Overview of Higher-Order Model Checking Project at UTokyo

Naoki Kobayashi
University of Tokyo

Outline

- Introduction
  - HO model checking
  - Applications to program verification
- Overview of HO model checking project
- Tutorial on type-based approach to HO model checking (if time permits)

Higher-Order Recursion Scheme (HORS)

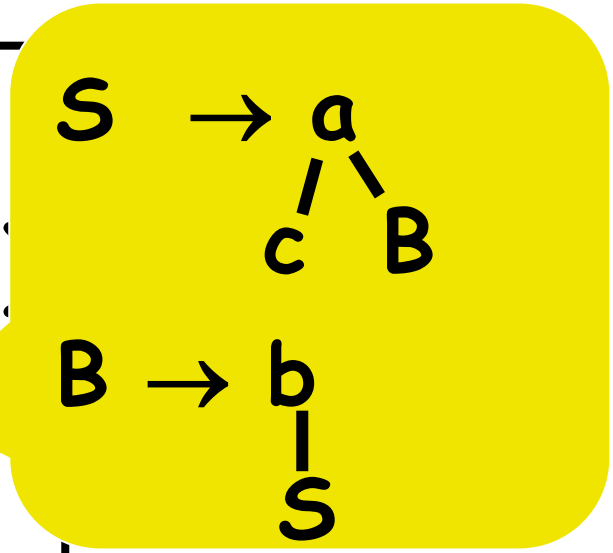
- Grammar for generating an infinite tree

Order-0 HORS

(regular tree grammar)

$S \to a\ c\ B$

$B \to b\ S$



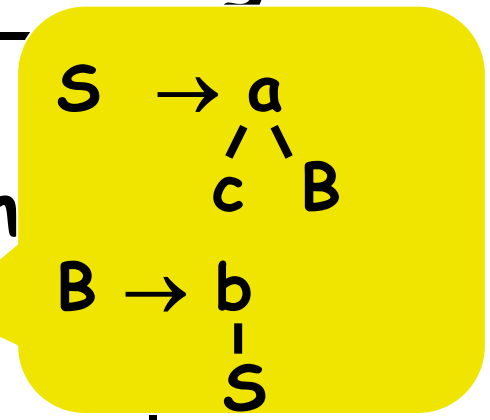
Higher-Order Recursion Scheme (HORS)

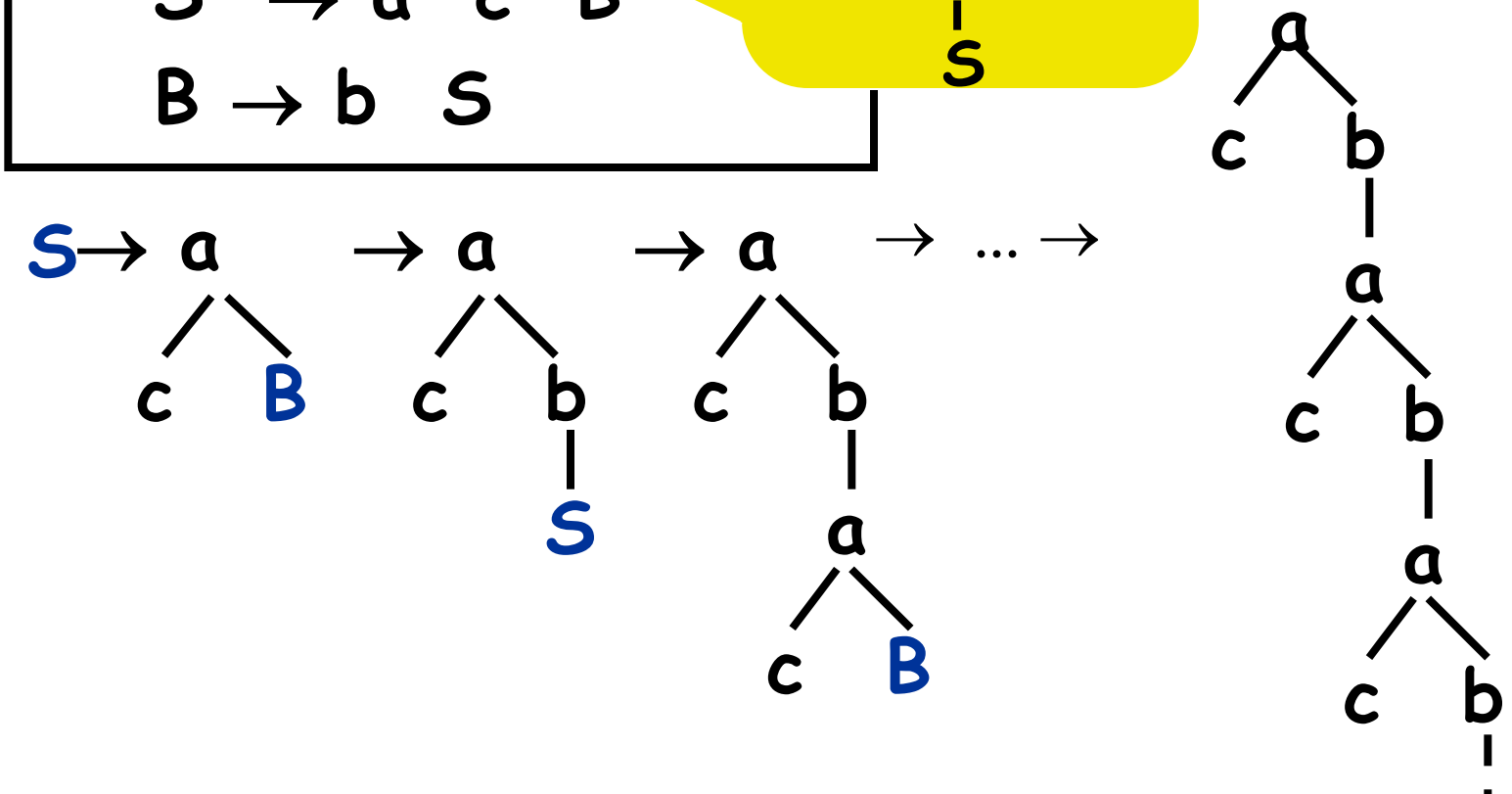
- Grammar for generating an infinite tree

Order-0 HORS
(regular tree gramm

$S \to a\ c\ B$

$B \to b\ S$





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 → o

Higher-Order Recursion Scheme (HORS)

- Grammar for generating an infinite tree

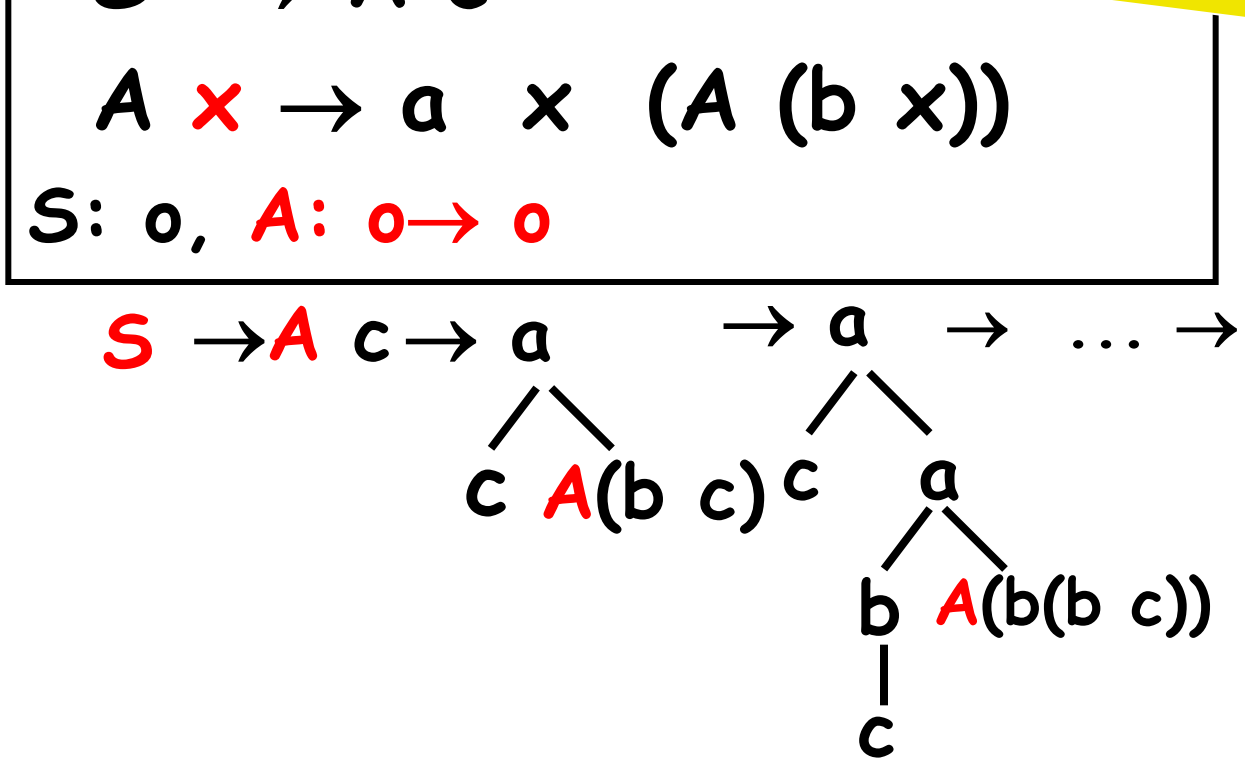
Order-1 HORS

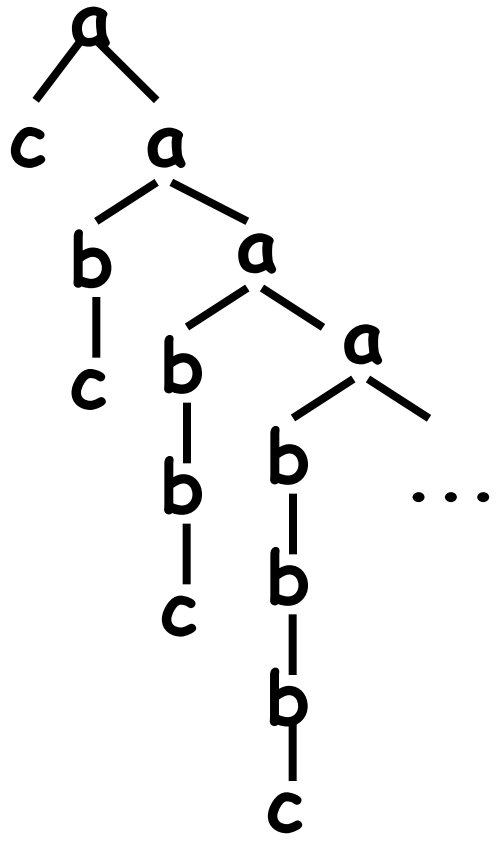
$S \rightarrow A\ c$

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

S: o, A: o→ o

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





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

≈

Call-by-name simply-typed λ-calculus

+

recursion, tree constructors

Higher-Order Model Checking

Given
 G: HORS
 A: alternating parity tree automaton
 (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"?

Higher-Order Model Checking

Order-1 HORS

$S \rightarrow A\ c$

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

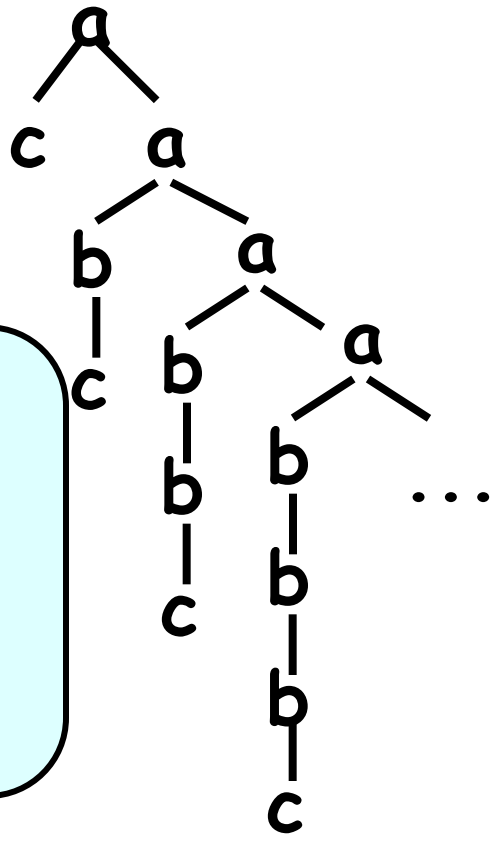
S: o, A: o→ o

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

YES!

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

NO!



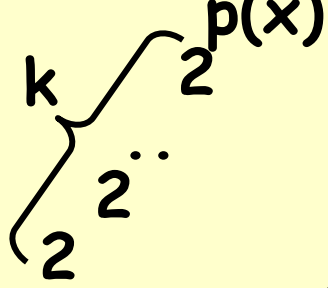
Higher-Order Model Checking

Given
- G: HORS
- 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"?

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



TRecS [K. PPDP09]

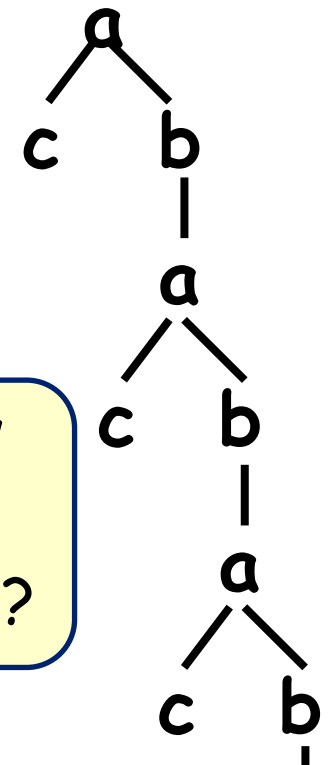
http://www-kb.is.s.u-tokyo.ac.jp/~koba/trecs/

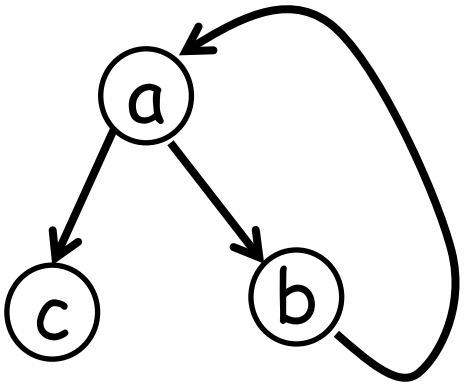
- The first practical model checker for HORS
- Does not immediately suffer from k-EXPTIME bottleneck
- A more recent model checker (HorSat2) can scale up to grammars consisting of 100,000 rules, depending on input

HO 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





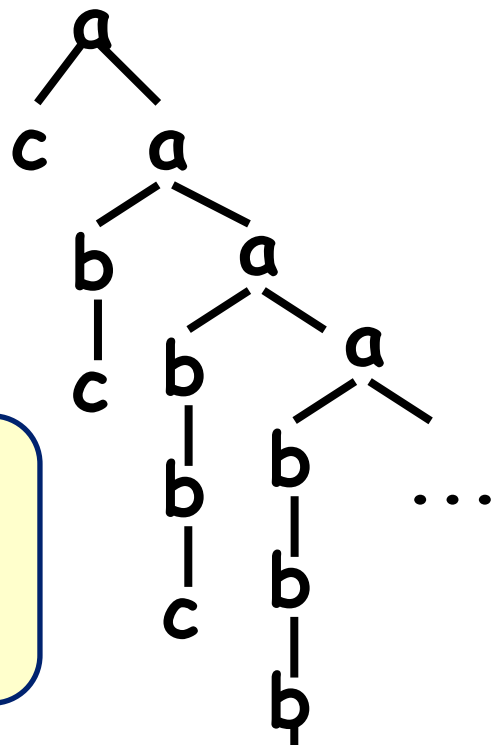
Does "a" occur below "b"?

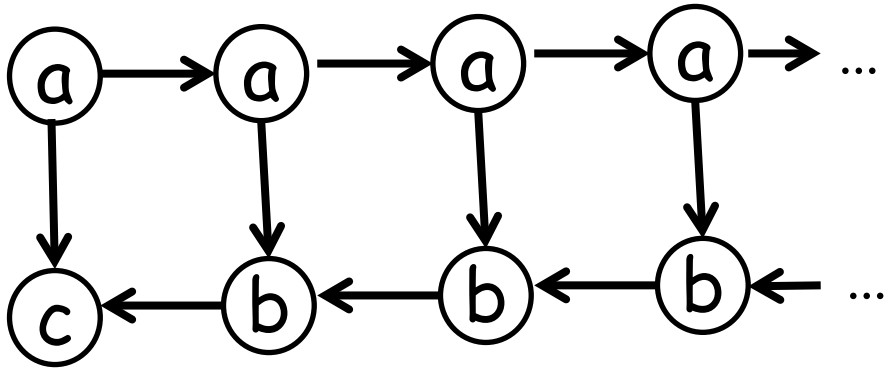
Is there a transition sequence in which "a" occurs after "b"?

HO 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



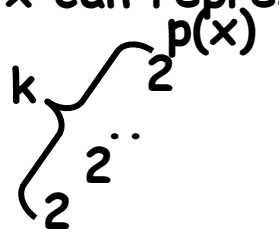


Does "a" occur below "b"?

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

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

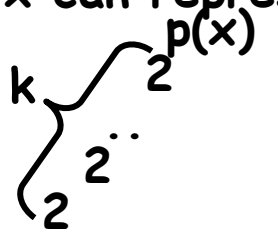
- Fixed-parameter polynomial time in the size of grammars (under certain assumptions)
- A "certificate" can be checked in polynomial time (cf. NP problems)
- For finite-state models, HO model checking can actually be faster than finite state model checking
  - HORS can compactly represent finite-state systems
    - An order-k HORS of size x can represent a system with states



  - 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 (under certain assumptions)
- A "certificate" 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



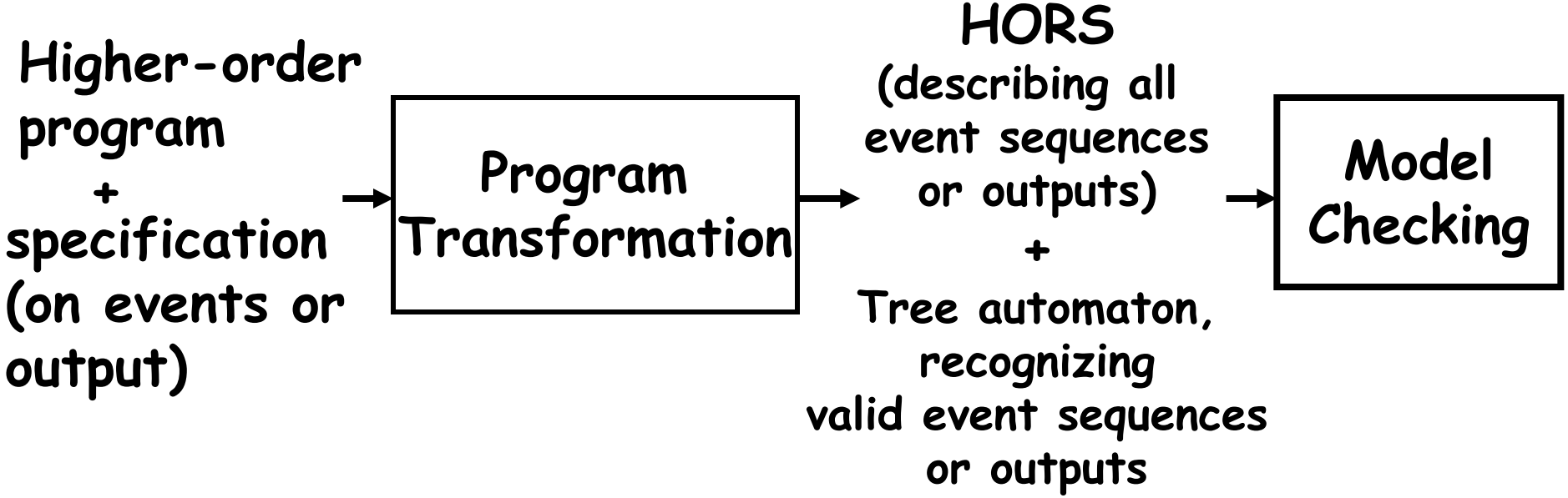
  - PTIME algorithm for HO model checking
    >> PTIME algorithm for finite-state model checking

Outline

- Introduction
  - HO model checking
  - Applications to program verification
- Overview of HO model checking project
- Type-based approach to HO model checking

From Program Verification to HO Model Checking

[K. POPL 2009]

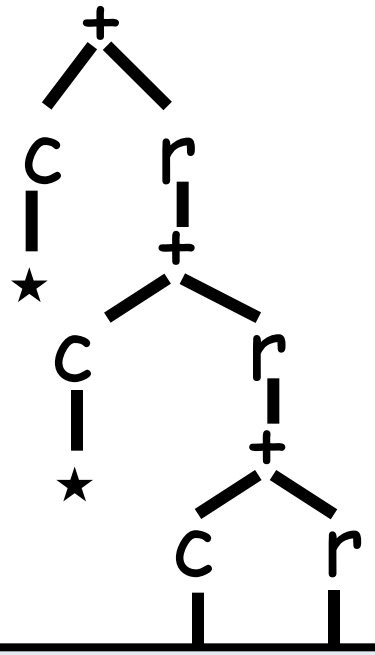


From Program Verification to Model Checking: Example

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

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

$S \rightarrow F\ d\ \bigstar$



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

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

From Program

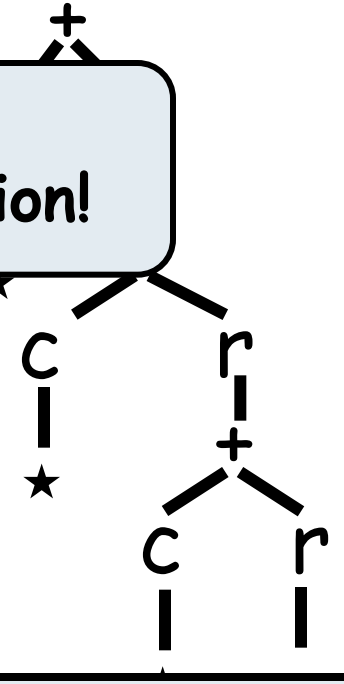
continuation parameter, expressing how "foo" is accessed after the call returns

```
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?

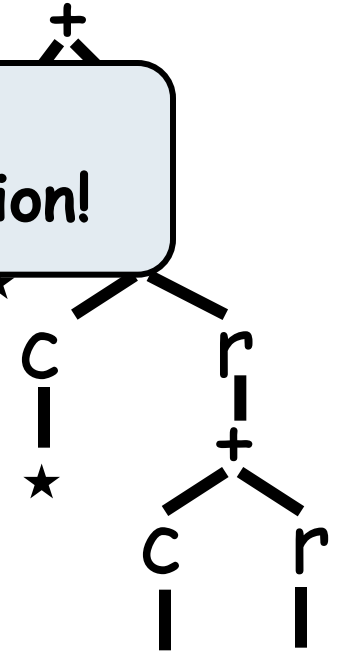
From Program Verification to Model Checking: Example

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

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

$S \rightarrow F\ d\ \bigstar$

CPS Transformation!



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

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

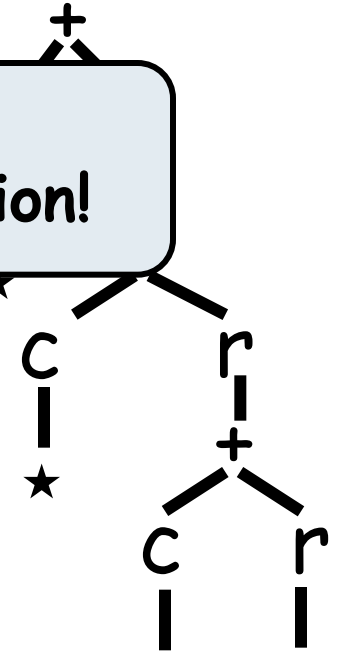
From Program Verification to Model Checking: Example

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

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

$S \rightarrow F\ d\ \bigstar$

CPS Transformation!



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

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

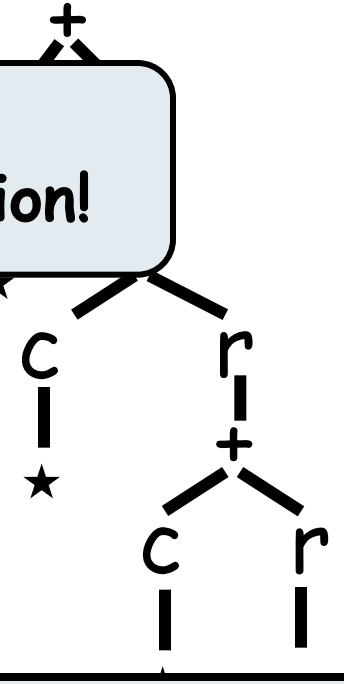
From Program Verification to Model Checking: Example

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

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

$S \rightarrow F\ d\ \bigstar$

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 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?

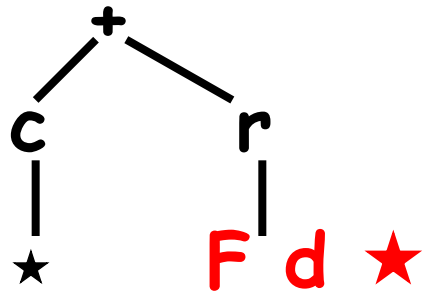
Is each path of the tree labeled by r*c?

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)
```

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

$S \rightarrow F\ d\ \bigstar$



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

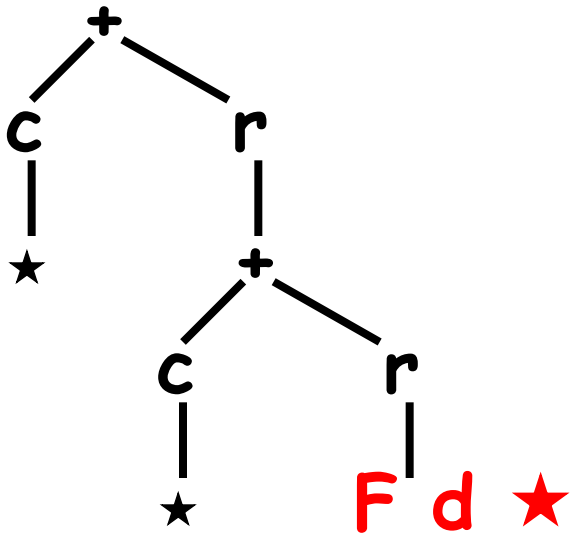
Is each path of the tree labeled by r*c?

From Program Verification to 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?

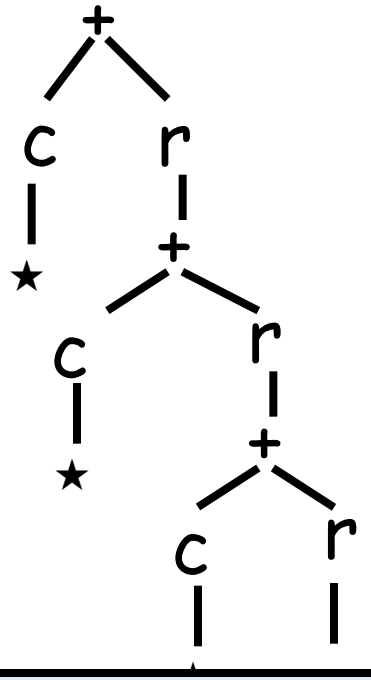
Is each path of the tree labeled by r*c?

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)
```

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

$S \rightarrow F\ d\ \bigstar$



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

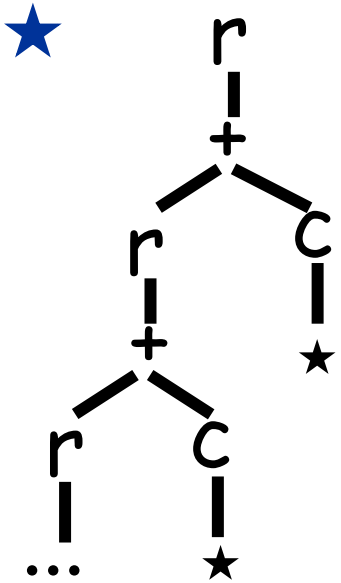
continuation for normal termination

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

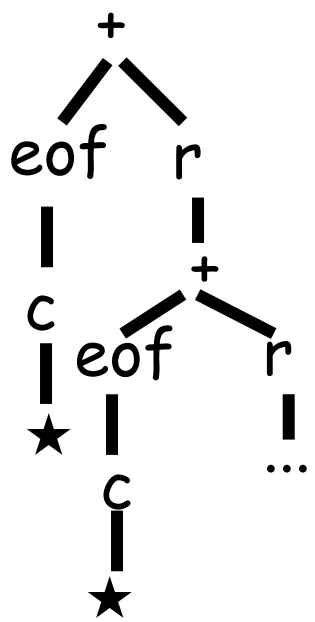
```
let f x =
 if eof(x)
 then
     close(x)
 else (read(x); f x)
in
let y = open "foo"
in
      f (y)
```

Return whether end of file has been reached

```
F x k →
 Eof x (λb.If b (c k) (r (F x k))).
S → F d ★.
Eof x k →
   + (eof (k True)) (k False).
If b x y → b x y.
True x y → x.
False x y → y.
```

eof has been reached

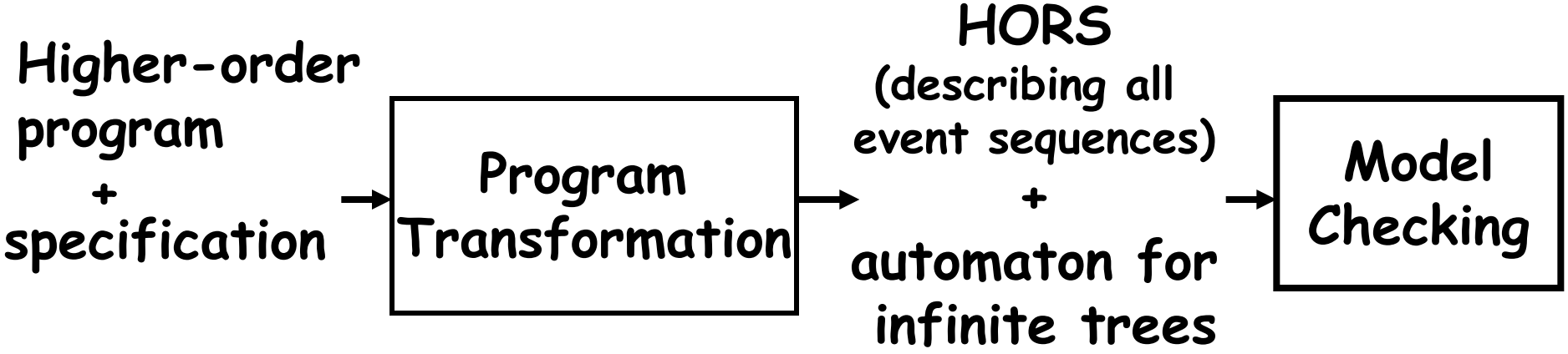
Church encoding of Booleans



Is the file closed only after eof has been reached?

Does c occur only below eof?

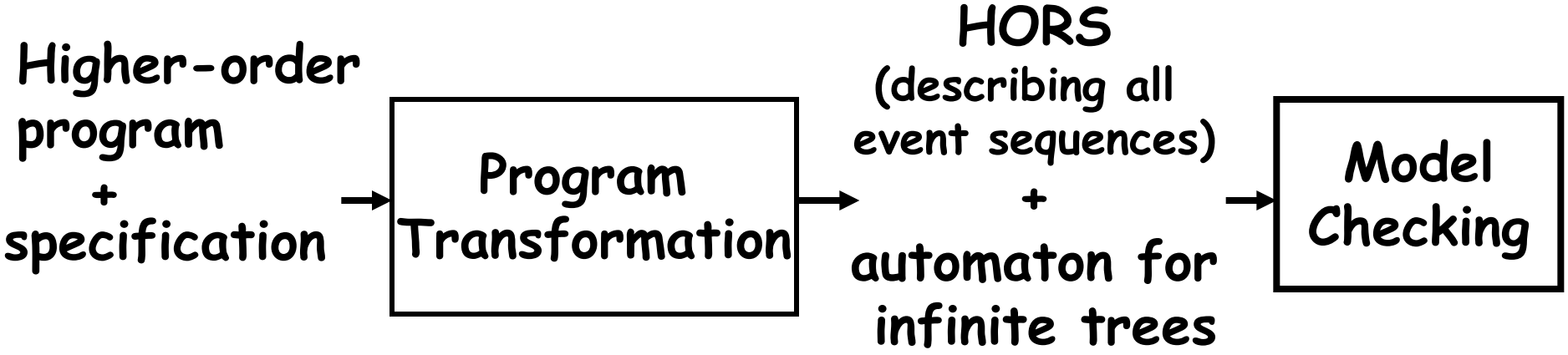
From Program Verification to HO Model Checking



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, ...

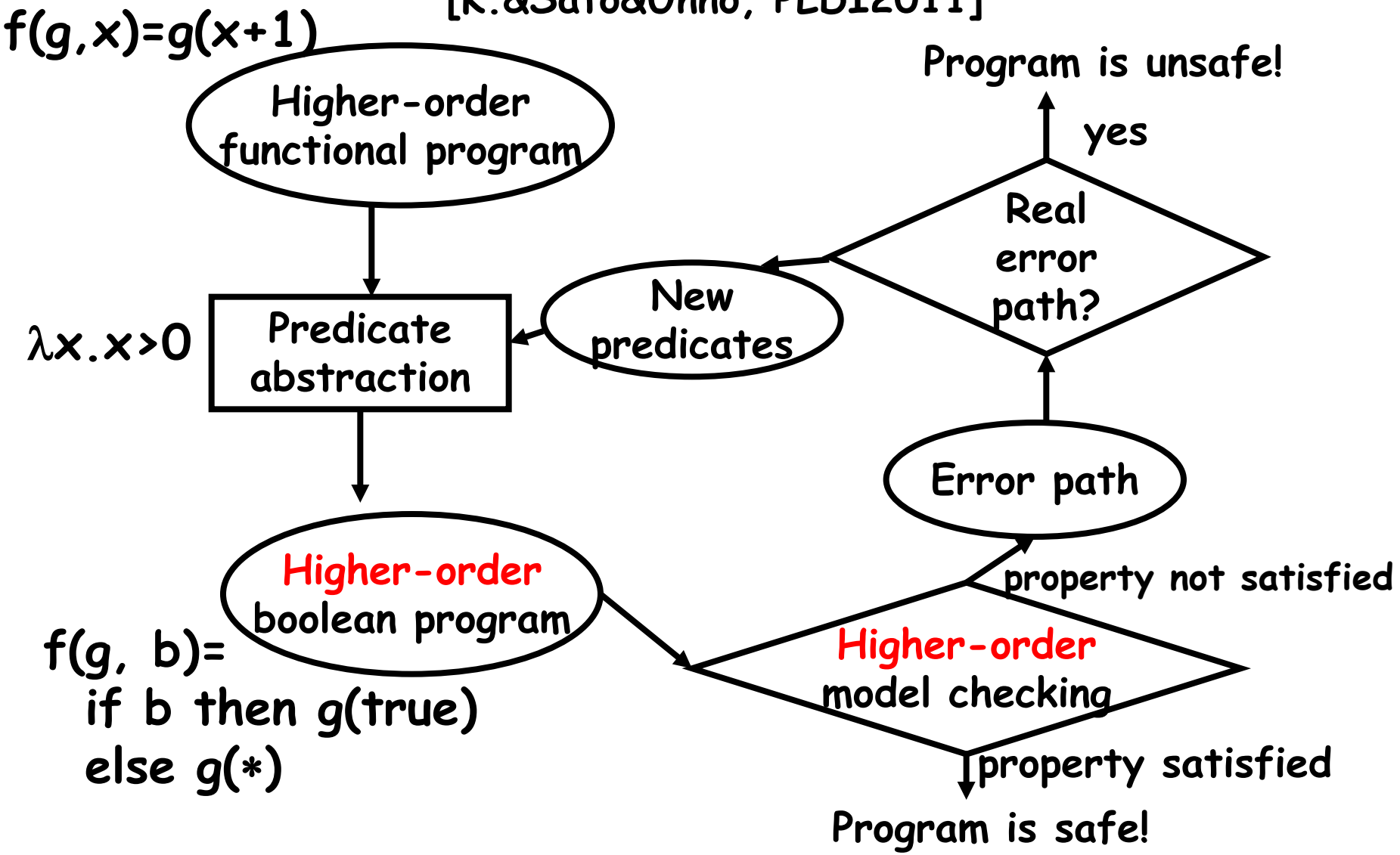
From Program Verification to HO Model Checking



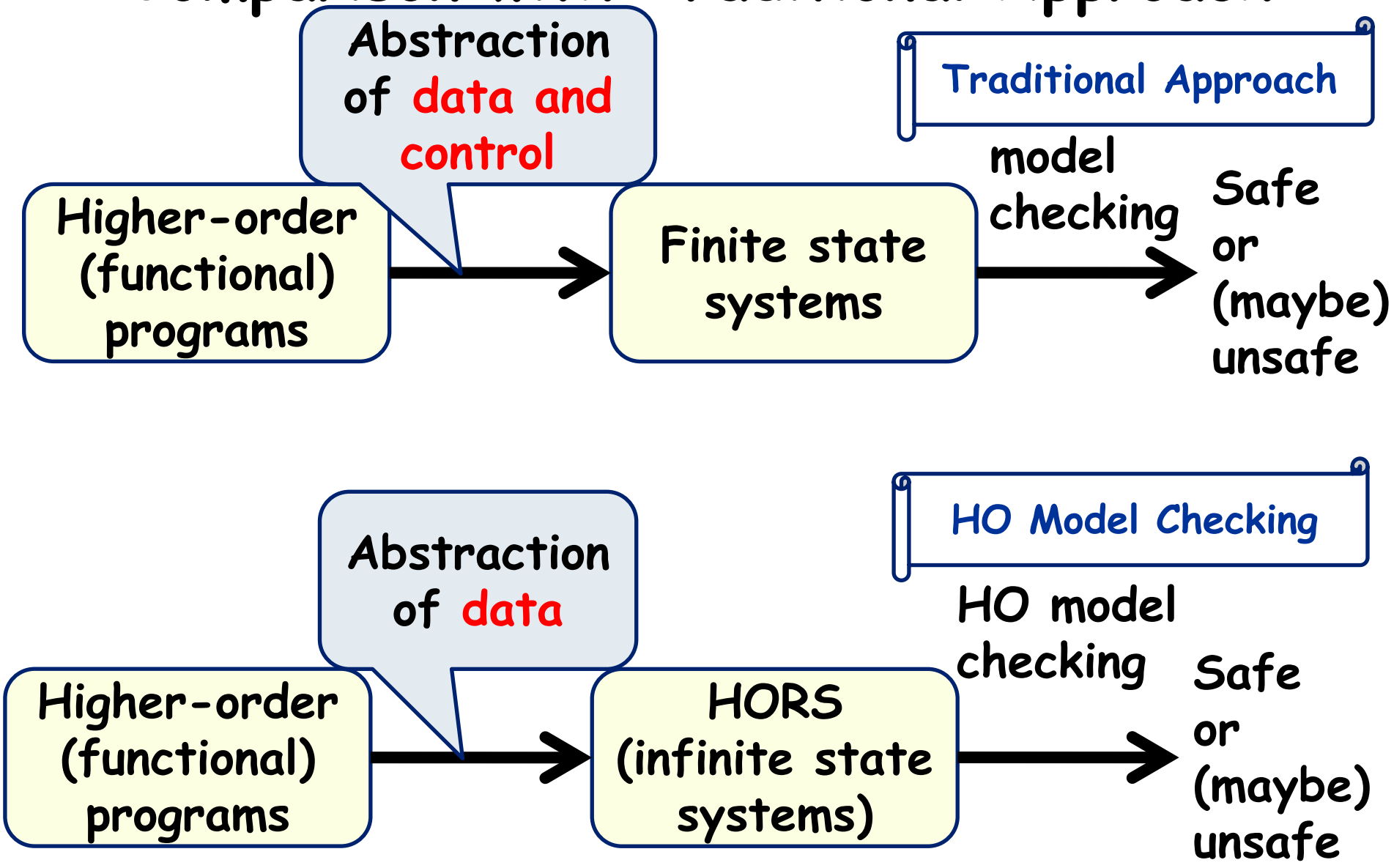
For finite-data HO programs, automated verification comes for free from HO model checking!

Predicate Abstraction and CEGAR for Higher-Order Model Checking

[K.&Sato&Unno, PLDI2011]



Comparison with Traditional Approach



Outline

- Introduction
  - HO model checking
  - Applications to program verification
- Overview of our HO model checking project
- Type-based approach to HO model checking

HO Model Checking Project at UTokyo

- Funded by JSPS Kakenhi, Scientific Research (S) (1st term: 2011-2014, 2nd term: 2015-2019)
- Researchers
  - Principal Investigator: Naoki Kobayashi
  - Collaborators:
    - Atsushi Igarashi (Kyoto), Ayumi Shinohara (Tohoku), Hiroshi Unno (Tsukuba), Tachio Terauchi (JAIST), ...
  - 3-4 Postdocs

HO Model Checking Project at UTokyo

- Topics:
  - Higher-order model checking
    - theoretical foundations
    - algorithms and implementation
  - Applications to program verification
  - Applications to data compression

Theoretical Foundations for HO model checking

- Properties of higher-order grammars
  - pumping lemmas
    - partial result: [K, LICS13]
  - context-sensitivity
    - partial result: [K+, FoSSaCS 14]
- HO model checking
  - theoretical justification for why HO model checking works in practice

HO Model Checking: Algorithms and Implementation

- Trivial automata model checking
  - TRecS [K 2009]
    - first implementation of HO model checker
  - GTRecS [K 2011]
    - first FPT algorithm
  - HorSat [Broadbent&K 2013] , HorSatZDD [Terao&K 2014]
    - saturation-based
  - HorSat2 [K 2015]
    - current state-of-the-art: scales up to 100,000 rules

(model checkers from other groups:
TracMC [Neatherway+2012], CShore [Broadbent+ 2013], Preface [Ramsay+ 2014])

HO Model Checking: Algorithms and Implementation

- APT model checking
  - APTRecS [Fujima&K 2013]
    - TrecS-style algorithm
  - HorSatP [Fujima 2015]
    - HorSat-style algorithm
    - current state-of-the-art, but not fast enough

  (model checkers from other groups:
  Thors [Lester+ 2012?], TracMC2 [Neatherway+2014])

HO Model Checking Project at UTokyo

- Topics:
  - Higher-order model checking
    - theoretical foundations
    - algorithms and implementation
  - Applications to program verification
  - Applications to data compression

Research Topics on Applications to Program Verification

- Expanding the class of programs
- Expanding the class of properties
- Improving efficiency

Research Topics on Applications to Program Verification

- Expanding the class of programs
  - Predicate abstraction and CEGAR for integers [K+, 2011]
  - Encoding algebraic types and exceptions [Sato+ 2013]
  - Extension of HO model checking with recursive types for OO&threads [K&Igarashi 2013][K&Li 2015]
  - Higher-order multi-threaded programs [Yasukata+ 2014]
- Expanding the class of properties
- Improving efficiency

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

- Encoding approach [Sato+ PEPM13] :
  - algebraic data as functions

length function from indices to elements

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

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

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

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

…

Research Topics on Applications to Program Verification

- Expanding the class of programs
- Expanding the class of properties
  - safety properties [K 09][K+ 11]
  - exact flow analysis [Tobita+ 12]
  - relational properties [Sato+ 14] (Sato's talk)
  - termination [Kuwahara+ 14]
  - non-termination [Kuwahara+ 15]
  - fair termination [Murase+ 16] (Terauchi's talk)
  - fair non-termination (ongoing)
- Improving efficiency

Research Topics on Applications to Program Verification

- Expanding the class of programs
- Expanding the class of properties
  - safety properties [K 09][K+ 11]
  - exact flow analysis [Tobita+ 12]
  - relational properties [Sato+ 14] (Sato's talk)
  - termination [Kuwahara+ 14]
  - non-termination [Kuwahara+ 15]
  - fair termination [Murase+ 16] (Terauchi's talk)
  - fair non-termination (ongoing)
- Improving efficiency

Trivial automata model checkers used as backend

Research Topics on Applications to Program Verification

- Expanding the class of programs
- Expanding the class of properties
  - safety properties [K 09][K+ 11]
  - exact flow analysis [Tobita+ 12]
  - relational properties [Sato+ 14] (Sato's talk)
  - termination [Kuwahara+ 14]
  - non-termination [Kuwahara+ 15]
  - fair termination [Murase+ 16] (Terauchi's talk)
  - fair non-termination (ongoing)

APT model checker used as backend

- Improving efficiency

Research Topics on Applications to Program Verification

- Improving efficiency
  - improving HO model checkers
    - Currently, trivial automata model checkers are fast enough, but APT model checkers are not.
  - predicate discovery
    - by reduction to Horn clause solving [Unno&K 09]
    - current major drawback of MoCHi
      - better Horn clause solving
      - machine learning? (ongoing)
  - modular verification (ongoing)
    - from refinement type checking to reachability checking [Sato+ 15]

HO Model Checking Project at UTokyo

- Topics:
  - Higher-order model checking
    - theoretical foundations
    - algorithms and implementation
  - Applications to program verification
  - Applications to data compression

Applications to Data Compression

- Compressed data as higher-order grammars (c.f. Kolmogorov complexity)
  - Hyper-exponential compression ratio
- Data processing without decompression using higher-order model checking

Compressed Data as HORS

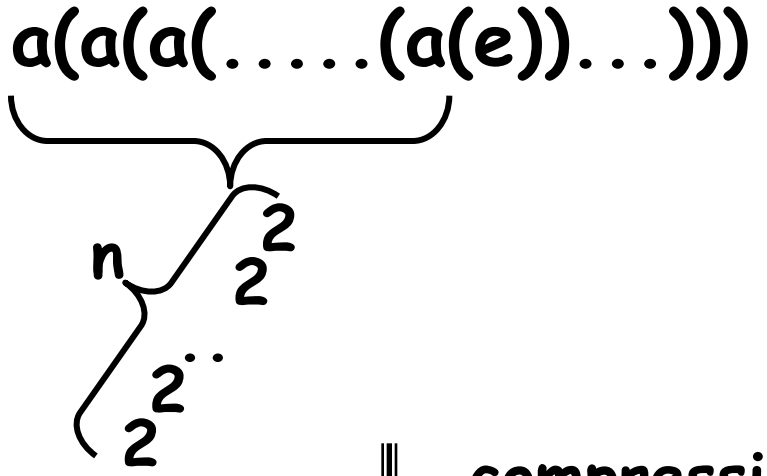
$\underbrace{a(a(a(.....(a(e))...)))}_{2^n}$

Compression ratio : $O(n/2^n)$

$S = \underbrace{Twice(Twice(...(Twice}_{n}\ a)...))\ e$

Twice f x = f(f(x))

Compressed Data as HORS



compression

$S = \underbrace{((\text{Twice Twice}) \ldots \text{Twice})}_{n}\ a\ e$

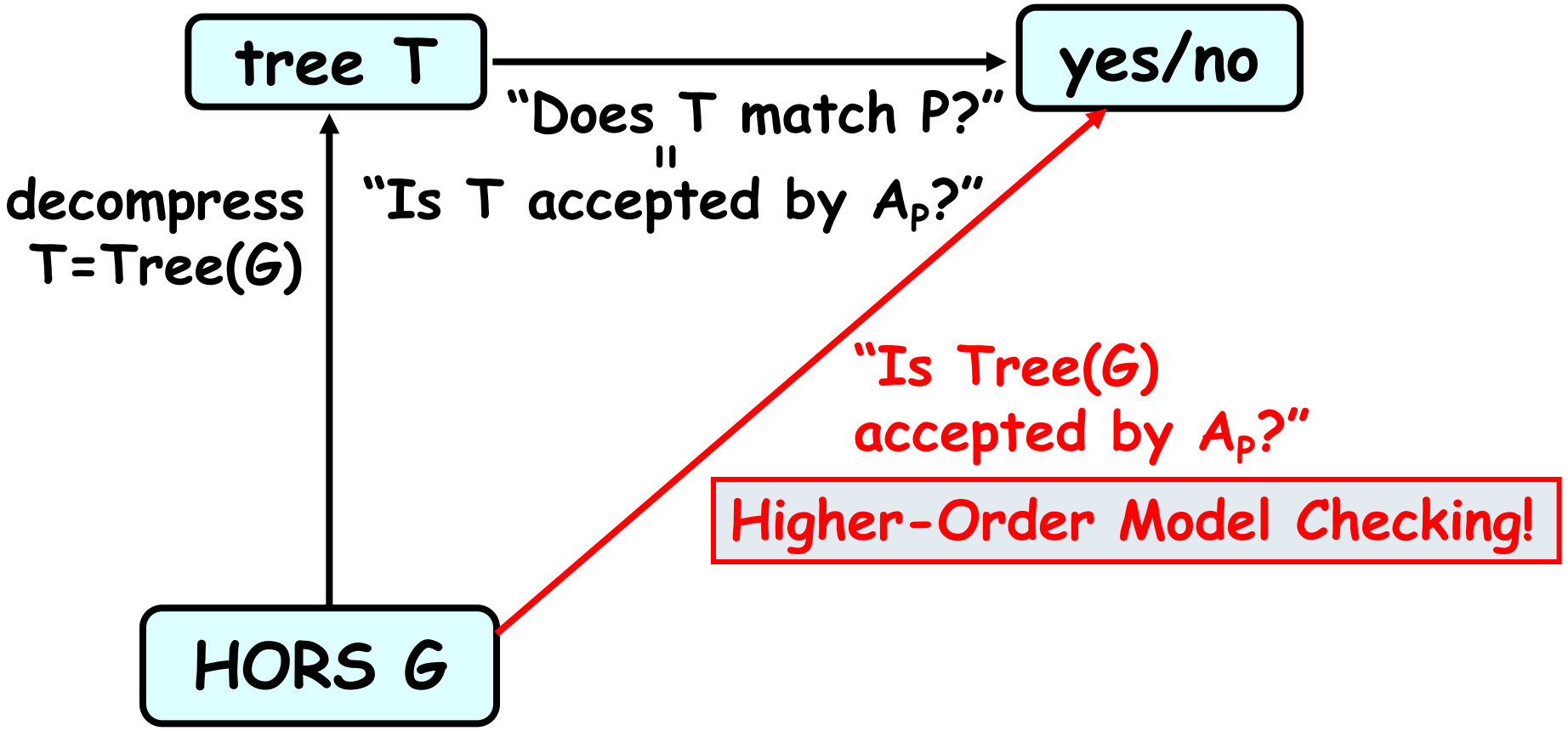
Twice f x = f(f(x))

Applications to Data Compression

- Compressed data as higher-order grammars
  - Hyper-exponential compression ratio

- Data processing without decompression using higher-order model checking
  - pattern match queries
  - associated data processing to compute:
    - matching positions
    - the number of matches
    - ... (whatever expressed by transducers)

Goal: data processing without decompression



Example: a Fibonacci word

Fibonacci word:

$w_0=b,\ w_1=a,\ w_2=w_1w_0=ab,\ w_3=w_2w_1=aba,\ldots,$

$w_n=w_{n-1}w_{n-2}$

Compression (case $n=2^m$)

m

Main = Twice(Twice(...(Twice Next)..)) Fst b a e

Next k u v = k v (concat v u)

Concat f g x = f(g(x))

Twice f x = f(f(x))

Query: Does w_{1024} contain "bb"?

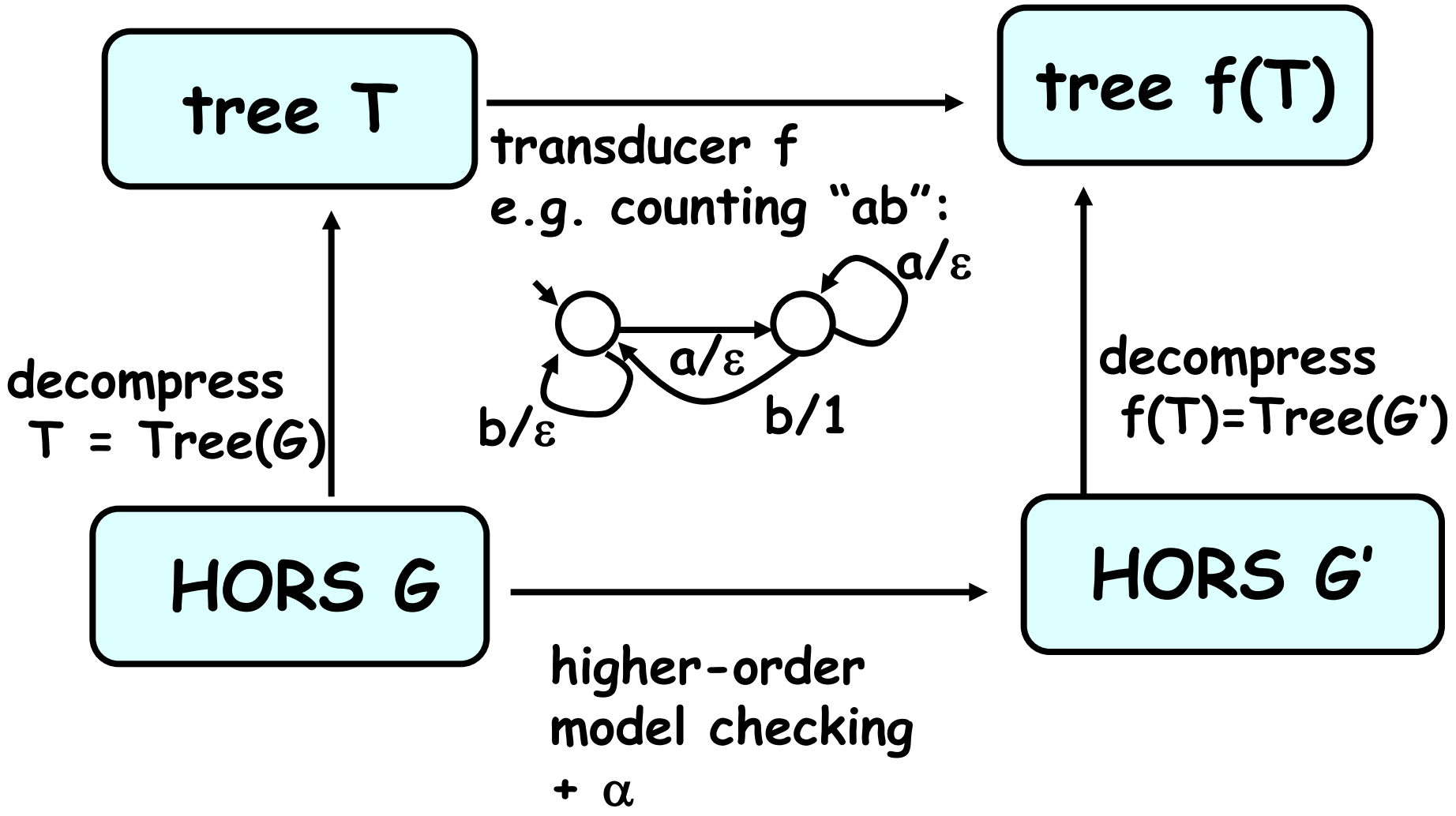
(Note: $|w_{1024}| > 10^{200}$)

Applications to Data Compression

- Compressed data as higher-order grammars
  - Hyper-exponential compression ratio

- Data processing without decompression using higher-order model checking
  - pattern match queries
  - associated data processing to compute:
    - matching positions
    - the number of matches
    - ... (whatever expressed by transducers)

Data Transformation without Decompression



Example: a Fibonacci word

Fibonacci word:

$w_0=b,\ w_1=a,\ w_2=w_1w_0=ab,\ w_3=w_2w_1=aba,\ldots,$
$w_n=w_{n-1}w_{n-2}$

Compression (case $n=2^m$)

m

Main = Twice(Twice(...(Twice Next)..)) Fst b a e
Next k u v = k v (concat v u)
Concat f g x = f(g(x))
Twice f x = f(f(x))

Query: How many occurrences of "ab" does w_{16} contain?

Applications to Data Compression: Current Status

- Higher-order data compressor
  - Naive algorithm [K+ 2012]
  - RePair-style, almost linear-time algorithm [Takeda+ 2014]
  - better algorithm? (ongoing)
- Data transformer (without compression)
  - Implemented as an extension of GTRecS model checking algorithm [K+ 2012]
    - currently not a major bottleneck, but possibly for larger data

HO Model Checking Project at UTokyo

- Topics:
  - Higher-order model checking
    - theoretical foundations
    - algorithms and implementation
  - Applications to program verification
  - Applications to data compression

Conclusion

- Higher-order model checking has rich applications:
  - fully automated, precise program verification
  - data compression
    (manipulation of data without decompression)
- Higher-order model checking works in practice, despite the k-EXPTIME complexity
- A lot of interesting theoretical/practical challenges remain
  - open problems on properties of HO languages
  - more scalable verification tools?