チュートリアル：
二種類の高階モデル検査と
プログラム検証の関係について

小林　直樹
東京大学

本チュートリアルの内容

- 二種類の高階モデル検査とそれらの相互の関係、プログラム検証への応用について

	Models	Logic
有限状態 モデル検査	有限状態システム	様相 μ計算, LTL/CTL/CTL*

本チュートリアルの内容

- 二種類の高階モデル検査とそれらの相互の関係、プログラム検証への応用について

	Models	Logic
有限状態 モデル検査	有限状態システム	様相 μ計算
HORS モデル検査 [Knapik+ 01; Ong 06]	高階再帰スキーム (HORS)	様相 μ計算

Useful for modeling a certain class of infinite state systems (such as higher-order functional programs)

本チュートリアルの内容

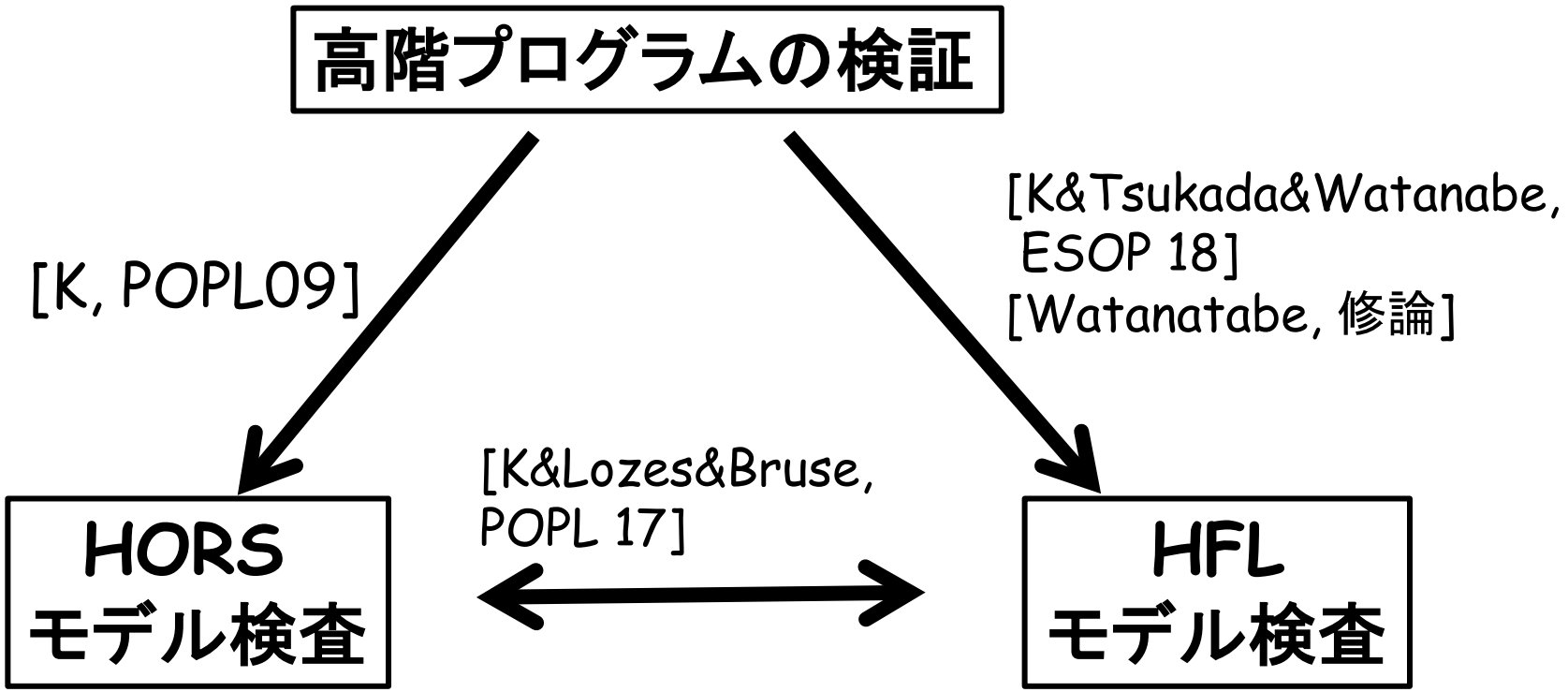
- 二種類の高階モデル検査とそれらの相互の関係、プログラム検証への応用について

	Models	Logic
有限状態 モデル検査	有限状態システム	様相 μ計算
HORS モデル検査 [Knapik+ 01; Ong 06]	高階再帰スキーム (HORS)	様相 μ計算
HFL モデル検査 [Viswanathan&Viswanathan 04]	有限状態システム	高階様相 不動点論理(HFL)

Useful for describing non-regular properties

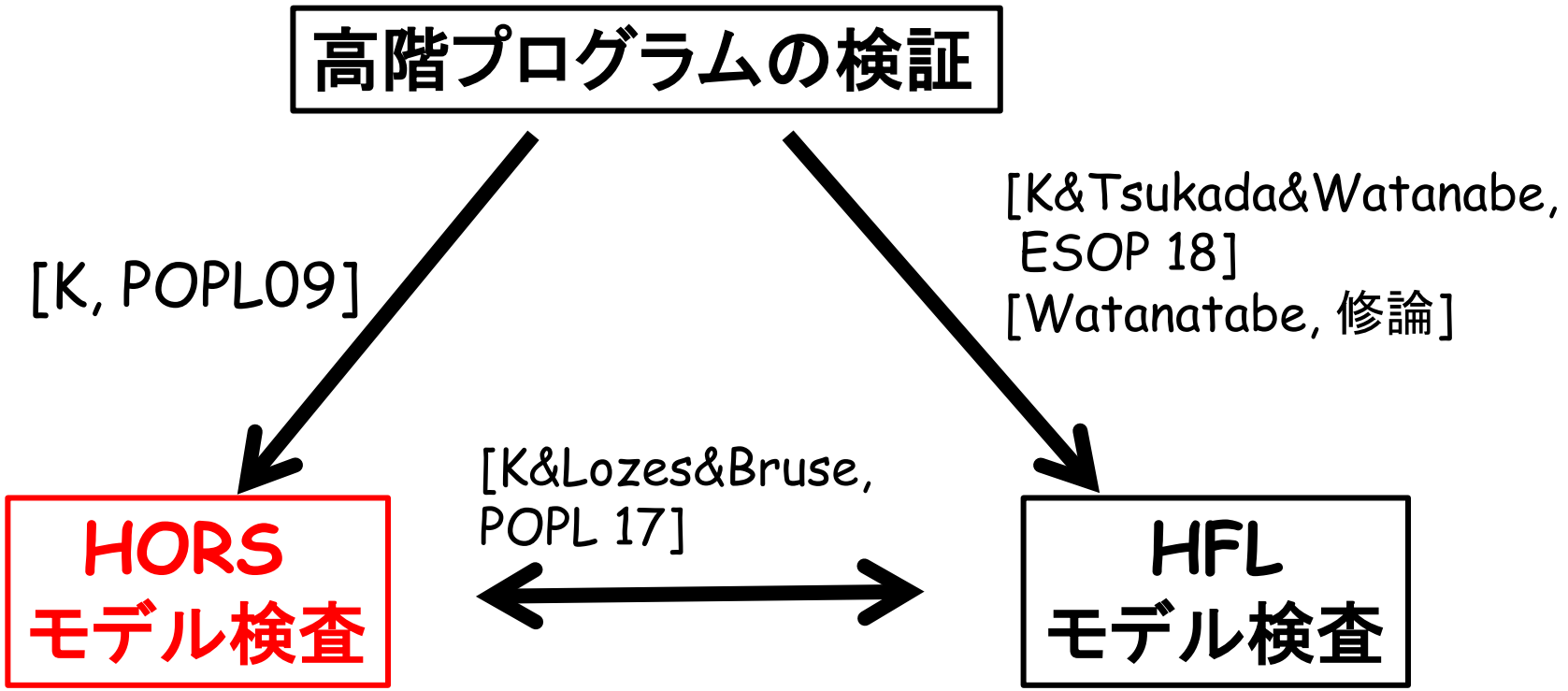
アウトライン

- HORSモデル検査とプログラム検証への応用
- HFLモデル検査とプログラム検証への応用
- HORSモデル検査とHFLモデル検査の関係



アウトライン

- HORSモデル検査とプログラム検証への応用
- HFLモデル検査とプログラム検証への応用
- HORSモデル検査とHFLモデル検査の関係



本チュートリアルの内容

- 二種類の高階モデル検査とそれらの相互の関係、プログラム検証への応用について

	Models	仕様記述
有限状態 モデル検査	有限状態システム	様相 μ計算
HORS モデル検査 [Knapik+ 01; Ong 06]	高階再帰スキーム (HORS)	様相 μ計算 (または木オートマトン)
HFL モデル検査 [Viswanathan&Viswanathan 04]	有限状態システム	高階様相 不動点論理(HFL)

Higher-Order Recursion Scheme (HORS)

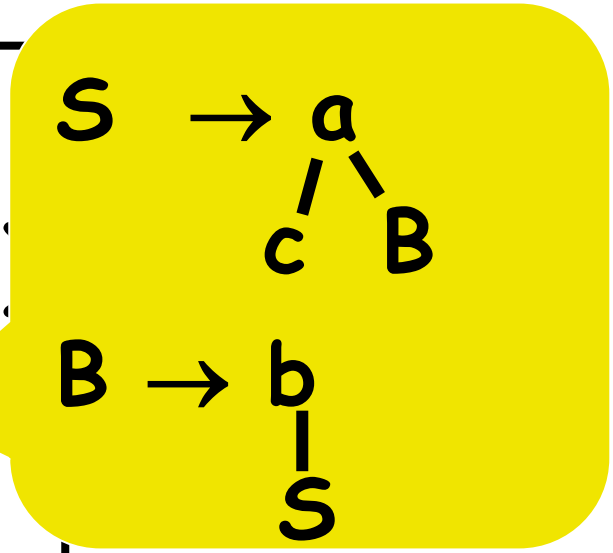
- Grammar for generating an infinite tree

Order-0 HORS

(regular tree grammar)

$S \to a\ c\ B$

$B \to b\ S$



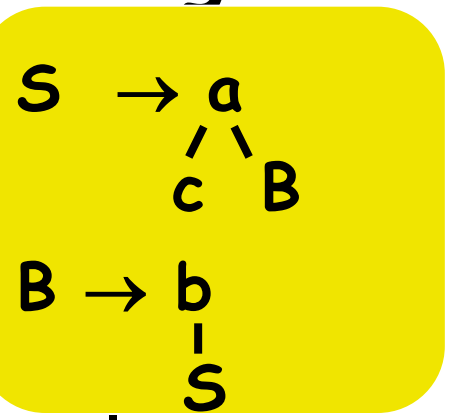
Higher-Order Recursion Scheme (HORS)

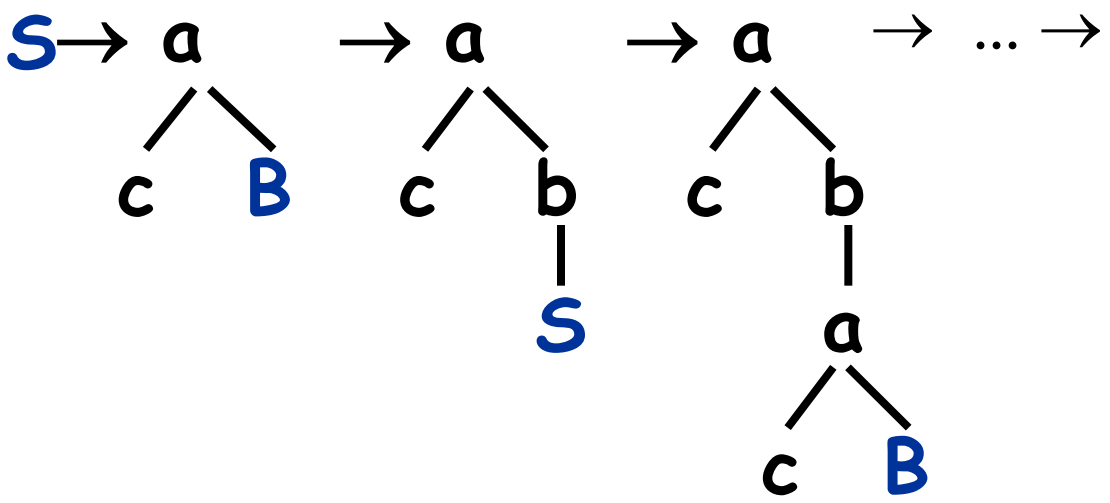
- Grammar for generating an infinite tree

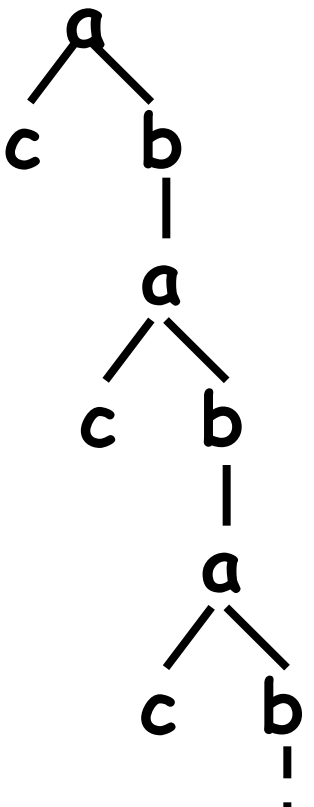
Order-0 HORS
(regular tree gramm

$S \to a\ c\ B$

$B \to b\ S$







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

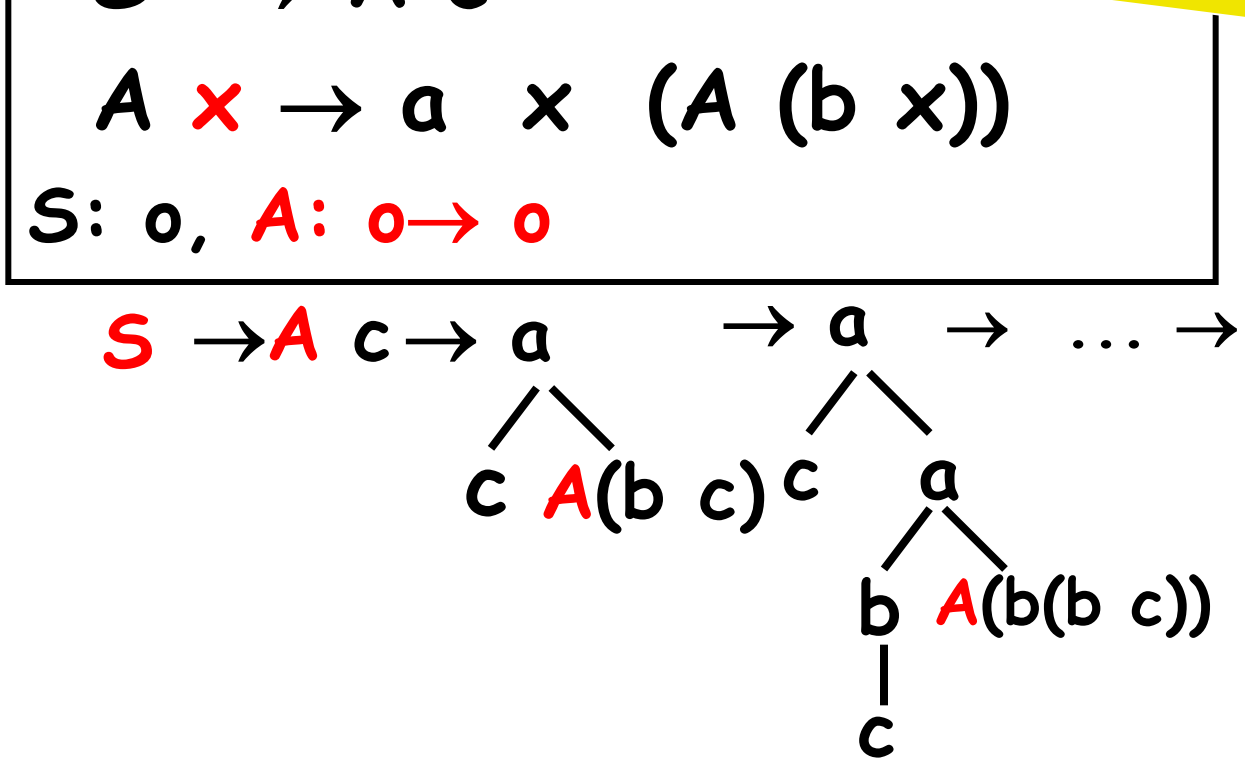
Order-1 HORS

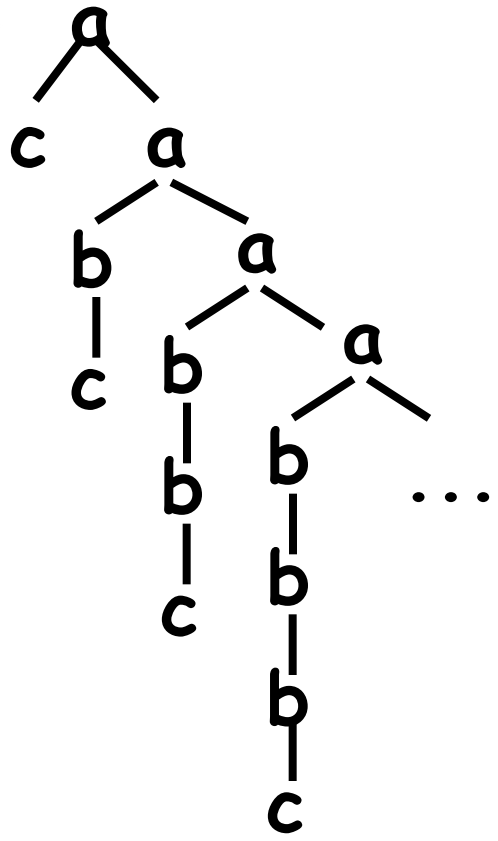
$S \rightarrow A\ c$

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

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

≈

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"?

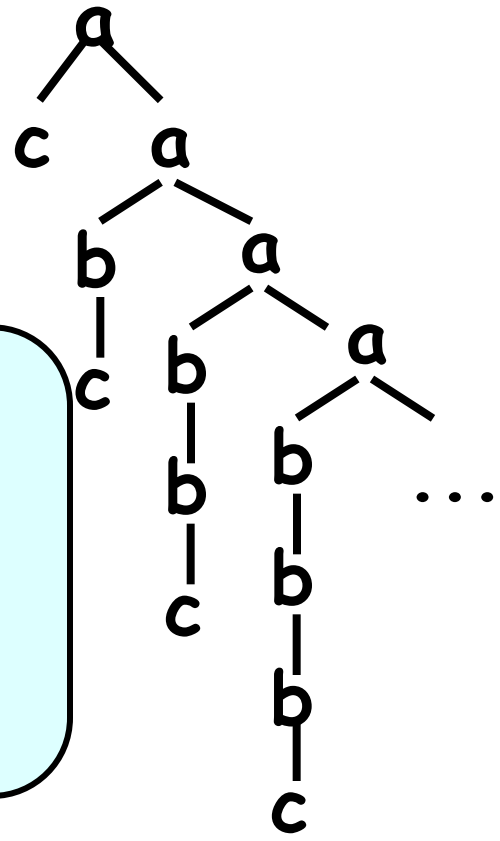
HORS Model Checking

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)

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

TRecS [K. PPDP09]

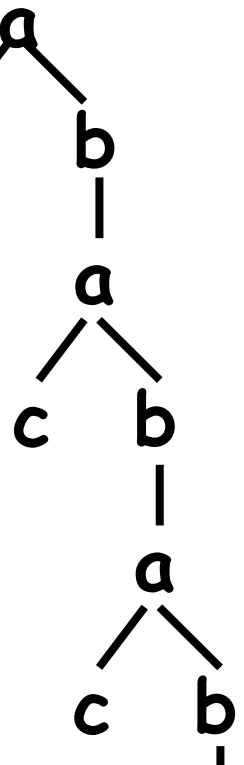
http://www-kb.is.s.u-tokyo.ac.jp/~koba/trecs/

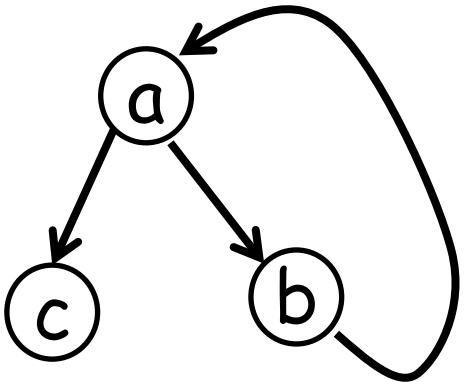
- The first practical model checker for HORS
- Does not immediately suffer from k-EXPTIME bottleneck
- A more recent model checker (HorSat2) can scale up to HORS consisting of 100,000 rules, depending on input

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





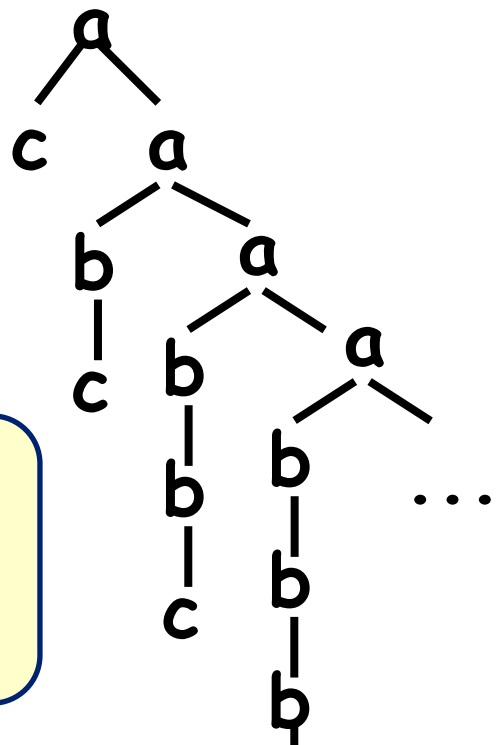
Does "a" occur below "b"?

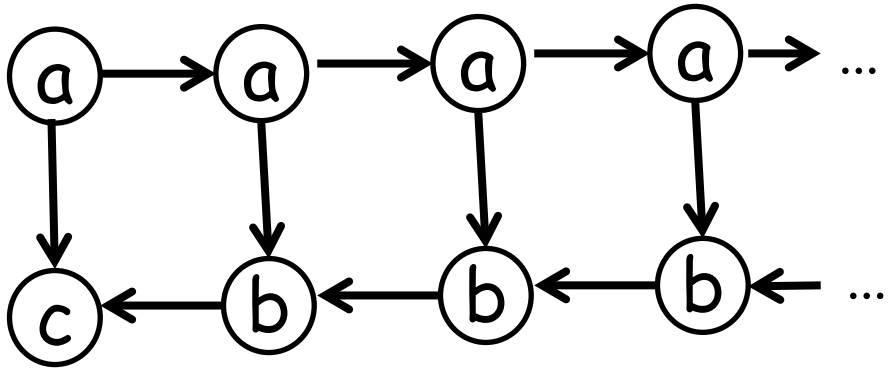
Is there a transition sequence in which "a" occurs after "b"?

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



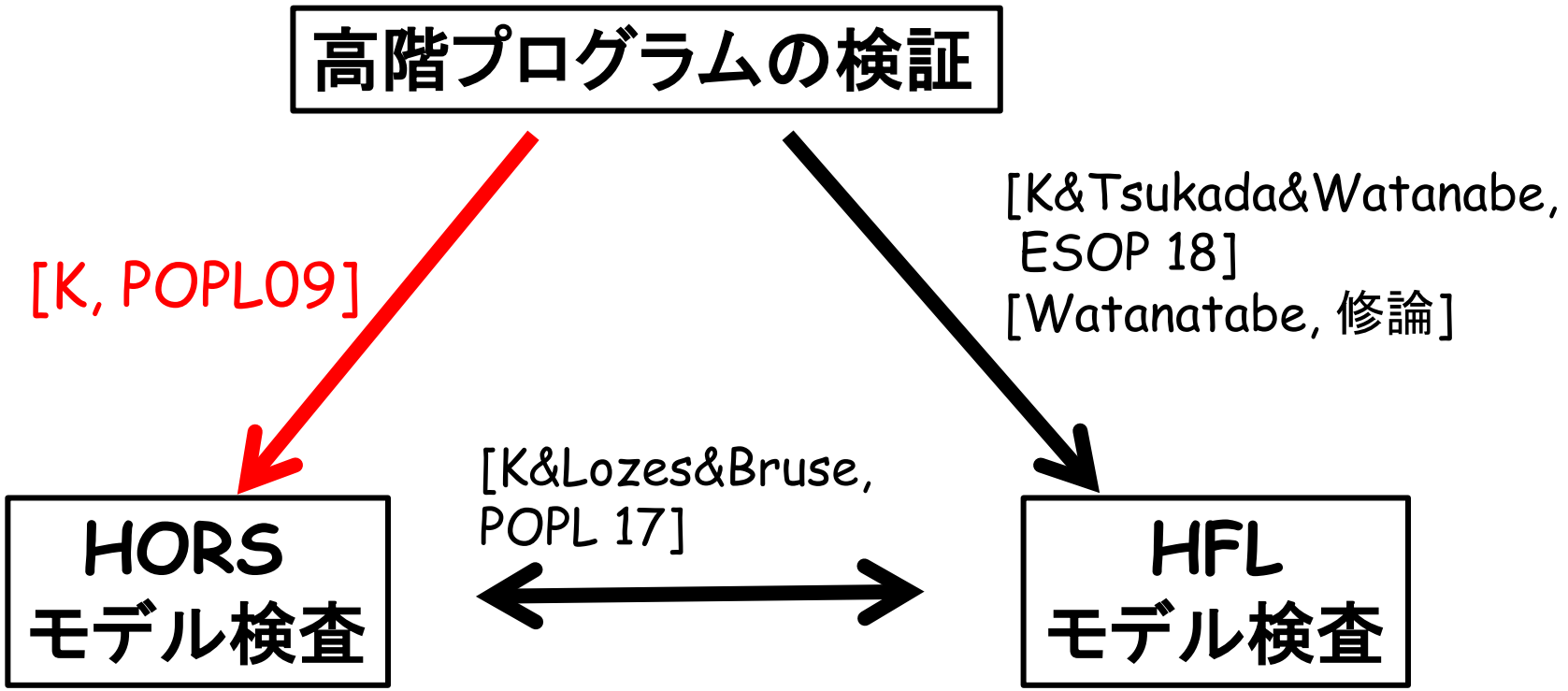


Does "a" occur below "b"?

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

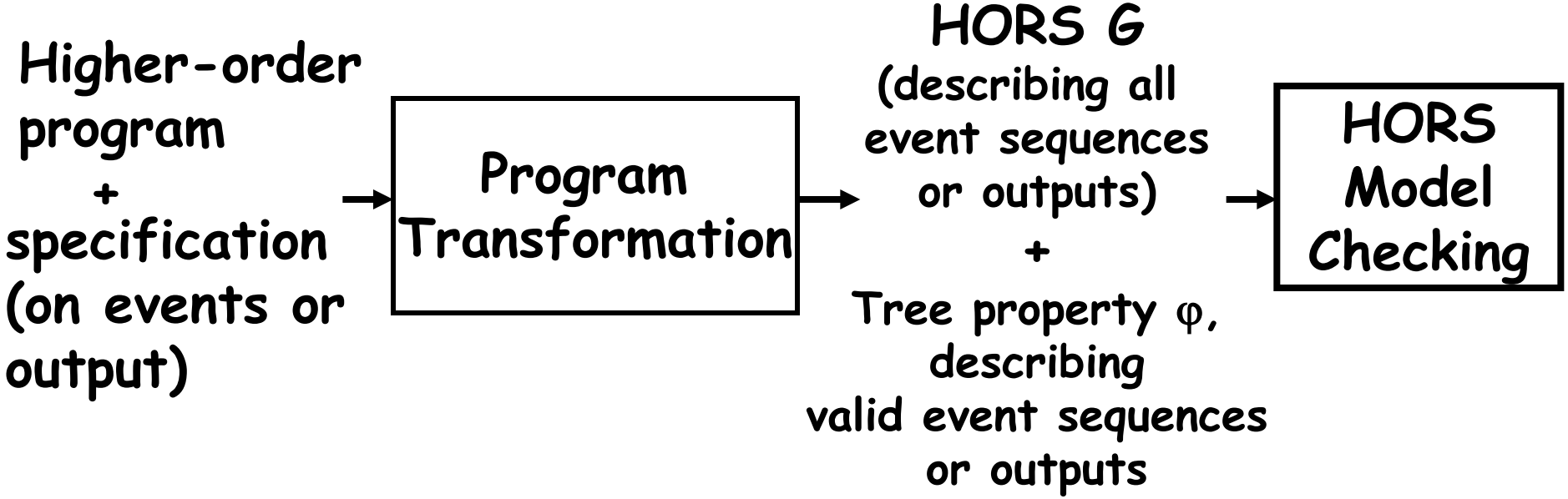
アウトライン

- HORSモデル検査とプログラム検証への応用
- HFLモデル検査とプログラム検証への応用
- HORSモデル検査とHFL検査の関係



From Program Verification to HORS Model Checking

[K. POPL 2009]

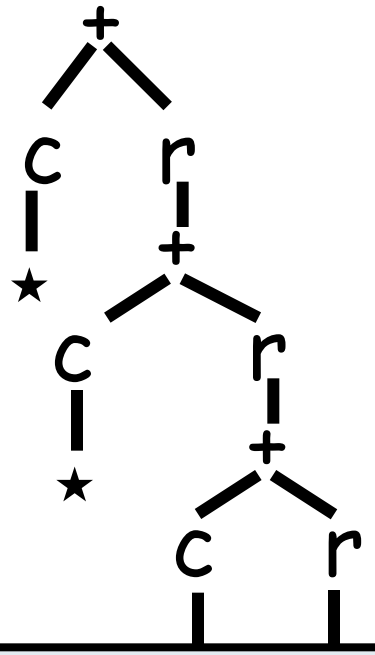


From Program Verification to Model Checking: Example

```
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

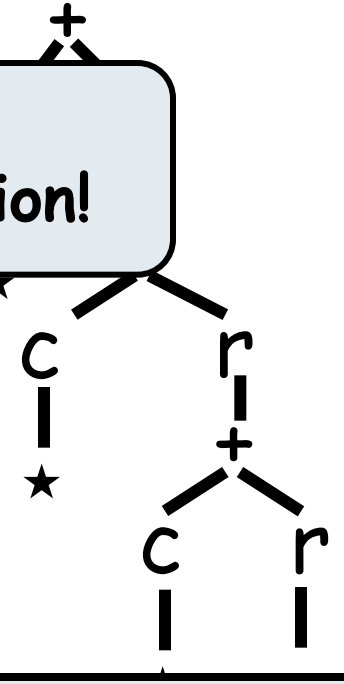
continuation parameter, expressing how "foo" is accessed after the call returns

```
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?

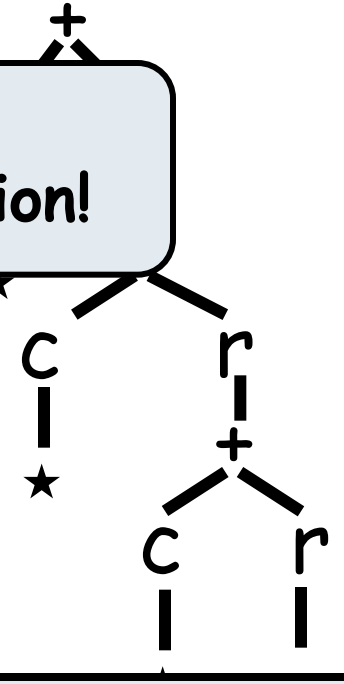
From Program Verification to Model Checking: Example

```
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?

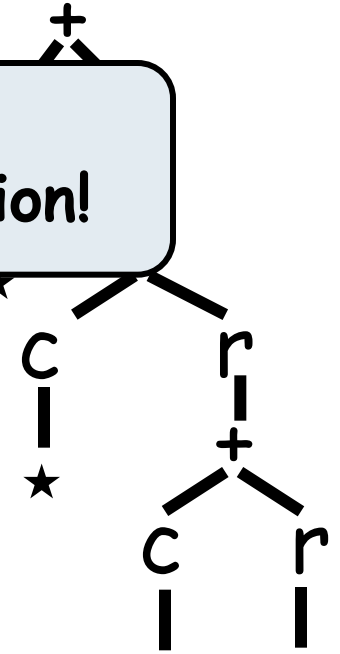
From Program Verification to Model Checking: Example

```
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?

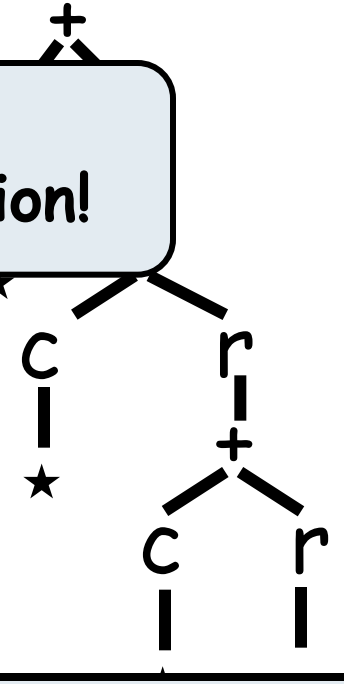
From Program Verification to Model Checking: Example

```
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?

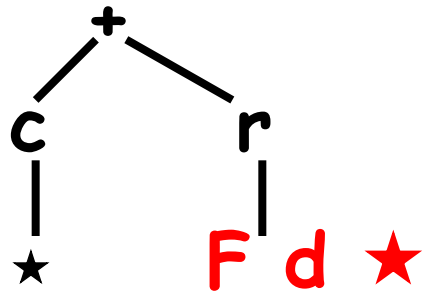
Is each path of the tree labeled by r*c?

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?

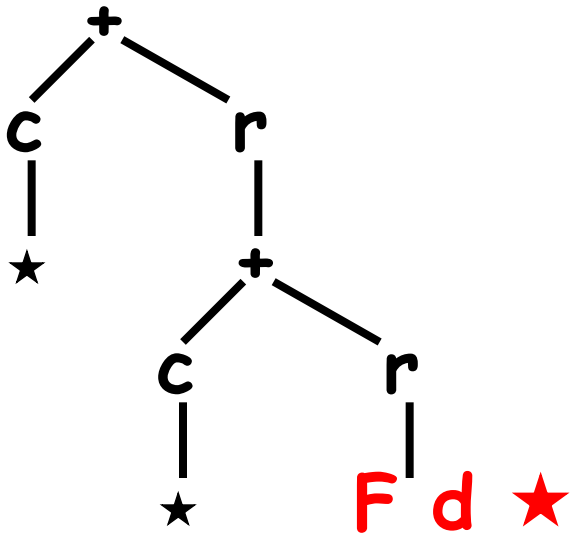
Is each path of the tree labeled by r*c?

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?

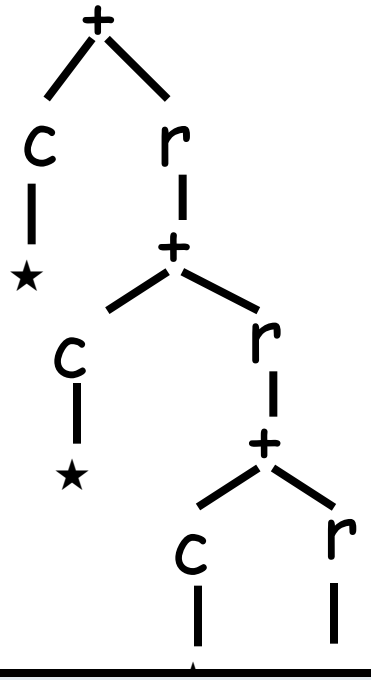
Is each path of the tree labeled by r*c?

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)
```

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

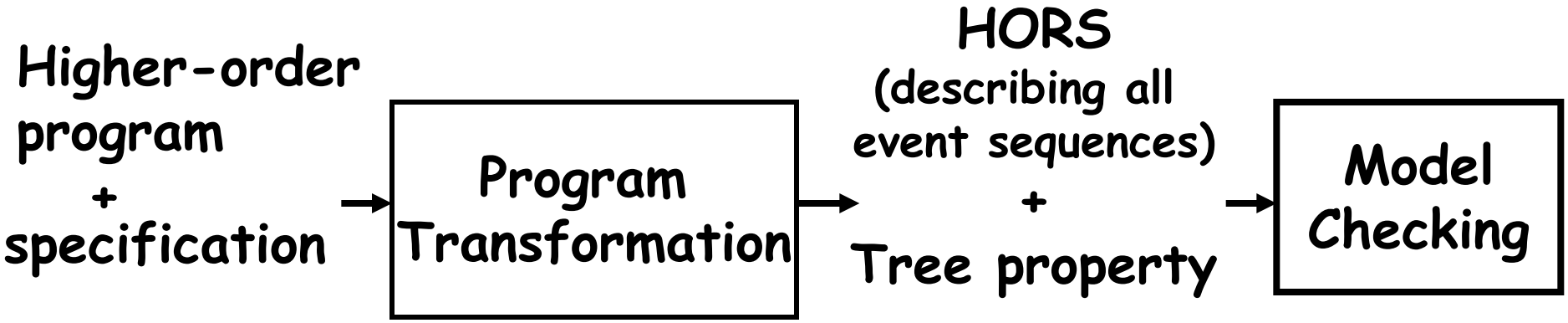
$S \rightarrow F\ d\ \bigstar$



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

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

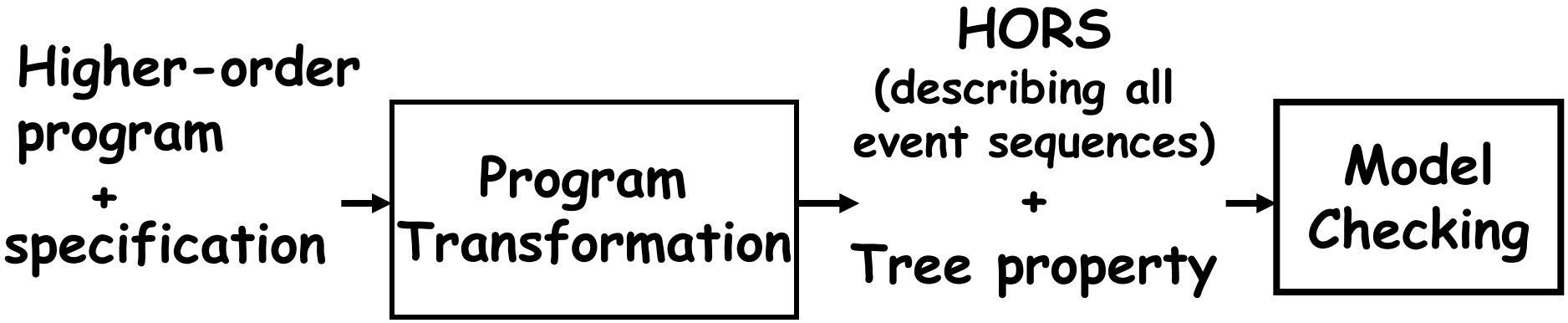
From Program Verification to HORS Model Checking



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, ...

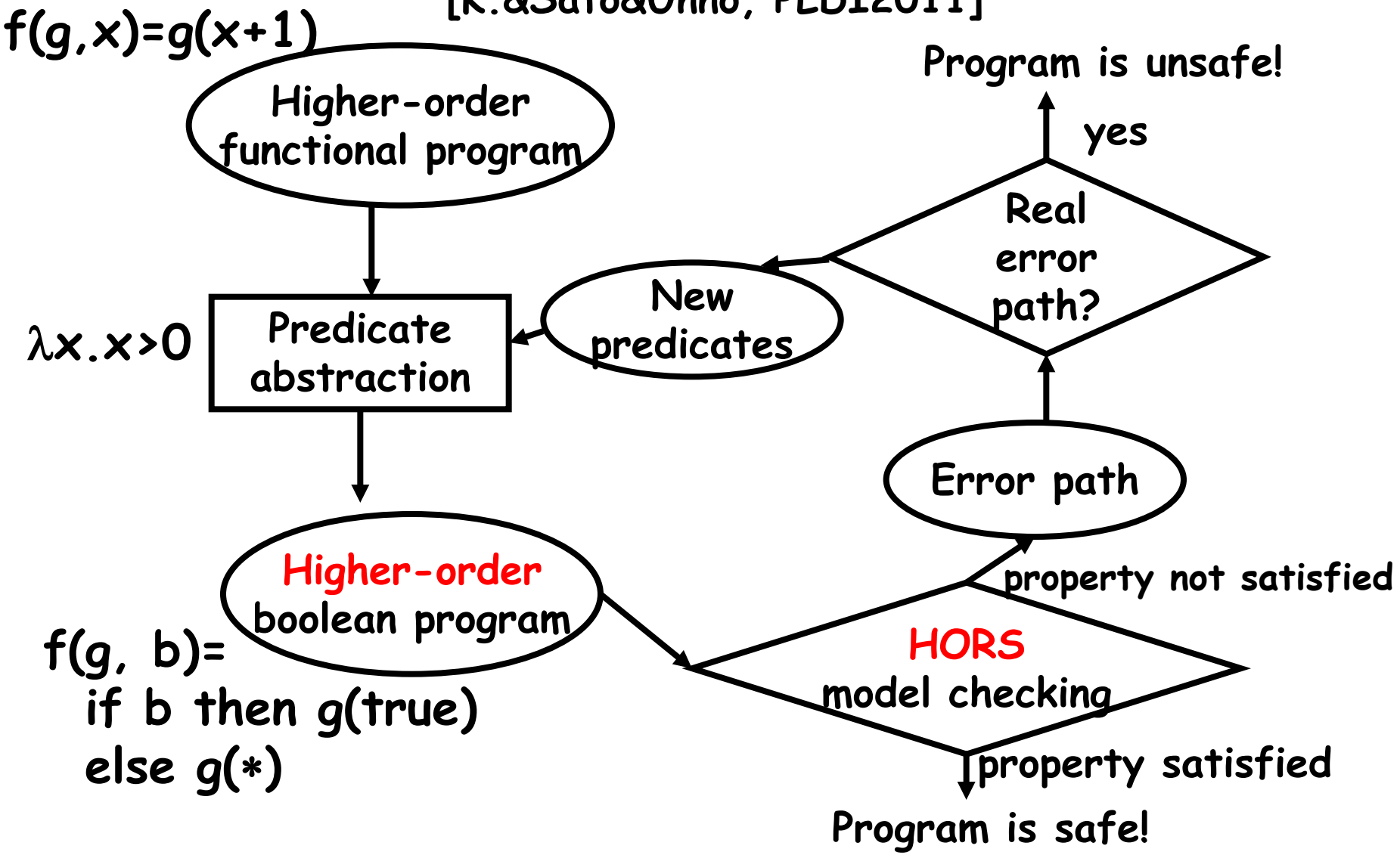
From Program Verification to HORS Model Checking



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

Predicate Abstraction and CEGAR for HORS Model Checking

[K.&Sato&Unno, PLDI2011]



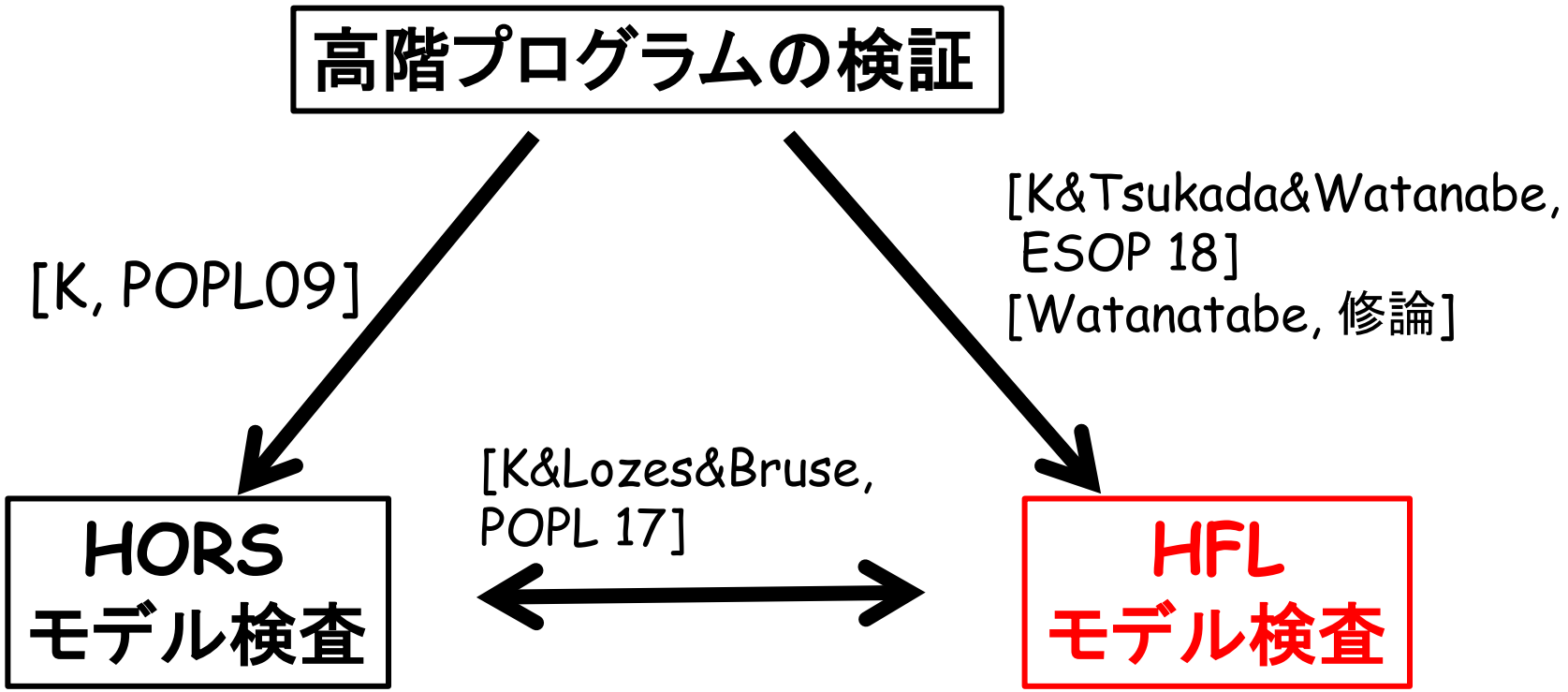
Tool demonstration: MoCHi

[K&Sato&Unno, 2011]

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

アウトライン

- HORSモデル検査とプログラム検証への応用
- HFLモデル検査とプログラム検証への応用
- HORSモデル検査とHFLモデル検査の関係



本チュートリアルの内容

- 二種類の高階モデル検査とそれらの相互の関係、プログラム検証への応用について

	Models	Logic
有限状態 モデル検査	有限状態システム	様相 μ計算
HORS モデル検査 [Knapik+ 01; Ong 06]	高階再帰スキーム (HORS)	様相 μ計算
HFL モデル検査 [Viswanathan&Viswanathan 04]	有限状態システム	高階様相 不動点論理(HFL)

Useful for describing non-regular properties

Modal μ-calculus

[Kozen 83]

- Propositional modal logic with fixpoint operators

  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 propositional variable

  $\mu X.\varphi$ least fixpoint

  $\nu X.\varphi$ greatest fixpoint

- Subsumes CTL, LTL, and CTL*
  - EF A ("eventually A may hold") = $\mu X.(A \vee \langle . \rangle X)$
- Equi-expressive as alternating parity tree automata and MSO (for describing tree properties)

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 ::= $\bullet$ | $\kappa_1 \rightarrow \kappa_2$

Selected Typing Rules for HFL

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

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

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

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

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

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

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

Semantics

$[\varphi]_I$: the set of states that satisfy φ

$L \models \varphi \Leftrightarrow s_{init} \in [\varphi]_\varnothing$ (s_{init}: initial state of L)

$[true]_I = States$ $\quad [\varphi \wedge \psi]_I = [\varphi]_I \cap [\psi]_I$

$[\varphi \vee \psi]_I = [\varphi]_I \cup [\psi]_I$

$[\,[\alpha]\,\varphi\,]_I = \{s \mid \forall t.(s \rightarrow_\alpha t \text{ implies } t \in [\varphi]_I)\}$

$[\,\langle\alpha\rangle\,\varphi\,]_I = \{s \mid \exists t.(s \rightarrow_\alpha t \text{ and } t \in [\varphi]_I)\}$

$[\mu X^\kappa.\varphi]_I = lfp(\lambda x \in [\kappa].[\varphi]_{I\{X=x\}})$

$[\nu X^\kappa.\varphi]_I = gfp(\lambda x \in [\kappa].[\varphi]_{I\{X=x\}})$

$[\lambda X^\kappa.\varphi]_I = \lambda x \in [\kappa].[\varphi]_{I\{X=x\}}$

$[\varphi\ \psi]_I = [\varphi]_I\,[\psi]_I$ $\quad [X]_I = I(X)$

$[\bullet] = 2^{States}$

$[\kappa_1 \rightarrow \kappa_2] = \{f \in [\kappa_1] \rightarrow [\kappa_2] \mid f: \text{monotonic}\}$

Example

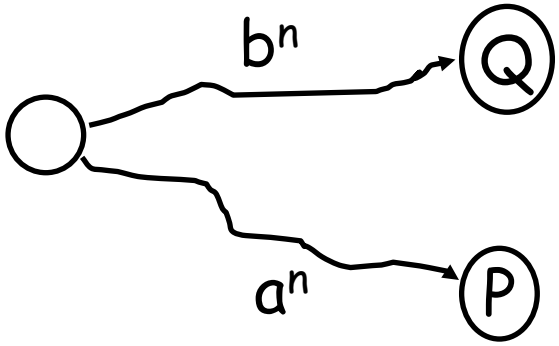
$(\mu F^{\bullet\to\bullet\to\bullet}.\lambda X.\lambda Y.\ (X \wedge Y) \vee F\ (\langle a\rangle X)\ (\langle b\rangle Y))\ P\ Q$

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

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

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

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

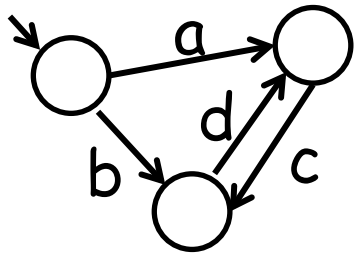


HFL Model Checking

Given
 L: (finite-state) labeled transition system
 φ: HFL formula,
does L satisfy φ?

e.g. $L \models \varphi$ for:

L:



φ: $(\mu F.\lambda X.\lambda Y.\ (X \wedge Y)$
$\vee\ F\ (\langle a \rangle X)\ (\langle b \rangle Y))$
$(\langle c \rangle true)\ \ (\langle d \rangle true)$

HES (Hierarchical Equation Systems) Representation of HFL Formulas

$X_1 =_{\alpha 1} \varphi_1; \ldots; X_n =_{\alpha n} \varphi_n$

$(\alpha_i \in \{\mu, \nu\})$

Example:

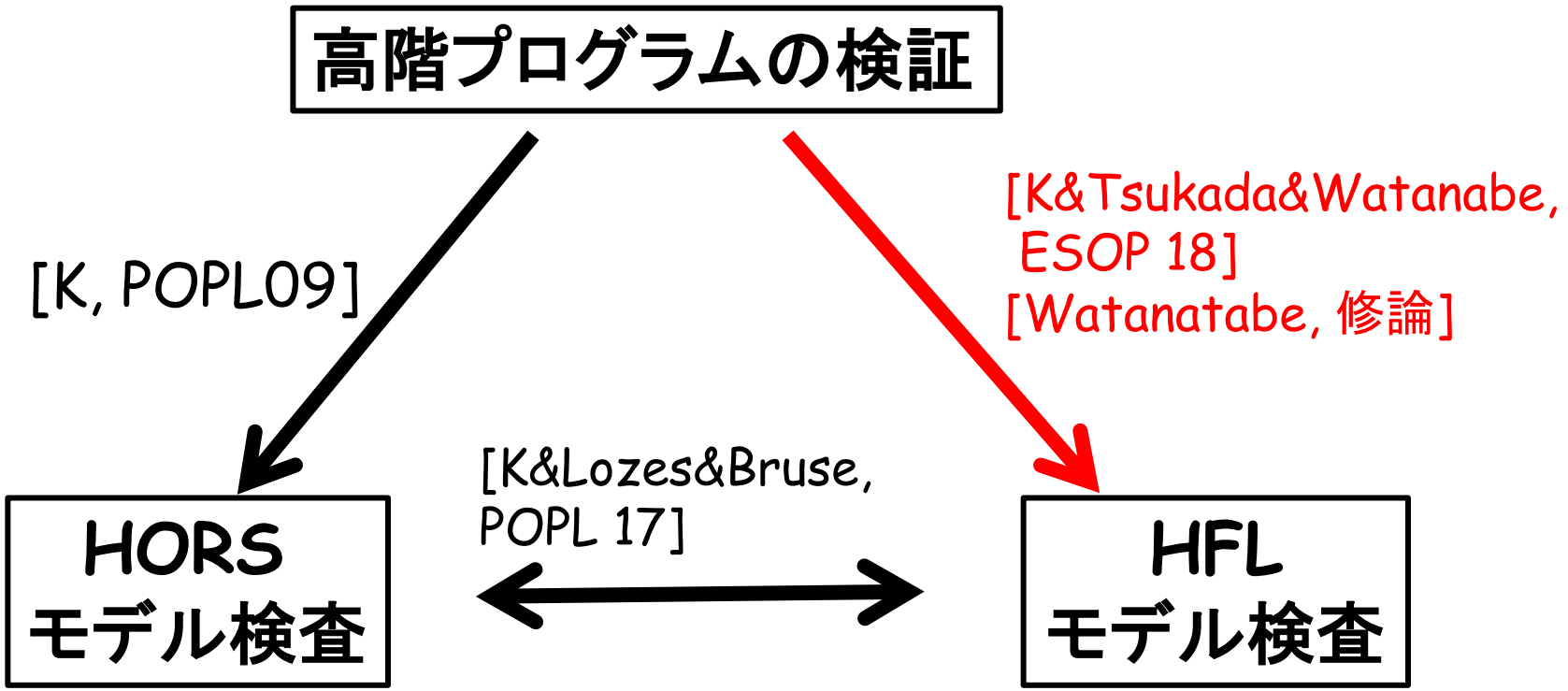
HFL: $\nu X.\mu Y.(\langle a \rangle X \vee \langle b \rangle Y)$

(there exists a path $(b^*a)^\omega$)

HES: $X =_\nu Y;\ Y =_\mu \langle a \rangle X \vee \langle b \rangle Y$

アウトライン

- HORSモデル検査とプログラム検証への応用
- HFLモデル検査とプログラム検証への応用
- HORSモデル検査とHFLモデル検査の関係



本チュートリアルの内容

- 二種類の高階モデル検査とそれらの相互の関係、プログラム検証への応用について

	Models	Logic
有限状態 モデル検査	有限状態システム	様相 μ計算
HORS モデル検査 [Knapik+ 01; Ong 06]	高階再帰スキーム (HORS)	様相 μ計算
HFL モデル検査 [Viswanathan&Viswanathan 04]	有限状態システム	高階様相 不動点論理(HFL)

Useful for describing non-regular properties

From Program Verification to HFL Model Checking: Example

```
let y = open "foo"
in
  read(y); close(y)
```

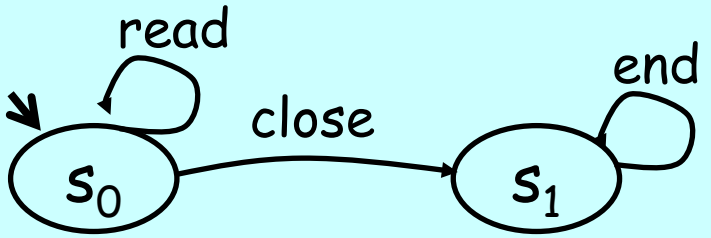
⟹ 「プログラムが正しい動作をする」ことを表す論理式

$\langle read \rangle \langle close \rangle \langle end \rangle true$

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)
```

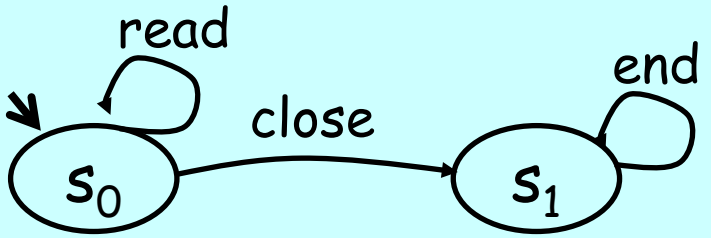
⇒ 「プログラムが正しい動作をする」ことを表す論理式

$\langle read \rangle \langle close \rangle \langle end \rangle true \wedge \langle close \rangle \langle end \rangle true$

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)
```

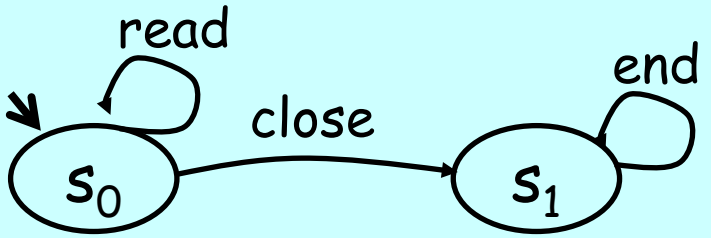
「プログラムが正しい動作をする」ことを表す論理式

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

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

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

Does LTS:



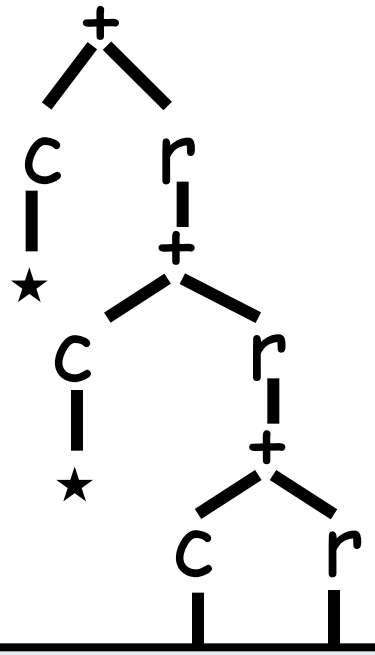
satisfy the formula S?

From Program Verification to 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 extended HFL Model Checking

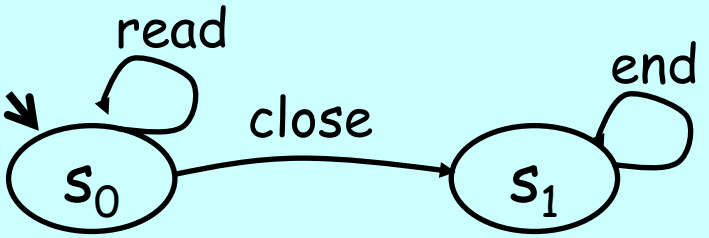
```
let f n x =
 if n≤0 then close(x)
 else
  (read(x); f (n-1) x)
in
let y = open "foo"
in
      f m (y)
```

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

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

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

Does LTS:



satisfy the formula S?

From Program Verification to extended HFL Model Checking

This approach provides a sound and complete logical characterization of:

- reachability problem
- termination problem
- linear/branching-time temporal properties

for higher-order functional programs with infinite data

[K+ ESOP 2018] [Watanabe, 修論]

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

satisfy the formula S?

From Termination Verification to extended HFL Model Checking

```
let sum n k =
 if n≤0 then k 0
 else
  sum (n-1) λr.k(r+n)
in sum m (λx.())
```

⇛

Termination:

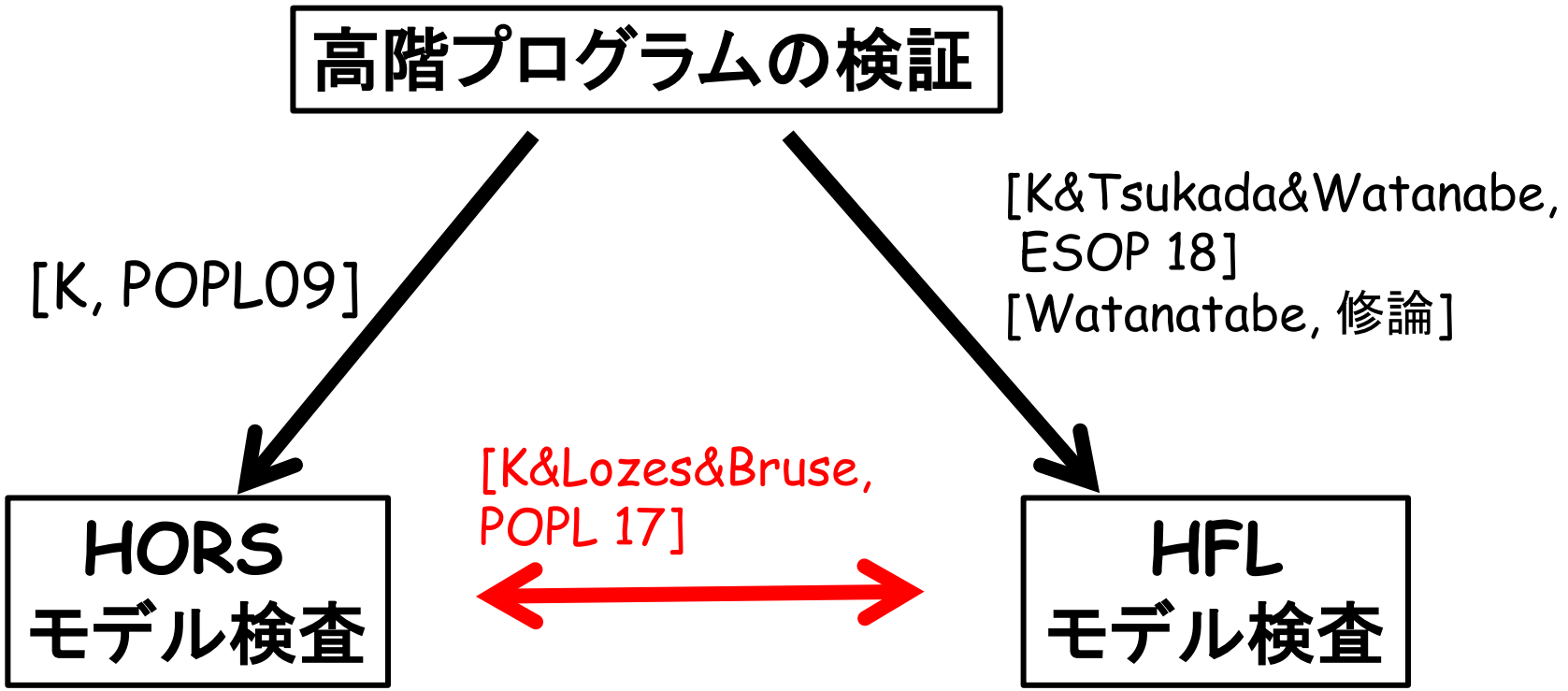
$(\mu\ sum.\lambda n.\lambda k.$
$(n \leq 0 \Rightarrow k\ 0) \wedge$
$(n > 0 \Rightarrow sum(n-1)\ \lambda r.k(r+n)))$
$m\ (\lambda x.true)$

Non-Termination:

$(\nu\ sum.\lambda n.\lambda k.$
$(n \leq 0 \wedge k\ 0) \vee$
$(n > 0 \wedge\ sum(n-1)\ \lambda r.k(r+n)))$
$m\ (\lambda x.false)$

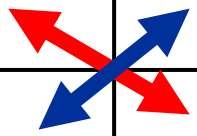
アウトライン

- HORSモデル検査とプログラム検証への応用
- HFLモデル検査とプログラム検証への応用
- HORSモデル検査とHFLモデル検査の関係
  - 純粋なモデル検査問題として
  - プログラム検証への応用として



HORS vs HFL model checking

	Model	Spec.	complexity	Applications
HORS model checking	HORS	APT	k-EXPTIME complete (for order-k HORS)	Automated verification of functional programs [K 09][K+11]...
HFL model checking	LTS	HFL	k-EXPTIME complete (for order-k HFL)	Assume-guarantee reasoning [VV 04] Process equivalence checking [Lange+ 14] Verification of functional programs [K+ 18][Watanabe 18]



APT: alternating parity tree automaton
LTS: finite-state labeled transition system

From HORS to HFL model checking

- Input:
  - HORS G
  - APT A (with largest priority p)
- Output:
  - LTS L_A
  - HFL formula $\varphi_{G,p}$

  such that $G \models A$ iff $L_A \models \varphi_{G,p}$

Intuition:

- L_A simulates the transitions of A
- $\varphi_{G,p}$ describes "L_A has transitions corresponding to an accepting run of A over Tree(G)"

From HFL to HORS model checking

- **Input:**
  - LTS L
  - HFL formula φ
- **Output:**
  - HORS $G_{\varphi,c}$
  - APT A_L

  such that $L \models \varphi$ iff $G_{\varphi,c} \models A_L$ for sufficiently large c

Intuition:

- $G_{\varphi,c}$ generates tree representation of the formula obtained from φ by unfolding fixedpoint operators sufficiently many times
- A_L accepts trees representing valid formulas

From HORS to HFL model checking

- Input:
  - HORS G
  - Parity tree automaton A (with largest priority p)
- Output:
  - LTS L_A
  - HFL formula $\varphi_{G,p}$

  such that $Tree(G) \models A$ iff $L_A \models \varphi_{G,p}$

Intuition:

- L_A simulates the transitions of A
- $\varphi_{G,p}$ describes "L_A has transitions corresponding to an accepting run of A over $Tree(G)$"

Construction of L_A

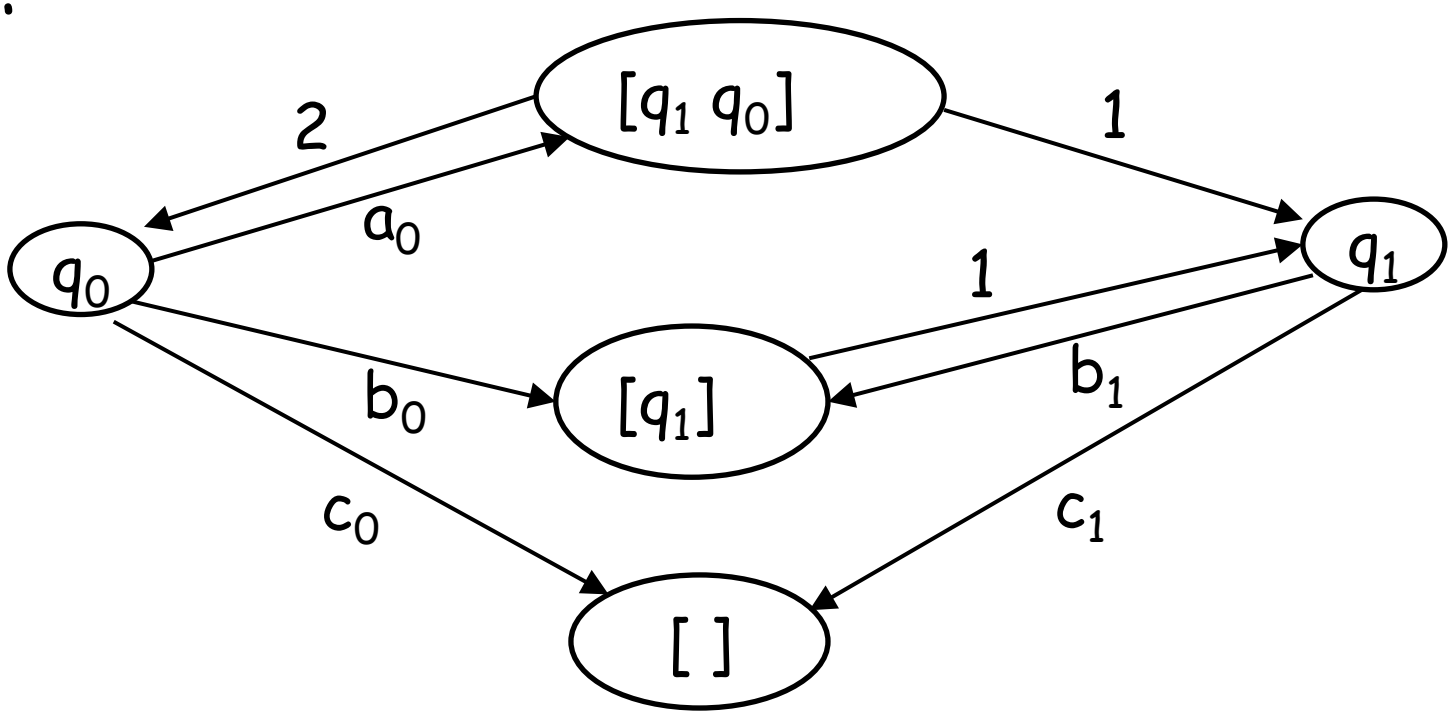
(non-deterministic case; see the paper for the case of APT)

A:

$q_0 \to_a q_1\, q_0 \quad q_0 \to_b q_1 \quad q_1 \to_b q_1 \quad q_0 \to_c \quad q_1 \to_c$

$\Omega(q_0)=0 \quad \Omega(q_1)=1$

L_A:



Construction of L_A

(non-deterministic case; see the paper for the case of APT)

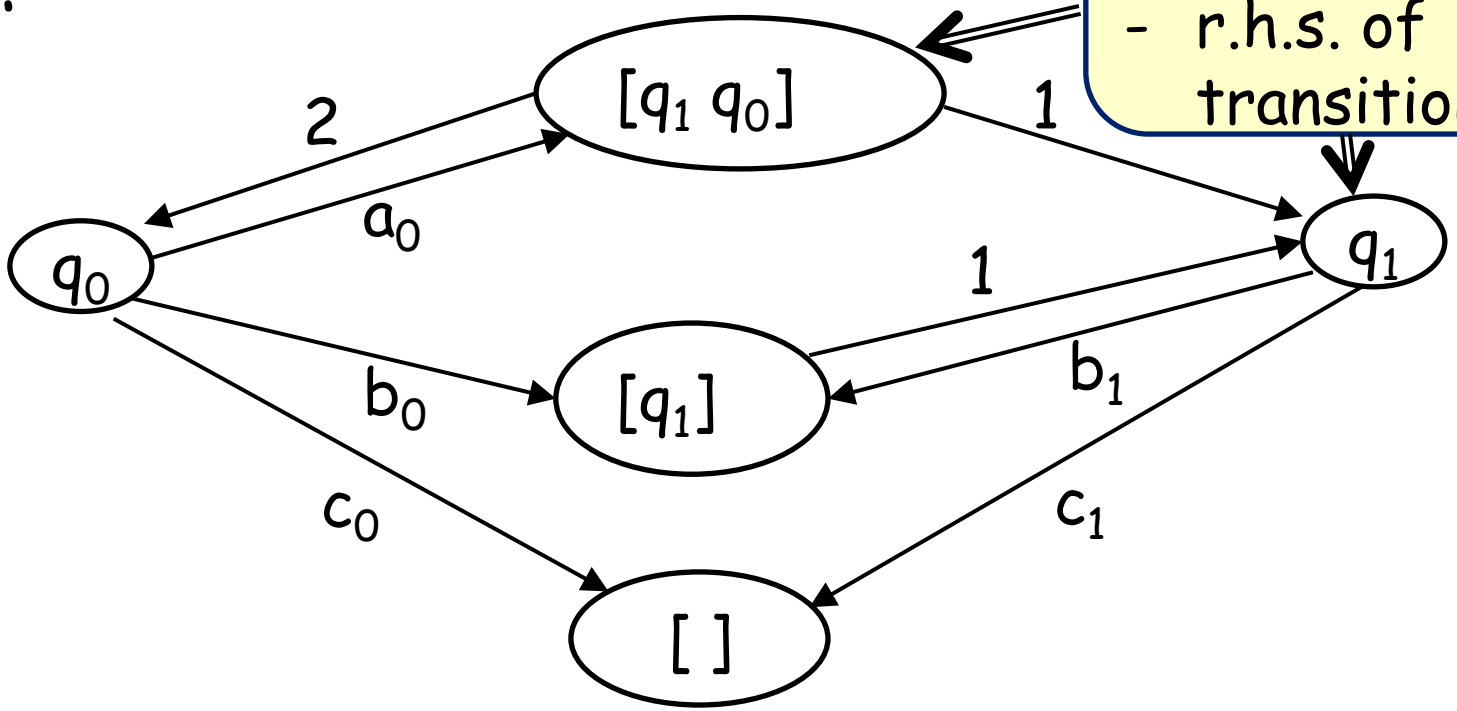
A:

$q_0 \to_a q_1\, q_0 \quad q_0 \to_b q_1 \quad q_1 \to_b q_1 \quad q_0 \to_c \quad q_1 \to_c$

$\Omega(q_0)=0 \quad \Omega(q_1)=1$

The states of L_A consist of:
- states of A and
- r.h.s. of transition rules

L_A:



Construction of L_A

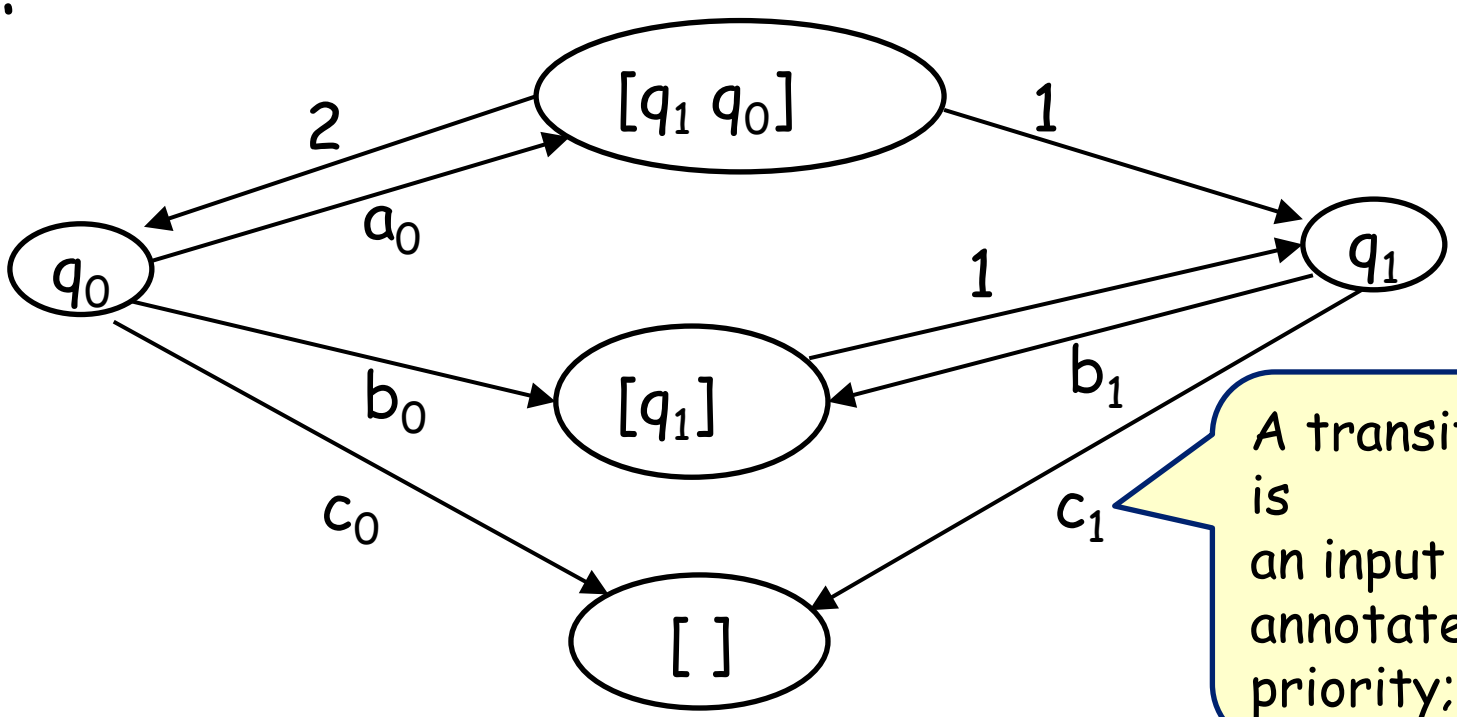
(non-deterministic case; see the paper for the case of APT)

A:

$q_0 \to_a q_1\, q_0 \quad q_0 \to_b q_1 \quad q_1 \to_b q_1 \quad q_0 \to_c \quad q_1 \to_c$

$\Omega(q_0)=0 \quad \Omega(q_1)=1$

L_A:



A transition label is
an input symbol annotated with a priority; or ...

Construction of L_A

(non-deterministic case; see the paper for the case of APT)

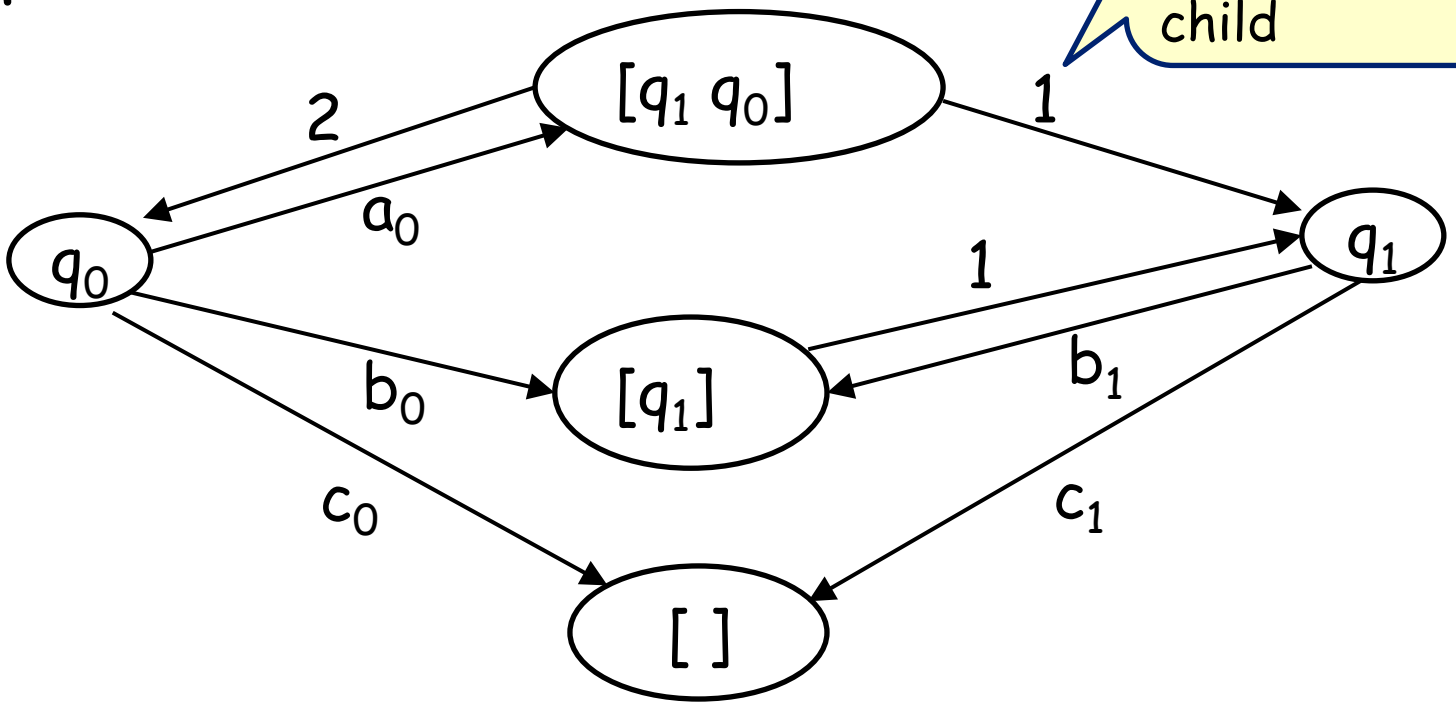
A:

$q_0 \rightarrow_a q_1\, q_0 \quad q_0 \rightarrow_b q_1 \quad q_1 \rightarrow_b q_1 \quad q_0 \rightarrow_c \quad q_1 \rightarrow_c$

$\Omega(q_0)=0 \quad \Omega(q_1)=1$

A transition label is ...; or a number to identify the visited child

L_A:



From HORS to HFL model checking

- Input:
  - HORS G
  - Parity tree automaton A (with largest priority p)
- Output:
  - LTS L_A
  - HFL formula $\varphi_{G,p}$

  such that $Tree(G) \models A$ iff $L_A \models \varphi_{G,p}$

Intuition:

- L_A simulates the transitions of A
- $\varphi_{G,p}$ describes "L_A has transitions corresponding to an accepting run of A over $Tree(G)$"

From trees to HFL formulas

(状態の優先度がすべて0の場合の特殊ケース)

φ_T: "the current state has transitions corresponding to an accepting run for T"

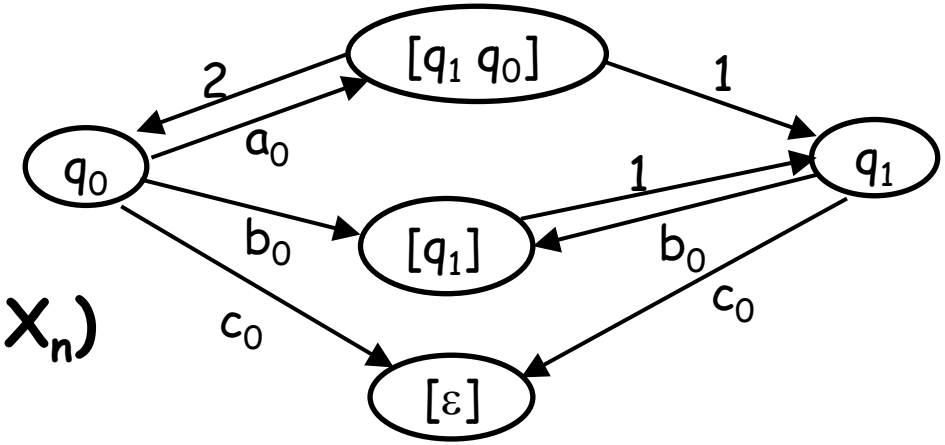
$\varphi_{a\ c\ (b\ c)}$ =

$\langle a_0 \rangle$ "can visit 1st and 2nd children with states satisfying φ_c and $\varphi_{b\ c}$ respectively"

$= \langle a_0 \rangle(\langle 1 \rangle \varphi_c \wedge \langle 2 \rangle \varphi_{b\ c})$

$= \langle a_0 \rangle(H_2\ \varphi_c\ \varphi_{b\ c})$

$(H_n\ X_1\ \ldots\ X_n \stackrel{def}{=} \langle 1 \rangle X_1 \wedge \ldots \langle n \rangle X_n)$



From trees to HFL formulas

φ_T: "the current state has transitions corresponding to an accepting run for T"

$\varphi_{a\ c\ (b\ c)}$ =

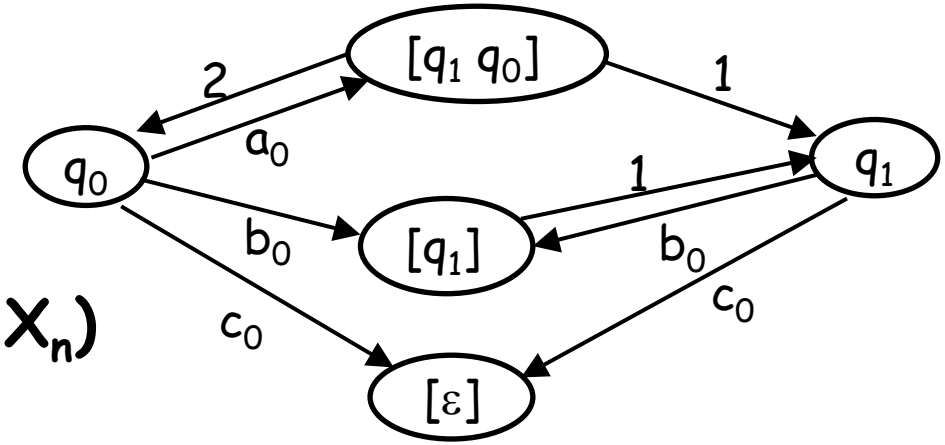
$\langle a_0 \rangle$ "can visit 1st and 2nd children with states satisfying φ_c and $\varphi_{b\ c}$ respectively"

$= \langle a_0 \rangle(\langle 1 \rangle \varphi_c \wedge \langle 2 \rangle \varphi_{b\ c})$

$= \langle a_0 \rangle(H_2\ \varphi_c\ \varphi_{b\ c})$

$= \langle a_0 \rangle(H_2\ (\langle c_0 \rangle H_0)\ (\langle b_0 \rangle H_1\ (\langle c_0 \rangle H_0)))$

$(H_n\ X_1\ \ldots\ X_n \stackrel{def}{=} \langle 1 \rangle X_1 \wedge \ldots \langle n \rangle X_n)$



From HORS to HFL

HORS G

$S \rightarrow F\ c$

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

A:

$q_0 \rightarrow_a q_1\ q_0$

$q_0 \rightarrow_b q_1$ $\quad q_1 \rightarrow_b q_1$

$q_0 \rightarrow_c$ $\quad q_1 \rightarrow_c$

$\varphi_{G,0}$

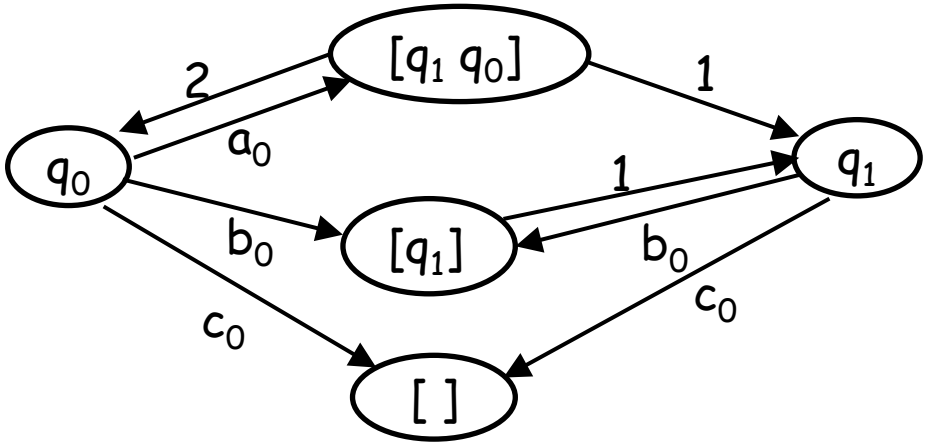
$S =_\nu F\ (\langle c_0 \rangle H_0)$

$F\ x =_\nu$

$\langle a_0 \rangle (H_2\ x\ (F(\langle b_0 \rangle (H_1\ x)))$

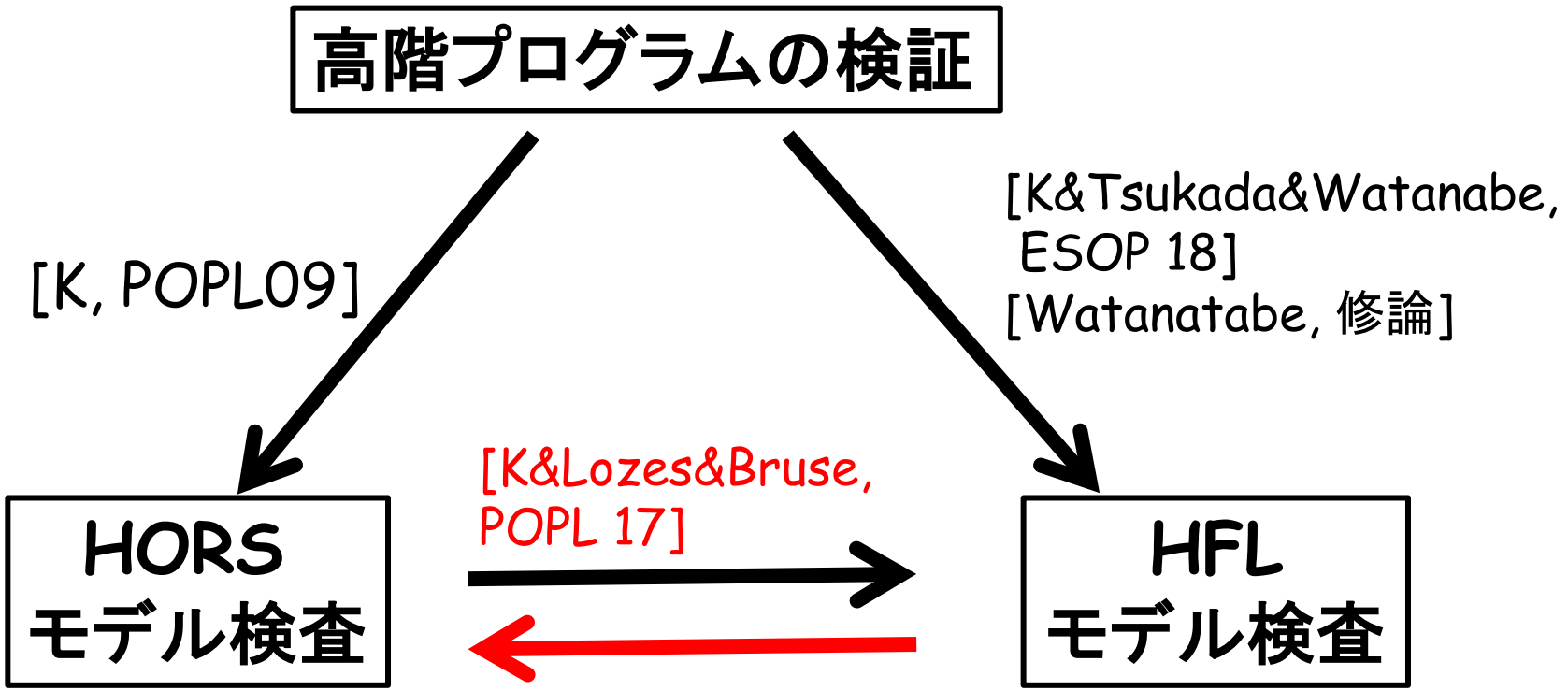
$(H_n\ X_1\ \ldots\ X_n = \langle 1 \rangle X_1 \wedge \ldots \langle n \rangle X_n)$

L_A



アウトライン

- HORSモデル検査とプログラム検証への応用
- HFLモデル検査とプログラム検証への応用
- HORSモデル検査とHFLモデル検査の関係
  - 純粋なモデル検査問題として
  - プログラム検証への応用として



From HFL to HORS model checking

- **Input:**
  - LTS L
  - HFL formula φ
- **Output:**
  - HORS $G_{\varphi,c}$
  - APT A_L

  such that $L \models \varphi$ iff $G_{\varphi,c} \models A_L$ for sufficiently large c

Intuition:

- $G_{\varphi,c}$ generates tree representation of the formula equivalent to φ, obtained by unfolding fixpoint formulas sufficiently many times
- A_L accepts trees representing valid formulas

HFL-to-HORS Translation: Overview

$F\ X =_\nu \varphi$

Remove fixpoint operators by finite unfoldings (cf. Kleene fixpoint theorem)

$F^{(c)}\ X=[F^{(c-1)}/F]\varphi\ ;\ldots;F^{(1)}\ X=[F^{(0)}/F]\ \varphi;F^{(0)}\ X=\mathrm{true}$

Convert it to HORS, which generates the tree representation of the formula

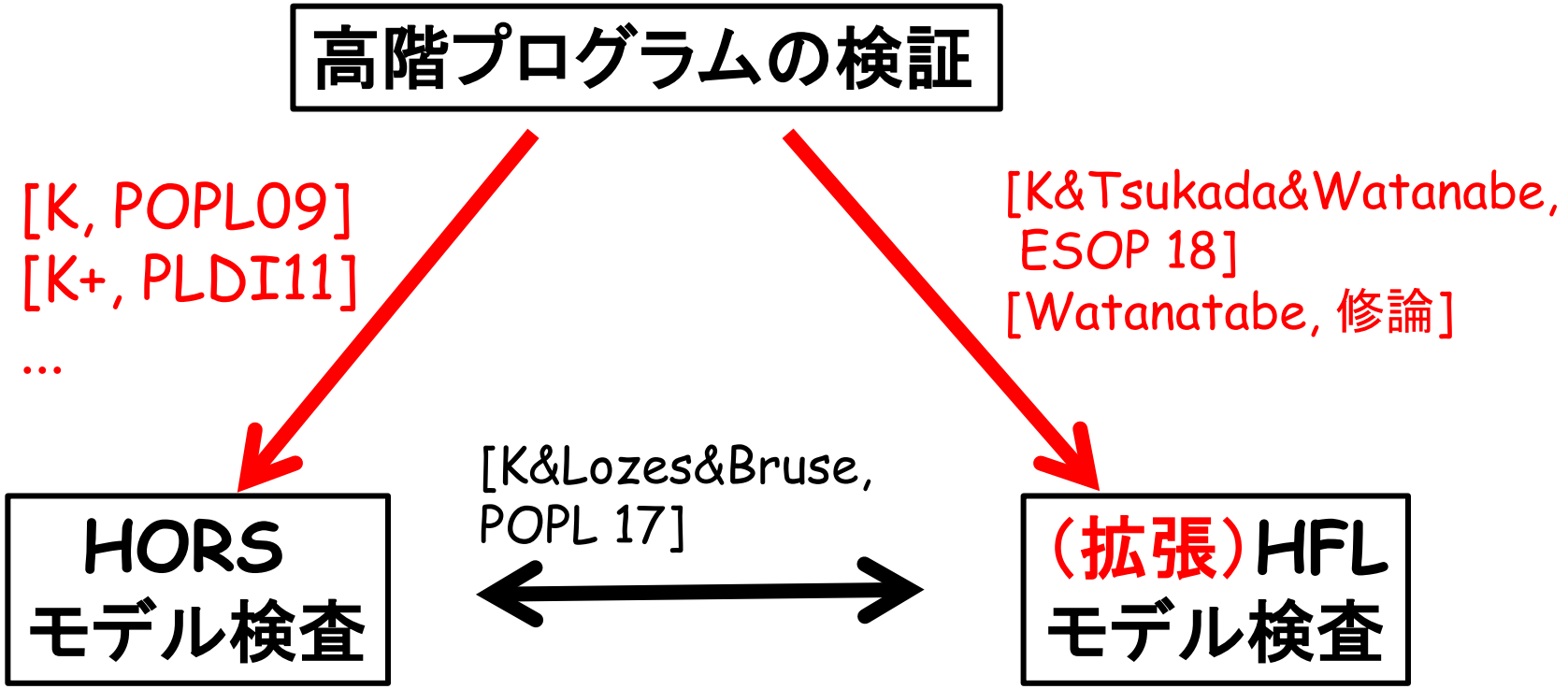
$F^{(c)}\ X\rightarrow[F^{(c-1)}/F]\varphi';\ldots;\ F^{(1)}\ X\rightarrow[F^{(0)}/F]\ \varphi';\ F^{(0)}\ X\rightarrow\mathrm{true}$

Parameterize F by a number, and implement numbers (up to $k\,2^{2^{\cdot^{\cdot^{2^n}}}}$) as functions (cf. [Jones01])

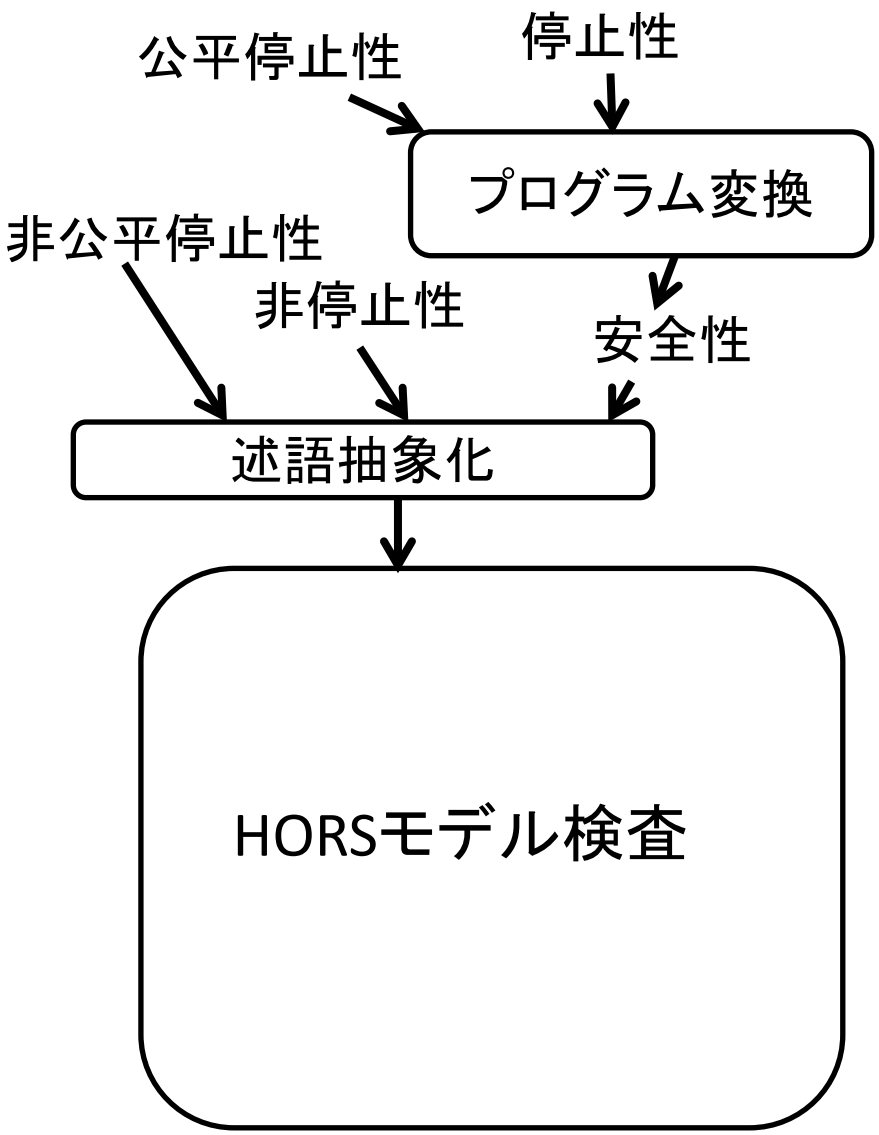
$F\ m\ X\rightarrow$ if (Zero? m) true ($[F\ (m-1)\ /F]\varphi'$)

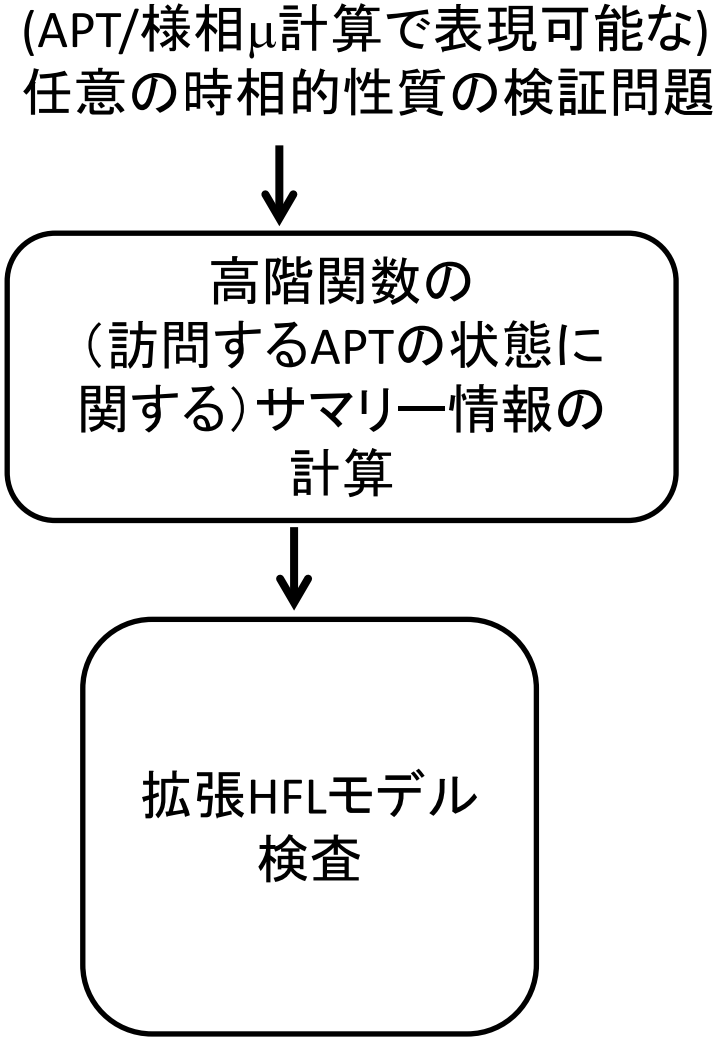
アウトライン

- HORSモデル検査とプログラム検証への応用
- HFLモデル検査とプログラム検証への応用
- HORSモデル検査とHFL検査の関係
  - 純粋なモデル検査問題として
  - プログラム検証への応用として

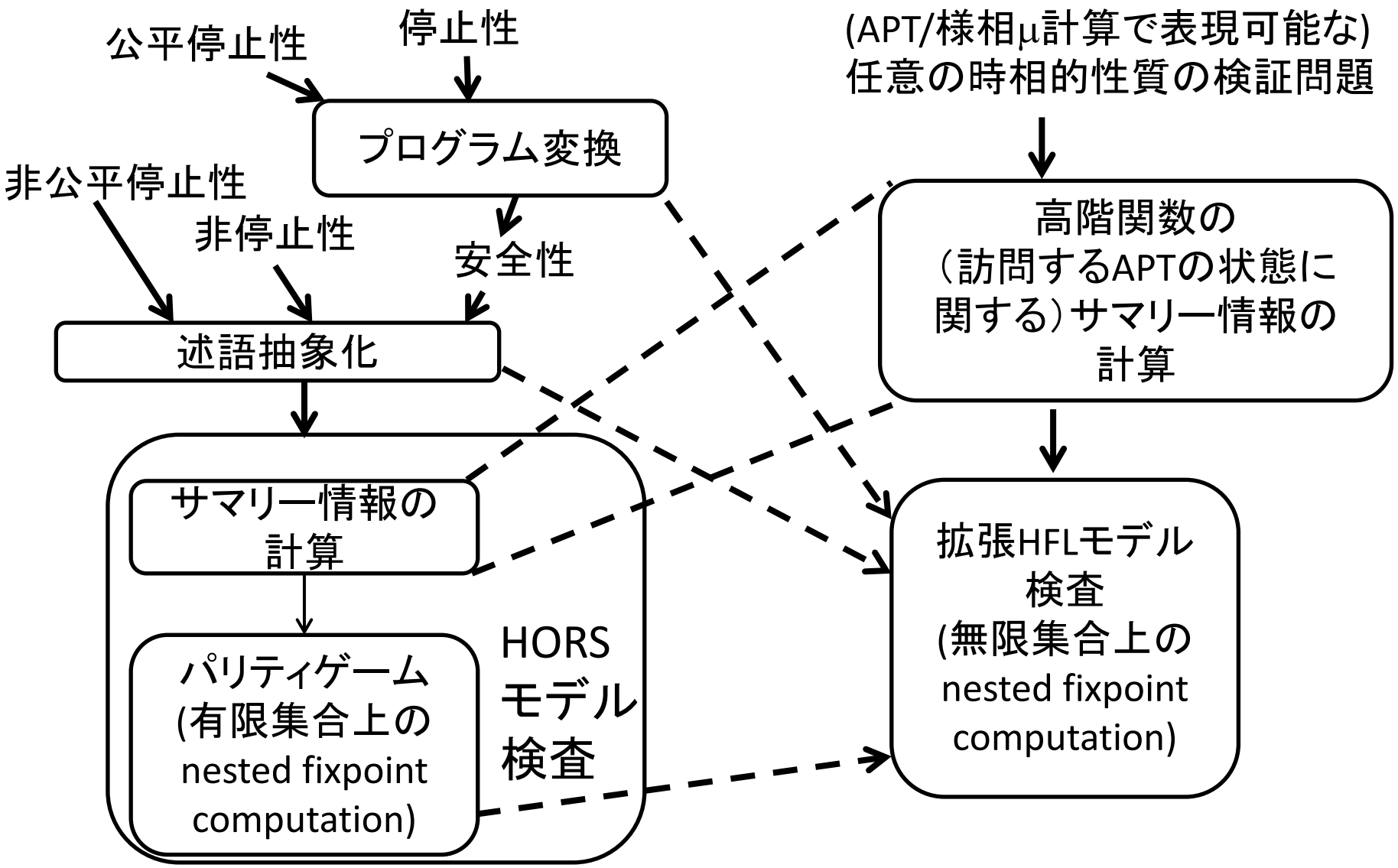


HORS-based vs (extended) HFL-based





HORS-based vs (extended) HFL-based



まとめ

- 2種類の高階モデル検査が存在

	Models	Logic
HORSモデル検査	高階再帰スキーム (HORS)	様相 μ計算
HFLモデル検査	有限状態システム	高階様相不動点論理 (HFL)

- HORS/HFLモデル検査問題は相互変換が可能
- 両者とも関数型プログラムの検証に有効。ただし…
  - HFLモデル検査の応用には発想の転換が必要(モデル⇔仕様)
  - 無限データを持つ関数型プログラムの検証には(拡張)HFLモデル検査を用いる方が有望(?)

参考文献

- HORSモデル検査
  - Ong, On Model-Checking Trees Generated by Higher-Order Recursion Schemes. LICS 2006: 81-90 (決定可能性)
  - K.&Ong, A type system equivalent to modal mu-calculus model-checking of recursion schemes, LICS09 (決定可能性の別証)
  - Kobayashi, Model Checking Higher-Order Programs, JACM, 2013 (プログラム検証への応用、モデル検査アルゴリズム)
  - Broadbent&K, Saturation-Based Model Checking of Higher-Order Recursion Schemes. CSL 2013
    (モデル検査アルゴリズム、現在最速のモデル検査器の基)
  - Watanabe+, Automatically disproving fair termination of higher-order functional programs. ICFP 2016
    (プログラム検証への応用. ここから遡ってPLID 2011, CAV 2015など)

参考文献

- HFLモデル検査
  - Mahesh Viswanathan, Ramesh Viswanathan, A Higher Order Modal Fixed Point Logic. CONCUR 2004 (HFLの原典)
  - Roland Axelsson, Martin Lange, Rafal Somla. The Complexity of Model Checking Higher-Order Fixpoint Logic. Logical Methods in Computer Science 3(2) (2007) (計算量)
  - Martin Lange, Étienne Lozes, Manuel Vargas Guzmán:
  - Model-checking process equivalences. Theor. Comput. Sci. 560: 326-347 (2014) (プロセス等価性への応用)
  - Naoki Kobayashi, Étienne Lozes, Florian Bruse, On the relationship between higher-order recursion schemes and higher-order fixpoint logic. POPL 2017 (HORS/HFLの関係)
  - Kobayashi, Tsukada, and Watanabe, Higher-Order Program Verification via HFL Model Checking, ESOP 2018 (プログラム検証への応用)
  - Watanabe, Reduction from Temporal Property Verification of Higher-Order Programs to Validity Checking of HFL Formulas, 修士論文, 東京大学, 2018 (プログラム検証への応用. ESOP 2018の手法を分岐を含む任意のω-regular propertyの検証に拡張)