Guarded Kleene Algebra with Tests
Verification of Uninterpreted Programs in Nearly Linear Time

Tobias Kappé
Cornell University

Programming Languages Discussion Group
Published @ POPL 2020; see arxiv.org/abs/1907.05920.
while a and b do
  e;
end
while a do
  f;
  while a and b do
    e;
  end
end
while $a$ and $b$ do
  $e$;
end
while $a$ do
  $f$;
  while $a$ and $b$ do
    $e$;
  end
end

while $a$ do
  if $b$ then
    $e$;
  else
    $f$;
  end
end
while a and b do
  e;
end
while a do
  f;
  while a and b do
    e;
  end
end

while a do
  if b then
    e;
  else
    f;
  end
end

≡
Contributions:

- Nearly linear time decision procedure for equivalence.\(^1\)

\(^1\)For fixed number of tests.
Contributions:

- Nearly linear time decision procedure for equivalence.\(^1\)
- Axiomatization of uninterpreted program equivalence.

\(^1\)For fixed number of tests.
Contributions:

- Nearly linear time decision procedure for equivalence.\(^1\)
- Axiomatization of uninterpreted program equivalence.
- Kleene theorem for uninterpreted programs.

\(^1\)For fixed number of tests.
Syntax

\[ a, b ::= t \in T \mid a + b \mid ab \mid \overline{a} \mid 0 \mid 1 \]

\[ e, f ::= a \mid p \in \Sigma \mid ef \mid e + a f \mid e^{(a)} \]
Syntax

\[ a, b ::= t \in T \mid a + b \mid ab \mid \overline{a} \mid 0 \mid 1 \]

\[ e, f ::= a \mid p \in \Sigma \mid ef \mid e + a f \mid e(a) \]
Syntax

\[ a, b ::= t \in T \mid a + b \mid ab \mid \overline{a} \mid 0 \mid 1 \]

\[ e, f ::= a \mid p \in \Sigma \mid ef \mid e +_a f \mid e^{(a)} \]
Syntax

\[ a, b ::= t \in T \mid a + b \mid ab \mid \overline{a} \mid 0 \mid 1 \]

\[ e, f ::= a \mid p \in \Sigma \mid ef \mid e + a f \mid e^{(a)} \]
Syntax

\[ \begin{align*}
a, b & ::= t \in T \mid a + b \mid ab \mid \overline{a} \mid 0 \mid 1 \\
e, f & ::= a \mid p \in \Sigma \mid ef \mid e + a f \mid e^{(a)}
\end{align*} \]
a, b ::= t ∈ T | a + b | ab | a | 0 | 1

e, f ::= a | p ∈ Σ | ef | e + a f | e^{(a)}
Syntax

\[ a, b ::= t \in T \mid a + b \mid ab \mid \overline{a} \mid 0 \mid 1 \]

\[ e, f ::= a \mid p \in \Sigma \mid ef \mid e +_a f \mid e^{(a)} \]

assert \ a
Syntax

\[ a, b ::= t \in T \mid a + b \mid ab \mid \overline{a} \mid 0 \mid 1 \]

\[ e, f ::= a \mid p \in \Sigma \mid ef \mid e + a f \mid e^{(a)} \]

\[ e; f \]
Syntax

\[
a, b ::= t \in T \mid a + b \mid ab \mid \overline{a} \mid 0 \mid 1
\]

\[
e, f ::= a \mid p \in \Sigma \mid ef \mid e +_a f \mid e^a
\]

\begin{itemize}
  \item if \(a\) then \(e\) else \(f\)
\end{itemize}
Syntax

\[ a, b ::= t \in T \mid a + b \mid ab \mid \overline{a} \mid 0 \mid 1 \]

\[ e, f ::= a \mid p \in \Sigma \mid ef \mid e + a f \mid e^a \]

\[ \text{while } a \text{ do } e \]
Syntax

\[
\text{while } a \text{ do } \frac{\text{e}}{\text{if } b \text{ then } e; \text{ else } f; \text{ end}}{\text{end}} \quad \downarrow \quad (e + b f)^{(a)}
\]

\[
\text{while } a \text{ and } b \text{ do } \frac{\text{e; end}}{\text{while } a \text{ do } f; \text{ while } a \text{ and } b \text{ do } e; \text{ end} \text{ end}} \quad \downarrow \quad e^{(ab)}(fe^{(ab)})^{(a)}
\]
\( \text{sat} : T \rightarrow 2^{\text{States}} \)
Semantics / relational

\[ sat : T \rightarrow 2^{States} \]

\[ eval : \Sigma \rightarrow States \rightarrow 2^{States} \]
Semantics / relational

\[ sat : T \rightarrow 2^{States} \]

\[ eval : \Sigma \rightarrow States \rightarrow 2^{States} \]

\[ i = (sat, eval) \]
Semantics / relational

\[ \text{sat} : T \rightarrow 2^{\text{States}} \]

\[ \text{eval} : \Sigma \rightarrow \text{States} \rightarrow 2^{\text{States}} \]

\[ i = (\text{sat}, \text{eval}) \]

\[ R_i[e] : \text{States} \rightarrow 2^{\text{States}} \]
$$sat : T \rightarrow 2^{States}$$
Semantics / probabilistic

\[
sat : T \rightarrow 2^{States} \\
 eval : \Sigma \rightarrow States \rightarrow D(States)
\]
Semantics / probabilistic

\[ sat : T \rightarrow 2^{States} \]

\[ eval : \Sigma \rightarrow States \rightarrow \mathcal{D}(States) \]

\[ i = (sat, eval) \]
Semantics/probabilistic

\( sat : T \rightarrow 2^{\text{States}} \)

\( eval : \Sigma \rightarrow \text{States} \rightarrow \mathcal{D}(\text{States}) \)

\( i = (sat, eval) \)

\( \mathcal{P}_i[e] : \text{States} \rightarrow \mathcal{D}(\text{States}) \)
Atoms = 2^T

GS(\Sigma, T) = Atoms \cdot (\Sigma \cdot Atoms)^*$
Semantics/uninterpreted

\[
\text{Atoms} = 2^T \quad \text{GS}(\Sigma, T) = \text{Atoms} \cdot (\Sigma \cdot \text{Atoms})^* \\

L \odot K = \{ w\alpha x : w\alpha \in L, \alpha x \in K \} \\
L^{(n)} = L \odot \cdots \odot L \\
L^{(\ast)} = \bigcup_{n \in \mathbb{N}} L^{(n)}
\]
### Semantics / uninterpreted

<table>
<thead>
<tr>
<th>Expression</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$e$</td>
<td>$[e]$</td>
</tr>
<tr>
<td>$t \in T$</td>
<td>${ \alpha \in Atoms : t \in \alpha }$</td>
</tr>
<tr>
<td>$a + b$</td>
<td>$[a] \cup [b]$</td>
</tr>
<tr>
<td>$ab$</td>
<td>$[a] \cap [b]$</td>
</tr>
<tr>
<td>$\overline{a}$</td>
<td>$Atoms \setminus [a]$</td>
</tr>
<tr>
<td>$p \in \Sigma$</td>
<td>${ \alpha p\beta : \alpha, \beta \in Atoms }$</td>
</tr>
<tr>
<td>$e +_a f$</td>
<td>$[a] \diamond [e] \cup [\overline{a}] \diamond [f]$</td>
</tr>
<tr>
<td>$ef$</td>
<td>$[e] \diamond [f]$</td>
</tr>
<tr>
<td>$e^{(a)}$</td>
<td>$( [a] \diamond [e] )^{(*)} \diamond [a]$</td>
</tr>
</tbody>
</table>
Theorem

The following are equivalent:

\[ [e] = [f] \quad \forall i. \mathcal{R}_i[e] = \mathcal{R}_i[f] \quad \forall i. \mathcal{P}_i[e] = \mathcal{P}_i[f] \]
The following are equivalent:

\[ \{ e \} = \{ f \} \quad \forall i. R_i \{ e \} = R_i \{ f \} \quad \forall i. P_i \{ e \} = P_i \{ f \} \]

How to check \( \{ e \} = \{ f \} \):

1. Create automata that accept \( \{ e \} \) and \( \{ f \} \).
2. Check automata for bisimilarity.
Axiomatization / without loops

e +_{a} e \equiv e
Axiomatization / without loops

\[ e +_a e \equiv e \quad e +_a f \equiv f +_e e \]
Axiomatization / without loops

\[
e + a\ e \equiv e \\
e + a\ f \equiv f + \overline{a}\ e \\
e + a\ f \equiv ae + a\ f
\]
Axiomatization / without loops

\[
e +_a e \equiv e \quad e +_a f \equiv f +_{\overline{a}} e \quad e +_a f \equiv ae +_a f \quad \overline{aa} \equiv 0
\]
Axiomatization / without loops

\[ e +_a e \equiv e \quad e +_a f \equiv f +_{\bar{a}} e \quad e +_a f \equiv a e +_a f \quad \bar{a}a \equiv 0 \quad 0e \equiv 0 \]
Axiomatization / without loops

\[ e +_a e \equiv e \quad e +_a f \equiv f +_a e \quad e +_a f \equiv ae +_a f \quad \overline{aa} \equiv 0 \quad 0e \equiv 0 \]

Example

\[
\text{if } a \text{ then } e \text{ else assert false } = e +_a 0
\]
Axiomatization / without loops

\[
\begin{align*}
  e + a e & \equiv e \\
  e + a f & \equiv f + a e & e + a f & \equiv a e + a f \\
  \overline{a} a & \equiv 0 \\
  0 e & \equiv 0
\end{align*}
\]

**Example**

\[
\text{if } a \text{ then } e \text{ else assert false } = e + a 0 \equiv a e + a 0
\]
Axiomatization / without loops

\[ e + a \cdot e \equiv e \]
\[ e + a \cdot f \equiv f + a \cdot e \]
\[ e + a \cdot f \equiv a \cdot e + a \cdot f \]
\[ \overline{a} a \equiv 0 \]
\[ 0 \cdot e \equiv 0 \]

Example

if \ a \ then \ e \ else \ assert \ false = e + a \ 0 \equiv a \cdot e + a \ 0
\equiv 0 + a \ a \cdot e
Axiomatization / without loops

\[ e + a e \equiv e \quad e + a f \equiv f + \overline{a} e \quad e + a f \equiv ae + a f \quad \overline{aa} \equiv 0 \]

Example

\[
\text{if } a \text{ then } e \text{ else assert false } = e + a 0 \equiv ae + a 0 \\
\equiv 0 + a ae \\
\equiv 0e + \overline{a} ae
\]
Axiomatization / without loops

\[
e +_a e \equiv e \\
e +_a f \equiv f +_\bar{a} e \\
e +_a f \equiv ae +_a f \\
\begin{array}{c}
\bar{a}a \equiv 0 \\
oe \equiv 0
\end{array}
\]

Example

\[
\text{if } a \text{ then } e \text{ else assert false } = e +_a 0 \equiv ae +_a 0 \\
\equiv 0 +_a ae \\
\equiv 0e +_a ae \\
\equiv \bar{a}ae +_a ae
\]
Axiomatization / without loops

\[
\begin{align*}
e + \alpha e & \equiv e \\
e + \alpha f & \equiv f + \alpha e \\
e + \alpha f & \equiv \alpha e + \alpha f \\
\alpha a & \equiv 0 \\
0 e & \equiv 0
\end{align*}
\]

Example

\[
\text{if } \alpha \text{ then } e \text{ else assert false} = e + \alpha 0 \equiv \alpha e + \alpha 0
\]

\[
\equiv 0 + \alpha \alpha e
\equiv 0 e + \alpha \alpha e
\equiv \bar{\alpha} \alpha e + \alpha \alpha e
\equiv \alpha e + \alpha \alpha e
\]
Axiomatization / without loops

\[
\begin{align*}
e +_a e & \equiv e \\
\bar{e} +_a f & \equiv f +_a e \\
e +_a f & \equiv ae +_a f \\
\bar{a}a & \equiv 0 \\
0e & \equiv 0
\end{align*}
\]

Example

\[
\begin{align*}
\text{if } a \text{ then } e \text{ else assert false} &= e +_a 0 \equiv ae +_a 0 \\
&\equiv 0 +_a ae \\
&\equiv 0e +_a ae \\
&\equiv \bar{a}ae +_a ae \\
&\equiv ae +_a ae \\
&\equiv ae \\
&= \text{assert } a; e
\end{align*}
\]
Axiomatization / with loops

\[ e \equiv \begin{array}{c} fe + a \\ \hline \end{array} g \]

\[ e \equiv f^{(a)} g \]
Axiomatization with loops

\[ e \equiv f e +_a g \]

\[ e \equiv f^{(a)} g \]

Allows to derive \( 1 \equiv 1^{(1)} \), i.e.,

\[ \text{assert true} \equiv \text{while true do assert true} \]
Axiomatization / with loops

\[ e \equiv fe + a g \quad \text{f is productive} \]

\[ \Rightarrow e \equiv f(a) g \]
Axiomatisation / with loops

\[
e \equiv fe + ag \quad f \text{ is productive}
\]

\[
e \equiv f^{(a)} g
\]

\[
e^{(a)} \equiv ee^a + a 1
\]

Lemma

For every \(e\), there exists a productive \(\hat{e}\) such that

\[
e(a) \equiv \hat{e}(a)
\]

\[
e(a) \equiv \hat{e}\hat{e}(a) + a 1
\]
Axiomatization / with loops

\[ e \equiv fe + a \ g \quad \text{f is productive} \]

\[ e \equiv f^{(a)} g \]

\[ e^{(a)} \equiv ee^a + a \ 1 \]

\[ (e + a \ 1)^{(b)} \equiv (ae)^{(b)} \]
Axiomatization / with loops

\[ e \equiv fe + a \ g \quad \text{f is productive} \]

\[ e \equiv f(a) g \]

\[ e^{(a)} \equiv ee^a + a \ 1 \]

\[ (e + a \ 1)^{(b)} \equiv (ae)^{(b)} \]

Lemma

For every \( e \), there exists a productive \( \hat{e} \) such that \( e^{(b)} \equiv \hat{e}^{(b)} \); also, \( \hat{ae} \equiv a\hat{e} \).
Axiomatization / with loops

\[ e \equiv fe + a \; g \quad f \text{ is productive} \]
\[ e \equiv f^{(a)} g \]

\[ e \equiv ee^a + a \; 1 \]
\[ (e + a \; 1)^{(b)} \equiv (ae)^{(b)} \]

Lemma

For every \( e \), there exists a productive \( \hat{e} \) such that \( e^{(b)} \equiv \hat{e}^{(b)} \); also, \( \hat{ae} \equiv ae \).

Lemma

\[ e^{(a)} \]
\[ (ae)^{(a)} \]
Axiomatization / with loops

\[
\begin{align*}
  e & \equiv fe + ag & \text{if } f \text{ is productive} \\
  e & \equiv f(a)g \\
  e(a) & \equiv ee^a + a1 \\
  (e + a1)(b) & \equiv (ae)(b)
\end{align*}
\]

**Lemma**

For every \(e\), there exists a productive \(\hat{e}\) such that \(e(b) \equiv \hat{e}(b)\); also, \(\hat{ae} \equiv a\hat{e}\).

**Lemma**

\[
\begin{align*}
  e(a) & \equiv \hat{e}(a) \\
  (ae)^a & \equiv \hat{ae}(a)
\end{align*}
\]
Axiomatization / with loops

\[ \begin{align*}
e & \equiv fe + ag & \text{f is productive} \\
e & \equiv f(a)g \\
\end{align*} \]

\[ e^{(a)} \equiv ee^a + a \ 1 \]

\[ (e + a \ 1)^{(b)} \equiv (ae)^{(b)} \]

Lemma

For every \(e\), there exists a productive \(\hat{e}\) such that \(e^{(b)} \equiv \hat{e}^{(b)}\); also, \(\hat{ae} \equiv a\hat{e}\).

Lemma

\[ e^{(a)} \equiv \hat{e}^{(a)} \equiv \hat{e}e^{(a)} + a \ 1 \]

\[ (ae)^{(a)} \]
Axiomatization with loops

\[ e \equiv fe + a \ g \quad \text{if } f \text{ is productive} \]
\[ e \equiv f^{(a)} g \]

\[ e^{(a)} \equiv ee^a + a \ 1 \]
\[ (e + a \ 1)^{(b)} \equiv (ae)^{(b)} \]

Lemma

For every \( e \), there exists a productive \( \hat{e} \) such that \( e^{(b)} \equiv \hat{e}^{(b)} \); also, \( \hat{ae} \equiv a\hat{e} \).

Lemma

\[ e^{(a)} \equiv \hat{e}^{(a)} \equiv \hat{e}e^{(a)} + a \ 1 \equiv a\hat{e}e^{(a)} + a \ 1 \]
\[ (ae)^{(a)} \]
Axiomatization / with loops

\[
\begin{align*}
  e &\equiv f(e) + a \ g \\
  e &\equiv f(a) g \\
  e^{(a)} &\equiv ee^a + a \ 1 \\
  (e + a \ 1)^{(b)} &\equiv (ae)^{(b)}
\end{align*}
\]

Lemma

For every \( e \), there exists a productive \( \hat{e} \) such that \( e^{(b)} \equiv \hat{e}^{(b)} \); also, \( \hat{ae} \equiv a\hat{e} \).

Lemma

\[
\begin{align*}
  e^{(a)} &\equiv \hat{e}^{(a)} \equiv \hat{e}e^{(a)} + a \ 1 \equiv a\hat{e}e^{(a)} + a \ 1 \equiv a\hat{e}e^{(a)} + a \ 1 \\
  (ae)^{(a)}
\end{align*}
\]
Axiomatization / with loops

\[
\begin{align*}
\quad & \quad e \equiv fe + a \ g \quad \text{f is productive} \\
\quad & \quad e \equiv f^{(a)}g \\
\quad & \quad e^{(a)} \equiv ee^a + a \ 1 \\
\quad & \quad (e + a \ 1)^{(b)} \equiv (ae)^{(b)}
\end{align*}
\]

Lemma

For every \(e\), there exists a productive \(\hat{e}\) such that \(e^{(b)} \equiv \hat{e}^{(b)}\); also, \(\hat{ae} \equiv a\hat{e}\).

Lemma

\[
\begin{align*}
\quad & \quad e^{(a)} \equiv \hat{e}^{(a)} \equiv \hat{e}e^{(a)} + a \ 1 \equiv a\hat{e}e^{(a)} + a \ 1 \equiv (\hat{ae})^{(a)} \equiv (ae)^{(a)}
\end{align*}
\]
Theorem (Soundness)

If $e \equiv f$, then $[e] = [f]$. 

How about the converse?

1. $A \mapsto \rightarrow S(A)$ and $e \mapsto \rightarrow e$ with $e \equiv S(Ae)$.

2. If $A \leadsto A'$, then $S(A) \equiv S(A')$. 

$J[e]K = J[f]K = \Rightarrow L(Ae) = L(Af) = \Rightarrow Ae \leadsto Af = \Rightarrow Se \equiv Sf$. 

Tobias Kappé
Guarded Kleene Algebra with Tests
Programming Languages Discussion Group
Axiomatization / soundness & completeness

Theorem (Soundness)

If $e \equiv f$, then $[e] = [f]$.

How about the converse?

1. $A \mapsto S(A)$ and $e \mapsto A_e$ with $e \equiv S(A_e)$.
2. If $A \sim A'$, then $S(A) \equiv S(A')$. 
Axiomatization / soundness & completeness

Theorem (Soundness)

If $e \equiv f$, then $[e] = [f]$.

How about the converse?

1. $A \mapsto S(A)$ and $e \mapsto A_e$ with $e \equiv S(A_e)$.
2. If $A \sim A'$, then $S(A) \equiv S(A')$.

\[
[e] = [f] \implies L(A_e) = L(A_f) \\
\implies A_e \sim A_f \\
\implies S(A_e) \equiv S(A_f) \\
\implies e \equiv f
\]
A Kleene theorem / automata model
A Kleene theorem / automata model
A Kleene theorem / automata model

\[ \alpha \in L(A) \]

\[ X, \delta : X \rightarrow (2 + \Sigma \times X) \]

\[ s_1 \xrightarrow{\beta/p} s_2 \]

\[ \gamma/q \]
A Kleene theorem / automata model

\[ \alpha \in L(A) \]

\[ X, \delta : X \rightarrow (2^{\Sigma} \times X) \]

\[ \text{Atoms} \]

\[ \beta \gamma \alpha \]

\[ \beta/p \]

\[ \gamma/q \]

\[ s_1 \]

\[ s_2 \]
A Kleene theorem/automata model

A Kleene algebra is a mathematical structure that generalizes the operations of union, concatenation and Kleene closure of formal languages. It is used to model computations and is particularly useful in the study of automata and formal languages.

In the context of automata, Kleene's theorem states that the set of languages accepted by a finite automaton is precisely the set of languages that can be described by regular expressions. This theorem is fundamental to the theory of computation and has many applications in computer science, particularly in the field of programming languages.

The automaton model shown in the diagram represents a simple finite automaton with two states, $s_1$ and $s_2$, and transitions labeled with symbols from the alphabet $\Sigma$. The transitions are labeled with the symbols $\alpha$, $\beta$, and $\gamma$, and the corresponding inputs are $p$, $q$, and $q$ respectively.

The language $L(A)$ is the set of all strings accepted by the automaton $A$. The expression $\beta q \gamma p \alpha \in L(A)$ indicates that a string formed by concatenating the symbols $\beta$, $q$, $\gamma$, $p$, and $\alpha$ is accepted by the automaton $A$.
A Kleene theorem / automata model

\[
\alpha \Leftarrow s_1 \xrightarrow{\beta/p} s_2 \xleftarrow{\gamma/q} \beta p \gamma q \alpha \in L(A)
\]

\[
(X, \delta : X \rightarrow (2 + \Sigma \times X)^{Atoms})
\]
A Kleene theorem/automata model

Not described by an expression $e$:

See [Kozen and Tseng 2008].
A Kleene theorem / automata model

\eta \uparrow \downarrow 

\alpha, \beta

\gamma, \delta

NB: This slide had the wrong signature for \eta when I gave the talk; this is fixed now.
A Kleene theorem / automata model

\[ \eta \uparrow \]

\[ (\alpha, \beta) \]

\[ (\gamma, \delta) \]

\[ \gamma \]

\[ \downarrow \]

\[ \eta \delta \]

\[ \alpha / p \]

\[ \gamma / q \]

NB: This slide had the wrong signature for \( h \) when I gave the talk; this is fixed now.
A Kleene theorem / automata model

\[
\eta \\
\uparrow \\
\begin{array}{l}
\eta \\
\end{array}
\]

\[
\begin{array}{l}
\gamma \\
\end{array}
\]

\[
\begin{array}{l}
\alpha, \beta \\
\end{array}
\]

\[
\begin{array}{l}
\gamma, \delta \\
\end{array}
\]

\[
h : Atoms \rightarrow 2 + \Sigma \times X
\]

\[
h(\alpha) = (p, \Box)
\]

\[
h(\beta) = 0
\]

\[
h(\gamma) = (q, \bigstar)
\]

\[
h(\delta) = 1
\]

\[
h(-) = 0
\]

NB: This slide had the wrong signature for \( h \) when I gave the talk; this is fixed now.
A Kleene theorem / automata model

\[ h : \text{Atoms} \rightarrow 2 + \Sigma \times X \]

\[ h(\alpha) = (p, \blacksquare) \]

\[ h(\beta) = 0 \]

\[ h(\gamma) = (q, \bullet) \]

\[ h(\delta) = 1 \]

\[ h(\neg) = 0 \]

NB: This slide had the wrong signature for \( h \) when I gave the talk; this is fixed now.
A Kleene theorem / expressions to automata

\[ e = f + a \cdot g \]
A Kleene theorem / expressions to automata

\[ e = f + a \cdot g \]
A Kleene theorem / expressions to automata

\[ e = f + a g \]

\[ e = fg \]
A Kleene theorem / expressions to automata

\[ e = f + a \, g \]

\[ e = fg \]
A Kleene theorem / expressions to automata

\[ \text{e} = f +_a g \]

\[ \text{e} = fg \]

\[ \text{e} = f^{(a)} \]
A Kleene theorem / expressions to automata

\[ e = f + a \cdot g \]

\[ e = f \cdot g \]

\[ e = f^{(a)} \]
A Kleene theorem / automata to expressions
A Kleene theorem / automata to expressions

\[ x = 1 + \alpha p \cdot x + \beta 0 \]
\[ x = q \cdot x = 1 + \beta 0 \]
\[ x = 0 + \alpha p \cdot x + \beta 0 \]
A Kleene theorem / automata to expressions

\[ x \equiv 1 + \alpha p \cdot x_\bullet + \beta 0 \]

\[ x_\bullet \equiv 1 + \alpha 1 + \beta 0 \]

\[ x_{\uparrow} \equiv 0 + \alpha p \cdot x_\bullet + \beta 0 \]
A Kleene theorem / main result

Theorem

Let $L \subseteq \text{GS}(\Sigma, T)$. The following are equivalent:

1. $L = [e]$ for some $e$.
2. $L$ is accepted by a well-nested and finite automaton.
Discussion

- Which theories could we embed while keeping decidability?
- Parameterised semantics besides relational and probabilistic?
- How do we recover a small program from an automaton?
- Which extensions of the syntax would be interesting?
kap.pe/slides

arxiv.org/abs/1907.05920