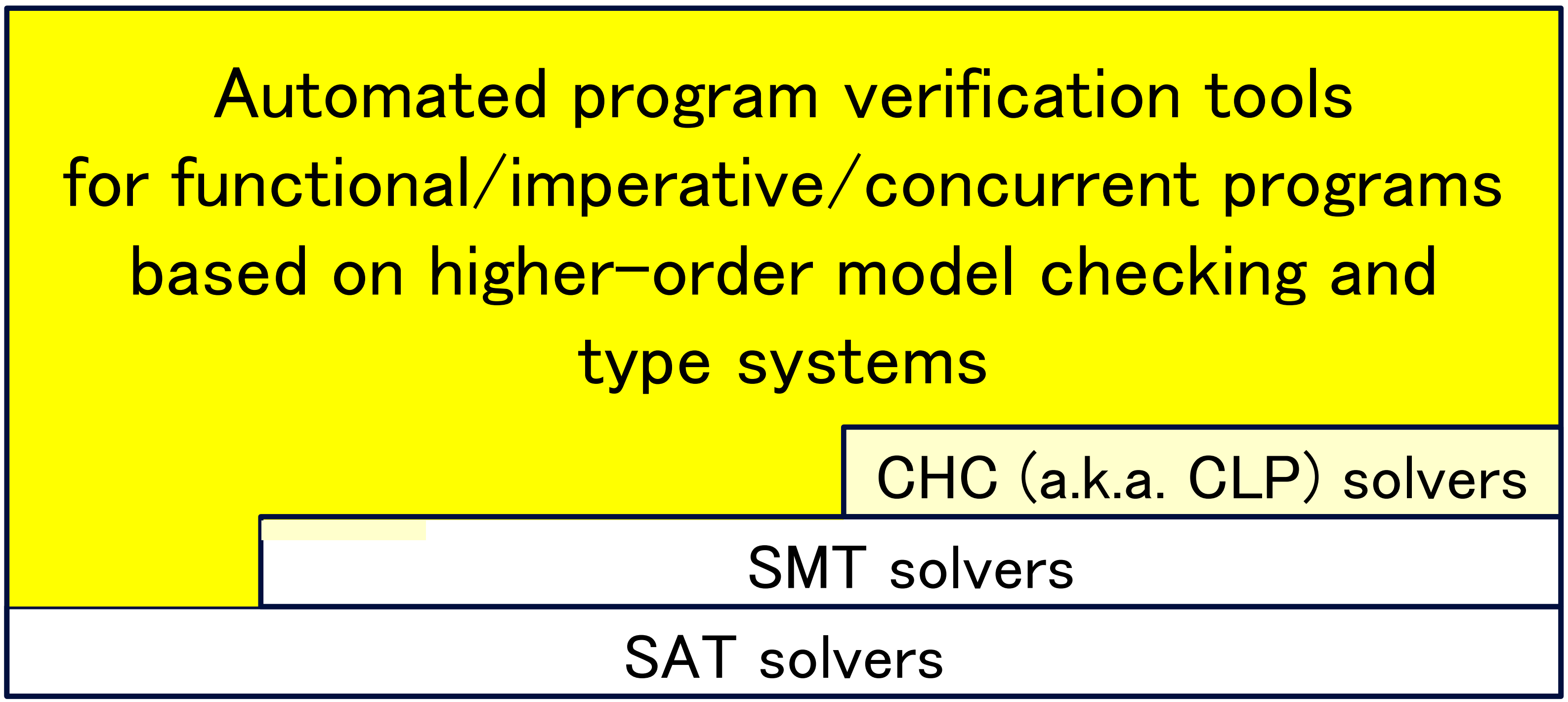
Higher-Order Model Checking and its Similarity (?) with SAT Problem

Naoki Kobayashi
The University of Tokyo

Self Introduction

- **Working on theory & practice for automated program verification**



Outline

- A brief introduction to higher-order model checking (HOMC)
  - What is HOMC?
  - Applications
- Why HOMC works in practice?
  - similarity/difference with SAT

Two Notions of Higher-Order Model Checking

	Models	Logic
finite state model checking	finite state systems	modal μ-calculus

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

A higher-order tree grammar,
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 (or tree automata)
HFL model checking [Viswanathan& Viswanathan 04]	finite state systems	higher-order modal fixpoint logic (HFL)

Higher-Order Recursion Scheme (HORS)

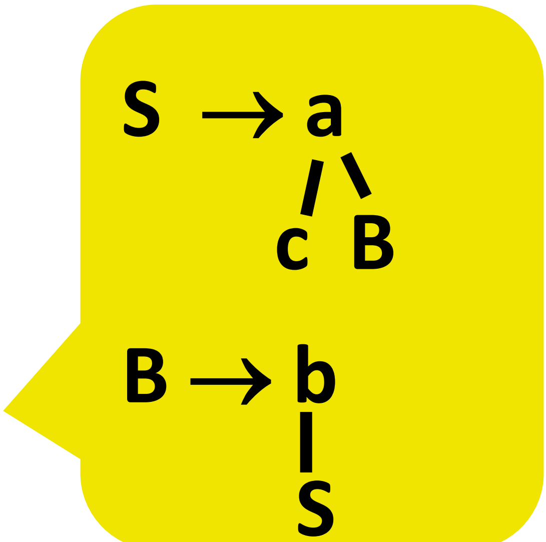
- Grammar for generating an infinite tree

Order-0 HORS

(regular tree grammar)

S → a c B

B → b S



Higher-Order Recursion Scheme (HORS)

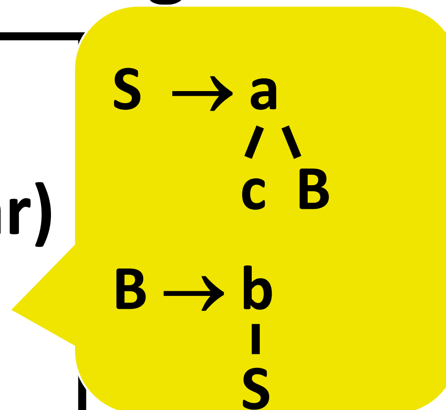
- Grammar for generating an infinite tree

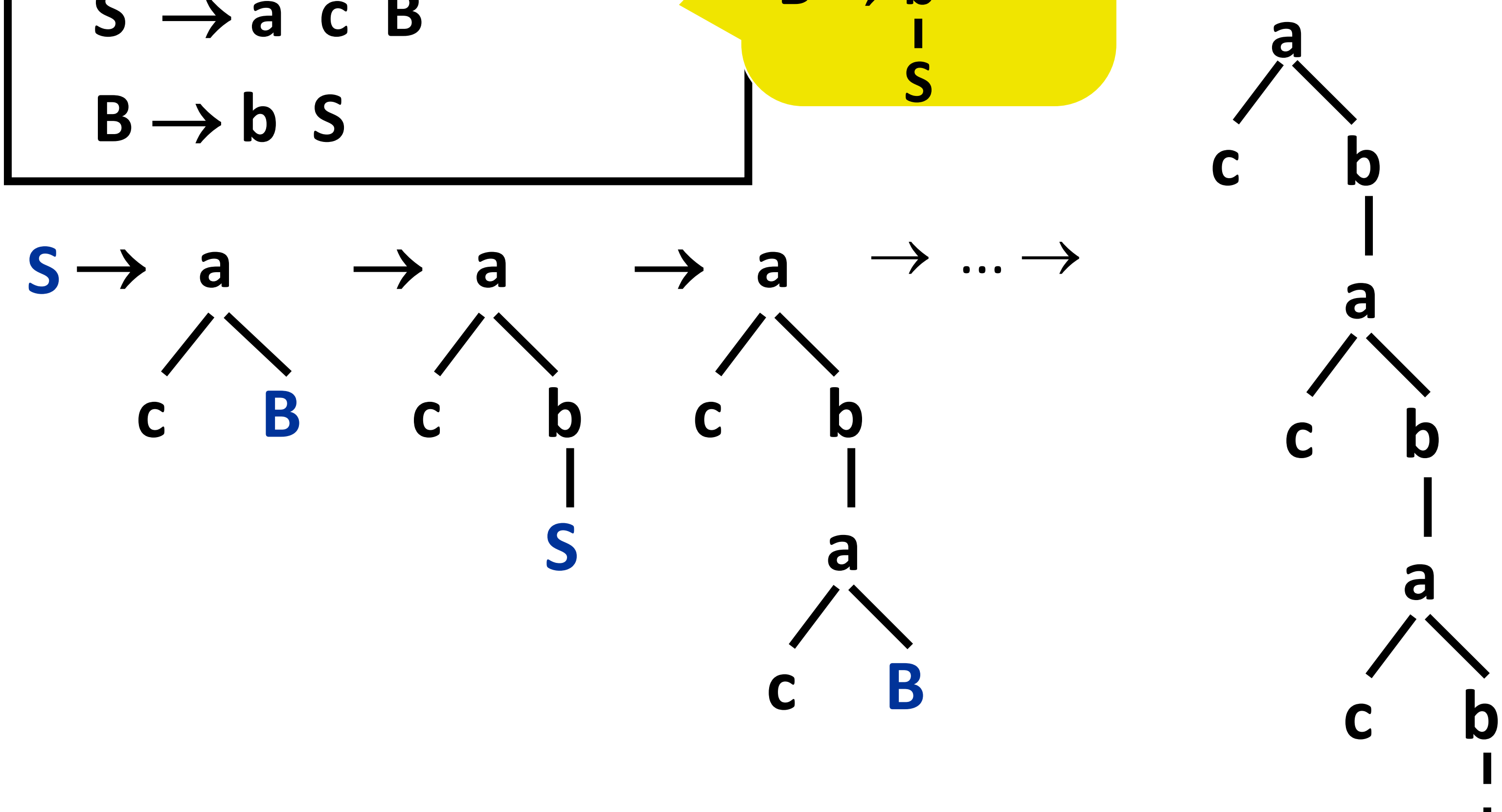
Order-0 HORS

(regular tree grammar)

S → a c B

B → b S





Higher-Order Recursion Scheme (HORS)

- Grammar for generating an infinite tree

Order-1 HORS

$S \rightarrow A\ c$

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

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

Notable restriction (compared with ordinary functional programs):
- Rules must be simply-typed.
- There are no pattern matching on trees.

Higher-Order Recursion Scheme (HORS)

- Grammar for generating an infinite tree

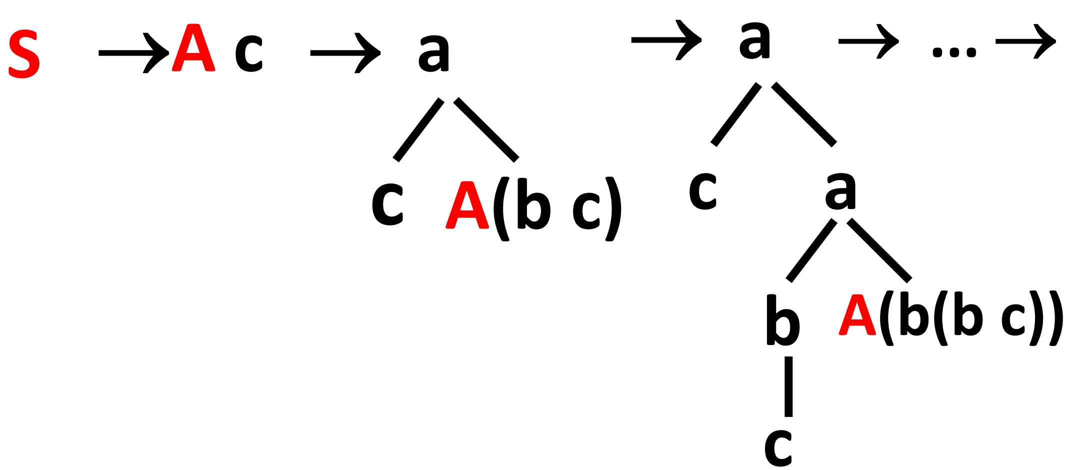
Order-1 HORS

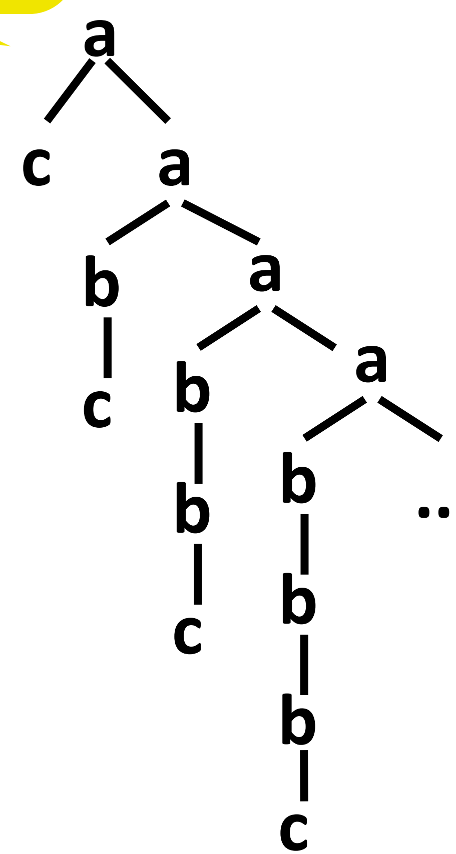
$S \rightarrow A\ c$

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

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

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

≈

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”?**

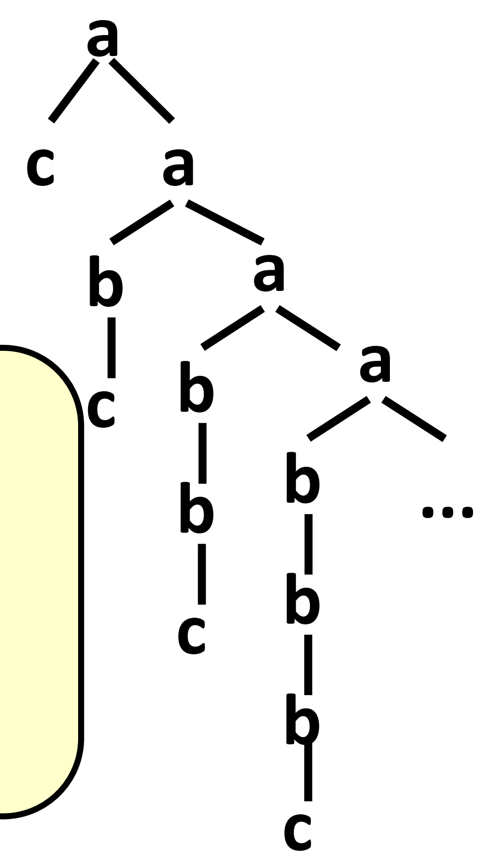
HORS Model Checking

Order-1 HORS

S $\rightarrow$ A c

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

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

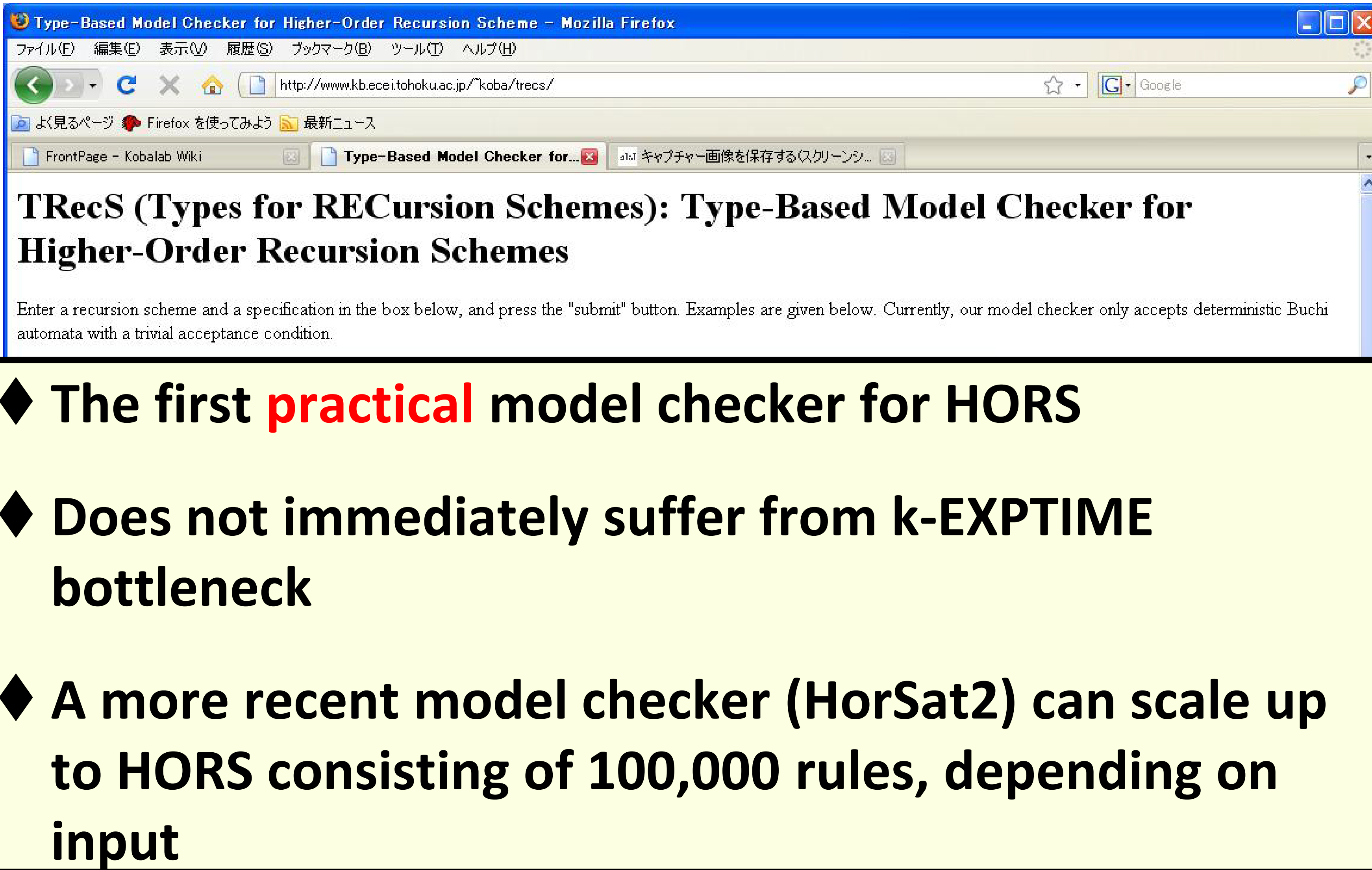
e.g.
- **Does every finite path end with “c”?**
- **Does “a” occur below “b”?**

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

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

TRecS [K. PPDP09]

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

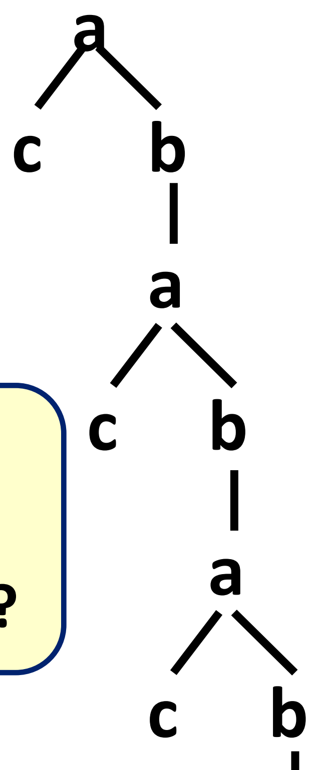


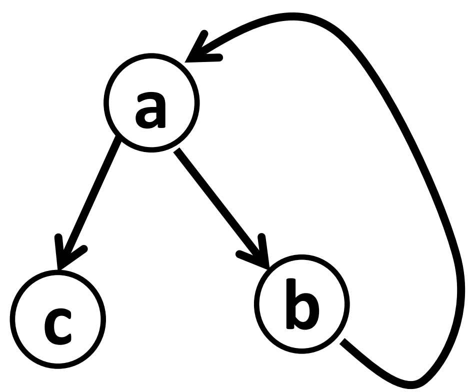
- The first practical model checker for HORS
- Does not immediately suffer from k-EXPTIME bottleneck
- A more recent model checker (HorSat2) can scale up to HORS consisting of 100,000 rules, depending on input

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**





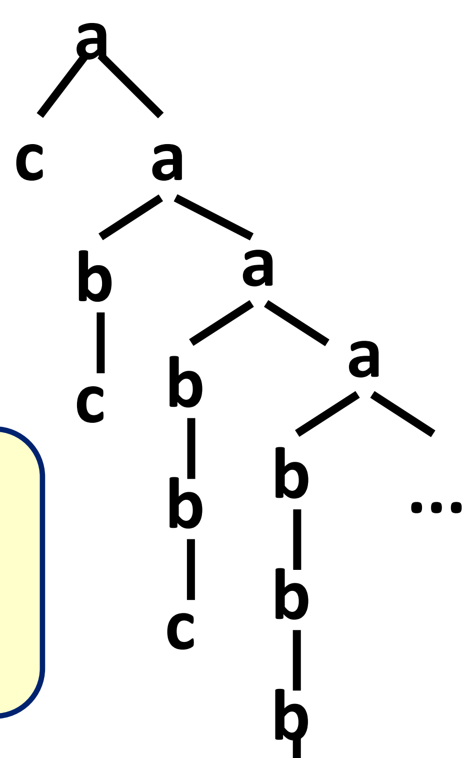
Does "a" occur below "b"?

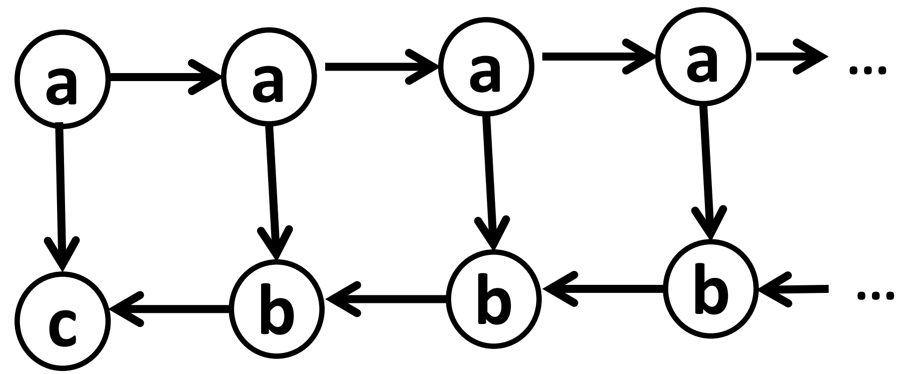
Is there a transition sequence in which "a" occurs after "b"?

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"?

Encoding QBF

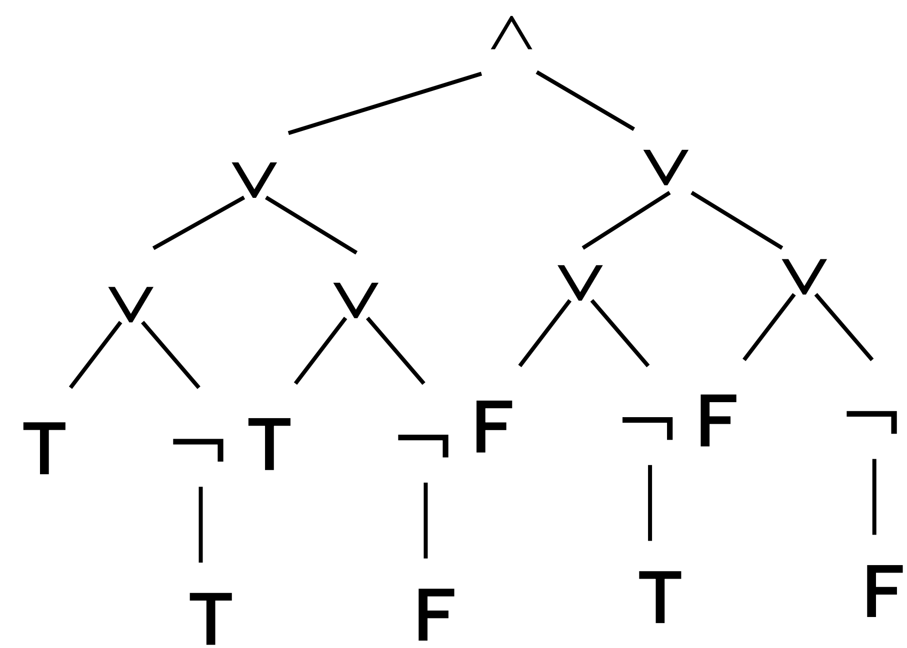
QBF: $\forall x. \exists y. (x \lor \neg y)$

HORS:

S = Forall (λx. Exists λy. $\lor$ x ($\neg$ y))

Forall f = $\land$ (f T) (f F)

Exists f = $\lor$ (f T) (f F)



Encoding QBF

QBF: $\varphi := \forall x.\ \exists y.\ (x \vee \neg y)$

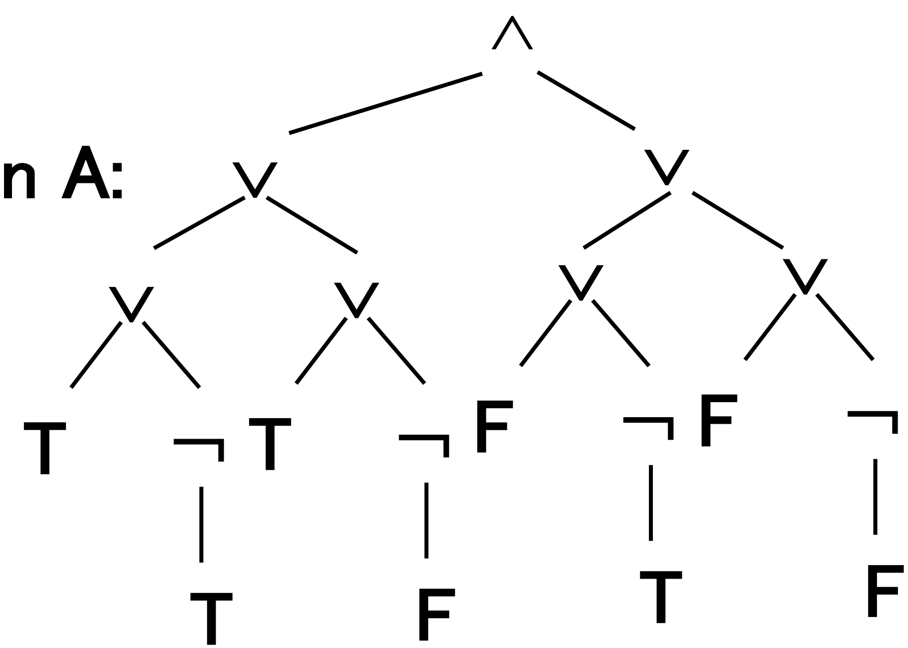
HORS G:

S = Forall (λx. Exists λy. $\vee$ x ($\neg$ y))

Forall f = $\wedge$ (f T) (f F)

Exists f = $\vee$ (f T) (f F)

(Bottom-up) tree automaton A:



$\wedge$ T T $\rightarrow$ T

$\wedge$ _ F $\rightarrow$ F

$\wedge$ F _ $\rightarrow$ F

$\neg$ T $\rightarrow$ F

$\vee$ F F $\rightarrow$ F

$\vee$ _ T $\rightarrow$ T

$\vee$ T _ $\rightarrow$ T

$\neg$ F $\rightarrow$ T

(with final state: T)

φ is true $\Leftrightarrow$ A accepts Tree(G)

Encoding QBF

QBF: $\varphi := \forall x.\ \exists y.\ (x \vee \neg y)$

HORS G:

$S = \text{Forall}\ (\lambda x.\ \text{Exists}\ \lambda y.\ \vee\ x\ (\neg\ y))$

$\text{Forall}\ f = \wedge\ (f\ T)\ (f\ F)$

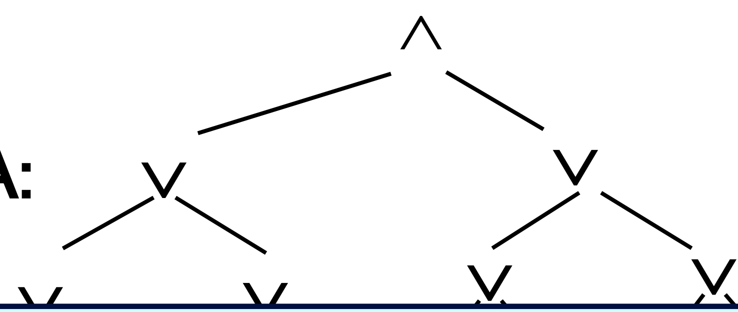
$\text{Exists}\ f = \vee\ (f\ T)\ (f\ F)$

(Bottom-up) tree automaton A:

$\wedge\ T\ T \rightarrow T$	$\vee\ F\ F \rightarrow F$
$\wedge\ \_\ F \rightarrow F$	$\vee\ \_\ T \rightarrow T$
$\wedge\ F\ \_ \rightarrow F$	$\vee\ T\ \_ \rightarrow T$
$\neg\ T \rightarrow F$	$\neg\ F \rightarrow T$

(with final state: T)

φ is true $\Leftrightarrow$ A accepts Tree(G)



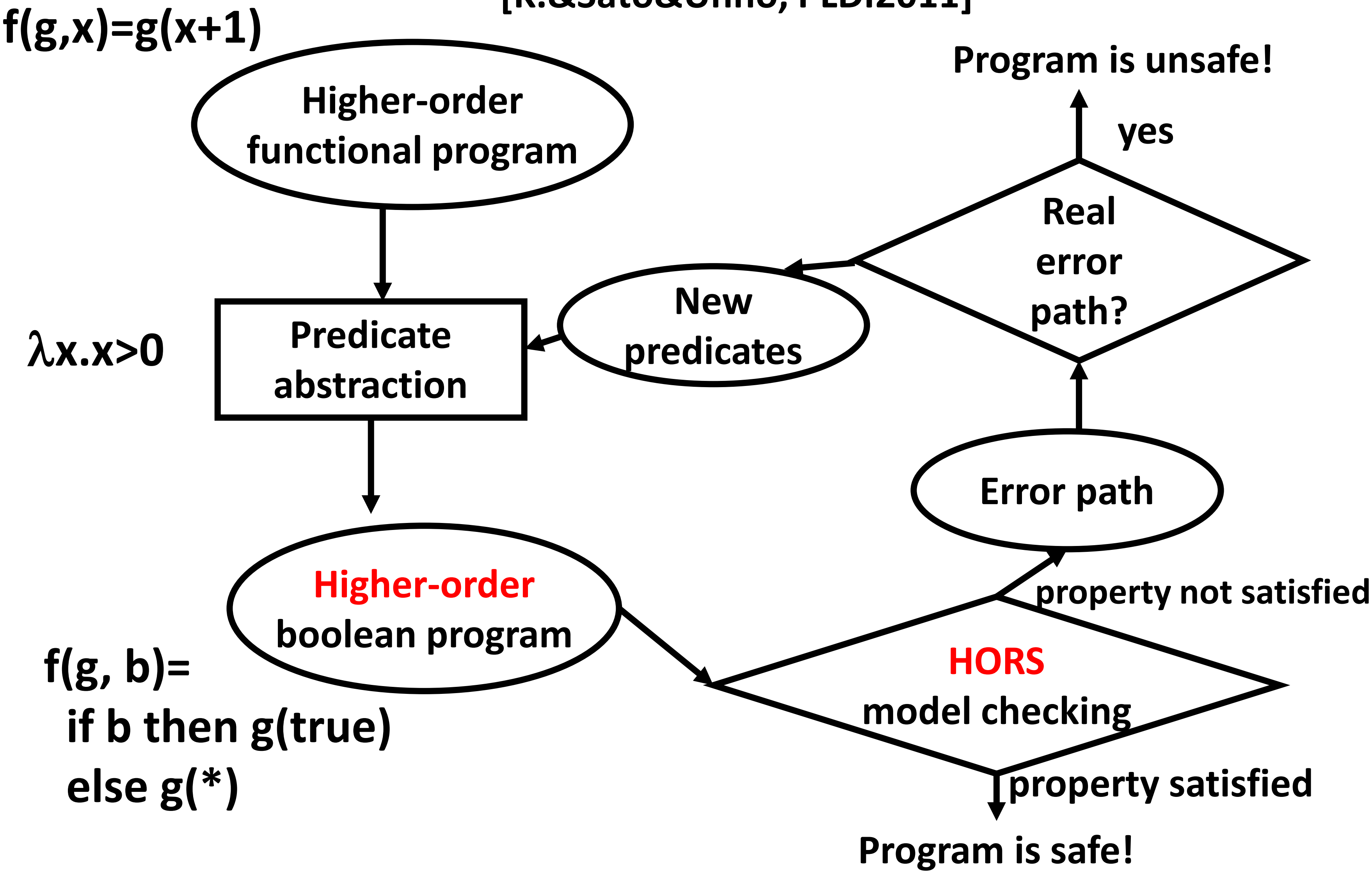
Remarks:
- HOMC for recursion-free order-1 HORS is PSPACE-complete
- May be useful when a formula is large but can be compactly represented by HORS

Outline

- A brief introduction to higher-order model checking (HOMC)
  - What is HOMC?
  - Applications to program verification
- Why HOMC works in practice?
  - similarity/difference with SAT

Predicate Abstraction and CEGAR for HORS Model Checking

[K.&Sato&Unno, PLDI2011]



Tool demonstration: MoChi

[K&Sato&Unno, 2011]

https://www.kb.is.s.u-Tokyo.ac.jp/~Ryosuke/mochi/

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

Outline

- A brief introduction to higher-order model checking (HOMC)
  - What is HOMC?
  - Applications to program verification
- Why HOMC works in practice?
  - similarity/difference with SAT

Why HORS Model Checking Works (despite k-EXPTIME completeness)

- ♦ Fixed-parameter tractable (FPT) in the size of G (fixed parameters: the largest size of types, the size of formulas)
- ♦ Given a “certificate” (intersection types), the validity of the certificate (for both yes/no answers) can be efficiently checked.
  (cf. NP problem)
  - Hypothesis: Many HO model checking problems (obtained from program verification problems) tend to have small certificates

Empirical Evidence for "Small Certificate" Hypothesis?

	order	rules	states	#cert.	$\|\Gamma_{max}\|$
Twofiles	4	11	5	37	$>10^{10\hat{}10\hat{}49}$
TwofilesE	4	12	5	42	$>10^{10\hat{}10\hat{}49}$
FileOcamlC	4	23	4	41	$>10^{10\hat{}10\hat{}20}$
Lock	4	11	4	41	$>10^{10\hat{}10\hat{}20}$
Order5	5	9	5	43	$>10^{10\hat{}10\hat{}10\hat{}48}$
mc91	4	49	1	115	$>10^{10\hat{}80}$
xhtml	2	64	50	101	$>10^{753}$
exp4-5-3	4	12	3	137	$>10^{10\hat{}10\hat{}7}$

#cert: the number of type bindings
in the certificate found by a model checker

Why HORS Model Checking Works (despite k-EXPTIME completeness)

- Fixed-parameter tractable (FPT) in the size of G (fixed parameters: the largest size of types, the size of formulas)
- Given a "certificate" (intersection types), the validity of the certificate (for both yes/no answers) can be efficiently checked.
 (cf. NP problem)
- High complexity is due to the expressive power of HORS (a finite state system of k-EXP(n) states can be represented in O(n)-size grammar)

HORS describing a finite-state system with k-EXP(m) states

Order-n HORS $R_{m,k}$

$S \to F_0\ G_{k-1}\ \ldots\ G_2\ G_1\ G_0$

$F_0\ f \to F_1\ (F_1\ f)$

$\ldots$

$F_{m-1}\ f \to F_m\ (F_m\ f)$

$F_m\ f \to G_n\ f$

$G_k\ f\ z \to f\ (f\ z)$

$\ldots$

$G_2\ f\ z \to f\ (f\ z)$

$G_1\ z \to a\ z$

$G_0 \to c$

$$S \to^* a^{\overbrace{2^{2^{\cdot^{\cdot^{2^m}}}}}^{k}}\ (G_0)$$

k-EXPTIME algorithm for order-k HORS

$\approx$

Polynomial time algorithm for finite state model checking

HORS describing a finite-state system with k-EXP(m) states

Order-n HORS $R_{m,k}$

$S \to F_0\ G_{k-1}\ \ldots\ G_2\ G_1\ G_0$

$F_0\ f \to F_1\ (F_1\ f)$

$\ldots$

$F_{m-1}\ f \to F_m\ (F_m\ f)$

$F_m\ f \to G_n\ f$

$G_k\ f\ z \to f\ (f\ z)$

$\ldots$

$G_2\ f\ z \to f\ (f\ z)$

$G_1\ z \to a\ z$

$G_0 \to c$

$$S \to^* a^{\left. 2^{2^{\cdot^{\cdot^{2^m}}}} \right\} k}\ (G_0)$$

fixed-parameter polynomial time algorithm for order-k HORS

>

Polynomial time algorithm for finite state model checking

Conclusion

- HOMC subsumes many decision problems at low-order
  - Finite state model checking
  - Pushdown model checking
  - SAT/QBF solving
- Applications to higher-order program verification
- HOMC works despite extremely high complexity
  … as long as there are small certificates
- More efficient HOMC solver using SAT technology?