Predicate Abstraction and CEGAR for Higher-Order Model Checking

Naoki Kobayashi, Ryosuke Sato, and Hiroshi Unno

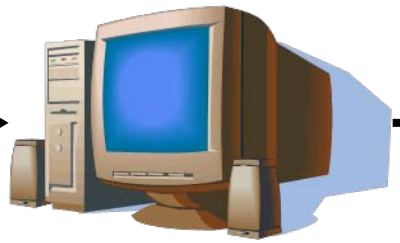
Tohoku University
(Presented at PLDI2011)

Our Goal

- Software model checker for functional programs (cf. SLAM, BLAST)

Functional program

```
let rec mc x =
  if x > 100 then
    x - 10
  else
    mc (mc (x + 11))
in
let n = randi() in
if n ≤ 101 then
  assert (mc n = 91)
```



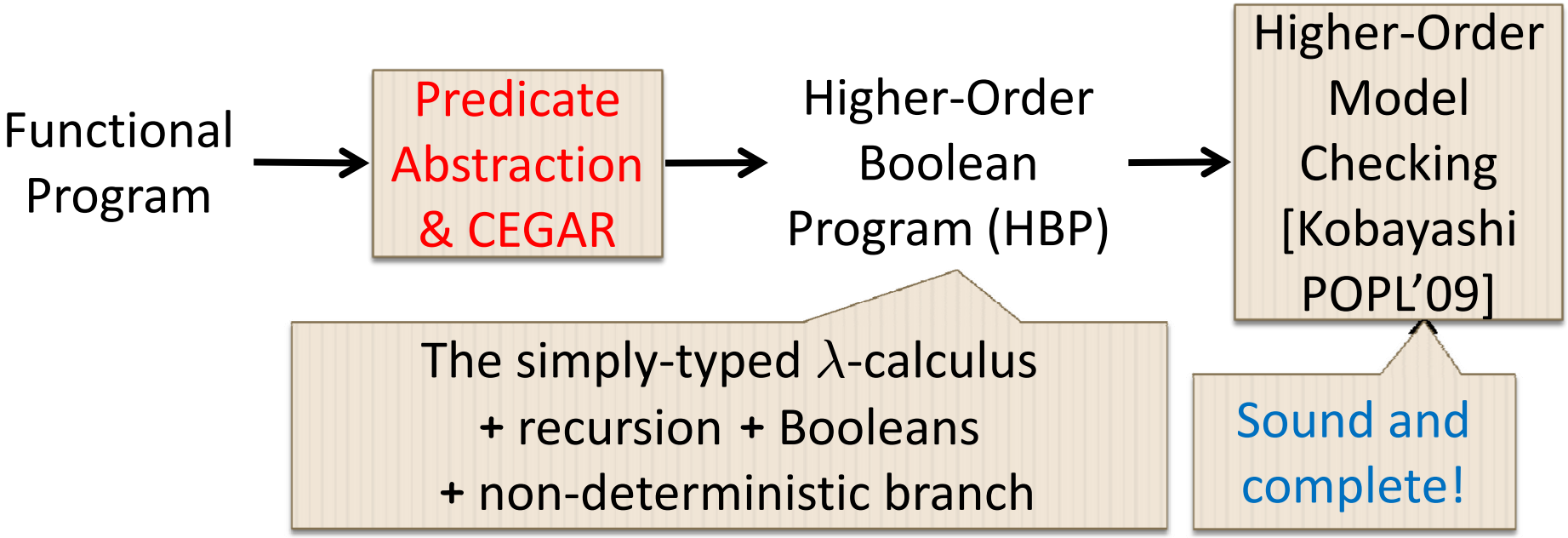
Software model checker

Result

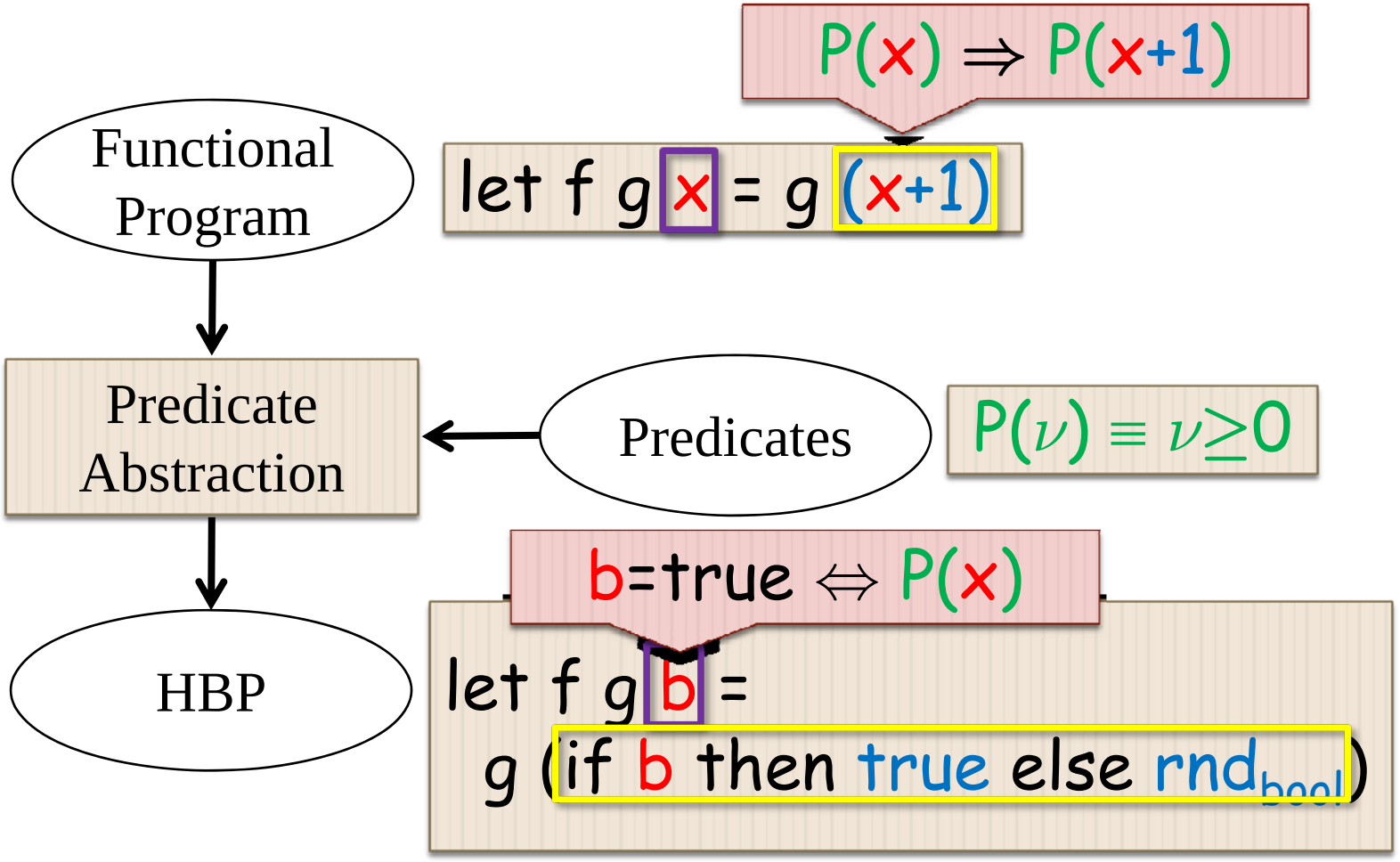
Certificate
or
Counterexample

Our Approach

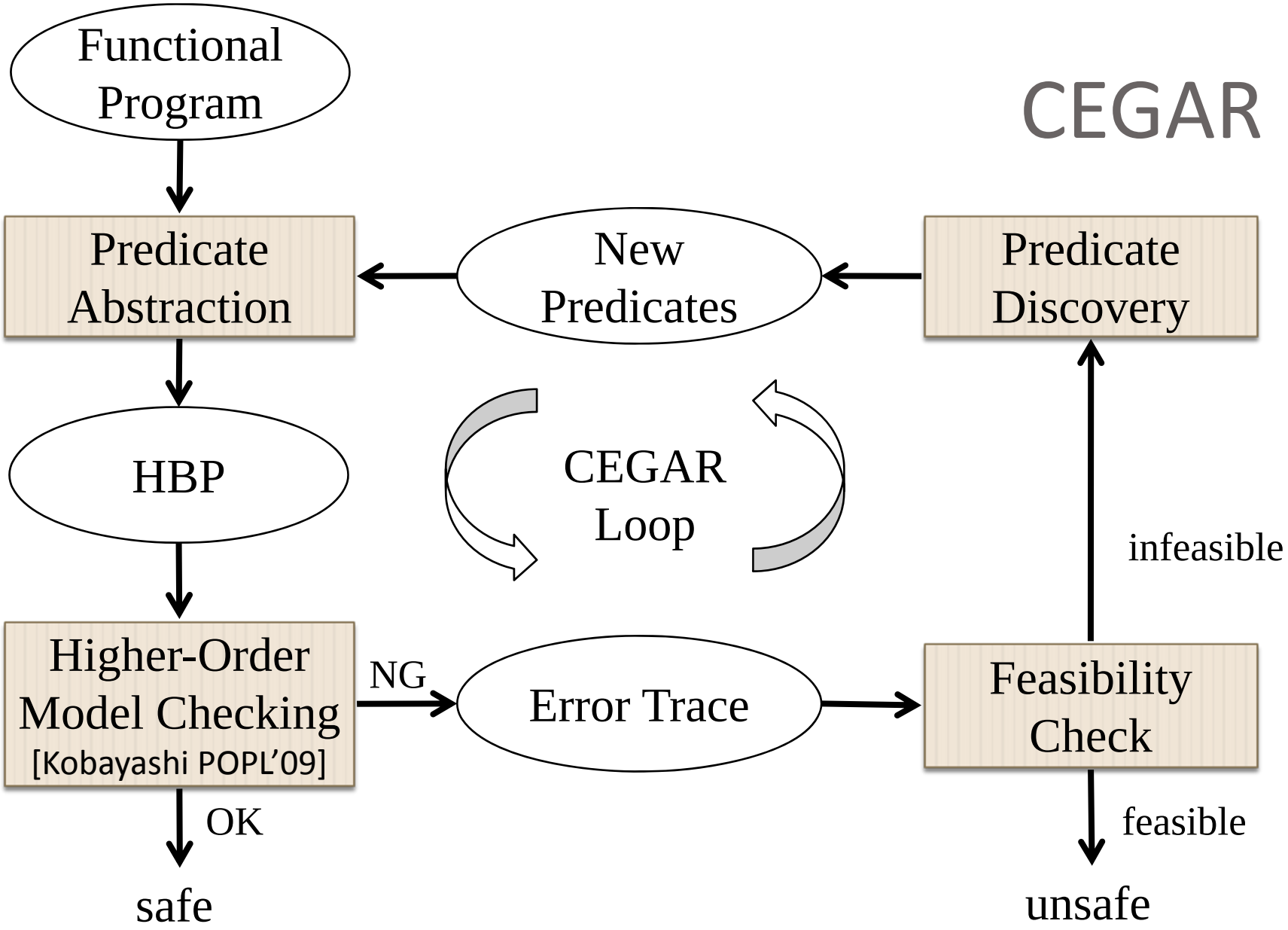
- Predicate abstraction [Graf & Saidi CAV'97] & CEGAR [Clarke et al. CAV'00] for higher-order model checking



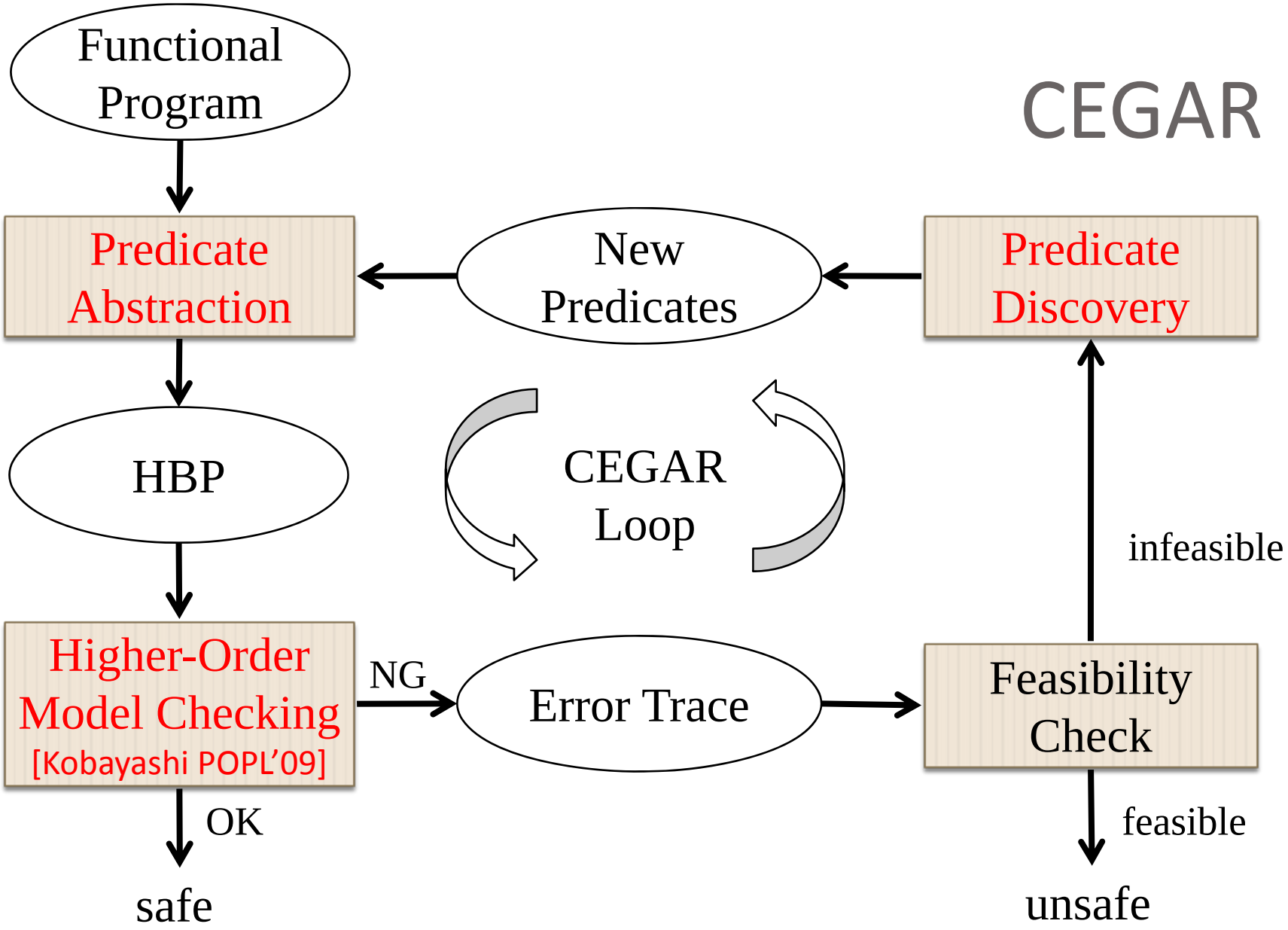
Predicate Abstraction



CEGAR


Functional Program
Predicate Abstraction
New Predicates
Predicate Discovery
HBP
CEGAR Loop
infeasible
Higher-Order Model Checking
[Kobayashi POPL'09]
NG
Error Trace
Feasibility Check
OK
feasible
safe
unsafe

CEGAR

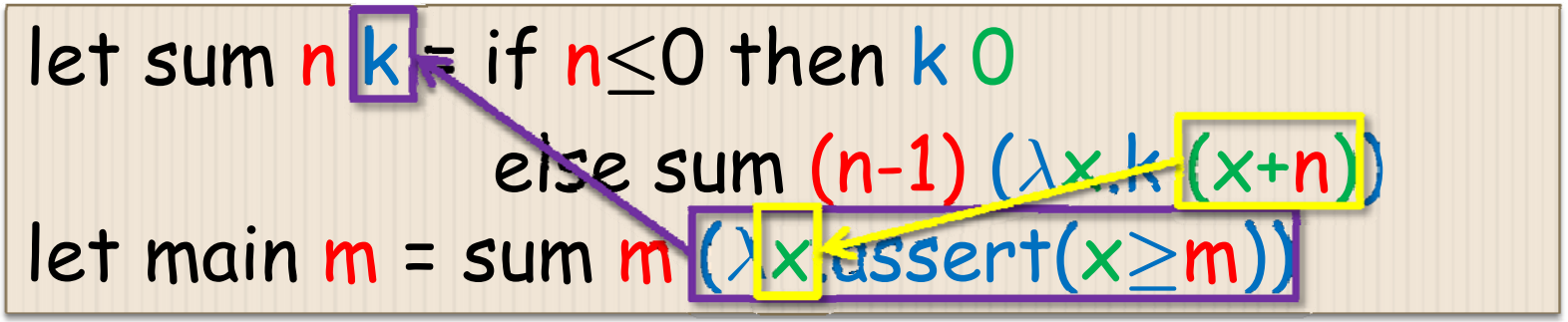

Functional Program
Predicate Abstraction
New Predicates
Predicate Discovery
HBP
CEGAR Loop
infeasible
Higher-Order Model Checking
[Kobayashi POPL'09]
NG
Error Trace
Feasibility Check
OK
feasible
safe
unsafe

Challenges in Higher-Order Setting

- Predicate Abstraction
  - How to ensure consistency of abstraction across function boundaries?
- Predicate Discovery
  - How to find new predicates that can eliminate an infeasible error trace from the abstraction?

Challenges in Higher-Order Setting

- Predicate Abstraction
  - How to ensure consistency of abstraction across function boundaries?



Our Solution

- Abstraction types to express abstraction interfaces of functions

```
let sum n k = if n≤0 then k 0
                     else sum (n-1) (λx.k (x+n))
let main m = sum m (λx.assert(x≥m))
```

no predicates for n

predicate for the 1st argument of k

Unit type

Ex...te...ion

$(int[P] \to \star) \to \star$
$P(\mu) \equiv \mu \geq n\text{-}1$

$int[Q] \to \star$
$Q(\mu) \equiv \mu \geq n$

```
let sum n k = if n≤0 then k 0
                else sum (n-1) (λx.k (x+n))
let main m = sum m (λx.assert(x≥m))
```

$n>0 \wedge P(x) \Rightarrow Q(x+n)$

$sum: (\nu{:}int[] \to (int[\lambda\mu.\mu \geq \nu] \to \star) \to \star)$

b=true $\Leftrightarrow$ P(x)

```
let sum () k =
  if * then k true
       else sum () (λb. k (if b then true else rnd_bool))
let main () = sum () (λb. assert(b))
```

Successfully model checked!

Type-Directed Predicate Abstraction

Functional Program

Abstraction Type

$$\Gamma \vdash M : \tau \triangleright t$$

Abstraction Type Environment

Boolean Program

$$\frac{\Gamma \vdash M : \tau' \to \tau \triangleright s \quad \Gamma \vdash N : \tau' \triangleright t}{\Gamma \vdash M\ N : \tau \triangleright s\ t}$$

Predicate Abstraction Rule for Function Applications

Challenges in Higher-Order Setting

- Predicate Abstraction
  - How to ensure consistency of abstraction across function boundaries?

- Predicate Discovery
  - How to find new predicates that can eliminate an infeasible error trace from the abstraction?

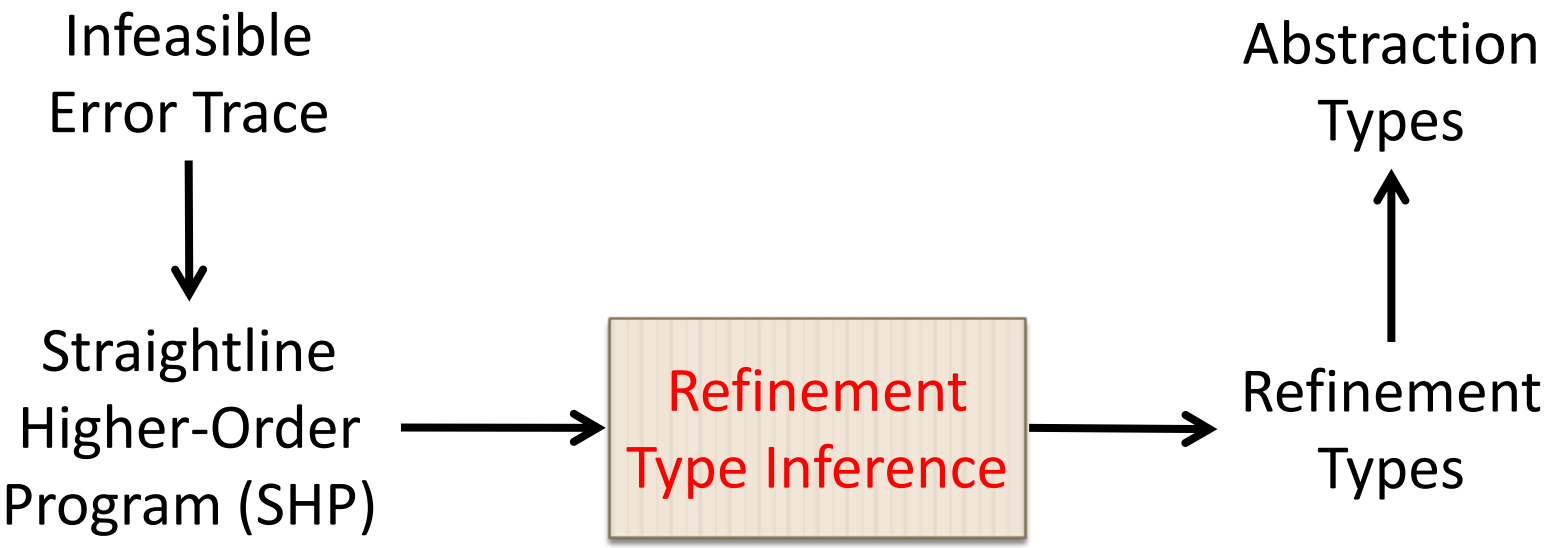
Challenges in Higher-Order Setting

- Predicate Abstraction
  - How to ensure consistency of abstraction across function boundaries?

- Predicate Discovery
  - How to find abstraction types that can eliminate an infeasible error trace from the abstraction?

Our Solution

- Reduction to refinement type inference of a straightline higher-order program (SHP)



Example: Abstraction Type Finding (1/2)

```
let sum n k = if n≤0 then k 0
                     else sum (n-1) (λx.k (x+n))
let main m = sum m (λx.assert(x≥m))
```

Infeasible error trace:

```
main m → sum m (λx.assert(x≥m))
→ if m≤0 then (λx.assert(x≥m)) 0 else …
```

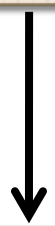
m≤0 → `(λx.assert(x≥m)) 0`

```
→ assert(0≥m)
```

0<m → `fail`

Example: Abstraction Type Finding (2/2)

```
let sum n k = if n≤0 then k 0
                     else sum (n-1) (λx.k (x+n))
let main m = sum m (λx.assert(x≥m))
```



main m →* if m≤0… $\rightarrow_{m\leq 0}$ assert(0≥m) $\rightarrow_{0<m}$ fail

Straightline Higher-Order Program (SHP):

```
let sum n k = assume(n≤0); k 0
let main m = sum m (λx.assume(x<m); fail)
```

[Unno and Kobayashi PPDP'09]

Abstraction Type:

sum: $(\nu{:}\mathrm{int}[] \rightarrow (\mathrm{int}[\lambda x.x \geq \nu] \rightarrow \star) \rightarrow \star)$

Properties of Our Model Checker

- No false negative
- No false positive
- Not guaranteed to terminate
- An error trace is eventually found (if any)
- Guaranteed to progress
  - Any infeasible error trace is eventually eliminated from the abstraction

MoCHi: Model Checker for Higher-Order Programs

- Prototype model checker for OCaml
- Implemented using:
  - TRecS [Kobayashi PPDP'09] as higher-order model checker
  - CVC3 [Barrett & Tinelli CAV'07] for predicate abstraction
  - CSIsat [Beyer et al. CAV'08] for predicate discovery

Preliminary Experiments (excerpted)

Program	CEGAR Loops	Time (s.)	
mc91	2	0.07	
ackermann	3	0.15	
repeat	3	0.15	
a-max	5	4.78	arrays
a-max-e	2	0.13	
l-zipunzip	3	0.12	lists
e-fact	2	0.01	exceptions
r-file	12	5.23	resource usage analysis

Encoded by functions:

let make n = λi.assert(0$\leq$i$\wedge$i<n); 0

let update i x n a =
  assert(0$\leq$i $\wedge$ i<n);
  λj.if i=j then x else a(i)

Eliminated by CPS transformation

Environment: Intel® Xeon® 3GHz + 8GB memory

Summary of Main Results

- Predicate abstraction and CEGAR for higher-order model checking
  - How to ensure consistency of abstraction?
    ▷ Abstraction types to express abstraction interfaces
  - How to find abstraction types?
    ▷ Reduction to refinement type inference of SHP
- Prototype model checker MoCHi for OCaml
  - Web interface available from:
    http://www.kb.ecei.tohoku.ac.jp/~ryosuke/cegar/
- Preliminary experiments