An Executable Semantics for Taint Analysis in the K Framework

Md. Imran Alam, Raju Halder, Harshita Goswami

Indian Institute of Technology Patna, India

Jorge Sousa Pinto

HASLab/INESC TEC & Universidade do Minho, Braga, Portugal

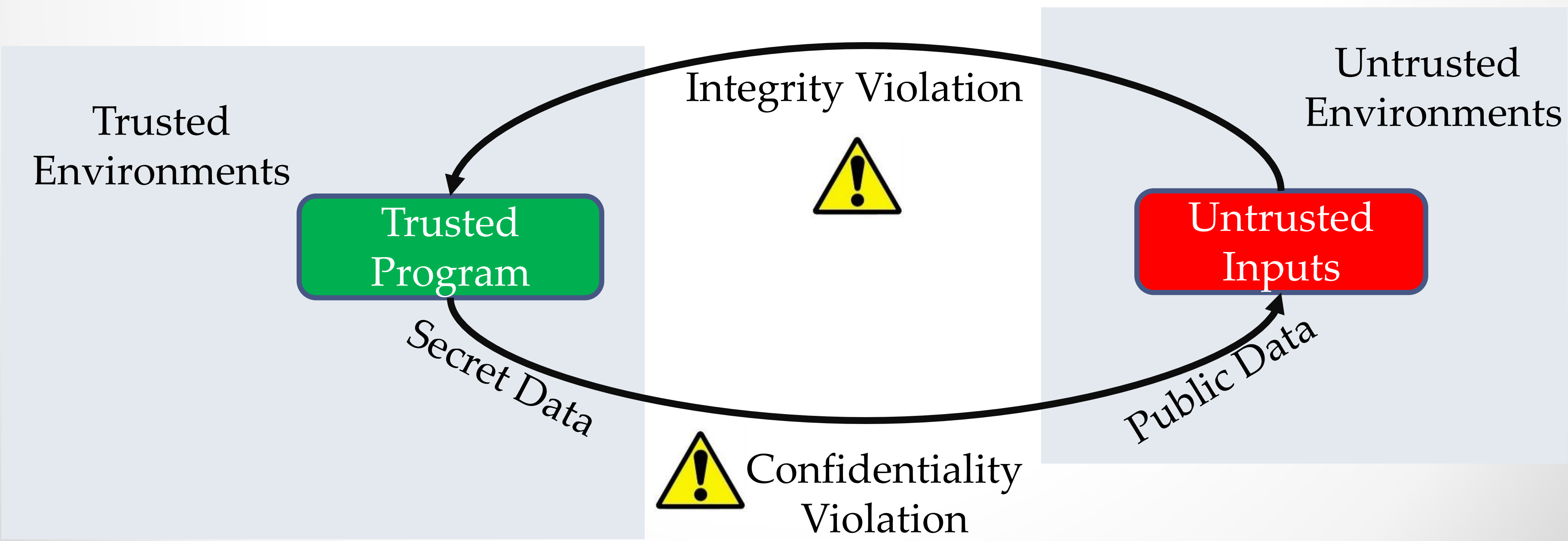
ENASE 2018

FUNCHAL, MADEIRA, PORTUGAL

23-24 MARCH 2018

Taint Analysis

- **Taint analysis** aims to averts effect of **malicious inputs** from corrupting values involved in **critical computations.**
- May compromise integrity.



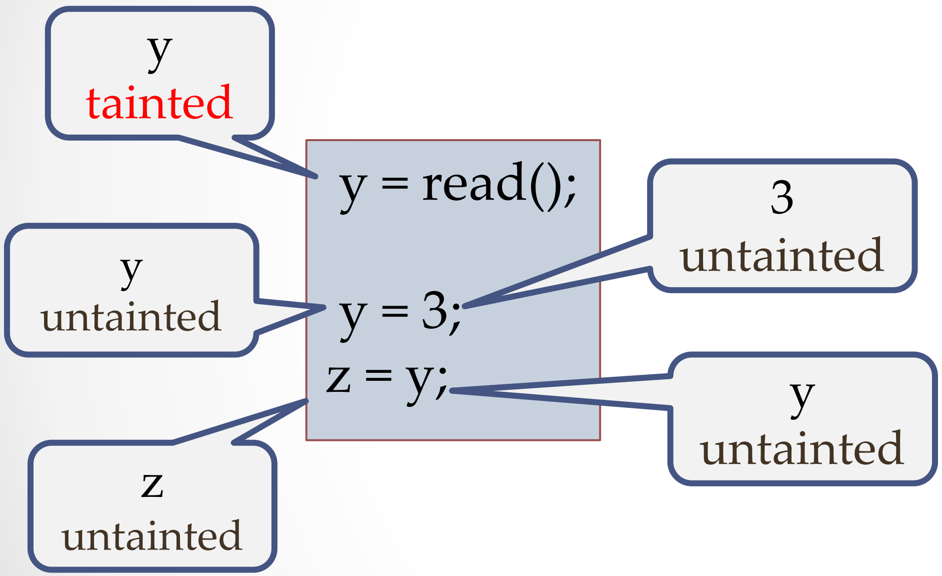
Taint Analysis

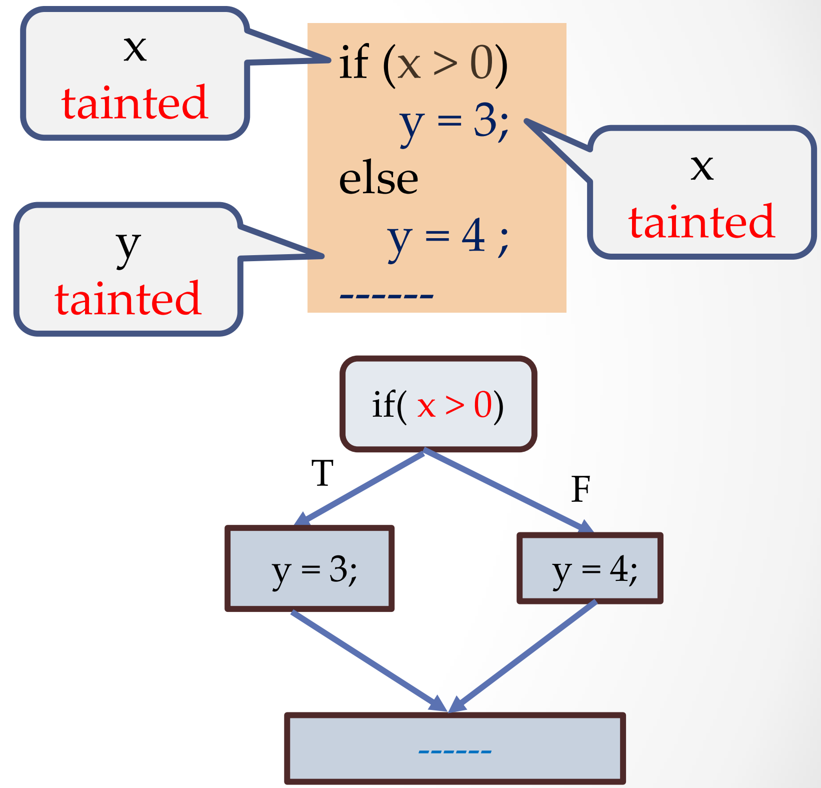
Tainted Input

```c
void foo(char *src){
 int i;
 char buf[20];
 for(i=0; i<= strlen(src); i++)
    buf[i] = src[i];
 return;
}
```

Out of array bound writing

Challenges: Information Flow





- **Explicit Flow**
- **Implicit Flow**

Challenges: **Flow Sensitivity**

- Order of statements taken into account.

y
tainted

1. read(y);

2. y = 3;

3. z = y;

y
untainted

z
untainted

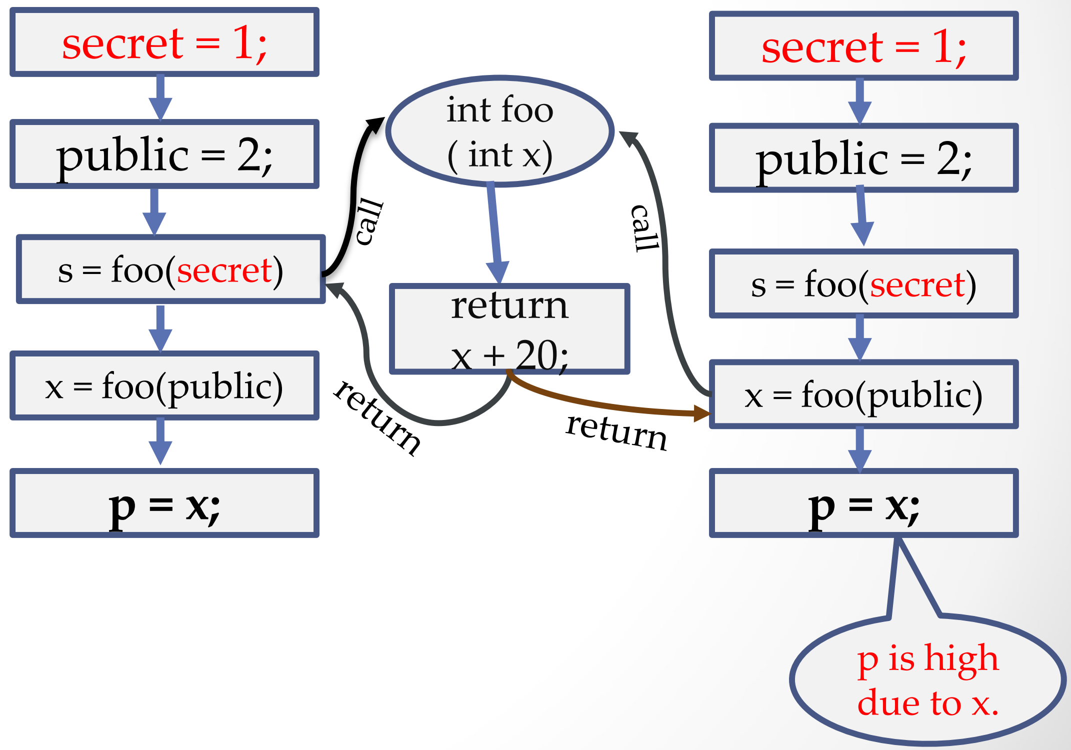
Challenges: **Context Sensitivity**

- Procedure calling context is taken into account.

1. secret = 1;   // taint
2. public = 2;   // untaint
3. s = foo(secret)
4. x = foo(public)

5. p = x;   // p value may be tainted in **context-insensitive** call.

```
int foo(int x) { return x + 20; }
```



Source: [*Hammer et. al. 2009*]

Challenges: Constant Functions

- Output the function is **independent** of its inputs.

- Examples:

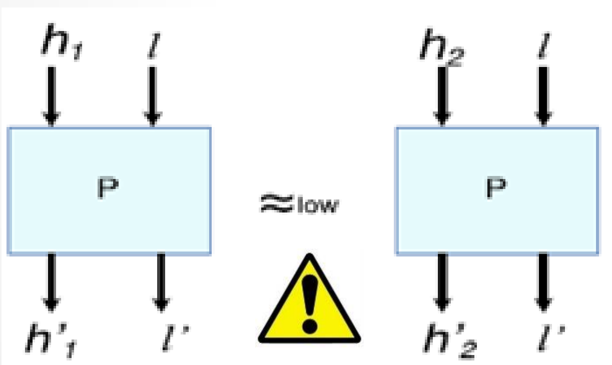
$v = 4 * X \bmod 2;$
$x = y - y;$
$z = x * 0;$
etc.

Related Works

Type System [Sabelfeld And Myers, 2003]

$$[pc] \vdash h := exp \qquad \frac{\vdash exp : low}{[low] \vdash l := exp}$$

$$\frac{\vdash exp : pc \quad [pc] \vdash C_1 \quad [pc] \vdash C_2}{[pc] \vdash \textsf{if } exp \textsf{ then } C_1 \textsf{ else } C_2}$$




```
1. if(secret == 1)
2.        public = 12;
3. else
4.        public = 3;
5.-------
6.-------
7.--------
8.--------
9.--------
10. public = 10;
```

type(secret)= high

type(public) = high

Post-dominates the "if" and there is no intermediate output of public.

type(public) = high

Related Works

Dynamic Taint Analysis

- Marking and tracking certain data in a program at run-time.
- Executes one program path at a time.
- More suitable in real time setting.
- More accurate.
- Less precision (False Negatives)
- DYTAN [13]

Related Works

Static Taint Analysis

- Analyze taint propagation along all possible path statically.
- Mostly based on data flow analysis.
- Sound
- False positives
- PIXY [11], TAINTGRIND [3], SAINT[7], TAJ[8], etc.

Related Works

Static Taint Analysis : PIXY

- Detect Cross-Site Scripting (XSS) vulnerabilities of server-side PHP Programs.
- Based on data flow.
- Flow-sensitive, inter-procedural, and context-sensitive.
- **System Overhead** (construct parse tree for control flow graph, generate three address code from parse tree).

source

sink

```
1.  x = read()       //   read() <- taint, x <- taint
1.  if (x > 0)
2.     y = 3;        //  3 <- untaint,  y <- untaint
3.  else
4.     y = 4 ;       //  4 <- untaint,  y <- untaint
6. print y;          // No sensitive leakage !
```

Related Works

Static Taint Analysis : SPLINT

- Data flow, control flow, flow-sensitive.
- Require annotations to various programming constructs such as function parameters, return values, global variables, etc.
- Manual annotation becomes impractical for large programs.

Related Works

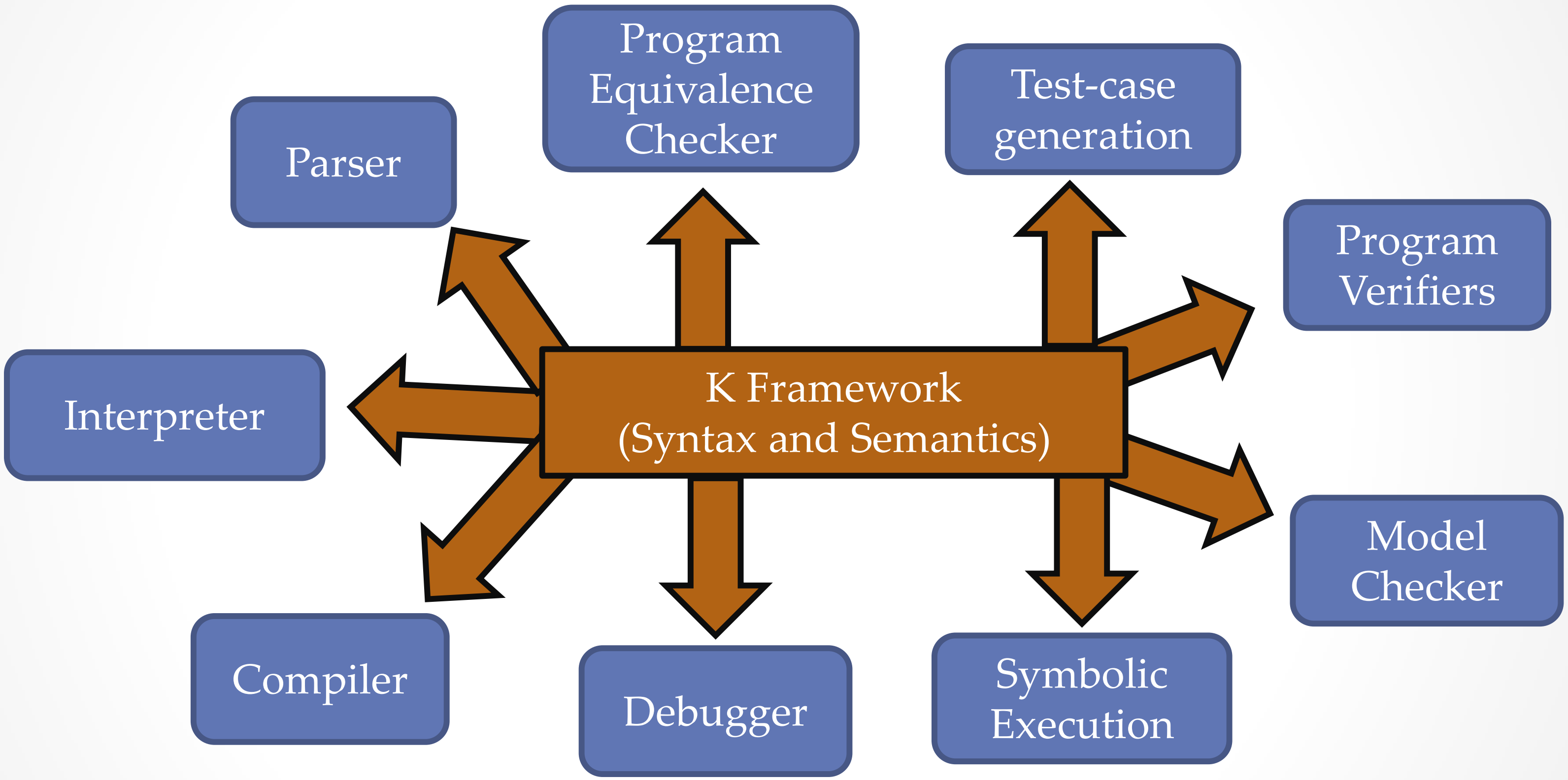
Comparative Summary: ☑ denotes partial successful

	K-Taint	Pixy	Taintgrind	SAINT	TAJ	Splint	Parfait	SFlow	CQual	KLEE
Semantics/Security Type System	✓	✗	✗	✗	✓	✗	✓	✓	✓	✓
Explicit Flow	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Implicit Flow	✓	✗	✗	✗	✗	✓	✓	✗	✗	✓
Constant Functions	☑	✗	✗	✗	✗	✗	✗	✗	✗	✗
Flow-Sensitivity	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
Context-Sensitivity	✓	✓	✗	✓	✓	✗	✓	✓	✗	✓
Language Supported	Imperative (including C-like syntax)	PHP	C	C	Java	C	C	Java	C	C

Motivational Factors

- Flow-Sensitive
- Context-Sensitive
- Semantics and Constant Functions
- Scalability and Precision

The K Framework



Source: [Grigore Rosue, ***LMCS***, 2017]

The K Framework

- Relies on *configuration* and **rewrite rules**.

- *Configuration* specifies the structure of the abstract machine and represented by labeled nested cells.

$$Configuration \equiv \langle \langle K \rangle_k \langle Map[Var \rightarrow Loc] \rangle_{env} \langle Map[Loc \rightarrow Val] \rangle_{store} \rangle \, T$$

$\langle\rangle_k$ - K cell hold list of computational task

$\langle\rangle_{env}$ - *env* cell maps variables (*Var*) to their locations (*Loc*)

$\langle\rangle_{store}$ - *store* cell maps locations (*Loc*) to values (*Val*)

$\langle\rangle \, T$ - *Top* is a special cell which contains all these cells denoted by *T*

The K Framework

- **Rewrite rules**
  - *Computational Rules*
    - X = &Y;

… represents remaining computations

match any where in the environment

$$\langle \frac{\&Y}{L} \ldots \rangle_k \langle \ldots Y \mapsto L \ldots \rangle_{env}$$

  - *Structural Rules*
    - `syntax` E ::= E_1 "+" E_2 `[strict]`

Hole (Freeze operation)

Heating Rules

Cooling Rules

$$\langle \frac{E_1 + E_2}{E_1 \curvearrowright \square + E_2} \ldots \rangle_k \text{ [structural]} \quad | \quad \langle \frac{E_1 + E_2}{E_2 \curvearrowright \square + E_1} \ldots \rangle_k \text{ [structural]}$$

$$\langle \frac{V_1 \curvearrowright \square + E_2}{V_1 + E_2} \ldots \rangle_k \text{ [structural]} \quad | \quad \langle \frac{V_2 \curvearrowright E_1 + \square}{E_1 + V_2} \ldots \rangle_k \text{ [structural]}$$

Abstract Syntax

Syntactic Elements
n : Numerical Values
id : Identifiers
E : Arithmetic Expressions
B : Boolean Expressions
C : Program Statements

Arithmetic Expressions

$$
\begin{aligned}
E ::= \ & n \\
\mid \ & id \\
\mid \ & \&id \\
\mid \ & {}^{*}E \\
\mid \ & id[E] \\
\mid \ & E \ op \ E \\
\mid \ & (E)
\end{aligned}
$$

where $op \in \{+, -, \times, /\}$

Boolean Expressions

$$
\begin{aligned}
B ::= \ & true \\
\mid \ & false \\
\mid \ & E \ rel \ E \\
\mid \ & \neg B \\
\mid \ & B \ AND \ B \\
\mid \ & B \ OR \ B
\end{aligned}
$$

where $rel \in \{\geqslant, \leqslant, <, >, ==\}$

$$
\begin{aligned}
\tau \ & ::= int \mid float \mid char \mid bool \mid \tau[n] \mid \tau^{*} \\
D \ & ::= \tau \ id \\
A \ & ::= id := E \mid {}^{*}E := E \mid id[E] := E \mid id := read() \\
C \ & ::= skip; \mid D; \mid A; \mid defun \ id(\vec{D})\{C\} \mid call \ id(\vec{E}); \mid return; \mid return \ E; \\
& \quad \mid C_1 \ C_2 \mid if \ B \ then \ \{C\} \mid if \ B \ then \ \{C_1\} \ else \ \{C_2\} \mid while \ B \ do \ \{C\}
\end{aligned}
$$

Extending Hunt and Sand Security Type System

- Incorporate flow-, context-sensitive.
- Security Type Rules:

[Expression] $$\frac{}{\Gamma \vdash E : \ \sqcup_{x \in \mathtt{FV}(E)} \Gamma(x)}$$

[skip] $$\frac{}{pc \vdash \Gamma\{skip\}\Gamma}$$

[Declaration] $$\frac{}{pc \ \vdash \ \Gamma \ \{\tau \ id\} \ \Gamma[\![id \mapsto pc \sqcup untaint]\!]}$$

[Read] $$\frac{}{pc \ \vdash \ \Gamma \ \{id = read()\} \ \Gamma[\![id \mapsto pc \sqcup taint]\!]}$$

[Assignment] $$\frac{\Gamma \ \vdash \ E : \ \mathtt{T}}{pc \ \vdash \ \Gamma \ \{id = E\} \ \Gamma[\![id \mapsto pc \sqcup \mathtt{T}]\!]}$$

[Function Call] $$\frac{\Gamma \ \vdash \ \vec{\mathtt{E}} : \vec{\mathtt{T}} \qquad \begin{array}{c} defun \ id(\vec{D})\{C\} \\ \vec{X} = getParam(\vec{D}) \\ \Gamma[\![\vec{X} \mapsto \vec{\mathtt{T}}]\!] \equiv \Gamma' \end{array} \qquad \dfrac{pc \vdash \Gamma' \ \{\mathtt{C}\} \ \Gamma''}{pc \vdash \Gamma' \ \{defun \ id(\vec{D})\{C\}\} \ \Gamma''}}{pc \ \vdash \ \Gamma \ \{call \ id(\vec{E})\} \ \Gamma''}$$

[if-else] $$\frac{\Gamma \ \vdash \ B : \mathtt{T} \qquad pc \sqcup \mathtt{T} \vdash \Gamma\{C_1\}\Gamma' \qquad pc \sqcup \mathtt{T} \vdash \Gamma\{C_2\}\Gamma''}{pc \ \vdash \ \Gamma \ \{if \ B \ then \ \{C_1\} \ else \ \{C_2\}\} \ \Gamma' \sqcup \Gamma''}$$

[while] $$\frac{\begin{array}{lll} \Gamma'_i \ \vdash \ B : \mathtt{T}_i & pc \sqcup \mathtt{T}_i \vdash \Gamma'_i\{\mathtt{C}\}\Gamma''_i & 0 \leq i \leq k \\ \Gamma'_0 = \Gamma & \Gamma'_{i+1} = \Gamma''_i \sqcup \Gamma & \Gamma'_{k+1} = \Gamma'_k \end{array}}{pc \ \vdash \ \Gamma \ \{while \ B \ do \ \{C\}\} \ \Gamma'_k}$$

K Specification of Extended Security Type System

- Typing Judgement
  - $Pc \vdash \Gamma\{C\}\Gamma'$ specifies Γ' derived after executing **C** on Γ under the security context *Pc*.
  - Equivalent K- *Configuration:*

$$configuration \equiv \left\langle \langle K \rangle_k \, \langle \mathtt{Map} \rangle_{env} \, \langle \mathtt{Map} \rangle_{context} \right\rangle_T$$

➢ Where cell *k* holds *C*, cell *env* holds Γ' *and* Γ' and cell *context* holds *Pc*.

➢ Corresponding *typing judgement* represented as following rule:

$$\langle \frac{C}{\cdot} \ldots \rangle_k \, \langle \frac{\Gamma}{\Gamma'} \rangle_{env} \, \langle pc \mapsto \_ \rangle_{context}$$

Taint Analysis in the K Framework

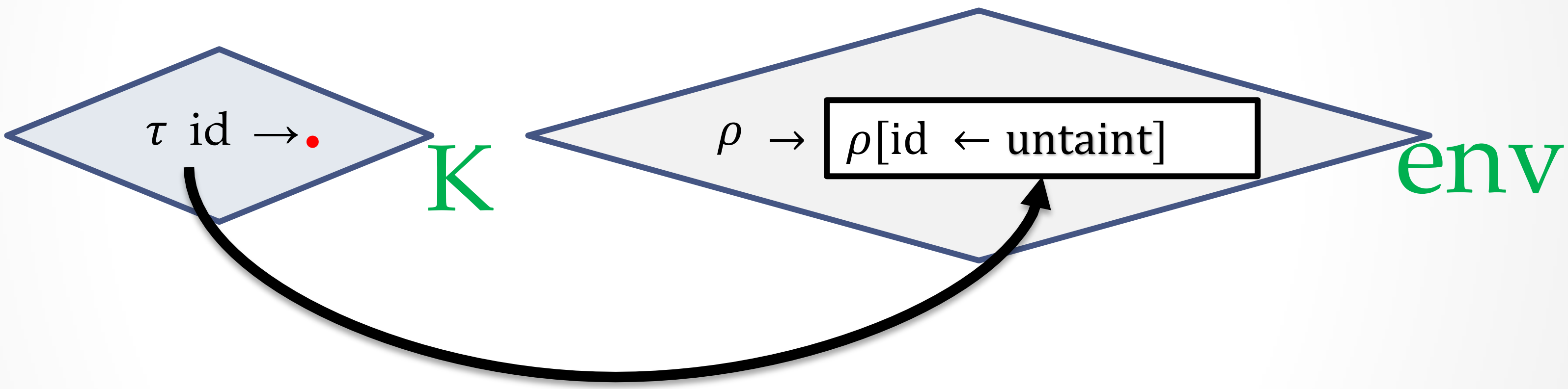
- Configuration

$$configuration \equiv \langle \langle K \rangle_k \ \langle Map \rangle_{env} \ \langle \mathtt{Map} \rangle_{context} \ \langle \ \langle \mathtt{Map} \ \rangle_{\lambda\text{-}Def} \ \langle \mathtt{List} \rangle_{fstack} \ \rangle_{control} \ \langle \mathtt{List} \rangle_{in} \ \langle \mathtt{List} \rangle_{out} \langle \ \langle \mathtt{Map} \rangle_{alias} \ \langle \mathtt{Set} \rangle_{ptr} \ \rangle_{ptr\text{-}alias} \ \rangle_T$$

- Cell ***in*** and ***out*** used to perform standard input-output operations
- Cell ***λ-Def*** supports inter-procedural holding the bindings of function names.
- Cell ***ptr-alias*** maintains pointer aliasing information in ***ptr*** and ***alias*** cells.
- All function calls are controlled by ***control*** cell maintaining a stack-based context switching using ***fstack*** cell.

K Semantics Rules

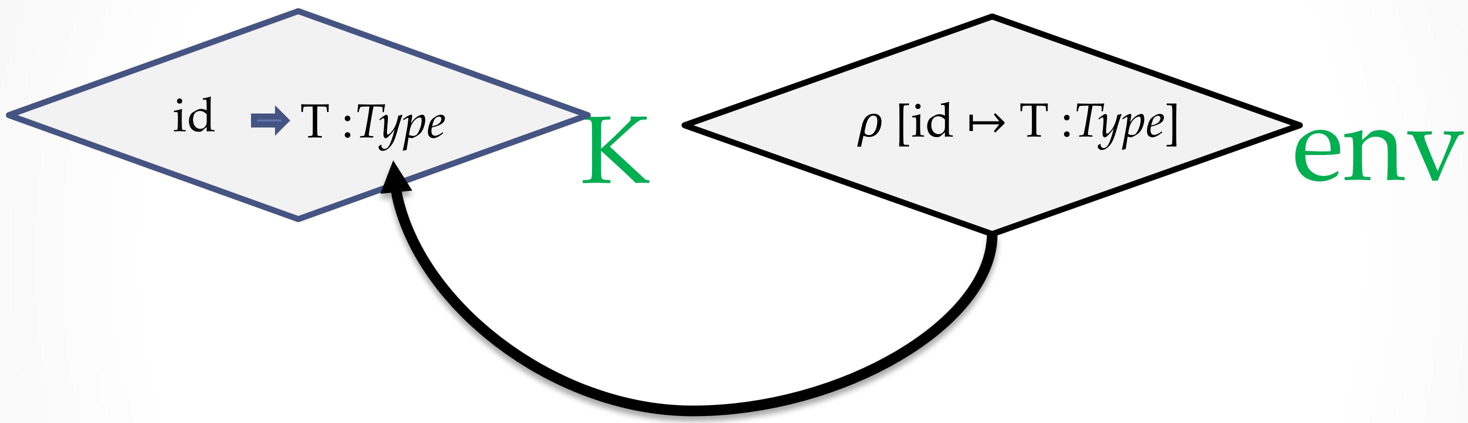
- Declaration: τ id;



$$(R_{dec}) :< \frac{\tau\ id}{\cdot} \ \ldots >_{K} < \frac{\rho}{\rho[\mathrm{id} \leftarrow \mathrm{untaint}]} >_{\mathrm{env}}$$

K Semantics Rules

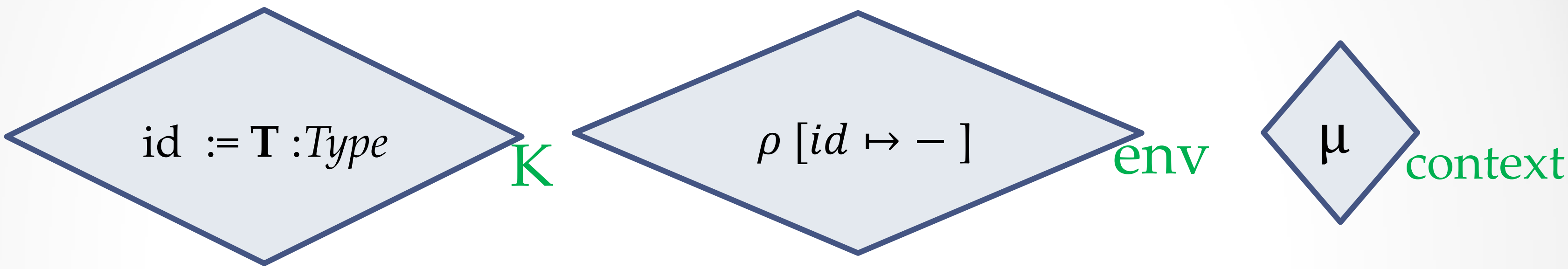
- Lookup: id



$$(R_{lookup}) :< \frac{id}{\mathrm{T}{:}Type} \; ... >_{K} < \cdots \; \mathrm{id} \; \mapsto \; \mathrm{T} :Type \; ... >_{\mathrm{env}}$$

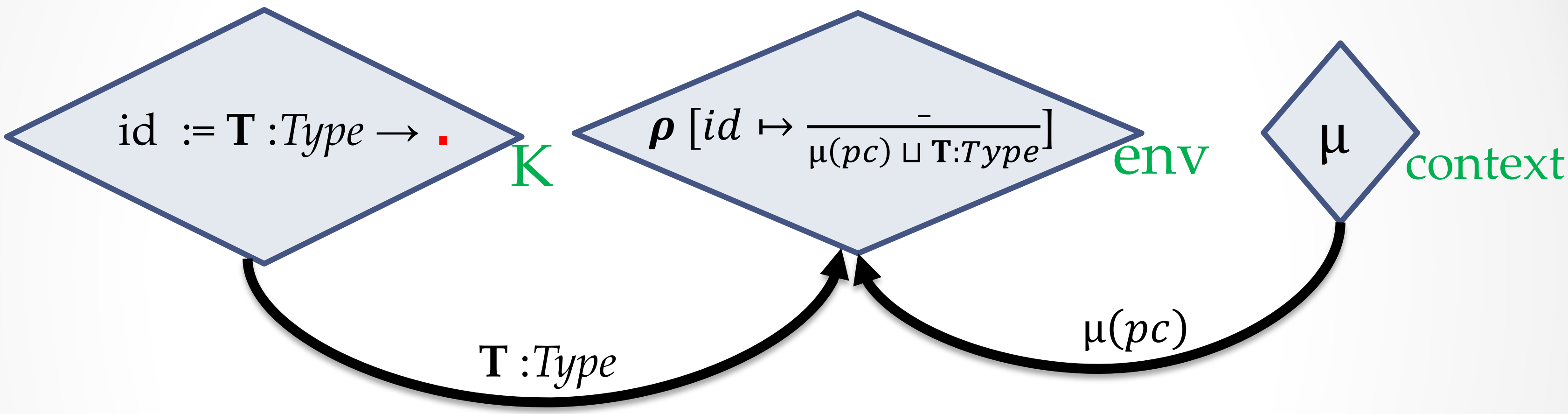
K Semantics Rules

- Assignment: id = T;



K Semantics Rules

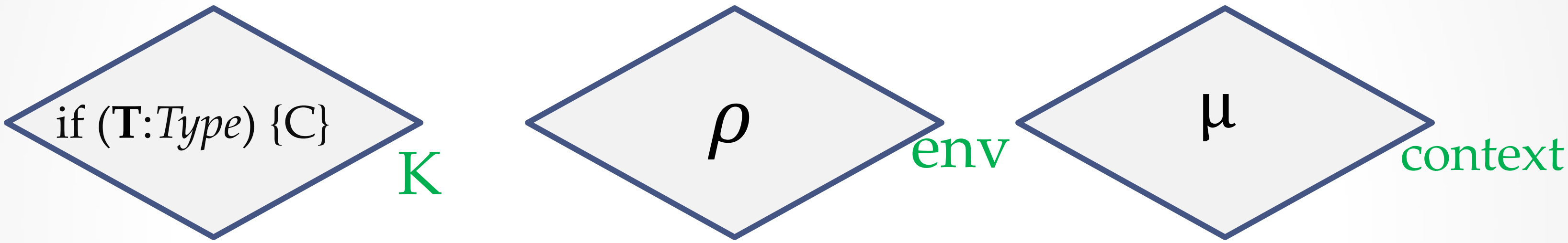
- Assignment: id = T;



$$(R_{lookup}): < \frac{id \coloneqq \mathbf{T}{:}Type}{.} \ldots >_{K} <\ldots\ \rho\ [id \mapsto \frac{-}{\mu(pc) \sqcup \mathbf{T}{:}Type}]\ \ldots>_{env} <\mu>_{context}$$

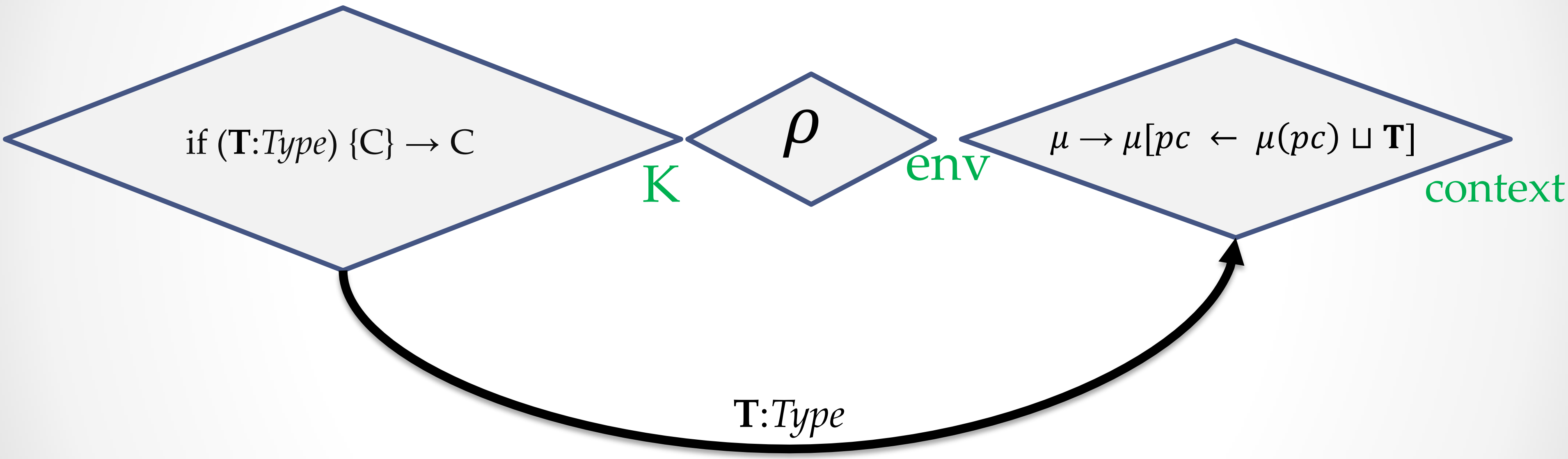
K Semantics Rules

- **if** : if (**T**:*Type*) {C}



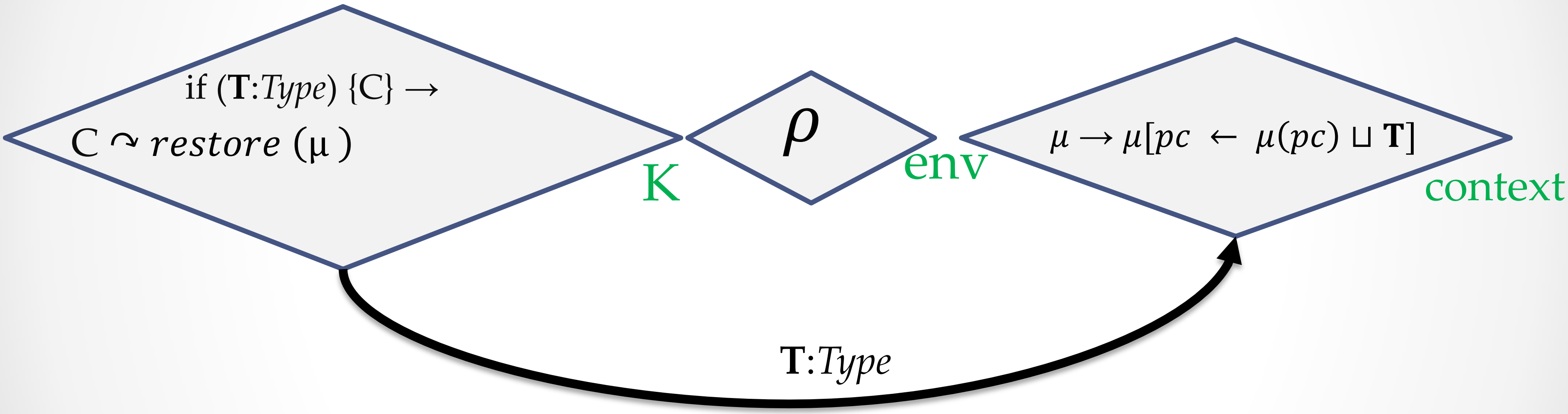
K Semantics Rules

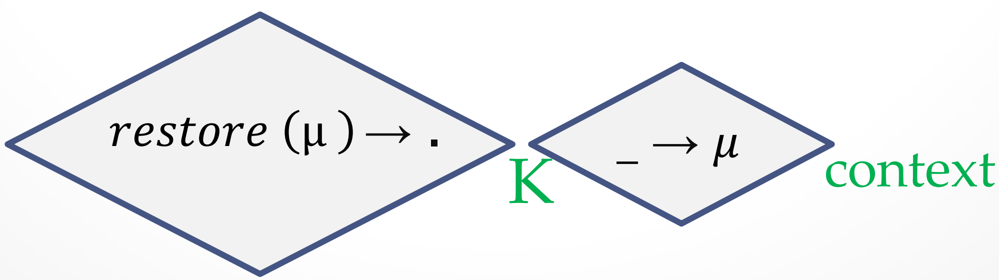
- **if** : if (**T**:*Type*) {C}



K Semantics Rules

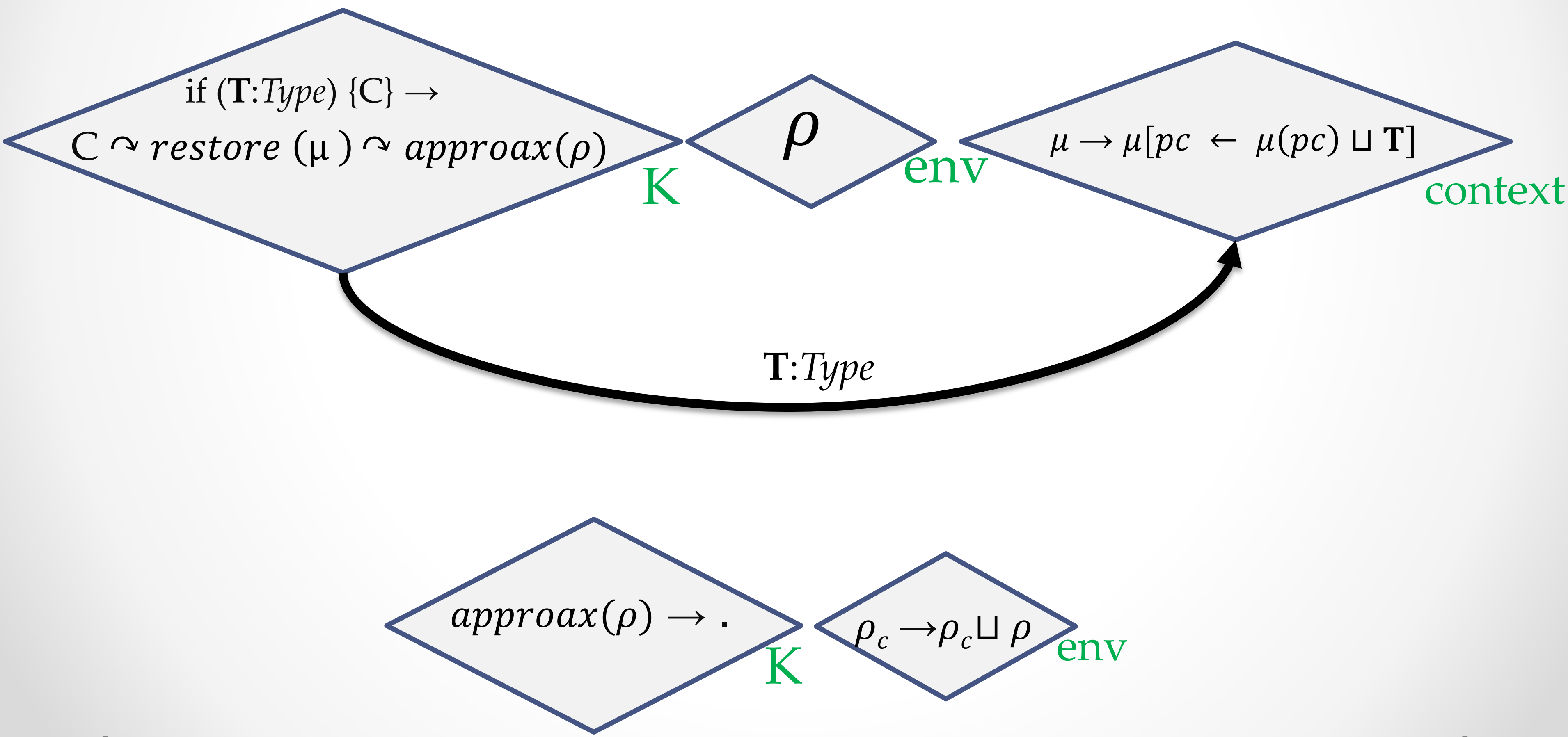
- **if** : if (**T**:*Type*) {C}





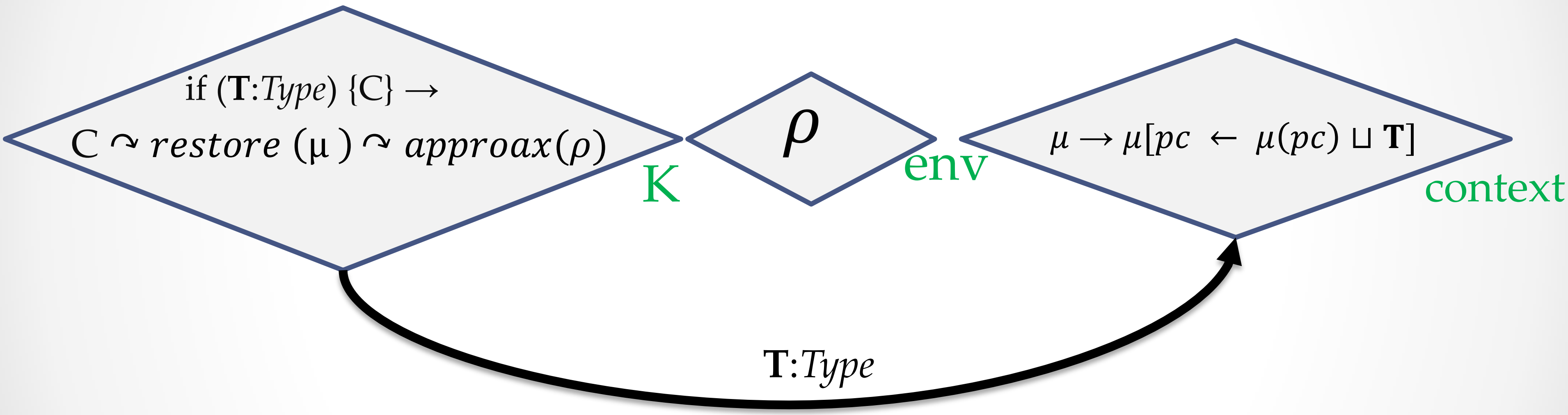
K Semantics Rules

- **if** : if (**T**:*Type*) {C}



K Semantics Rules

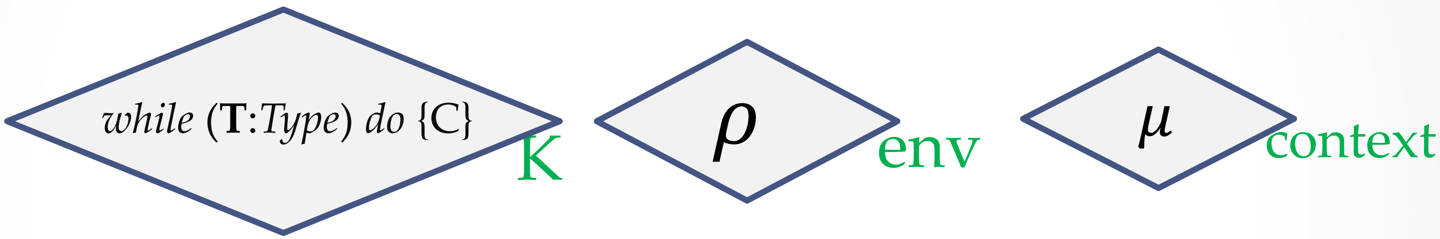
- **if** : if (**T**:*Type*) {C}



$$(R_{if}) : < \frac{if\ (\mathrm{T}: \mathrm{Type})\ then\ \{\mathrm{C}\}}{\mathrm{C} \curvearrowright restore\ (\mu\ ) \curvearrowright approax(\rho)} \ldots >_{\mathrm{K}} < \frac{\mu}{\mu[pc \leftarrow \mu(pc) \sqcup \mathrm{T}]} >_{\mathrm{context}} \ <\rho>_{\mathrm{env}}$$

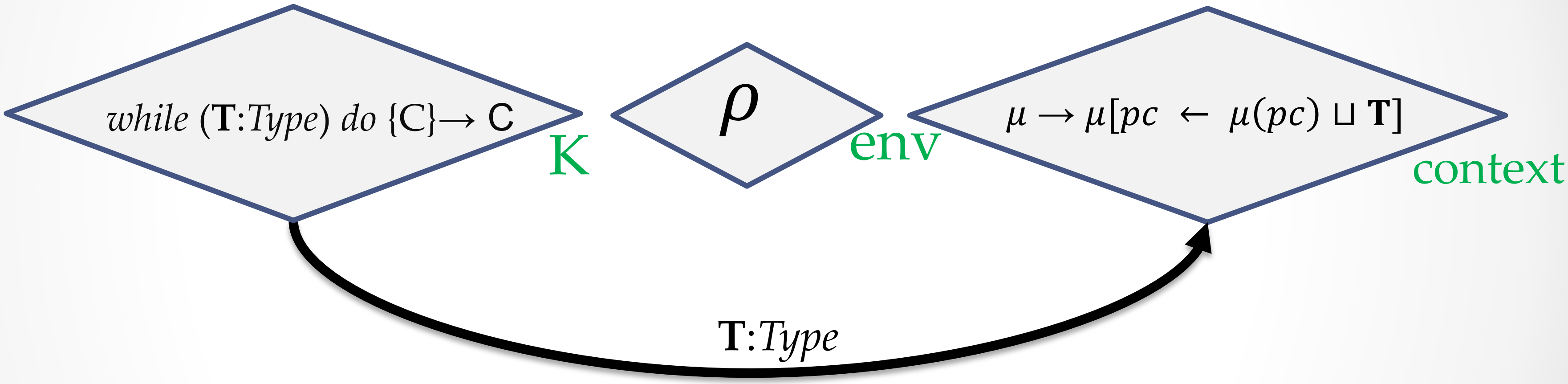
K Semantics Rules

- while: *while* (**T**:*Type*) *do* {C}



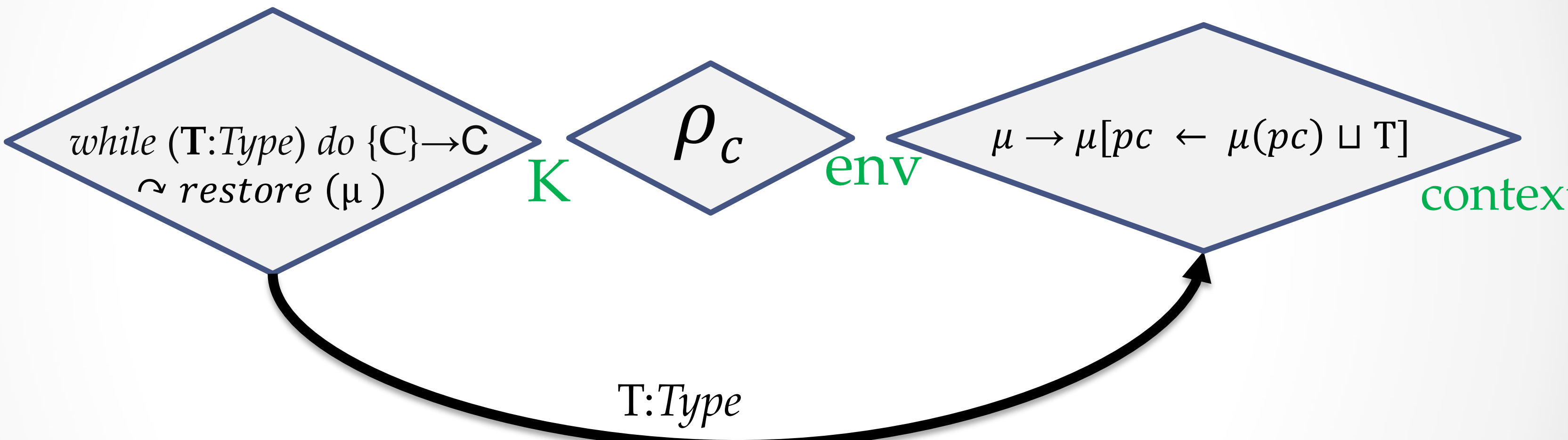
K Semantics Rules

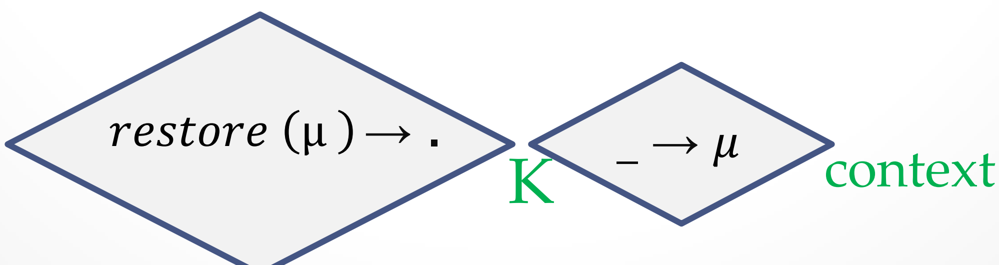
- while: *while* (**T**:*Type*) *do* {C}



K Semantics Rules

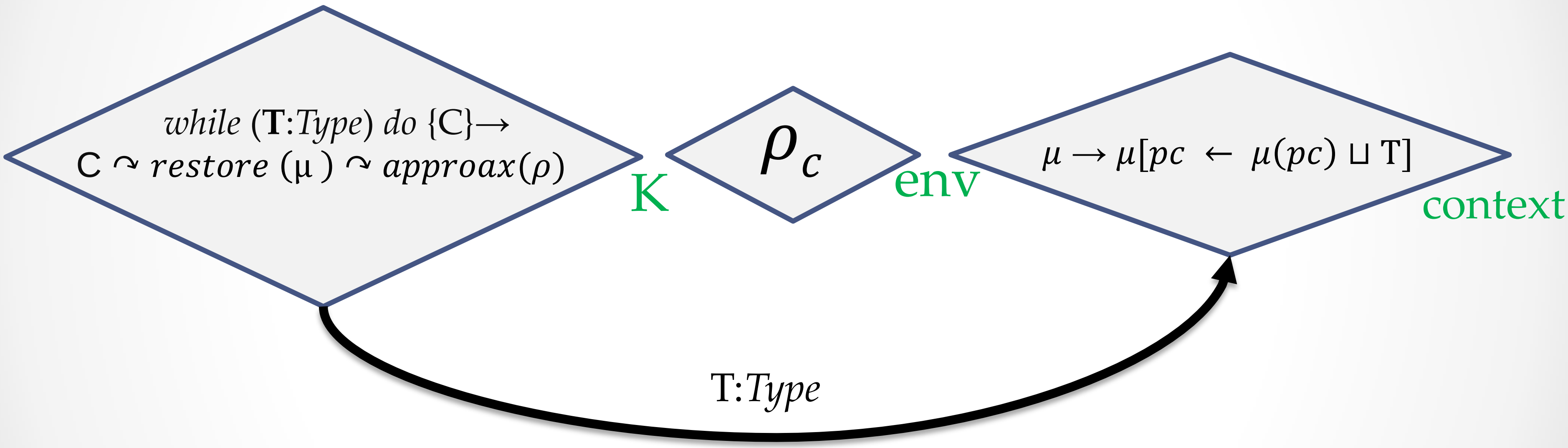
- while: *while* (T:*Type*) *do* {C}

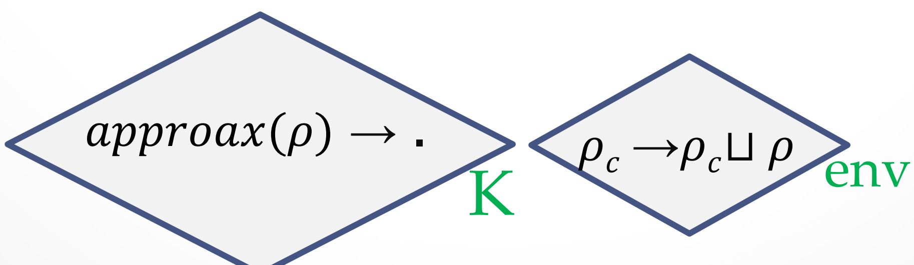




K Semantics Rules

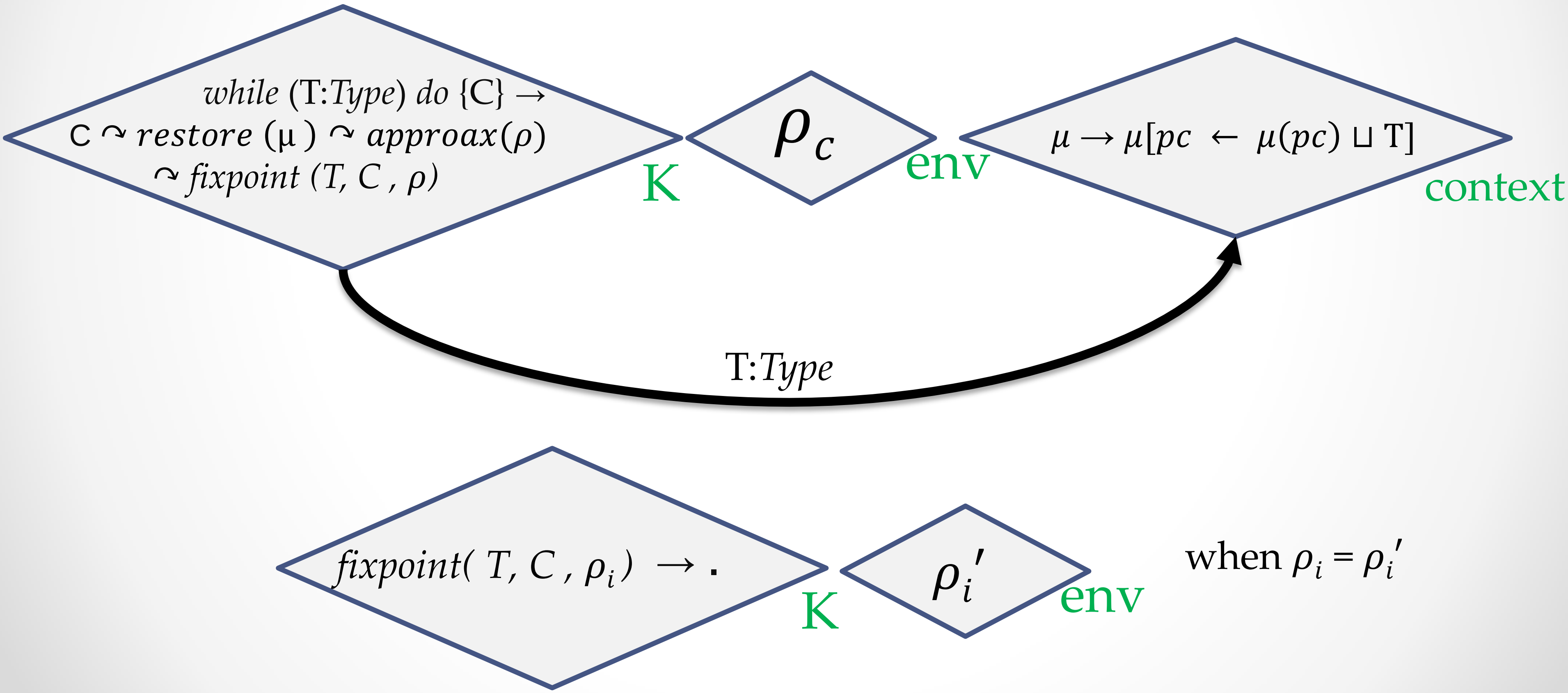
- while: *while* (T:*Type*) *do* {C}





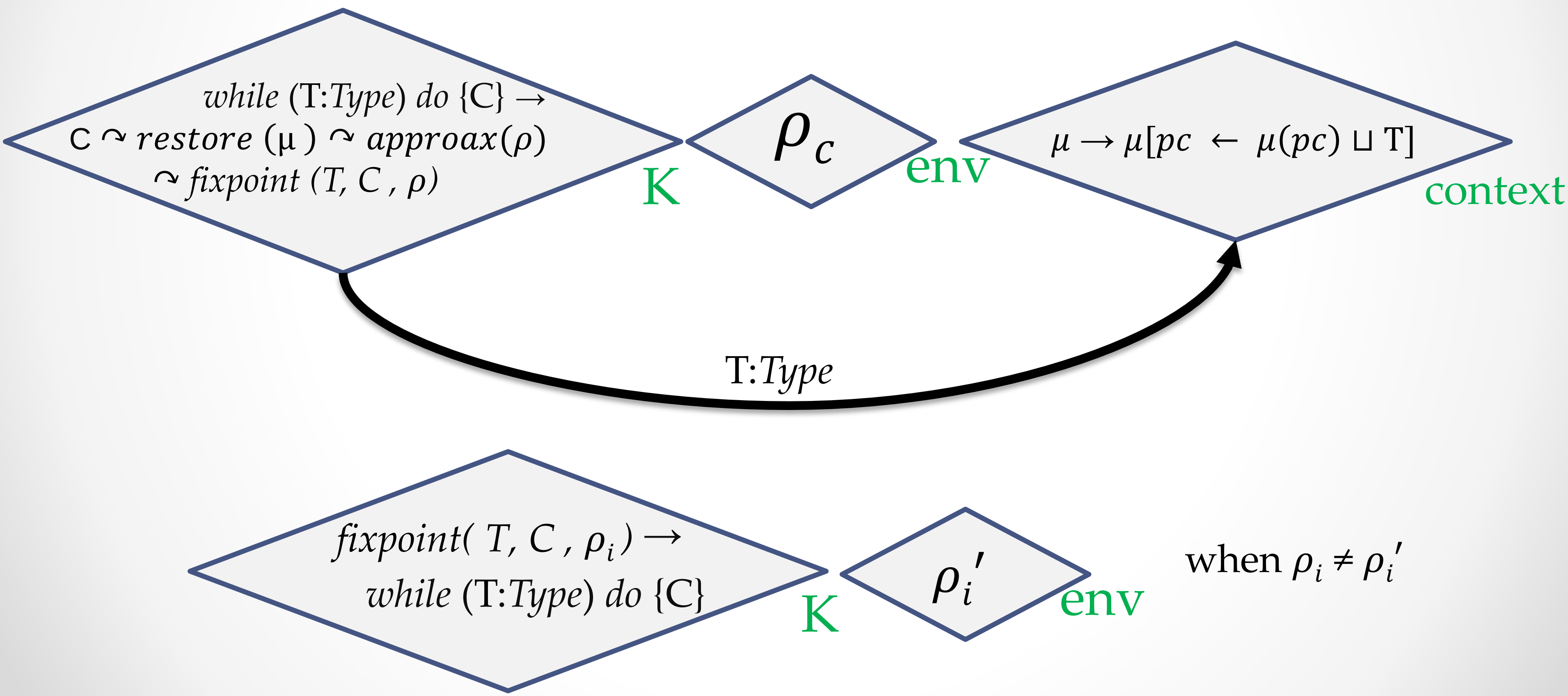
K Semantics Rules

- while: *while* (T:*Type*) *do* {C}



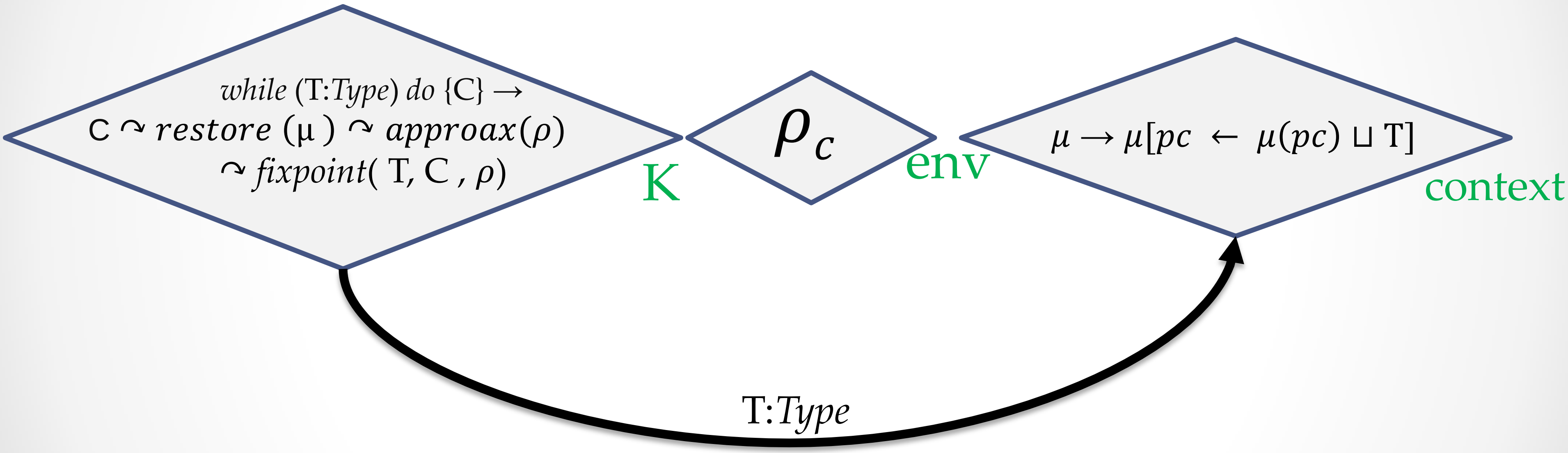
K Semantics Rules

- while: *while* (T:*Type*) *do* {C}



K Semantics Rules

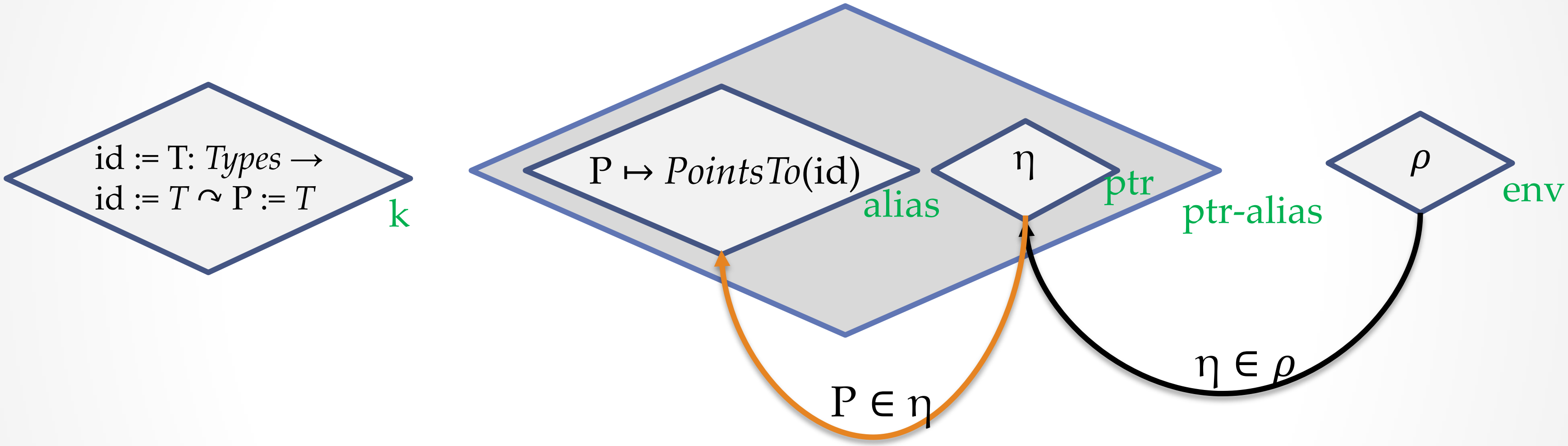
- while: *while* (T:*Type*) *do* {C}



$$(R_{while}): < \frac{while\ (\mathrm{T: Type})\ do\ \{\mathrm{C}\}}{\mathrm{C} \curvearrowright restore\ (\mu\ ) \curvearrowright approax(\rho) \curvearrowright fixpoint(T,C,\rho)} \ldots >_{\mathrm{K}} < \frac{\mu}{\mu[pc \leftarrow \mu(pc) \sqcup \mathrm{T}]} >_{\mathrm{context}} <\rho>_{\mathrm{env}}$$

K Semantics Rules

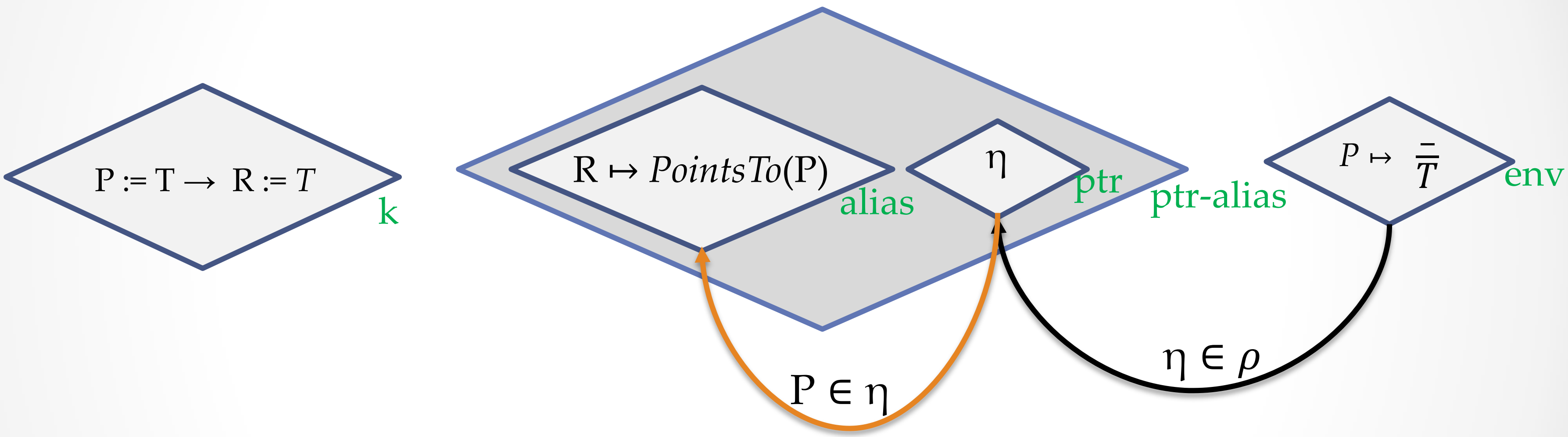
- Pointers: *id := T: Types*



$$(R_{pointers}): < \frac{id := T{:}Types}{id := T \curvearrowright P := T} \ldots >_{K} << \ldots P \mapsto PointsTo(id) \ldots >_{alias} < \eta >_{ptr} >_{ptr\text{-}alias} < \rho >_{env} \text{ when } P \in \eta$$

K Semantics Rules

- Pointers: *id := T: Types*



$(R_{pointers})$: $< \frac{P := T}{R := T} \ldots >_{K}$ $<< \ldots R \mapsto PointsTo(P) \ldots >_{alias}$ $< \eta >_{ptr} >_{ptr\text{-}alias}$ $< \ldots P \mapsto \overset{=}{T} \ldots >_{env}$ when P ∈ η

Illustration with Example

Source: [Cavallaro et al., 2008]

Prg Pts	Prg Stmt	Security Types	Context

Experimental Results

- ✗_+ And ✗_- this denotes **false positives** and **false negatives**.

Progs.	Descriptions	K-Taint	Splint [Evans and Larochelle, 2002]	Pixy [Jovanovic et al., 2006]	SFlow [Huang et al., 2014]	CQual [Foster et al., 2002]
Prog1	Explicit Flow	✓	✓	✓	✓	✓
Prog2	Implicit Flow	✓	✗_-	✗_-	✗_-	✗_-
Prog3	Malware Attack	✓	✗_-	✗_-	✗_-	✗_-
Prog4	XSS Attack	✓	✗_-	✗_-	✗_-	✗_-
Prog5	Buffer Overflow	✗_+	✓	✓	✗_+ ✗_-	✗_-
Prog6	Constant Function "`subtraction`"	✓	✗_+	✗_+	✗_+	✗_+
Prog7	Program consists of multiple functions	✓	✗_-, ✗_+	✗_-	✓	✗_-
Prog8	Program with context-sensitivity	✓	✗_-, ✗_+	✓	✓	✗_+
Prog9	Factorial Program	✓	✗_-	✗_-	✗_-	✗_-
Prog10	Binary Search	✗_+	✗_-	✗_-	✗_-	✗_-
Prog11	Merge Sort	✗_+	✗_-	✗_-	✗_-	✗_-
Prog12	Program with flow-sensitivity	✓	✗_-	✓	✗_-	✗_-
Prog13	Swapping of two numbers using pointers	✓	✓	✓	✓	✗_-

Illustration with Examples

Source: [Vogt et al., 2007]

Prog. Pts	Prog Stmts	Security Types
1	`int` x,y,z,a;	x|->U y|->U z|->U a|->U
2	$x = 0; y = 0;$	x|->Z y|->Z z|->U a|->U
3	z = `read();`	x|->Z y|->Z z|->T a|->U
4	`if` $(z \leq a)\{$	x|->Z y|->Z z|->T a|->U
5	x=1;}	x|->T y|->Z z|->T a|->U
6	`else`{	x|->T y|->Z z|->T a|->U
7	y=1;}	x|->T y|->T z|->T a|->U
8	`if`(x==0){	x|->T y|->T z|->T a|->U
9	$x = a;\}$	x|->T y|->T z|->T a|->U
10	if(y==0){	x|->T y|->T z|->T a|->U
11	$y = a;\}$	x|->T y|->T z|->T a|->U
`Output by K-Taint`		x|->T y|->T z|->T a|->U

Illustration with Examples

Source: [Russo and Sabelfeld, 2010]

Prog. Pts	Prog Stmts	Security Types
1	`int` *pub,temp,secret,input*;	*input* |->U *pub* |->U *temp* |->U *secret* |->U
2	*input* = `read();`	*input* |->T *pub* |->U *temp* |->U *secret* |->U
3	*pub*=1;	*input* |->T *pub* |->U *temp* |->U *secret* |->U
4	*temp*=0;	*input* |->T *pub* |->U *temp* |->Z *secret* |->U
5	`if`(*secret*≤*input*){	*input* |->T *pub* |->U *temp* |->Z *secret* |->U
6	*temp*=1;}	*input* |->T *pub* |->U *temp* |->T *secret* |->U
7	`if`(*temp*≤0){	*input* |->T *pub* |->U *temp* |->T *secret* |->U
8	*pub*=0;}	*input* |->T *pub* |->T *temp* |->T *secret* |->U
	`Output by K-Taint`	*input* |->T *pub* |->T *temp* |->T *secret* |->U

Conclusion

- We proposed an executable static taint analysis of an imperative programming language in the K framework.

- Incorporate flow-, context- sensitivity and pointer analysis.

- Our proposed technique improve the precision compared to the literature.

- We will incorporate some more programming features.

References

[1] W. Huang, Y. Dong, and A. Milanova. Type-based taint analysis for java web applications. In In Proc. of Int. Conf. on Fundamental Approaches to Software Engineering, pages 140–154. Springer, 2014.

[2] David Evans, David Larochelle, and David Evans. Splint manual: Version 3.1.1-1, 2003. http://www.splint.org/manual. [Online; accessed 15-June-2016].

[3] Christian Hammer and Gregor Snelting. Flow-sensitive, context- sensitive, and object-sensitive information flow control based on program dependence graphs. International Journal of Information Security, 8(6):399–422, 2009.

[4] G. Ros ̧u and T. F. S ̧erb ̆anut ̆a. An overview of the k semantic framework. The Journal of Logic and Algebraic Programming, 79(6):397–434, 2010.

[5] Xavier NOUMBISSI NOUNDOU. SAINT taint analyzer. https: //github.com/ Xavier Noumbis/ saint. [Online; accessed 15-August-2016].

[6] Andrei Sabelfeld and Andrew C Myers. Language-based information- flow security. IEEE Journal on selected areas in communications, 21(1):5–19, 2006.

[7] N. Jovanovic, C. Kruegel, and E. Kirda. PIXY: A static analysis tool for detecting web application vulnerabilities. In IEEE Symposium on Security and Privacy (S&P'06), pages pp. 258–263. IEEE. IEEE, 2006.

Thank You

?

Kindly mail to

imran.pcs16@iitp.ac.in

http://www.iitp.ac.in/~halder/ktaint/