Algorithms and Applications of Higher-Order Model Checking

Naoki Kobayashi
Tohoku University

This Talk

- Application of HO model checking to CFA
  (joint work with Tobita and Tsukada)
  - Aims of the talk:
    - Familiarize audience with connections between HO model checking and program verification
    - Get feedbacks from CFA community
- Algorithms for HO model checking
  - Aims of the talk:
    - Familiarize audience with type-based approach to HO model checking
    - Get feedbacks from game semantics community

Outline

- Brief Review of HO model checking problem
- Application to CFA (20 min.)
- Algorithms for higher-order model checking (25 min.)
- Future directions

HO Model Checking Problem

Given
- G: higher-order recursion scheme
- A: alternating parity tree automaton (APT) (a formula of modal μ-calculus or MSO),

does A accept Tree(G)?

e.g.
- Does every finite path end with "c"?
- Does "a" occur below "b"?

n-EXPTIME-complete [Ong, LICS06] (for order-n recursion scheme)

$$n\left\{\begin{array}{l} 2^{p(x)} \\ 2^{2^{\cdots}} \\ 2 \end{array}\right.$$

Outline

- Brief Review of HO model checking problem
- Application to CFA
  - Definition of CFA
  - Reduction from CFA to HO model checking
  - Discussion
- Algorithms for higher-order model checking
- Future directions

CFA Problem

Given a closed (simply-typed) λ-term M and labels l and m, decide whether

$$M \rightarrow^* E[\ (\lambda^l x.N)@^m V\ ]$$

for some x, E, N, V.

Syntax:

$M,N ::= () \mid x \mid \lambda^l x.M \mid M@^l N$ (terms)

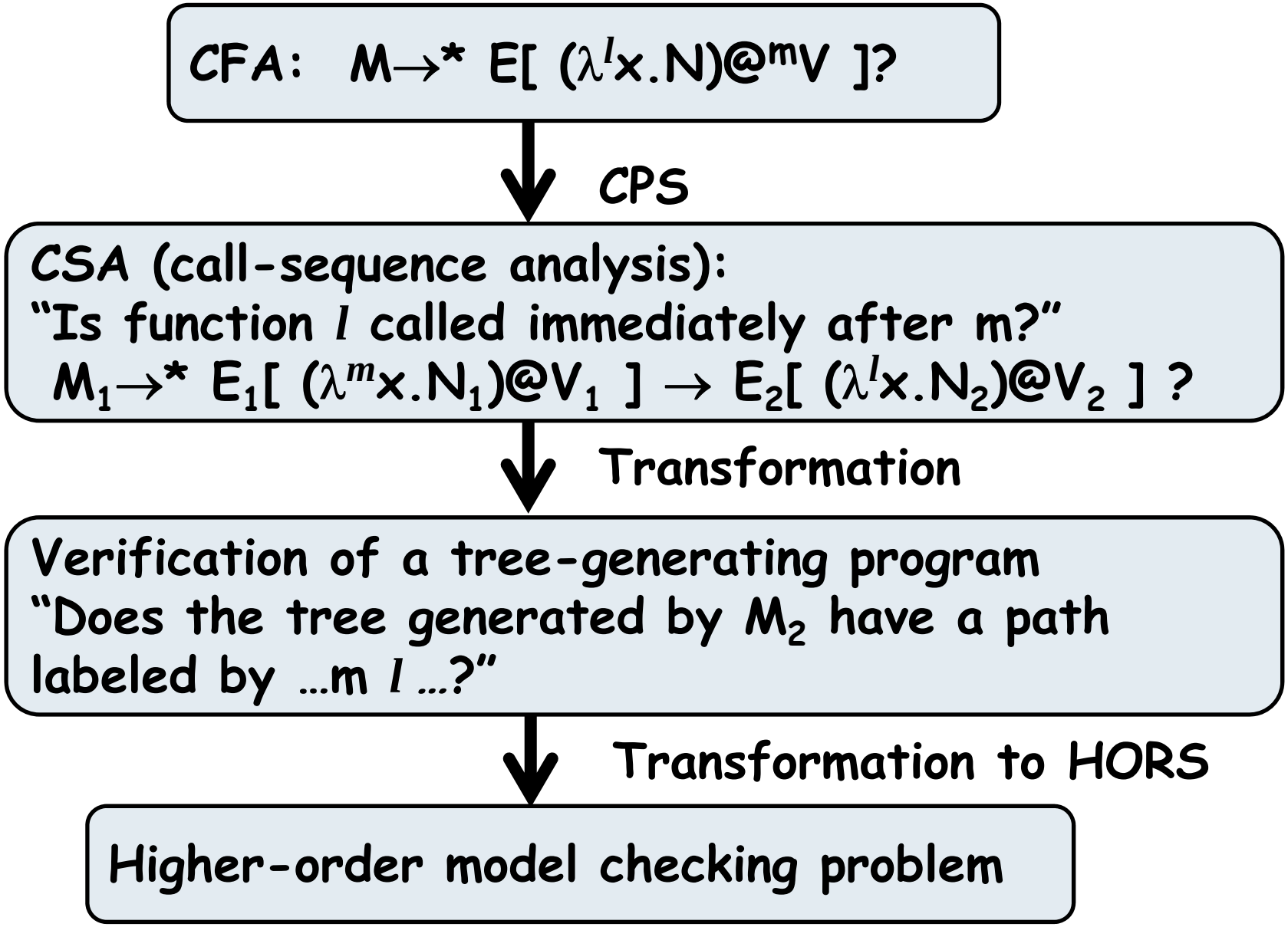
$E ::= [\ ] \mid [\ ]\ M \mid V\ [\ ]$ (evaluation contexts)

$V ::= () \mid \lambda^l x.M$ (values)

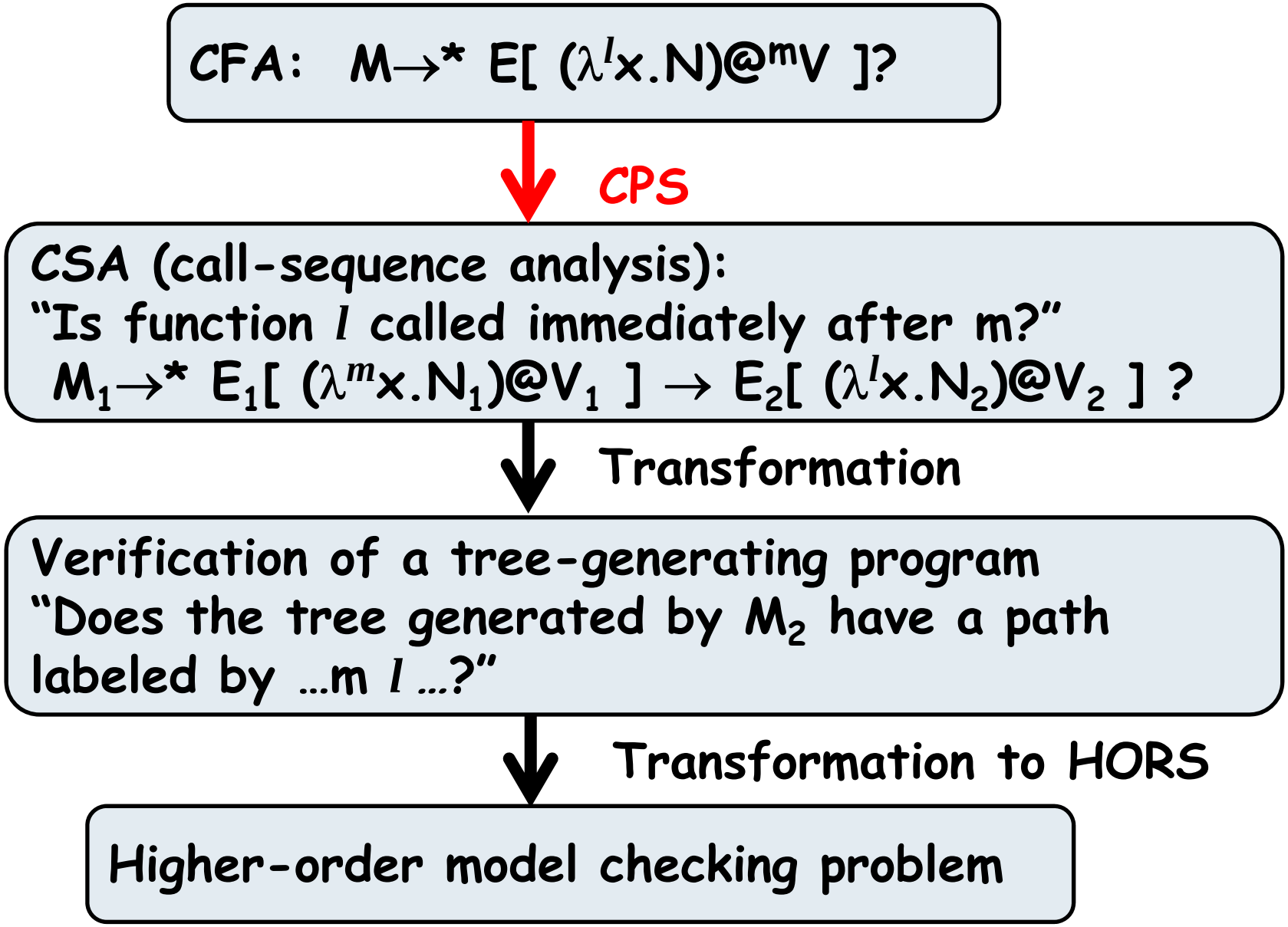
Evaluation rules:

$E[(\lambda^l x.M)@^m V] \rightarrow E[\ [V/x]M\ ]$

From CFA to HO model checking



From CFA to HO model checking



From CFA to CSA

Call-by-value CPS

$$[x] = \lambda k.k\ x$$
$$[\lambda^l x,M] = \lambda k.k\ \lambda^l x.[M]$$
$$[M@^m N] = \lambda k.[M](\lambda f.[N](\lambda^m y.f@y@k))$$

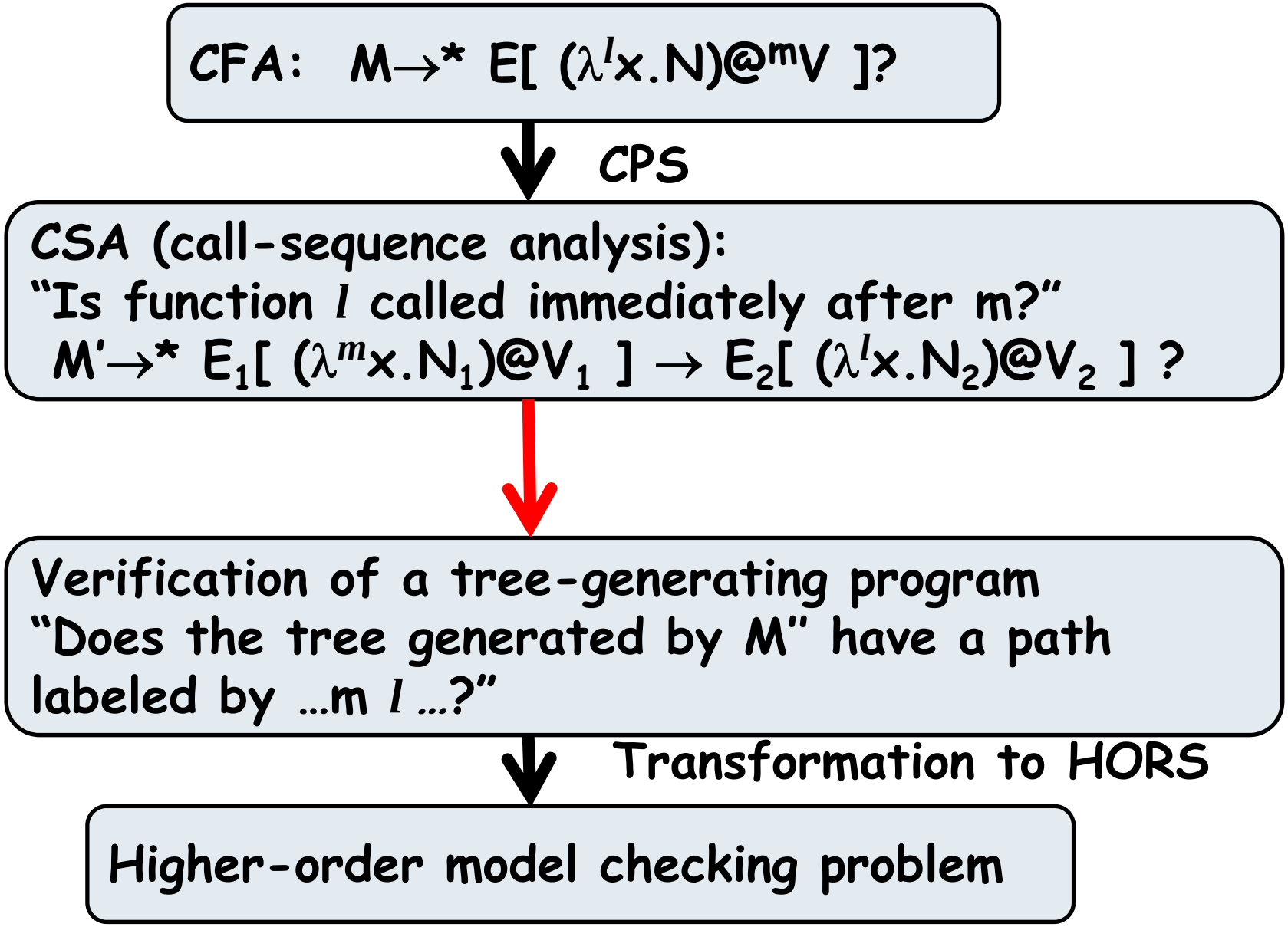
Before CPS

$$M$$
$$\rightarrow^* E[M_1@^m M_2]$$
$$\rightarrow^* E[(\lambda^l x.N)@^m M_2]$$
$$\rightarrow^* E[(\lambda^l x.N)@^m V]$$

After CPS

$$[M]\lambda x.x$$
$$\rightarrow^* [M_1@^m M_2]K$$
$$\rightarrow [M_1]@(\lambda f.[M_2](\lambda^m y.f@y@K))$$
$$\rightarrow^* [M_2]@(\lambda^m y.(\lambda^l x.[N])@y@K))$$
$$\rightarrow^* (\lambda^m y.(\lambda^l x.[N])@y@K))@[V]$$
$$\rightarrow (\lambda^l x.[N])@[V]@K$$

From CFA to HO model checking



From CSA (for CPS programs) to analysis of tree-generating program

Transformation of types:

$\langle Ans \rangle$ = o (tree type) $\quad \langle \tau_1 \to \tau_2 \rangle = \langle \tau_1 \rangle \to \langle \tau_2 \rangle$ ...

Transformation of terms:

$\langle \lambda^m x.M \rangle = \lambda x.m(\langle M \rangle)$ (for continuation)

$\langle \lambda^l x.\lambda k.M \rangle = \lambda x.\lambda k.l(\langle M \rangle)$ (for user function)

$\langle M_1 M_2 \rangle = \langle M_1 \rangle \langle M_2 \rangle$ ...

Before transformation

M

$\to^* (\lambda^m x.N_1)@V_1$

$\to (\lambda^l x.\lambda k.N_3)@V_2@K$

After transformation

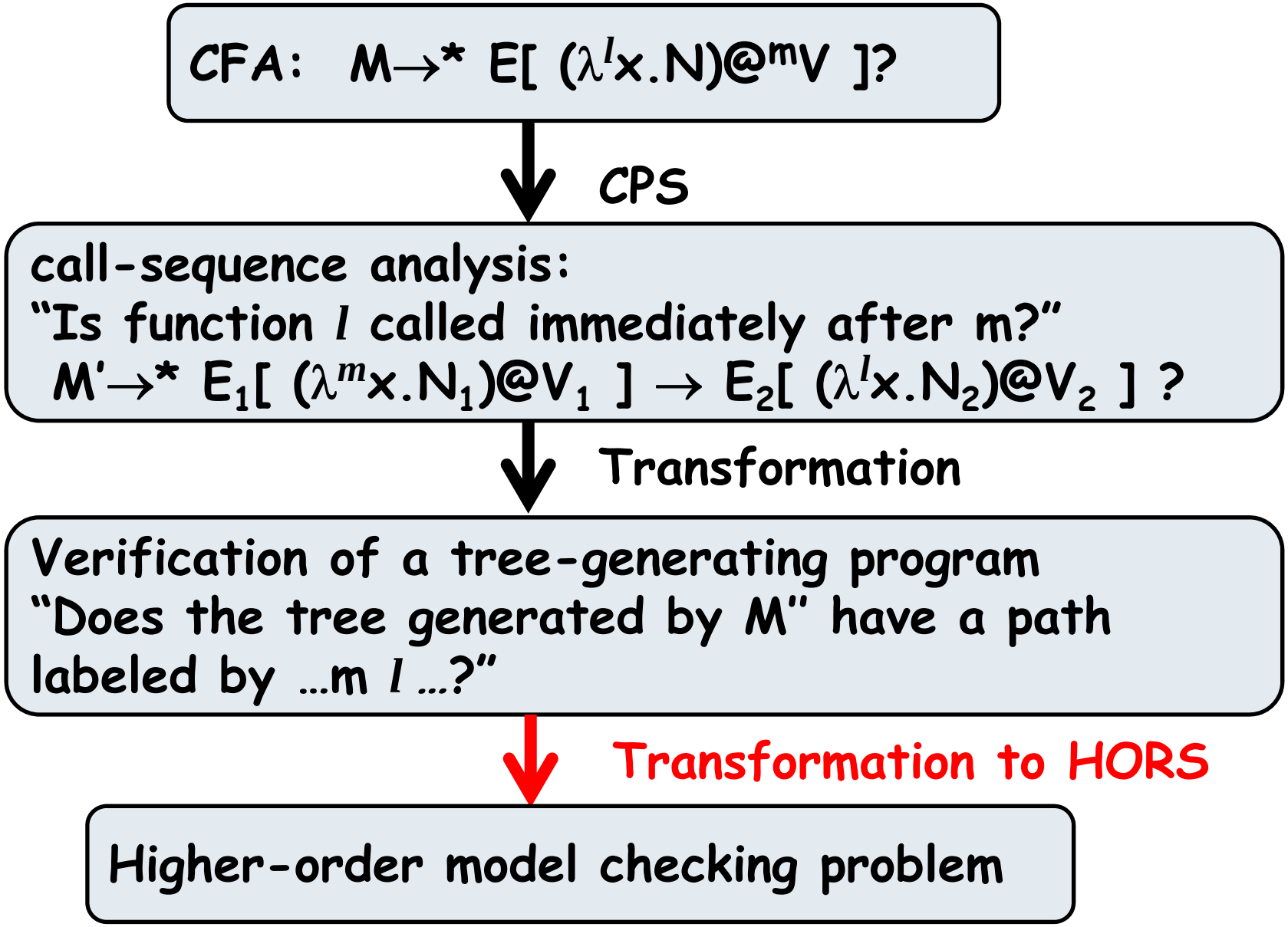
$\langle M \rangle$

$\to^* C[(\lambda x.m(\langle N_1 \rangle))@\langle V_1 \rangle]$

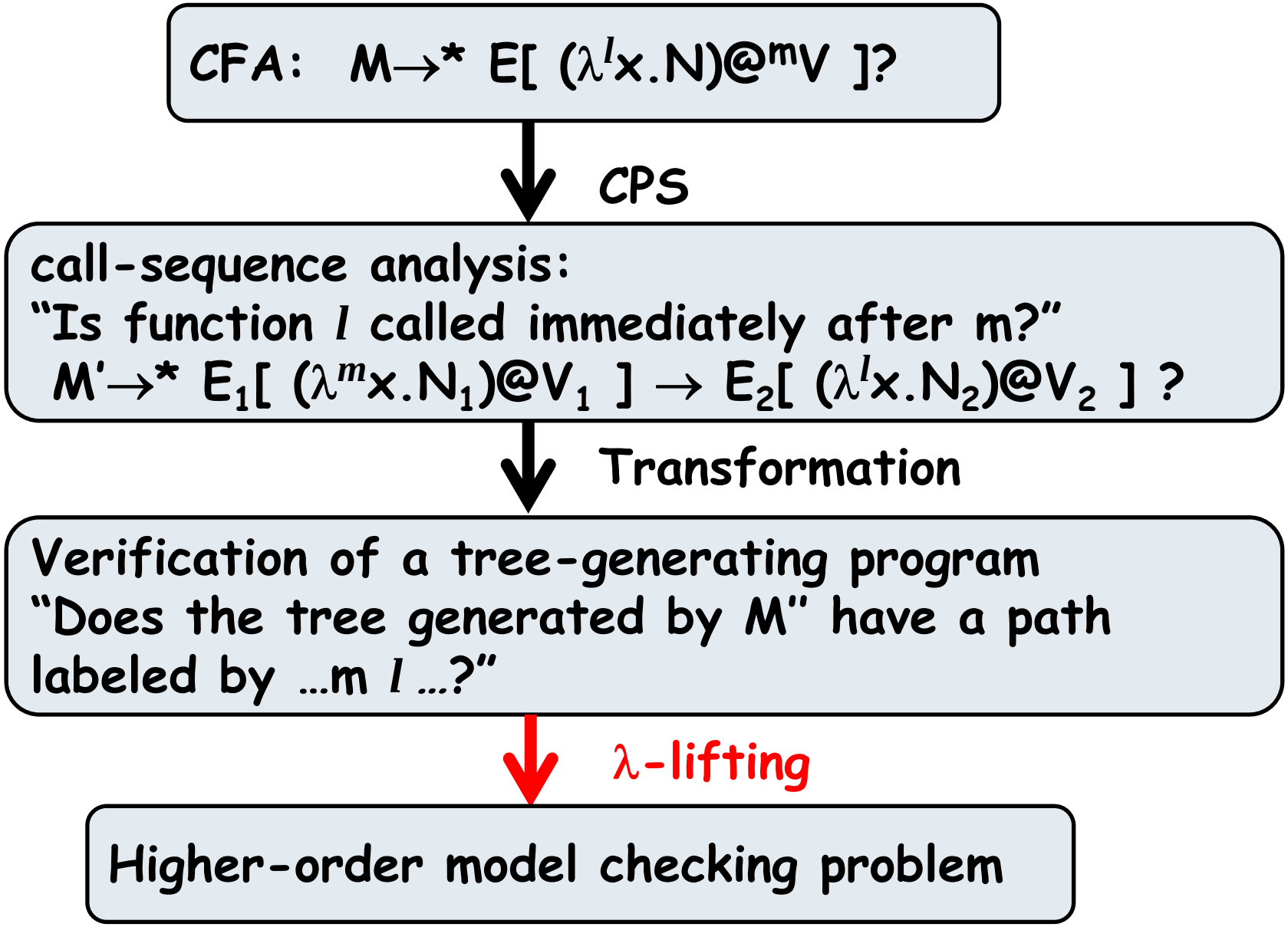
$\to C[m((\lambda x.\lambda k.l(\langle N_3 \rangle))@\langle V_2 \rangle@\langle K \rangle)]$

$\to^* C[m(l(...))]$

From CFA to HO model checking



From CFA to HO model checking



Example

$(\lambda^1 x.x@^2())\ @^3\ (\lambda^4 y.())$

$\Downarrow$ CPS

$(\lambda^3 u.(\lambda^1 x.\lambda k.(\lambda^2 v.x\ v\ k)())\ u\ K))\ (\lambda^4 y.\lambda k.k())$

$\Downarrow$ conversion to tree-generating program

$(\lambda u.3(\lambda x.\lambda k.1(\lambda v.2(x\ v\ k))())\ u\ K))\ (\lambda y.\lambda k.4(k()))$

$\rightarrow 3(\lambda x.\lambda k.1(\lambda v.2(x\ v\ k))())\ (\lambda y.\lambda k.4(k()))\ K)$

$\rightarrow^* 3(1(\lambda v.2((\lambda y.\lambda k.4(k()))\ v\ K))()))$

$\rightarrow\ \ 3(1(2((\lambda y.\lambda k.4(k()))\ ()\ K)))$

$\rightarrow^* 3(1(2(4(K()))))$

Discussion

- Cons
  - Too slow compared with 0CFA
  - Can handle only simply (or intersection) typed, purely functional programs with recursion
- Pros
  - \+ Exact for finitary PCF
  - \+ Runnable (c.f. Mossin's exact flow analysis)
  - \+ Linear time in program size for each flow query if the type size is fixed
    (cubic time for all flow information)
    (c.f. k-CFA)

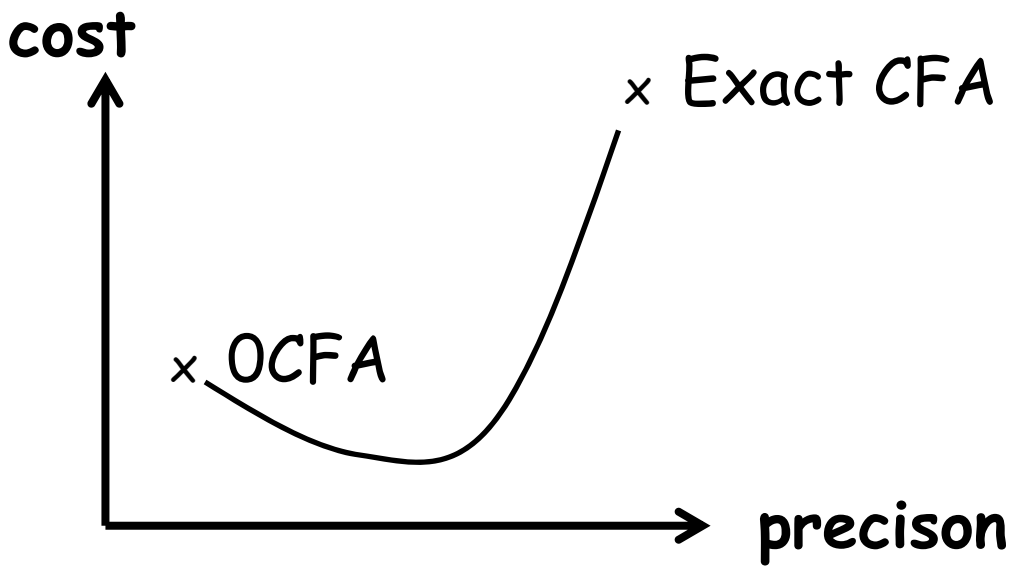
Relevant in practice?

- Useful for analyzing critical flow?
- Useful for evaluating precision of other (non-exact) CFAs and comparing them
- Can be made efficient by some restrictions?
  - Limit the nesting of intersection types
  - Limit the width of intersection types

  $\Rightarrow$ new hierarchies of CFA's?

Open Questions

- Which flow analysis has the best balance between precision and cost?
  - Precise analysis can often be faster than imprecise one



Open Questions

- Which flow analysis has the best balance between precision and cost?
  - Precise analysis can often be faster than imprecise one
- Relationship to CFA2 [Vardoulakis&Shivers] ?
  - CFA2 models programs as PDS
  - HO model checking is equivalent to model checking of HO (collapsible) PDS

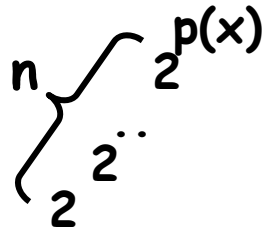
Higher-order extension of CFA2?

Outline

- Brief Review of HO model checking problem
- Application to CFA
- Algorithms for higher-order model checking
  - From model checking to type checking
  - Practical algorithms

Difficulty of higher-order model checking

- Extremely high worst-case complexity
  - n-EXPTIME complete [Ong, LICS06]



  - Earlier algorithms [Ong06;Aehlig06;Hague et al.08] almost always suffer from n-EXPTIME bottleneck.

Our approach: from model checking to typing

Construct a type system TS(*A*) s.t.

Tree(*G*) is accepted by tree automaton *A*

if and only if

G is typable in TS(*A*)

Model Checking as Type Checking
(c.f. [Naik & Palsberg, ESOP2005])

Model Checking Problem

Given
- G: higher-order recursion scheme (without safety restriction)
- A: alternating parity tree automaton (APT) (a formula of modal μ-calculus or MSO),

does A accept Tree(G)?

n-EXPTIME-complete [Ong, LICS06]
(for order-n recursion scheme)

Model Checking Problem: Restricted version

Given

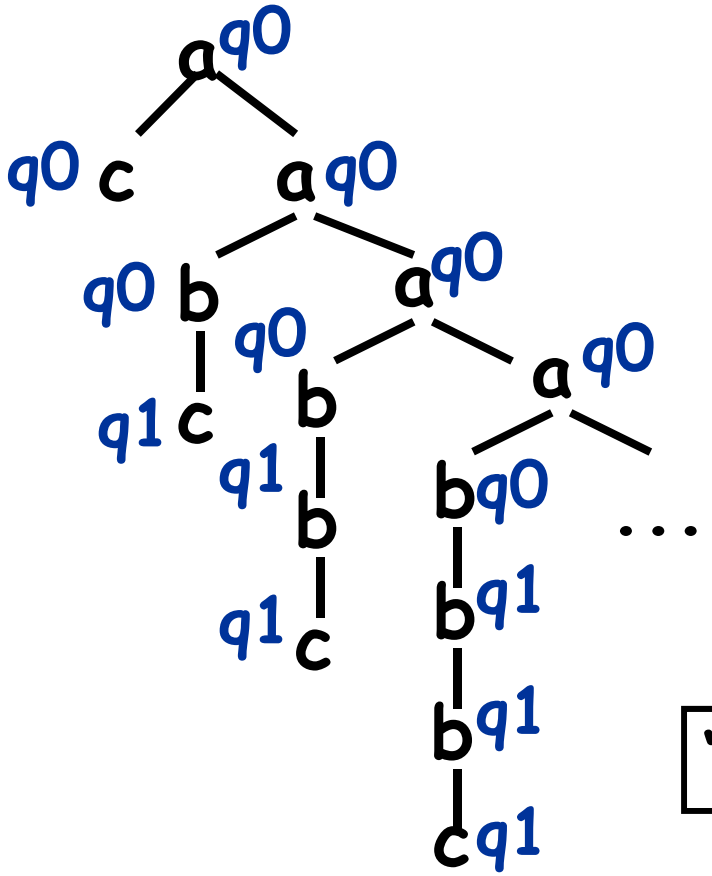
G: higher-order recursion scheme
(without safety restriction)

A: trivial automaton [Aehlig CSL06]
(Büchi tree automaton where
all the states are accepting states)

does A accept Tree(G)?

See [K.&Ong, LICS09] for the general case
(full modal μ-calculus model checking)

Trivial tree automaton for infinite trees

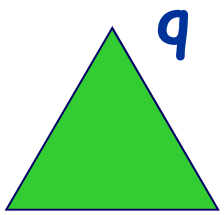


$\delta(q0, a) = q0\ q0$
$\delta(q0, b) = q1$
$\delta(q1, b) = q1$
$\delta(q0, c) = \varepsilon$
$\delta(q1, c) = \varepsilon$

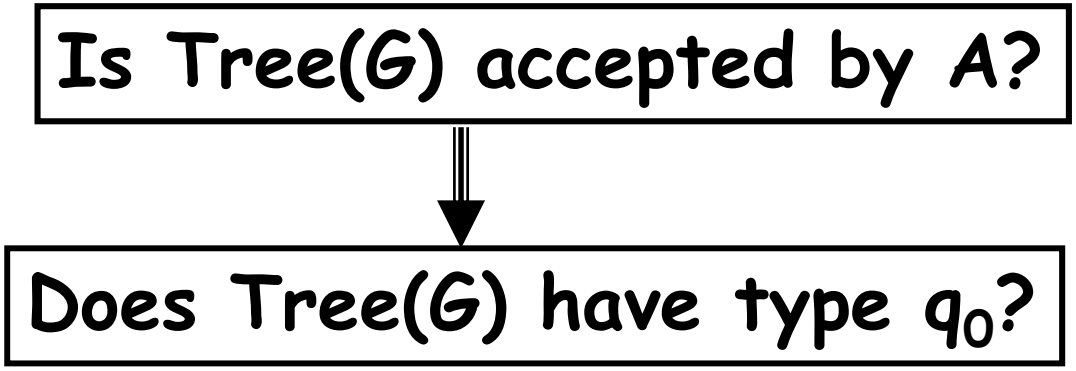
"a" does not occur below "b"

Types for Recursion Schemes

- Automaton state as the type of trees
  - q: trees accepted from state q



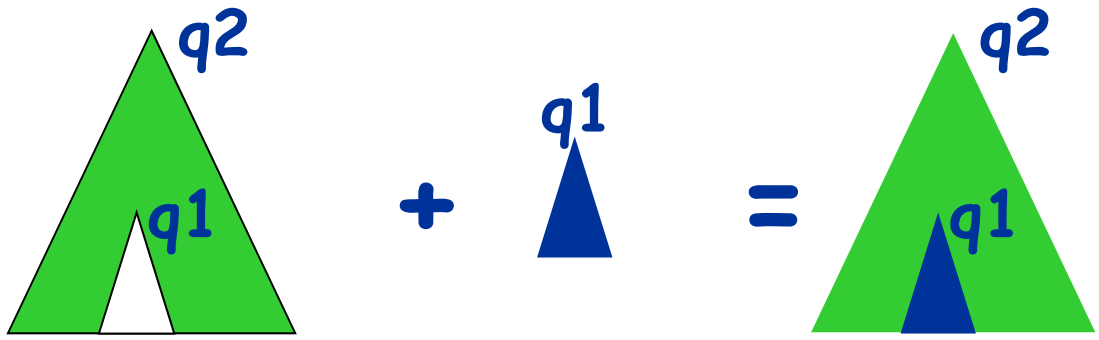
  - $q1 \wedge q2$: trees accepted from both q1 and q2



Types for Recursion Schemes

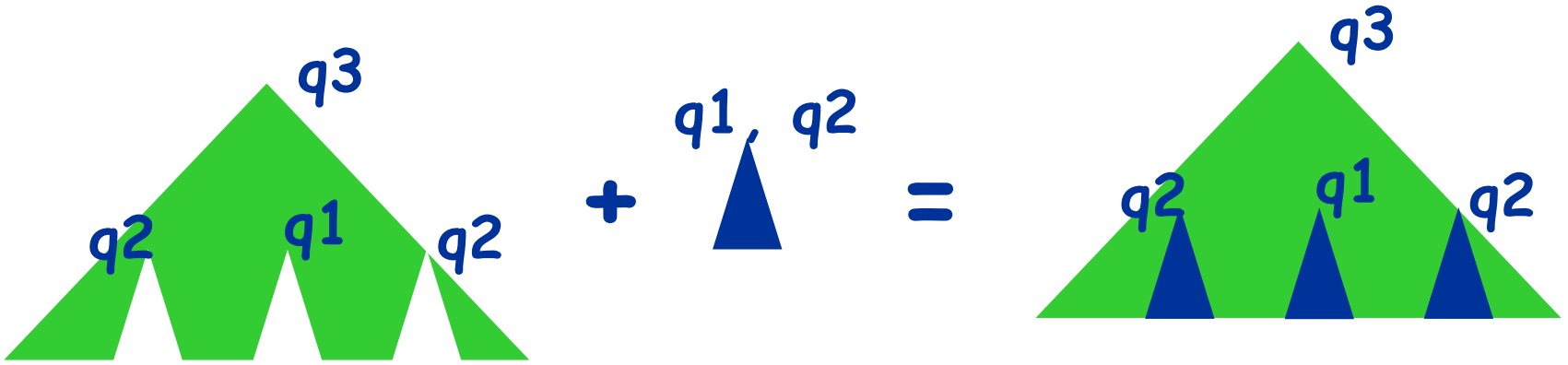
- Automaton state as the type of trees

  - $q1 \to q2$: functions that take a tree of type q1 and return a tree of q2



Types for Recursion Schemes

- Automaton state as the type of trees
  - $q1 \wedge q2 \to q3$:
    functions that take a tree of type $q1 \wedge q2$ and return a tree of type $q3$

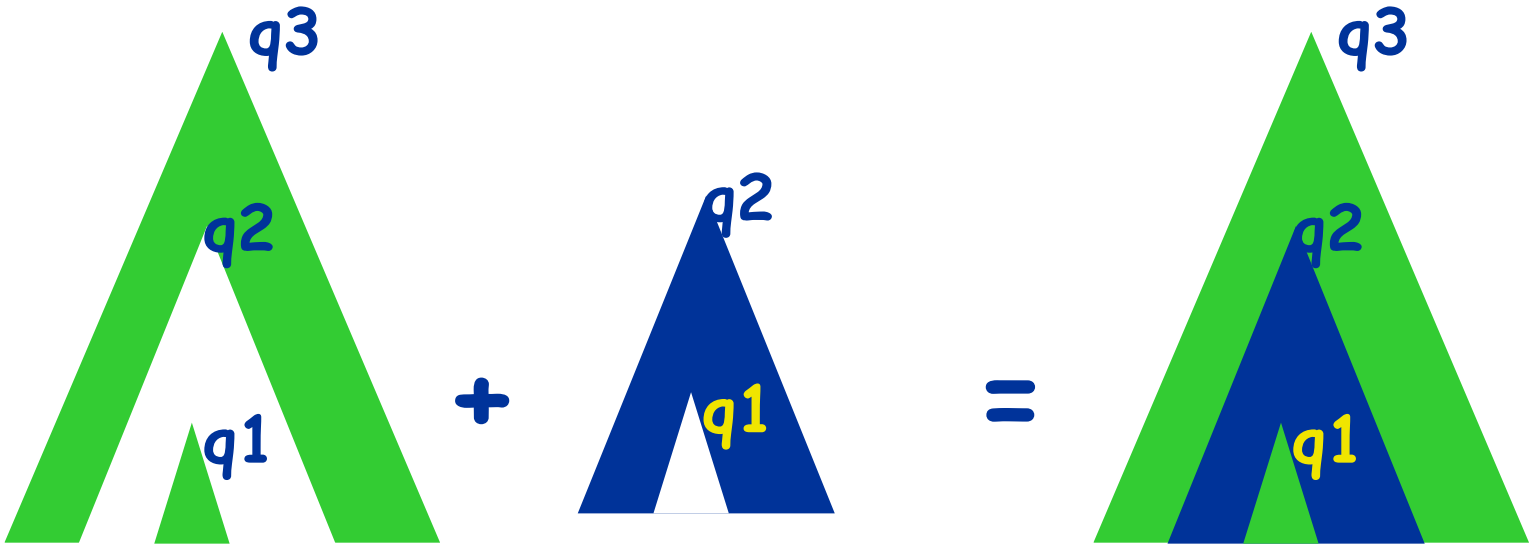


Types for Recursion Schemes

- Automaton state as the type of trees

  $(q1 \to q2) \to q3$:

  functions that take a function of type $q1 \to q2$ and return a tree of type $q3$



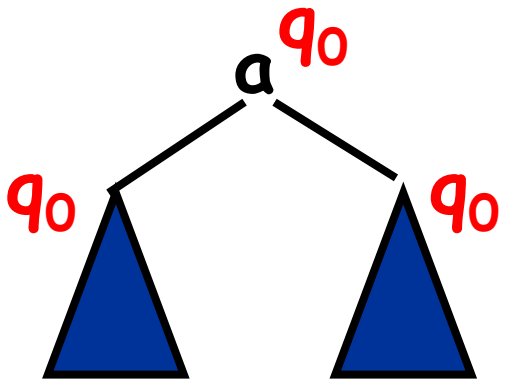
Example

Automaton:

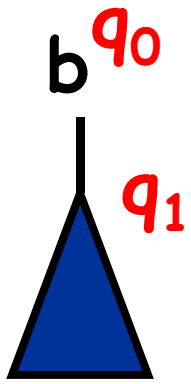
$\delta(q_0, a) = q_0\ q_0 \qquad \delta(q_0, b) = \delta(q_1, b) = q_1$

$\delta(q_0, c) = \delta(q_1, c) = \varepsilon$

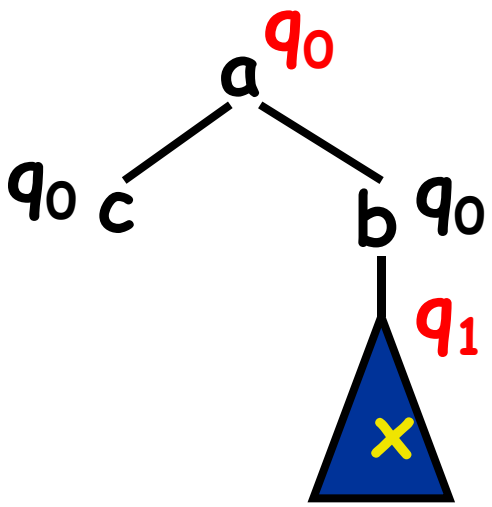
a: $q_0 \rightarrow q_0 \rightarrow q_0$



b: $q_1 \rightarrow q_0$

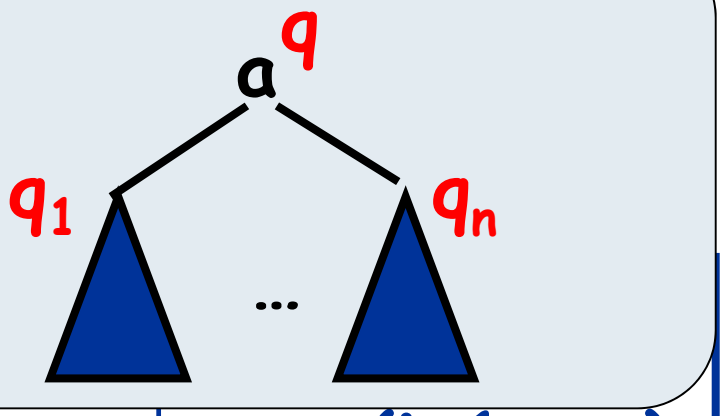


$\lambda x.a\ c\ (b\ x)$: $q_1 \rightarrow q_0$



Typing

$$\frac{\delta(q,\ a) = q_1 \ldots q_n}{\vdash a : q_1 \to \ldots \to q_n \to q}$$



$$\frac{\Gamma, x{:}\tau_1, \ldots, x{:}\tau_n \vdash t{:}\tau}{\Gamma \vdash \lambda x.t{:}\ \tau_1 \wedge \ldots \wedge \tau_n \to \tau}$$

$$\frac{\Gamma \vdash t_2{:}\tau_i \ (i=1,..n)}{\Gamma \vdash t_1\ t_2{:}\tau}$$

$$\frac{\Gamma \vdash t_k : \tau \ (\text{for every } F_k{:}\tau \in \Gamma)}{\vdash \{F_1 \to t_1, \ldots, F_n \to t_n\} : \Gamma}$$

Typing

$$\frac{\delta(q,\ a) = q_1 \ldots q_n}{\vdash a : q_1 \to \ldots \to q_n \to q}$$

$$\Gamma, x{:}\tau \ \vdash x : \tau$$

$$\frac{\Gamma, x{:}\tau_1, \ldots, x{:}\tau_n \ \vdash t{:}\tau}{\Gamma \vdash \lambda x.t{:}\ \tau_1 \wedge \ldots \wedge \tau_n \to \tau}$$

$$\frac{\Gamma \vdash t_1{:}\ \tau_1 \wedge \ldots \wedge \tau_n \to \tau \qquad \Gamma \vdash t_2{:}\tau_i \ (i=1,..n)}{\Gamma \vdash t_1\ t_2{:}\tau}$$

$$\frac{\Gamma \vdash t_k : \tau \ (\text{for every } F_k{:}\tau \in \Gamma)}{\vdash \{F_1 \to t_1, \ldots, F_n \to t_n\} : \Gamma}$$

Soundness and Completeness

[K., POPL2009]

Tree(G) is accepted by A
if and only if

S has type q_0 in TS(A),

i.e. $\exists\Gamma.(S{:}q_0 \in \Gamma \wedge \vdash \{F_1 \to t_1, \ldots, F_n \to t_n\} : \Gamma)$

if and only if

$\exists\Gamma.(S{:}\ q_0 \in \Gamma \wedge \forall(F_k{:}\tau)\in\Gamma.\ \Gamma \vdash t_k : \tau\ )$

$G = \{F_1 \to t_1, \ldots, F_m \to t_m\}$ (with $S=F_1$)

A: Trivial automaton with initial state q_0

TS(A): Intersection type system for A

Soundness and Completeness

[K., POPL2009]

Tree(G) is accepted by A
if and only if

S has type q_0 in TS(A),

i.e. $\exists\Gamma.(S{:}q_0 \in \Gamma \wedge \vdash \{F_1 \to t_1, \ldots, F_n \to t_n\} : \Gamma)$
if and only if

$\exists\Gamma.(S{:}\ q_0 \in \Gamma \wedge \forall(F_k{:}\tau)\in\Gamma.\ \Gamma \vdash t_k : \tau\ )$
if and only if

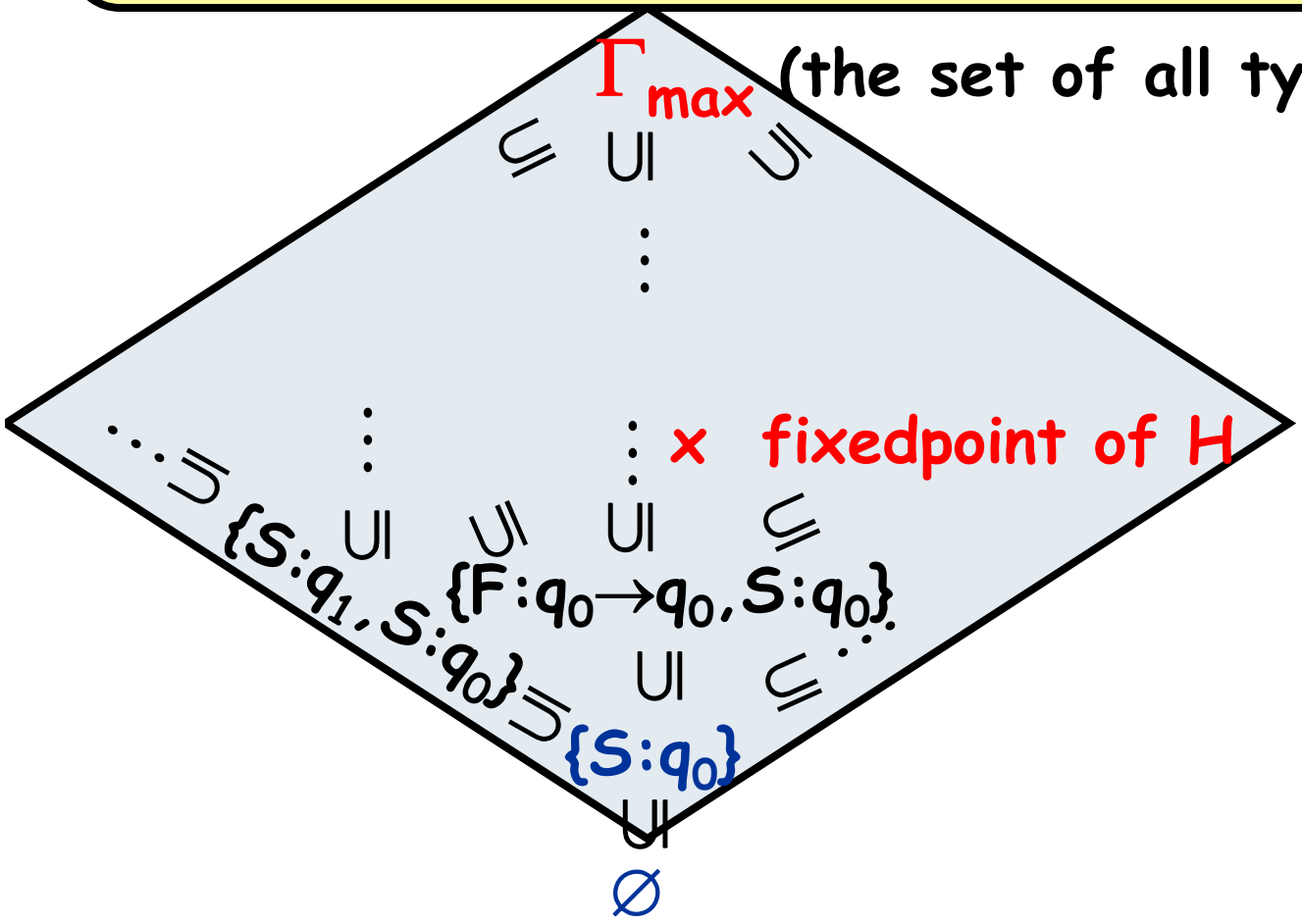
$\exists\Gamma.(S{:}\ q_0 \in \Gamma \wedge \Gamma = H(\Gamma)\ )$

for $H(\Gamma) = \{\ F_k{:}\tau \in \Gamma \mid \Gamma \vdash t_k{:}\tau\ \}$

Function to filter out invalid type bindings

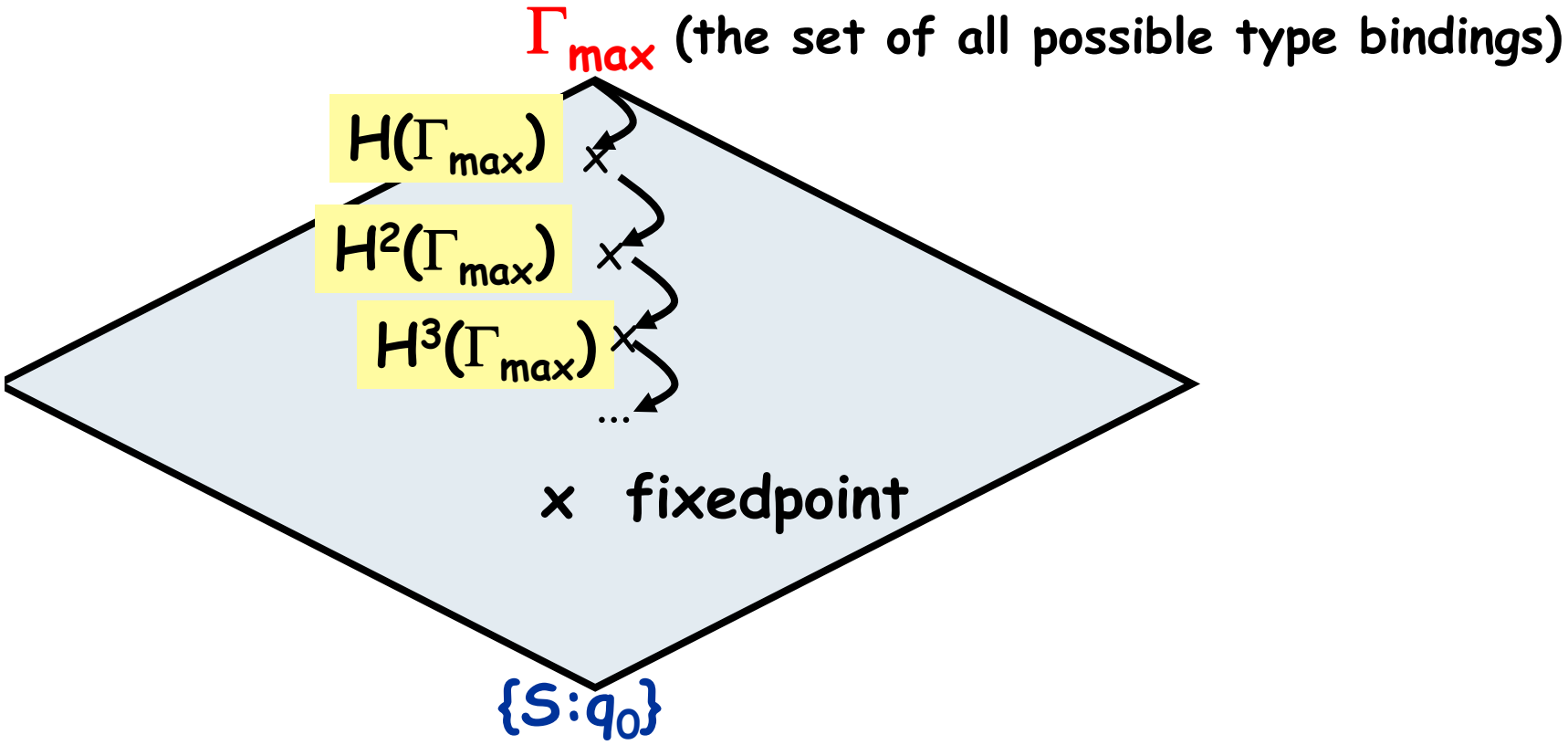
Type checking (=model checking) problem

Is there a fixedpoint of H greater than $\{S:q_0\}$?
(where $H(\Gamma) = \{ F_j:\tau \in \Gamma \mid \Gamma \vdash t_j:\tau \}$)



Naive Algorithm [K. POPL09]

1. Compute the greatest fixedpoint Γ_{gfp} of H
   $(H(\Gamma) = \{ F_j:\tau \in \Gamma \mid \Gamma \mid\!- t_j:\tau \})$
2. Check whether $S:q_0 \in \Gamma_{gfp}$



Example

- **Recursion scheme:**

  $S \to F\ c \qquad F \to \lambda x.a\ x\ (F\ (b\ x))$

  $(S{:}o,\ F{:}\ o \to o)$

- **Automaton:**

  $\delta(q_0,\ a) = q_0\ q_0 \quad \delta(q_0,\ b) = q_1 \quad \delta(q_1,\ b) = q_1$

  $\delta(q_0,\ c) = \delta(q_1,\ c) = \varepsilon$

$\Gamma_{max} = \{S{:}q_0,\ S{:}q_1,\ F{:}\ T \to q_0,\ F{:}\ q_0 \to q_0,\ F{:}\ q_1 \to q_0,\ F{:}\ q_0 \wedge q_1 \to q_0,$
$F{:}\ T \to q_1,\ F{:}\ q_0 \to q_1,\ F{:}\ q_1 \to q_1,\ F{:}\ q_0 \wedge q_1 \to q_1\}$

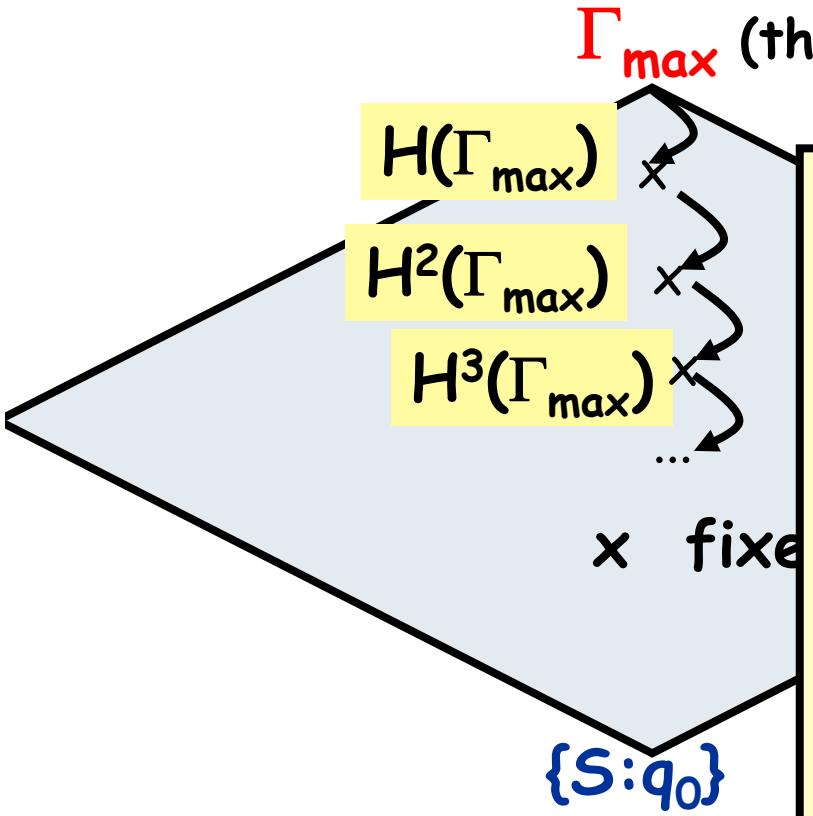
$H(\Gamma_{max}) = \{\ S{:}\tau \in \Gamma_{max} \mid \Gamma_{max} \vdash F\ c{:}\tau\ \}$
$\cup\ \{\ F{:}\tau \in \Gamma_{max} \mid \Gamma_{max} \vdash \lambda x.a\ x\ (F(b\ x))\ {:}\tau\ \}$
$= \{S{:}q_0,\ S{:}q_1,\ F{:}\ q_0 \to q_0,\ F{:}\ q_0 \wedge q_1 \to q_0\}$

$H^2(\Gamma_{max}) = \{S{:}q_0,\ F{:}\ q_0 \wedge q_1 \to q_0\}$

$H^3(\Gamma_{max}) = \{S{:}q_0,\ F{:}\ q_0 \wedge q_1 \to q_0\} = H^2(\Gamma_{max})$

Naive Algorithm [K. POPL09]

1. Compute the greatest fixedpoint Γ_{gfp} of H
   $(H(\Gamma) = \{ F_j:\tau \in \Gamma \mid \Gamma \mid\!- t_j:\tau \})$
2. Check whether $S:q_0 \in \Gamma_{gfp}$



Drawbacks:
- Huge cost for computing H
- Huge number of iterations

(both as huge as $|\Gamma_{max}| = O(|G| \times n \, 2^{2^{\cdot^{\cdot^{2}}}}{}^{(AQ)^{1+\varepsilon}})$)

A: largest arity
Q: automaton size

How large is Γ_{max}?

Γ_{max}: the set of all possible type bindings for non-terminals

sort	# of types for each sort ($Q=\{q_0,q_1,q_2,q_3\}$)
o (trees)	4 (q_0,q_1,q_2,q_3)
$o \to o$	$2^4 \times 4 = 64$ $(\wedge S \to q$, with $S \in 2^Q$, $q \in Q)$
$(o \to o) \to o$	$2^{64} \times 4 = 2^{66}$
$((o \to o) \to o) \to o$	$2^{2^{66}} \times 4 > 10^{10000000000000000000}$

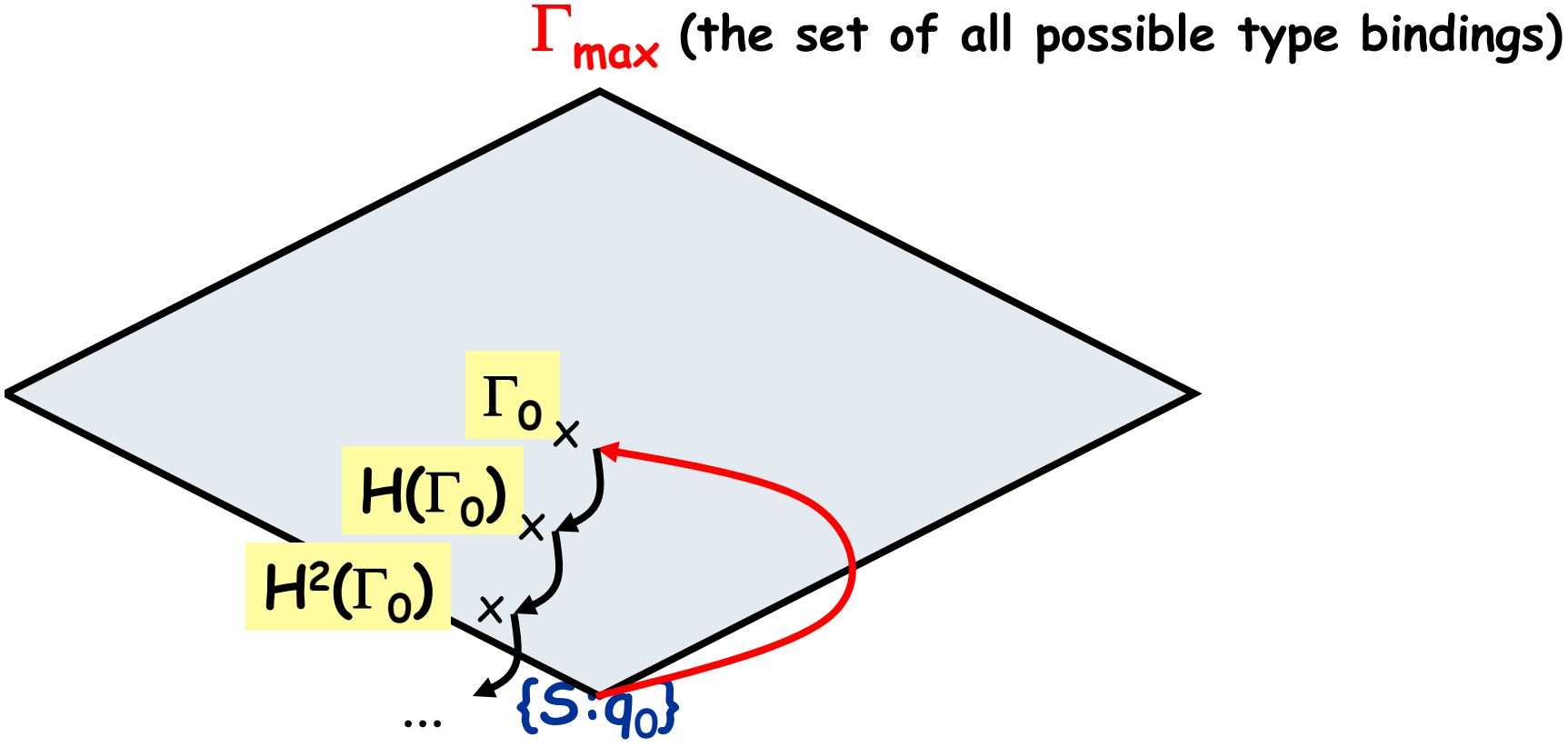
$$|\Gamma_{max}| = O(|G| \times \underbrace{2^{2^{\cdot^{\cdot^{2}}}}}_{n}{}^{(A|Q|)^{1+\varepsilon}})$$

Outline

- Brief Review of HO model checking problem
- Application to CFA
- Algorithms for higher-order model checking
  - From model checking to type checking
  - Practical algorithms

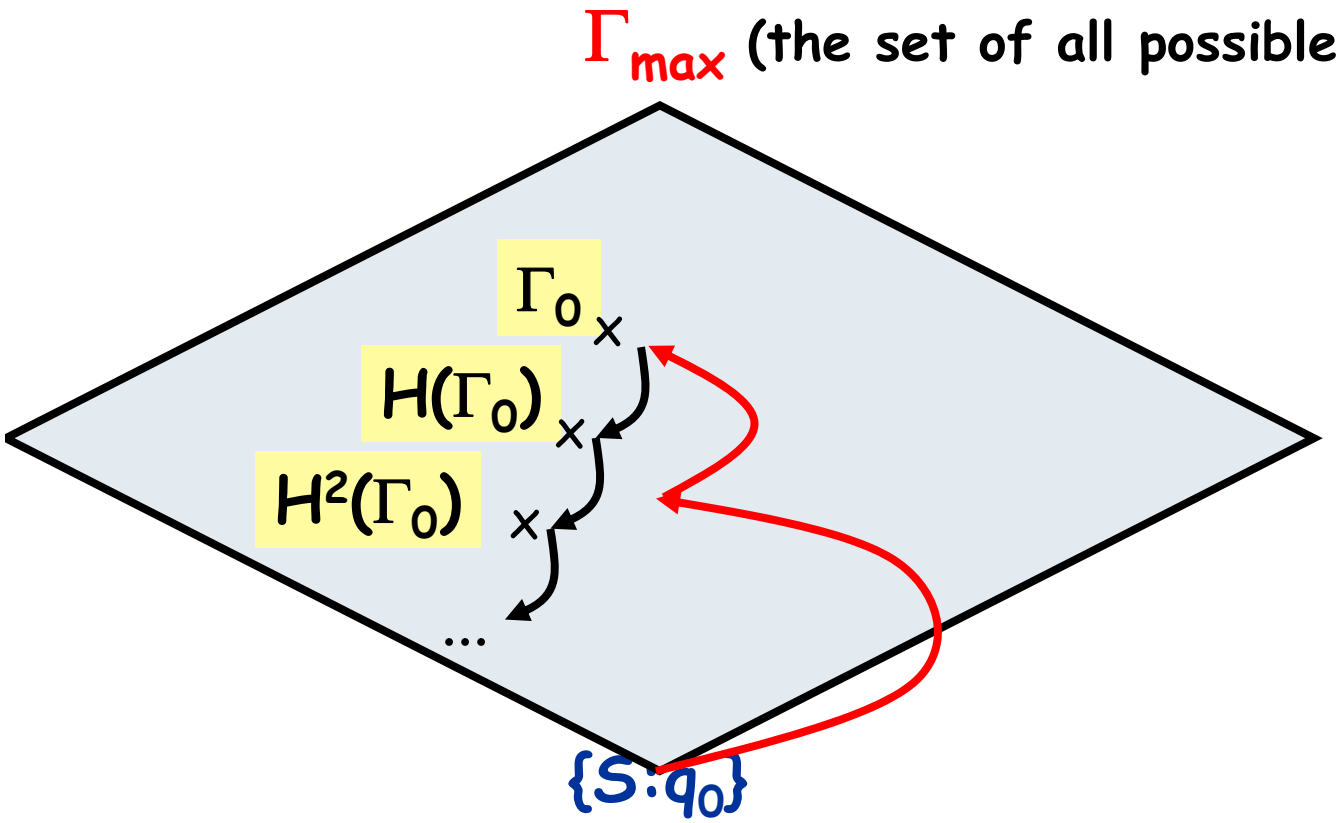
Practical Algorithms [K. PPDP09] [K.FoSSaCS11]

1. Guess a type environment Γ_0
2. Compute greatest fixedpoint Γ smaller than Γ_0
3. Check whether $S{:}q_0 \in \Gamma$
4. Repeat 1-3 until the property is proved or refuted.



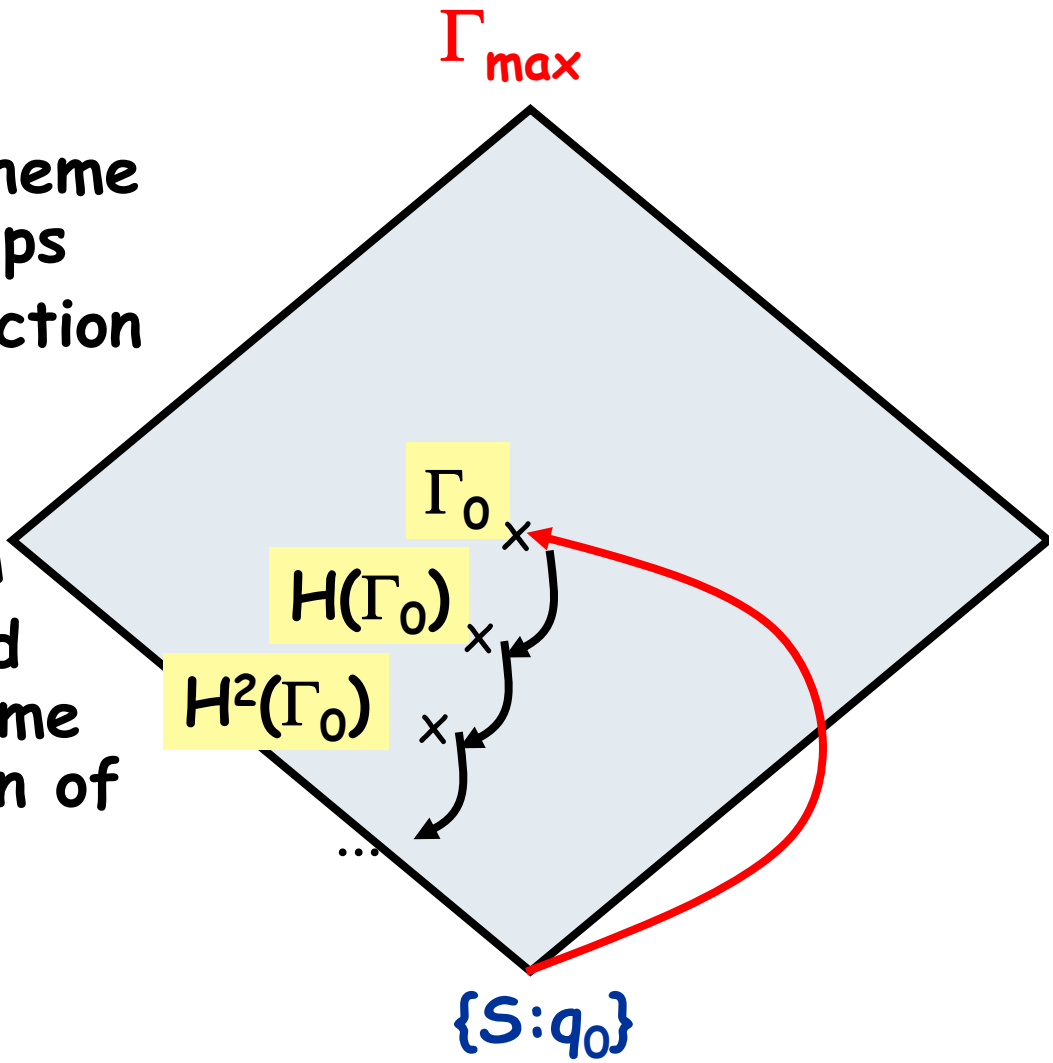
Practical Algorithms [K. PPDP09] [K.FoSSaCS11]

1. Guess a type environment Γ_0
2. Compute greatest fixedpoint Γ smaller than Γ_0
3. Check whether $S{:}q_0 \in \Gamma$
4. Repeat 1-3 until the property is proved or refuted.



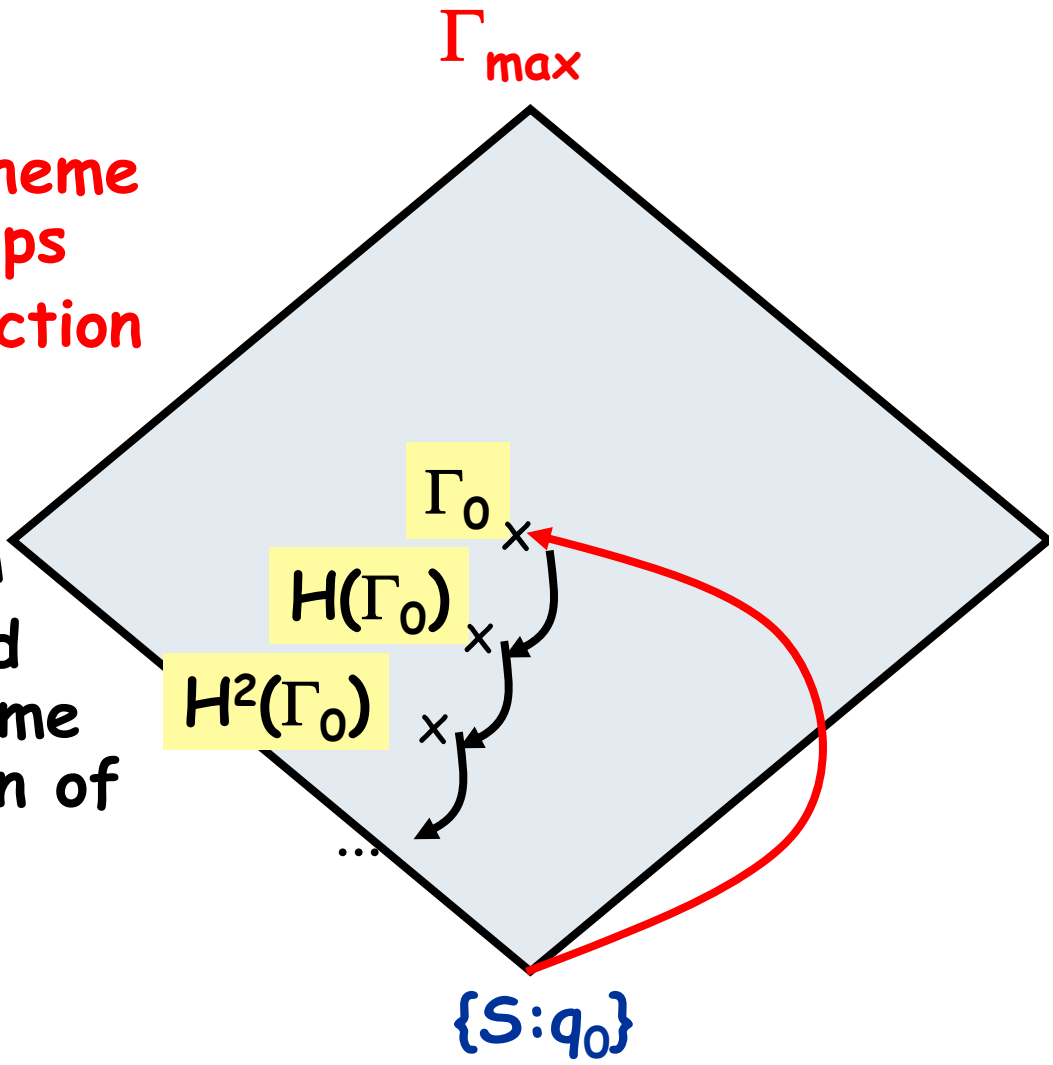
How to guess Γ_0?

- PPDP09 algorithm
  - Reduce a recursion scheme a finite number of steps
  - Observe how each function is used and express it as types
- FoSSaCS11 algorithm
  - Like PPDP09, but avoid reductions by using game semantic interpretation of types



How to guess Γ_0?

- PPDP09 algorithm
  - Reduce a recursion scheme a finite number of steps
  - Observe how each function is used and express it as types
- FoSSaCS11 algorithm
  - Like PPDP09, but avoid reductions by using game semantic interpretation of types



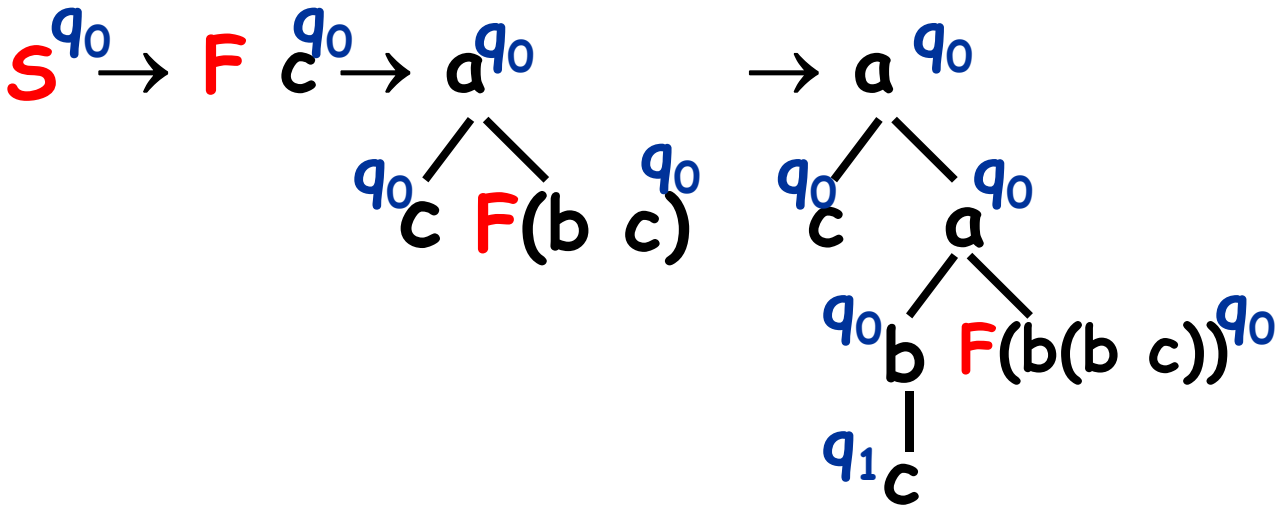
Example

- **Recursion scheme:**

$S \to F\ c \qquad F \to \lambda x.a\ x\ (F\ (b\ x))$

- **Automaton:**

$\delta(q_0, a) = q_0\ q_0 \quad \delta(q_0, b) = q_1 \quad \delta(q_1, b) = q_1$
$\delta(q_0, c) = \delta(q_1, c) = \varepsilon$



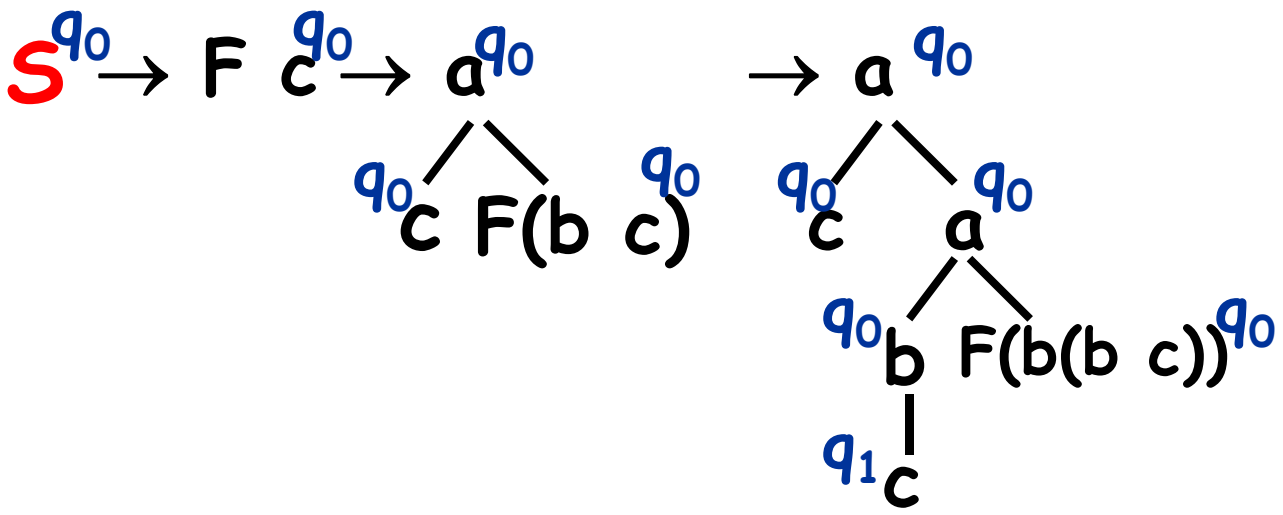
Example

- Recursion scheme:

$S \to F\ c \qquad F \to \lambda x.a\ x\ (F\ (b\ x))$

- Automaton:

$\delta(q_0, a) = q_0\ q_0 \quad \delta(q_0, b) = q_1 \quad \delta(q_1, b) = q_1$
$\delta(q_0, c) = \delta(q_1, c) = \varepsilon$



Γ_0 :

$S: q_0$

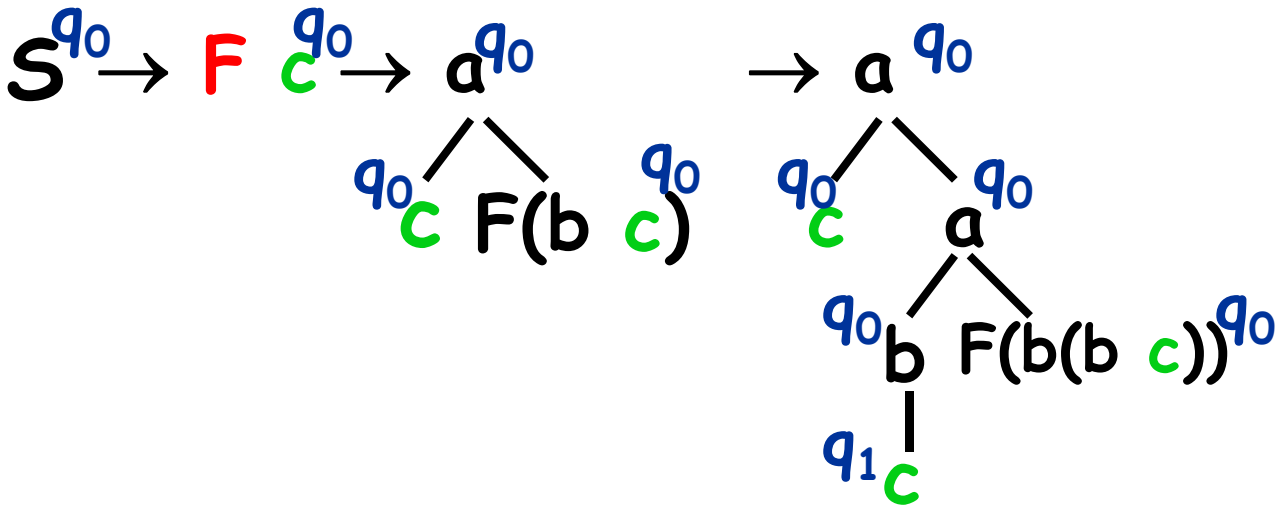
Example

- Recursion scheme:

$S \to F\ c \qquad F \to \lambda x.a\ x\ (F\ (b\ x))$

- Automaton:

$\delta(q_0, a) = q_0\ q_0 \quad \delta(q_0, b) = q_1 \quad \delta(q_1, b) = q_1$
$\delta(q_0, c) = \delta(q_1, c) = \varepsilon$



Γ_0 :

S: q_0

F: ? → q_0

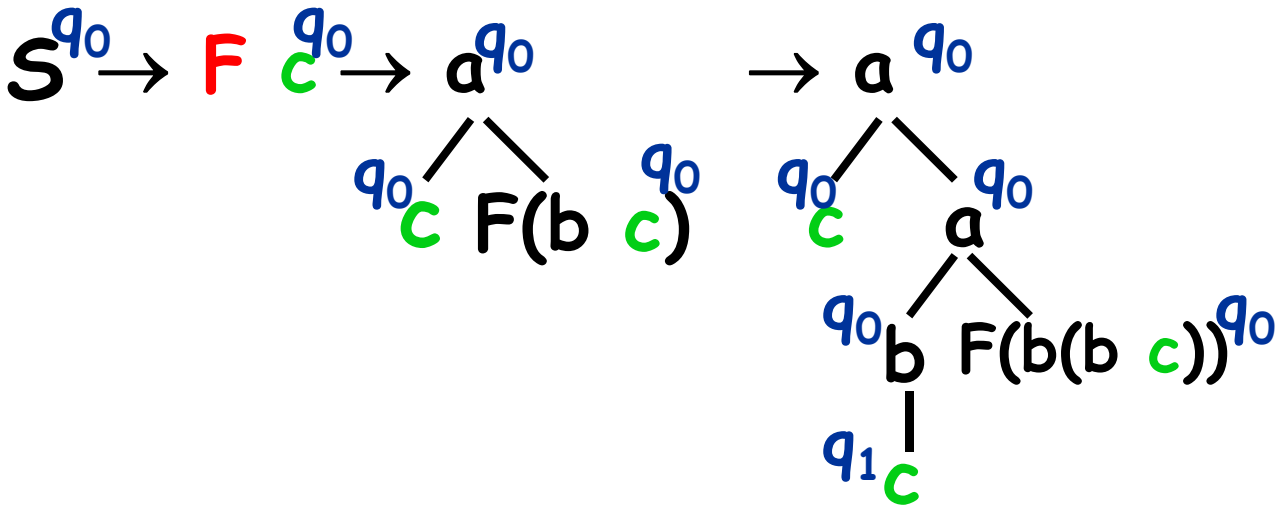
Example

- Recursion scheme:

$S \to F\ c \qquad F \to \lambda x.a\ x\ (F\ (b\ x))$

- Automaton:

$\delta(q_0, a) = q_0\ q_0 \quad \delta(q_0, b) = q_1 \quad \delta(q_1, b) = q_1$
$\delta(q_0, c) = \delta(q_1, c) = \varepsilon$



Γ_0 :

S: q_0

F: $q_0 \wedge q_1 \to q_0$

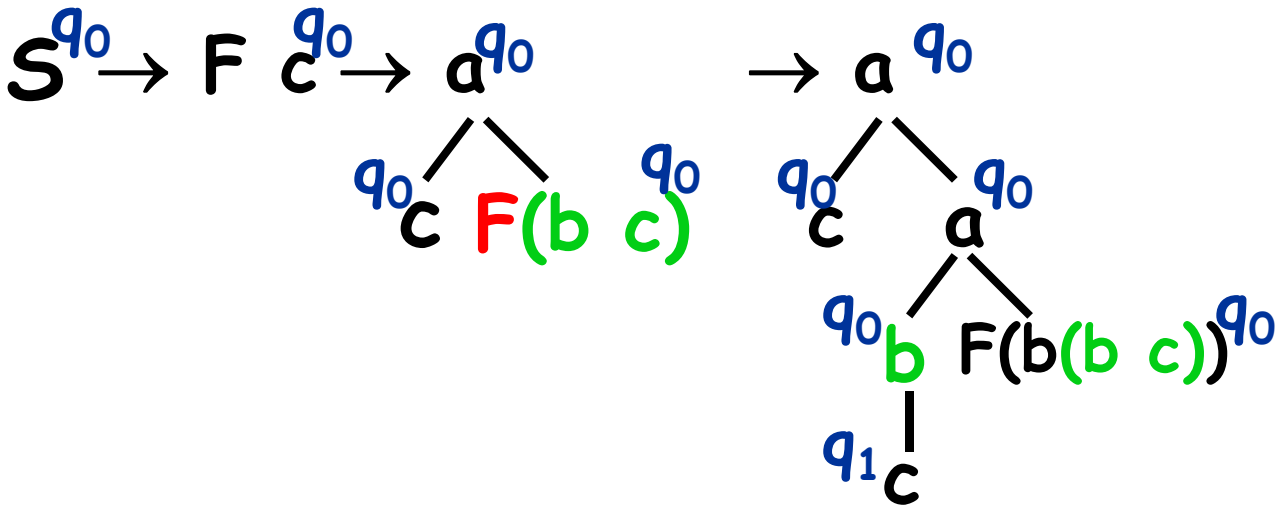
Example

- Recursion scheme:

$S \to F\ c \qquad F \to \lambda x.a\ x\ (F\ (b\ x))$

- Automaton:

$\delta(q_0, a) = q_0\ q_0 \quad \delta(q_0, b) = q_1 \quad \delta(q_1, b) = q_1$
$\delta(q_0, c) = \delta(q_1, c) = \varepsilon$



Γ_0 :

S: q_0

F: $q_0 \wedge q_1 \to q_0$

F: $q_0 \to q_0$

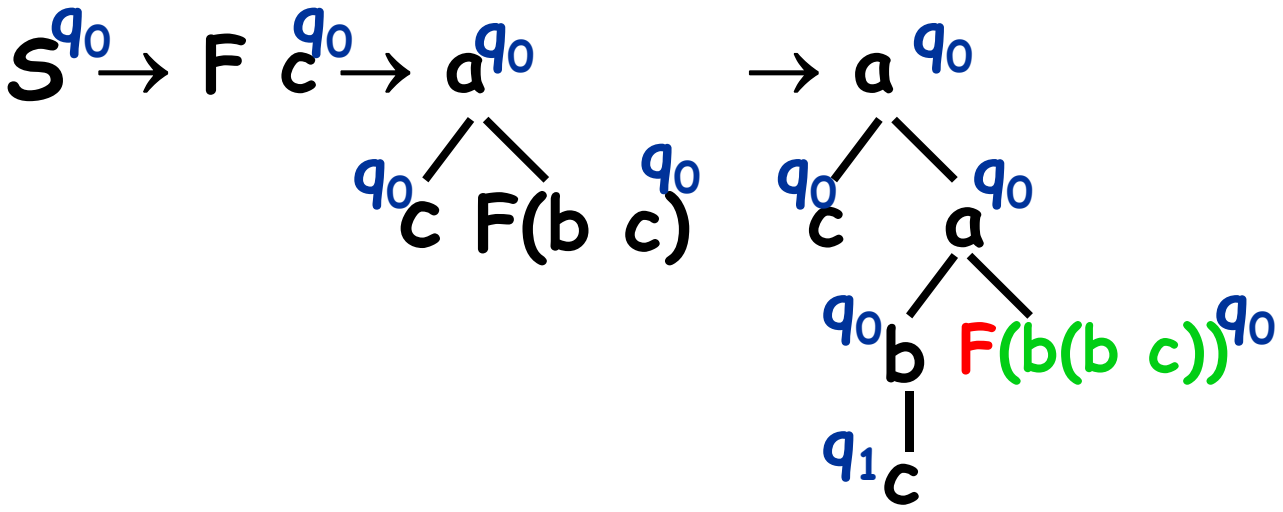
Example

- Recursion scheme:

$S \to F\ c \qquad F \to \lambda x.a\ x\ (F\ (b\ x))$

- Automaton:

$\delta(q_0, a) = q_0\ q_0 \quad \delta(q_0, b) = q_1 \quad \delta(q_1, b) = q_1$
$\delta(q_0, c) = \delta(q_1, c) = \varepsilon$



Γ_0 :

S: q_0

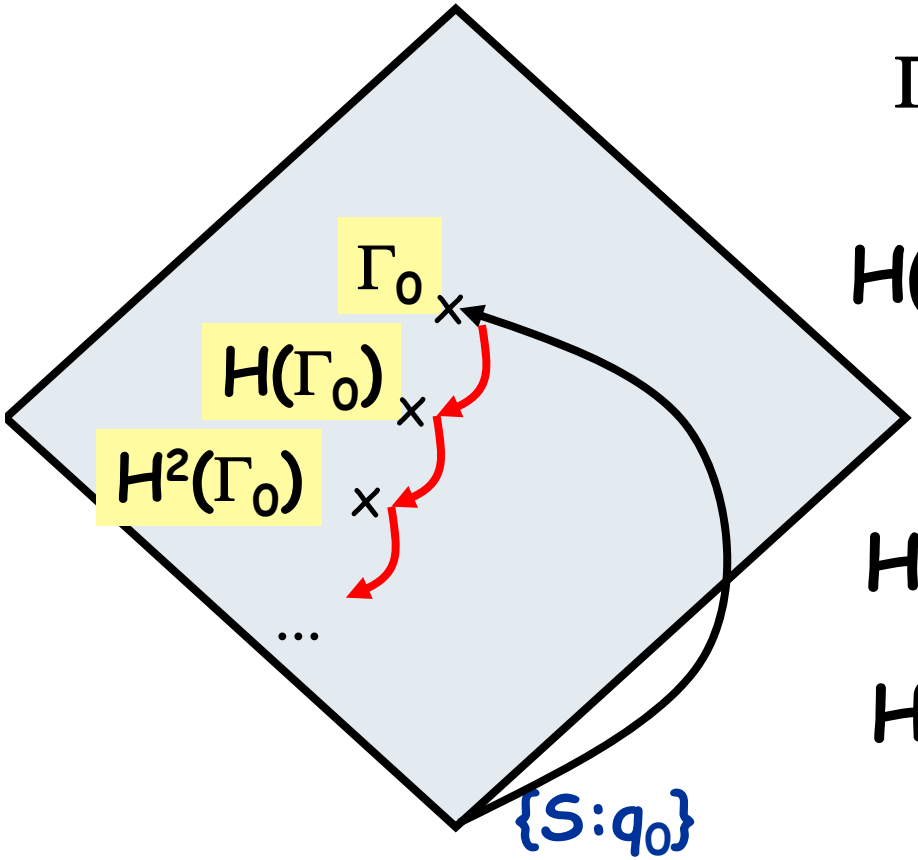
F: $q_0 \wedge q_1 \to q_0$

F: $q_0 \to q_0$

F: $T \to q_0$

Practical Algorithms [K. PPDP09] [K.FoSSaCS11]

1. Guess a type environment Γ_0
2. Compute greatest fixedpoint Γ smaller than Γ_0
3. Check whether $S{:}q_0 \in \Gamma$
4. Repeat 1-3 until the property is proved or refuted.



$$\Gamma_0 = \{S{:}\ q_0,\ F{:}\ q_0 \wedge q_1 \to q_0,\ F{:}\ q_0 \to q_0\ ,\ F{:}\ T \to q_0\}$$

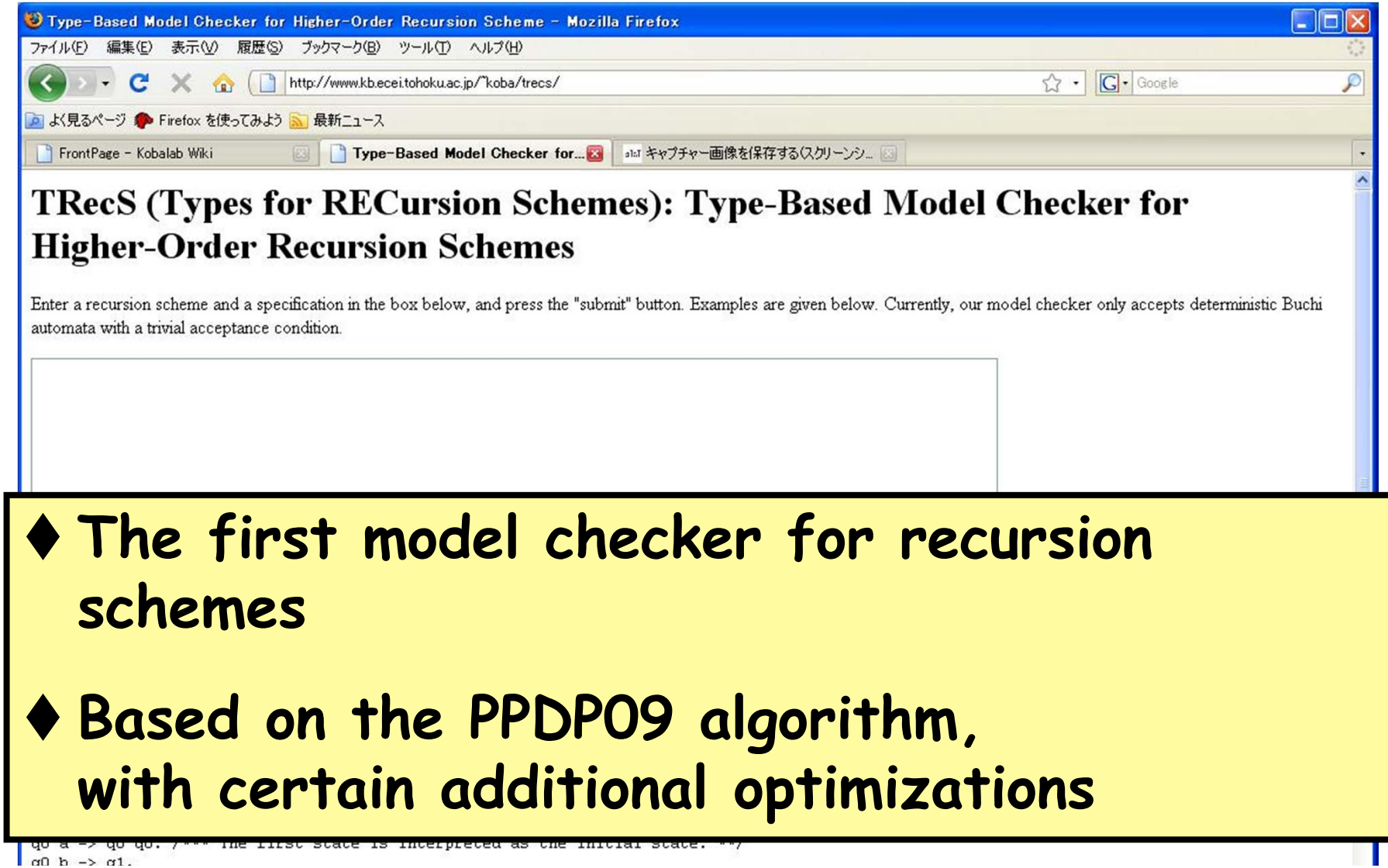
$$H(\Gamma_0) = \{\ F_k{:}\tau \in \Gamma_0 \mid \Gamma_0 \vdash t_k{:}\tau\ \} = \{S{:}\ q_0,\ F{:}\ q_0 \wedge q_1 \to q_0,\ F{:}\ q_0 \to q_0\ \}$$

$$H^2(\Gamma_0) = \{S{:}\ q_0,\ F{:}\ q_0 \wedge q_1 \to q_0\}$$

$$H^3(\Gamma_0) = \{S{:}\ q_0,\ F{:}\ q_0 \wedge q_1 \to q_0\}$$

TRecS [K. PPDP09]

http://www.kb.ecei.tohoku.ac.jp/~koba/trecs/



- The first model checker for recursion schemes
- Based on the PPDP09 algorithm, with certain additional optimizations

Limitation of PPDP09 Algorithm

- Worst-case time complexity is even worse than the naive algorithm
  - Upper-bound

    $O(\exp_{n+1}(|G|^2))$
  - Lower-bound

    $O(\exp_n(|G|))$
  - Naive algorithm:

    $O(|G|)$

  (with the largest arity and automaton fixed)

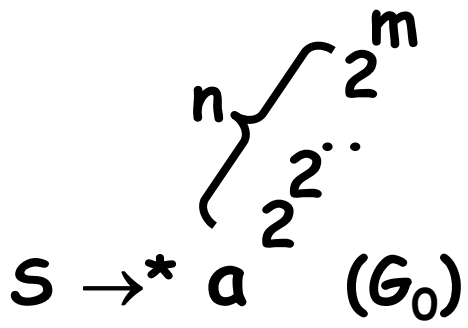
Order-1 recursion scheme that requires exponential reduction steps

$$S \to F_0\ G_0$$
$$F_0\ x \to F_1\ (F_1\ x)$$
$$\ldots$$
$$F_{m-1}\ x \to F_m\ (F_m\ x)$$
$$F_m\ x \to a\ x$$
$$G_0 \to c$$

$$S \to^* a^{2^m}\ (G_0) \to^* a^{2^m}\ (c)$$

Order-n recursion scheme $R_{m,n}$

$$S \to F_0\ G_{n-1}\ \ldots\ G_2\ G_1\ G_0$$
$$F_0\ f \to F_1\ (F_1\ f)$$
$$\ldots$$
$$F_{m-1}\ f \to F_m\ (F_m\ f)$$
$$F_m\ f \to G_n\ f$$
$$G_n\ f\ z \to f\ (f\ z)$$
$$\ldots$$
$$G_2\ f\ z \to f\ (f\ z)$$
$$G_1\ z \to a\ z$$
$$G_0 \to c$$



Verification Time for $R_{m,n}$

	m=1	m=2	m=3	m=4	m=5	m=10	m=15
n=1	0.002	0.002	0.002	0.002	0.003	0.036	2.866
n=2	0.002	0.002	0.011	228.4	-	-	-
n=3	0.002	394.3	-	-	-	-	-

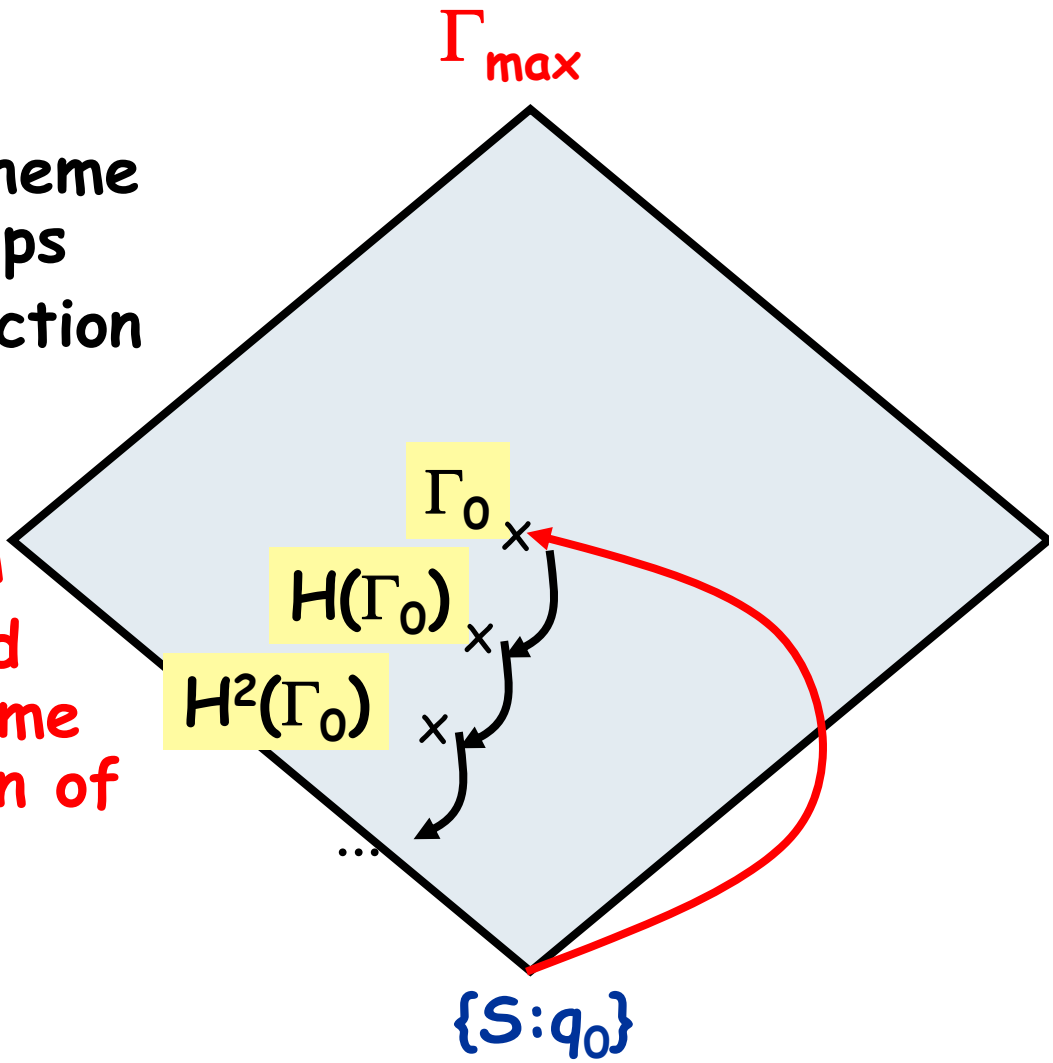
Specification: $\delta(q_0, a) = q_0 \quad \delta(q_0, c) = \varepsilon$
Environment: Intel(R) Xeon(R) 3Ghz with 8GB memory

Better Algorithm?

	Pros	Cons
Naive algorithm [POPL09]	Linear time in |G| (but n-EXPTIME in other parameters)	Always suffer from n-EXPTIME bottleneck
PPDP09 algorithm	Efficient in practice	Bad worst case behavior (n-EXPTIME in |G|)
?	Linear time in |G| Efficient in practice	

How to guess Γ_0?

- PPDP09 algorithm
  - Reduce a recursion scheme a finite number of steps
  - Observe how each function is used and express it as types
- FoSSaCS11 algorithm
  - Like PPDP09, but avoid reductions by using game semantic interpretation of types



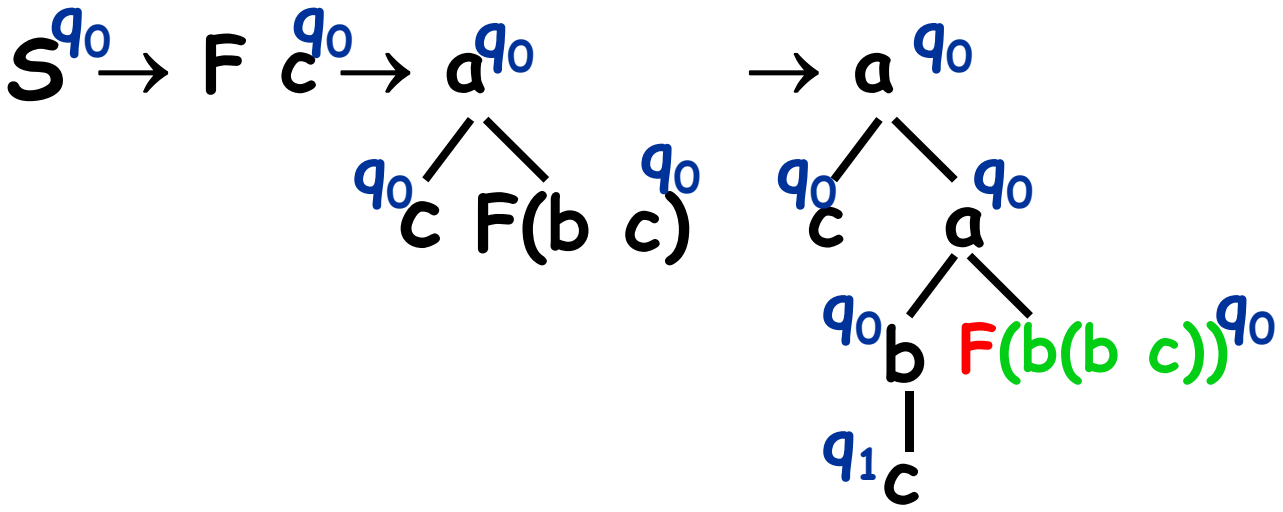
Example

- Recursion scheme:

$S \to F\ c \qquad F \to \lambda x.a\ x\ (F\ (b\ x))$

- Automaton:

$\delta(q_0, a) = q_0\ q_0 \qquad \delta(q_0, b) = q_1$
$\delta(q_0, c) = \delta(q_1, c) = \varepsilon$



Γ_0 :

S: q_0

F: $q_0 \wedge q_1 \to q_0$

F: $q_0 \to q_0$

F: $T \to q_0$

Example

- **Recursion scheme:**

$S \to F\ c \qquad F \to \lambda x.a\ x\ (F\ (b\ x))$

- **Automaton:**

$\delta(q_0, a) = q_0\ q_0 \qquad \delta(q_0, b) = q_1$
$\delta(q_0, c) = \delta(q_1, c) = \varepsilon$

$S \xrightarrow{q_0} F\ c \xrightarrow{q_0} a$ (at q_0)

q_0: c $\quad F(b\ c)$ (q_0)

$H(\Gamma_0) = \{ F_k{:}\tau \in \Gamma_0 \mid \Gamma_0 \vdash t_k{:}\tau \}$
$= \{S{:}\ q_0,\ F{:}\ q_0 \to q_0\}$
$H^2(\Gamma_0) = \{S{:}\ q_0\} \qquad H^3(\Gamma_0) = \varnothing$

Γ_0 :

$S{:}\ q_0$

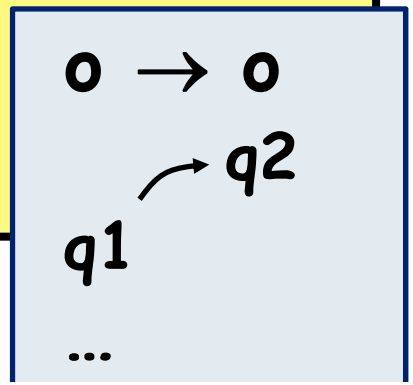
$F{:}\ q_0 \to q_0$

$F{:}\ T \to q_0$

Set-theoretic vs game-semantic interpretation of types

Set-theoretic view of $q1 \to q2$:
Given a tree of type q1, returns a tree of type q2.

Game-semantic view of $q1 \to q2$:
Given a request to return a tree of type q2,
issues a request for a tree of type q1
(and then returns a tree of type q2)



Example

- Recursion scheme:

$S \to F\ c \qquad F \to \lambda x.a\ x\ (F\ (b\ x))$

- Automaton:

$\delta(q_0, a) = q_0\ q_0 \qquad \delta(q_0, b) = q_1$
$\delta(q_0, c) = \delta(q_1, c) = \varepsilon$

$S \overset{q_0}{\to} F\ c \overset{q_0}{\to} a \overset{q_0}{\to}$

$q_0\ c \qquad F(b\ c)\ q_0$

Γ_0 :

S: q_0

F: $q_0 \to q_0$

F: $T \to q_0$

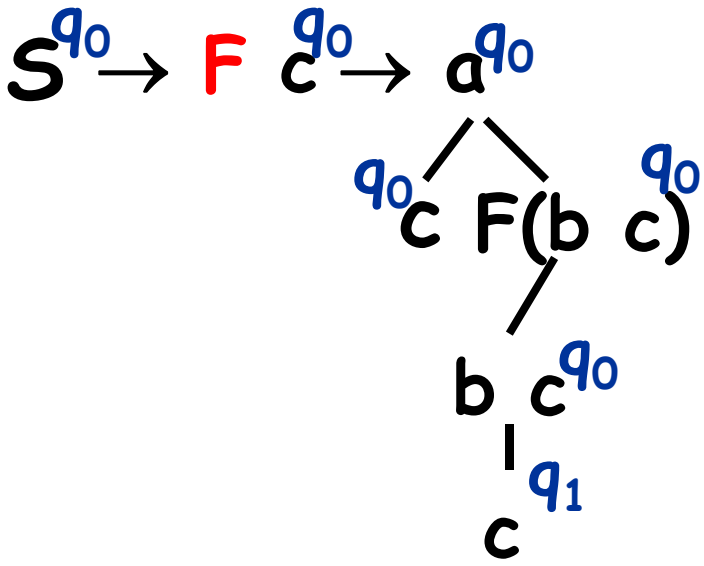
Example

- Recursion scheme:

$S \to F\ c \qquad F \to \lambda x.a\ x\ (F\ (b\ x))$

- Automaton:

$\delta(q_0, a) = q_0\ q_0 \qquad \delta(q_0, b) = q_1$
$\delta(q_0, c) = \delta(q_1, c) = \varepsilon$



Γ_0 :

S: q_0

F: $q_0 \to q_0$

F: $q_0 \wedge q_1 \to q_0$

Experiments

	order	PPDP09	FoSSaCS11
$R_{3,1}$	3	0.002	0.021
$R_{3,5}$	3	timeout	0.135
$R_{3,10}$	3	timeout	0.382
$R_{4,10}$	4	timeout	43.8
Twofiles	3	0.001	0.228
Twofiles-e	3	0.001	0.116
FileOcamlc	3	0.003	1.162
Nondet	3	N.A.	0.013

(Times are in seconds. Environment: Intel(R) Xeon(R) 3Ghz with 8GB memory)

Better Algorithm?

	Pros	Cons
Naive algorithm [POPL09]	Linear time in |G| (but n-EXPTIME in other parameters)	Always suffer from n-EXPTIME bottleneck
PPDP09 algorithm	Efficient in practice	Bad worst case behavior (n-EXPTIME in |G|)
FoSSaCS 2011 algorithm	Linear time in |G| Efficient in practice	Often slower than PPDP09 algorithm for program verification problems

Algorithms for Higher-Order Model Checking: Summary

- Model checking can be reduced to type checking, which in turn becomes a fixedpoint problem
- Greatest fixedpoint is too costly to compute
- Practical algorithms guess a type environment and use it as a start point of fixedpoint computation
- FoSSaCS11 algorithm (for trivial automata model checking) achieves fixed-parameter linear time complexity in the size of grammar by incorporating game-semantic view

Discussion

- Our FoSSaCS11 algorithm may be seen as abstract interpretation of game semantics (type as an abstraction of a set of plays)
  - Is this view correct?
  - Can we make this view precise, and use it to refine the algorithm and the correctness proof?