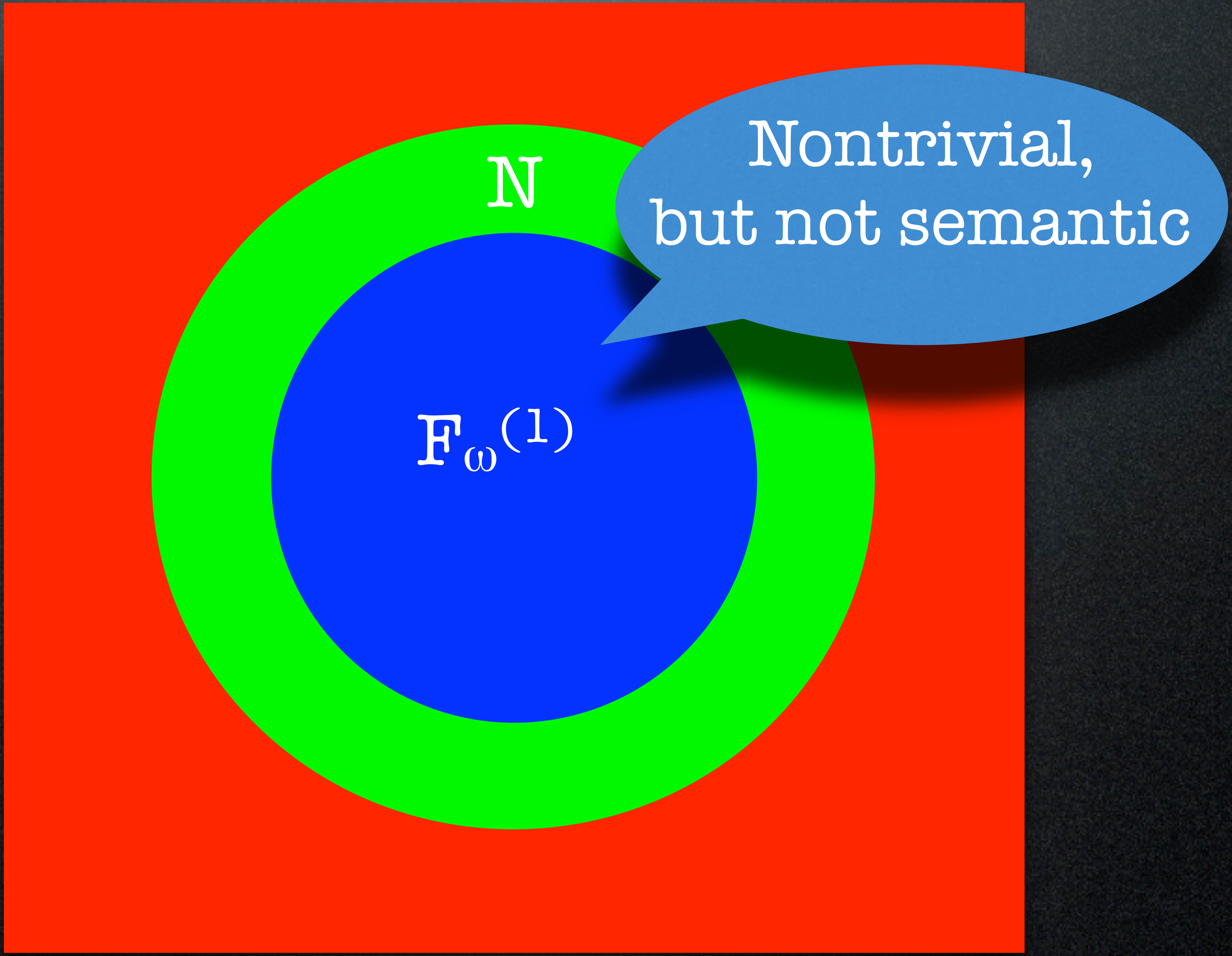


Another analysis for N



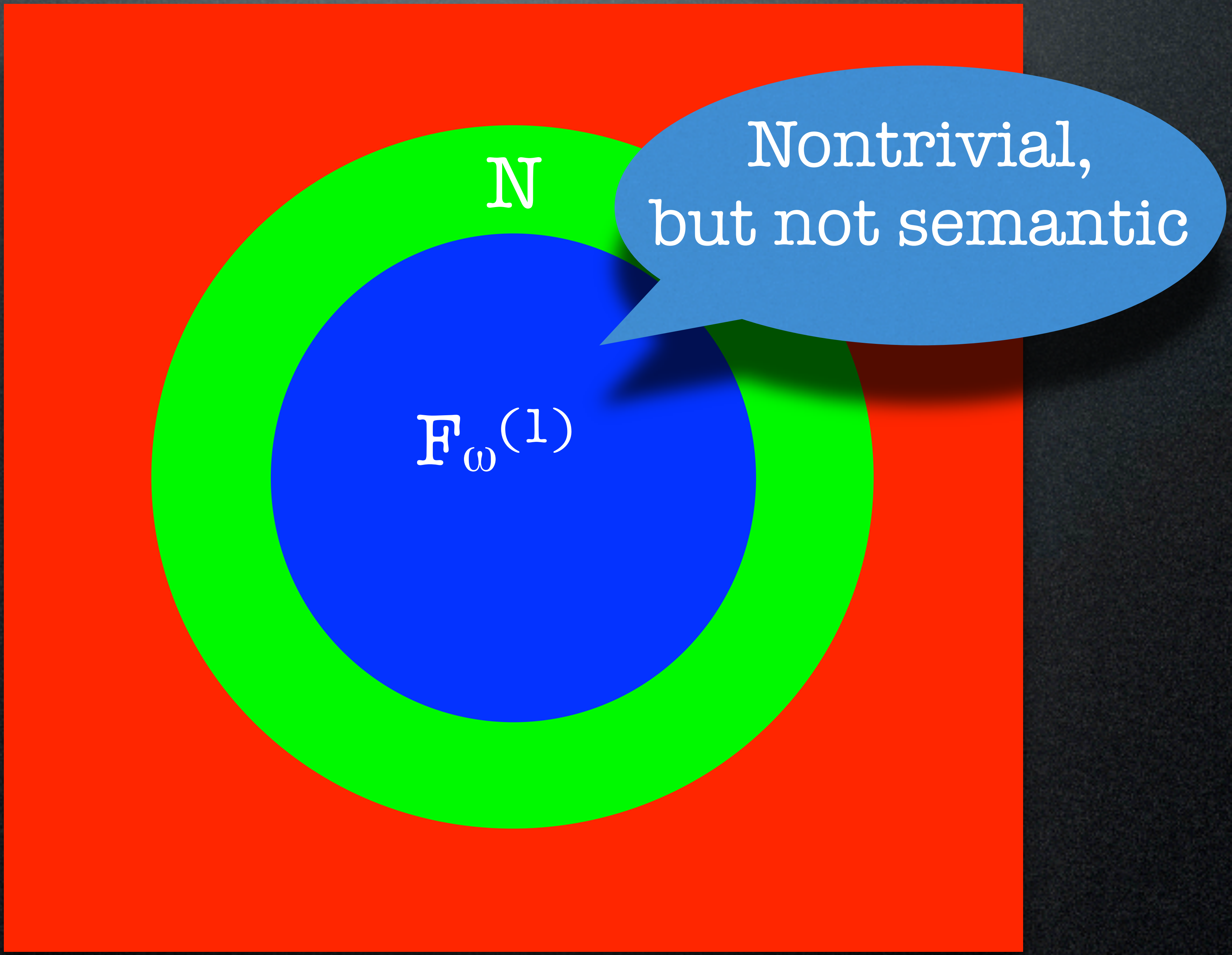
Nontrivial,
but not semantic

Another analysis for N



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

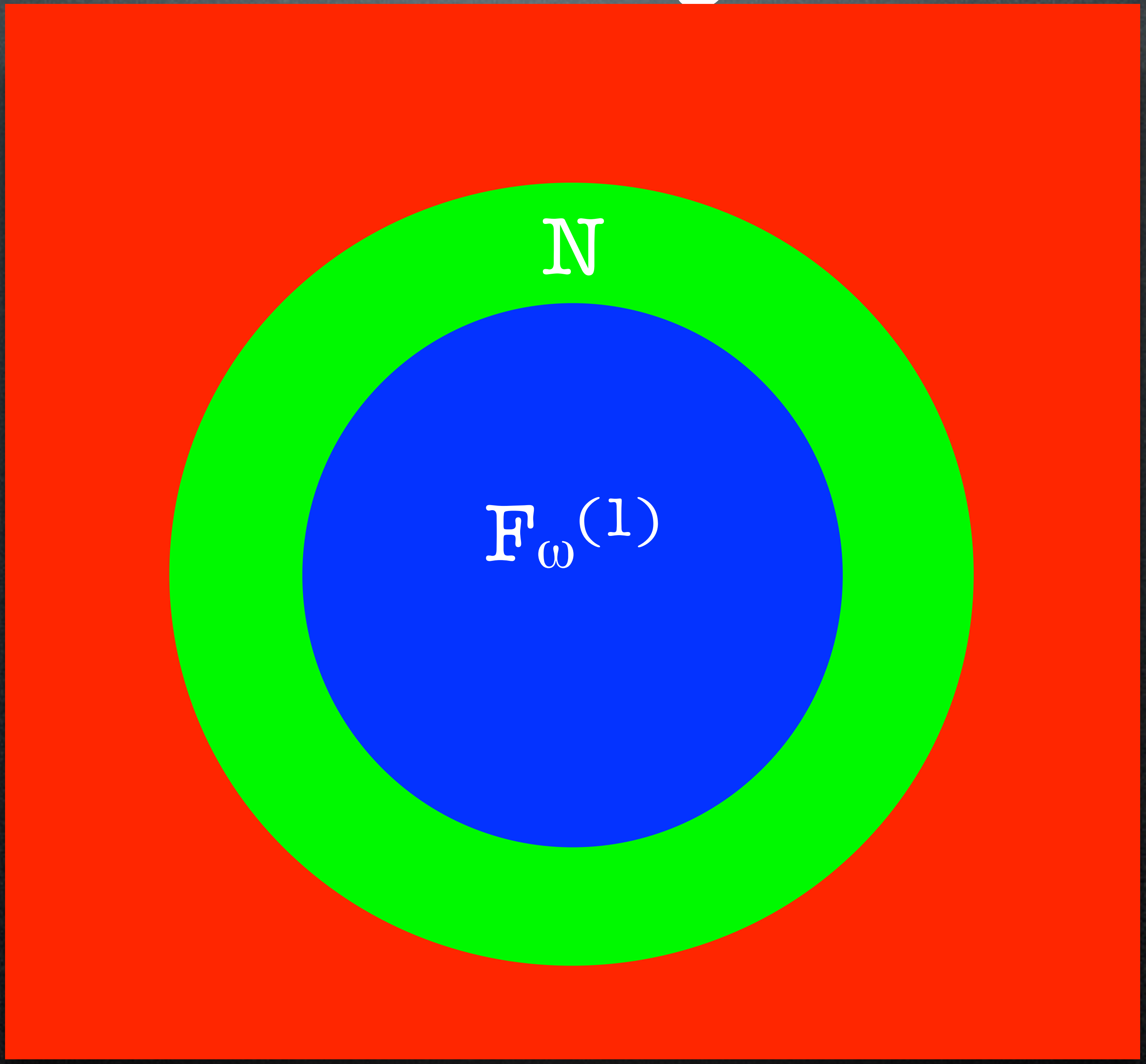
Another analysis for N



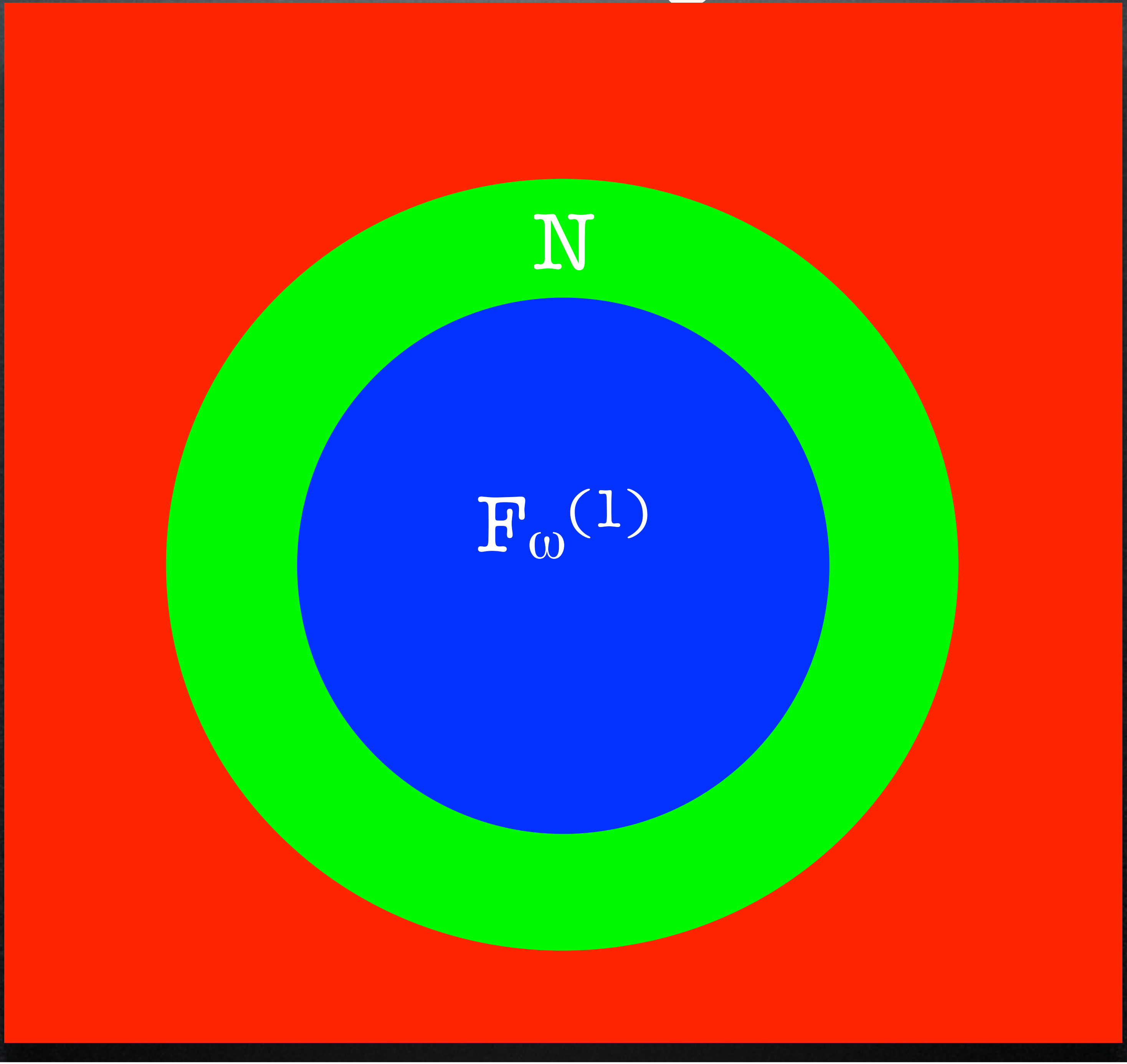
Theorem: $F_\omega^{(1)}$ is undecidable

Urzyczyn, Type reconstruction in Fω, MSCS 1997

Another analysis for N

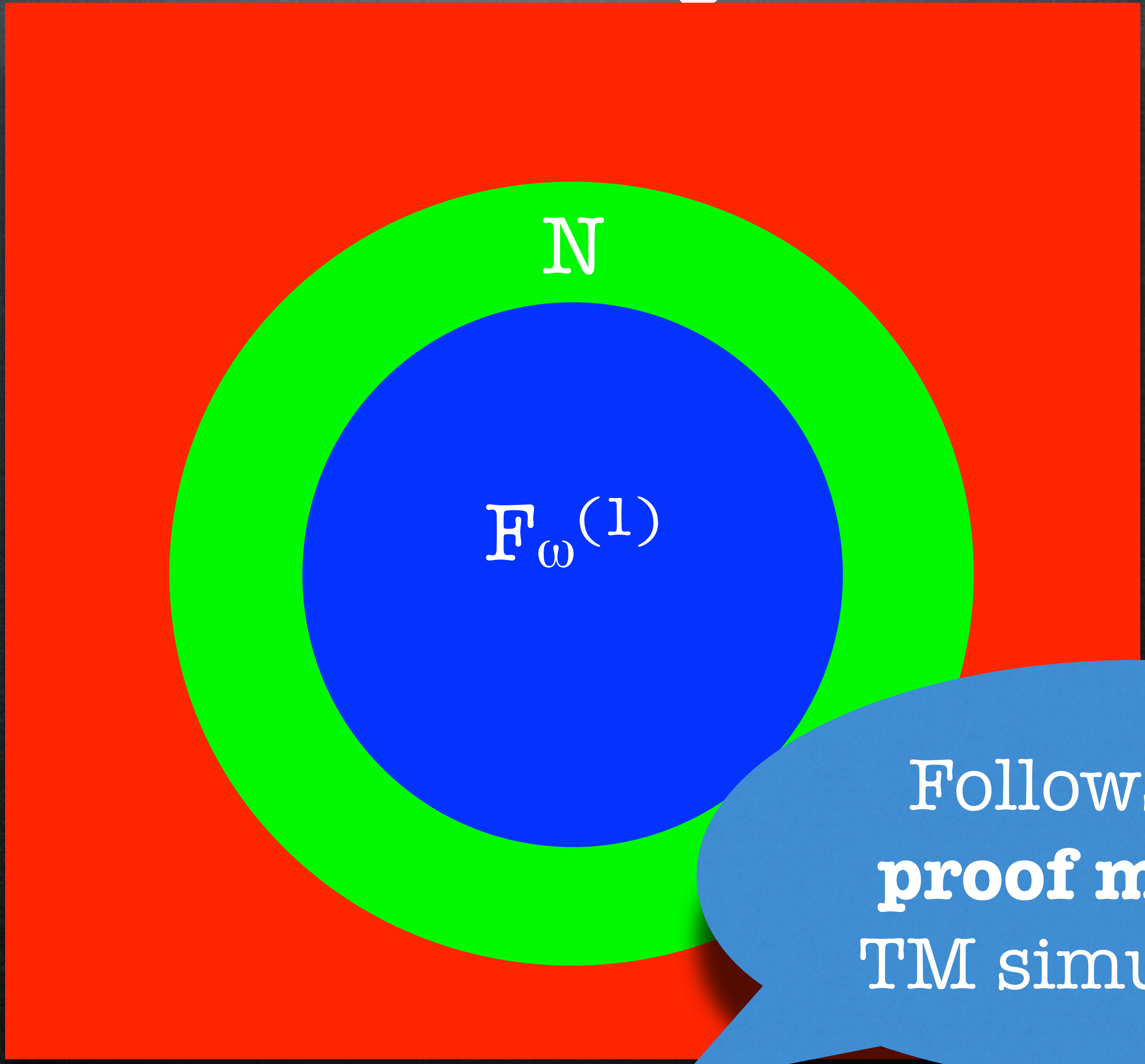


Another analysis for N



Theorem: $F_\omega^{(1)}$ is recursively inseparable from N

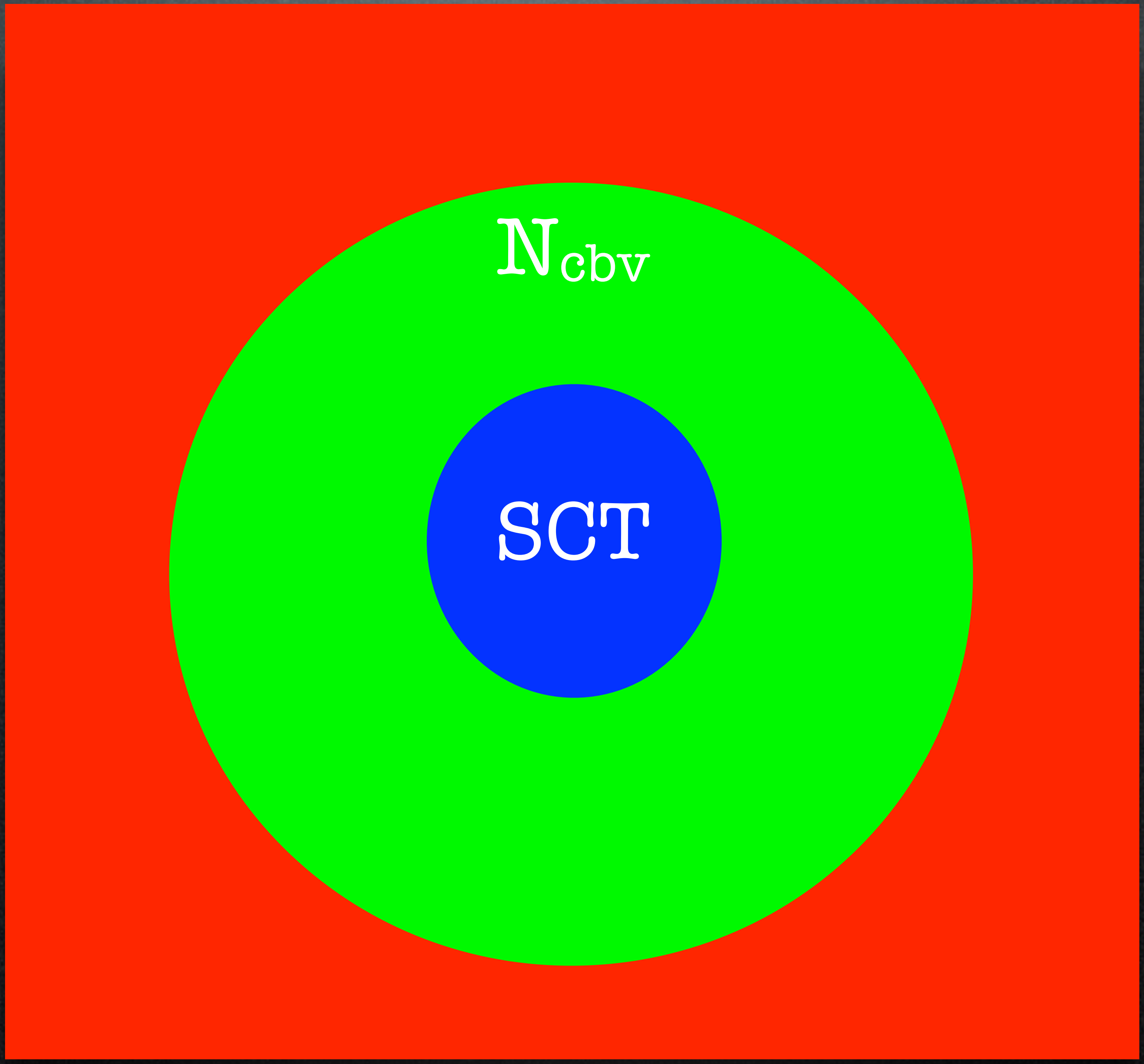
Another analysis for N



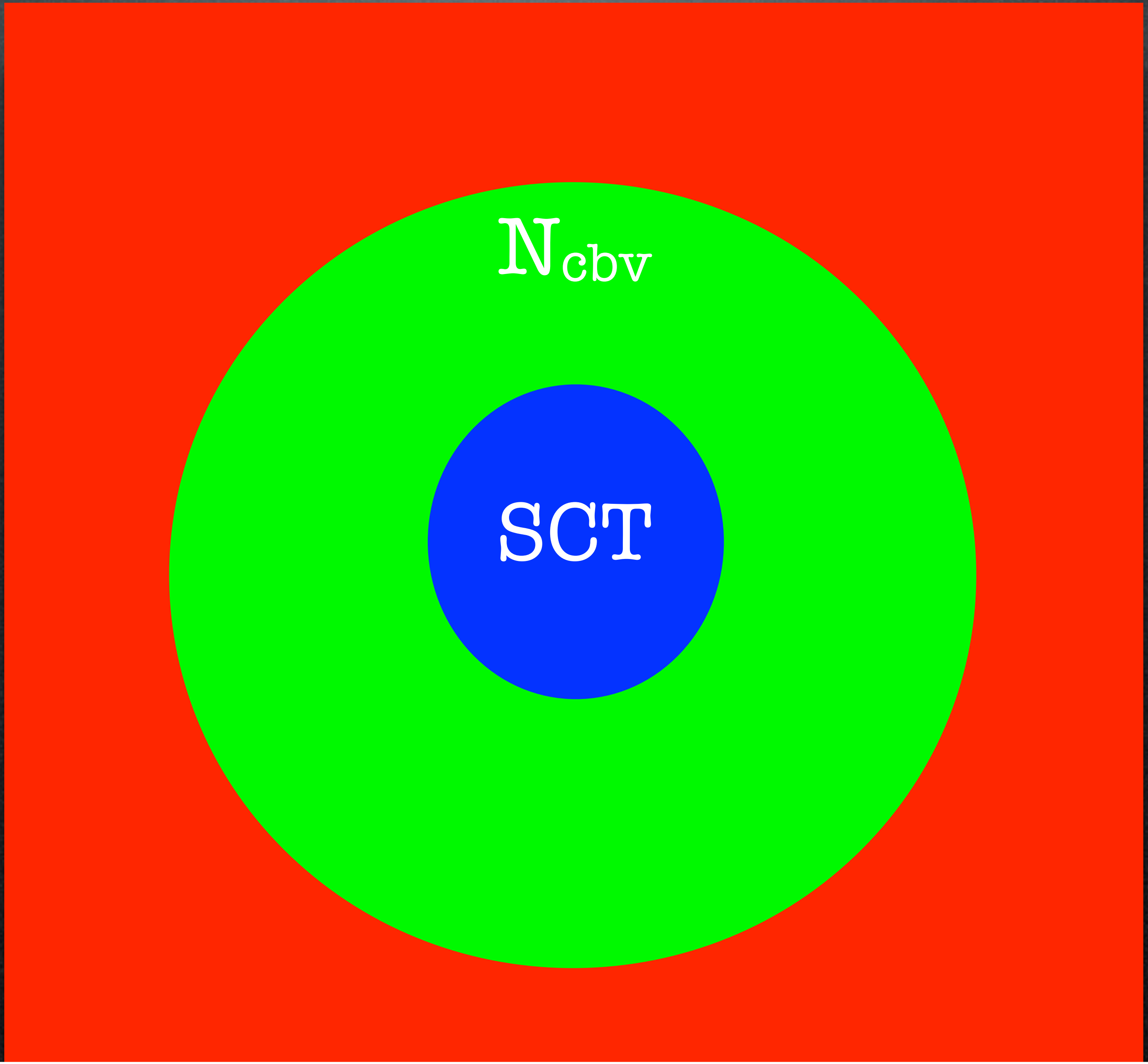
Follows from **proof method**: TM simulation

Theorem: $F_\omega^{(1)}$ is recursively inseparable from N

SCT for N

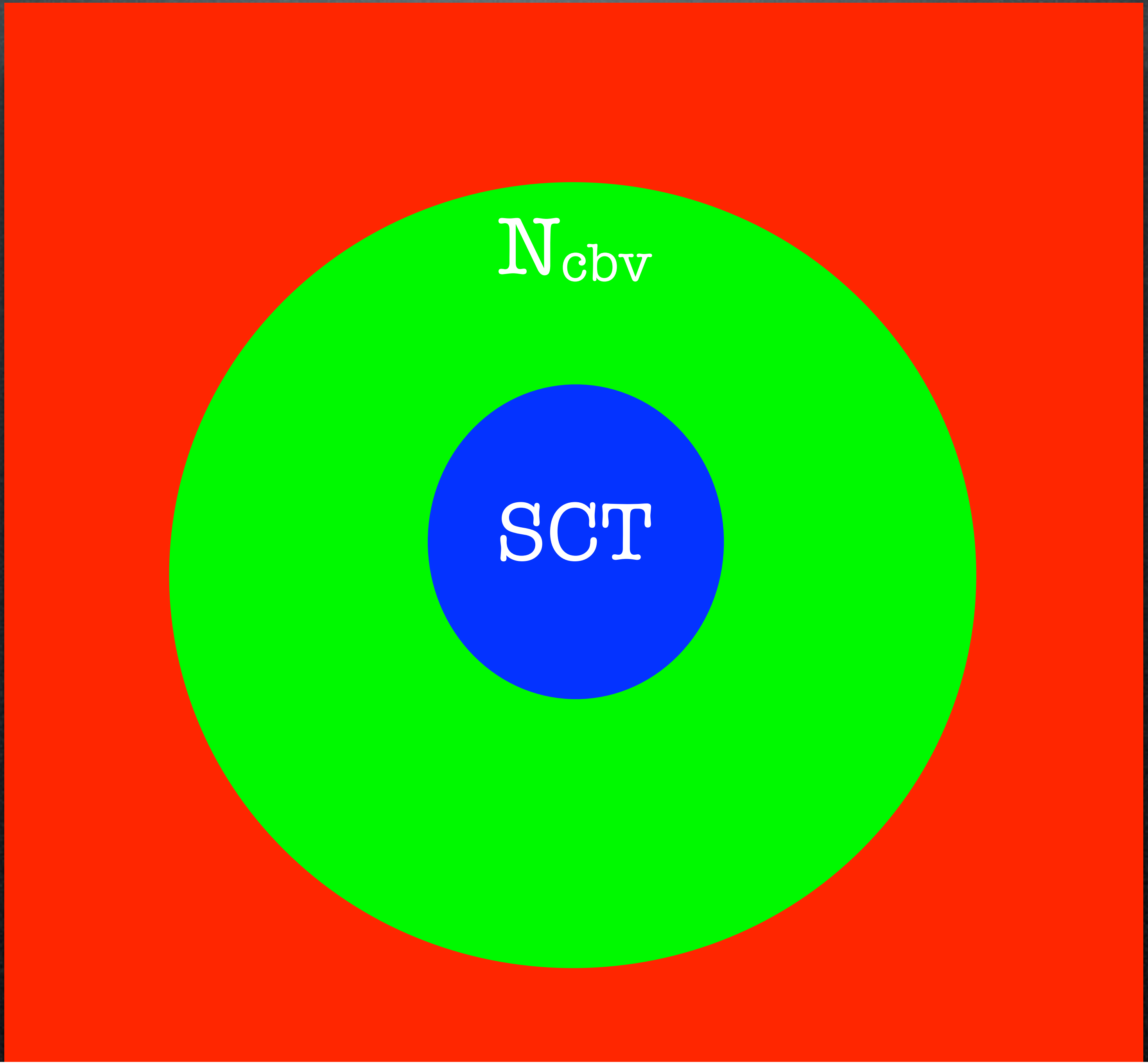


SCT for N



Theorem: SCT is decidable.
(Complexity?)

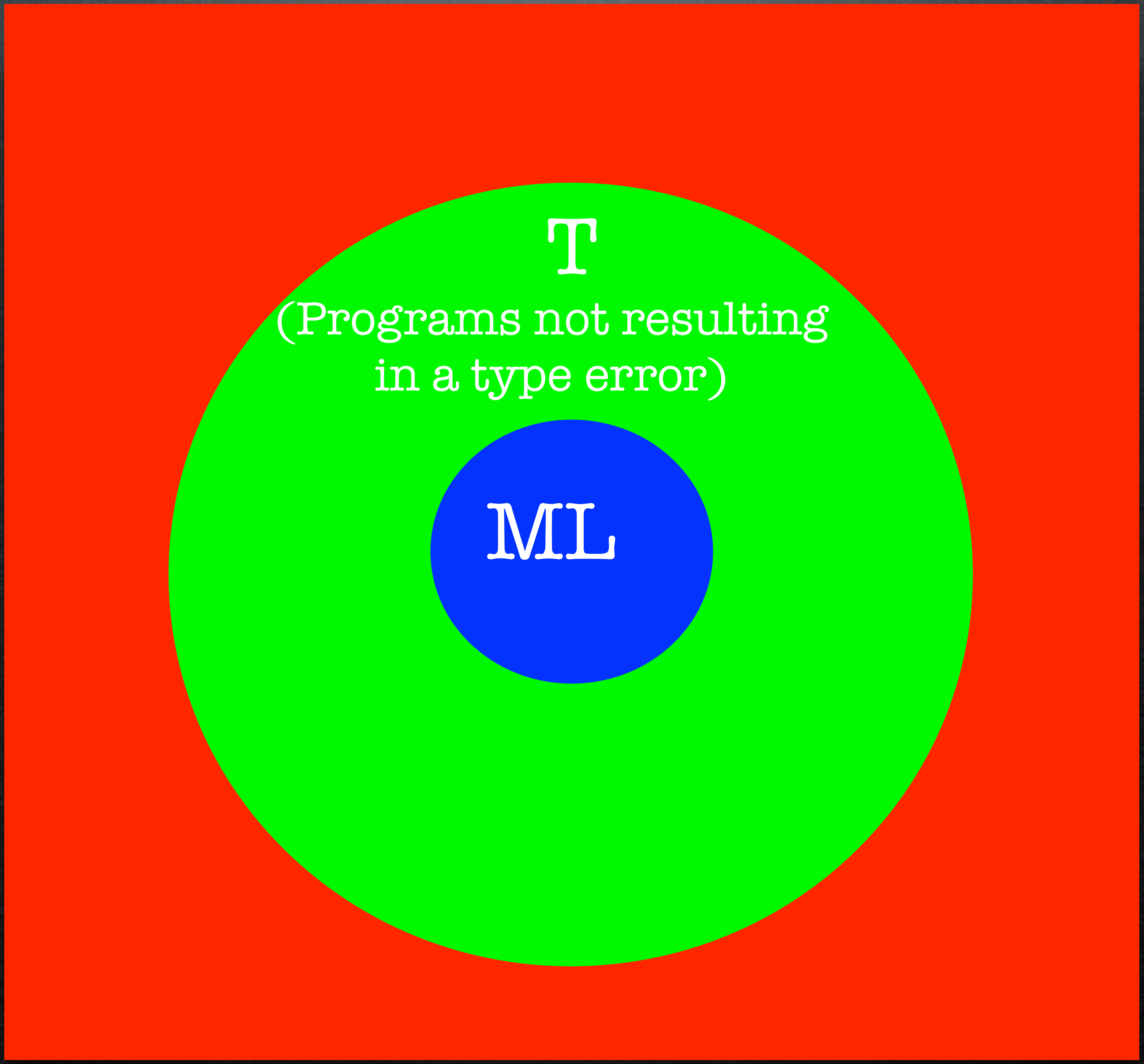
SCT for N



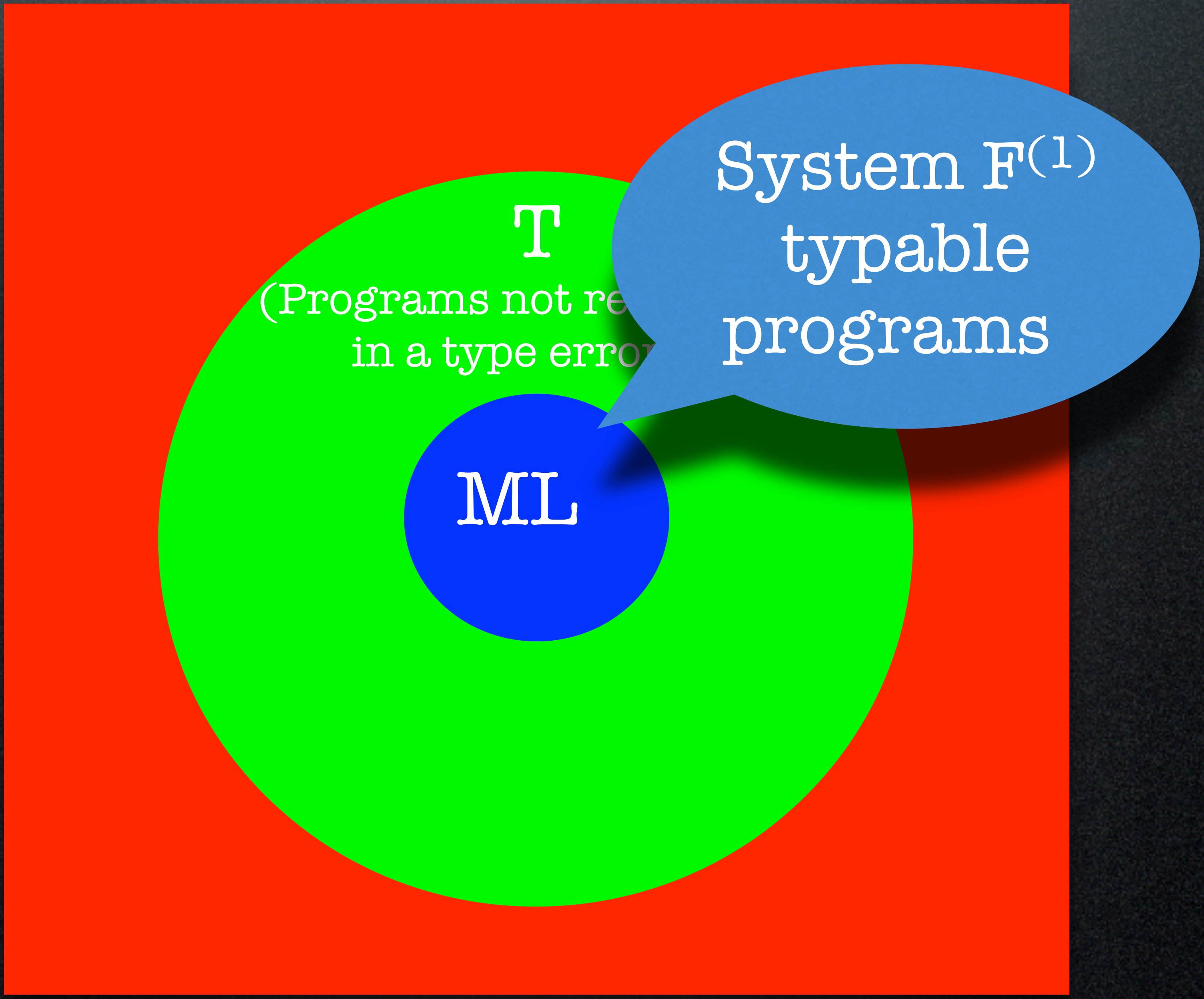
Theorem: SCT is decidable.
(Complexity?)

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

An analysis for type error freeness



An analysis for type error freeness



ML goodness

ML goodness

- Predictability:

ML goodness

- Predictability:
  - **Invariant** under let-reduction

ML goodness

- Predictability:
  - **Invariant** under let-reduction
    - ML(let x = e in e’) <=> ML(e’[e/x])

ML goodness

- Predictability:
  - **Invariant** under let-reduction
    - ML(let x = e in e’) <=> ML(e’[e/x])
  - **Preservation** under beta-reduction

ML goodness

- Predictability:
  - **Invariant** under let-reduction
    - ML(let x = e in e’) <=> ML(e’[e/x])
  - **Preservation** under beta-reduction
    - ML((λx.e)e’) => ML(e[e’/x])

ML goodness

- Predictability:
  - **Invariant** under let-reduction
    - ML(let x = e in e’) <=> ML(e’[e/x])
  - **Preservation** under beta-reduction
    - ML((λx.e)e’) => ML(e[e’/x])
  - **Preservation** under eta-reduction

ML goodness

- Predictability:
  - **Invariant** under let-reduction
    - ML(let x = e in e’) <=> ML(e’[e/x])
  - **Preservation** under beta-reduction
    - ML((λx.e)e’) => ML(e[e’/x])
  - **Preservation** under eta-reduction
    - ML(λx.ex) => ML(e)

ML goodnes

- Predictability:
  - **Invariant** under let-red
    - ML(let x = e in e’) <=> ML(e’[e/x])
  - **Preservation** under beta-reduction
    - ML((λx.e)e’) => ML(e[e’/x])
  - **Preservation** under eta-reduction
    - ML(λx.ex) => ML(e)

ML is “semantic” for let-expressions: Context sensitivity for 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 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

ML typability as static analysis for type error free

No, ML is not improvable for type error detection

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

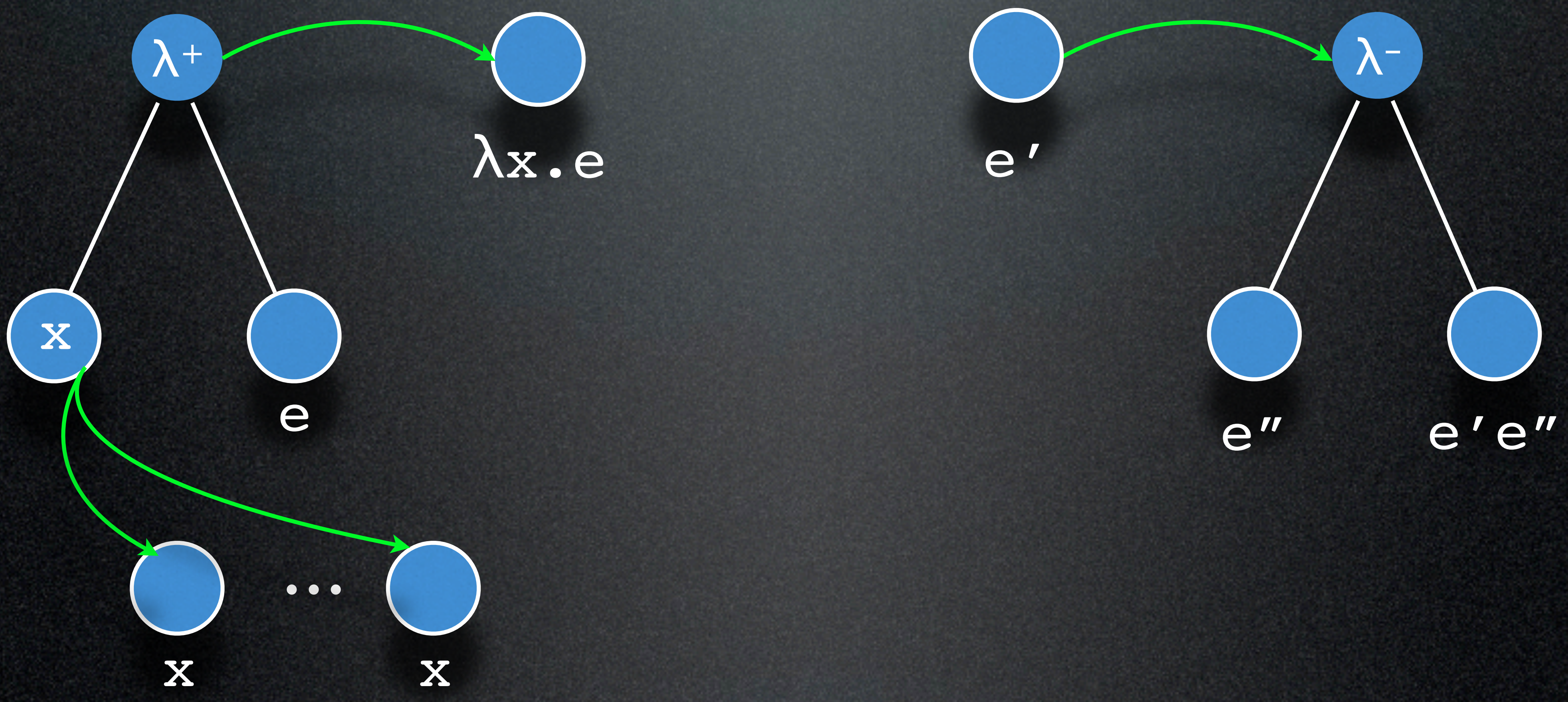
Henglein, A Lower Bound for Full Polymorphic Type Inference: Girard-Reynolds Typability is DEXPTIME-hard, Utrecht U. TR RUU-CS-90-14, 1990

mVFA

0CFA in direct style

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

1. Base flow rules, resulting in graph G:



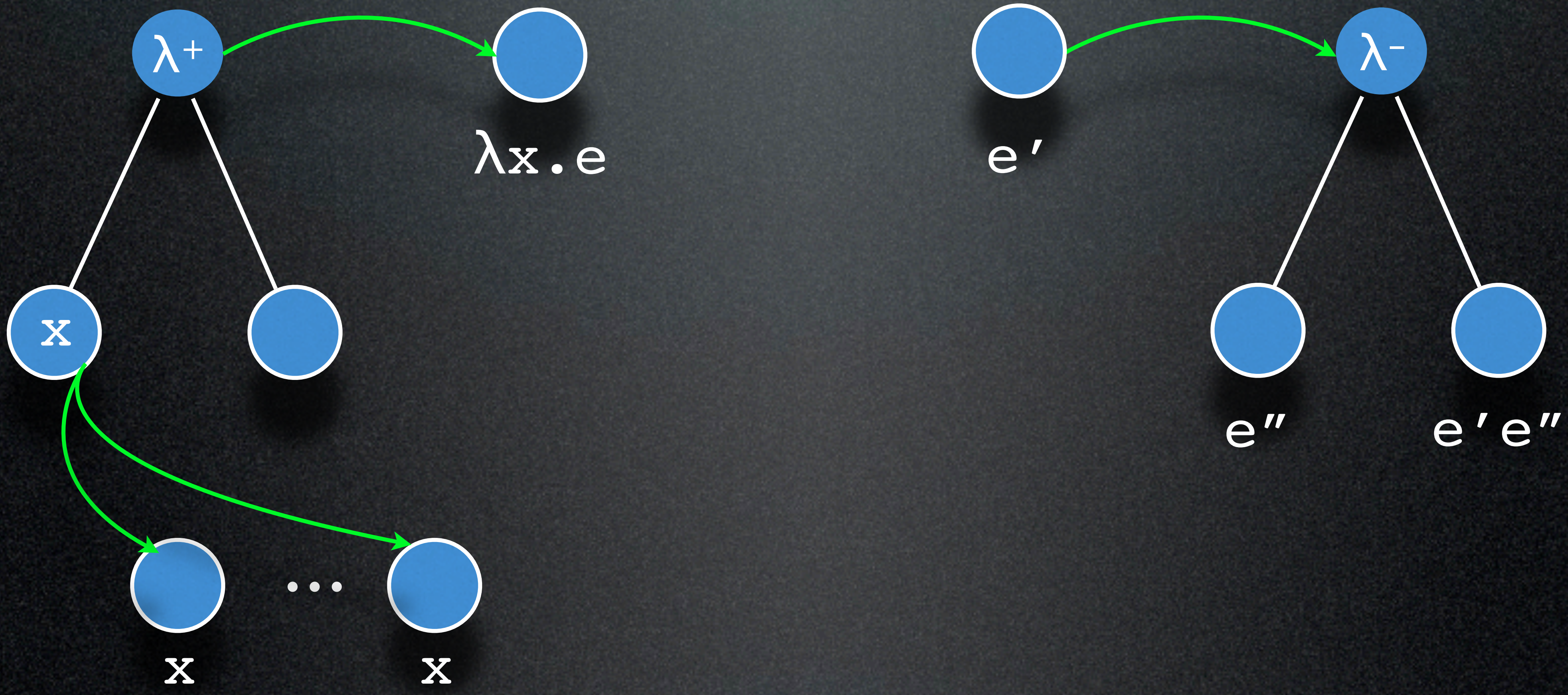
mVFA

0CFA in direct style

O(n) nodes

O(n) edges

Out- and indegree **1**,λ if affine λ-term



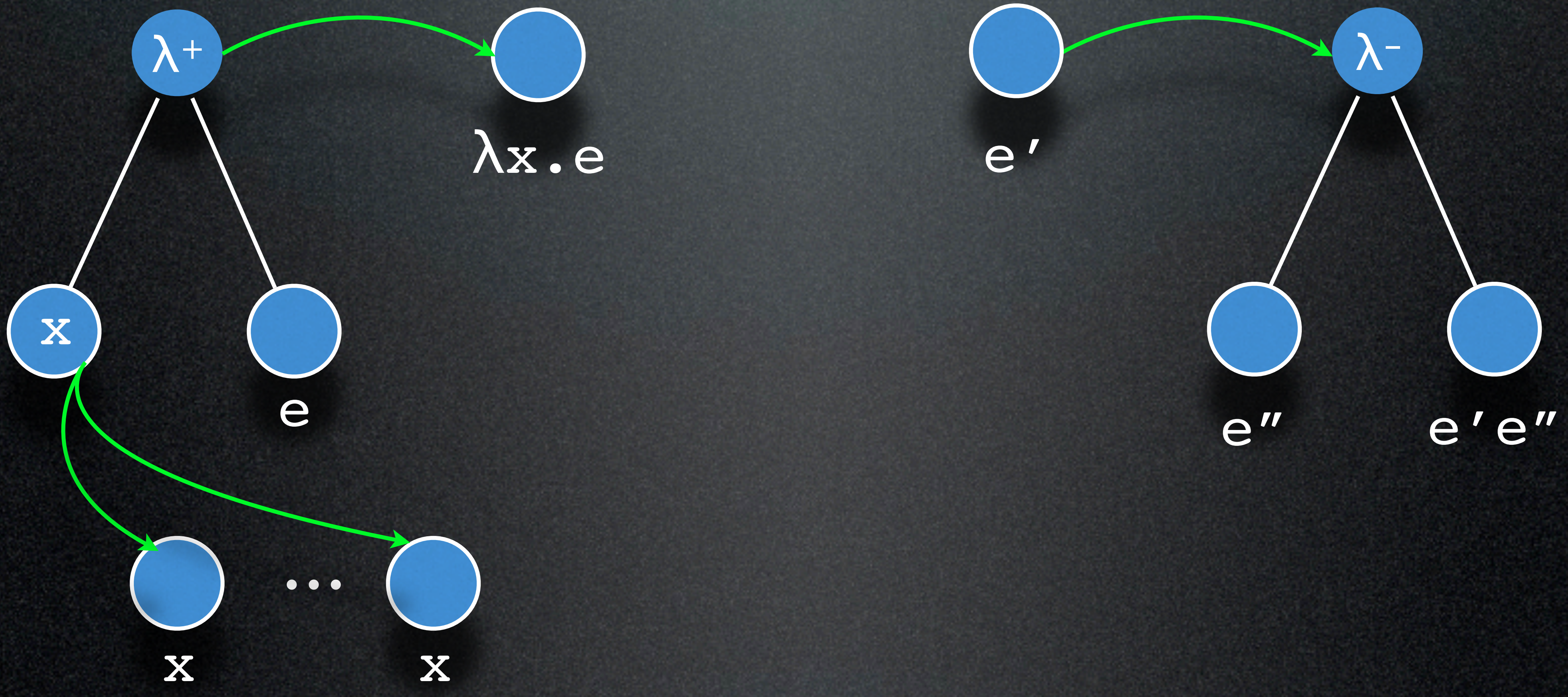
mVFA

0CFA in direct style

O(n) nodes

O(n) edges

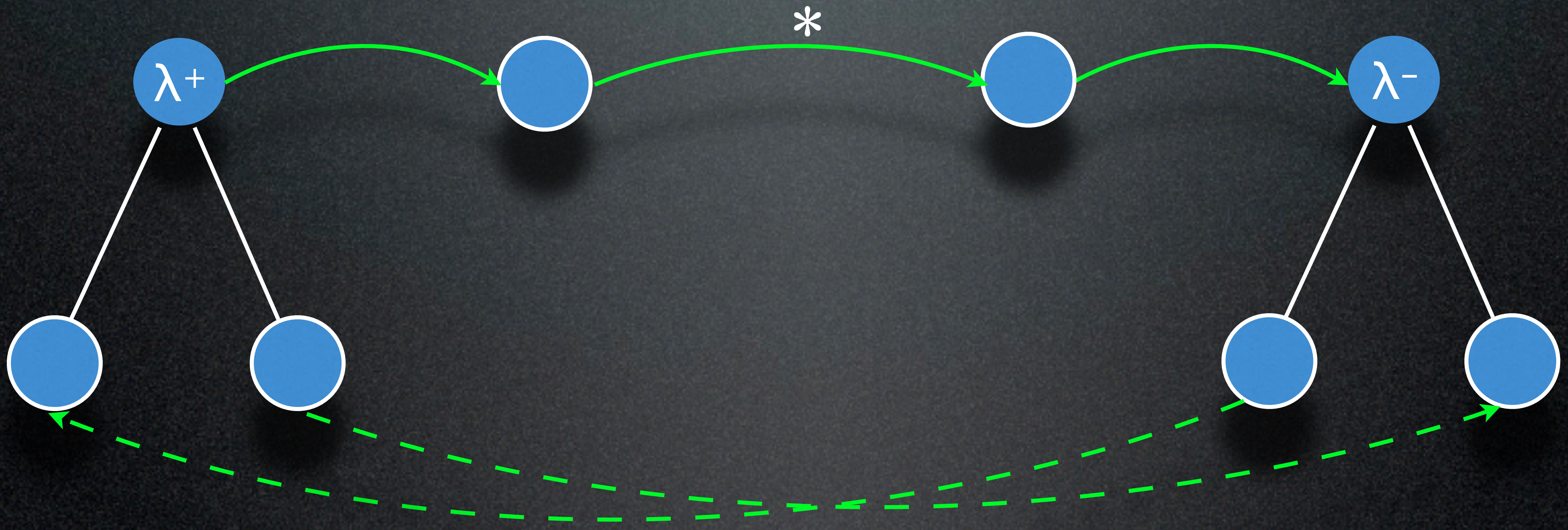
Out- and indegree **1**,λ if affine λ-term



mVFA

0CFA in direct style

2. Closure rule:



mVFA

OCFA in direct style

Algorithm:
Close base graph under closure rule, resulting in graph G.

mVFA

OCFA in direct style

Theorem: mVFA can be implemented in time $O(d\, m^* + p\, n + q)$, where

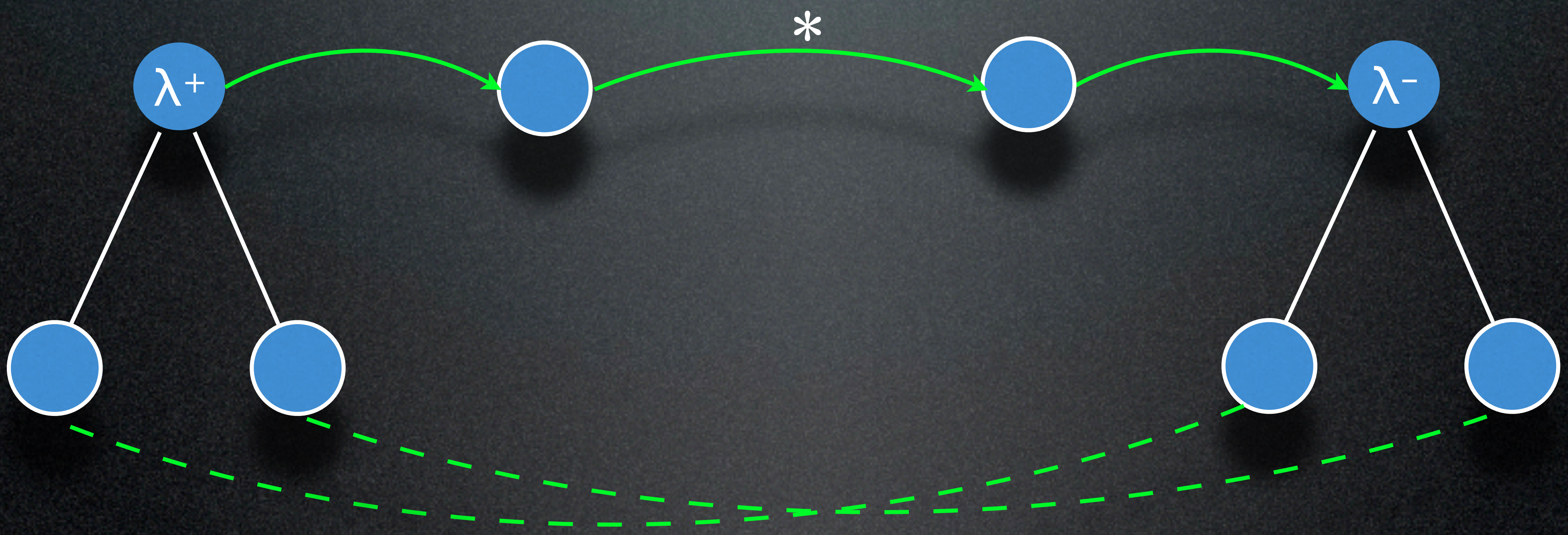
- n: number of nodes
- d: maximum outdegree of G,
- m^*: number of flow edges in G^* (flow-transitive closure of G),
- p: number of closure rule applications.
- q: number of reachability queries

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

sVFA

Simple monomorphic VFA

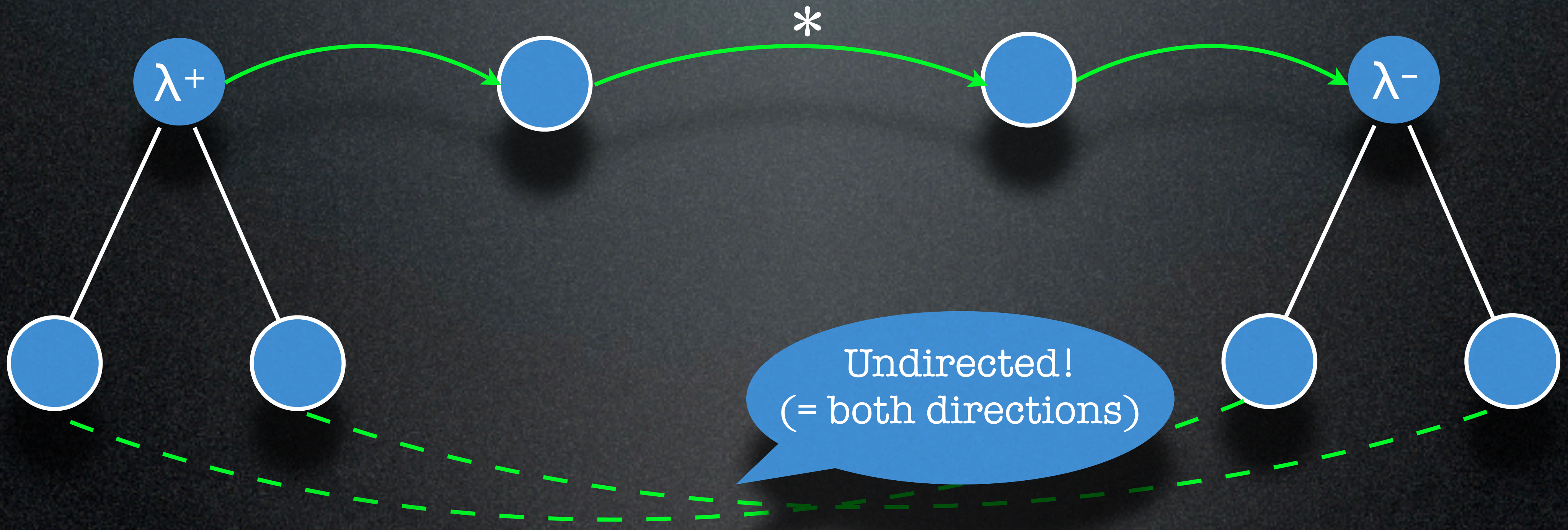
1. Base rules: As for mVFA
2. Closure rule:



sVFA

Simple monomorphic VFA

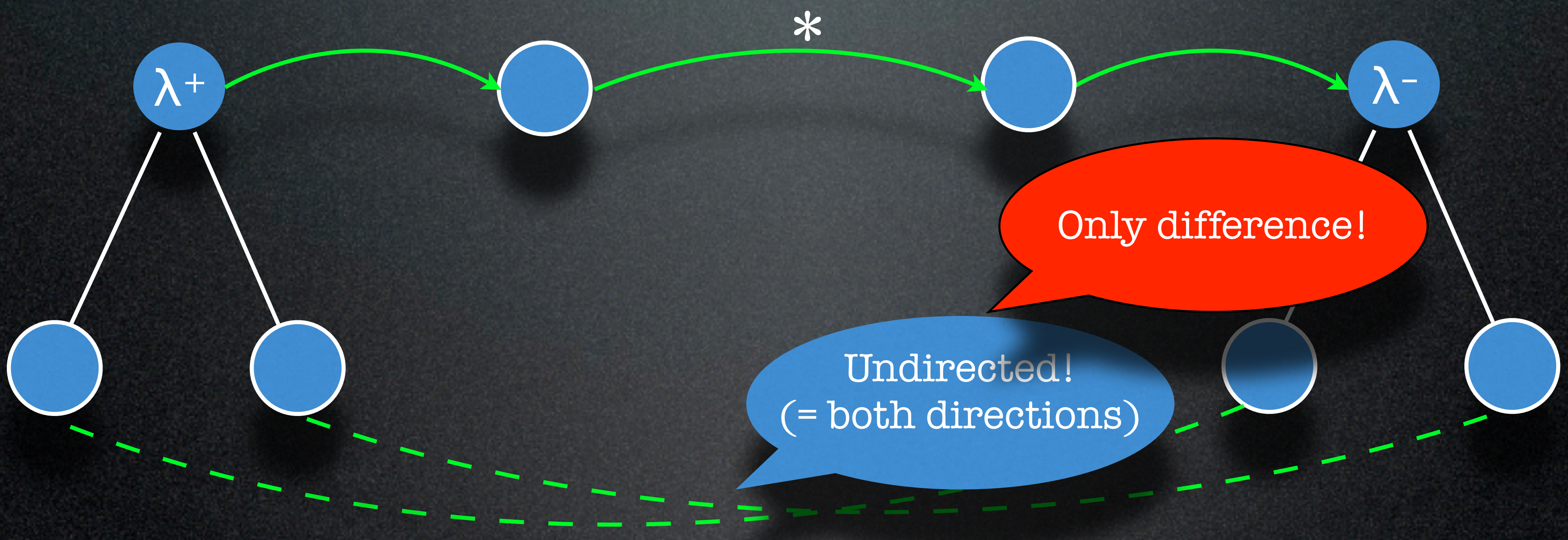
1. Base rules: As for mVFA
2. Closure rule:



sVFA

Simple monomorphic VFA

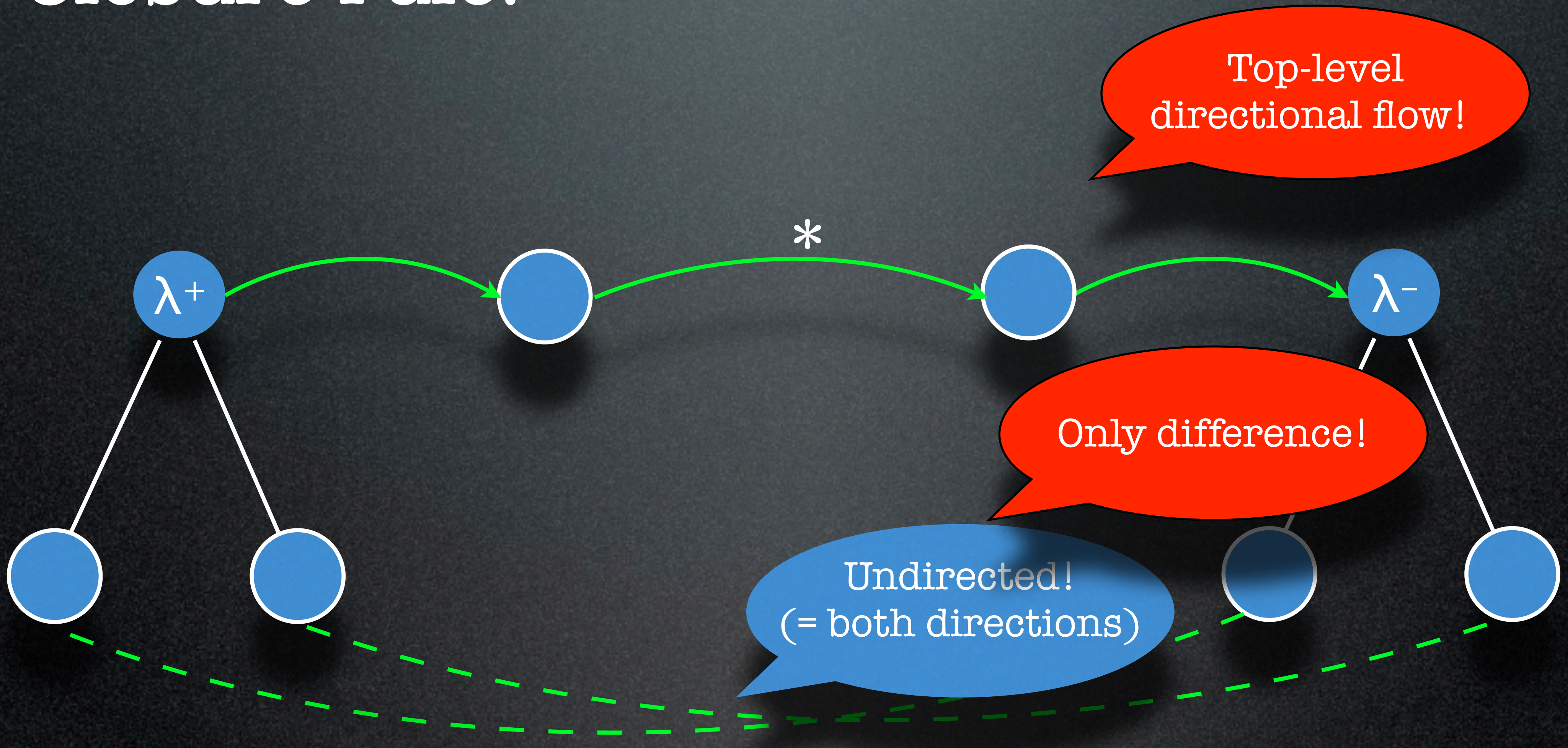
1. Base rules: As for mVFA
2. Closure rule:



sVFA

Simple monomorphic VFA

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



sVFA

Simple monomorphic VFA

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

sVFA

Simple monomorphic VFA

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 monomorphic VFA

- 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

sub0-CFA

…

(similar characterization)

sVFA predictability

sVFA predictability

- sVFA is invariant under

sVFA predictability

- sVFA is invariant under
  - linear beta-reduction

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

sVFA predictability

also for sub0-CFA

Theorem:
sVFA reachability is P-complete

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

sVFA predictability

also for sub0-CFA

Theorem:
sVFA reachability is P-complete

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

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

sVFA predictability

also for sub0-CFA

Theorem:
sVFA rea te

Van Horn, Mairson

Follows from proof method 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

Adaptiveness

- Assume S0 $\subseteq$ S1 $\subseteq$ P, with algorithms A0, A1 for S0, S1, respectively.

Adaptiveness

- Assume S0 ⊆ S1 ⊆ P, with algorithms A0, A1 for S0, S1, respectively.
- A1 is **adaptive** over A0 if its (time) complexity is < 2 times the complexity of A0 on instances from S0.

Adaptiveness

- Assume S0 $\subseteq$ S1 $\subseteq$ P, with algorithms A0, A1 for S0, S1, respectively.
- A1 is **adaptive** over A0 if its (time) complexity is < 2 times the complexity of A0 on instances from S0.
  - A1 is allowed to take substantially more time than A0 on instances outside S0.

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

Questions

- Are the various kCFA-algorithms adaptive (over sVFA or sub0-CFA)?

Questions

- Are the various kCFA-algorithms adaptive (over sVFA or sub0-CFA)?
- Is (functional) kCFA improvable for $k \geq 1$?

Questions

- Are the various kCFA-algorithms adaptive (over sVFA or sub0-CFA)?
- Is (functional) kCFA improvable for $k \geq 1$?
- Is SCT improvable? How predictable is it?

Questions

- Are the various kCFA-algorithms adaptive (over sVFA or sub0-CFA)?
- Is (functional) kCFA improvable for k≥1?
- Is SCT improvable? How predictable is it?
- ...