10 Years of the Higher-Order Model Checking Project (at UTokyo)

Naoki Kobayashi
The University of Tokyo

Thanks to numerous collaborators:
Kazuyuki Asada, Atsushi Igarashi, Etienne Lozes, Luke Ong, Ryosuke Sato, Ayumi Shinohara, Takeshi Tsukada, Hiroshi Unno, (ex-)students at UTokyo and Tohoku University, ...

This Talk

- **Summary of the Higher-Order Model Checking (HOMC) Project at UTokyo, which started in 2009, following the two papers:**

POPL 2009

PPDP 2009

Types and Higher-Order Recursion Schemes for Verification of Higher-Order Programs

Naoki Kobayashi

Model-Checking Higher-Order Functions

Model Checking Higher-Order Programs

JACM 2013

NAOKI KOBAYASHI, The University of Tokyo

We propose a novel verification method for higher-order functional programs based on higher-order model checking, or more precisely, model checking of higher-order recursion schemes (recursion schemes, for short). The most distinguishing feature of our verification method for higher-order programs is that it is sound, complete, and automatic for the simply typed λ-calculus with recursion and finite base types, and for various program verification problems such as reachability, flow analysis, and resource usage verification. We first show that a variety of program verification problems can be reduced to model checking problems for recursion schemes, by transforming a program into a recursion scheme that generates a tree representing all the interesting possible event sequences of the program. We then develop a new type-based model-checking algorithm for recursion schemes and implement a prototype recursion scheme model checker. To our knowledge, this is the first implementation of a recursion scheme model checker. Experiments show that our model

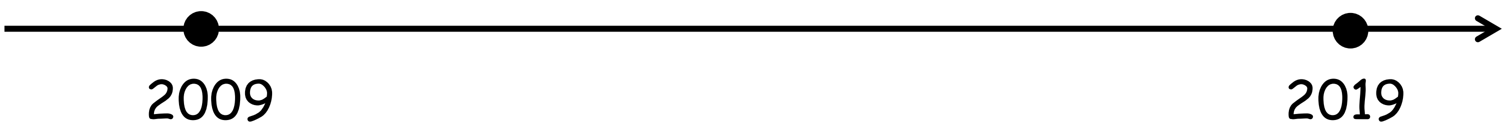
Tool demonstration: MoChi

[K&Sato&Unno, PLDI 2011]

(a software model checker for a subset of functional programming language OCaml)

Outline

- **What is Higher-Order Model Checking?**
- **History of the Project**
  - **... with (hopefully) gentle introduction to foundations, algorithms and applications of higher-order model checking**



- **Conclusion**

Two Notions of Higher-Order Model Checking

	Models	Logic
finite state model checking	finite state systems	modal μ-calculus (or LTL, CTL, ...)

Two Notions of Higher-Order Model Checking

	Models	Logic
finite state model checking	finite state systems	modal μ-calculus
HORS model checking [Knapik+ 01; Ong 06]	higher-order recursion schemes (HORS)	modal μ-calculus

Higher-order tree grammars,
useful for modeling a certain class of
infinite state systems
(such as higher-order functional programs)

Two Notions of Higher-Order Model Checking

	Models	Logic
finite state model checking	finite state systems	modal μ-calculus
HORS model checking [Knapik+ 01; Ong 06]	higher-order recursion schemes (HORS)	modal μ-calculus
HFL model checking [Viswanathan& Viswanathan 04]	finite state systems	higher-order modal fixpoint logic (HFL)

Useful for describing non-regular properties

Two Notions of Higher-Order Model Checking

	Models	Logic
finite state model checking	finite state systems	modal μ-calculus
HORS model checking [Knapik+ 01; Ong 06]	higher-order recursion schemes (HORS)	modal μ-calculus
HFL model checking [Viswanathan& Viswanathan 04]	finite state systems	higher-order modal fixpoint logic (HFL)

Higher-Order Recursion Scheme (HORS)

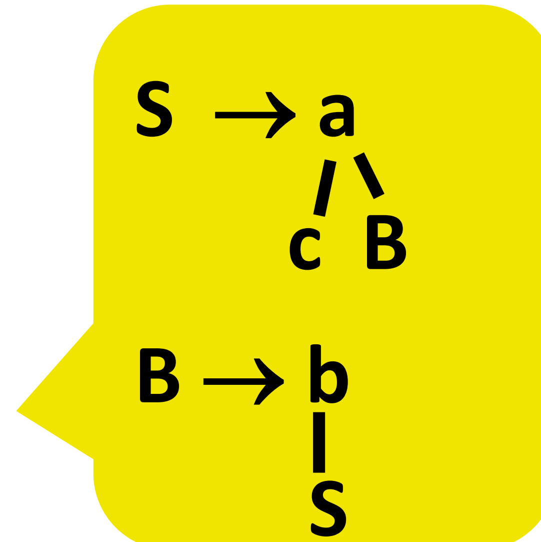
- Grammar for generating an infinite tree

Order-0 HORS

(regular tree grammar)

S → a c B

B → b S



Higher-Order Recursion Scheme (HORS)

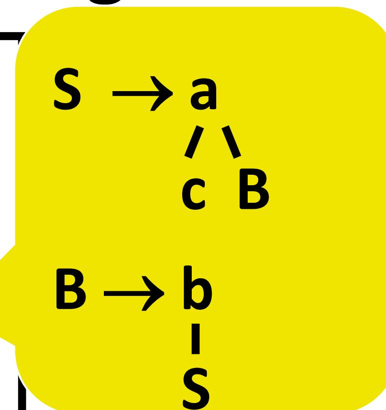
- Grammar for generating an infinite tree

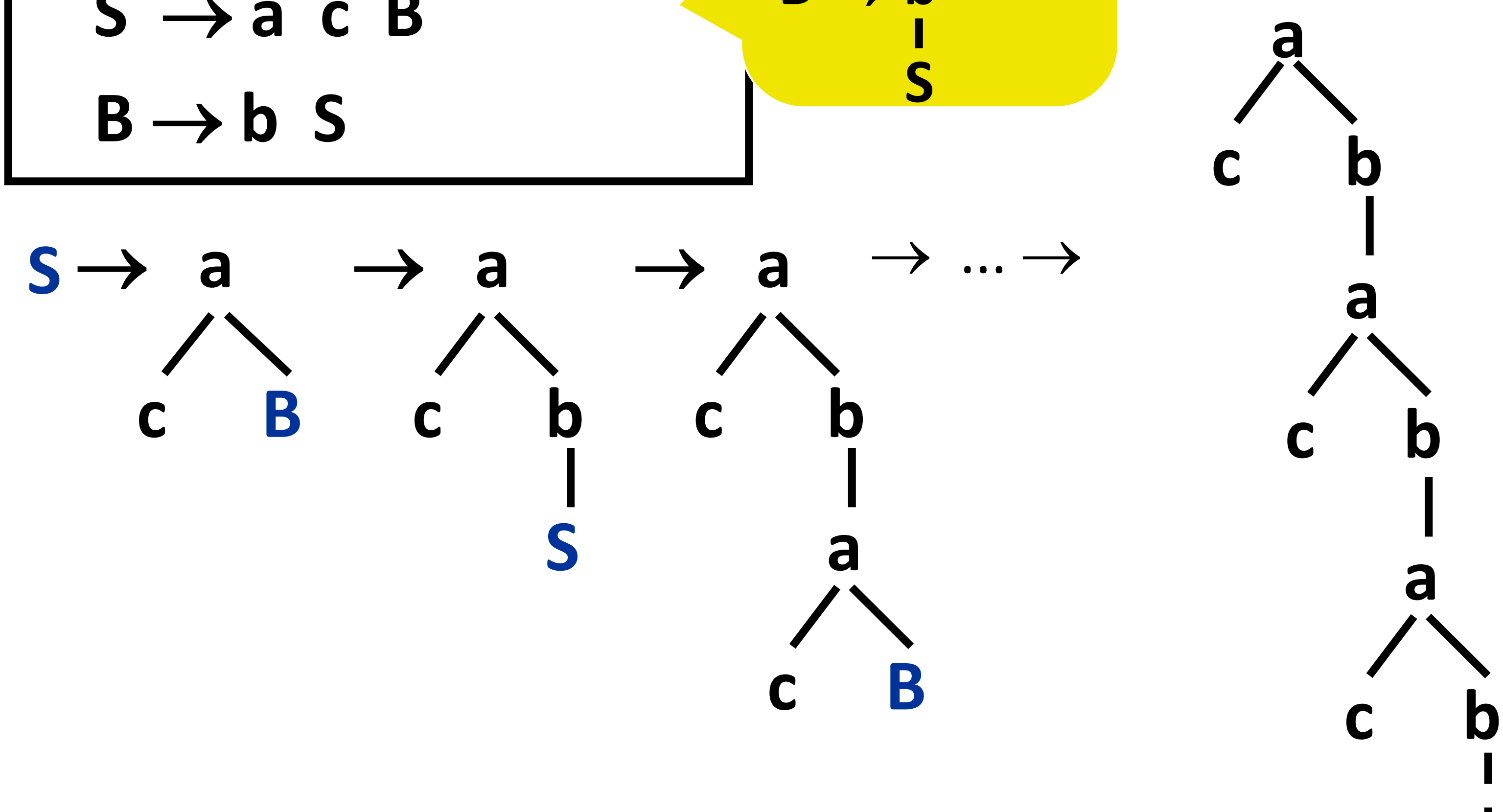
Order-0 HORS

(regular tree grammar)

$S \rightarrow a\ c\ B$

$B \rightarrow b\ S$





Higher-Order Recursion Scheme (HORS)

- Grammar for generating an infinite tree

Order-1 HORS

$$S \rightarrow A\ c$$

$$A\ x \rightarrow a\ x\ (A\ (b\ x))$$

$S: o,\ A: o \rightarrow o$

Key restrictions on rewriting rules:
- Rules must be simply-typed.
- There are no pattern matching on trees.

Higher-Order Recursion Scheme (HORS)

- Grammar for generating an infinite tree

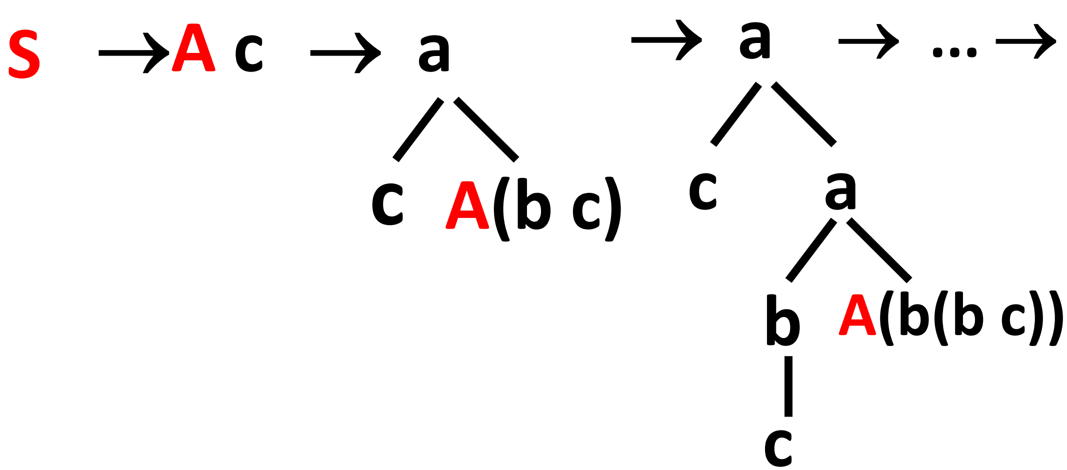
Tree whose paths are labeled by $a^{m+1}\, b^{m}\, c$

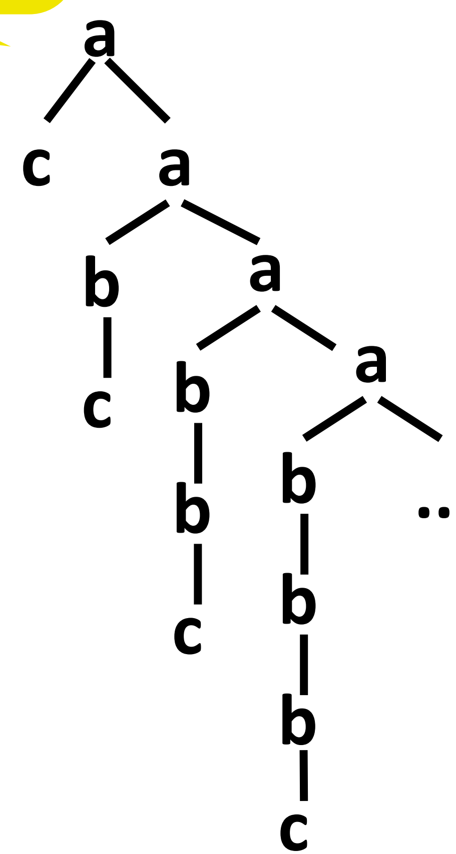
Order-1 HORS

$$S \;\rightarrow A\ c$$

$$A\ x \rightarrow a\ \ x\ \ (A\ (b\ x))$$

S: o, A: o$\rightarrow$ o





Higher-Order Recursion Scheme (HORS)

- **Grammar for generating an infinite tree**

Order-1 HORS

$S \rightarrow A\ c$

$A\ x \rightarrow a\ \ x\ \ (A\ (b\ x))$

S: o, A: o→ o

HORS

≈

A simply-typed functional program for generating a tree

HORS Model Checking

Given

G: HORS

φ: a formula of modal μ-calculus

(or a tree automaton),

does Tree(G) satisfy φ?

e.g.

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

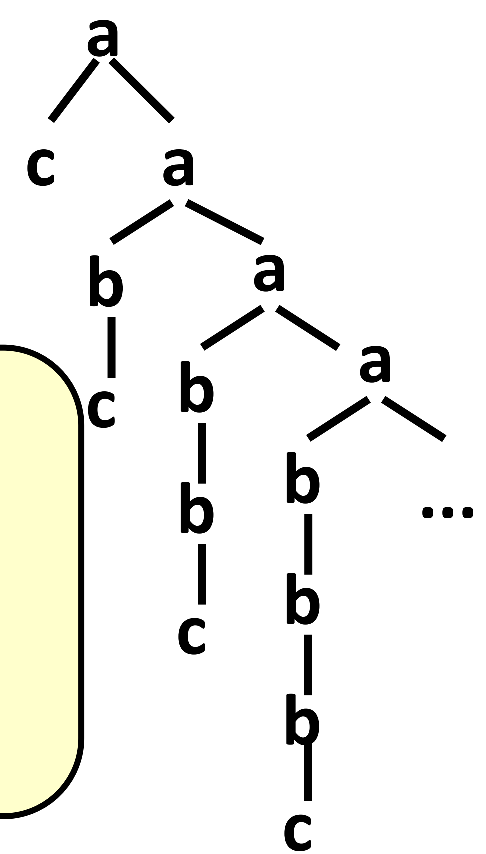
HORS Model Checking

Order-1 HORS

$S \rightarrow A\ c$

$A\ x \rightarrow a\ \ x\ \ (A\ (b\ x))$

S: o, A: o→ o



Q1. Does every finite path end with "c"?

YES!

Q2. Does "a" occur below "b"?

NO!

HORS Model Checking

Given
G: HORS
φ: a formula of modal μ-calculus
(or a tree automaton),
does Tree(G) satisfy φ?

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

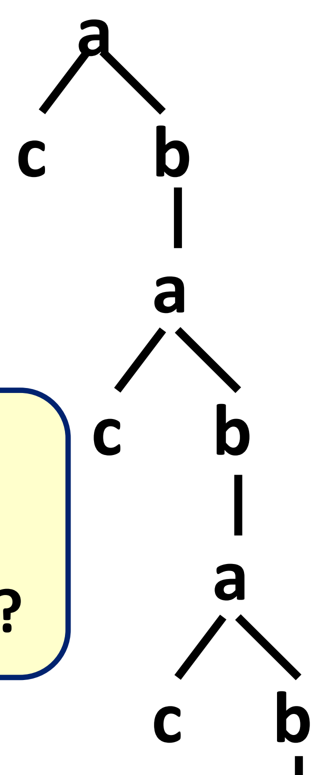
k-EXPTIME-complete [Ong, LICS06]
(for order-k HORS)

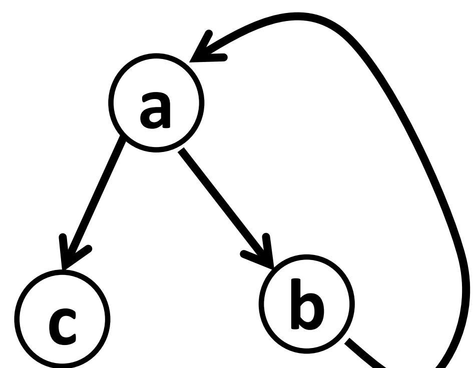
$$k\left\{2^{2^{\cdot^{\cdot^{2^{p(x)}}}}}\right.$$

HORS Model Checking as Generalization of Finite State/Pushdown Model Checking

- order-0 ≈ finite state model checking
- order-1 ≈ pushdown model checking

infinite tree ≈ transition system





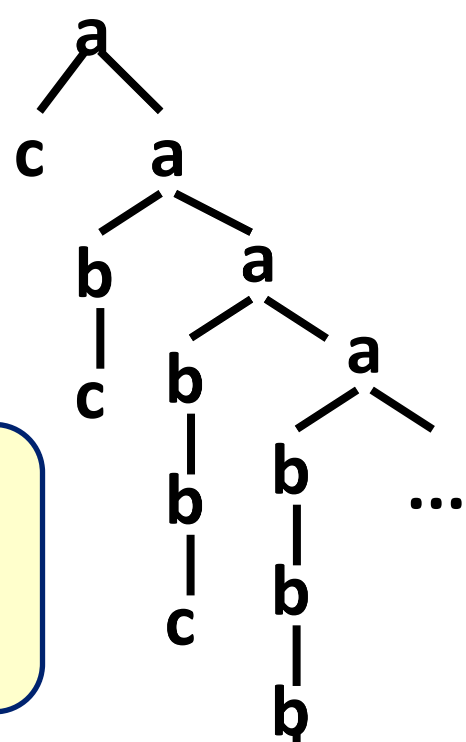
Does “a” occur below “b”?

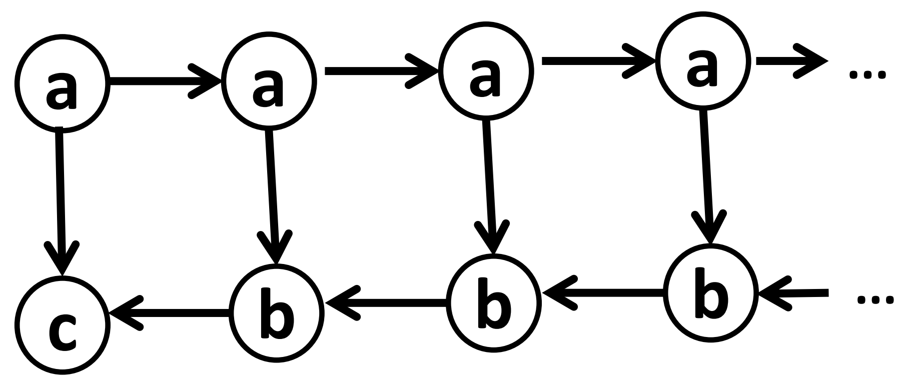
Is there a transition sequence in which “a” occurs after “b”?

HORS Model Checking as Generalization of Finite State/Pushdown Model Checking

- order-0 ≈ finite state model checking
- order-1 ≈ pushdown model checking

infinite tree ≈ (infinite-state) transition system





Does "a" occur below "b"?

Is there a transition sequence in which "a" occurs after "b"?

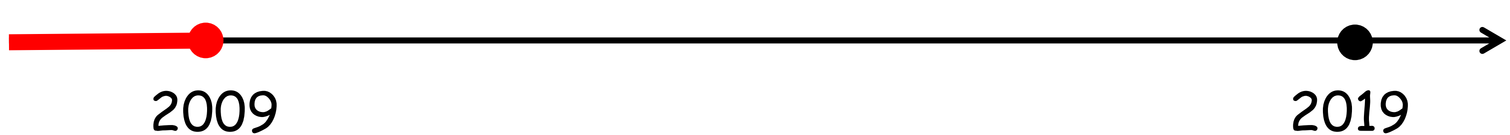
Outline

- What is Higher-Order Model Checking?
- History of the Project
  - start of the project (through 2009)
    - application to program verification [POPL09]
    - type-theoretic foundation [POPL09]
    - practical algorithm [PPDP09]
  - tool development and quest for better algorithms and more foundations (2010-2016)
  - shift to HFL model checking (2017-)

- Conclusion

Background of the Project

- I attended two talks by Luke Ong on HORS model checking
  - IFIP WG 2.2 meeting in 2007
    “Theoretically interesting, but ...”
  - FoSSaCS 2008 invited talk
    “Maybe useful for program verification?”



Background of the Project

- I attended two talks by Luke Ong on HORS model checking
  - IFIP WG 2.2 meeting in 2007
  - FoSSaCS 2008 invited talk
- I was working with Atsushi Igarashi on resource usage analysis [Igarashi&K, POPL02]

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

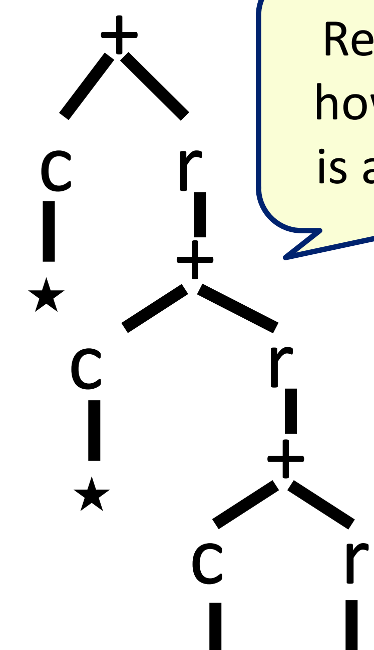
Is the file “foo” accessed according to read* close?

From Program Verification to HORS Model Checking

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

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

$S \rightarrow F\ d\ \star$



Represents how the file is accessed.

Is the file “foo” accessed according to read* close?

⇛

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

Fro

to

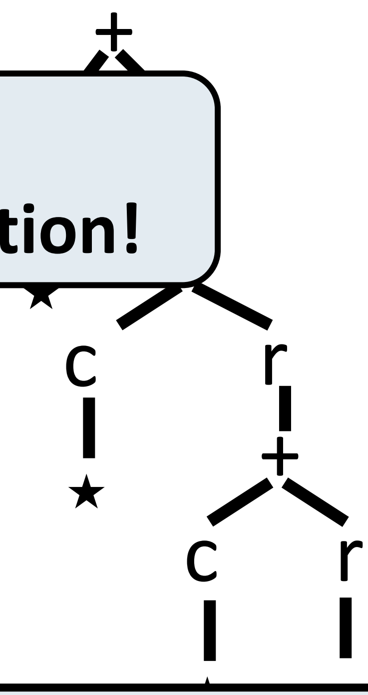
continuation parameter, expressing how "foo" is accessed after the call returns

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

F x k → + (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?

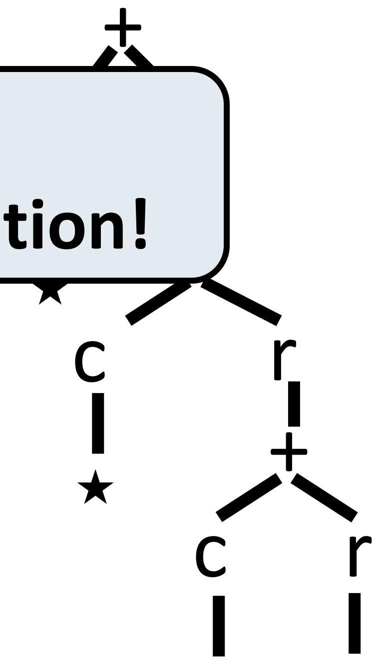
From Program Verification to HORS Model Checking

```
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?

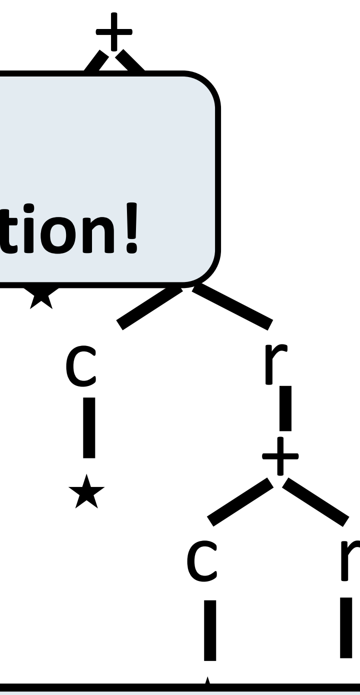
From Program Verification to HORS Model Checking

```
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?

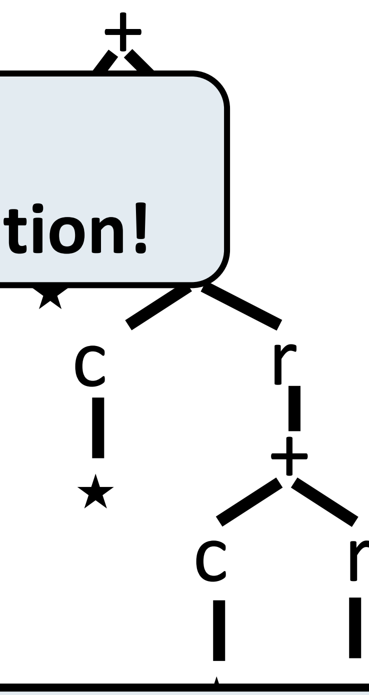
From Program Verification to HORS Model Checking

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

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

$S \rightarrow F\ d\ \bigstar$

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 HORS Model Checking

```
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 HORS Model Checking

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

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

$S \rightarrow F\ d\ \bigstar$

$F\ d\ \bigstar$

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

⟹

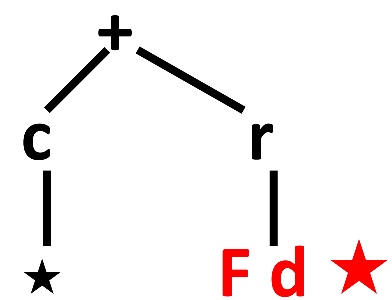
Is each path of the tree labeled by r*c?

From Program Verification to HORS Model Checking

```
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?

⟹

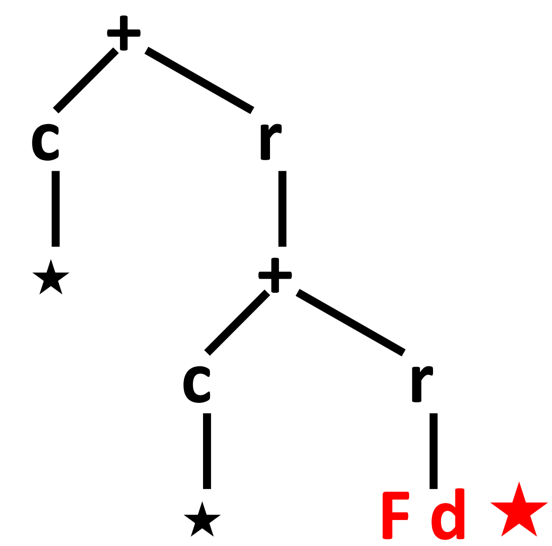
Is each path of the tree labeled by r*c?

From Program Verification to HORS Model Checking

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

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

$S \rightarrow F\ d\ \bigstar$



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

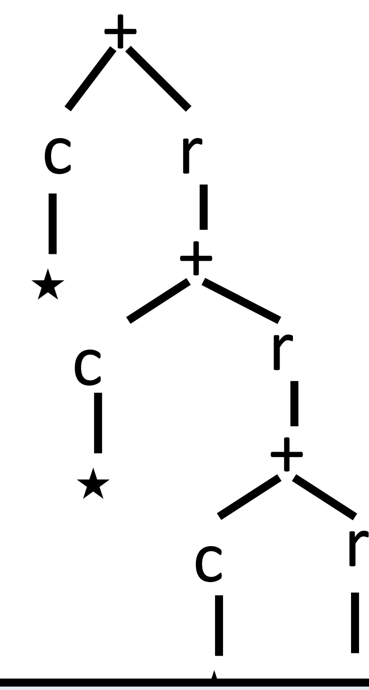
Is each path of the tree labeled by r*c?

From Program Verification to HORS Model Checking

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

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

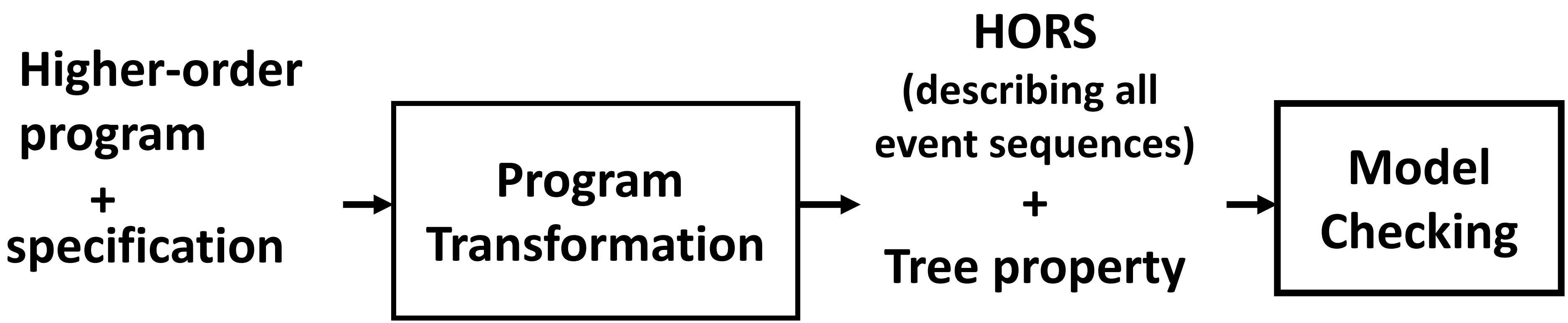
$$S \rightarrow F\ d\ \star$$



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

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

From Program Verification to HORS Model Checking



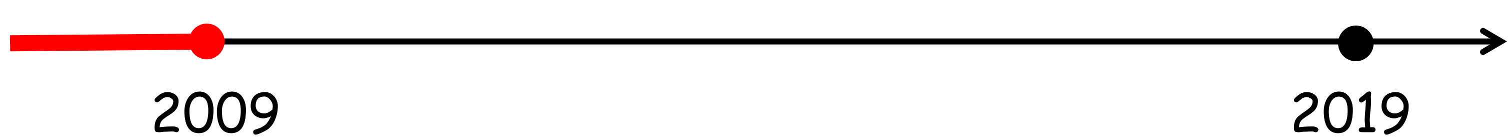
Sound, complete, and automatic for:

- 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 HORS Model Checking

Higher-order program + specification → **Program Transformation** → HORS (describing all event sequences) + Tree property → **Model Checking**

For finite-data HO programs,
automated verification comes for free
from HORS model checking!
But ...
is HORS model checking feasible in practice?
(recall: HORS model checking is k-EXPTIME complete)



How to solve HORS MC problems?

On model-checking trees generated by higher-order recursion schemes

[Ong, LICS 2006]

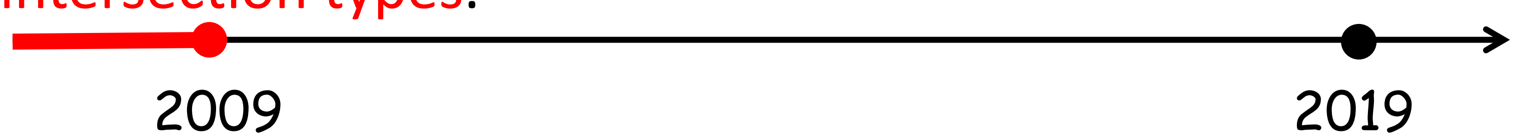
C.-H. L. Ong*
Oxford University Computing Laboratory

Abstract

We prove that the modal mu-calculus model-checking problem for (ranked and ordered) node-labelled trees that are generated by order-n recursion schemes (whether safe or not, and whether homogeneously typed or not) is n-EXPTIME complete, for every $n \geq 0$. It follows that the

Shupp [13] proved that the *configuration graphs of push-down systems* have decidable MSO theories. In the 90's, as finite-state technologies matured, researchers embraced the challenges of software verification. A highlight from this period was Caucal's result [5] that *prefix-recognizable graphs* have decidable MSO theories. In 2002 a flurry of discoveries significantly extended and unified earlier devel-

- The decidability proof (in a 55 page paper) was based on game semantics.
- The proof included an algorithm, which always suffers from k-EXPTIME bottleneck.
- The key notion of "variable profiles" reminded me of intersection types.



Outline

- ♦ What is Higher-Order Model Checking?
- ♦ History of the Project
  - – start of the project (through 2009)
    - • application to program verification [POPL09]
    - • type-theoretic foundation [POPL09]
    - • practical algorithm [PPDP09]
  - – tool development and quest for better algorithms and more foundations (2010-2016)
  - – shift to HFL model checking (2017-)

- ♦ Conclusion

Type-Theoretic Approach to HORS Model Checking [K, POPL09][K&Ong, LICS09]

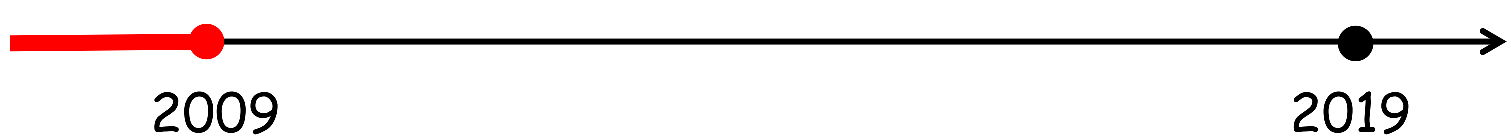
Construct a type system TS(A) s.t.

Tree(G) is accepted by tree automaton A

if and only if

G is typable in TS(A)

> **cf. “Model Checking as Type Checking”**
> **[Naik & Palsberg, ESOP2005]**



HORS Model Checking Problem: Restricted version

Given

G: HORS

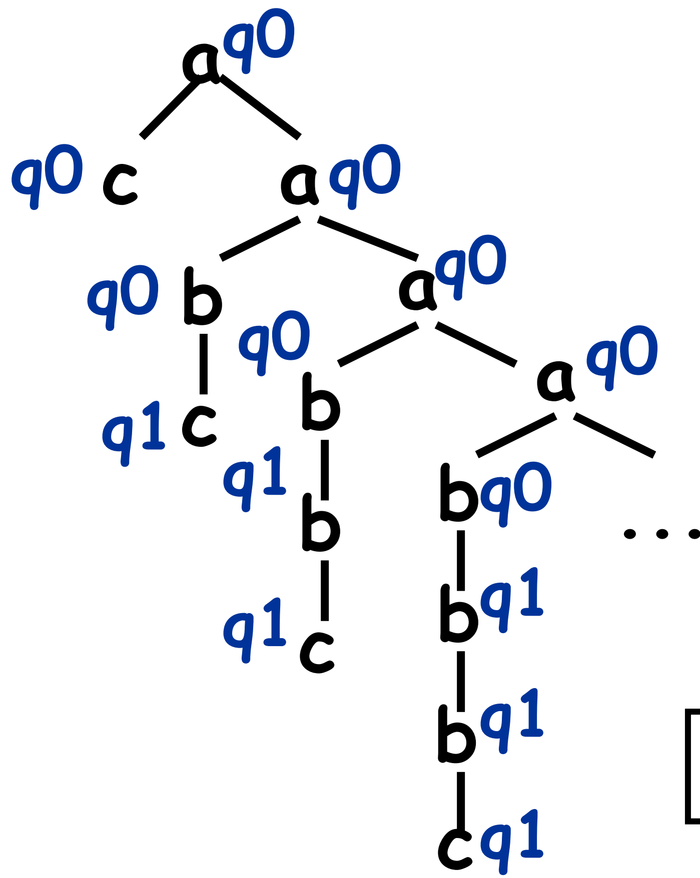
A: trivial automaton [Aehlig CSL06]

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

does A accept Tree(G)?

k-EXPTIME-complete [K&Ong, ICALP09]
(for order-k HORS)

Trivial tree automaton for infinite trees

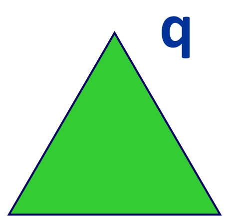


$\delta(q0,\ a) = q0\ q0$
$\delta(q0,\ b) = q1$
$\delta(q1,\ b) = q1$
$\delta(q0,\ c) = \varepsilon$
$\delta(q1,\ c) = \varepsilon$

"a" does not occur below "b"

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?

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
i.e. $\exists\Gamma.(S: q_0 \in \Gamma \wedge \forall(F_k{:}\tau)\in \Gamma.\ \Gamma \vdash t_k : \tau)$
($G = \{F_1 \rightarrow t_1, \ldots, F_m \rightarrow t_m\}$ with $S=F_1$; A: Trivial automaton with initial state q_0)

Type environment for non-terminals $F_1,\ldots,F_m$

Consequences:

- **Straightforward algorithm, which runs in time *linear* in |G| (if the other parameters are fixed):**

$\Gamma := \Gamma_{max}$ (all the possible typings for non-terminals)
repeat $\Gamma := \text{Shrink}(\Gamma)$ until $\Gamma = \text{Shrink}(\Gamma)$
return $(S: q_0 \in \Gamma)$

$\text{Shrink}(\Gamma) = \{F_k{:}\tau \in \Gamma \mid \Gamma \vdash t_k : \tau\}$
filters out invalid typings

Soundness and Completeness [K., POPL2009]

Tree(G) is accepted by A
if and only if
S has type q_0
i.e. $\exists\Gamma.(S{:}q_0 \in \Gamma \wedge \forall(F_k{:}\tau)\in\Gamma.\ \Gamma \vdash t_k : \tau)$

($G = \{F_1 \to t_1, \ldots, F_m \to t_m\}$ with $S=F_1$; A: Trivial automaton with initial state q_0)

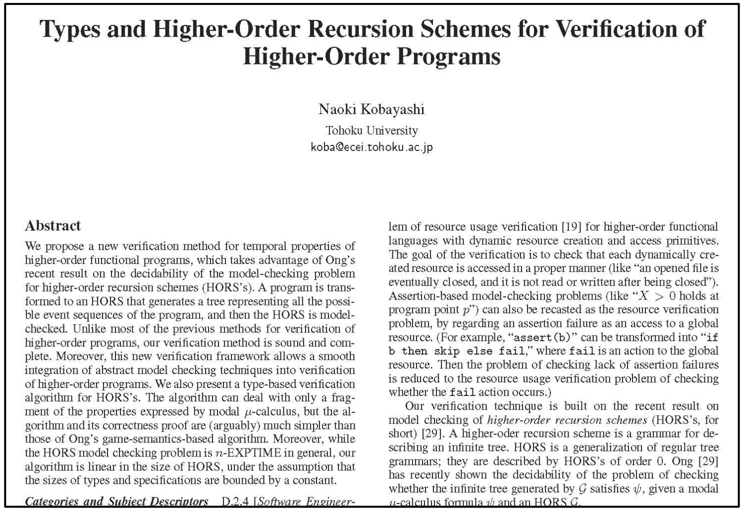
Consequences:

- **Straightforward algorithm, which runs in time *linear* in |G| (if certain parameters are fixed):**

  $\Gamma := \Gamma_{max}$ (all the possible typings for non-terminals)
  repeat Γ := Shrink(Γ) until Γ= Shrink(Γ)
  return (S: $q_0 \in \Gamma$)

- **Γ serves as a certificate, which can be checked efficiently (cf. NP problems)**

Summary of POPL 09 Paper



Types and Higher-Order Recursion Schemes for Verification of Higher-Order Programs

Naoki Kobayashi
Tohoku University
koba@ecei.tohoku.ac.jp

Abstract

We propose a new verification method for temporal properties of higher-order functional programs, which takes advantage of Ong's recent result on the decidability of the model-checking problem for higher-order recursion schemes (HORS's). A program is transformed to an HORS that generates a tree representing all the possible event sequences of the program, and then the HORS is model-checked. Unlike most of the previous methods for verification of higher-order programs, our verification method is sound and complete. Moreover, this new verification framework allows a smooth integration of abstract model checking techniques into verification of higher-order programs. We also present a type-based verification algorithm for HORS's. The algorithm can deal with only a fragment of the properties expressed by modal μ-calculus, but the algorithm and its correctness proof are (arguably) much simpler than those of Ong's game-semantics-based algorithm. Moreover, while the HORS model checking problem is n-EXPTIME in general, our algorithm is linear in the size of HORS, under the assumption that the sizes of types and specifications are bounded by a constant.

Categories and Subject Descriptors D.2.4 [*Software Engineer-*

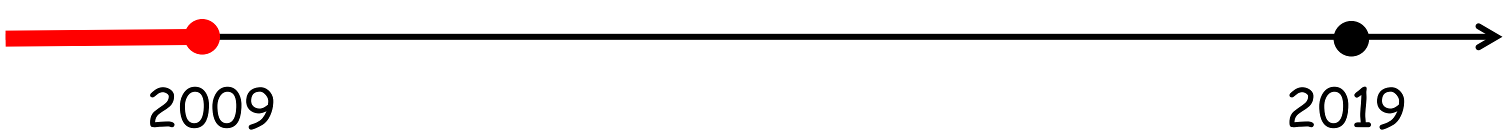
lem of resource usage verification [19] for higher-order functional languages with dynamic resource creation and access primitives. The goal of the verification is to check that each dynamically created resource is accessed in a proper manner (like "an opened file is eventually closed, and it is not read or written after being closed"). Assertion-based model-checking problems (like "$X > 0$ holds at program point p") can also be recasted as the resource verification problem, by regarding an assertion failure as an access to a global resource. (For example, "`assert(b)`" can be transformed into "`if b then skip else fail`," where `fail` is an action to the global resource. Then the problem of checking lack of assertion failures is reduced to the resource usage verification problem of checking whether the `fail` action occurs.)

Our verification technique is built on the recent result on model checking of *higher-order recursion schemes* (HORS's, for short) [29]. A higher-oder recursion scheme is a grammar for describing an infinite tree. HORS is a generalization of regular tree grammars; they are described by HORS's of order 0. Ong [29] has recently shown the decidability of the problem of checking whether the infinite tree generated by $\mathcal{G}$ satisfies ψ, given a modal μ-calculus formula ψ and an HORS $\mathcal{G}$.

+ Sound and complete reduction from higher-order program verification to HORS model checking

+ Type-based characterization of (a subclass of) HORS model checking, which yields a naive fixed-parameter linear-time algorithm

- It remained open whether HORS model checking is feasible in practice.
(The naive algorithm is impractical due to the huge constant factor.)



Outline

- ♦ **What is Higher-Order Model Checking?**
- ♦ **History of the Project**
  - **start of the project (through 2009)**
    - **application to program verification [POPL09]**
    - **type-theoretic foundation [POPL09]**
    - **practical algorithm [PPDP09]**
  - **tool development and quest for better algorithms (2010-2016)**
  - **shift to HFL model checking (2017-)**

- ♦ **Conclusion**

Practical Algorithm for HORS Model Checking?

- **Naive algorithm:**

Too large: k-fold exponential in the size of automata and the largest arity of functions

Γ := Γ_{max} (all the possible typings for non-terminals)
repeat Γ := Shrink(Γ) until Γ= Shrink(Γ)
return (S: $q_0 \in \Gamma$)

- **Practical algorithm [K, PPDP09]**

```
while true do {
    Γ := (guess typings for non-terminals)
    repeat Γ := Shrink(Γ) until Γ= Shrink(Γ)
    if S: q0 ∈ Γ then return true
}
```

Practical Algorithm for HORS Model Checking?

- **Practical algorithm [K, PPDP09]**

```
while true do {
    Γ := (guess typings for non-terminals)
    repeat Γ := Shrink(Γ) until Γ= Shrink(Γ)
    if S: q0 ∈ Γ then return true
}
```

How can we guess types?

- The type of a function describes how it will be used in a program

=> Guess the type of a function by executing the program
and observing how the function is used.

Example

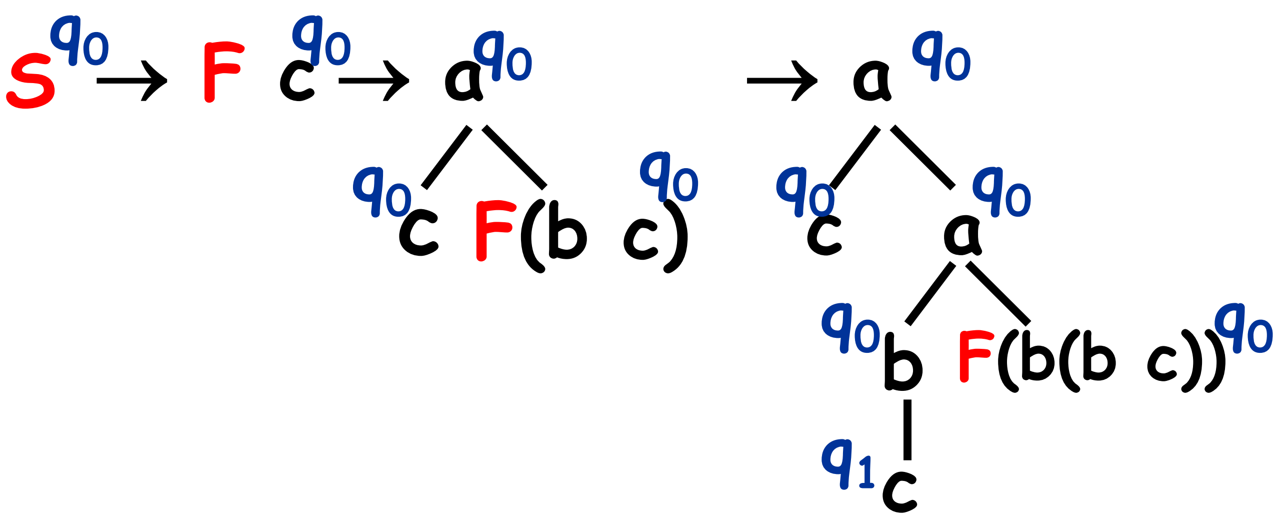
- **HORS:**

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

- **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

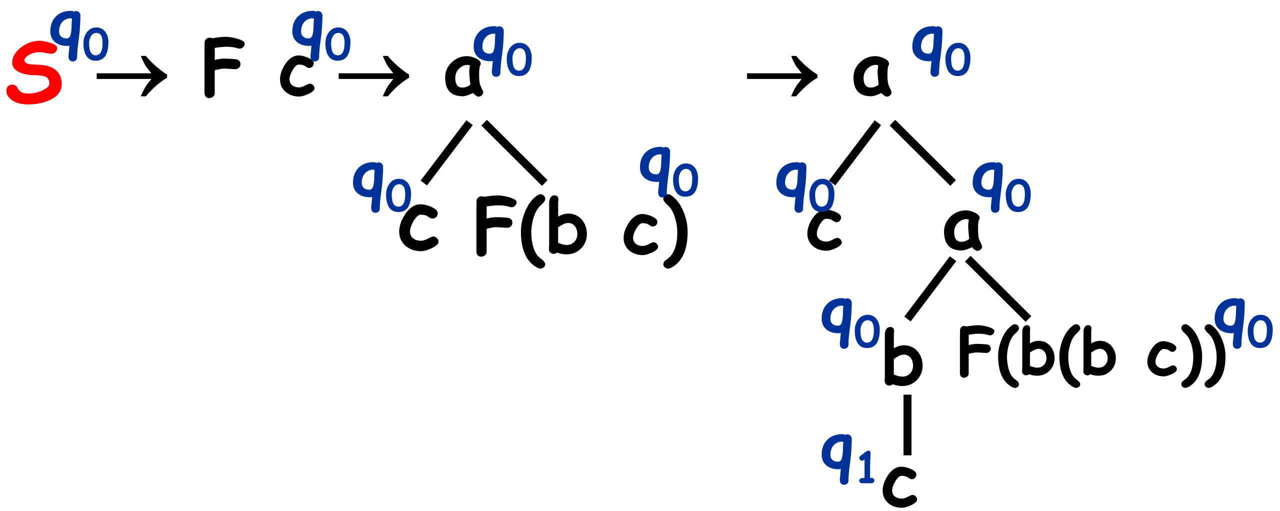
- **HORS:**

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

- **Automaton:**

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

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



Γ_0 :

S: q_0

Example

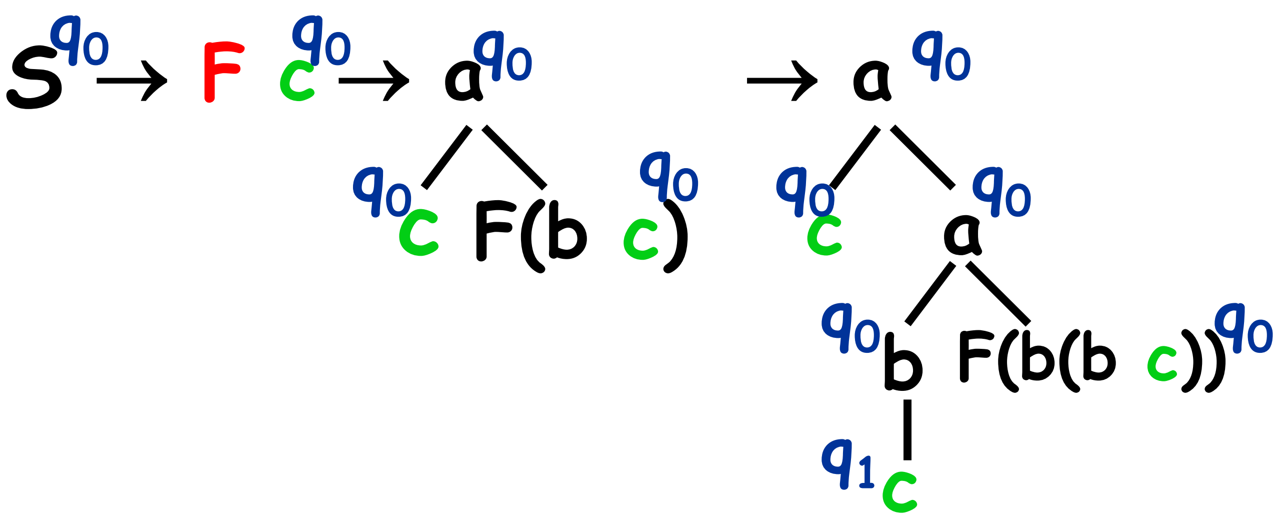
- **HORS:**

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

- **Automaton:**

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

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



Γ_0 :

$S:\ q_0$

$F:\ ? \rightarrow q_0$

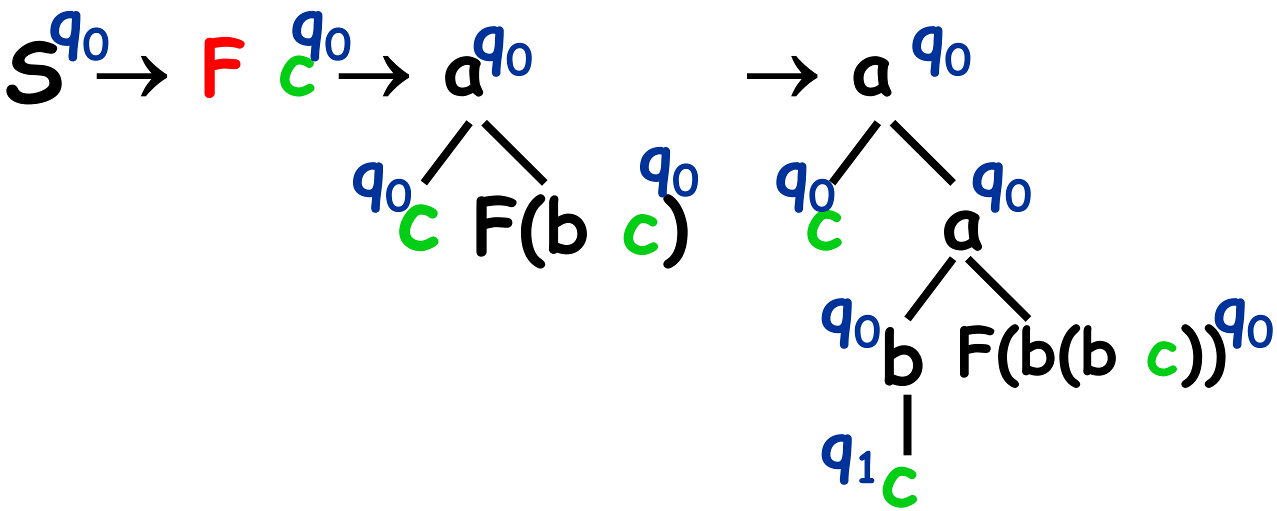
Example

- **HORS:**

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

- **Automaton:**

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



Γ_0 :

S: q_0

F: $q_0 \wedge q_1 \to q_0$

Example

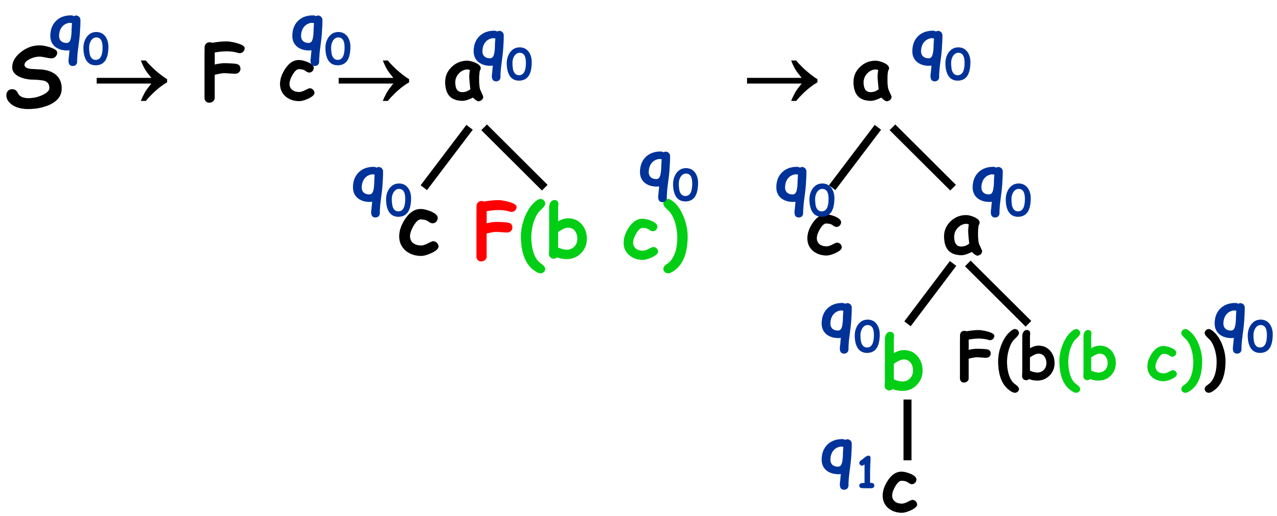
- **HORS:**

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

- **Automaton:**

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

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



Γ_0 :

$S: q_0$

$F: q_0 \wedge q_1 \to q_0$

$F: q_0 \to q_0$

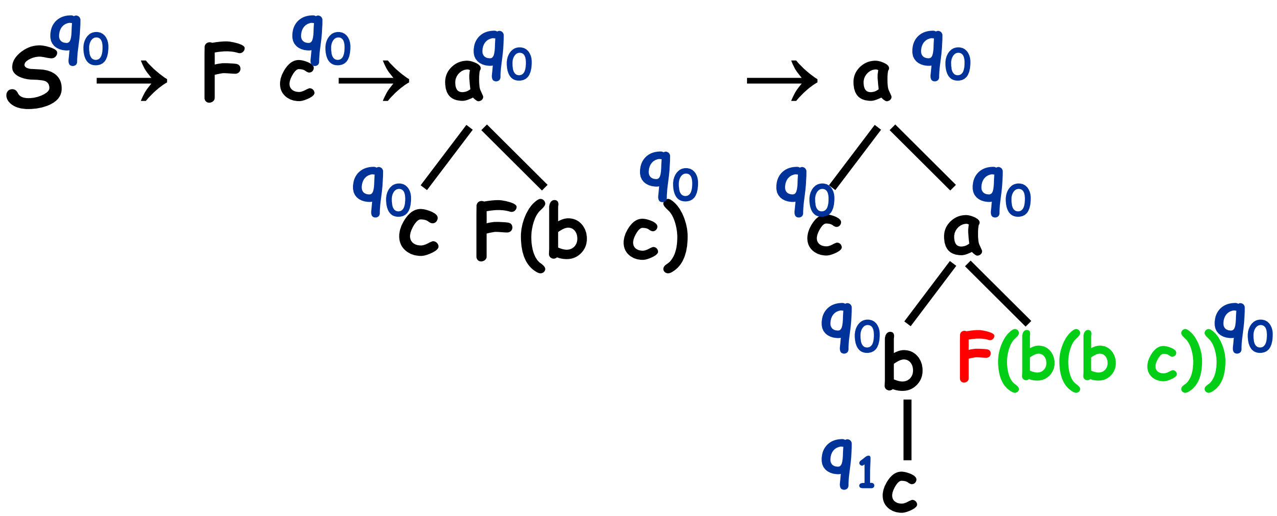
Example

- **HORS:**

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

- **Automaton:**

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



Γ_0 :

$S:\ q_0$

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

$F:\ q_0 \to q_0$

$F:\ \top \to q_0$

Example

- **HORS:**

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

- **Automaton:**

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

```
while true do {
    Γ := (guess typings for non-terminals)
    repeat Γ := Shrink(Γ) until Γ= Shrink(Γ)
    if S: q0 ∈ Γ then return true
}
```

q_1 c

Γ_0 :

$S:\ q_0$

$F:\ q_0 \wedge q_1 \rightarrow q_0$

$F:\ q_0 \rightarrow q_0$

$F:\ \top \rightarrow q_0$

Example

- **HORS:**

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

- **Automaton:**

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

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

```
while true do {
   Γ := (guess typings for non-terminals)
   repeat Γ := Shrink(Γ) until Γ= Shrink(Γ)
   if S: q0 ∈ Γ then return true
}
```

q_1 c

Γ_0 :

$S:\ q_0$

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

$F:\ q_0 \to q_0$

$F:\ \top \to q_0$

Example

- **HORS:**

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

- **Automaton:**

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

while true do {
  Γ := (guess typings for non-terminals)
  repeat Γ := Shrink(Γ) until Γ= Shrink(Γ)
  if S: $q_0 \in \Gamma$ then return true
}

q_1 c

Γ_0 :

S: q_0

F: $q_0 \wedge q_1 \rightarrow q_0$

F: $q_0 \rightarrow q_0$

F: $\top \rightarrow q_0$

Example

- **HORS:**

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

- **Automaton:**

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

```
while true do {
   Γ := (guess typings for non-terminals)
   repeat Γ := Shrink(Γ) until Γ= Shrink(Γ)
   if S: q0 ∈ Γ then return true
}
```

q_1 c

Γ_0 :

$S:\ q_0$

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

$F:\ q_0 \to q_0$

$F:\ \top \to q_0$

Example

- **HORS:**

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

- **Automaton:**

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

```
while true do {
   Γ := (guess typings for non-terminals)
   repeat Γ := Shrink(Γ) until Γ= Shrink(Γ)
   if S: q0 ∈ Γ then return true
}
```

q_1 c

Γ_0 :

S: q_0

F: $q_0 \wedge q_1 \rightarrow q_0$

F: $q_0 \rightarrow q_0$

F: $\top \rightarrow q_0$

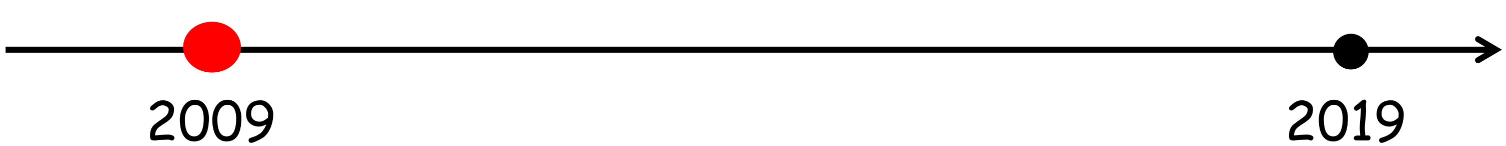
TRecS [K. PPDP09]

http://www-kb.is.s.u-tokyo.ac.jp/~koba/trecs/

- The first practical model checker for HORS
- Does not immediately suffer from k-EXPTIME bottleneck
- Used as a backend of the software model checker MoChi

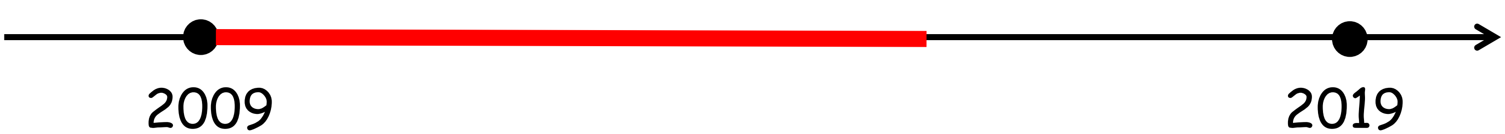
Summary of the Results in 2009

- **Applications to program verification [POPL09]**
- **Type-theoretic foundations**
  - [POPL09] for trivial automata model checking
  - [LICS09, with Ong] for full μ-calculus model checking
- **The first practical algorithm [PPDP09]**
- **Complexity**
  - parameterized complexity [POPL09, LICS09]
  - complexity of subclasses [ICALP09, with Ong]



Outline

- ♦ **What is Higher-Order Model Checking?**
- ♦ **History of the Project**
  - – **start of the project (through 2009)**
    - **application to program verification [POPL09]**
    - **type-theoretic foundation [POPL09]**
    - **practical algorithm [PPDP09]**
  - – **tool development and quest for better algorithms and more foundations (2010-2016)**
  - – **shift to HFL model checking (2017-)**



- ♦ **Conclusion**

HOMC Project: 2010 - 2016

- **Applications**
  - Automated program verification
    - MoCHi [K+, PLDI 11]
    - Termination and temporal properties
      [Kuwahara+ ESOP14, CAV15][Murase+ POPL16][Watanabe+ ICFP16]
  - Data compression [K+ PEM12]
- **Quest for better HORS MC algorithms**
  - GTRecS, HorSat, HorSat2, HorSatP, ...
- **Foundations (properties on HO languages)**
  - HO languages vs context-sensitive languages
  - Pumping lemmas [K, LICS13] [Asada&K, ICALP17]

HOMC Project: 2010 - 2016

- **Applications**
  - Automated program verification
    - MoChi [K+, PLDI 11]
    - Termination and temporal properties [Kuwahara+ ESOP14, CAV15][Murase+ POPL16][Watanabe+ ICFP16]
  - Data compression [K+ PEM12]
- **Quest for better HORS MC algorithms**
  - GTRecS, HorSat, HorSat2, HorSatP, ...
- **Foundations (properties on HO languages)**
  - HO languages vs context-sensitive languages
  - Pumping lemmas [K, LICS13] [Asada&K, ICALP17]

MoCHi: Software Model Checker for OCaml [K, Sato&Unno, PLDI11]

- Based on HORS MC + predicate abstraction

MoCHi	SLAM [Ball+]	Blast [Beyer+]
HORS MC	pushdown MC	finite-state MC

- Support:
  - higher-order functions + recursion (by HORS MC)
  - integers (by predicate abstraction)
  - exceptions (by extended CPS transformation)
  - (restricted) ADT (by encoding into functions)

$[\,\tau\ \text{list}\,] = \text{int} \times (\text{int} \to [\tau])$

length

function from indices to elements

MoCHi: Software Model Checker for OCaml [K, Sato&Unno, PLDI11]

- **Based on HORS MC + predicate abstraction**

MoCHi	SLAM [Ball+]	Blast [Beyer+]
HORS MC	pushdown MC	finite-state MC

- **Support:**
  - higher-order functions + recursion (by HORS MC)
  - integers (by predicate abstraction)
  - exceptions (by extended CPS transformation)
  - (restricted) ADT (by encoding into functions)

$[\tau \text{ list}] = \text{int} \times (\text{int} \rightarrow [\tau])$

nil = (0, λx. fail)

cons = λx.λ(len,f). (len+1, λi.if i=0 then x else f(i-1))

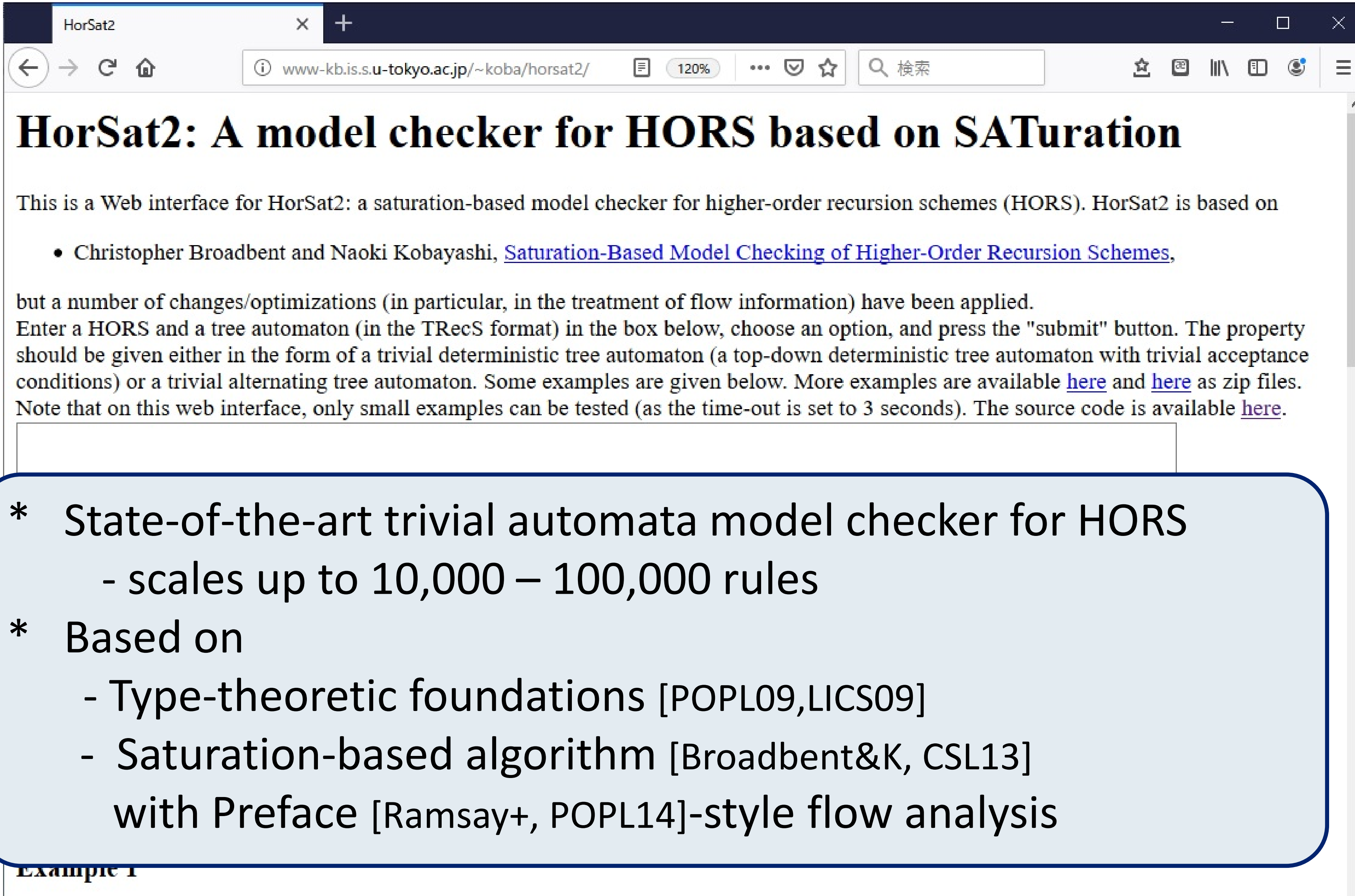
hd (len,f) = f(0)

...

HOMC Project: 2010 - 2016

- **Applications**
  - Automated program verification
    - MoChi [K+, PLDI 11]
    - Termination and temporal properties [Kuwahara+ ESOP14, CAV15][Murase+ POPL16][Watanabe+ ICFP16]
  - Data compression [K+ PEM12]
- **Quest for better HORS model checkers**
  - GTRecS, HorSat, HorSat2, HorSatP, ...
- **Foundations**
  - HO languages vs context-sensitive languages
  - Pumping lemmas [K, LICS13] [Asada&K, ICALP17]

HorSat2 [K, 2014]



* State-of-the-art trivial automata model checker for HORS
  - scales up to 10,000 – 100,000 rules

* Based on
  - Type-theoretic foundations [POPL09,LICS09]
  - Saturation-based algorithm [Broadbent&K, CSL13]
    with Preface [Ramsay+, POPL14]-style flow analysis

Outline

- ♦ **What is Higher-Order Model Checking?**
- ♦ **History of the Project**
  - – **start of the project (through 2009)**
    - **application to program verification [POPL09]**
    - **type-theoretic foundation [POPL09]**
    - **practical algorithm [PPDP09]**
  - – **tool development and quest for better algorithms and more foundations (2010-2016)**
  - – **shift to HFL model checking (2017-)**

- ♦ **Conclusion**

HOMC Project: 2017-

- From HORS to HFL model checking

	Models	Logic
finite state model checking	finite state systems	modal μ-calculus
HORS model checking [Knapik+ 01; Ong 06]	higher-order recursion schemes (HORS)	modal μ-calculus
HFL model checking [Viswanathan& Viswanathan 04]	finite state systems	higher-order modal fixpoint logic (HFL)

Higher-Order Modal Fixpoint Logic (HFL)

[Viswanathan&Viswanathan 04]

♦ Higher-order extension of the modal μ-calculus

φ ::=	true	
	$\varphi_1 \wedge \varphi_2$	
	$\varphi_1 \vee \varphi_2$	
	$[a]\varphi$	φ *must* hold after a
	$\langle a \rangle\varphi$	φ *may* hold after a
	X	variable
	$\mu X.\varphi$	*least* fixpoint
	$\nu X.\varphi$	*greatest* fixpoint

e.g. $\mu X.\ \langle b \rangle \mathrm{true} \vee \langle a \rangle X$

"b" may occur after a finite number of "a" transitions

Higher-Order Modal Fixpoint Logic (HFL)

[Viswanathan&Viswanathan 04]

♦ **Higher-order extension of the modal μ-calculus**

φ ::=	true	
	$\varphi_1 \wedge \varphi_2$	
	$\varphi_1 \vee \varphi_2$	
	$[a]\varphi$	φ must hold after a
	$\langle a \rangle\varphi$	φ may hold after a
	X	predicate variable
	$\mu X^{\kappa}.\varphi$	*least* fixpoint
	$\nu X^{\kappa}.\varphi$	*greatest* fixpoint
	$\lambda X^{\kappa}.\varphi$	(higher-order) predicate
	$\varphi_1\ \varphi_2$	application
κ ::=	$\bullet$	the type of propositions
	$\kappa_1 \to \kappa_2$	

Selected Typing Rules for HFL

$$\Gamma \vdash \mathbf{true}: \bullet$$

$$\frac{\Gamma \vdash \varphi: \bullet}{\Gamma \vdash [a]\varphi: \bullet}$$

$$\frac{\Gamma \vdash \varphi: \bullet \quad \Gamma \vdash \psi: \bullet}{\Gamma \vdash \varphi \wedge \psi: \bullet}$$

$$\frac{\Gamma, X:\kappa_1 \vdash \varphi:\kappa_2}{\Gamma \vdash \lambda X.\varphi: \kappa_1 \to \kappa_2}$$

$$\Gamma, X:\kappa \vdash X:\kappa$$

$$\frac{\Gamma \vdash \varphi: \kappa_1 \to \kappa_2 \quad \Gamma \vdash \psi: \kappa_1}{\Gamma \vdash \varphi\ \psi: \kappa_2}$$

$$\frac{\Gamma, X:\kappa \vdash \varphi:\kappa}{\Gamma \vdash \mu X.\varphi: \kappa}$$

Example

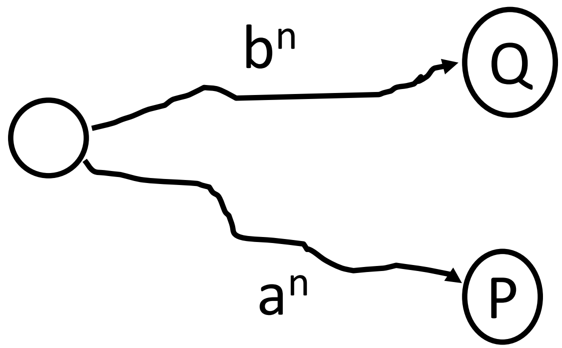
$$\left(\mu F^{\bullet\to\bullet\to\bullet}.\lambda X.\lambda Y.\ (X\wedge Y) \vee F\ (\langle a\rangle X)\ (\langle b\rangle Y)\right)\ P\ Q$$

$$= \left(\lambda X.\lambda Y.\ (X\wedge Y) \vee (\mu F\ldots)\ (\langle a\rangle X)\ (\langle b\rangle Y))\right)\ P\ Q$$

$$= (P\wedge Q) \vee$$
$$\left(\mu F^{\bullet\to\bullet\to\bullet}.\lambda X.\lambda Y.\ (X\wedge Y) \vee F(\langle a\rangle X)(\langle b\rangle Y)\right)\ (\langle a\rangle P)(\langle b\rangle Q)$$

$$= (P\wedge Q) \vee (\langle a\rangle P\wedge\langle b\rangle Q) \vee (\langle a\rangle\langle a\rangle P\wedge\langle b\rangle\langle b\rangle Q) \vee \ldots$$

For some n, $\langle a\rangle^n$ P and $\langle b\rangle^n$ Q hold



HFL Model Checking

[Viswanathan&Viswanathan 2004]

Given

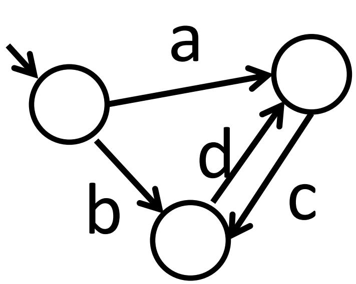
L: (finite-state) labeled transition system

φ: HFL formula,

does L satisfy φ?

e.g. L |= φ for:

L:

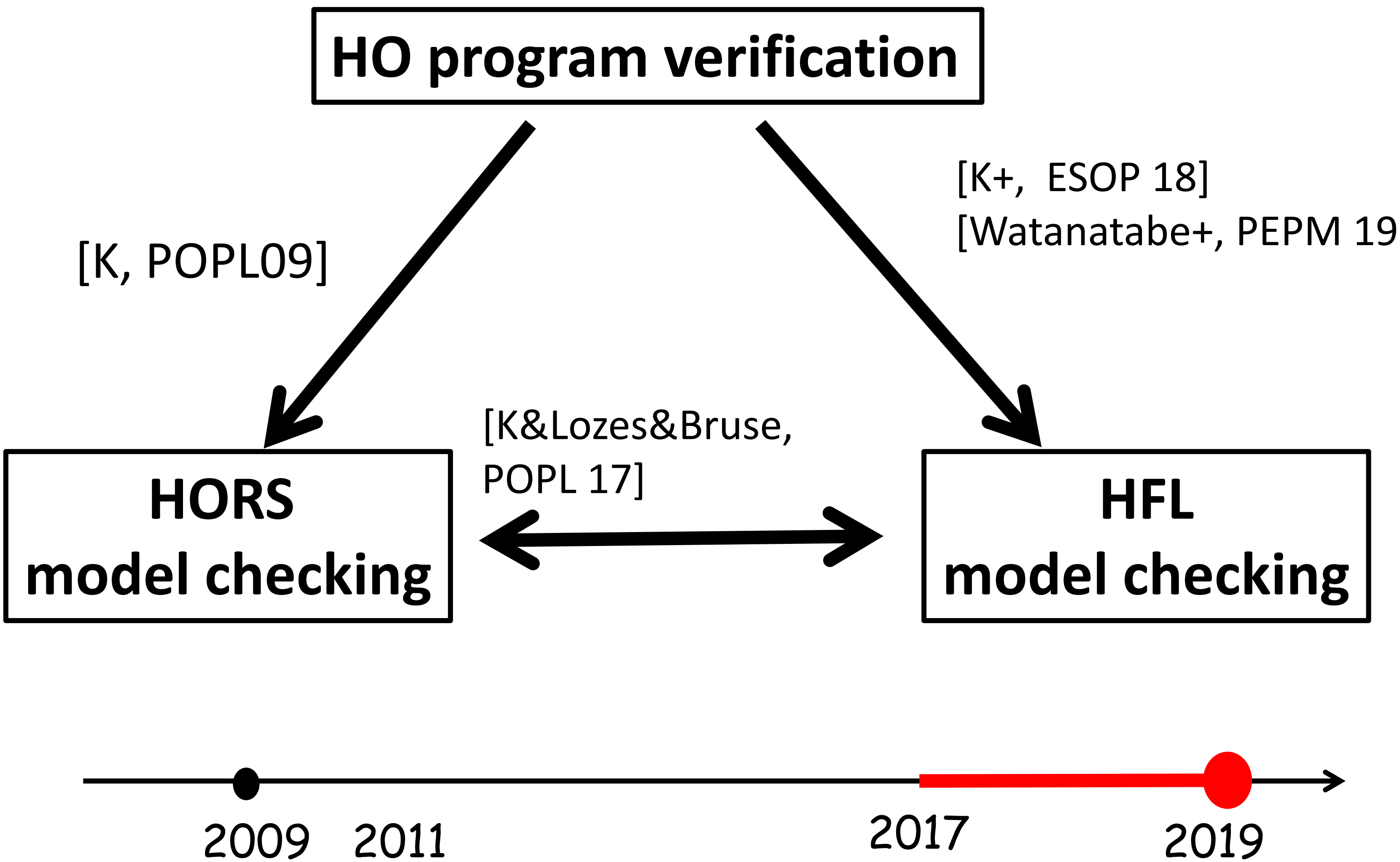


$\varphi: (\mu F.\lambda X.\lambda Y.\ (X \wedge Y)$

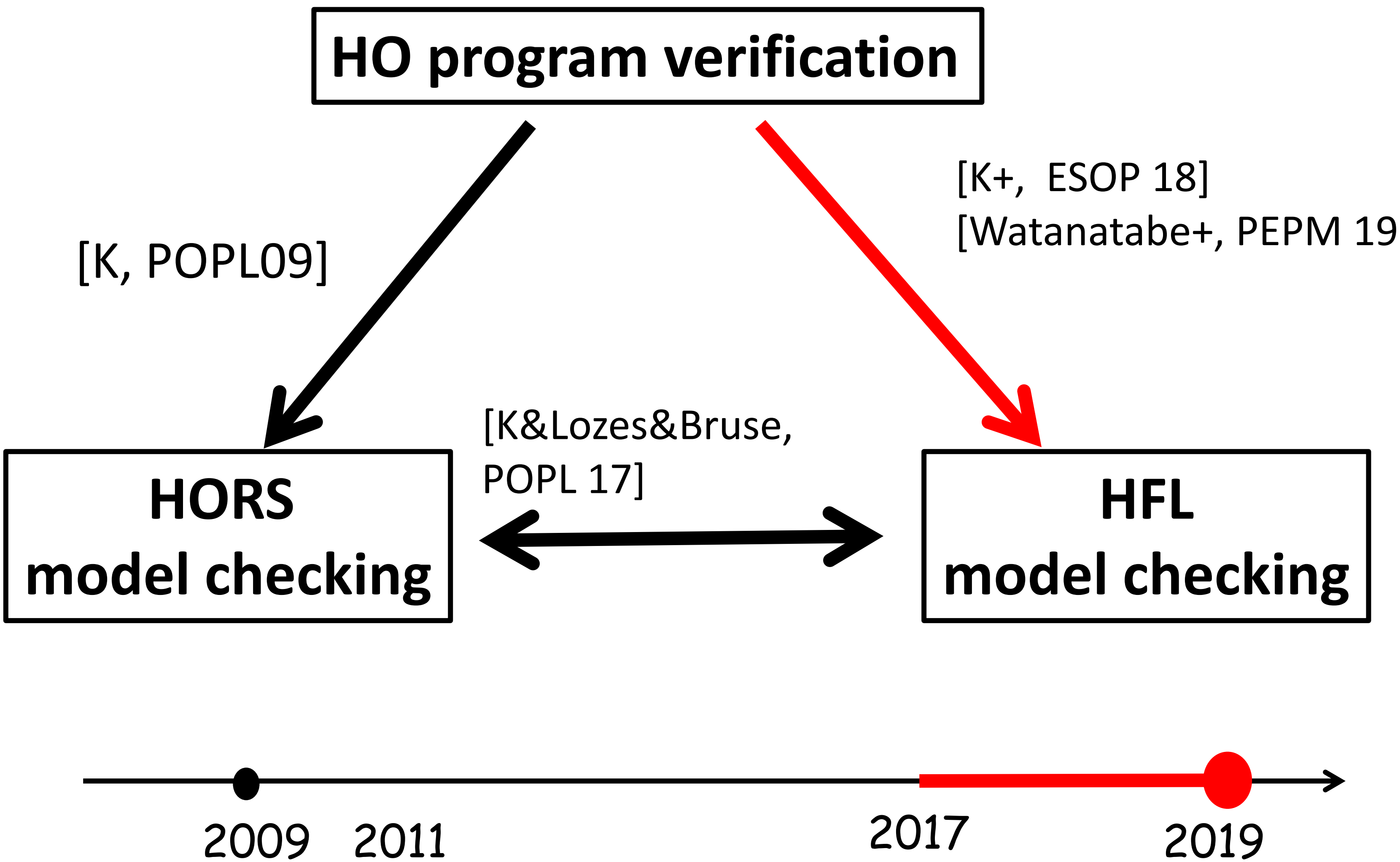
$\quad \vee F\ (\langle a\rangle X)\ (\langle b\rangle Y))$

$\quad (\langle c\rangle \mathrm{true})\ \ (\langle d\rangle \mathrm{true})$

HORS/HFL Model Checking and Program Verification



HORS/HFL Model Checking and Program Verification



Higher-Order Program Verification vs HFL/HORS Model Checking

	Models	Spec
HO program verification	HO programs	safety, termination, ...
HORS model checking [Knapik+ 01; Ong 06]	higher-order recursion schemes (HORS)	modal μ-calculus formula
HFL model checking [Viswanathan& Viswanathan 04]	finite state systems	HFL formula

[K, POPL09], ...

??

Higher-Order Program Verification vs HFL/HORS Model Checking

	Models	Spec
HO program verification	HO programs	safety, termination, ...
HORS model checking [Knapik+ 01; Ong 06]	higher-order recursion schemes (HORS)	modal μ-calculus formula
HFL model checking [Viswanathan& Viswanathan 04]	finite state systems	HFL formula

[K, POPL09], ...

"The program's behavior is correct"

From Program Verification to HFL Model Checking: Example

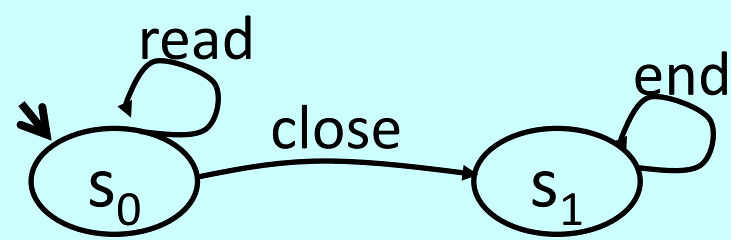
let y = open “foo”
in
  read(y); close(y)

HFL formula that says “the behavior of the program is correct”

<read><close><end>true

Is the file “foo” accessed according to read* close?

LTS:



From Program Verification to HFL Model Checking: Example

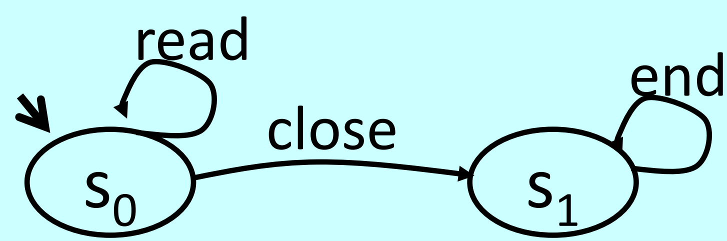
let y = open “foo”
in
  read(y); close(y)

HFL formula that says “the behavior of the program is correct”

<read><close><end>true

Is the file “foo” accessed according to read* close?

Does LTS:



satisfy the formula S?

From Program Verification to HFL Model Checking: Example

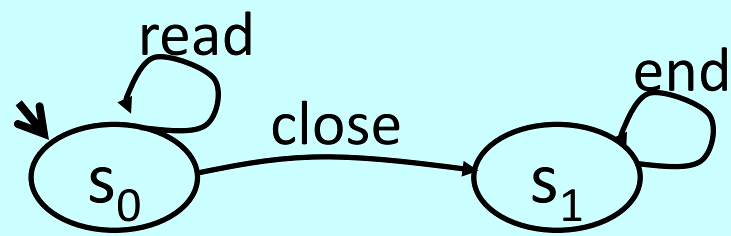
```
let y = open "foo"
in
  if * then
    (read(y); close(y))
  else close(y)
```

HFL formula that says
"the behavior of the program is correct"

<read><close><end>true
∧
<close><end>true

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

Does LTS:



satisfy the formula S?

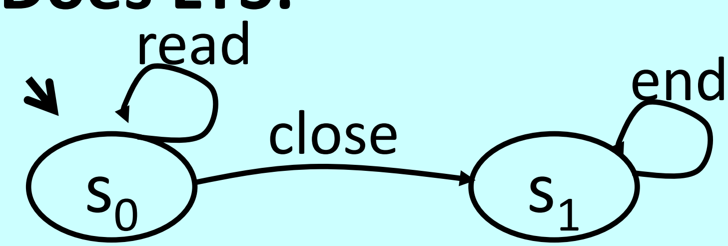
From Program Verification to HFL Model Checking: Example

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

HFL formula that says "the behavior of the program is correct"

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

Does LTS:



satisfy the formula S?

From Program Verification to HFL Model Checking: Example

```
let f x k =
 if * then close x k
 else read x (f x k)
in
let y = open “foo”
in
    f y ()
```

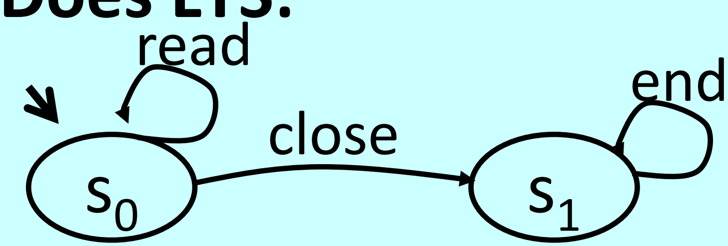
HFL formula that says “the behavior of the program is correct”

$$F\ x\ k =_\nu \langle close \rangle k \wedge (\langle read \rangle (F\ x\ k))$$

$$S =_\nu F\ true\ (\langle end \rangle true)$$

Is the file “foo” accessed according to read* close?

Does LTS:



satisfy the formula S?

From Program Verification to extended HFL (HFL_Z) Model Checking

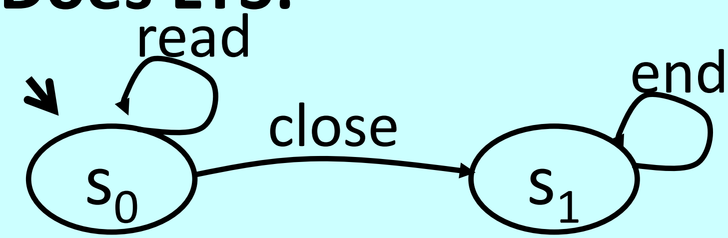
```
let f n x k =
 if n≤0 then close x k
 else
  read x (f (n-1) x k)
in
let y = open "foo"
in  f m y ()
```

$$F\ n\ x\ k =_{\mu} (n \leq 0 \Rightarrow \langle close \rangle k) \wedge (\neg n \leq 0 \Rightarrow \langle read \rangle (F\ (n-1)\ x\ k))$$

$$S =_{\mu} F\ m\ true\ (\langle end \rangle true)$$

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

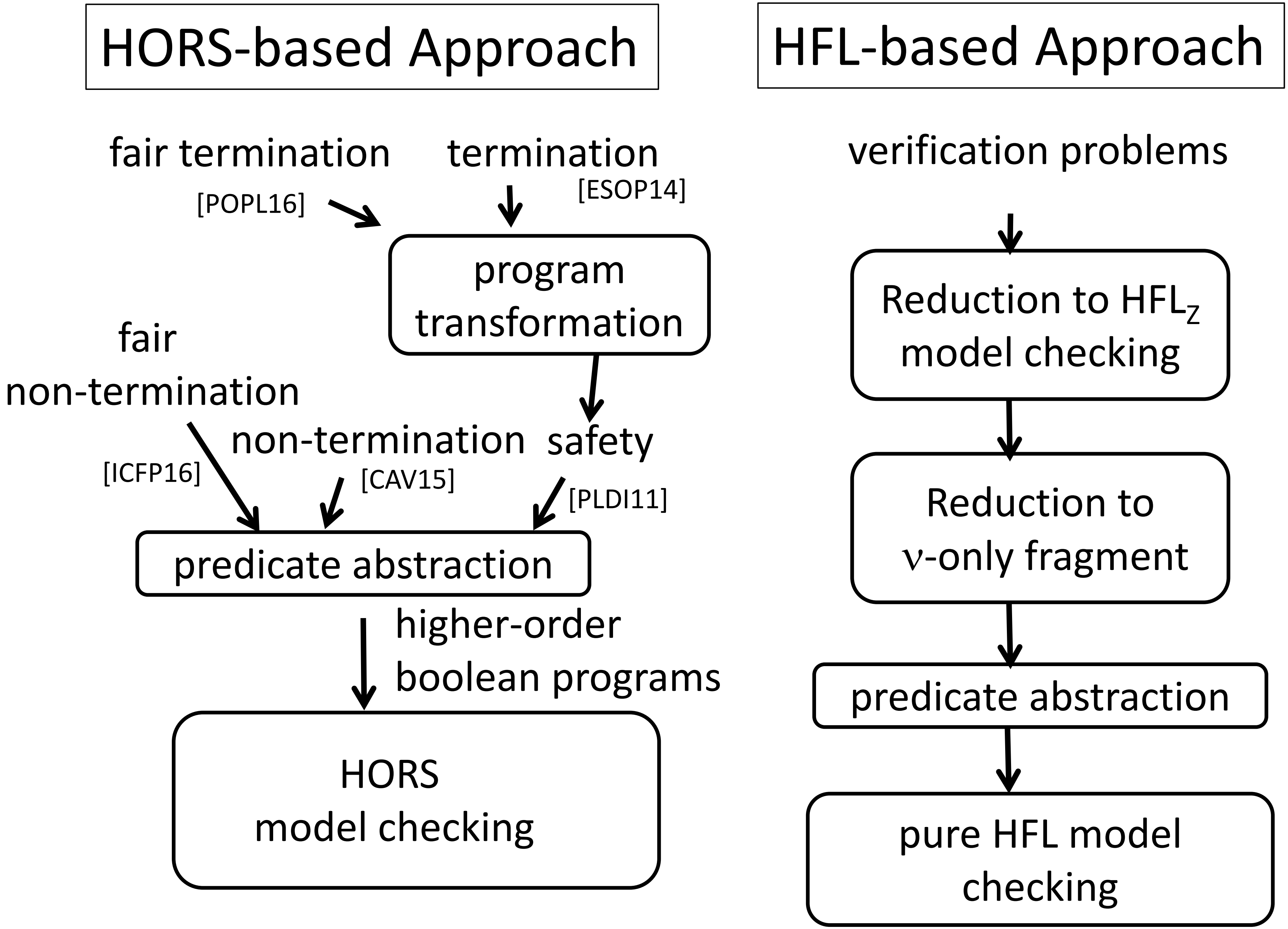
Does LTS:



satisfy the formula S?

HOMC Project: 2017-

- **HFL approach to program verification**
  - **More streamlined than HORS-based approach**

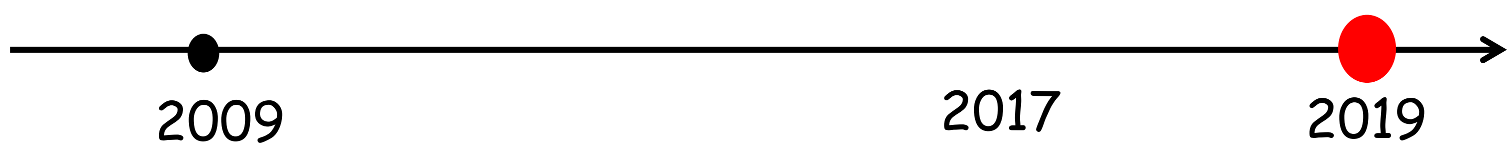

HORS-based Approach
fair termination
termination
[POPL16]
[ESOP14]
program transformation
fair non-termination
non-termination
safety
[ICFP16]
[CAV15]
[PLDI11]
predicate abstraction
higher-order boolean programs
HORS model checking
HFL-based Approach
verification problems
Reduction to HFL_Z model checking
Reduction to ν-only fragment
predicate abstraction
pure HFL model checking

HOMC Project: 2017-

- **HFL approach to program verification**
  - More streamlined than HORS-based approach
  - Natural extension of other approaches
    - Constrained Horn Clauses (CHC)
      + higher-order predicates + fixpoint alternations
      (cf. SeaHorn [Gurfinkel+], JayHorn [Kahsai+])
    - HoCHC [Burn+, 2018] + fixpoint alternations
- **Improving scalability of MoCHi**
  - modular verification [Sato&K, ESOP17]
  - machine-learning for predicate discovery
    [Champion+ TACAS18][Sato+ PEPM19]

HOMC Project: Where are we heading now?

- **Tool constructions for HFL-based approach**
  - Pure HFL model checker [Hosoi+, APLAS19]
  - validity checker for first-order fragment of HFL_Z (or, CHC + fixpoint alternations) [K+, SAS19]
- **Average-case complexity of HOMC**
  - Why does HOMC work in practice?
- **Probabilistic HORS model checking** [K, Dal Lago&Grellois, LICS19]



Conclusion

- **Summarized HOMC Project at UTokyo**
  - HOMC works in practice, despite k-EXPTIME completeness
  - Applicable to program verification and data compression
  - Of the two kinds of HOMC, the HFL-based approach seems more promising
- **Remaining challenges**
  - More tool constructions
    - scalability to larger programs,
    - non-functional features (references, concurrency, etc.)
  - More theories
    - Justification for why HOMC works in practice
    - open problems about higher-order languages