Deciding equivalence using type checking

Colin Stirling
cps@inf.ed.ac.uk

LFCS
School of Informatics
University of Edinburgh

IHP Abstraction and Verification in Semantics
Paris, 27th June 2014
Methods for verifying finite and infinite state systems

- Notable success in Computer Science
- **model checking + equivalence checking**
- System = finite/infinite state transition graph
Methods for verifying finite and infinite state systems

- Notable success in Computer Science
- **model checking + equivalence checking**
- System = finite/infinite state transition graph
- Model checking: does state $s$ have property $\Phi$?
- apply automata/game theoretic techniques to solve it: mostly computing monadic fixed points, reachability sets by traversing graph (possibly repeatedly)
Methods for verifying finite and infinite state systems

Notable success in Computer Science

**model checking + equivalence checking**

System = finite/infinite state transition graph

Model checking: does state $s$ have property $\Phi$?

apply automata/game theoretic techniques to solve it: mostly computing monadic fixed points, reachability sets by traversing graph (possibly repeatedly)

Equivalence checking: is state $s$ equivalent to $t$?

mostly computing dyadic fixed points e.g. bisimulations to solve it. May need algebraic/combinatorial properties of reachability sets/generators of graph
Transfer these techniques to systems with binding

1. Model checking higher-order trees/schemes
   Using parity automata and geometry of interaction or game semantics or Krivine machines; alternative type checking

Transfer these techniques to systems with binding

1. Model checking higher-order trees/schemes
   Using parity automata and geometry of interaction or game semantics or Krivine machines; alternative type checking

2. Deciding observational equivalence for fragments of idealized Algol and ML (w.r.t. finite value sets)
   Reduce to equivalence of automata (such as deterministic pushdown automata)
   [Ghica, McCusker 2000; Ong 2002; . . .; Hopkins, Murawski, Ong 2012; . . .]
Transfer these techniques to systems with binding

1. Model checking higher-order trees/schemes
   Using parity automata and geometry of interaction or game semantics or Krivine machines; alternative type checking

2. Deciding observational equivalence for fragments of idealized Algol and ML (w.r.t. finite value sets)
   Reduce to equivalence of automata (such as deterministic pushdown automata)
   [Ghica, McCusker 2000; Ong 2002; . . .; Hopkins, Murawski, Ong 2012; . . .]

3. : : :
Example scheme: second-order

\[ F_{x_1 x_2 x_3} \overset{\text{def}}{=} f \left( F(G_{x_1})(H_{x_2})x_3 \right) x_1(x_2 x_3) \]
\[ G_{y_1 y_2} \overset{\text{def}}{=} g(y_1(y_2)) \]
\[ H_{z_1 z_2} \overset{\text{def}}{=} h(z_1(z_2)) \]
\[ Fgha \quad \text{Start} \]
Example scheme: second-order

\[
Fx_1x_2x_3 \overset{\text{def}}{=} f\left(F(Gx_1)(Hx_2)x_3\right)x_1(x_2x_3)
\]

\[
Gy_1y_2 \overset{\text{def}}{=} g(y_1(y_2))
\]

\[
Hz_1z_2 \overset{\text{def}}{=} h(z_1(z_2))
\]

\[Fgha\]

\[\text{Start}\]
Model checking a scheme

Want to run a finite state automaton on tree generated by scheme

May have a transition

Need to do this on its finite (lambda calculus) description
Model checking a scheme

Want to run a finite state automaton on tree generated by scheme
Model checking a scheme

Want to run a finite state automaton on tree generated by scheme

May have a transition $q \xrightarrow{f} (q_1, q_2)$
Model checking a scheme

Want to run a finite state automaton on tree generated by scheme
May have a transition $q \xrightarrow{f} (q_1, q_2)$
Need to do this on its finite (lambda calculus) description
Model checking a scheme

\[
F_{x_1 x_2 x_3} \overset{\text{def}}{=} f(F(G_{x_1})(H_{x_2})x_3)x_1(x_2 x_3)
\]

\[
G_{y_1 y_2} \overset{\text{def}}{=} g(y_1(y_2))
\]

\[
H_{z_1 z_2} \overset{\text{def}}{=} h(z_1(z_2))
\]

\[
Fgha \quad \text{Start}
\]

Want to run a finite state automaton on tree generated by scheme

May have a transition \( q \xrightarrow{f} (q_1, q_2) \)

Need to do this on its finite (lambda calculus) description
Model checking a scheme

\[ Fx_1x_2x_3 \overset{\text{def}}{=} f(F(Gx_1)(Hx_2)x_3)x_1(x_2x_3) \]
\[ Gy_1y_2 \overset{\text{def}}{=} g(y_1(y_2)) \]
\[ Hz_1z_2 \overset{\text{def}}{=} h(z_1(z_2)) \]
\[ Fgha \quad \text{Start} \]

Want to run a finite state automaton on tree generated by scheme

May have a transition \( q \xrightarrow{f} (q_1, q_2) \)

Need to do this on its finite (lambda calculus) description

Application operator is essential component
Model checking a scheme

\[
F_{x_1x_2x_3} \overset{\text{def}}{=} f \left( F(G_{x_1})(H_{x_2})x_3 \right) x_1(x_2x_3)
\]

\[
G_{y_1y_2} \overset{\text{def}}{=} g(y_1(y_2))
\]

\[
H_{z_1z_2} \overset{\text{def}}{=} h(z_1(z_2))
\]

\[
Fgha \quad \text{Start}
\]

Want to run a finite state automaton on tree generated by scheme
May have a transition \( q \xrightarrow{f} (q_1, q_2) \)
Need to do this on its finite (lambda calculus) description
Application operator is essential component
Leads to non-standard automata
Automaton running on lambda term applied to two terms
Automaton running on lambda term applied to two terms
Automaton running on lambda term applied to two terms
Automaton running on lambda term applied to two terms
Automaton running on lambda term applied to two terms
Automaton running on lambda term applied to two terms
Automaton running on lambda term applied to two terms
Automaton running on lambda term applied to two terms
Automaton running on lambda term applied to two terms
Automaton running on lambda term applied to two terms
Automaton running on lambda term applied to two terms
Model checking as type checking

- Encode running of automaton using intersection types

\[ \theta := q \mid \tau \rightarrow \theta q \]

state of automaton

\[ \tau := \bigwedge_{i \in I_1} \theta i_1 \land \ldots \land \bigwedge_{i \in I_m} \theta i_m \]

finite

- Thm Automaton accepts Scheme iff \[ \vdash \text{Scheme} : \text{initial state} \]

- Decision procedure: via the (finite) typing rules

- Can this technique work for equivalence checking?

- Use type checking to solve equivalence problem (for real-time strict deterministic pushdown automata)

Inspired by [Tsukada, Kobayashi 2012] which looks at special language inclusion problems
Encode running of automaton using intersection types

\[
\begin{align*}
 \theta &:= q \mid \tau \rightarrow \theta \\
 \tau &:= \bigwedge_{i_1 \in I_1} \theta_{i_1} \land \ldots \land \bigwedge_{i_m \in I_m} \theta_{i_m} \\
 &\text{\hspace{1em}} q \text{ state of automaton} \\
 &\text{\hspace{1em}} I_j \text{ finite}
\end{align*}
\]
Model checking as type checking

- Encode running of automaton using intersection types

\[ \theta ::= q \mid \tau \rightarrow \theta \quad \text{\( q \) state of automaton} \]
\[ \tau ::= \bigwedge_{i_1 \in I_1} \theta_{i_1} \land \ldots \land \bigwedge_{i_m \in I_m} \theta_{i_m} \quad \text{\( l_j \) finite} \]

- Thm Automaton accepts Scheme iff \( \vdash \) Scheme : initial state
Model checking as type checking

- Encode running of automaton using intersection types

\[ \theta := q \mid \tau \rightarrow \theta \quad q \text{ state of automaton} \]
\[ \tau := \bigwedge_{i_1 \in I_1} \theta_{i_1} \land \ldots \land \bigwedge_{i_m \in I_m} \theta_{i_m} \quad I_j \text{ finite} \]

- Thm Automaton accepts Scheme iff \( \vdash \) Scheme : initial state

- Decision procedure: via the (finite) typing rules

Inspired by [Tsukada, Kobayashi 2012] which looks at special language inclusion problems
Encode running of automaton using intersection types

\[ \theta := q \mid \tau \rightarrow \theta \]
\[ \tau := \bigwedge_{i_1 \in I_1} \theta_{i_1} \land \ldots \land \bigwedge_{i_m \in I_m} \theta_{i_m} \]

\text{q state of automaton}

\text{\tau \ j \ finite}

Thm Automaton accepts Scheme iff \ \vdash \ Scheme : initial state

Decision procedure: via the (finite) typing rules

Can this technique work for equivalence checking?
Model checking as type checking

 Encode running of automaton using intersection types

\[ \theta := q \mid \tau \rightarrow \theta \]
\[ \tau := \bigwedge_{i_1 \in I_1} \theta_{i_1} \land \ldots \land \bigwedge_{i_m \in I_m} \theta_{i_m} \]

- \( q \) state of automaton
- \( I_j \) finite

Thm Automaton accepts Scheme iff \( \vdash \) Scheme : initial state

Decision procedure: via the (finite) typing rules

Can this technique work for equivalence checking?

Use type checking to solve equivalence problem (for real-time strict deterministic pushdown automata)

Inspired by [Tsukada, Kobayshi 2012] which looks at special language inclusion problems
Real-time strict deterministic pushdown automata

Finite sets: $Q$ states, $\Gamma$ stack symbols, $A$ alphabet and $T$ basic transitions
Real-time strict deterministic pushdown automata

Finite sets: $Q$ states, $\Gamma$ stack symbols, $A$ alphabet and $T$ basic transitions

$pX \xrightarrow{a} q\alpha$ where $p, q \in Q$, $a \in A$, $X \in \Gamma$ and $\alpha \in \Gamma^*$
Real-time strict deterministic pushdown automata

Finite sets: $Q$ states, $\Gamma$ stack symbols, $A$ alphabet and $T$ basic transitions

$pX \xrightarrow{a} q\alpha$ where $p, q \in Q$, $a \in A$, $X \in \Gamma$ and $\alpha \in \Gamma^*$

Deterministic: if $pX \xrightarrow{a} q\alpha \in T$ and $pX \xrightarrow{a} r\gamma \in T$ then $q = r$ and $\alpha = \gamma$
Real-time strict deterministic pushdown automata

Finite sets: $Q$ states, $\Gamma$ stack symbols, $A$ alphabet and $T$ basic transitions

$pX \xrightarrow{a} q\alpha$ where $p, q \in Q$, $a \in A$, $X \in \Gamma$ and $\alpha \in \Gamma^*$

Deterministic: if $pX \xrightarrow{a} q\alpha \in T$ and $pX \xrightarrow{a} r\gamma \in T$ then $q = r$ and $\alpha = \gamma$

Configuration $p\alpha$ where $p \in Q$ and $\alpha \in \Gamma^*$
Real-time strict deterministic pushdown automata

Finite sets: $Q$ states, $\Gamma$ stack symbols, $A$ alphabet and $T$ basic transitions

$pX \xrightarrow{a} q\alpha$ where $p, q \in Q$, $a \in A$, $X \in \Gamma$ and $\alpha \in \Gamma^*$

Deterministic: if $pX \xrightarrow{a} q\alpha \in T$ and $pX \xrightarrow{a} r\gamma \in T$ then $q = r$ and $\alpha = \gamma$

Configuration $p\alpha$ where $p \in Q$ and $\alpha \in \Gamma^*$

Transitions of configuration: if $pX \xrightarrow{a} q\alpha \in T$ then $pX\beta \xrightarrow{a} q\alpha\beta$
Real-time strict deterministic pushdown automata

Finite sets: $Q$ states, $\Gamma$ stack symbols, $A$ alphabet and $T$ basic transitions

$pX \xrightarrow{a} q\alpha$ where $p, q \in Q$, $a \in A$, $X \in \Gamma$ and $\alpha \in \Gamma^*$

Deterministic: if $pX \xrightarrow{a} q\alpha \in T$ and $pX \xrightarrow{a} r\gamma \in T$ then $q = r$ and $\alpha = \gamma$

Configuration $p\alpha$ where $p \in Q$ and $\alpha \in \Gamma^*$

Transitions of configuration: if $pX \xrightarrow{a} q\alpha \in T$ then $pX\beta \xrightarrow{a} q\alpha\beta$

Transition relation extended to words $\xrightarrow{w}$, $w \in A^*$
Real-time strict deterministic pushdown automata

Finite sets: \( Q \) states, \( \Gamma \) stack symbols, \( A \) alphabet and \( T \) basic transitions

\[ pX \xrightarrow{a} q\alpha \quad \text{where } p, q \in Q, \ a \in A, \ X \in \Gamma \text{ and } \alpha \in \Gamma^* \]

Deterministic: if \( pX \xrightarrow{a} q\alpha \in T \) and \( pX \xrightarrow{a} r\gamma \in T \) then \( q = r \) and \( \alpha = \gamma \)

Configuration \( p\alpha \) where \( p \in Q \) and \( \alpha \in \Gamma^* \)

Transitions of configuration: if \( pX \xrightarrow{a} q\alpha \in T \) then \( pX\beta \xrightarrow{a} q\alpha\beta \)

Transition relation extended to words \( \xrightarrow{w}, \ w \in A^* \)

Language accepted \( L(p\alpha) = \{ w \mid p\alpha \xrightarrow{w} q\varepsilon \text{ for some } q \} \)
Real-time strict deterministic pushdown automata

Finite sets: $Q$ states, $\Gamma$ stack symbols, $A$ alphabet and $T$ basic transitions

$pX \xrightarrow{a} q\alpha$ where $p, q \in Q$, $a \in A$, $X \in \Gamma$ and $\alpha \in \Gamma^*$

Deterministic: if $pX \xrightarrow{a} q\alpha \in T$ and $pX \xrightarrow{a} r\gamma \in T$ then $q = r$ and $\alpha = \gamma$

Configuration $p\alpha$ where $p \in Q$ and $\alpha \in \Gamma^*$

Transitions of configuration: if $pX \xrightarrow{a} q\alpha \in T$ then $pX\beta \xrightarrow{a} q\alpha\beta$

Transition relation extended to words $\xrightarrow{w}$, $w \in A^*$

Language accepted $L(p\alpha) = \{w | p\alpha \xrightarrow{w} q\varepsilon \text{ for some } q\}$

Equivalence: given $p\alpha$ and $q\beta$ is $L(p\alpha) = L(q\beta)$?
Example

$Q = \{p, p_1, p_2, p_3\}$, $\Gamma = \{X, Y\}$ and $A = \{a, b, c\}$. $T$ is

\[
\begin{align*}
pX & \xrightarrow{a} p_1X & pX & \xrightarrow{b} p_2\epsilon & p_2X & \xrightarrow{c} p_3X \\
p_1X & \xrightarrow{a} pXX & p_1X & \xrightarrow{b} p_3X & p_3X & \xrightarrow{c} p_2\epsilon \\
pY & \xrightarrow{a} pYY & pY & \xrightarrow{b} p_1\epsilon & p_1Y & \xrightarrow{c} p_1\epsilon
\end{align*}
\]
Example

\[ Q = \{ p, p_1, p_2, p_3 \}, \Gamma = \{ X, Y \} \text{ and } A = \{ a, b, c \}. \]  
\[ T \text{ is} \]

\[
pX \xrightarrow{a} p_1 X \\
p_1 X \xrightarrow{a} pXX \\
pY \xrightarrow{a} pYYY \\
pX \xrightarrow{b} p_2 \epsilon \\
p_1 X \xrightarrow{b} p_3 X \\
pY \xrightarrow{b} p_1 \epsilon \\
p_2 X \xrightarrow{c} p_3 X \\
p_3 X \xrightarrow{c} p_2 \epsilon \\
p_1 Y \xrightarrow{c} p_1 \epsilon
\]

\[ p_1 YYY \xrightarrow{c} p_1 YY \text{ because } p_1 Y \xrightarrow{c} p_1 \epsilon \in T \]
For $n > 0$, $L(pX^n) = L(pY^{2n-1})$
Where is application?

- Assume states \( \{p_1, \ldots, p_k\} \)
Where is application?

- Assume states \( \{p_1, \ldots, p_k\} \)
- Configuration \( p_\gamma \alpha \) is \( p_\gamma \) applied to \( p_1 \alpha, \ldots, p_k \alpha \)
Assume states \( \{p_1, \ldots, p_k\} \)

Configuration \( p_\gamma\alpha \) is \( p_\gamma \) applied to \( p_1\alpha, \ldots, p_k\alpha \)

Types \( \tau ::= (\theta_1, \ldots, \theta_k) \rightarrow q_\beta \)

\( q_\beta \) is a configuration and \( \theta_i \) finite set of configurations
Meaning of a type

\[ p\alpha : (\theta_1, \ldots, \theta_k) \rightarrow q\beta \text{ iff} \]

\[ \frac{p\alpha \sqsupset q\beta \text{ (consonance)}}{\text{iff} \quad \text{if } w \in L(p\alpha) \text{, then there is a } v, wv \in L(q\beta)} \]

Key property: there is a \( m \), if \( p\alpha \sqsupset q\beta \) then there is a prefix \( \beta' \) of \( \beta \), \( p\alpha \sqsupset q\beta' \) and \(|\beta'| \leq m |\alpha| \)
Meaning of a type

\[ p\alpha : (\theta_1, \ldots, \theta_k) \rightarrow q\beta \text{ iff} \]

1. \( p\alpha \preceq q\beta \) (consonance)
2. if \( p\alpha \xrightarrow{w} p_i \varepsilon \) and \( q\beta \xrightarrow{w} q'\gamma \) then \( q'\gamma \in \theta_i \)

where \( p\alpha \preceq q\beta \) iff

- if \( w \in L(p\alpha) \), then there is a \( v \), \( wv \in L(q\beta) \)
- if \( w \in L(q\beta) \), then there is a prefix of \( w \), \( w' \in L(p\alpha) \)
Meaning of a type

\[ p_\alpha : (\theta_1, \ldots, \theta_k) \rightarrow q_\beta \text{ iff } \]

1. \( p_\alpha \sqsubseteq q_\beta \) (consonance)
2. if \( p_\alpha \xrightarrow{w} p_i \varepsilon \) and \( q_\beta \xrightarrow{w} q'_\gamma \) then \( q'_\gamma \in \theta_i \)

where \( p_\alpha \sqsubseteq q_\beta \) iff

- if \( w \in L(p_\alpha) \), then there is a \( v \), \( wv \in L(q_\beta) \)
- if \( w \in L(q_\beta) \), then there is a prefix of \( w \), \( w' \in L(p_\alpha) \)

Key property: there is a \( m \), if \( p_\alpha \sqsubseteq q_\beta \) then there is a prefix \( \beta' \) of \( \beta \), \( p_\alpha \sqsubseteq q_\beta' \) and \( |\beta'| \leq m|\alpha| \)
Proof system

Type assumptions $qX : τ ∈ Δ$
Proof system

Type assumptions $qX : \tau \in \Delta$

Axiom

$\Delta \vdash pX : (\theta_1, \ldots, \theta_k) \rightarrow q/\beta\delta$ if $pX : (\theta'_1, \ldots, \theta'_k) \rightarrow q/\beta \in \Delta$ and $\theta_i = \{ r\lambda\delta \mid r\lambda \in \theta'_i \}$
Proof system

Type assumptions $qX : \tau \in \Delta$

Axiom

$\Delta \vdash pX : (\theta_1, \ldots, \theta_k) \rightarrow q_\beta \delta$ if $pX : (\theta'_1, \ldots, \theta'_k) \rightarrow q_\beta \in \Delta$ and

$\theta_i = \{ r_\lambda \delta \mid r_\lambda \in \theta'_i \}$

Application rule

$\Delta \vdash pX : (\theta'_1, \ldots, \theta'_k) \rightarrow q_\beta \ldots$, $\Delta \vdash p_j Y_\alpha : (\theta''_1, \ldots, \theta''_k) \rightarrow r_{ji} \lambda_{ji}$

$\Delta \vdash pXY_\alpha : (\theta_1, \ldots, \theta_k) \rightarrow q_\beta$

for all $j$ and $r_{ji} \lambda_{ji} \in \theta'_j$ and $\theta_m = \bigcup_j \bigcup_i \theta''_m$
Proof system

Type assumptions $qX : \tau \in \Delta$

**Axiom**

$\Delta \vdash pX : (\theta_1, \ldots, \theta_k) \rightarrow q_\beta \delta$ if $pX : (\theta'_1, \ldots, \theta'_k) \rightarrow q_\beta \in \Delta$ and

$\theta_i = \{ r\lambda\delta \mid r\lambda \in \theta'_i \}$

**Application rule**

$$
\Delta \vdash pX : (\theta'_1, \ldots, \theta'_k) \rightarrow q_\beta \ldots, \Delta \vdash p_jY\alpha : (\theta''_1, \ldots, \theta''_k) \rightarrow r_{ji\lambda_{ji}}
$$

$$
\Delta \vdash pXY\alpha : (\theta_1, \ldots, \theta_k) \rightarrow q_\beta
$$

for all $j$ and $r_{ji\lambda_{ji}} \in \theta'_j$ and $\theta_m = \bigcup_j \bigcup_i \theta''_m$

$\Delta$ needs to be closed under transitions
Δ closed under transitions

If \( pX : (\theta_1, \ldots, \theta_k) \rightarrow q\beta \in \Delta \) then

1. if \( pX \xrightarrow{\alpha} p' \) then \( q\beta \xrightarrow{\beta'} q' \) and vice versa
2. if \( pX \xrightarrow{\alpha} p_i \in \varepsilon \) and \( q\beta \xrightarrow{\beta'} q' \) then \( q' \beta' \in \theta_i \)
3. if \( pX \xrightarrow{\alpha} rZ \) and \( q\beta \xrightarrow{\beta'} q' \) then \( \Delta \vdash rZ : (\theta'_1, \ldots, \theta_k) \rightarrow q' \beta' \) for \( \theta'_i \subseteq \theta_i \)
4. \( \theta_i \) is union of cases 2 and 3
$\Delta$ closed under transitions

If $pX : (\theta_1, \ldots, \theta_k) \rightarrow q\beta \in \Delta$ then

1. if $pX \xrightarrow{a} p'\alpha$ then $q\beta \xrightarrow{a} q'\beta'$ and vice versa
\[\Delta \text{ closed under transitions}\]

If \( pX : (\theta_1, \ldots, \theta_k) \rightarrow q\beta \in \Delta \) then

1. If \( pX \xrightarrow{a} p'\alpha \) then \( q\beta \xrightarrow{a} q'\beta' \) and vice versa
2. If \( pX \xrightarrow{a} p;\epsilon \) and \( q\beta \xrightarrow{a} q'\beta' \) then \( q'\beta' \in \theta_i \) for \( \theta_i \subseteq \theta_i \)
$\Delta$ closed under transitions

If $pX : (\theta_1, \ldots, \theta_k) \rightarrow q\beta \in \Delta$ then

1. if $pX \xrightarrow{a} p'\alpha$ then $q\beta \xrightarrow{a} q'\beta'$ and vice versa
2. if $pX \xrightarrow{a} p_i \varepsilon$ and $q\beta \xrightarrow{a} q'\beta'$ then $q'\beta' \in \theta_i$
3. if $pX \xrightarrow{a} rZ\alpha$ and $q\beta \xrightarrow{a} q'\beta'$ then
   $\Delta \vdash rZ\alpha : (\theta_1', \ldots, \theta_k') \rightarrow q'\beta'$ for $\theta_i' \subseteq \theta_i$
\[\Delta\] closed under transitions

If \( pX : (\theta_1, \ldots, \theta_k) \rightarrow q\beta \in \Delta \) then

1. if \( pX \overset{a}{\rightarrow} p'\alpha \) then \( q\beta \overset{a}{\rightarrow} q'\beta' \) and vice versa
2. if \( pX \overset{a}{\rightarrow} p_i \varepsilon \) and \( q\beta \overset{a}{\rightarrow} q'\beta' \) then \( q'\beta' \in \theta_i \)
3. if \( pX \overset{a}{\rightarrow} rZ\alpha \) and \( q\beta \overset{a}{\rightarrow} q'\beta' \) then
   \( \Delta \vdash rZ\alpha : (\theta_1', \ldots, \theta_k) \rightarrow q'\beta' \) for \( \theta_i' \subseteq \theta_i \)
4. \( \theta_i \) is union of cases 2 and 3
Equivalence checking as type checking

- Reduce \( L(p\alpha) = L(q\beta) \) to
Equivalence checking as type checking

- Reduce $L(p^\alpha) = L(q^\beta)$? to
- $\Delta \vdash p^\alpha : (\theta_1^0, \ldots, \theta_k^0) \to q^\beta$? where each $\theta_i^0 \subseteq \{p_1^\varepsilon, \ldots, p_k^\varepsilon\}$
Example

\[
pX \xrightarrow{a} p_1 X \xrightarrow{a} pXX \xrightarrow{a} p_1 XX \xrightarrow{a} \ldots
\]
\[
downarrow b \downarrow b \downarrow b \downarrow b \downarrow b \ldots
\]
\[
p_2 \varepsilon \xleftarrow{c} p_3 X \xleftarrow{c} p_2 X \xleftarrow{c} p_3 XX \xleftarrow{c} \ldots
\]
\[
pY \xrightarrow{a} pYY \xrightarrow{a} pYYY \xrightarrow{a} pYYYY \xrightarrow{a} \ldots
\]
\[
downarrow b \downarrow b \downarrow b \downarrow b \downarrow b \ldots
\]
\[
p_1 \varepsilon \xleftarrow{c} p_1 Y \xleftarrow{c} p_1 YY \xleftarrow{c} p_1 YYY \xleftarrow{c} \ldots
\]

Assume states are ordered \( p, p_1, p_2, p_3 \) and \( \pi = (\emptyset, \emptyset, \{p_1 \varepsilon\}, \emptyset) \)

For any \( n > 0 \), \( \Delta \vdash pX^n : \pi \rightarrow pY^{2n-1} \)

where \( \Delta = \{pX : \pi \rightarrow pY, p_1 X : \pi \rightarrow pYY, p_2 X : \pi \rightarrow p_1 YY, p_3 X : \pi \rightarrow p_1 Y\} \)
Proof tree upside down

\[ \Delta \vdash pX^4 : \pi \rightarrow pY^7 \]

\[ \Delta \vdash pX : (\emptyset, \emptyset, \{ p_Y^6 \}, \emptyset) \rightarrow pY^7 \]
\[ \Delta \vdash p_2 X^3 : \pi \rightarrow p_Y^6 \]
\[ \Delta \vdash p_2 X : (\emptyset, \emptyset, \{ p_Y^4 \}, \emptyset) \rightarrow p_Y^6 \]

where ... is the subtree

\[ \Delta \vdash p_2 X^2 : \pi \rightarrow p_Y^4 \]

\[ \Delta \vdash p_2 X : (\emptyset, \{ p_Y^2 \}, \emptyset, \emptyset) \rightarrow p_Y^4 \]
\[ \Delta \vdash p_2 X : \pi \rightarrow p_Y^2 \]

where \( \Delta = \{ pX : \pi \rightarrow pY, p_1 X : \pi \rightarrow pYY, p_2 X : \pi \rightarrow p_1 YY, p_3 X : \pi \rightarrow p_Y \} \) and \( \pi = (\emptyset, \emptyset, \{ p_1 \epsilon \}, \emptyset) \)
Δ closed under transitions

\[
\begin{array}{llll}
pX : \pi \to pY & pX : \pi \to pY & p1X : \pi \to pY^2 \\
da & da & db & da \\
p1X : \pi \to pY^2 & p2\varepsilon & p1\varepsilon & pX^2 : \pi \to pY^3 \\
b & b & c & c \\
p3X : \pi \to p1Y & p3X : \pi \to p1Y & p3X : \pi \to p1Y \\
b & b & c & c \\
p3X : \pi \to p1Y & p3X : \pi \to p1Y & p3X : \pi \to p1Y \\
\end{array}
\]

\[\Delta \vdash pX^2 : \pi \to pY^3\]

\[\Delta \vdash pX : (\emptyset, \emptyset, \{p1Y^2\}, \emptyset) \to pY^3 \quad \Delta \vdash p2X : \pi \to p1Y^2\]

where \(\Delta = \{pX : \pi \to pY, p1X : \pi \to pYY, p2X : \pi \to p1YY, p3X : \pi \to p1Y\}\) and \(\pi = (\emptyset, \emptyset, \{p1\varepsilon\}, \emptyset)\)
Conclusion

- Use type checking to solve equivalence problem (for real-time strict deterministic pushdown automata)

[Oyamaguchi, Honda, Inagaki 1980] showed decidability without complexity upper bound. Only known upper bound is the one for equivalence of full deterministic pushdown automata [Stirling 2002].


At one stage I was convinced it did lead to better bound via an upper bound on $m$ in key property. Key property: there is a $m$, if $pX \sqsubseteq q \beta$ then there is a prefix $\beta'$ of $\beta$, $pX \sqsubseteq q \beta'$ and $|\beta'| \leq m$. Does the technique naturally extend to schema?
Conclusion

- Use type checking to solve equivalence problem (for real-time strict deterministic pushdown automata)
- [Oyamaguchi, Honda, Inagaki 1980] showed decidability without complexity upper bound

Only known upper bound is the one for equivalence of full deterministic pushdown automata [Stirling 2002]
Conclusion

- Use type checking to solve equivalence problem (for real-time strict deterministic pushdown automata)

- [Oyamaguchi, Honda, Inagaki 1980] showed decidability without complexity upper bound
  
  Only known upper bound is the one for equivalence of full deterministic pushdown automata [Stirling 2002]

- Type checking algorithm and its correctness proof much simpler than [S 2002]. Like [OHI 1980] and unlike [S 2002] algorithm is nondeterministic

Key property: there is a \( m \), if \( pX \sqsubset q \beta \) then there is a prefix \( \beta' \) of \( \beta \), \( pX \sqsubset q \beta' \) and \( |\beta'| \leq m \)
Conclusion

- Use type checking to solve equivalence problem (for real-time strict deterministic pushdown automata)

- [Oyamaguchi, Honda, Inagaki 1980] showed decidability without complexity upper bound
  
  Only known upper bound is the one for equivalence of full deterministic pushdown automata [Stirling 2002]

- Type checking algorithm and its correctness proof much simpler than [S 2002]. Like [OHI 1980] and unlike [S 2002] algorithm is nondeterminisitc

- At one stage I was convinced it did lead to better bound via an upper bound on $m$ in key property

  Key property: there is a $m$, if $pX \preceq q\beta$ then there is a prefix $\beta'$ of $\beta$, $pX \preceq q\beta'$ and $|\beta'| \leq m$
Conclusion

- Use type checking to solve equivalence problem (for real-time strict deterministic pushdown automata)
- [Oyamaguchi, Honda, Inagaki 1980] showed decidability without complexity upper bound
  
  Only known upper bound is the one for equivalence of full deterministic pushdown automata [Stirling 2002]

- Type checking algorithm and its correctness proof much simpler than [S 2002]. Like [OHI 1980] and unlike [S 2002] algorithm is nondeterministic

- At one stage I was convinced it did lead to better bound via an upper bound on $m$ in key property
  
  Key property: there is a $m$, if $pX \preceq q\beta$ then there is a prefix $\beta'$ of $\beta$, $pX \preceq q\beta'$ and $|\beta'| \leq m$

- Does the technique naturally extend to schema?