# POPL 2016 Tutorial: Higher-Order Model Checking


**Naoki Kobayashi**
University of Tokyo

**Luke Ong**
University of Oxford

# Outline

- Part 1: Introduction [by Ong, 15 minutes]
- Part 2: Applications to program verification [by Kobayashi, 25 minutes]
- Part 3: Type systems and algorithms for higher-order model checking [by Kobayashi, 25 minutes]
- Part 4: Advanced topics [by Ong, 25 minutes]

# Tool demonstration: MoCHi (a software model checker for a subset of OCaml)

# Higher-Order Model Checking

Given
G: HORS
A: alternating parity tree automaton (APT)
(a formula of modal $\mu$-calculus or MSO),
does A accept Tree(G)?

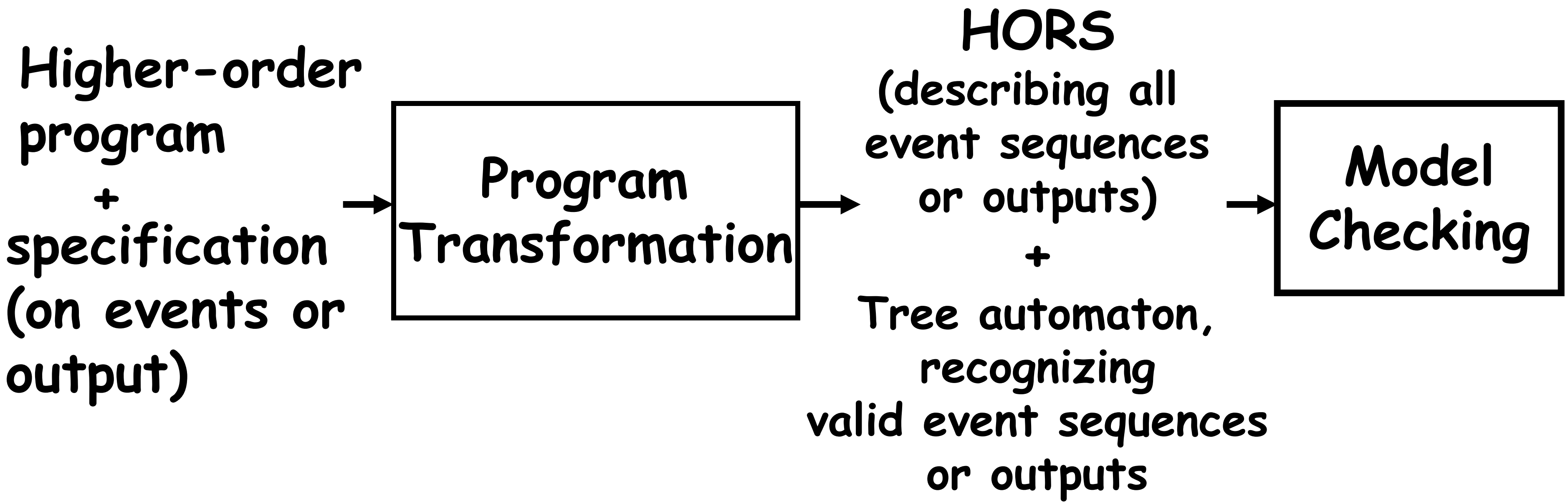
e.g.
- Does every finite path end with "c"?
- Does "a" occur below "b"?

k-EXPTIME-complete [Ong, LICS06]
(for order-k HORS),
but practical algorithms exist

$$k\left\{ 2^{2^{\cdot^{\cdot^{\cdot^{2^{p(x)}}}}}} \right.$$

# From Program Verification to HO Model Checking

[K. POPL 2009]

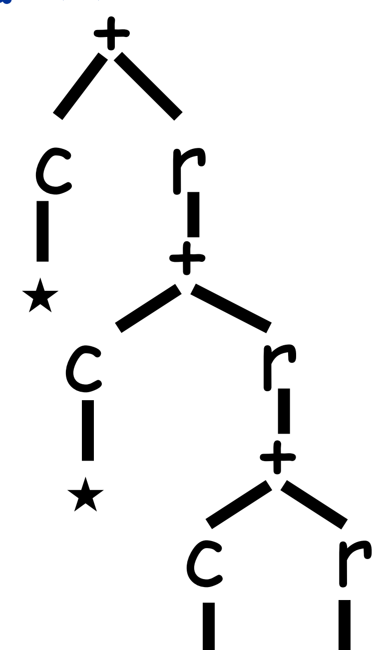


# From Program Verification to Model Checking: Example

```
let f x =
 if * then close(x)
 else (read(x); f x)
in
let y = open "foo"
in
     f (y)
```

$F\ x\ k \rightarrow +\ (c\ k)\ (r(F\ x\ k))$

$S \rightarrow F\ d\ \star$




Is the file "foo" accessed according to read* close?

Is each path of the tree labeled by r*c?

# From Program

g:

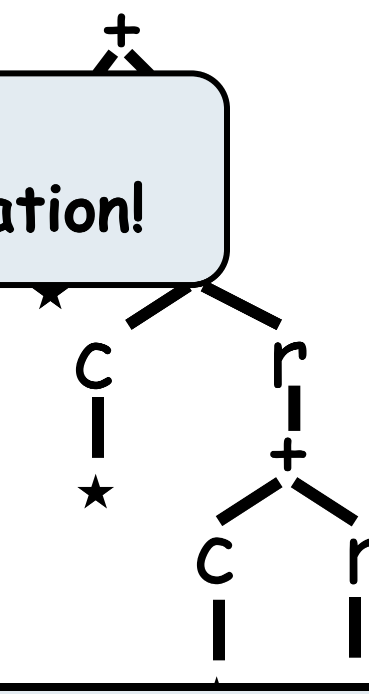
continuation parameter, expressing how "foo" is accessed after the call returns

```
let f x =
 if * then close(x)
 else (read(x); f x)
in
let y = open "foo"
in
      f (y)
```

$$F\ x\ k \rightarrow +\ (c\ k)\ (r(F\ x\ k))$$

$$S \rightarrow F\ d\ \star$$

CPS Transformation!




Is the file "foo" accessed according to read* close?

Is each path of the tree labeled by r*c?

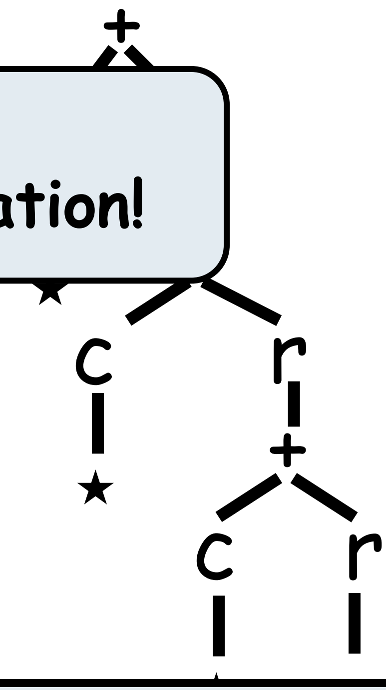
# From Program Verification to Model Checking: Example

```
let f x =
 if * then close(x)
 else (read(x); f x)
in
let y = open "foo"
in
      f (y)
```

F x k → + (c k) (r(F x k))

S → F d ★

CPS Transformation!




Is the file "foo" accessed according to read* close?

Is each path of the tree labeled by r*c?

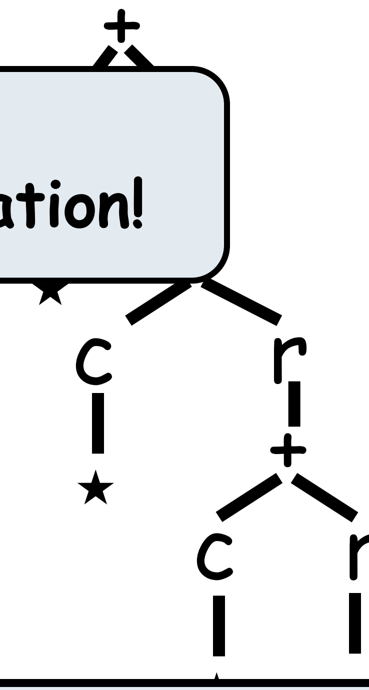
# From Program Verification to Model Checking: Example

```
let f x =
  if * then close(x)
  else (read(x); f x)
in
let y = open "foo"
in
      f (y)
```

$F\ x\ k \rightarrow +\ (c\ k)\ (r(F\ x\ k))$

$S \rightarrow F\ d\ \star$

CPS Transformation!




Is the file "foo" accessed according to read* close?

Is each path of the tree labeled by r*c?

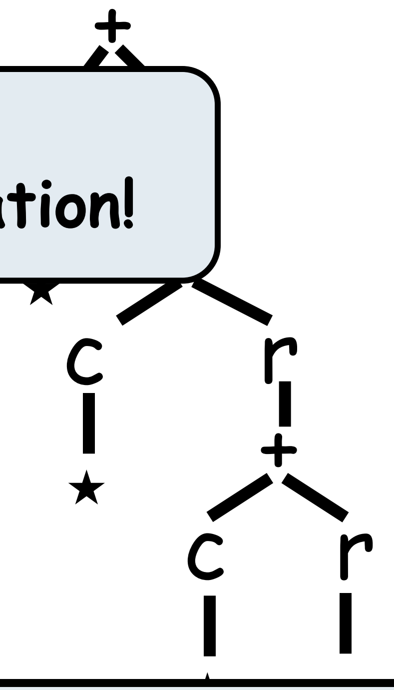
# From Program Verification to Model Checking: Example

```
let f x =
  if * then close(x)
  else (read(x); f x)
in
let y = open "foo"
in
     f (y)
```

F x k → + (c k) (r(F x k))

S → F d ★

CPS Transformation!




Is the file "foo" accessed according to read* close?

Is each path of the tree labeled by r*c?

# From Program Verification to Model Checking: Example

```
let f(x) =
 if * then close(x)
 else (read(x); f x)
in
let y = open "foo"
in
      f (y)
```

$F\ x\ k \rightarrow +\ (c\ k)\ (r(F\ x\ k))$

$S \rightarrow F\ d\ \star$

S

Is the file "foo" accessed according to read* close?

Is each path of the tree labeled by r*c?

# From Program Verification to Model Checking: Example

```
let f(x) =
 if * then close(x)
 else (read(x); f x)
in
let y = open "foo"
in
     f (y)
```

$$F\ x\ k \rightarrow +\ (c\ k)\ (r(F\ x\ k))$$

$$S \rightarrow F\ d\ \star$$

$$F\ d\ \star$$

Is the file "foo" accessed according to read* close?

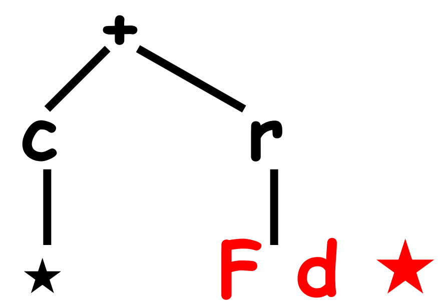
Is each path of the tree labeled by r*c?

# From Program Verification to Model Checking: Example

```
let f(x) =
 if * then close(x)
 else (read(x); f x)
in
let y = open "foo"
in
      f (y)
```

$F\ x\ k \to +\ (c\ k)\ (r(F\ x\ k))$

$S \to F\ d\ \star$




Is the file "foo" accessed according to read* close?

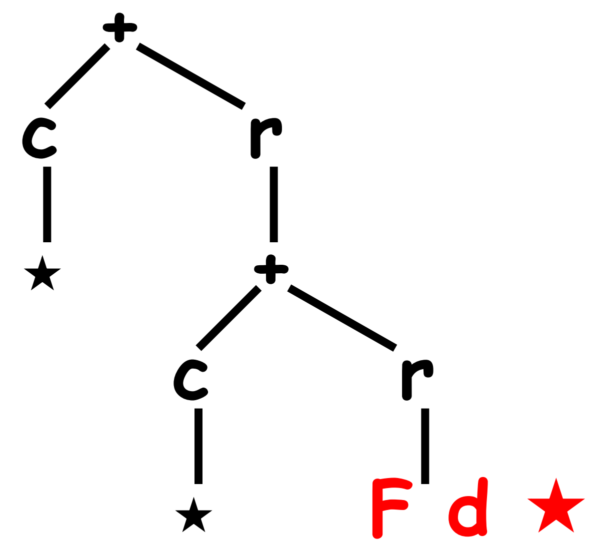
Is each path of the tree labeled by r*c?

# From Program Verification to Model Checking: Example

```
let f(x) =
 if * then close(x)
 else (read(x); f x)
in
let y = open "foo"
in
      f (y)
```

$$F\ x\ k \rightarrow +\ (c\ k)\ (r(F\ x\ k))$$

$$S \rightarrow F\ d\ \star$$




Is the file "foo" accessed according to read* close?

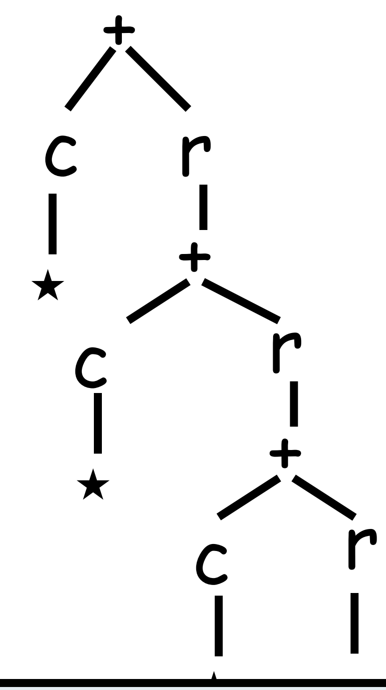
Is each path of the tree labeled by r*c?

# From Program Verification to Model Checking: Example

```
let f(x) =
 if * then close(x)
 else (read(x); f x)
in
let y = open "foo"
in
     f (y)
```

$F\ x\ k \rightarrow +\ (c\ k)\ (r(F\ x\ k))$

$S \rightarrow F\ d\ \star$




Is the file "foo" accessed according to read* close?

Is each path of the tree labeled by r*c?

# Example 2: handling exceptions

```
let read'(x) =
  read(x);
  if * then ()
  else raise Eof
let f(x) =
   read'(x); f(x)
in
let y = open "foo"
in try f(y) with
     Eof -> close y
```

exception handler

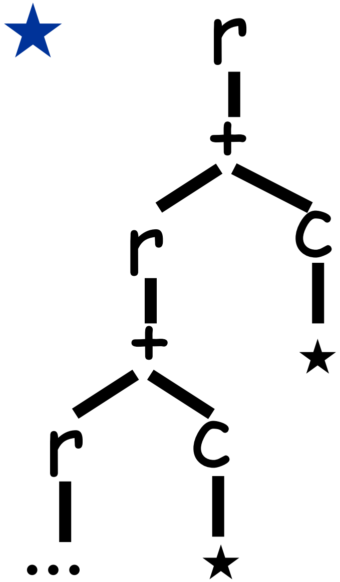
continuation for normal termination

Read' x h k → r (+ k h)

F x h k

→ Read' x h (F x h k)

S → F d (c ★) ★




Is the file "foo" accessed according to read* close?

Is each path of the tree labeled by r*c?

# Example 3: handling Booleans

```
let f x =
  if eof(x)
  then
     close(x)
  else (read(x); f x)
in
let y = open "foo"
in
     f (y)
```

Return whether end of file has been reached

$F\ x\ k \to$
$\quad Eof\ x\ (\lambda b.If\ b\ (c\ k)\ (r\ (F\ x\ k))).$

$S \to F\ d\ \star.$

$Eof\ x\ k \to$
$\quad +\ (eof\ (k\ True))\ (k\ False).$

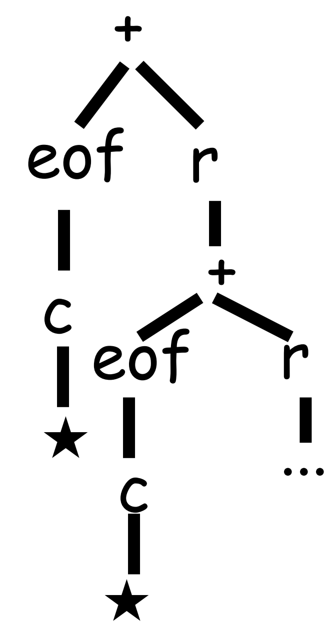
eof has been reached

$If\ b\ x\ y \to b\ x\ y.$

$True\ x\ y \to x.$

$False\ x\ y \to y.$

Church encoding of Booleans




**Is the file closed only after eof has been reached?** ⟹ **Does c occur only below eof?**

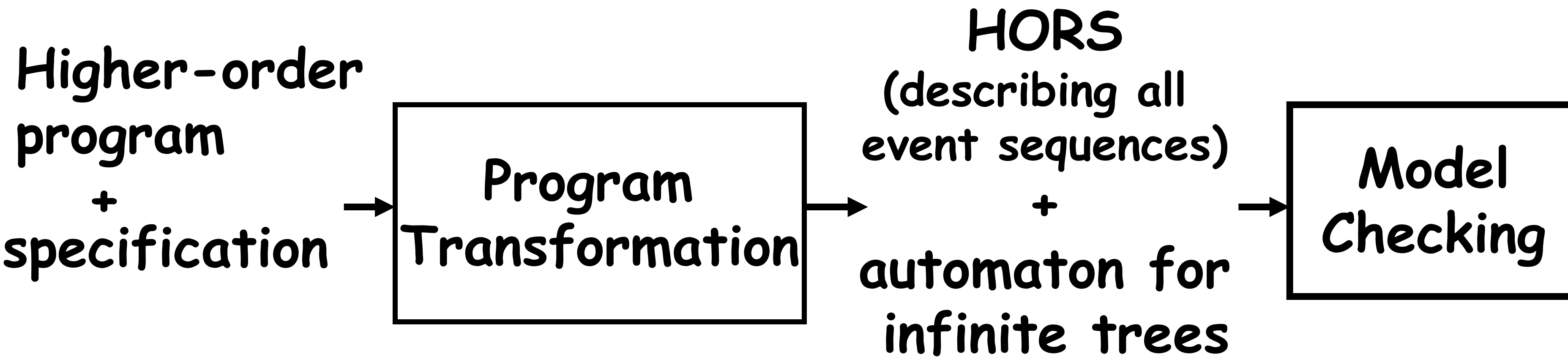
# From Program Verification to HO Model Checking

Higher-order program + specification → **Program Transformation** → HORS (describing all event sequences) + automaton for infinite trees → **Model Checking**

Sound, complete, and automatic for:

- A large class of higher-order programs:
  simply-typed λ-calculus + recursion
  + finite base types (e.g. booleans) + exceptions + ...
- A large class of verification problems:
  resource usage verification (or typestate checking), reachability, flow analysis, strictness analysis, ...

# From Program Verification to HO Model Checking




**For finite-data HO programs, automated verification comes for free from HO model checking!**

# Outline

- Introduction [by Ong, 15 minutes]
- Applications to program verification
  [by Kobayashi, 25 minutes]
  - Verification of finite-data programs
  - Verification of infinite-data programs
- Type systems and algorithms for higher-order model checking [by Kobayashi, 25 minutes]
- Advanced topics [by Ong, 25 minutes]

# Verification of Higher-order Programs with Infinite Data Domains (integers, lists, trees, ...)

- For safety properties (e.g. reachability), overapproximation by abstraction of infinite data suffices.
- For other properties (e.g. termination), combinations of problem reduction and abstraction are required.

# Verification of Higher-order Programs with Infinite Data Domains (integers, lists, trees, ...)

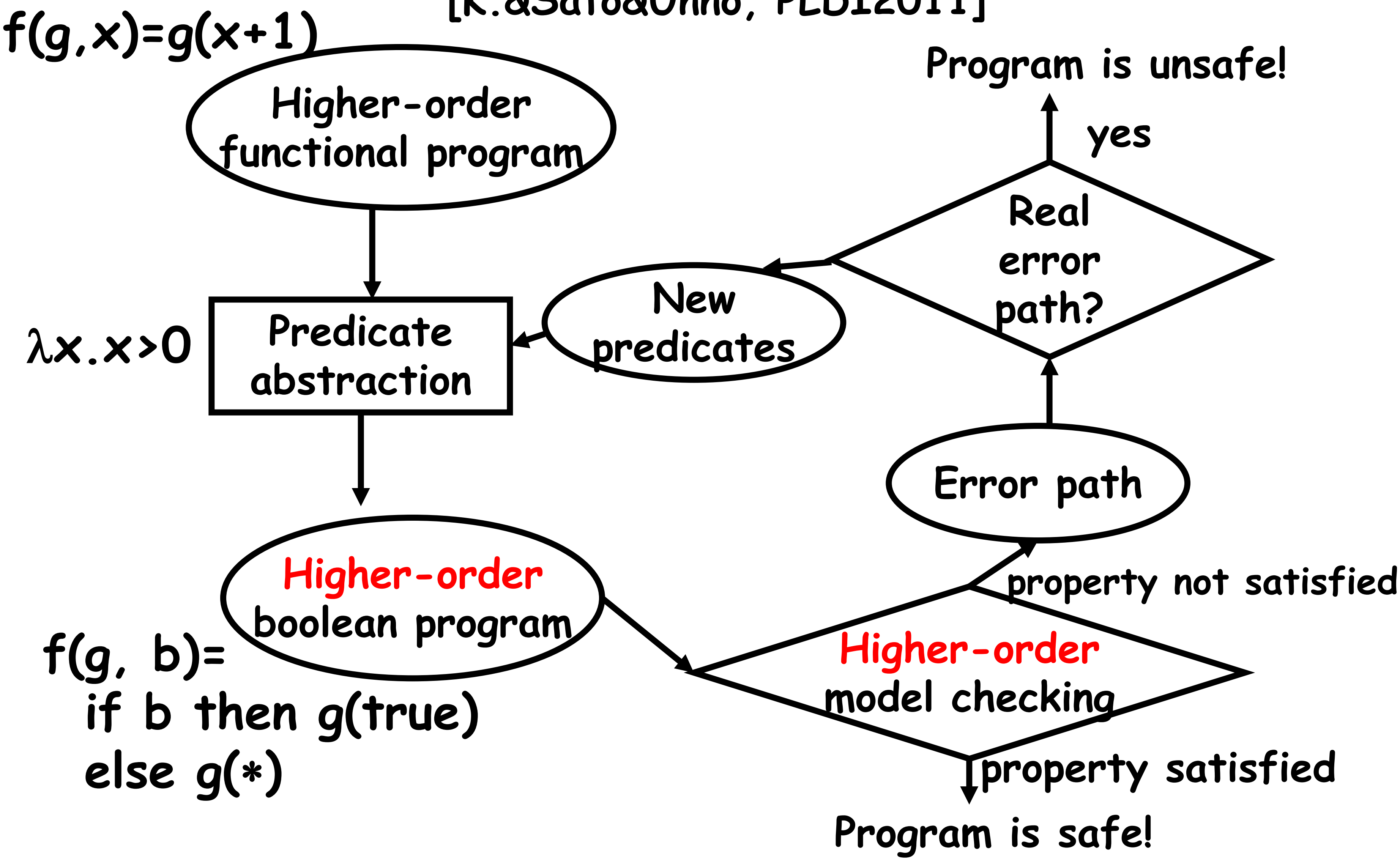
- For safety properties (e.g. reachability), overapproximation by abstraction of infinite data suffice.

- For other properties (e.g. termination), combinations of problem reduction and abstraction are required.

  => see our papers in ESOP 2014, CAV 2015 and POPL 2016

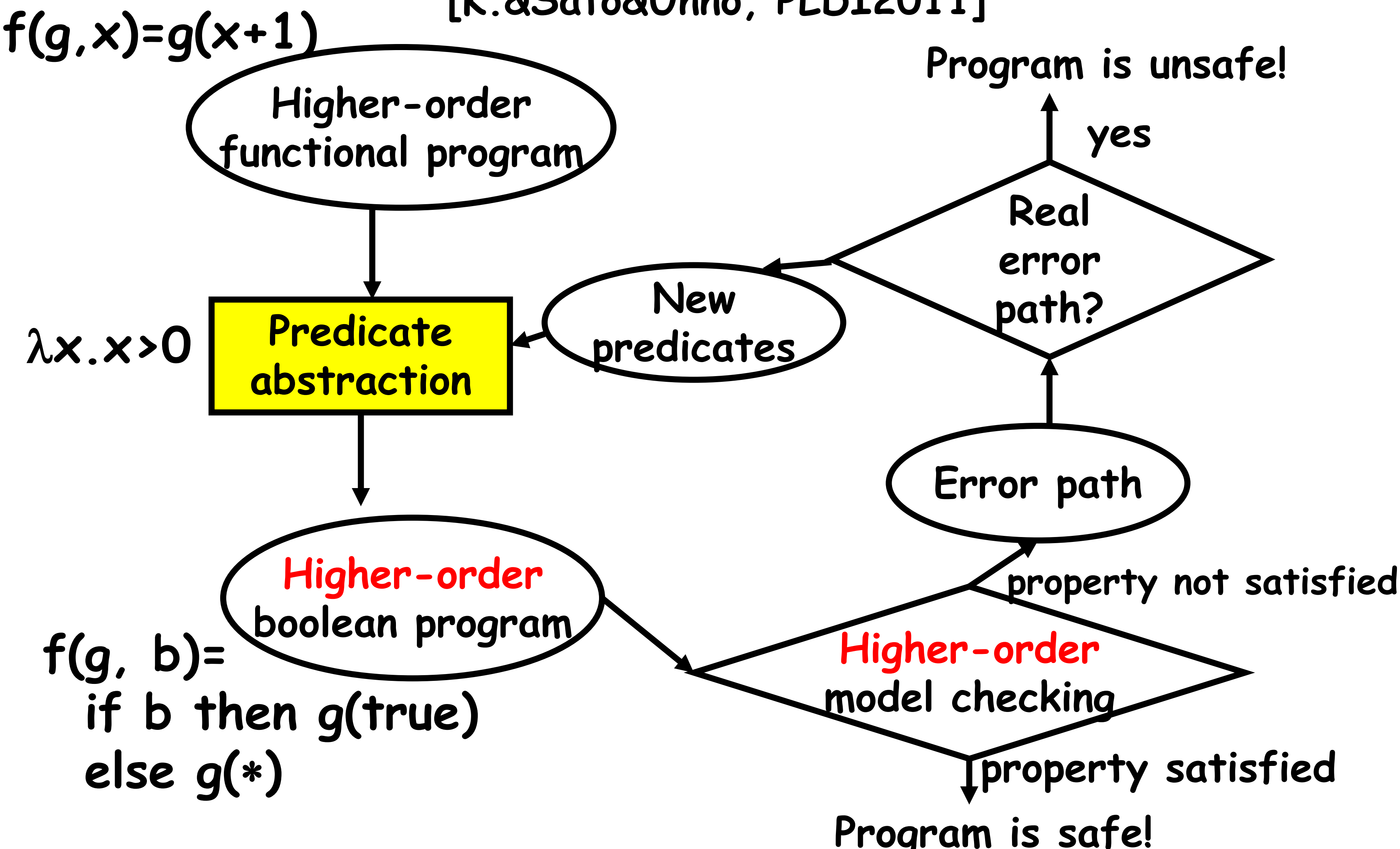
# Predicate Abstraction and CEGAR for Higher-Order Model Checking

[K.&Sato&Unno, PLDI2011]



# Predicate Abstraction and CEGAR for Higher-Order Model Checking

[K.&Sato&Unno, PLDI2011]



# Abstraction Types

Used to specify which predicates should be used for abstraction of each expression

- $\mathrm{int}[P_1,...,P_n]$

  Integers that should be abstracted by $P_1,...,P_n$

  e.g. $3$: $\mathrm{int}[\lambda x.x>0,\ \mathrm{even?}] \Rightarrow (\mathrm{true},\ \mathrm{false})$

- $x{:}\mathrm{int}[P_1,...,P_n] \rightarrow \mathrm{int}[Q_1,...,Q_m]$

  Assuming that argument $x$ is abstracted by $P_1,...,P_n$, abstract the return value by $Q_1,...,Q_m$

  e.g. $\lambda x.x+x$: $(x{:}\mathrm{int}[\lambda x.x>0] \rightarrow \mathrm{int}[\lambda y.y>x]) \Rightarrow \lambda b.?$

x>0?

x+x>x?

# Abstraction Types

Used to specify which predicates should be used for abstraction of each expression

- $\mathrm{int}[P_1,\ldots,P_n]$

  Integers that should be abstracted by $P_1,\ldots,P_n$

  e.g. $3: \mathrm{int}[\lambda x.x>0,\ \mathrm{even?}] \Rightarrow (\mathrm{true},\ \mathrm{false})$

- $x{:}\mathrm{int}[P_1,\ldots,P_n] \rightarrow \mathrm{int}[Q_1,\ldots,Q_m]$

  Assuming that argument x is abstracted by $P_1,\ldots,P_n$, abstract the return value by $Q_1,\ldots,Q_m$

  e.g. $\lambda x.x+x: (x{:}\mathrm{int}[\lambda x.x>0] \rightarrow \mathrm{int}[\lambda y.y>x]) \Rightarrow \lambda b.b$

x>0?

x+x>x?

# Abstraction Types

Used to specify which predicates should be used for abstraction of each expression

- $\mathrm{int}[P_1,\ldots,P_n]$

  Integers that should be abstracted by $P_1,\ldots,P_n$

  e.g. $3\colon \mathrm{int}[\lambda x.x>0,\ \mathrm{even?}] \Rightarrow (\mathrm{true},\ \mathrm{false})$

- $x\colon\mathrm{int}[P_1,\ldots,P_n] \rightarrow \mathrm{int}[Q_1,\ldots,Q_m]$

  Assuming that argument x is abstracted by $P_1,\ldots,P_n$, abstract the return value by $Q_1,\ldots,Q_m$

  e.g. $\lambda x.x+x\colon (x\colon\mathrm{int}[\lambda x.x>0] \rightarrow \mathrm{int}[\lambda y.y>x]) \Rightarrow \lambda b.b$

  $\lambda x.x+x\colon (x\colon\mathrm{int}[\lambda x.x>1,\ \mathrm{even?}] \rightarrow \mathrm{int}[\lambda y.y>0])$

  $\Rightarrow \lambda(b_1,b_2).\mathrm{if}\ b_1\ \mathrm{then\ true\ else}\ *$

# Example (predicate abstraction)

```
let mc91 x = if x > 100 then x - 10
               else mc91 (mc91 (x + 11))
let main n = if n <= 101 then assert (mc91 n = 91)
```

**Abstraction type of mc91:**

**$x\text{:int}[\lambda x.x>101] \rightarrow \text{int}[\lambda r.r=91,\ \lambda r.r=x-10]$**

```
let mc91 b_{x>101} =
   if (if b_{x>101} then true else *) then (not(b_{x>101}), true)
   else let (b_{r1=91},b_{r1=x-10}) = mc91 * in
        let (b_{r=91},b_{r=r1-10})=
             mc91 (if b_{r1=91} || b_{r1=x-10} then false else *)
        in (b_{r=91}, *)
let main () = if * then
          assert(let (b_{r=91},b_{r=x-10}) = mc91 false in b_{r=91})
```

# Example (predicate abstraction)

```
let mc91 x = if x > 100 then x - 10
                 else mc91 (mc91 (x + 11))
let main n = if n <= 101 then assert (mc91 n = 91)
```

**Abstraction type of mc91:**

**$x{:}\mathrm{int}[\lambda x.x>101] \rightarrow \mathrm{int}[\lambda r.r=91,\ \lambda r.r=x-10]$**

let mc91 $b_{x>101}$ =

  if (if $b_{x>101}$ then true else $*$) then (not($b_{x>101}$), true)

  else let ($b_{r1=91}$,$b_{r1=x-10}$) = mc91 $*$ in

    let ($b_{r=91}$,$b_{r=r1-10}$)=

      mc91 (if $b_{r1=91}$ || $b_{r1=x-10}$ then false else $*$)

    in ($b_{r=91}$, $*$)

let main () = if $*$ then

      assert(let ($b_{r=91}$,$b_{r=x-10}$) = mc91 false in $b_{r=91}$)

# Example (predicate abstraction)

```
let mc91 x = if x > 100 then x - 10
                 else mc91 (mc91 (x + 11))
let main n = if n <= 101 then assert (mc91 n = 91)
```

**Abstraction type of mc91:**

**$x{:}\mathrm{int}[\lambda x.x>101] \rightarrow \mathrm{int}[\lambda r.r=91,\ \lambda r.r=x-10]$**

let mc91 $b_{x>101}$ =

  if (if $b_{x>101}$ then true else $*$) then (not($b_{x>101}$), true)

  else let ($b_{r1=91}$,$b_{r1=x-10}$) = mc91 $*$ in

    let ($b_{r=91}$,$b_{r=r1-10}$)=

      mc91 (if $b_{r1=91}$ || $b_{r1=x-10}$ then false else $*$)

    in ($b_{r=91}$, $*$)

let main () = if $*$ then

      assert(let ($b_{r=91}$,$b_{r=x-10}$) = mc91 false in $b_{r=91}$)

# Example (predicate abstraction)

```
let mc91 x = if x > 100 then x - 10
                 else mc91 (mc91 (x + 11))
let main n = if n <= 101 then assert (mc91 n = 91)
```

**Abstraction type of mc91:**

**$x{:}int[\lambda x.x>101] \rightarrow int[\lambda r.r=91,\ \lambda r.r=x-10]$**

let mc91 $b_{x>101}$ =

if (if $b_{x>101}$ then true else $*$) then (not($b_{x>101}$), true)

else let ($b_{r1=91}$,$b_{r1=x-10}$) = mc91 $*$ in

let ($b_{r=91}$,$b_{r=r1-10}$)=

mc91 (if $b_{r1=91}$ || $b_{r1=x-10}$ then false else $*$)

in ($b_{r=91}$, $*$)

let main () = if $*$ then

assert(let ($b_{r=91}$,$b_{r=x-10}$) = mc91 false in $b_{r=91}$)

# Example (predicate abstraction)

```
let mc91 x = if x > 100 then x - 10
                 else mc91 (mc91 (x + 11))
let main n = if n <= 101 then assert (mc91 n = 91)
```

**Abstraction type of mc91:**

**$x{:}\mathrm{int}[\lambda x.x>101] \rightarrow \mathrm{int}[\lambda r.r=91,\ \lambda r.r=x-10]$**

let mc91 $b_{x>101}$ =

  if (if $b_{x>101}$ then true else $*$) then (not($b_{x>101}$), true)

  else let ($b_{r1=91}$,$b_{r1=x-10}$) = mc91 $*$ in

    let ($b_{r=91}$,$b_{r=r1-10}$)=

      mc91 (if $b_{r1=91}$ || $b_{r1=x-10}$ then false else $*$)

    in ($b_{r=91}$, $*$)

let main () = if $*$ then

    assert(let ($b_{r=91}$,$b_{r=x-10}$) = mc91 false in $b_{r=91}$)

# Example (predicate abstraction)

```
let mc91 x = if x > 100 then x - 10
                 else mc91 (mc91 (x + 11))
let main n = if n <= 101 then assert (mc91 n = 91)
```

**Abstraction type of mc91:**

$x\mathtt{:int}[\lambda x.x>101] \rightarrow \mathtt{int}[\lambda r.r=91,\ \lambda r.r=x-10]$

let mc91 $b_{x>101}$ =

if (if $b_{x>101}$ then true else $*$) then (not($b_{x>101}$), true)

else let ($b_{r1=91}$,$b_{r1=x-10}$) = mc91 $*$ in

let ($b_{r=91}$,$b_{r=r1-10}$)=

mc91 (if $b_{r1=91}$ || $b_{r1=x-10}$ then false else $*$)

in ($b_{r=91}$, $*$)

let main () = if $*$ then

assert(let ($b_{r=91}$,$b_{r=x-10}$) = mc91 false in $b_{r=91}$)

# Dealing with algebraic data types (e.g. lists)

- **Abstraction approach:**
  - automata-based [K+ POPL10][Unno+ APLAS 10]...
  - pattern-based [Ong&Ramsay POPL11]
- **Encoding approach** [Sato+ PEPM13] **:**
  - algebraic data as functions

$$[\,\tau\ \mathrm{list}\,] = \underbrace{\mathrm{int}}^{\text{length}} \times (\overbrace{\mathrm{int} \rightarrow [\tau]}^{\text{function from indices to elements}})$$

$\mathrm{nil} = (0,\ \lambda x.\ \mathrm{fail}\ )$

$\mathrm{cons} = \lambda x.\lambda(\mathrm{len}, f).$
$\qquad (\mathrm{len}+1,\ \lambda i.\mathrm{if}\ i=0\ \mathrm{then}\ x\ \mathrm{else}\ f(i-1))$

$\mathrm{hd}\ (\mathrm{len}, f) = f(0)$

...

# Summary of Part 2

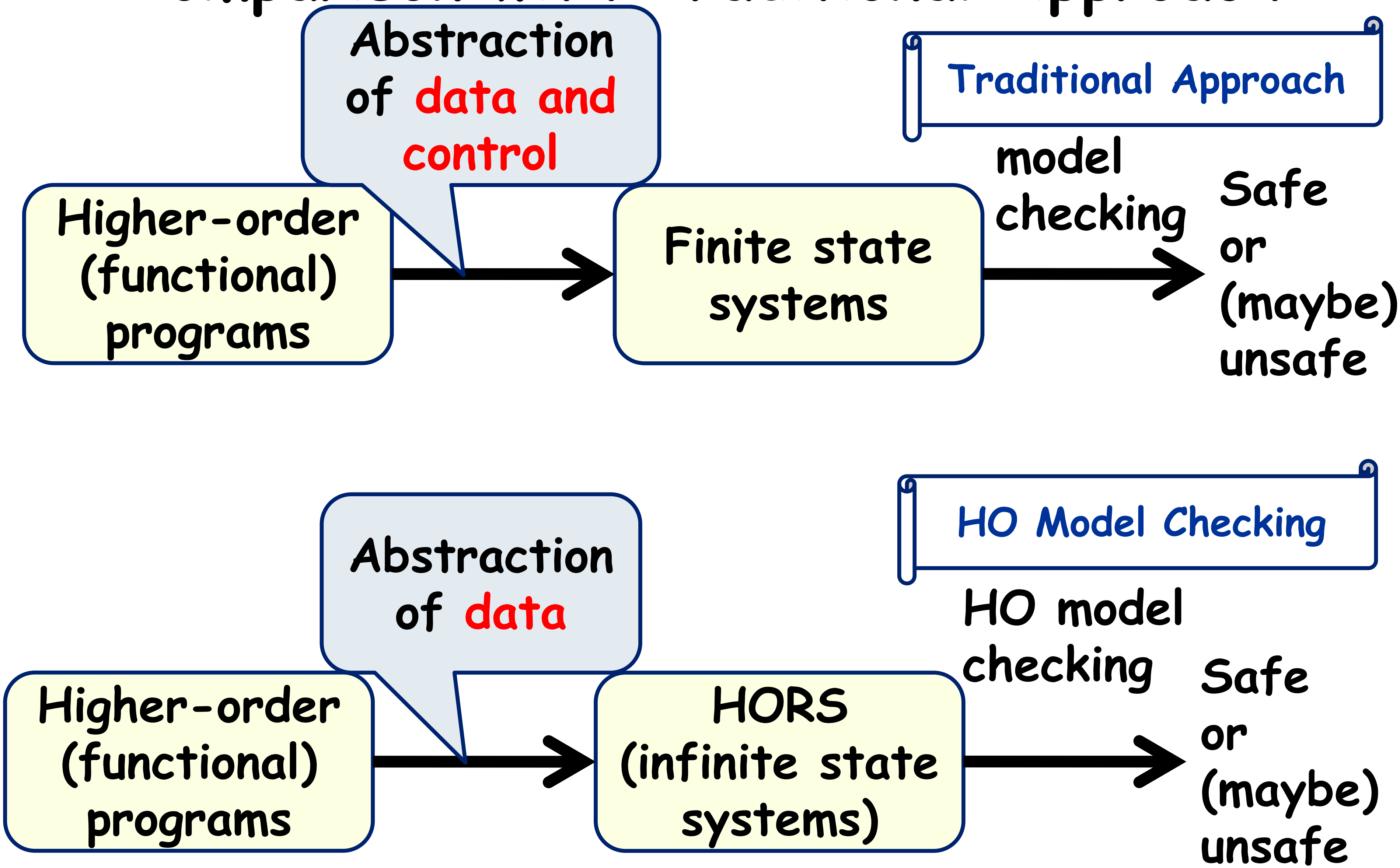
- For finite-data HO programs:
  sound, complete, and fully automatic verification is achieved by reduction to HO model checking

- For infinite-data HO programs:
  sound and automatic (but incomplete) verification is achieved by a combination of:
  - HO model checking
  - abstraction, and
  - program transformation

  Verification methods are necessarily incomplete,
  but often more precise than other approaches;

  sometimes relatively complete modulo certain assumptions
  [Unno, Terauchi &K, POPL 2013]

# Comparison with Traditional Approach



# References on Part 2


- Naoki Kobayashi, Model checking higher-order programs, J. ACM, 60(3), 2013 [reductions of various problems to HO model checking]
- Yoshihiro Tobita, Takeshi Tsukada, Naoki Kobayashi, Exact Flow Analysis by Higher-Order Model Checking, FLOPS 2012 [reduction from flow analysis to HO model checking]
- Luke Ong and Steven Ramsay, Verifying higher-order functional programs with pattern-matching algebraic data types. POPL 2011: 587-598 [verification of functional programs with pattern matching]
- Naoki Kobayashi, Ryosuke Sato, and Hiroshi Unno, Predicate abstraction and CEGAR for higher-order model checking, PLDI 2011 [predicate abstraction]
- Ryosuke Sato, Hiroshi Unno, and Naoki Kobayashi, Towards a scalable software model checker for higher-order programs, PEPM 2013 [exceptions and algebraic data types]

# References on Part 2


- Takuya Kuwahara, Tachio Terauchi, Hiroshi Unno, and Naoki Kobayashi, Automatic Termination Verification for Higher-Order Functional Programs, ESOP 2014
  [termination verification]
- Takuya Kuwahara, Ryosuke Sato, Hiroshi Unno and Naoki Kobayashi, Predicate Abstraction and CEGAR for Disproving Termination of Higher-Order Functional Programs. CAV 2015
  [non-termination verification]
- Akihiro Murase, Tachio Terauchi, Naoki Kobayashi, Ryosuke Sato and Hiroshi Unno, Temporal Verification of Higher-order Functional Programs, POPL 2016
  [liveness]
- Kazuhide Yasukata, Naoki Kobayashi, Kazutaka Matsuda, Pairwise Reachability Analysis for Higher Order Concurrent Programs by Higher-Order Model Checking. CONCUR 2014
  [verification of concurrent programs]

# Outline

- Part 1: Introduction [by Ong, 15 minutes]
- Part 2: Applications to program verification [by Kobayashi, 25 minutes]
- Part 3: Type systems and algorithms for higher-order model checking [by Kobayashi, 25 minutes]
- Part 4: Advanced topics [by Ong, 25 minutes]

# Practical algorithms for HO model checking?

- Many program verification problems can be reduced to HO model checking
- Unfortunately, HO model checking is k-EXPTIME complete for order-k HORS
- Fortunately, there are practical algorithms that work well for typical inputs
  - The state-of-the-art HO model checker HorSat2 can handle 1,000 – 100,000 lines of input
- Most of those algorithms are based on type-based characterization of HO model checking

# Type-based approach to HO model checking [K POPL09][KO LICS09]

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])

# HO Model Checking Problem

Given
  G: HORS
  A: alternating parity tree automaton (APT)
    (a formula of modal $\mu$-calculus or MSO),
does A accept Tree(G)?

k-EXPTIME-complete [Ong, LICS06]
(for order-k HORS)

# HO Model Checking Problem: Restricted version

Given

G: HORS

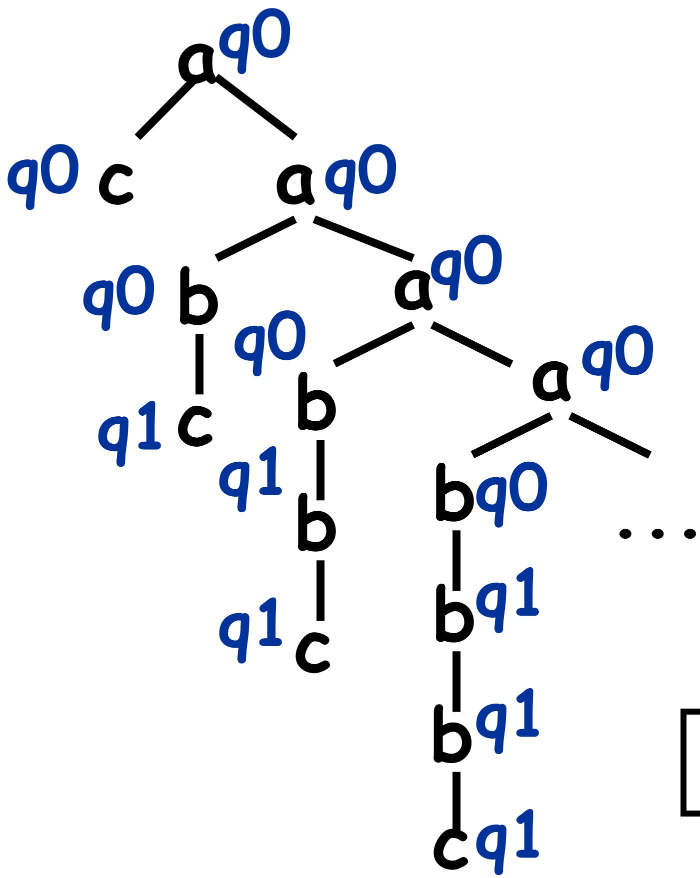
A: trivial automaton [Aehlig CSL06]

(Büchi tree automaton where all the states are accepting states)

does A accept Tree(G)?

k-EXPTIME-complete [KO, ICALP09]
(for order-k HORS)

# Trivial tree automaton for infinite trees



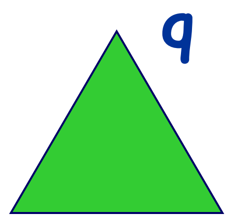

$$\delta(q0,\ a) = q0\ q0$$
$$\delta(q0,\ b) = q1$$
$$\delta(q1,\ b) = q1$$
$$\delta(q0,\ c) = \varepsilon$$
$$\delta(q1,\ c) = \varepsilon$$

"a" does not occur below "b"

A tree is accepted just if a run of the automaton does not get stuck (no acceptance conditions, such as Buchi/Muller/parity)

# Types for HORS

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




  - q1∧q2: trees accepted from both q1 and q2

Is Tree(G) accepted by A?

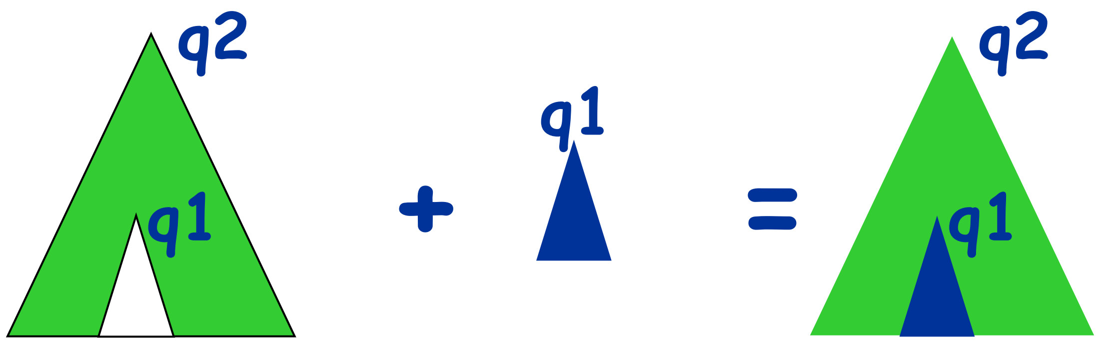
⇓

Does Tree(G) have type $q_0$?

# Types for HORS

$q1 \rightarrow q2$:

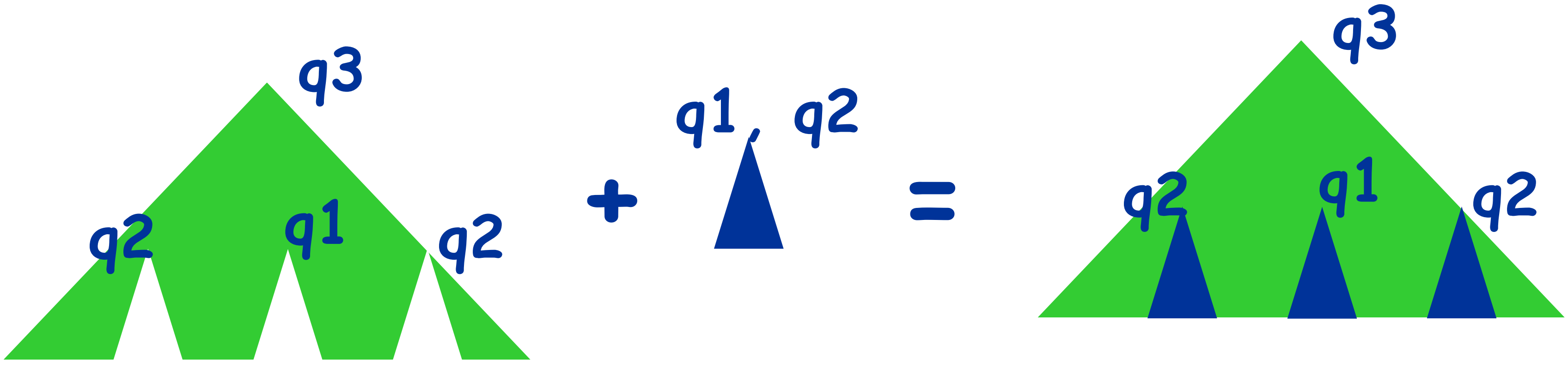
functions that take a tree of type q1 and return a tree of q2



# Types for HORS

$q1 \wedge q2 \rightarrow q3$:

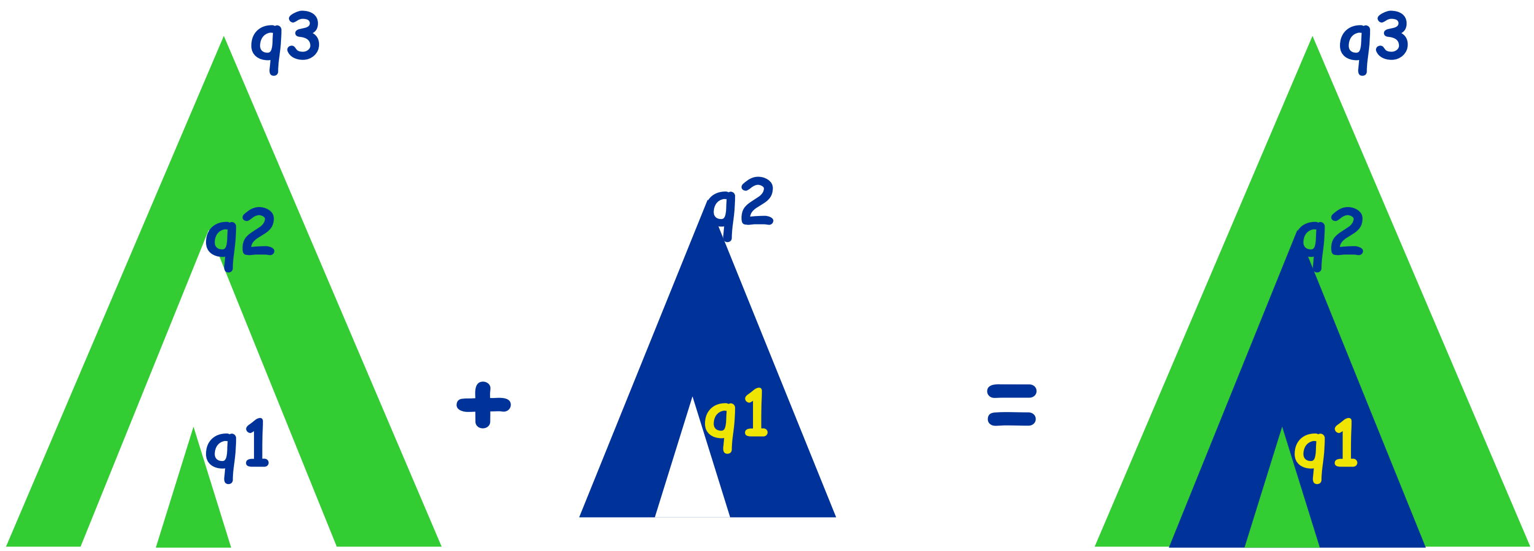
functions that take a tree of type $q1 \wedge q2$ and return a tree of type $q3$



# Types for HORS

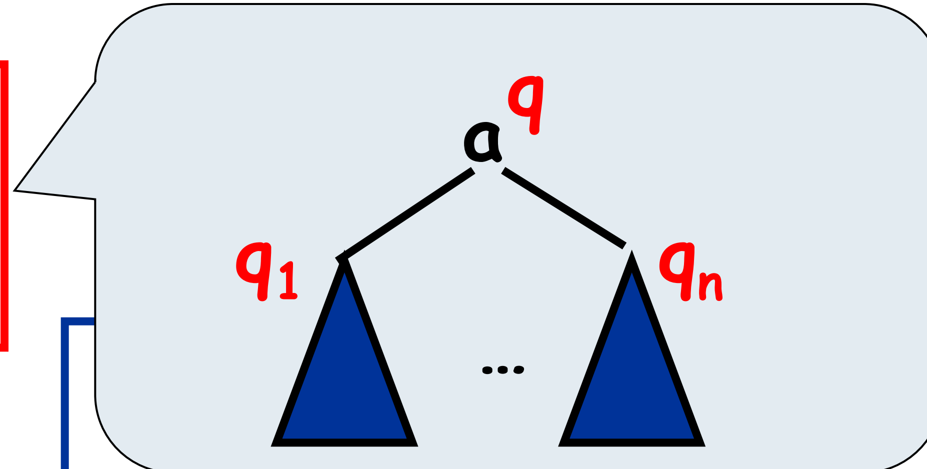
$(q1 \to q2) \to q3$:

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



# Typing

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




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

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

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

# Typing

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

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

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

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

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

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

if and only if

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

$G = \{F_1 \rightarrow t_1,\ \ldots,\ F_m \rightarrow t_m\}$ (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

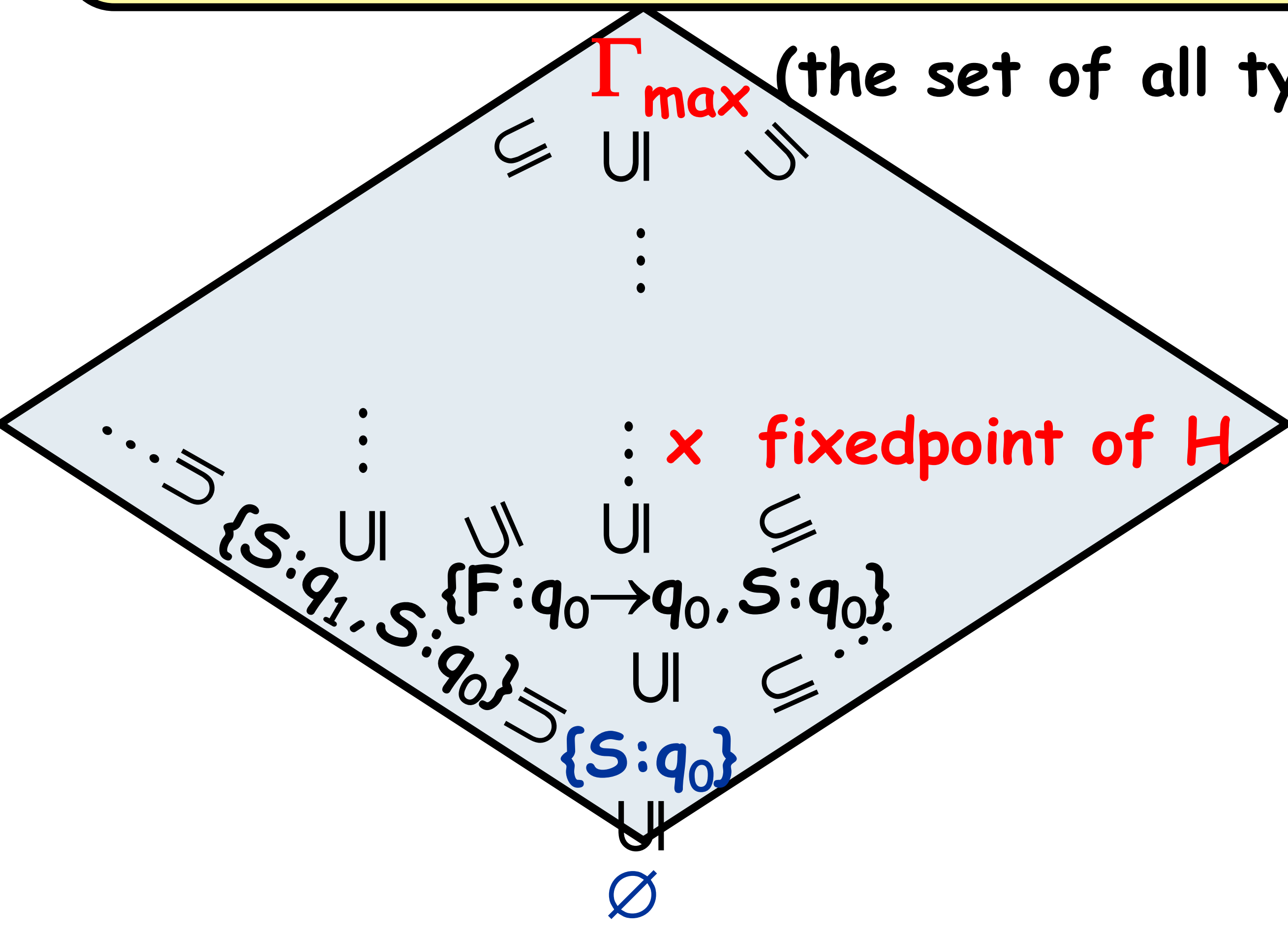
$\exists\Gamma.(S{:}\ q_0 \in \Gamma \wedge \Gamma = H(\Gamma))$

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

Function to filter out invalid type bindings

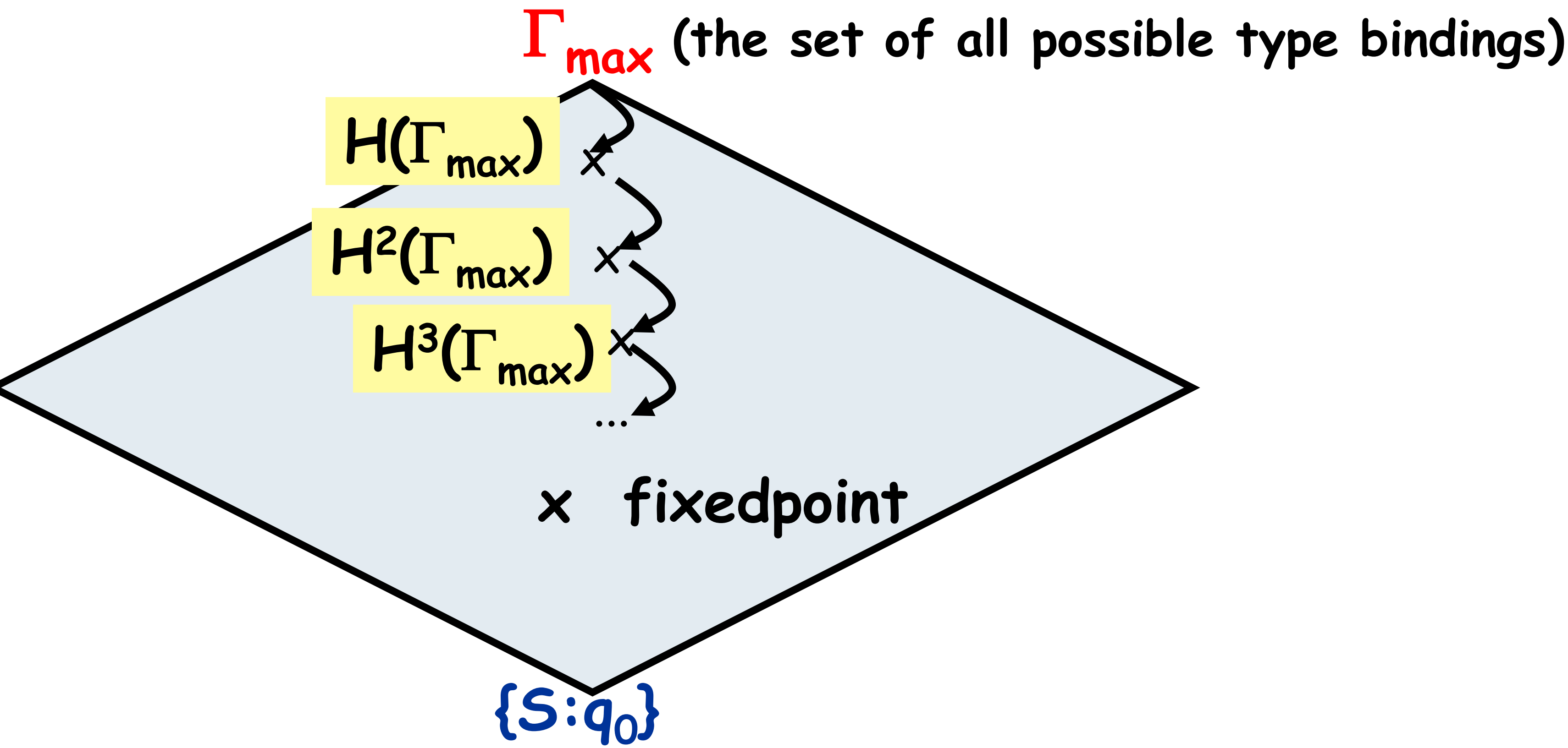
# Type checking (=model checking) problem

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 $\Gamma_{gfp}$ of H
   ($H(\Gamma) = \{ F_j : \tau \in \Gamma \mid \Gamma \vdash t_j : \tau \}$)
2. Check whether $S{:}q_0 \in \Gamma_{gfp}$



# Example

- HORS:

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

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

- Automaton:

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

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

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

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

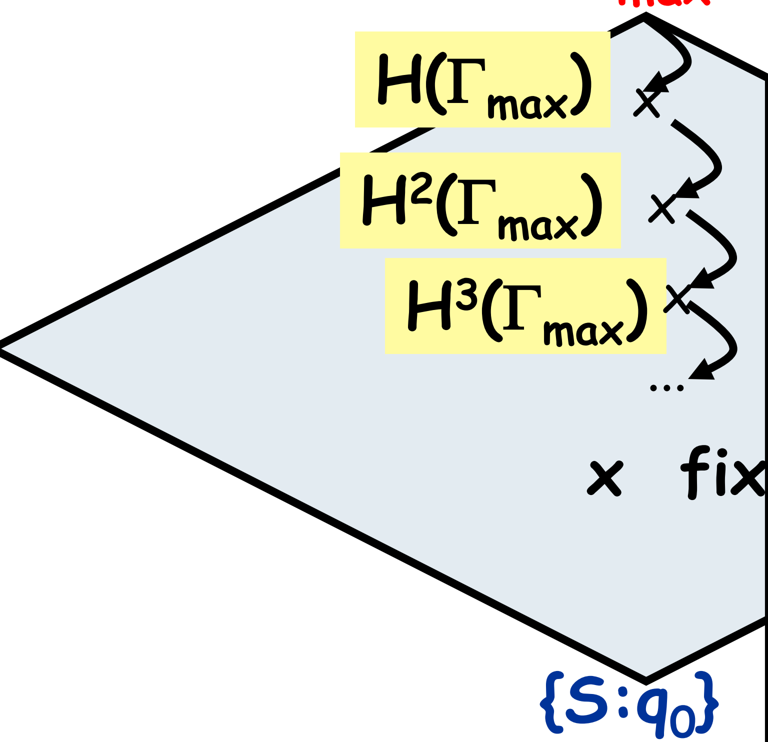
$H^2(\Gamma_{max}) = \{S{:}q_0,\ F{:}\ q_0 \wedge q_1 \to q_0\}$

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

# Naive Algorithm [K. POPL09]

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

$\Gamma_{max}$ (the set of all possible type bindings)




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

(both as huge as $|\Gamma_{max}|$ =

$$O\Big(|G| \times \underbrace{2^{2^{\cdot^{\cdot^{2^{(AQ)^{1+\varepsilon}}}}}}}_{k}\Big)$$

)

k: order of G
A: largest arity
Q: automaton size

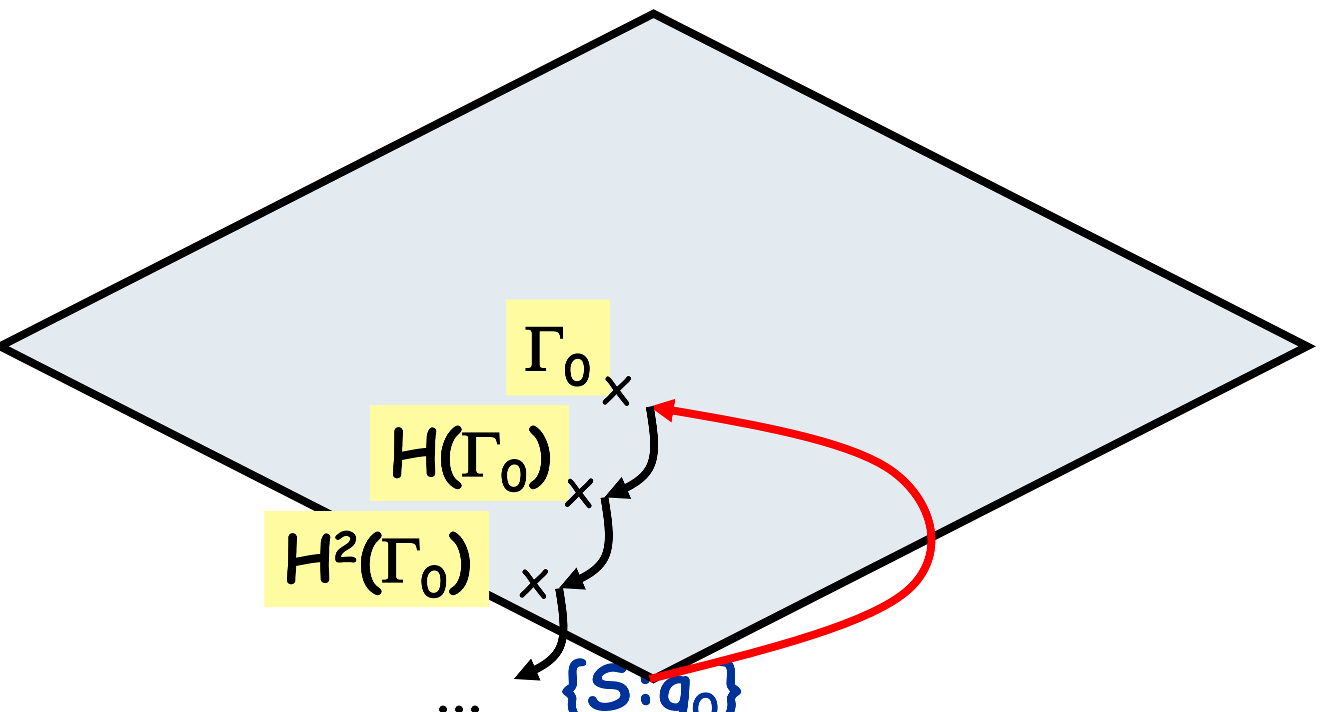
# Outline

- ♦ Introduction [by Ong, 15 minutes]
- ♦ Applications to program verification
  [by Kobayashi, 25 minutes]
- ♦ Type systems and algorithms for higher-order model checking [by Kobayashi, 25 minutes]
  - type-based characterization
  - practical algorithms
    - TRecS
    - HorSat
    - other algorithms
- ♦ Advanced topics [by Ong, 25 minutes]

# Practical Algorithm (TRecS [K. PPDP09])

1. Guess a type environment $\Gamma_0$
2. Compute greatest fixedpoint $\Gamma$ smaller than $\Gamma_0$
3. Check whether $S{:}q_0 \in \Gamma$
4. Repeat 1-3 until the property is proved or refuted.

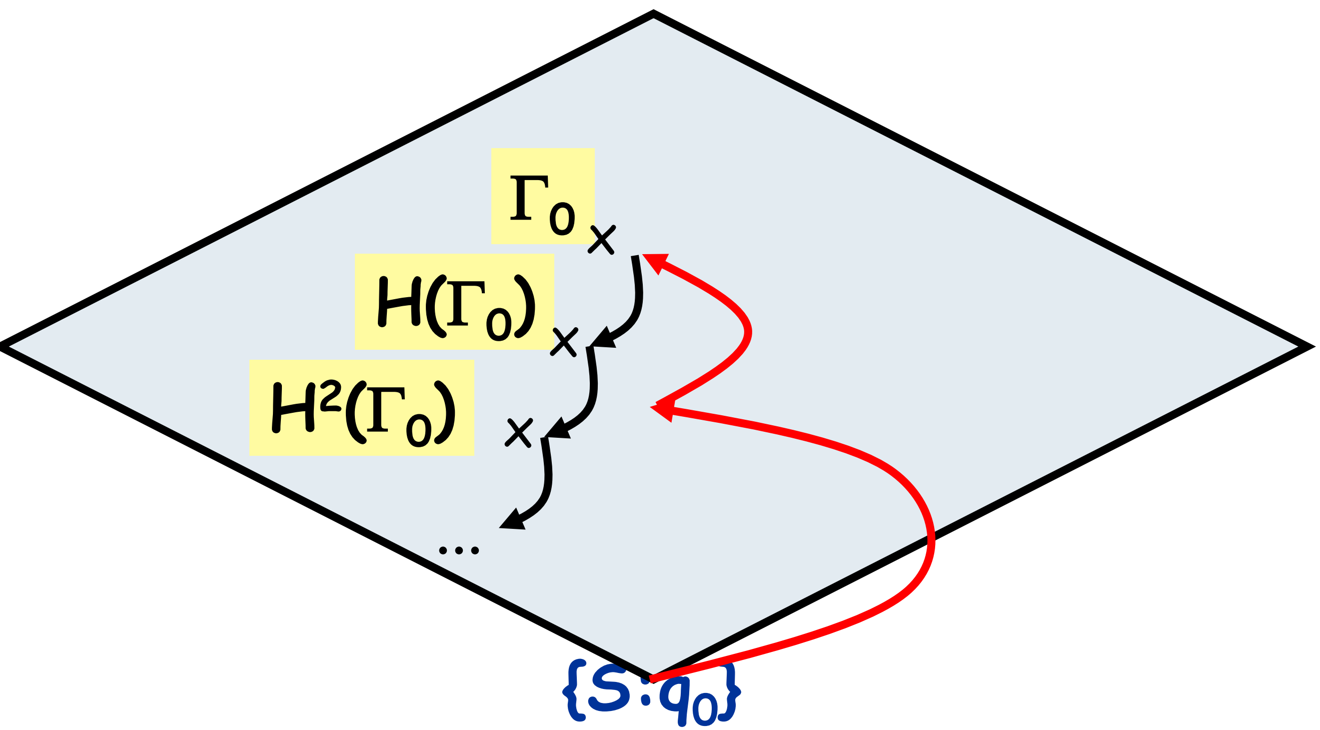
$\Gamma_{max}$ (the set of all possible type bindings)



# Practical Algorithm (TRecS [K. PPDP09])

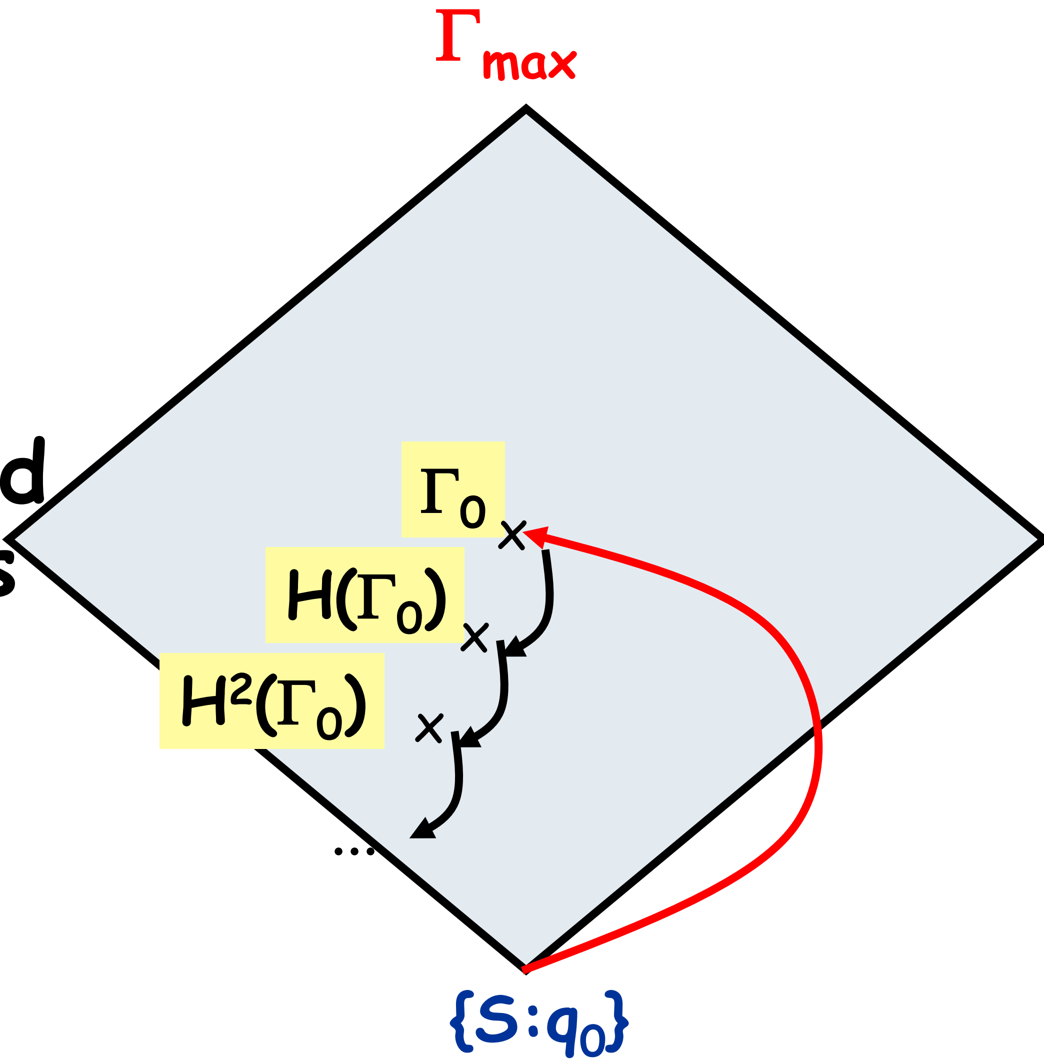
1. Guess a type environment $\Gamma_0$
2. Compute greatest fixedpoint $\Gamma$ smaller than $\Gamma_0$
3. Check whether $S{:}q_0 \in \Gamma$
4. Repeat 1-3 until the property is proved or refuted.

$\Gamma_{max}$ (the set of all possible type bindings)



# How to guess $\Gamma_0$?

- Reduce HORS a finite number of steps
- Observe how each function is used and express it as types



# Example

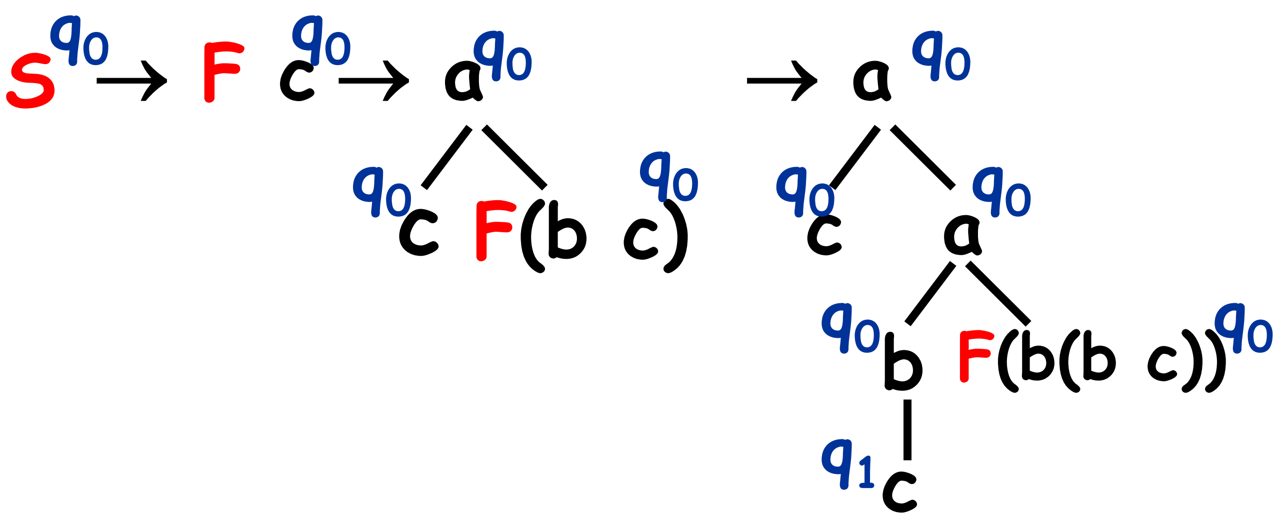
- **HORS:**

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

- **Automaton:**

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

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



# Example

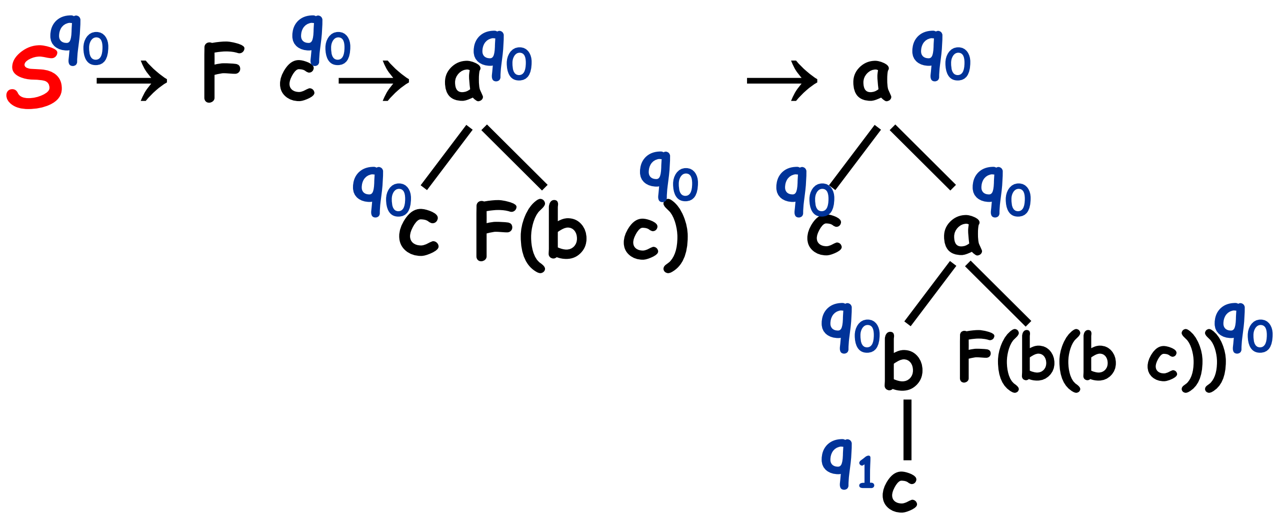
- **HORS:**

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

- **Automaton:**

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

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




$\Gamma_0$ :

S: $q_0$

# Example

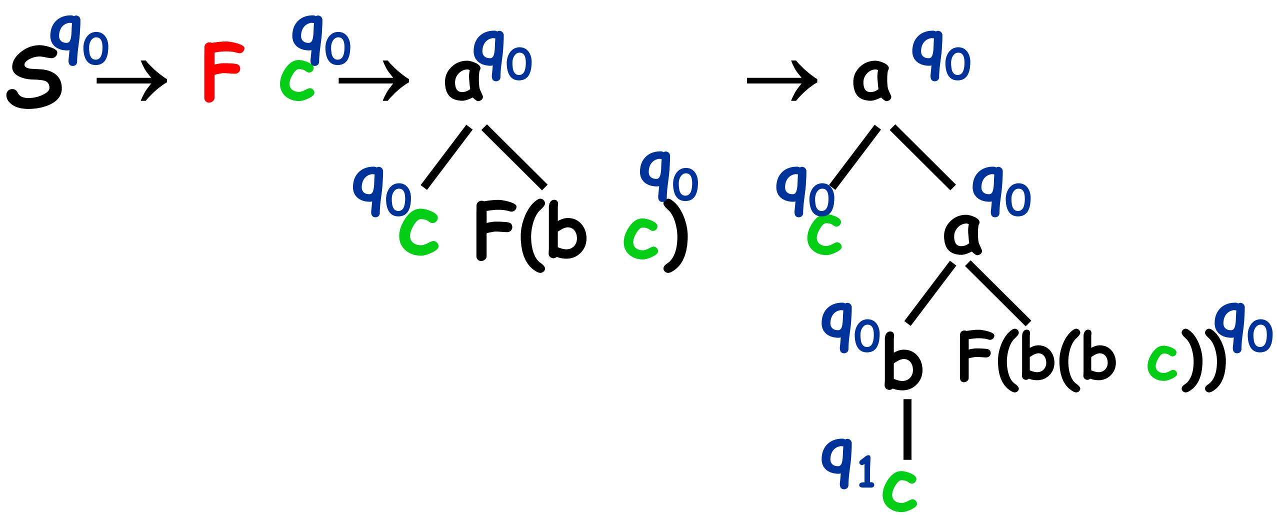
- **HORS:**

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

- **Automaton:**

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

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




$\Gamma_0$ :

$S: q_0$

$F: ? \to q_0$

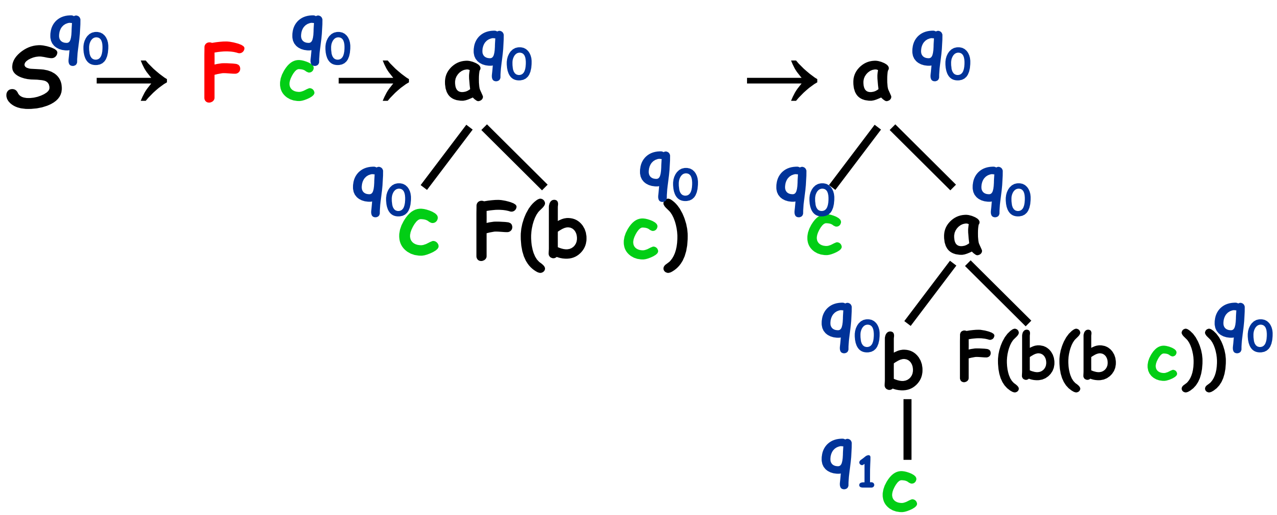
# Example

- **HORS:**

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

- **Automaton:**

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




$\Gamma_0$ :

$S:\ q_0$

$F:\ q_0 \wedge q_1 \to q_0$

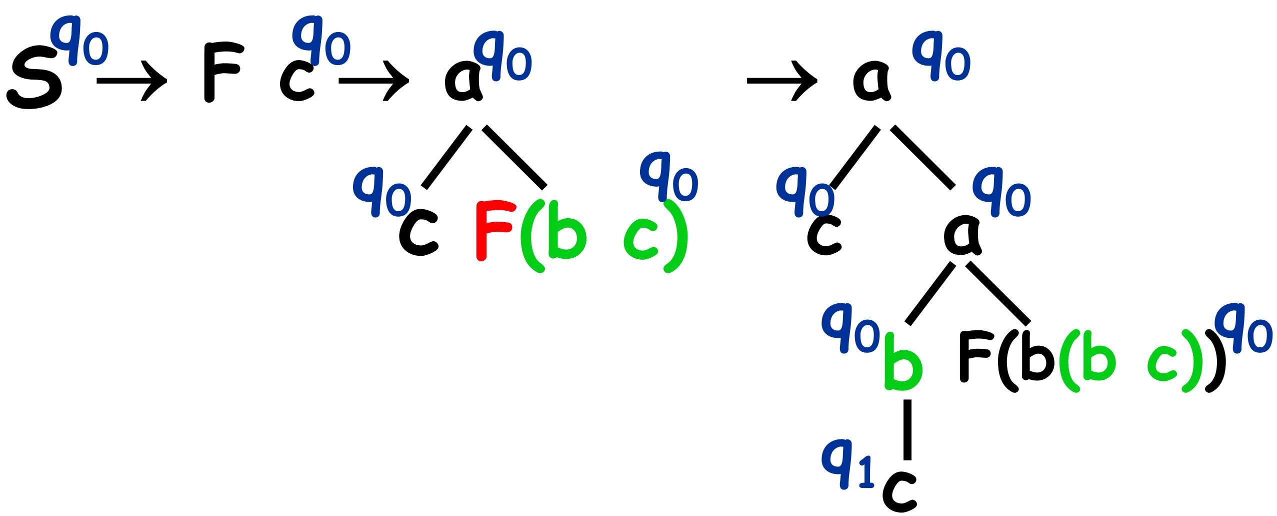
# Example

- **HORS:**

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

- **Automaton:**

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




$\Gamma_0$ :

S: $q_0$

F: $q_0 \wedge q_1 \rightarrow q_0$

F: $q_0 \rightarrow q_0$

# Example

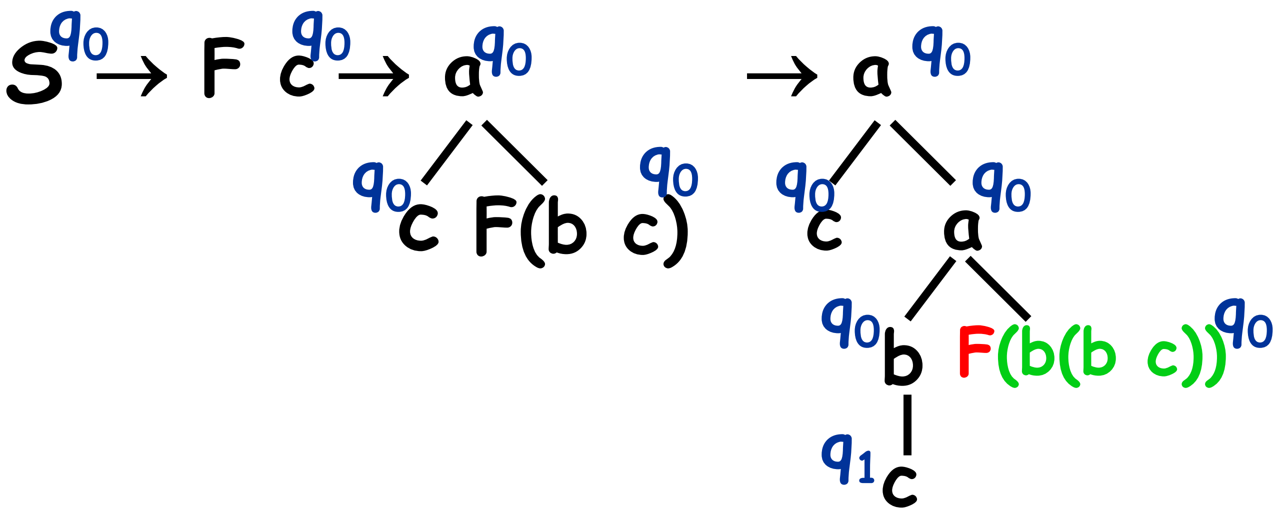
- **HORS:**

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

- **Automaton:**

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

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




$\Gamma_0$ :

$S:\ q_0$

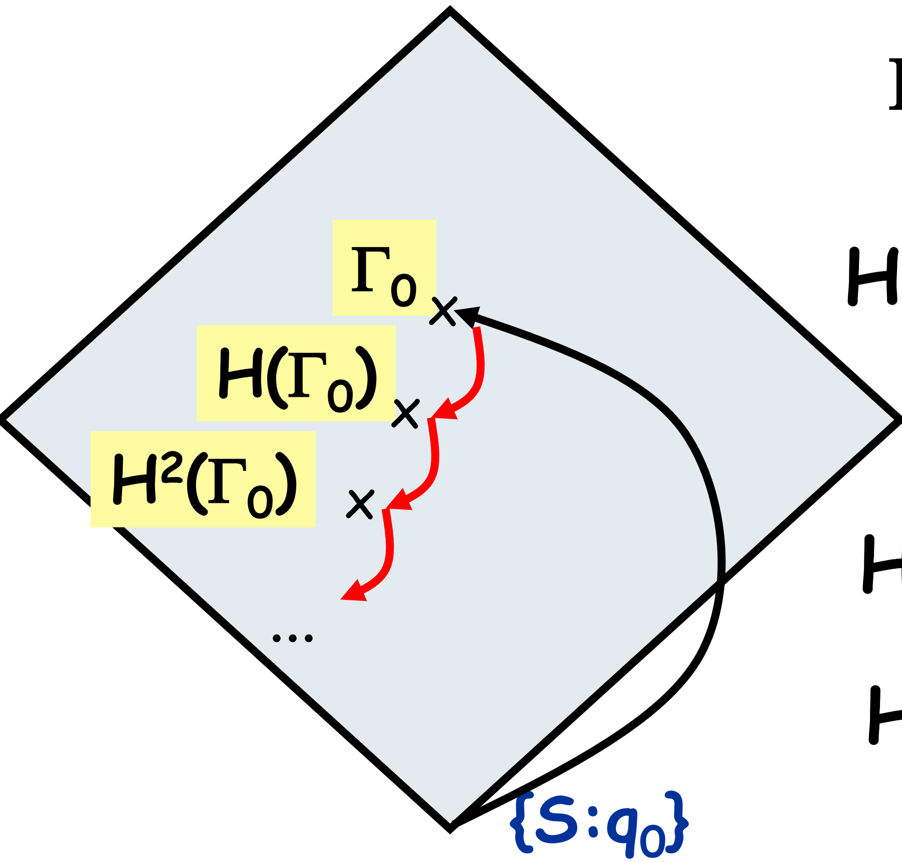
$F:\ q_0 \wedge q_1 \to q_0$

$F:\ q_0 \to q_0$

$F:\ \top \to q_0$

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

1. Guess a type environment $\Gamma_0$
2. Compute greatest fixedpoint $\Gamma$ smaller than $\Gamma_0$
3. Check whether $S{:}q_0 \in \Gamma$
4. Repeat 1-3 until the property is proved or refuted.




$$\Gamma_0 = \{S: q_0,\ F: q_0 \wedge q_1 \rightarrow q_0,\ F: q_0 \rightarrow q_0,\ F: \top \rightarrow q_0\}$$

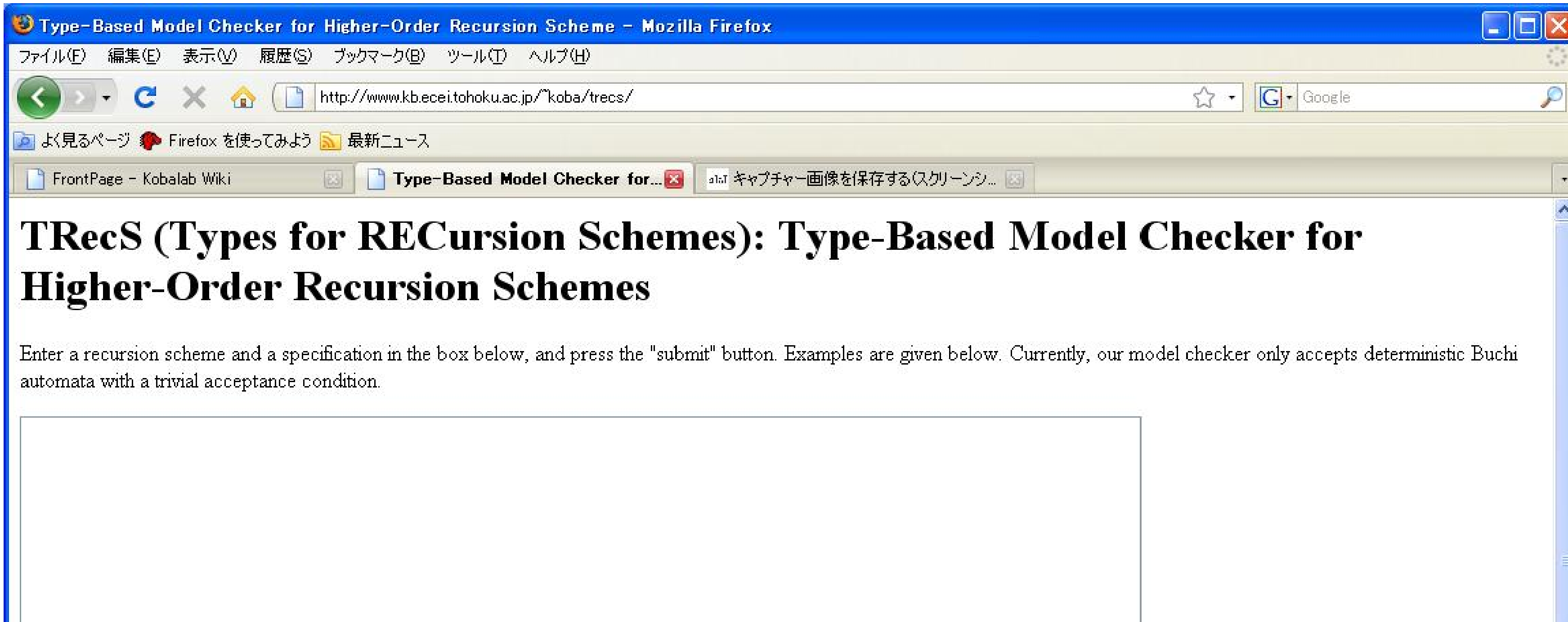
$$H(\Gamma_0) = \{ F_k{:}\tau \in \Gamma_0 \mid \Gamma_0 \vdash t_k{:}\tau \}$$
$$= \{S: q_0,\ F: q_0 \wedge q_1 \rightarrow q_0,\ F: q_0 \rightarrow q_0 \}$$

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

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

# TRecS [K. PPDP09]

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




- The first model checker for HORS
- Used as a backend of MoCHi [K+11,Sato+13]

# Outline

- Introduction [by Ong, 15 minutes]
- Applications to program verification
  [by Kobayashi, 25 minutes]
- Type systems and algorithms for higher-order model checking [by Kobayashi, 25 minutes]
  - type-based characterization
  - practical algorithms
    - TRecS
    - HorSat
    - other algorithms
- Advanced topics [by Ong, 25 minutes]

# HorSat algorithm [Broadbent&K, CSL13]

- Basis of the state-of-the-art HO model checker HorSat2 (http://www-kb.is.s.u-tokyo.ac.jp/~koba/horsat2)
- Based on the "dual" type system
  - use the complement of property automaton *A*, to characterize invalid trees
  - least fixed-point computation instead of greatest

# Yet another characterization of HO model checking

- $G$: HORS, $A$: trivial tree automaton

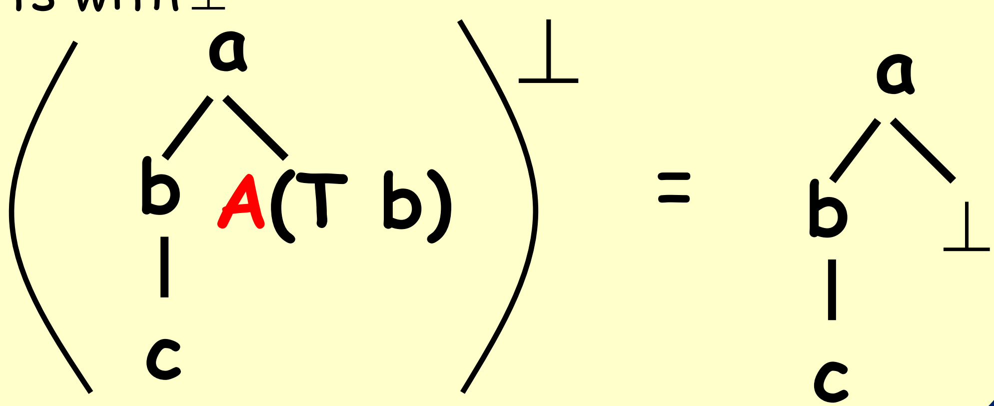
$$\mathrm{Tree}(G) \in \mathrm{Lang}(A)$$

iff $\quad S \notin \mathrm{Pre}^*(\mathrm{Error})$ where:

$$\mathrm{Pre}^*(t) = \{s \mid s \rightarrow_G^* t\}$$

$$\mathrm{Error} = \{t \mid t^{\perp} \in \mathrm{Lang}(\overline{A})\}$$

Tree obtained by replacing non-tree parts with $\perp$



# Yet another characterization of HO model checking

- $G$: HORS, $A$: trivial tree automaton

$$\mathrm{Tree}(G) \in \mathrm{Lang}(A)$$

iff $S \notin \mathrm{Pre}^*(\mathrm{Error})$ where:

$$\mathrm{Pre}^*(t) = \{ s \mid s \rightarrow_G^* t \}$$

$$\mathrm{Error} = \{ t \mid t^{\perp} \in \mathrm{Lang}(\overline{A}) \}$$

Pre*(Error) may be infinite,
but can be finitely represented (and computed)
by using intersection types:

$$\mathrm{Pre}^*(\mathrm{Error}) = \{ t \mid \mathrm{lfp}(\mathrm{Pre}_{TE}) \vdash t : q_0 \}$$

where $\mathrm{Pre}_{TE}(\Gamma) = \{ F : \sigma_1 \rightarrow \ldots \rightarrow \sigma_n \rightarrow q \mid$

$F\ x_1 \ \ldots\ x_n \rightarrow t \in G$ and $\Gamma, x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash t : q \}$

# Other HO model checking algorithms

- GTRecS [K 11]
  - first fixed-parameter linear time algorithm
  - collect type candidates like TRecS, but avoid reductions by using game-semantic interpretation of types
- C-SHORe [Broadbent+ 13]
  - based on CPDS; the only practical algorithm not based on types
- Preface [Ramsay+ 14]
  - abstract interpretation of HORS, with type-based refinement using (TRecS-style) positive types and (HorSat-style) negative types
- Thors [Lester+ 11], APTRecS [Fujima+ 13]
  - extend TRecS-style algorithm for liveness properties
- HorSatP [Fujima 15]
  - extend Horsat-style algorithm for liveness properties

# Why HO Model Checking Works? (despite k-EXPTIME completeness)

- Fixed-parameter polynomial time in the size of grammars:

$$O(|G| \times \underbrace{2^{2^{\cdot^{\cdot^{2^{(a\,Q)^{1+\varepsilon}}}}}}}_{k})$$

k: order of G
a: largest arity
Q: automaton size

for trivial automata model checking of HORS

# Why HO Model Checking Works? (despite k-EXPTIME completeness)

- Fixed-parameter polynomial time in the size of grammars
- Type environment serves as a "certificate", which can be checked in polynomial time (cf. NP problems)
- For finite-state models, HO model checking can actually be faster than finite state model checking
  - HORS can compactly represent finite-state systems
    - An order-k HORS of size x can represent a system with states $\left.2^{2^{\cdot^{\cdot^{2^{p(x)}}}}}\right\}k$
  - k-EXPTIME algorithm for HO model checking ≈ PTIME algorithm for finite-state model checking

# Why HO Model Checking Works? (despite k-EXPTIME completeness)

- Fixed-parameter polynomial time in the size of grammars
- Type environment serves as a “certificate”, which can be checked in polynomial time
 (cf. NP problems)
- For finite-state models, HO model checking can actually be faster than finite state model checking
  - HORS can compactly represent finite-state systems
    - An order-k HORS of size x can represent a system with states $\left.2^{2^{\cdot^{\cdot^{\cdot^{2^{p(x)}}}}}}\right\}k$
  - (fixed-parameter) PTIME algorithm for HO model checking
    >> PTIME algorithm for finite-state model checking

# References on Part 3

- Type-based characterization of HO model checking
  - Naoki Kobayashi: Model checking higher-order programs. J. ACM 60(3): 20 (2013)
  - Naoki Kobayashi and Luke Ong: A type system equivalent to the modal mu-calculus model checking of higher-order recursion schemes, LICS 2009
- HO model checking algorithms
  - JACM paper above (for TRecS algorithm)
  - Christopher Broadbent and Naoki Kobayashi, Saturation-Based Model Checking of Higher-Order Recursion Schemes, CSL 13 (for HorSat algorithm)
  - Steven J. Ramsay, Robin P. Neatherway, Luke Ong, A type-based abstraction refinement approach to higher-order model checking, POPL 2014 (for Preface algorithm)

# Outline

- Introduction [by Ong, 15 minutes]
- Applications to program verification
  [by Kobayashi, 25 minutes]
- Type systems and algorithms for higher-order model checking [by Kobayashi, 25 minutes]
- Advanced topics [by Ong, 25 minutes]

# Advertisement

- We are looking for
  - a postdoc
  - PhD students

  to work in our project on HO model checking at University of Tokyo.

  Interested candidates should contact Naoki Kobayashi.