Partial Reductions for Kleene Algebra with Linear Hypotheses

Liam Chung, Tobias Kappé

14th April, 2026

LIACS, Leiden University
FoSSaCS 2026

These slides on my website:

~~Partial Reductions for Kleene Algebra with Linear Hypotheses~~

Liam Chung, Tobias Kappé

14th April, 2026

LIACS, Leiden University
FoSSaCS 2026

These slides on
my website:

Partially Decidable Program Equivalence

~~Partial Reductions for Kleene Algebra with Linear Hypotheses~~

Liam Chung, Tobias Kappé

14th April, 2026

LIACS, Leiden University
FoSSaCS 2026

These slides on my website:

Partially Decidable Program Equivalence

~~Partial Reductions for Kleene Algebra with Linear Hypotheses~~

...with Kleene Algebra

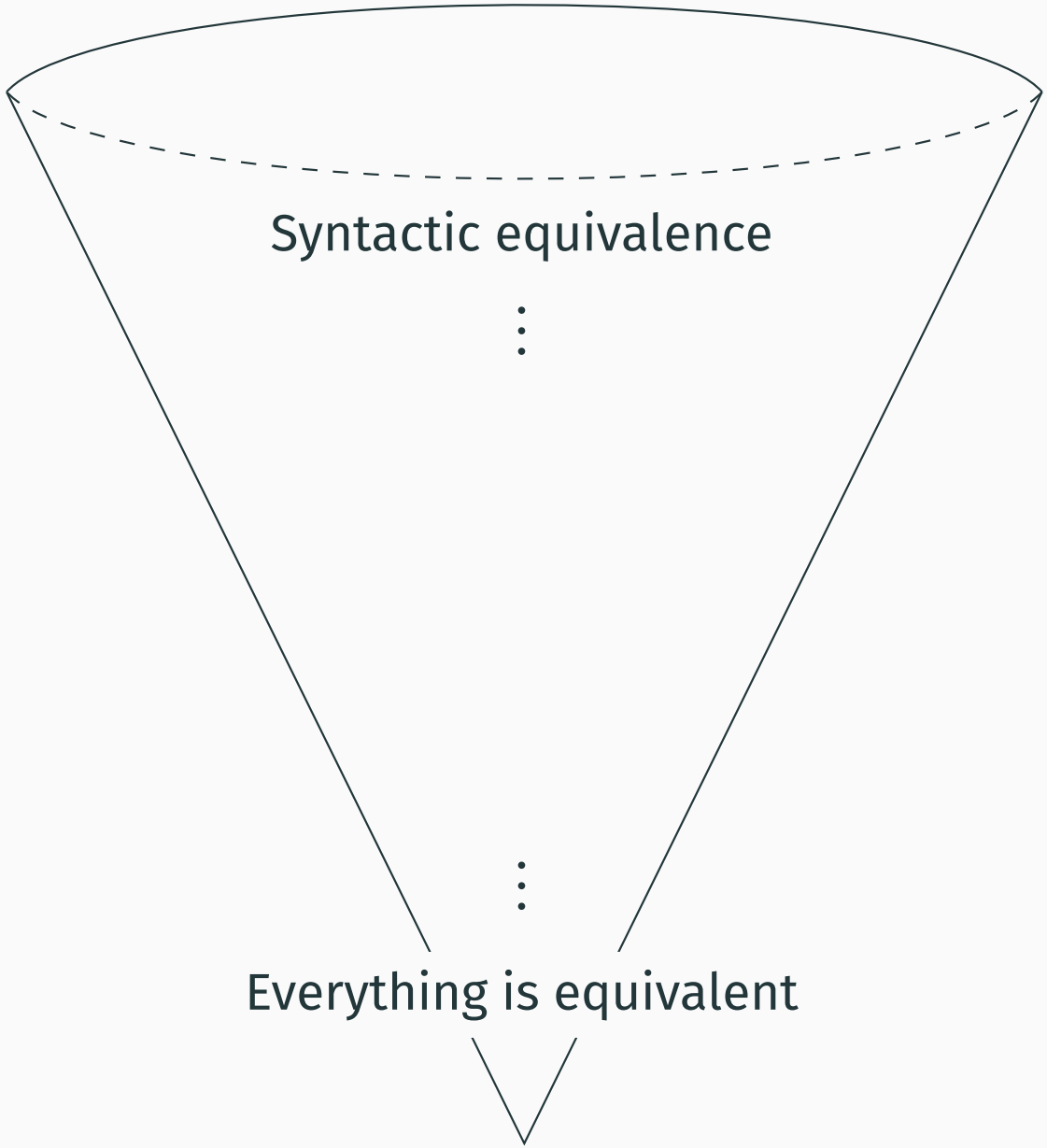
Liam Chung, Tobias Kappé

14th April, 2026

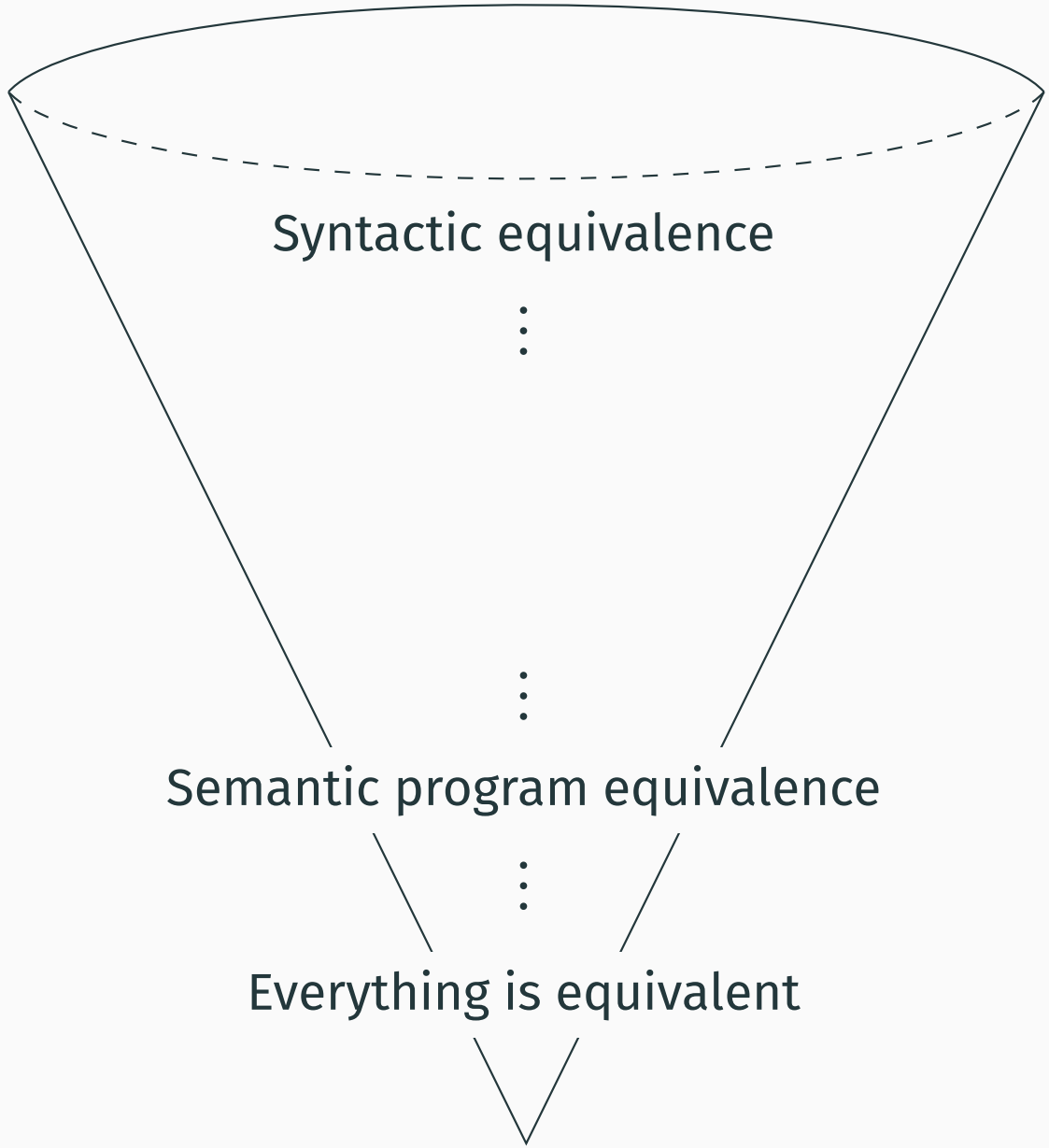
LIACS, Leiden University
FoSSaCS 2026

These slides on my website:

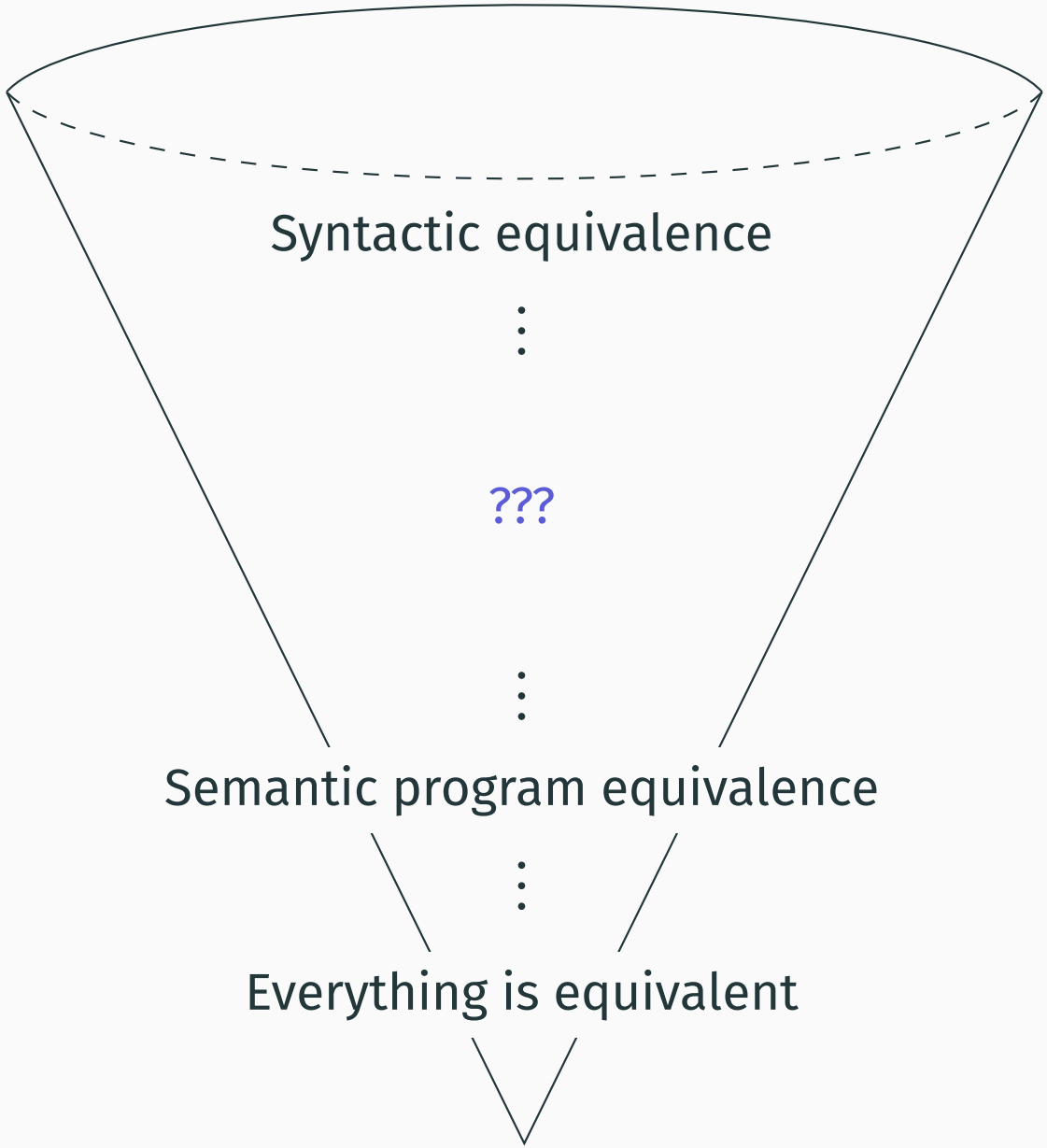
A Top-Down View on Program Equivalence Semantics



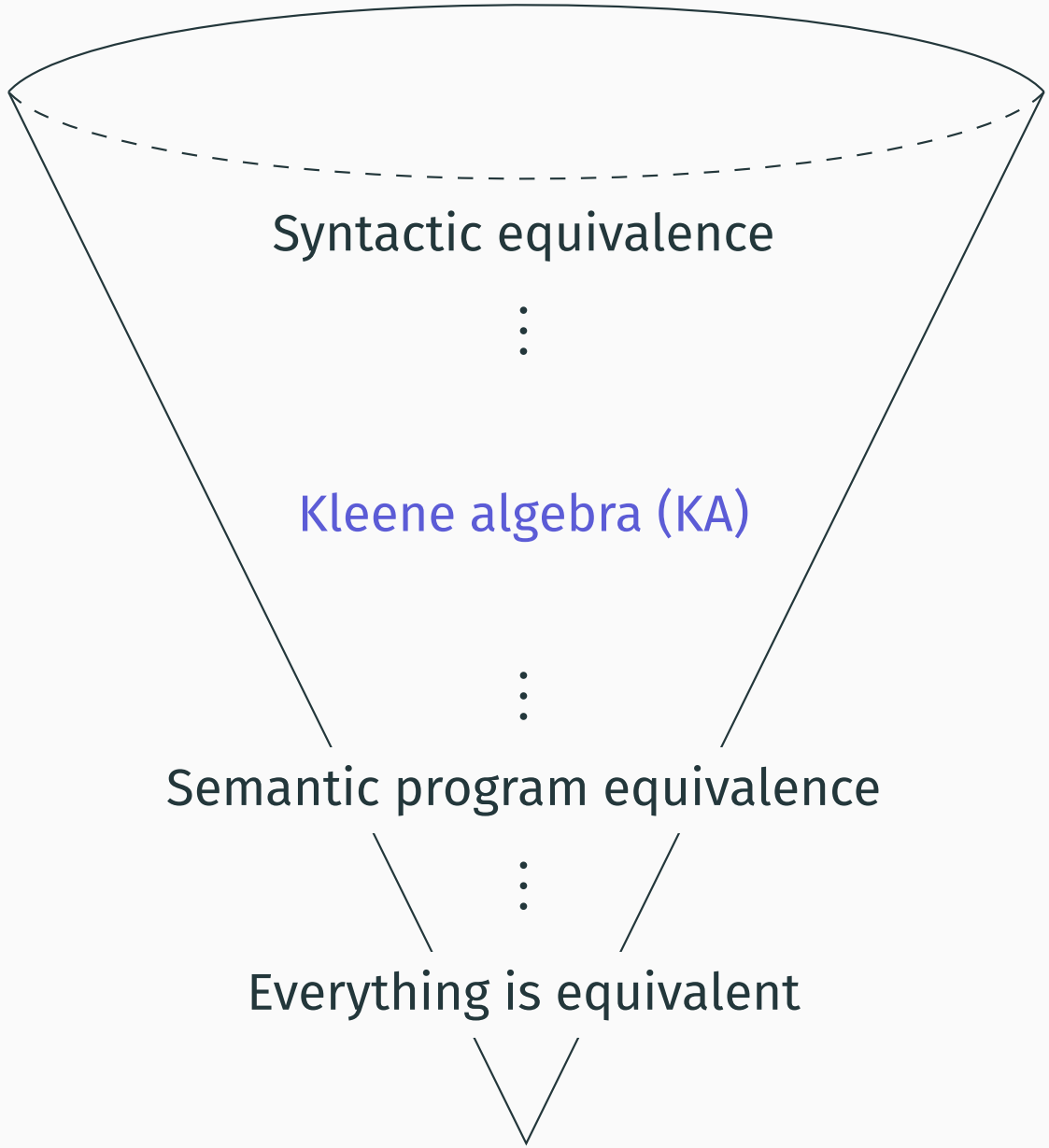
A Top-Down View on Program Equivalence Semantics



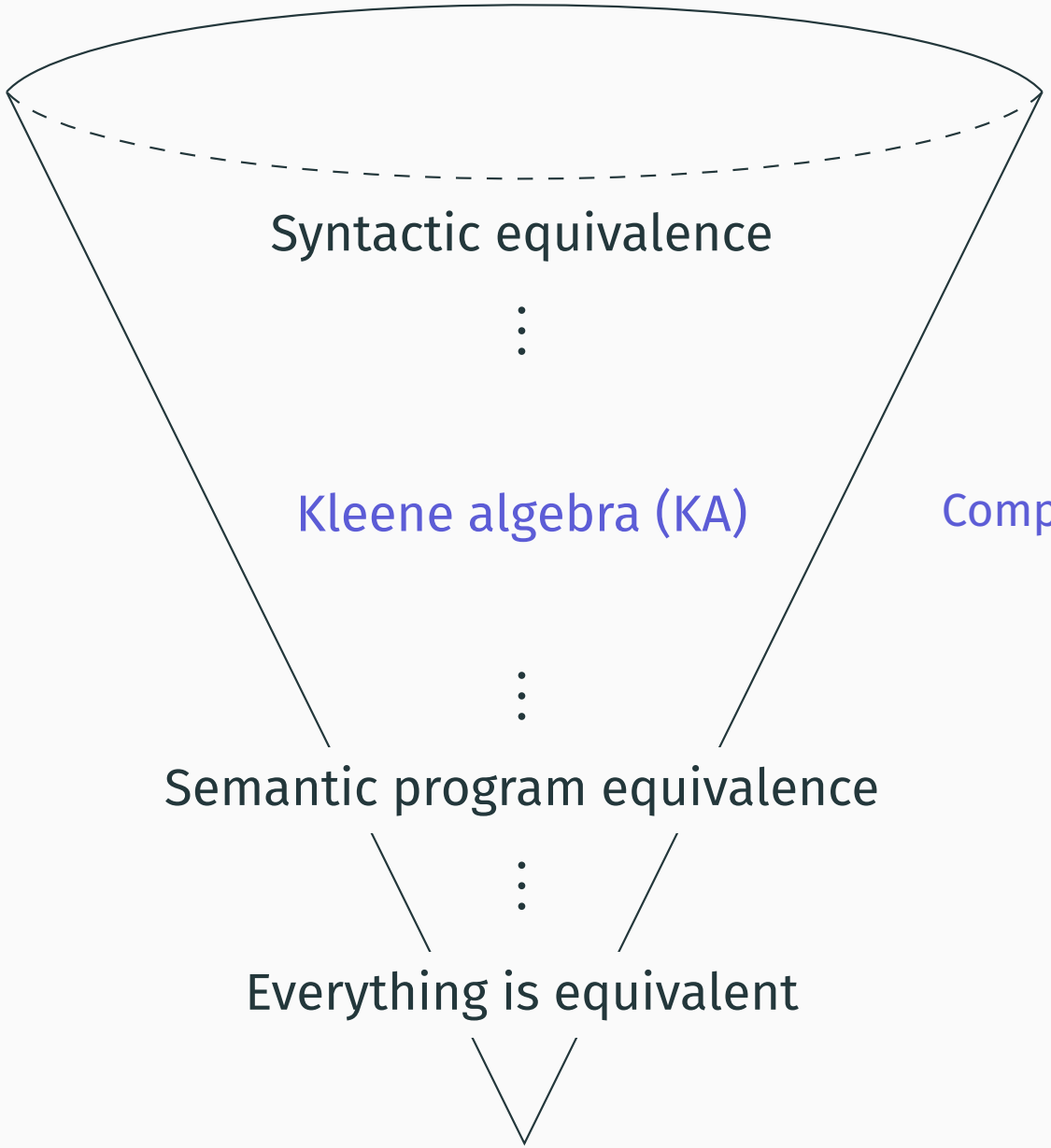
A Top-Down View on Program Equivalence Semantics



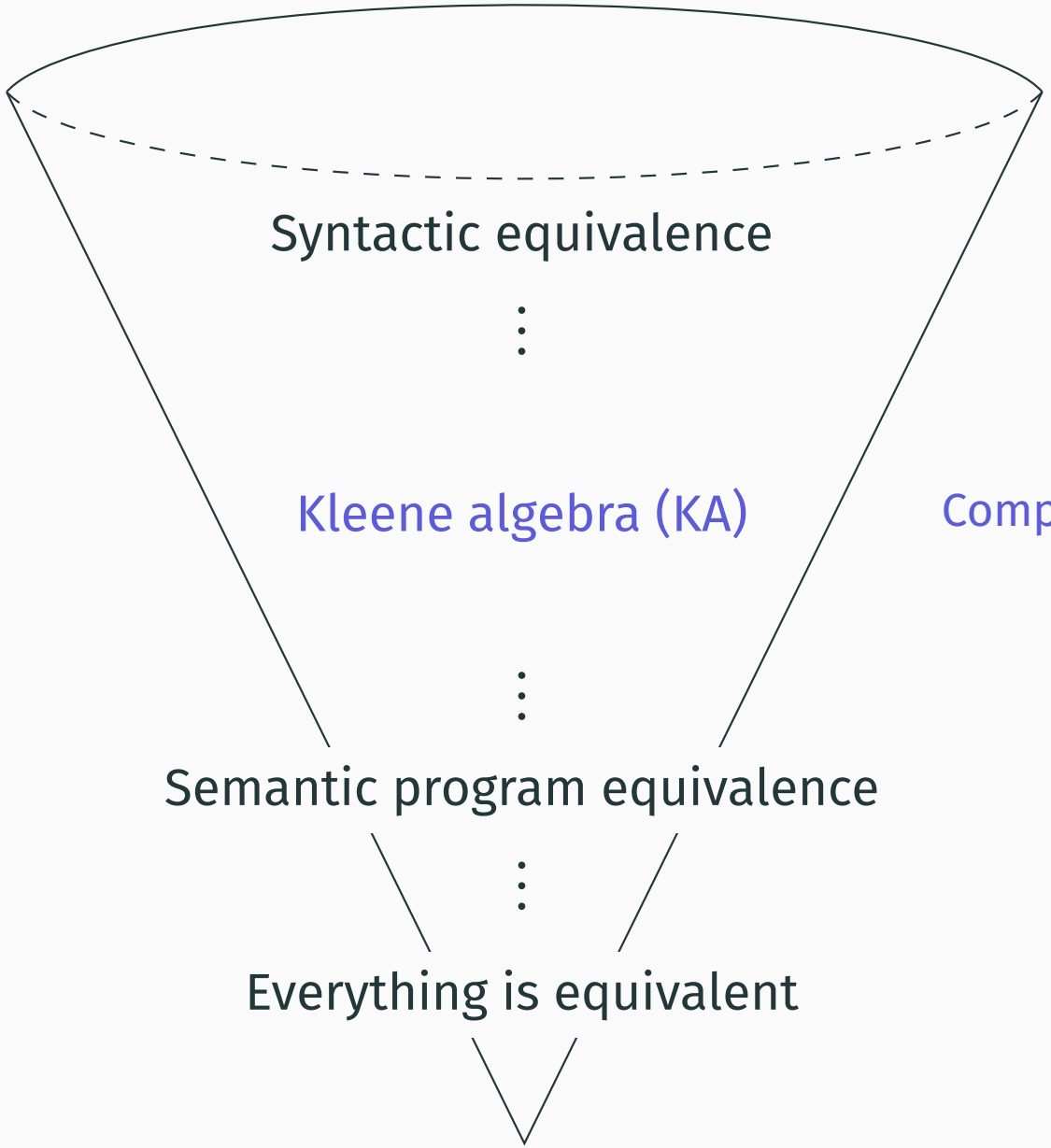
A Top-Down View on Program Equivalence Semantics



A Top-Down View on Program Equivalence Semantics



A Top-Down View on Program Equivalence Semantics



“Propositional” Equivalence

Regex ignore if operations can be...

- reordered
- replaced
- removed

“Propositional” Equivalence

Regex ignore if operations can be...

- reordered $\rightarrow$ $ab \leq ba$
- replaced $\rightarrow$ $b \leq a$
- removed $\rightarrow$ $a \leq aa$

“Propositional” Equivalence

Regex ignore if operations can be...

- reordered $\rightarrow$ $ab \leq ba$
- replaced $\rightarrow$ $b \leq a$
- removed $\rightarrow$ $a \leq aa$
- in general: sets of $e \leq f$

“Propositional” Equivalence

Regex ignore if operations can be...

- reordered $\rightarrow$ $ab \leq ba$
- replaced $\rightarrow$ $b \leq a$
- removed $\rightarrow$ $a \leq aa$
- in general: sets of $e \leq f$
- **in this paper:** $e \leq w$**, one at a time** “linear”

“Propositional” Equivalence

Regex ignore if operations can be...

- reordered $\rightarrow$ $ab \leq ba$
- replaced $\rightarrow$ $b \leq a$
- removed $\rightarrow$ $a \leq aa$
- in general: sets of $e \leq f$
- **in this paper: $e \leq w$, one at a time** “linear”

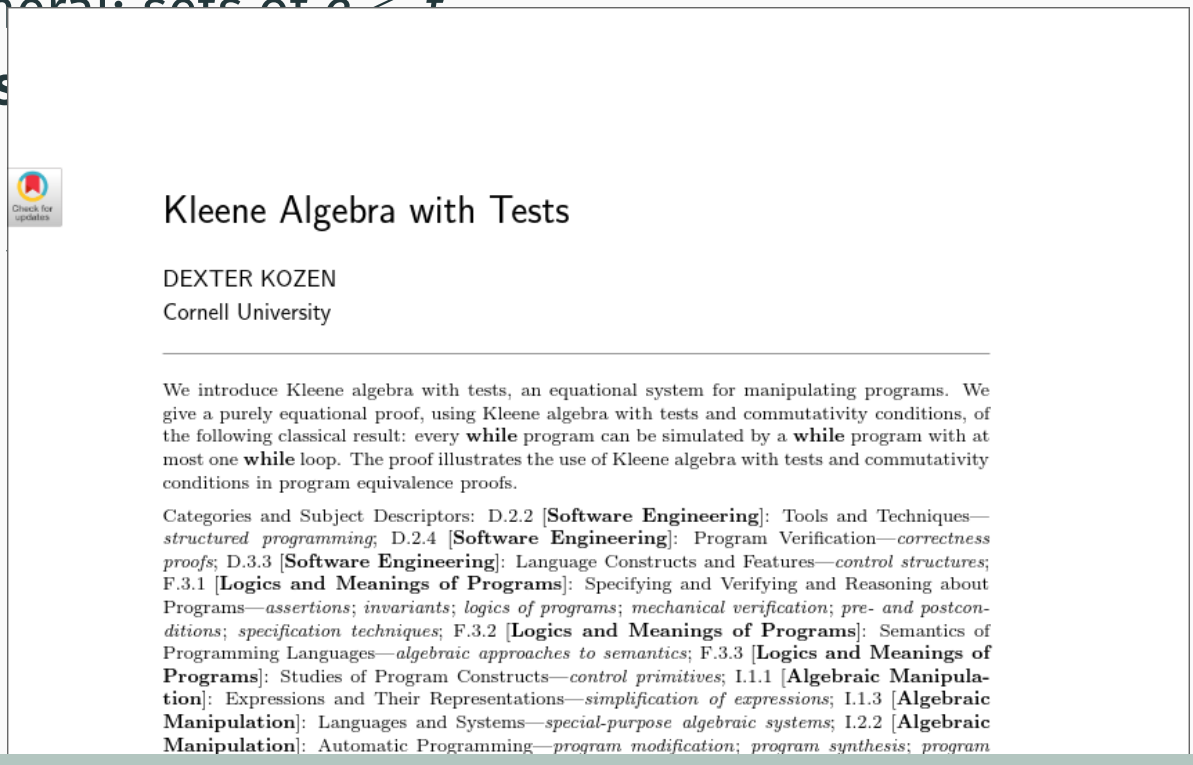
Approach 1: add “features” to KA booleans, concurrency...

“Propositional” Equivalence

Regex ignore if operations can be...

- reordered $\rightarrow$ $ab \leq ba$
- replaced $\rightarrow$ $b \leq a$
- removed $\rightarrow$ $a \leq aa$
- in general: sets of $e \leq f$
- **in this** “linear”

Approach



Kleene Algebra with Tests

DEXTER KOZEN
Cornell University

We introduce Kleene algebra with tests, an equational system for manipulating programs. We give a purely equational proof, using Kleene algebra with tests and commutativity conditions, of the following classical result: every **while** program can be simulated by a **while** program with at most one **while** loop. The proof illustrates the use of Kleene algebra with tests and commutativity conditions in program equivalence proofs.

Categories and Subject Descriptors: D.2.2 [**Software Engineering**]: Tools and Techniques—*structured programming*; D.2.4 [**Software Engineering**]: Program Verification—*correctness proofs*; D.3.3 [**Software Engineering**]: Language Constructs and Features—*control structures*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and Verifying and Reasoning about Programs—*assertions*; *invariants*; *logics of programs*; *mechanical verification*; *pre- and postconditions*; *specification techniques*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages—*algebraic approaches to semantics*; F.3.3 [**Logics and Meanings of Programs**]: Studies of Program Constructs—*control primitives*; I.1.1 [**Algebraic Manipulation**]: Expressions and Their Representations—*simplification of expressions*; I.1.3 [**Algebraic Manipulation**]: Languages and Systems—*special-purpose algebraic systems*; I.2.2 [**Algebraic Manipulation**]: Automatic Programming—*program modification*; *program synthesis*; *program*

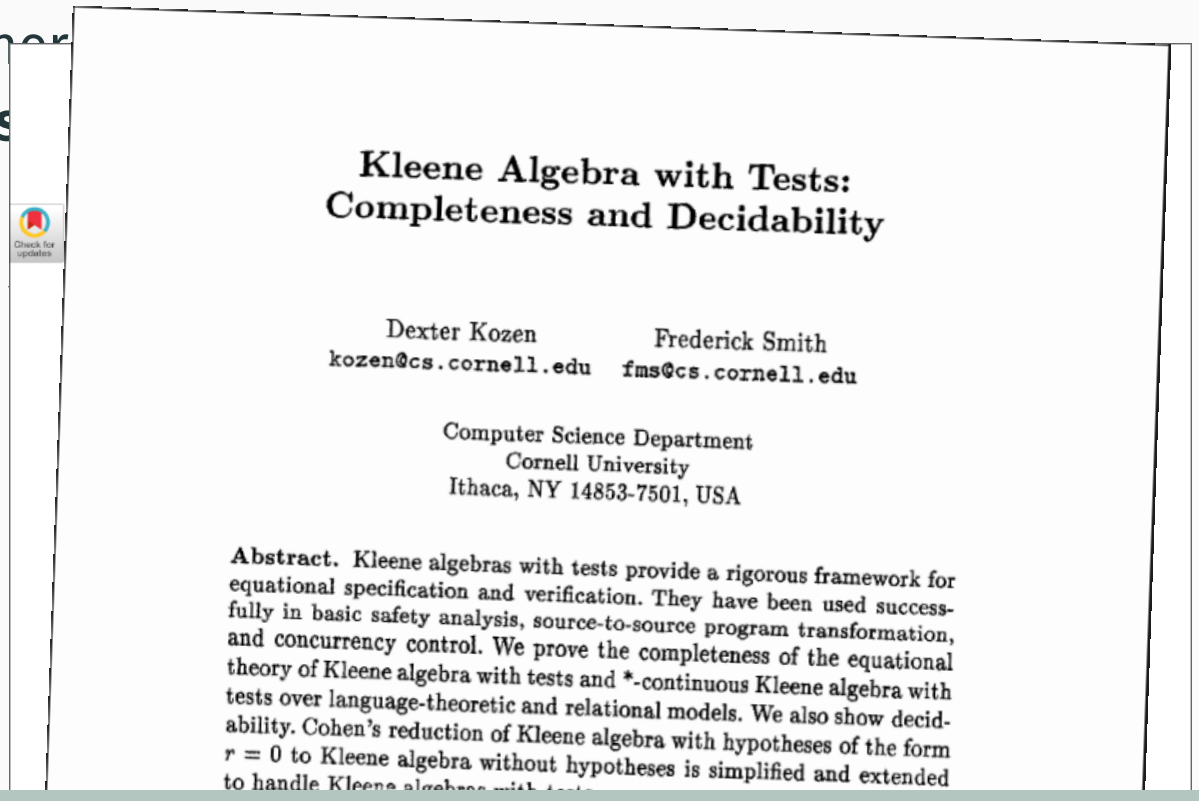
“Propositional” Equivalence

Regex ignore if operations can be...

- reordered $\rightarrow$ $ab \leq ba$
- replaced $\rightarrow$ $b \leq a$
- removed $\rightarrow$ $a \leq aa$
- in gen
- **in this**

“linear”

Approach



Kleene Algebra with Tests:
Completeness and Decidability

Dexter Kozen
kozen@cs.cornell.edu

Frederick Smith
fms@cs.cornell.edu

Computer Science Department
Cornell University
Ithaca, NY 14853-7501, USA

Abstract. Kleene algebras with tests provide a rigorous framework for equational specification and verification. They have been used successfully in basic safety analysis, source-to-source program transformation, and concurrency control. We prove the completeness of the equational theory of Kleene algebra with tests and *-continuous Kleene algebra with tests over language-theoretic and relational models. We also show decidability. Cohen's reduction of Kleene algebra with hypotheses of the form $r = 0$ to Kleene algebra without hypotheses is simplified and extended

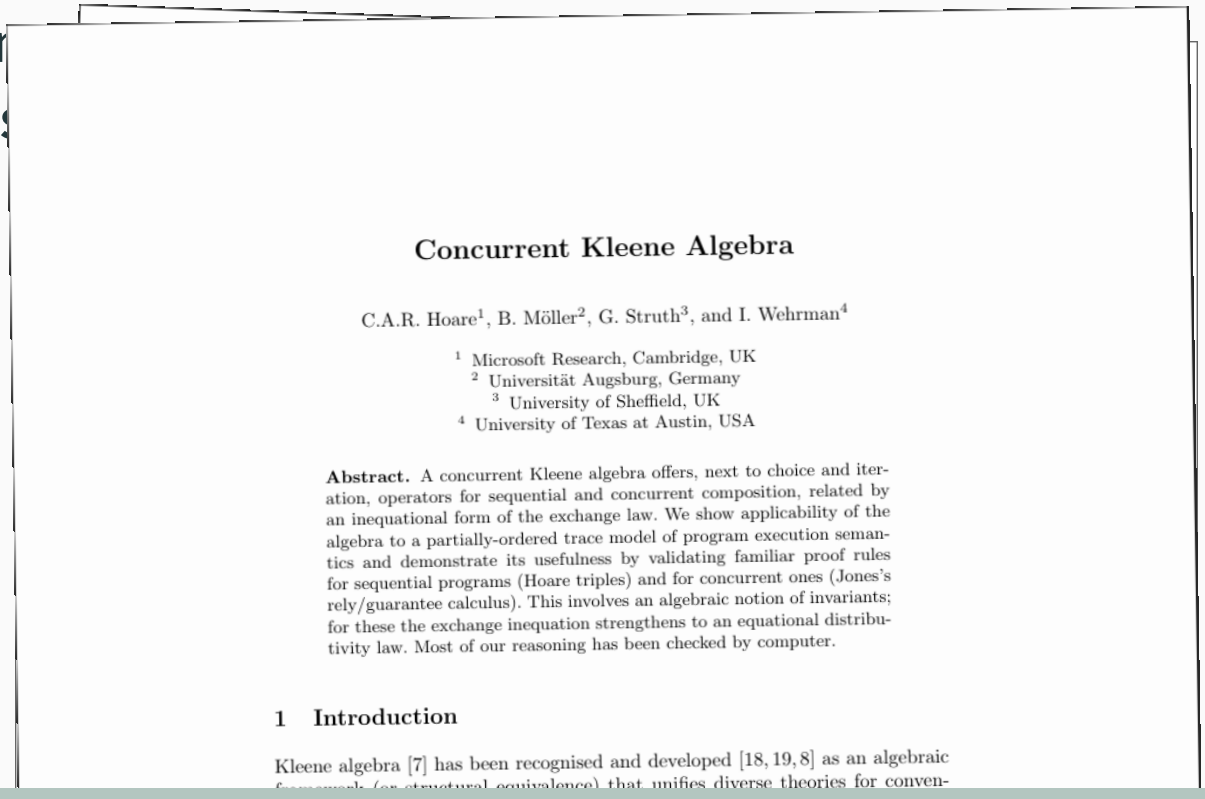
“Propositional” Equivalence

Regex ignore if operations can be...

- reordered $\rightarrow$ $ab \leq ba$
- replaced $\rightarrow$ $b \leq a$
- removed $\rightarrow$ $a \leq aa$
- in ge
- **in thi**

“linear”

Approach



Concurrent Kleene Algebra

C.A.R. Hoare[1], B. Möller[2], G. Struth[3], and I. Wehrman[4]

[1] Microsoft Research, Cambridge, UK
[2] Universität Augsburg, Germany
[3] University of Sheffield, UK
[4] University of Texas at Austin, USA

Abstract. A concurrent Kleene algebra offers, next to choice and iteration, operators for sequential and concurrent composition, related by an inequational form of the exchange law. We show applicability of the algebra to a partially-ordered trace model of program execution semantics and demonstrate its usefulness by validating familiar proof rules for sequential programs (Hoare triples) and for concurrent ones (Jones's rely/guarantee calculus). This involves an algebraic notion of invariants; for these the exchange inequation strengthens to an equational distributivity law. Most of our reasoning has been checked by computer.

1 Introduction

Kleene algebra [7] has been recognised and developed [18, 19, 8] as an algebraic

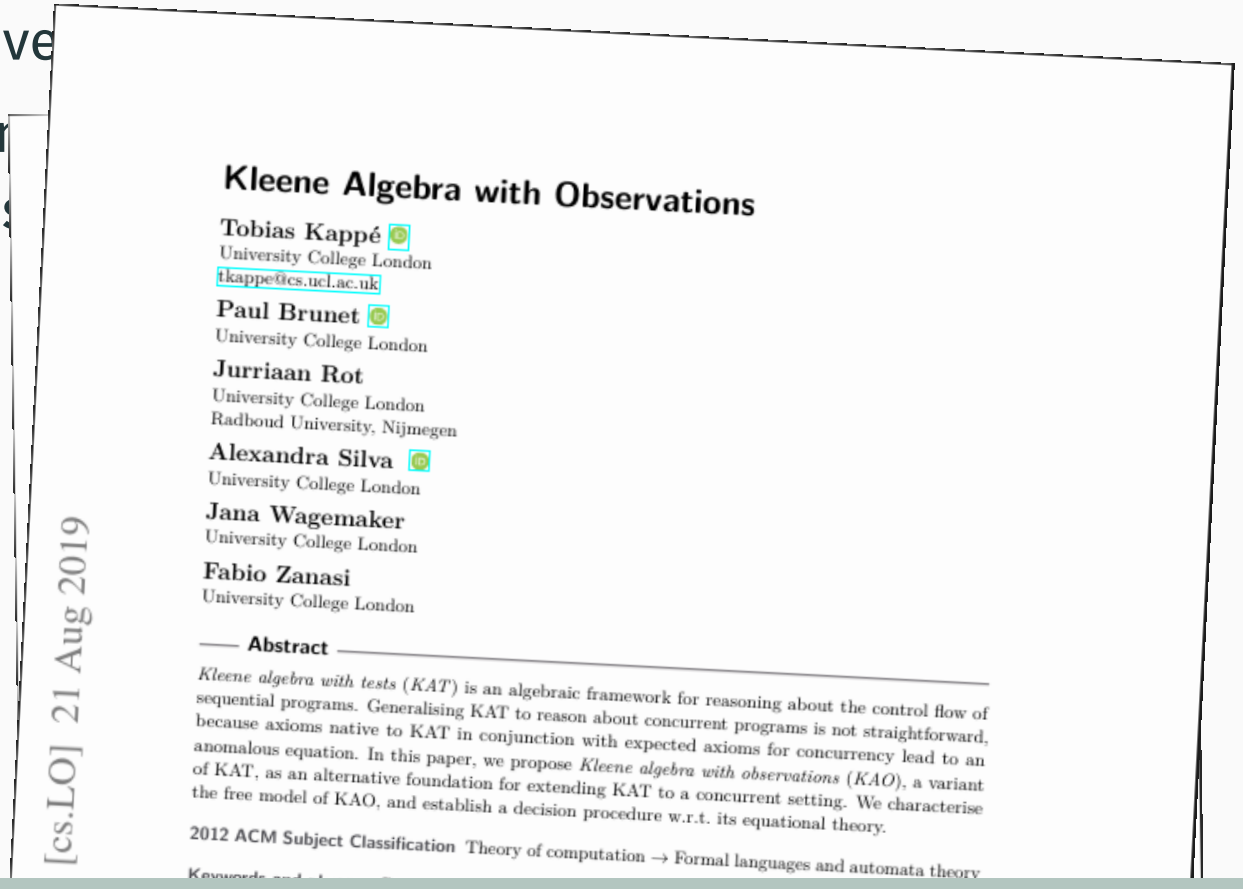
“Propositional” Equivalence

Regex ignore if operations can be...

- reordered $\rightarrow$ $ab \leq ba$
- replaced $\rightarrow$ $b \leq a$
- remove
- in ge
- **in thi**

“linear”

Approach



[cs.LO] 21 Aug 2019

Kleene Algebra with Observations

Tobias Kappé
University College London
tkappe@cs.ucl.ac.uk

Paul Brunet
University College London

Jurriaan Rot
University College London
Radboud University, Nijmegen

Alexandra Silva
University College London

Jana Wagemaker
University College London

Fabio Zanasi
University College London

Abstract

Kleene algebra with tests (*KAT*) is an algebraic framework for reasoning about the control flow of sequential programs. Generalising KAT to reason about concurrent programs is not straightforward, because axioms native to KAT in conjunction with expected axioms for concurrency lead to an anomalous equation. In this paper, we propose *Kleene algebra with observations* (*KAO*), a variant of KAT, as an alternative foundation for extending KAT to a concurrent setting. We characterise the free model of KAO, and establish a decision procedure w.r.t. its equational theory.

2012 ACM Subject Classification Theory of computation → Formal languages and automata theory

Keywords and

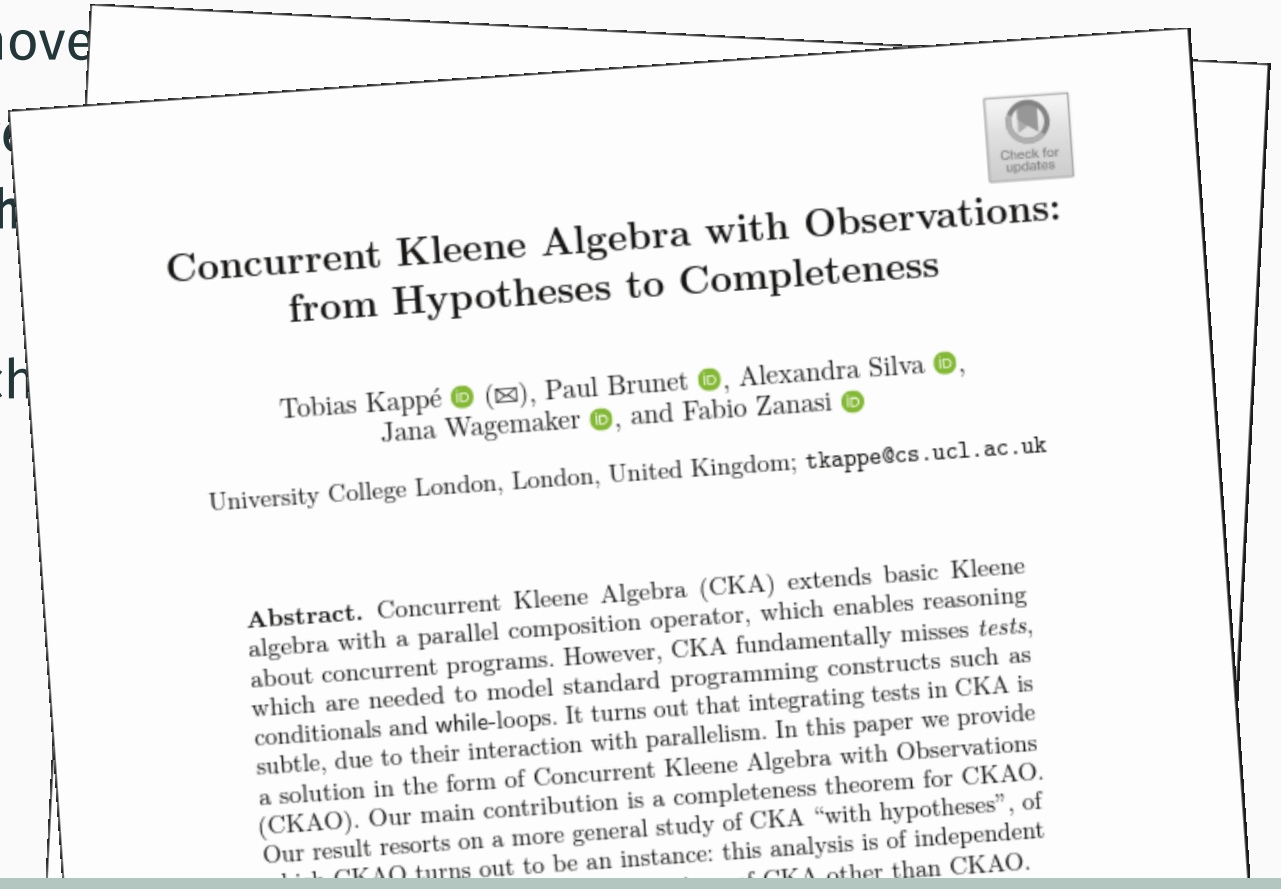
“Propositional” Equivalence

Regex ignore if operations can be...

- reordered $\rightarrow$ $ab \leq ba$
- replaced $\rightarrow$ $b \leq a$
- remove
- in g
- **in th**

“linear”

Approach



Concurrent Kleene Algebra with Observations: from Hypotheses to Completeness

Tobias Kappé (✉), Paul Brunet, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi

University College London, London, United Kingdom; tkappe@cs.ucl.ac.uk

Abstract. Concurrent Kleene Algebra (CKA) extends basic Kleene algebra with a parallel composition operator, which enables reasoning about concurrent programs. However, CKA fundamentally misses *tests*, which are needed to model standard programming constructs such as conditionals and while-loops. It turns out that integrating tests in CKA is subtle, due to their interaction with parallelism. In this paper we provide a solution in the form of Concurrent Kleene Algebra with Observations (CKAO). Our main contribution is a completeness theorem for CKAO. Our result resorts on a more general study of CKA “with hypotheses”, of

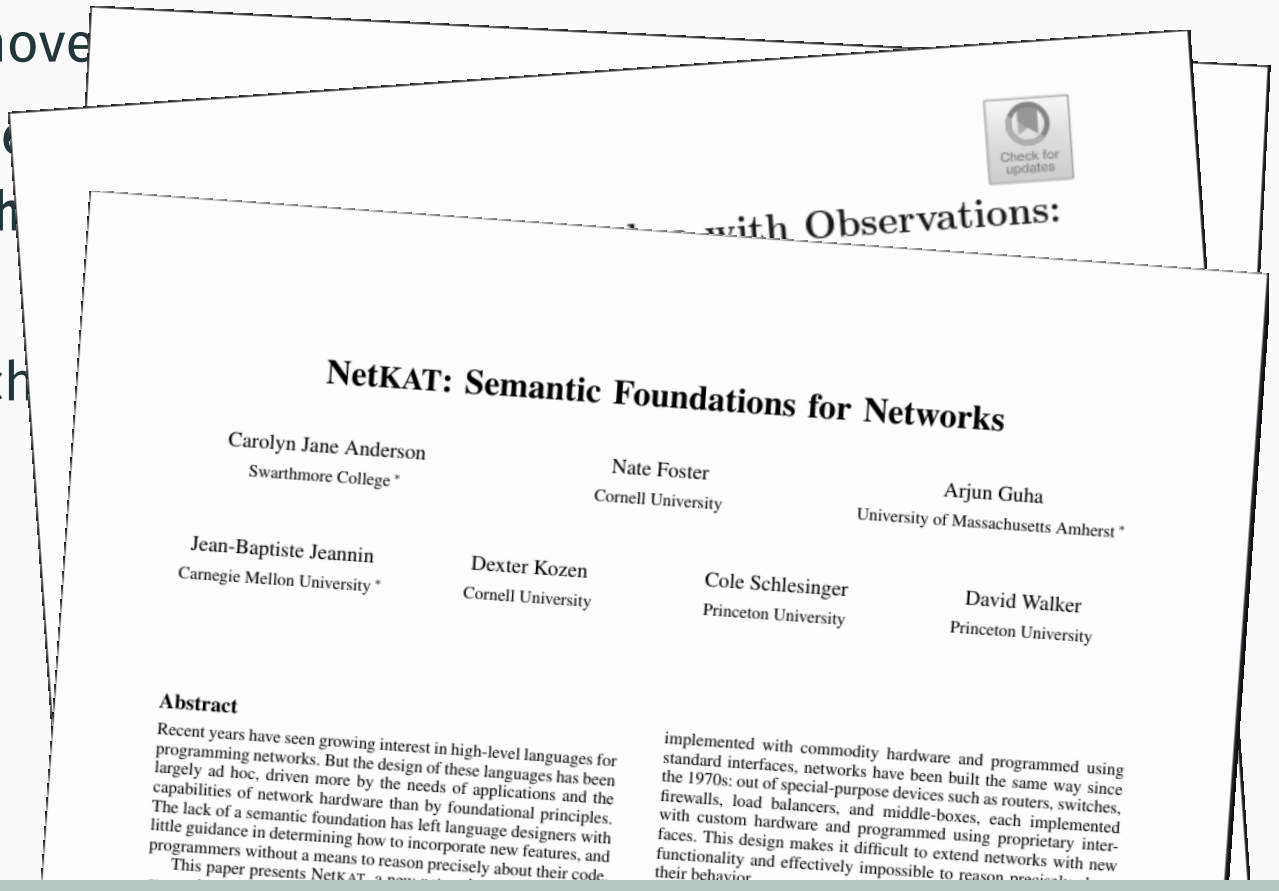
“Propositional” Equivalence

Regex ignore if operations can be...

- reordered $\rightarrow$ $ab \leq ba$
- replaced $\rightarrow$ $b \leq a$
- remove
- in g
- **in th**

“linear”

Approach



Check for updates

with Observations:

NetKAT: Semantic Foundations for Networks

Carolyn Jane Anderson
Swarthmore College *

Nate Foster
Cornell University

Arjun Guha
University of Massachusetts Amherst *

Jean-Baptiste Jeannin
Carnegie Mellon University *

Dexter Kozen
Cornell University

Cole Schlesinger
Princeton University

David Walker
Princeton University

Abstract

Recent years have seen growing interest in high-level languages for programming networks. But the design of these languages has been largely ad hoc, driven more by the needs of applications and the capabilities of network hardware than by foundational principles. The lack of a semantic foundation has left language designers with little guidance in determining how to incorporate new features, and programmers without a means to reason precisely about their code. This paper presents NetKAT, a new

implemented with commodity hardware and programmed using standard interfaces, networks have been built the same way since the 1970s: out of special-purpose devices such as routers, switches, firewalls, load balancers, and middle-boxes, each implemented with custom hardware and programmed using proprietary interfaces. This design makes it difficult to extend networks with new functionality and effectively impossible to reason precisely about their behavior.

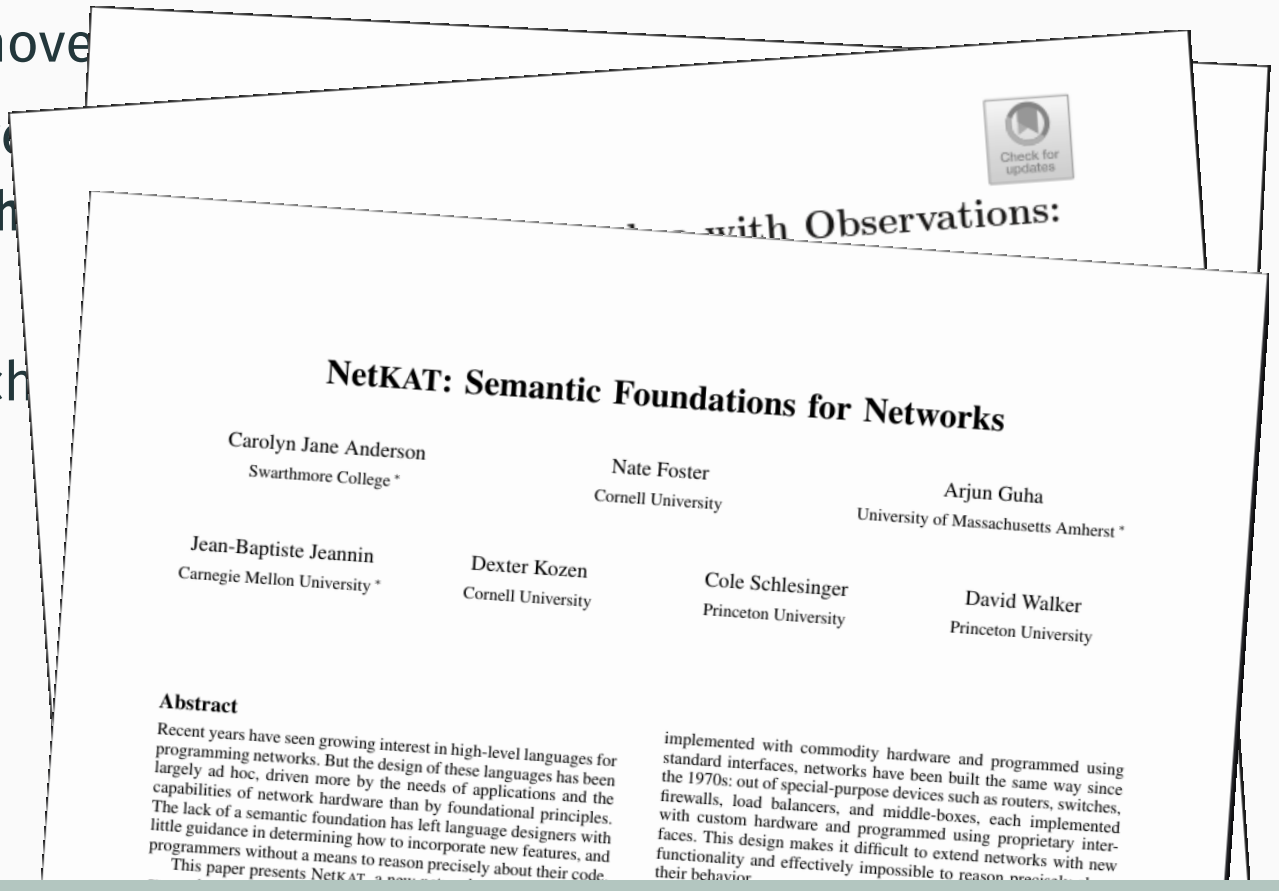
“Propositional” Equivalence

Regex ignore if operations can be...

- reordered $\rightarrow$ $ab \leq ba$
- replaced $\rightarrow$ $b \leq a$
- remove
- in g
- **in th**

“linear”

Approach



NetKAT: Semantic Foundations for Networks

Carolyn Jane Anderson, Swarthmore College *
Nate Foster, Cornell University
Arjun Guha, University of Massachusetts Amherst *
Jean-Baptiste Jeannin, Carnegie Mellon University *
Dexter Kozen, Cornell University
Cole Schlesinger, Princeton University
David Walker, Princeton University

Abstract

Recent years have seen growing interest in high-level languages for programming networks. But the design of these languages has been largely ad hoc, driven more by the needs of applications and the capabilities of network hardware than by foundational principles. The lack of a semantic foundation has left language designers with little guidance in determining how to incorporate new features, and programmers without a means to reason precisely about their code. This paper presents NetKAT, a new

implemented with commodity hardware and programmed using standard interfaces, networks have been built the same way since the 1970s: out of special-purpose devices such as routers, switches, firewalls, load balancers, and middle-boxes, each implemented with custom hardware and programmed using proprietary interfaces. This design makes it difficult to extend networks with new functionality and effectively impossible to reason precisely about their behavior.

“Propositional” Equivalence

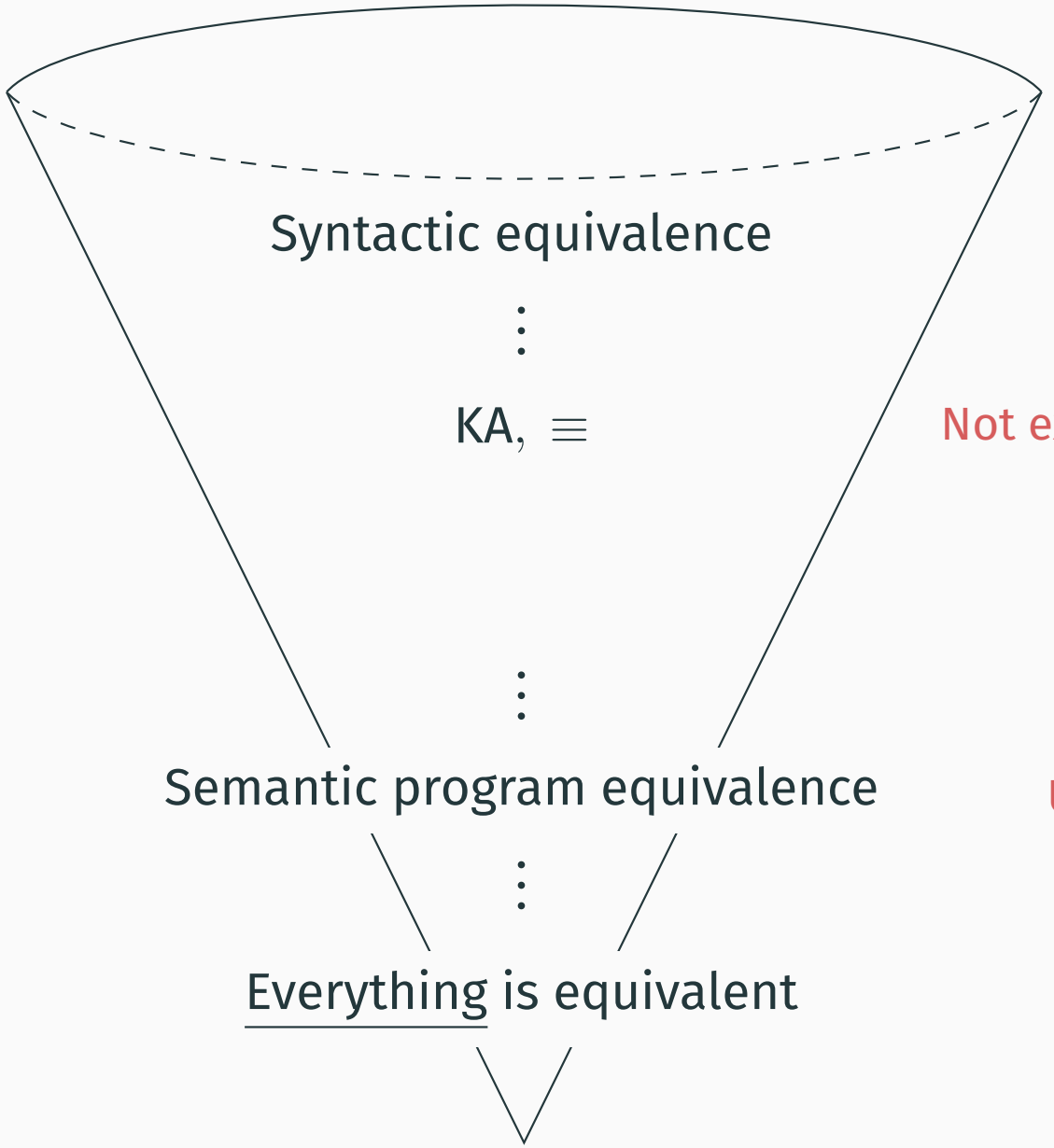
Regex ignore if operations can be...

- reordered $\rightarrow$ $ab \leq ba$
- replaced $\rightarrow$ $b \leq a$
- removed $\rightarrow$ $a \leq aa$
- in general: sets of $e \leq f$
- **in this paper:** $e \leq w$, **one at a time** “linear”

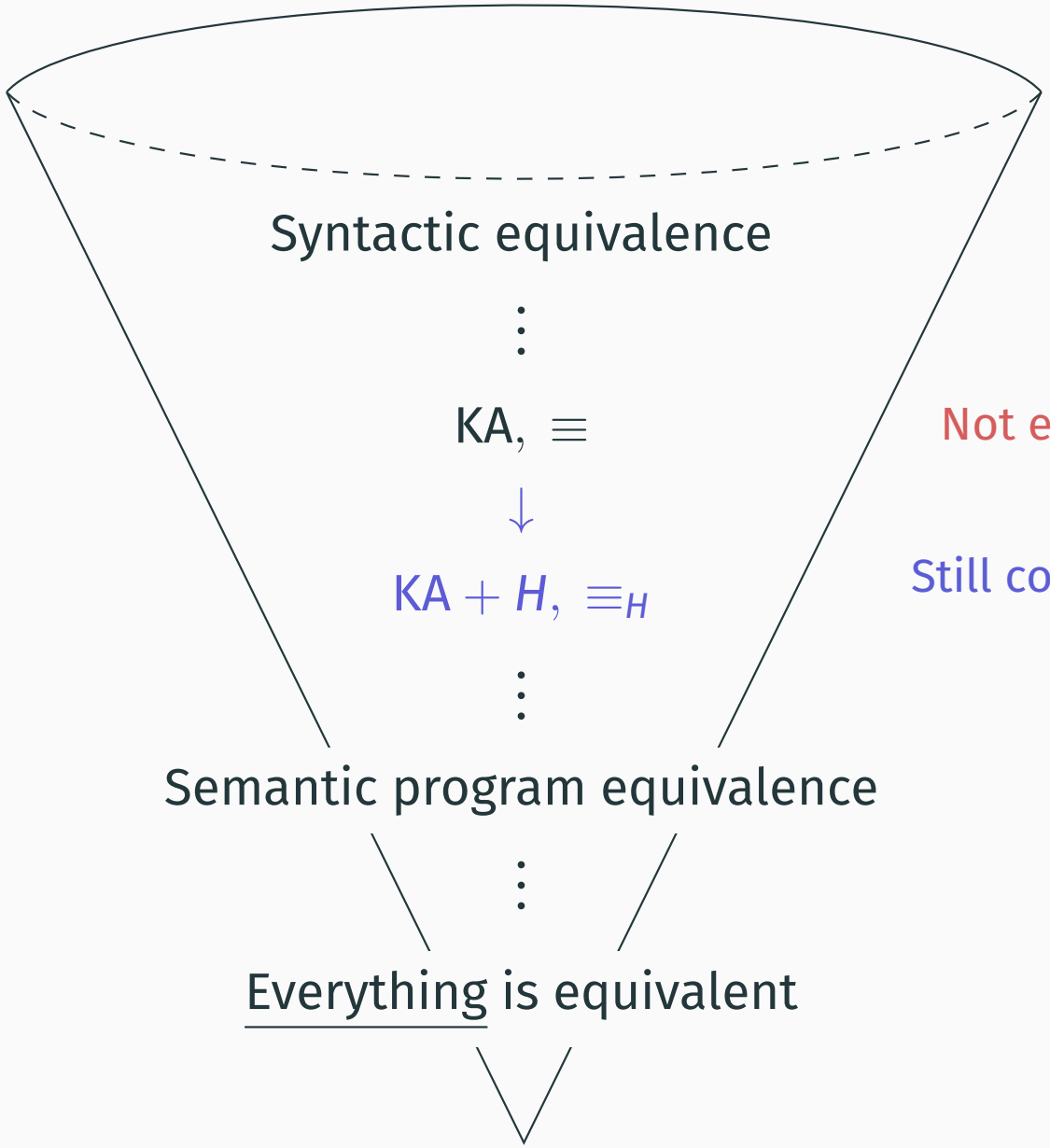
Approach 1: add “features” to KA booleans, concurrency...

Approach 2: add ~~axioms~~ “hypotheses” to KA directly $\rightarrow$ KA $+ H$

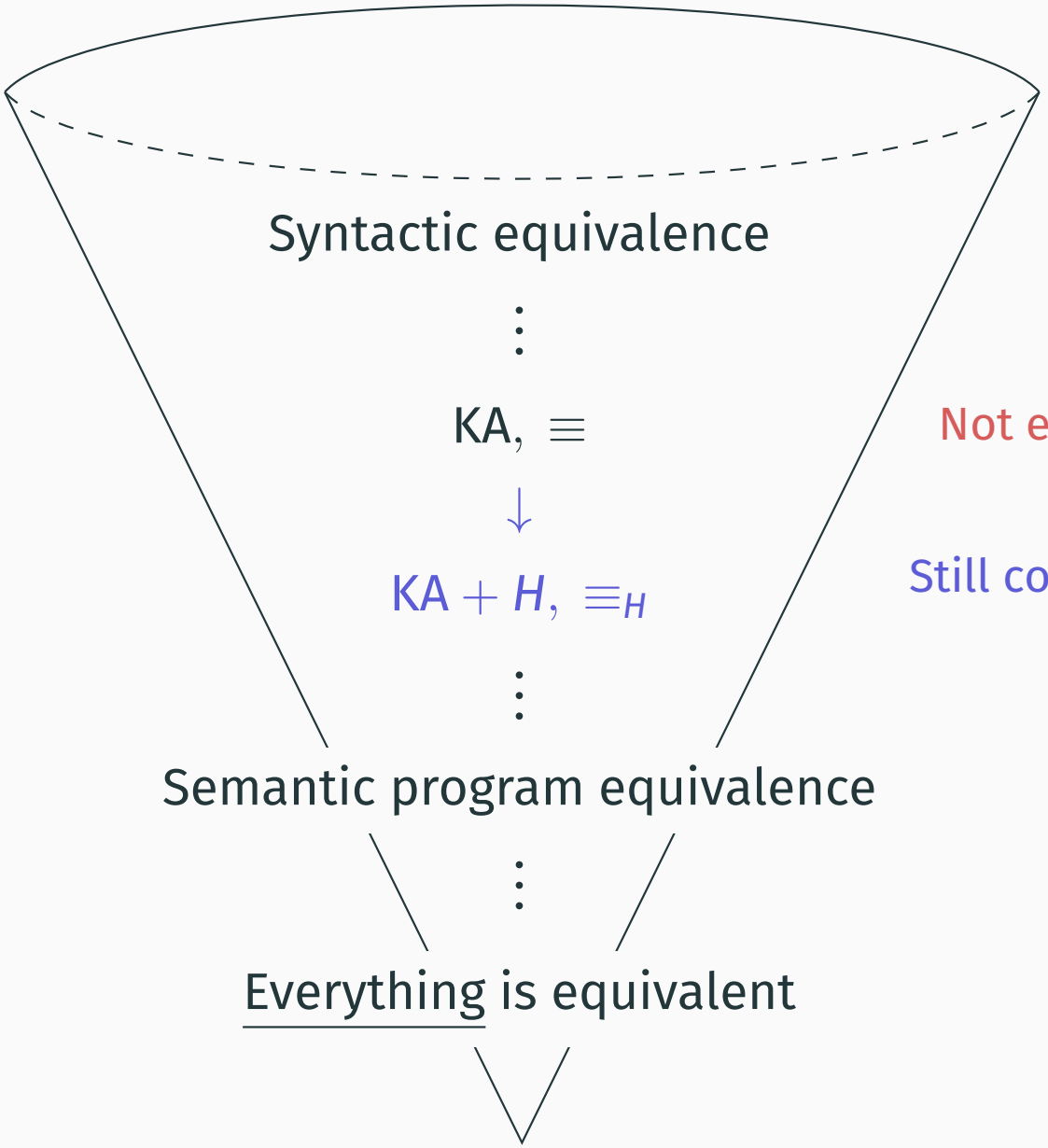
A Top-Down View on Program Equivalence Semantics



A Top-Down View on Program Equivalence Semantics



A Top-Down View on Program Equivalence Semantics



Hypothesis Closure

cf. Doumane et al. 2019

Definition

The *hypothesis closure* $H^*(L)$ for $e \leq w$ is L closed under:

$$L \mapsto L \cup \{ uw_e v : uwv \in L, w_e \in [\![e]\!] \}$$

Hypothesis Closure

cf. Doumane et al. 2019

Definition

The *hypothesis closure* $H^*(L)$ for $e \leq w$ is L closed under:

$$L \mapsto L \cup \{ uw_e v : uwv \in L, w_e \in [\![e]\!] \}$$

If $uwv \in H^*(L)$, then $\forall\, w_e \in [\![e]\!]$, $uw_e v \in H^*(L)$.

Hypothesis Closure

cf. Doumane et al. 2019

Definition

The *hypothesis closure* $H^*(L)$ for $e \leq w$ is L closed under:

$$L \mapsto L \cup \{ uw_e v : uwv \in L, w_e \in [\![e]\!] \}$$

If $uwv \in H^*(L)$, then $\forall\, w_e \in [\![e]\!]$, $uw_e v \in H^*(L)$.

For example: $H = \{\mathtt{a}\mathtt{a} \leq \mathtt{a}\}$.

Hypothesis Closure

cf. Doumane et al. 2019

Definition

The *hypothesis closure* $H^*(L)$ for $e \leq w$ is L closed under:

$$L \mapsto L \cup \{ uw_e v : uwv \in L, w_e \in [\![e]\!] \}$$

If $uwv \in H^*(L)$, then $\forall\, w_e \in [\![e]\!]$, $uw_e v \in H^*(L)$.

For example: $H = \{\mathtt{aa} \leq \mathtt{a}\}$. Apply to language $\mathtt{ba}$.

Hypothesis Closure

cf. Doumane et al. 2019

Definition

The *hypothesis closure* $H^*(L)$ for $e \leq w$ is L closed under:

$$L \mapsto L \cup \{ uw_e v : uwv \in L, w_e \in [\![e]\!] \}$$

If $uwv \in H^*(L)$, then $\forall\, w_e \in [\![e]\!]$, $uw_e v \in H^*(L)$.

For example: $H = \{\mathtt{aa} \leq \mathtt{a}\}$. Apply to language $\mathtt{ba}$.

$$\mathtt{ba} \to \mathtt{baa} \to \mathtt{baaa} \to \mathtt{baaaa} \to \cdots$$

Hypothesis Closure — cf. Doumane et al. 2019

Definition

The *hypothesis closure* $H^*(L)$ for $e \leq w$ is L closed under:

$$L \mapsto L \cup \{ uw_e v : uwv \in L, w_e \in [\![e]\!] \}$$

If $uwv \in H^*(L)$, then $\forall\, w_e \in [\![e]\!]$, $uw_e v \in H^*(L)$.

For example: $H = \{\mathtt{aa} \leq \mathtt{a}\}$. Apply to language $\mathtt{ba}$.

$$\mathtt{ba} \to \mathtt{baa} \to \mathtt{baaa} \to \mathtt{baaaa} \to \cdots$$

Semantics for KA + H, $\equiv_H$

$$[\![-]\!]_H := H^*([\![-]\!])$$

Hypothesis Closure — cf. Doumane et al. 2019

Definition

The *hypothesis closure* $H^*(L)$ for $e \leq w$ is L closed under:

$$L \mapsto L \cup \{ uw_e v : uwv \in L, w_e \in [\![e]\!] \}$$

If $uwv \in H^*(L)$, then $\forall\, w_e \in [\![e]\!]$, $uw_e v \in H^*(L)$.

For example: $H = \{\mathtt{aa} \leq \mathtt{a}\}$. Apply to language $\mathtt{ba}$.

$$\mathtt{ba} \to \mathtt{baa} \to \mathtt{baaa} \to \mathtt{baaaa} \to \cdots$$

Semantics for KA + H, $\equiv_H$

$$[\![-]\!]_H := H^*([\![-]\!])$$

Now $f \equiv_H g \;\Rightarrow\; [\![f]\!]_H = [\![g]\!]_H$. (soundness)

Game Plan

How to get completeness/decidability of KA $+$ H from KA?

Game Plan

How to get completeness/decidability of KA $+$ H from KA?

As per Pous, Rot, and Wagemaker 2024, construct a "reduction".

Game Plan

How to get completeness/decidability of KA $+ H$ from KA?

As per Pous, Rot, and Wagemaker 2024, construct a "reduction".

Theorem

Let $r : \mathbb{E} \to \mathbb{E}$ be a (computable) map. r is a reduction if $\forall\, g \in \mathbb{E}$,

$$[\![g]\!]_H = [\![r(g)]\!] \qquad g \equiv_H r(g).$$

Game Plan

How to get completeness/decidability of KA $+$ H from KA?

As per Pous, Rot, and Wagemaker 2024, construct a "reduction".

Theorem

Let $r : \mathbb{E} \to \mathbb{E}$ be a (computable) map. r is a reduction if $\forall\, g \in \mathbb{E}$,

$$[\![g]\!]_H \;=\; [\![r(g)]\!] \qquad\qquad g \;\equiv_H\; r(g).$$

Then KA $+$ H *is complete w.r.t $[\![-]\!]_H$, and $\equiv_H$ is decidable.*

Game Plan

How to get completeness/decidability of KA $+ H$ from KA?

As per Pous, Rot, and Wagemaker 2024, construct a "reduction".

Theorem

Let $r : \mathbb{E} \to \mathbb{E}$ be a (computable) map. r is a reduction if $\forall\, g \in \mathbb{E}$,

$$[\![g]\!]_H = [\![r(g)]\!] \qquad g \equiv_H r(g).$$

Then KA $+ H$ *is complete w.r.t $[\![-]\!]_H$, and $\equiv_H$ is decidable.*

Suppose $[\![f]\!]_H = [\![g]\!]_H$. WTS $f \equiv_H g$.

f g

Game Plan

How to get completeness/decidability of KA $+$ H from KA?

As per Pous, Rot, and Wagemaker 2024, construct a "reduction".

Theorem

Let $r : \mathbb{E} \to \mathbb{E}$ be a (computable) map. r is a reduction if $\forall\, g \in \mathbb{E}$,

$$[\![g]\!]_H = [\![r(g)]\!] \qquad g \equiv_H r(g).$$

Then KA $+$ H *is complete w.r.t* $[\![-]\!]_H$*, and* $\equiv_H$ *is decidable.*

Suppose $[\![f]\!]_H = [\![g]\!]_H$. WTS $f \equiv_H g$.

$$f \equiv_H r(f) \qquad r(g) \equiv_H g$$

Game Plan

How to get completeness/decidability of KA $+$ H from KA?

As per Pous, Rot, and Wagemaker 2024, construct a "reduction".

Theorem

Let $r : \mathbb{E} \to \mathbb{E}$ be a (computable) map. r is a reduction if $\forall\, g \in \mathbb{E}$,

$$[\![g]\!]_H = [\![r(g)]\!] \qquad g \equiv_H r(g).$$

Then KA $+$ H *is complete w.r.t* $[\![-]\!]_H$*, and* $\equiv_H$ *is decidable.*

Suppose $[\![f]\!]_H = [\![g]\!]_H$. WTS $f \equiv_H g$.

$$f \equiv_H r(f) \qquad r(g) \equiv_H g$$

Game Plan

How to get completeness/decidability of KA $+$ H from KA?

As per Pous, Rot, and Wagemaker 2024, construct a "reduction".

Theorem

Let $r : \mathbb{E} \to \mathbb{E}$ be a (computable) map. r is a reduction if $\forall\, g \in \mathbb{E}$,

$$[\![g]\!]_H = [\![r(g)]\!] \qquad g \equiv_H r(g).$$

Then KA $+$ H *is complete w.r.t* $[\![-]\!]_H$*, and* $\equiv_H$ *is decidable.*

Suppose $[\![f]\!]_H = [\![g]\!]_H$. WTS $f \equiv_H g$.

$$f \equiv_H r(f) \equiv r(g) \equiv_H g$$

Reduction Example

Remember this example? $H = \{\mathsf{aa} \leq \mathsf{a}\}$

$$\mathsf{ba} \to \mathsf{baa} \to \mathsf{baaa} \to \mathsf{baaaa} \to \cdots$$

Reduction Example

Remember this example? $H = \{\mathtt{aa} \leq \mathtt{a}\}$

$$\mathtt{ba} \to \mathtt{baa} \to \mathtt{baaa} \to \mathtt{baaaa} \to \cdots$$

What's an expression representing this language?

Reduction Example

Remember this example? $H = \{\mathtt{aa} \leq \mathtt{a}\}$

$$\mathtt{ba} \to \mathtt{baa} \to \mathtt{baaa} \to \mathtt{baaaa} \to \cdots$$

What's an expression representing this language? $r(\mathtt{a}) := \mathtt{baa}^*$

Reduction Example

Remember this example? $H = \{\mathtt{aa} \leq \mathtt{a}\}$

$$\mathtt{ba} \to \mathtt{baa} \to \mathtt{baaa} \to \mathtt{baaaa} \to \cdots$$

What's an expression representing this language? $r(\mathtt{a}) := \mathtt{baa}^*$

Is it a reduction?

Reduction Example

Remember this example? $H = \{\mathrm{aa} \leq \mathrm{a}\}$

$$\mathrm{ba} \to \mathrm{baa} \to \mathrm{baaa} \to \mathrm{baaaa} \to \cdots$$

What's an expression representing this language? $r(\mathrm{a}) := \mathrm{baa}^*$

Is it a reduction?

$$[\![g]\!]_H = [\![r(g)]\!] \qquad g \equiv_H r(g)$$

Reduction Example

Remember this example? $H = \{\mathsf{aa} \leq \mathsf{a}\}$

$$\mathsf{ba} \to \mathsf{baa} \to \mathsf{baaa} \to \mathsf{baaaa} \to \cdots$$

What's an expression representing this language? $r(\mathsf{a}) := \mathsf{baa}^*$

Is it a reduction?

$$[\![\mathsf{ba}]\!]_H = [\![\mathsf{baa}^*]\!] \qquad \mathsf{ba} \equiv_H \mathsf{baa}^*$$

Reduction Example

Remember this example? $H = \{\mathtt{aa} \leq \mathtt{a}\}$

$$\mathtt{ba} \to \mathtt{baa} \to \mathtt{baaa} \to \mathtt{baaaa} \to \cdots$$

What's an expression representing this language? $r(\mathtt{a}) := \mathtt{baa}^*$

Is it a reduction?

$$[\![\mathtt{ba}]\!]_H = [\![\mathtt{baa}^*]\!] \qquad \mathtt{ba} \equiv_H \mathtt{baa}^*$$

Is $[\![\mathtt{baa}^*]\!]$ actually the hyp. closed language above?

Reduction Example

Remember this example? $H = \{\mathtt{aa} \leq \mathtt{a}\}$

$$\mathtt{ba} \to \mathtt{baa} \to \mathtt{baaa} \to \mathtt{baaaa} \to \cdots$$

What's an expression representing this language? $r(\mathtt{a}) := \mathtt{baa}^*$

Is it a reduction?

$$[\![\mathtt{ba}]\!]_H = [\![\mathtt{baa}^*]\!] \qquad \mathtt{ba} \equiv_H \mathtt{baa}^*$$

Is $[\![\mathtt{baa}^*]\!]$ actually the hyp. closed language above?

✅

Reduction Example

Remember this example? $H = \{\mathsf{aa} \leq \mathsf{a}\}$

$$\mathsf{ba} \to \mathsf{baa} \to \mathsf{baaa} \to \mathsf{baaaa} \to \cdots$$

What's an expression representing this language? $r(\mathsf{a}) := \mathsf{baa}^*$

Is it a reduction?

$$[\![\mathsf{ba}]\!]_H = [\![\mathsf{baa}^*]\!] \qquad \mathsf{ba} \equiv_H \mathsf{baa}^*$$

Is $[\![\mathsf{baa}^*]\!]$ actually the hyp. closed language above?

✅

Can we prove ba and baa^* equivalent with $\mathsf{KA} + \{\mathsf{aa} \leq \mathsf{a}\}$?

Reduction Example

Remember this example? $H = \{\mathsf{aa} \leq \mathsf{a}\}$

$$\mathsf{ba} \to \mathsf{baa} \to \mathsf{baaa} \to \mathsf{baaaa} \to \cdots$$

What's an expression representing this language? $r(\mathsf{a}) := \mathsf{baa}^*$

Is it a reduction?

$$[\![\mathsf{ba}]\!]_H = [\![\mathsf{baa}^*]\!] \qquad \mathsf{ba} \equiv_H \mathsf{baa}^*$$

Is $[\![\mathsf{baa}^*]\!]$ actually the hyp. closed language above?

✅

Can we prove ba and baa^* equivalent with $\mathsf{KA} + \{\mathsf{aa} \leq \mathsf{a}\}$?

✅

Reduction Strategy

REDUCTION TODO LIST:

- New semantics
- Define map $\mathbb{E} \to \mathbb{E}$
- Prove it is a reduction

Reduction Strategy

REDUCTION TODO LIST:

- New semantics ("Hypothesis closure")
- Define map $\mathbb{E} \to \mathbb{E}$
- Prove it is a reduction

Reduction Strategy

REDUCTION TODO LIST:

- New semantics (“Hypothesis closure”)
- Define map $\mathbb{E} \to \mathbb{E}$???
- Prove it is a reduction ???

Reduction Strategy

REDUCTION TODO LIST:

- New semantics (“Hypothesis closure”)
- Define map $\mathbb{E} \to \mathbb{E}$???
- Prove it is a reduction ???

Our contribution: a procedure for defining generic reductions, and a proof that when the procedure terminates (📌), it is correct.

Reduced Expressions via Automata

g

Reduced Expressions via Automata

g $r(g)$

Reduced Expressions via Automata

$g \to A_g$ $r(g)$

Reduced Expressions via Automata

$$g \to A_g \to T_H(A_g) \to r(g)$$

Reduced Expressions via Automata

$$g \to A_g \to T_H(A_g) \to r(g)$$

Use a construction on automata to define a reduction.

Reduced Expressions via Automata

$$g \to A_g \to T_H(A_g) \to r(g)$$

Use a construction on automata to define a reduction.

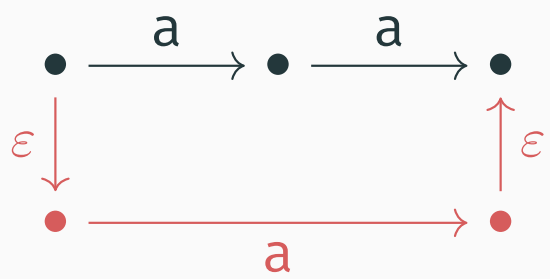
$$\mathrm{a} \leq \mathrm{aa}$$

Reduced Expressions via Automata

$$g \to A_g \to T_H(A_g) \to r(g)$$

Use a construction on automata to define a reduction.

$$a \leq aa$$

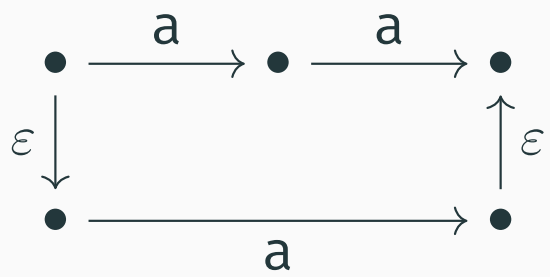


Reduced Expressions via Automata

$$g \to A_g \to T_H(A_g) \to r(g)$$

Use a construction on automata to define a reduction.

$a \leq aa$



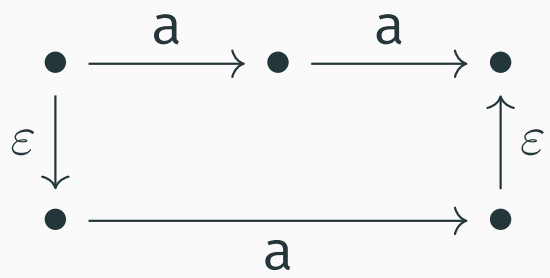
$e \leq w$

Reduced Expressions via Automata

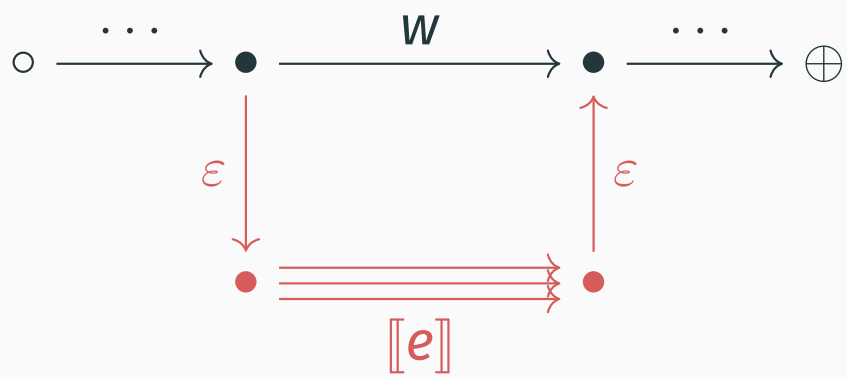
$$g \to A_g \to T_H(A_g) \to r(g)$$

Use a construction on automata to define a reduction.

$\mathrm{a} \leq \mathrm{aa}$



$e \leq w$

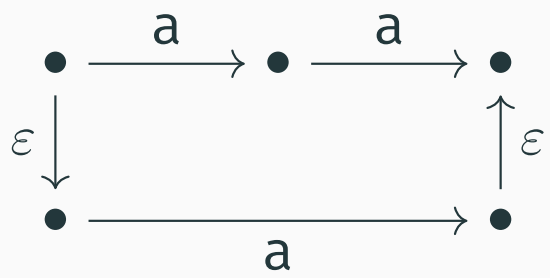


Reduced Expressions via Automata

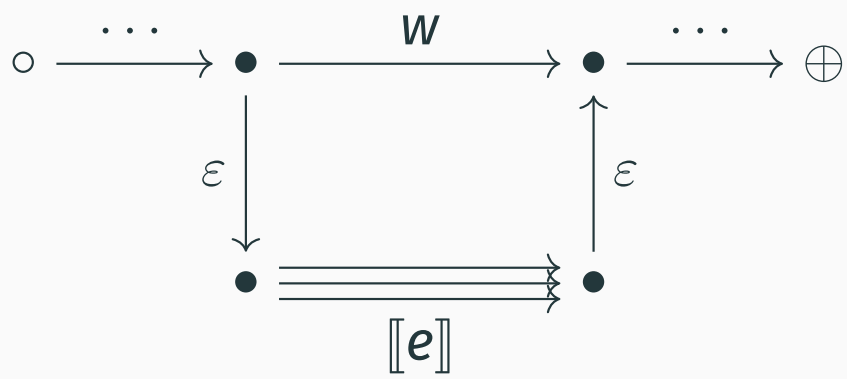
$$g \to A_g \to T_H(A_g) \to r(g)$$

Use an iterated construction on automata to define a reduction.

$\mathrm{a} \leq \mathrm{aa}$



$e \leq w$

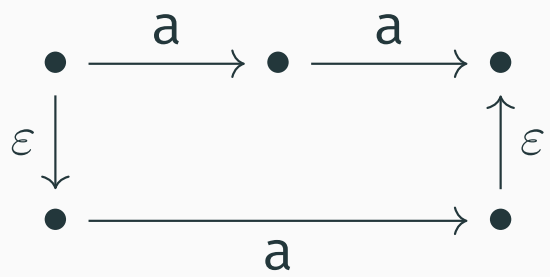


Reduced Expressions via Automata

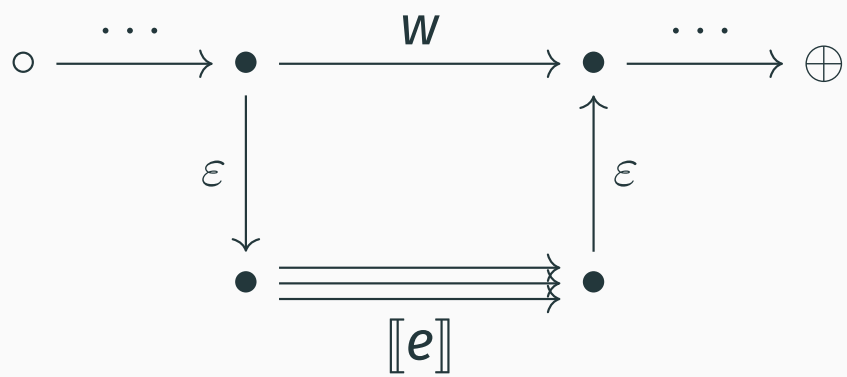
$$g \to A_g \to T_H(A_g) \to r(g)$$

Use an iterated construction on automata to define a reduction.

$\mathrm{a} \leq \mathrm{aa}$



$e \leq w$



Various versions of this are well-known as strategies for “Approach 1”: see Book and Otto 1993; Kozen and Mamouras 2014; Pous, Rot, and Wagemaker 2024 for more.

Problem: Automaton Constructions Need to be Finite

What if $T_H(A_g)$ isn't finite...?

Problem: Automaton Constructions Need to be Finite

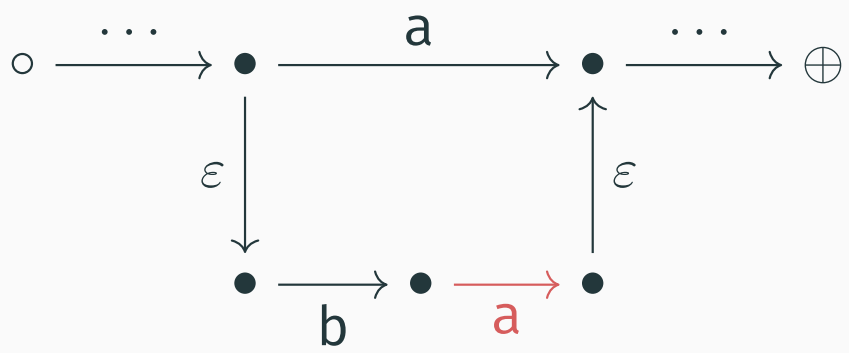
What if $T_H(A_g)$ isn't finite...?

Consider the hypothesis $ba \leq a$.

Problem: Automaton Constructions Need to be Finite

What if $T_H(A_g)$ isn't finite...?

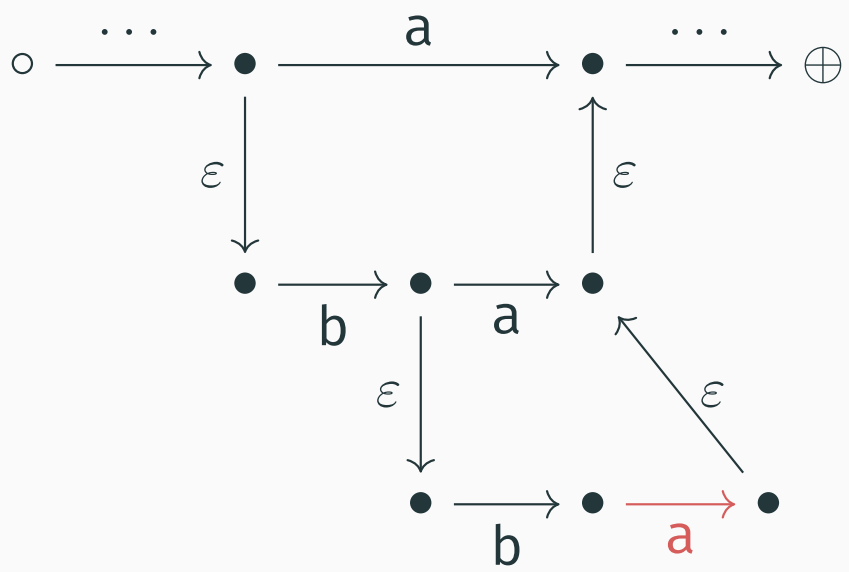
Consider the hypothesis $\mathsf{ba} \leq \mathsf{a}$.



Problem: Automaton Constructions Need to be Finite

What if $T_H(A_g)$ isn't finite...?

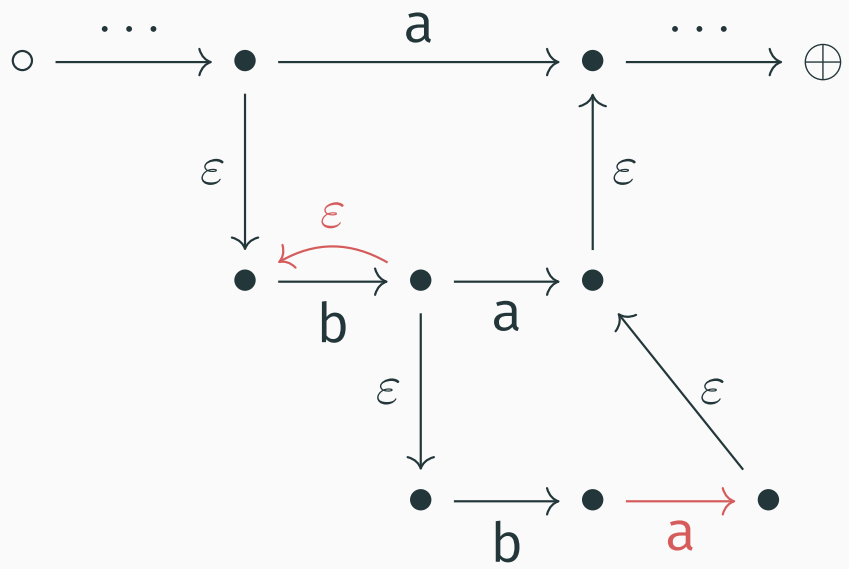
Consider the hypothesis ba $\leq$ a.



Problem: Automaton Constructions Need to be Finite

What if $T_H(A_g)$ isn't finite...?

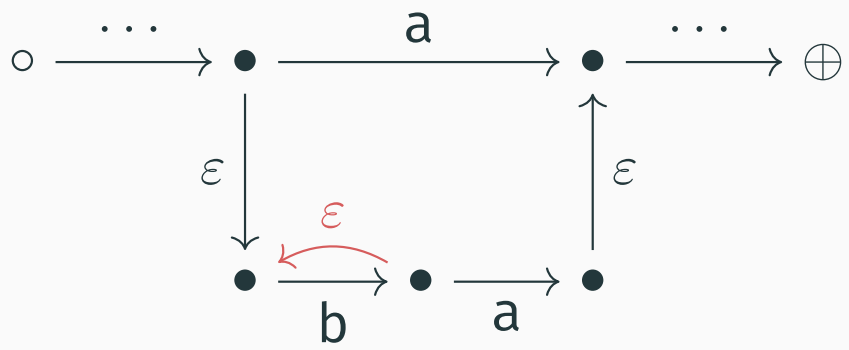
Consider the hypothesis ba $\leq$ a.



Problem: Automaton Constructions Need to be Finite

What if $T_H(A_g)$ isn't finite...?

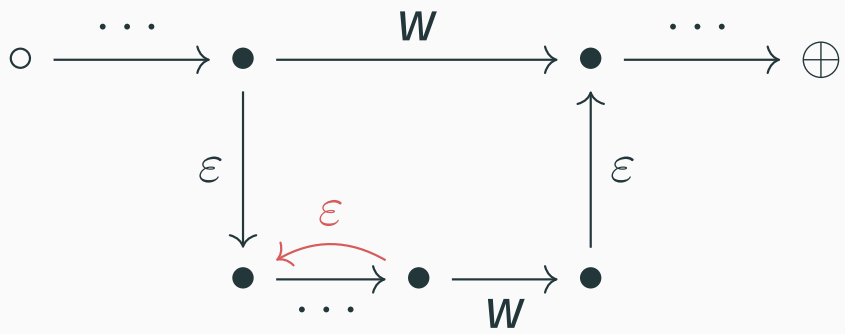
Consider the hypothesis ba $\leq$ a.



Problem: Automaton Constructions Need to be Finite

What if $T_H(A_g)$ isn't finite...?

Consider the hypothesis $e \leq w$.

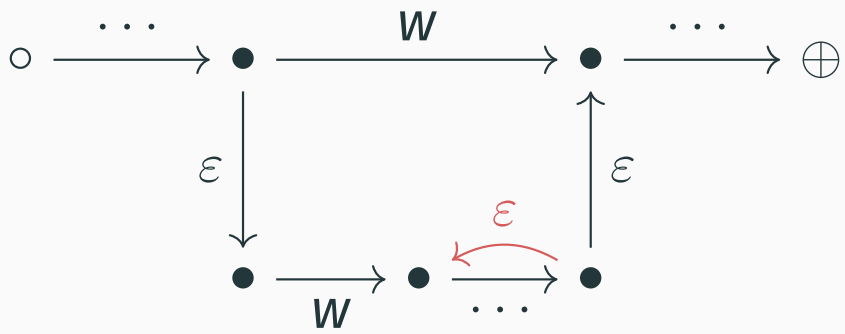


Will happen for any hypothesis where w is a suffix of a word in $[\![e]\!]$.

Problem: Automaton Constructions Need to be Finite

What if $T_H(A_g)$ isn't finite...?

Consider the hypothesis $e \leq w$.



Will happen for any hypothesis where w is a suffix of a word in $[\![e]\!]$.

Symmetric if w is a prefix as well.

Problem: Reductions Don't Always Exist

Theorem?

Define $r_H : \mathbb{E} \to \mathbb{E}$ using the output of T_H.

Then r_H is a reduction.

Therefore, KA $+ H$ is complete and decidable.

Problem: Reductions Don't Always Exist

Theorem?

Define $r_H : \mathbb{E} \to \mathbb{E}$ using the output of T_H.

Then r_H is a reduction.

Therefore, KA $+ H$ is complete and decidable.

Let $H = \{\mathsf{aba} \leq \mathsf{b}\}$.

Problem: Reductions Don't Always Exist

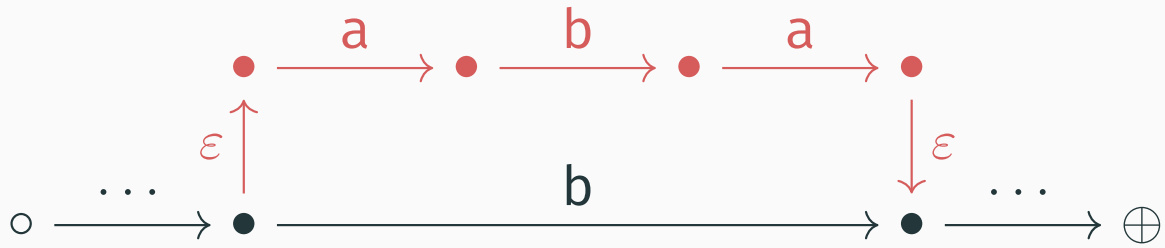
Theorem?

Define $r_H : \mathbb{E} \to \mathbb{E}$ using the output of T_H.

Then r_H is a reduction.

Therefore, KA $+ H$ is complete and decidable.

Let $H = \{\mathsf{aba} \leq \mathsf{b}\}$.



Problem: Reductions Don't Always Exist

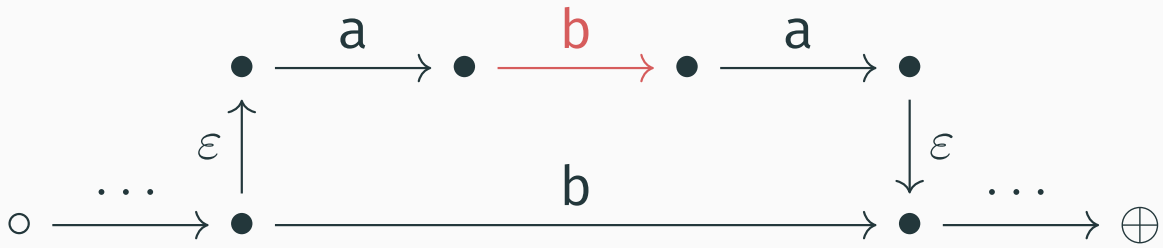
Theorem?

Define $r_H : \mathbb{E} \to \mathbb{E}$ using the output of T_H.

Then r_H is a reduction.

Therefore, KA $+ H$ is complete and decidable.

Let $H = \{\mathsf{aba} \leq \mathsf{b}\}$.



Problem: Reductions Don't Always Exist

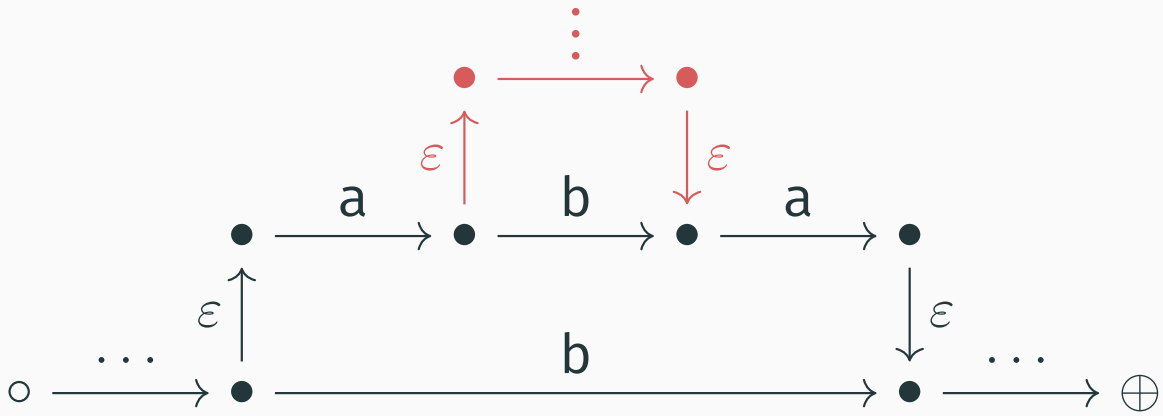
Theorem?

Define $r_H : \mathbb{E} \to \mathbb{E}$ using the output of T_H.

Then r_H is a reduction.

Therefore, KA $+ H$ is complete and decidable.

Let $H = \{\mathsf{aba} \leq \mathsf{b}\}$.



Problem: Reductions Don't Always Exist

Theorem?

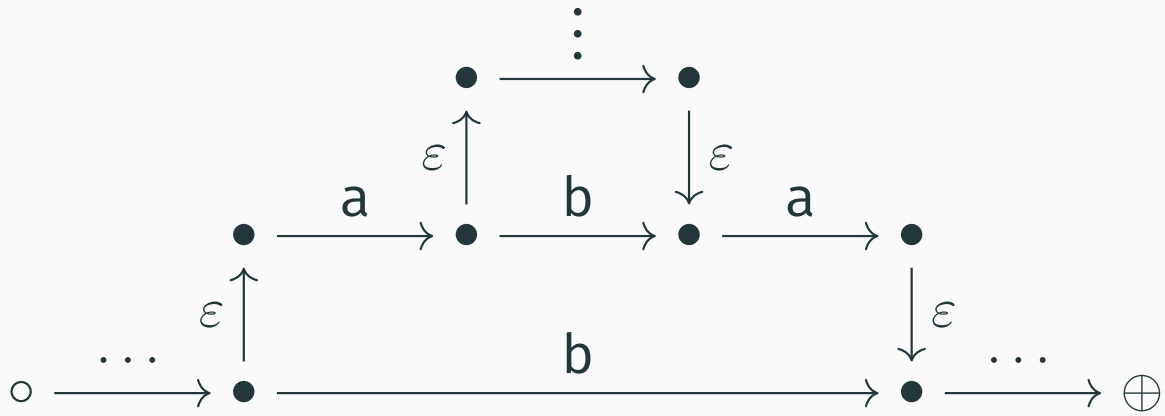
Define $r_H : \mathbb{E} \to \mathbb{E}$ using the output of T_H.

Then r_H is a reduction.

Therefore, KA $+ H$ is complete and decidable.

Let $H = \{\mathsf{aba} \leq \mathsf{b}\}$.

is this... $\mathsf{a}^n\mathsf{ba}^n$?



Problem: Reductions Can't Always Exist

Theorem? IMPOSSIBLE

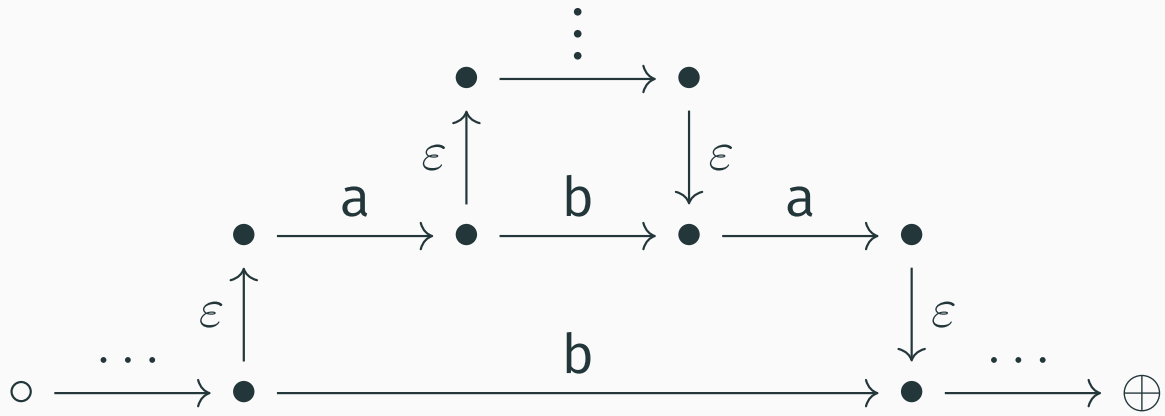
Define $r_H : \mathbb{E} \to \mathbb{E}$ using the output of T_H.

Then r_H is a reduction.

Therefore, KA $+ H$ is complete and decidable.

Let $H = \{\mathsf{aba} \leq \mathsf{b}\}$.

is this... $\mathsf{a}^n\mathsf{ba}^n$?



Problem: Reductions Can't Always Exist

Theorem? IMPOSSIBLE

Define $r_H : \mathbb{E} \to \mathbb{E}$ using the output of T_H.

Then r_H is a reduction.

Therefore, KA $+ H$ is complete and decidable.

Key insight: a reduction gives completeness.

Problem: Reductions Can't Always Exist

Theorem? IMPOSSIBLE

Define $r_H : \mathbb{E} \to \mathbb{E}$ using the output of T_H.

Then r_H is a reduction.

Therefore, KA $+ H$ is complete and decidable.

Key insight: a reduction gives completeness.

$$f \equiv_H r(f) \equiv r(g) \equiv_H g$$

Problem: Reductions Can't Always Exist

Theorem? IMPOSSIBLE

Define $r_H : \mathbb{E} \to \mathbb{E}$ using the output of T_H.

Then r_H is a reduction.

Therefore, KA $+ H$ is complete and decidable.

Key insight: a reduction gives a proof. ~~completeness~~

$$f \equiv_H r(f) \equiv r(g) \equiv_H g$$

We only need $r(f)$ and $r(g)$ to decide $f \stackrel{?}{\equiv}_H g$!

Problem: Reductions Can't Always Exist

Theorem? IMPOSSIBLE

Define $r_H : \mathbb{E} \to \mathbb{E}$ using the output of T_H.

Then r_H is a reduction.

Therefore, KA $+ H$ is complete and decidable.

Key insight: a reduction gives a proof. ~~completeness~~

$$f \equiv_H r(f) \equiv r(g) \equiv_H g$$

We only need $r(f)$ and $r(g)$ to decide $f \stackrel{?}{\equiv}_H g$!

Our new motto: take what you can get. Partial reductions!

Problem: Reductions Can't Always Exist

Theorem? IMPOSSIBLE

Define $r_H : \mathbb{E} \to \mathbb{E}$ using the output of T_H.

Then r_H is a reduction.

Therefore, KA $+ H$ is complete and decidable.

Key insight: a reduction gives a proof. ~~completeness~~

$$f \equiv_H r(f) \equiv r(g) \equiv_H g$$

We only need $r(f)$ and $r(g)$ to decide $f \stackrel{?}{\equiv}_H g$!

Our new motto: take what you can get. Partial reductions!

Problem: Reductions Can't Always Exist

Theorem (REAL)

Define $r_H : \mathbb{E} \rightharpoonup \mathbb{E}$ using the output of T_H *when it terminates* (📌).

Then r_H is a *partial* reduction.

Therefore, KA $+ H$ is complete and decidable *over r_H's domain*.

Let's fix up our todo list.

Problem: Reductions Can't Always Exist

Theorem (REAL)

Define $r_H : \mathbb{E} \rightharpoonup \mathbb{E}$ using the output of T_H *when it terminates* (📌).

Then r_H is a *partial* reduction.

Therefore, KA $+ H$ is complete and decidable *over r_H's domain*.

Let's fix up our todo list.

PARTIAL REDUCTION TODO LIST:

- New semantics
- Define partial map $\mathbb{E} \rightharpoonup \mathbb{E}$
- Prove it is a reduction (when defined)

Problem: Reductions Can't Always Exist

Theorem (REAL)

Define $r_H : \mathbb{E} \rightharpoonup \mathbb{E}$ using the output of T_H *when it terminates* (📌).

Then r_H is a *partial* reduction.

Therefore, KA $+ H$ is complete and decidable *over r_H's domain*.

Let's fix up our todo list.

PARTIAL REDUCTION TODO LIST:

- ~~New semantics~~ ✅ (Hypothesis closure)
- ~~Define partial map $\mathbb{E} \rightharpoonup \mathbb{E}$~~ ✅ 📌 (Automaton construction)
- Prove it is a reduction (when defined) ???

Generic Reduction Proof

Let $g \in \mathbb{E}$ such that $r(g)$ is defined.

Generic Reduction Proof

Let $g \in \mathbb{E}$ such that $r(g)$ is defined.

Want to show:

$$[\![g]\!]_H = [\![r(g)]\!] \qquad\qquad g \equiv_H r(g)$$

Generic Reduction Proof

Let $g \in \mathbb{E}$ such that $r(g)$ is defined.

Want to show:

$$[\![g]\!]_H = [\![r(g)]\!] \qquad\qquad g \equiv_H r(g)$$

Equivalent to...

$$H^*([\![g]\!]) = L_{T_H(A_g)}$$

Generic Reduction Proof

Let $g \in \mathbb{E}$ such that $r(g)$ is defined.

Want to show:

$$[\![g]\!]_H = [\![r(g)]\!] \qquad g \equiv_H r(g)$$

Equivalent to...

$$H^*([\![g]\!]) = L_{T_H(A_g)}$$

Verify constructed automaton

Generic Reduction Proof

Let $g \in \mathbb{E}$ such that $r(g)$ is defined.

Want to show:

$$[\![g]\!]_H = [\![r(g)]\!] \qquad\qquad g \equiv_H r(g)$$

Equivalent to...

$$H^*([\![g]\!]) = L_{T_H(A_g)}$$

Verify constructed automaton

✅ **“easy”**

Generic Reduction Proof

Let $g \in \mathbb{E}$ such that $r(g)$ is defined.

Want to show:

$$[\![g]\!]_H = [\![r(g)]\!] \qquad g \equiv_H r(g)$$

Equivalent to...

$$H^*([\![g]\!]) = L_{T_H(A_g)}$$

Verify constructed automaton

✅ **“easy”**

The hard part.

Provability on automata...?

Generic Reduction Proof

Let $g \in \mathbb{E}$ such that $r(g)$ is defined.

Want to show:

$$[\![g]\!]_H = [\![r(g)]\!] \qquad g \equiv_H r(g)$$

Equivalent to...

$$H^*([\![g]\!]) = L_{T_H(A_g)}$$

Verify constructed automaton

✅ **“easy”**

The hard part.
Provability on automata...?

Invariance (up to H) at each step of automaton “solutions”

✅

Next Steps

Possible applications:

- Combine reductions (Pous, Rot, and Wagemaker 2024)
- Characterise hypothesis/expression subclasses
- Augment KA solver in Rocq (Braibant and Pous 2012)

Next Steps

Possible applications:

- Combine reductions (Pous, Rot, and Wagemaker 2024)
- Characterise hypothesis/expression subclasses
- Augment KA solver in Rocq (Braibant and Pous 2012)

Other future work:

- Does T_H terminate when it can?
- General hypotheses $e \leq f$
- Sets of hypotheses

Thank you!

This paper on arxiv:

These slides on my website:

References

- Book, Ronald V. and Friedrich Otto (1993). ***String-Rewriting Systems.*** Springer.
- Braibant, Thomas and Damien Pous (2012). **"Deciding Kleene Algebras in Coq".** In: *Log. Methods Comput. Sci.* 8.1. DOI: 10.2168/LMCS-8(1:16)2012.
- Doumane, Amina et al. (2019). **"Kleene Algebra with Hypotheses".** In: *FoSSaCS*. DOI: 10.1007/978-3-030-17127-8˙12.
- Kozen, Dexter (1994). **"A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events".** In: *Inf. Comput.* 110.2, pp. 366–390.
- Kozen, Dexter and Konstantinos Mamouras (2014). **"Kleene Algebra with Equations".** In: *ICALP*. DOI: 10.1007/978-3-662-43951-7˙24.
- Pous, Damien, Jurriaan Rot, and Jana Wagemaker (2024). **"On Tools for Completeness of Kleene Algebra with Hypotheses".** In: *Log. Methods Comput. Sci.* 20.2. DOI: 10.46298/LMCS-20(2:8)2024.