Verifying Pointer and String Analyses with Region Type Systems

Lennart Beringer, Robert Grabowski, Martin Hofmann

Ludwig-Maximilians-Universität, Munich, Germany
Princeton University, Princeton, NJ

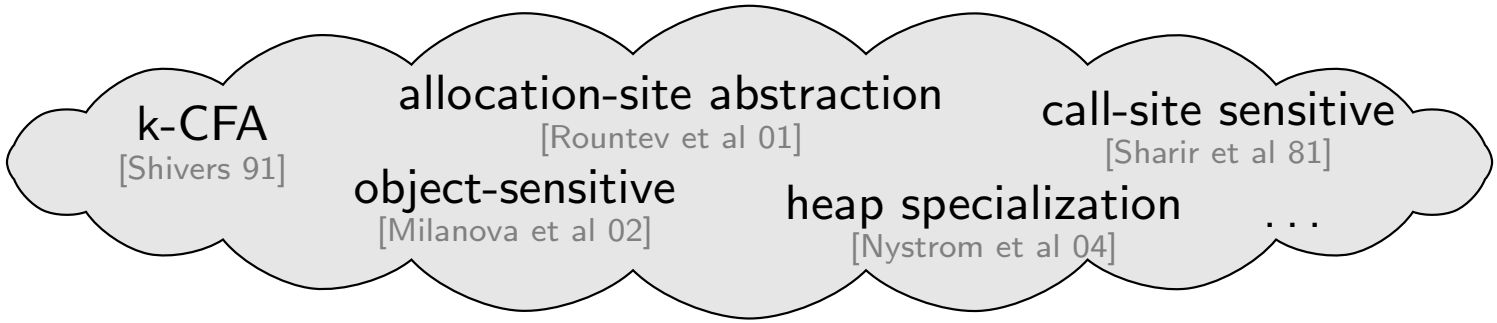
LPAR-16
April 2010, Dakar

Pointer analysis

analysis algorithms for Java programs obtaining non-alias relations:

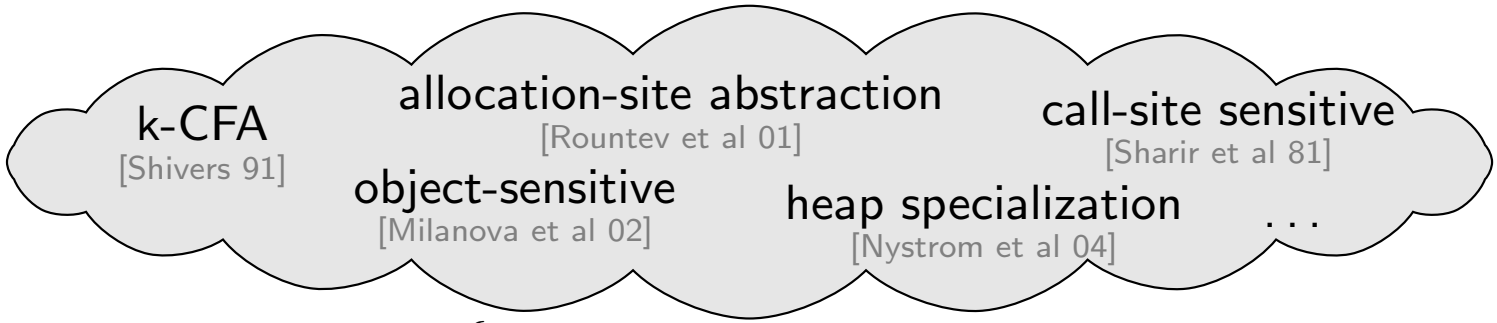
Pointer analysis

analysis algorithms for Java programs obtaining non-alias relations:



Pointer analysis

analysis algorithms for Java programs obtaining non-alias relations:

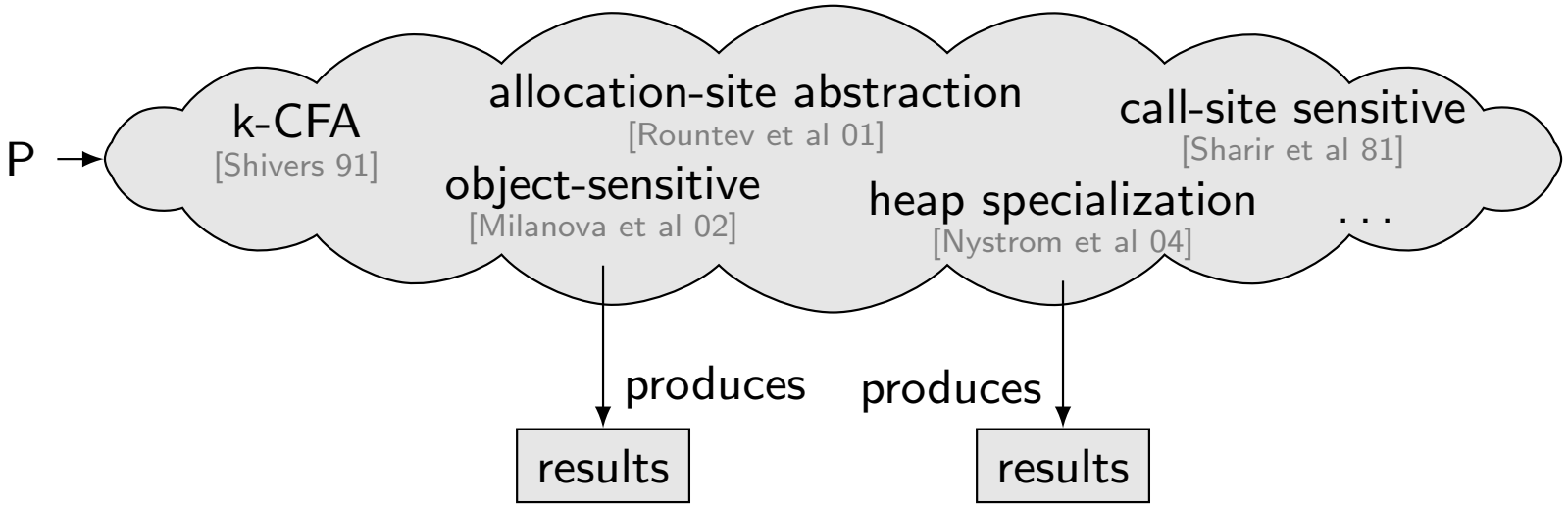



```
class List {
  Elem elem; List next;
   List copy() {
     List l = new List();
     l.elem = elem.clone();
     l.next = next.copy();
     return l;
   }
}
```

"`x.copy()` does not alias with `x`"
"`x.copy().next` does not alias with `x.next`"

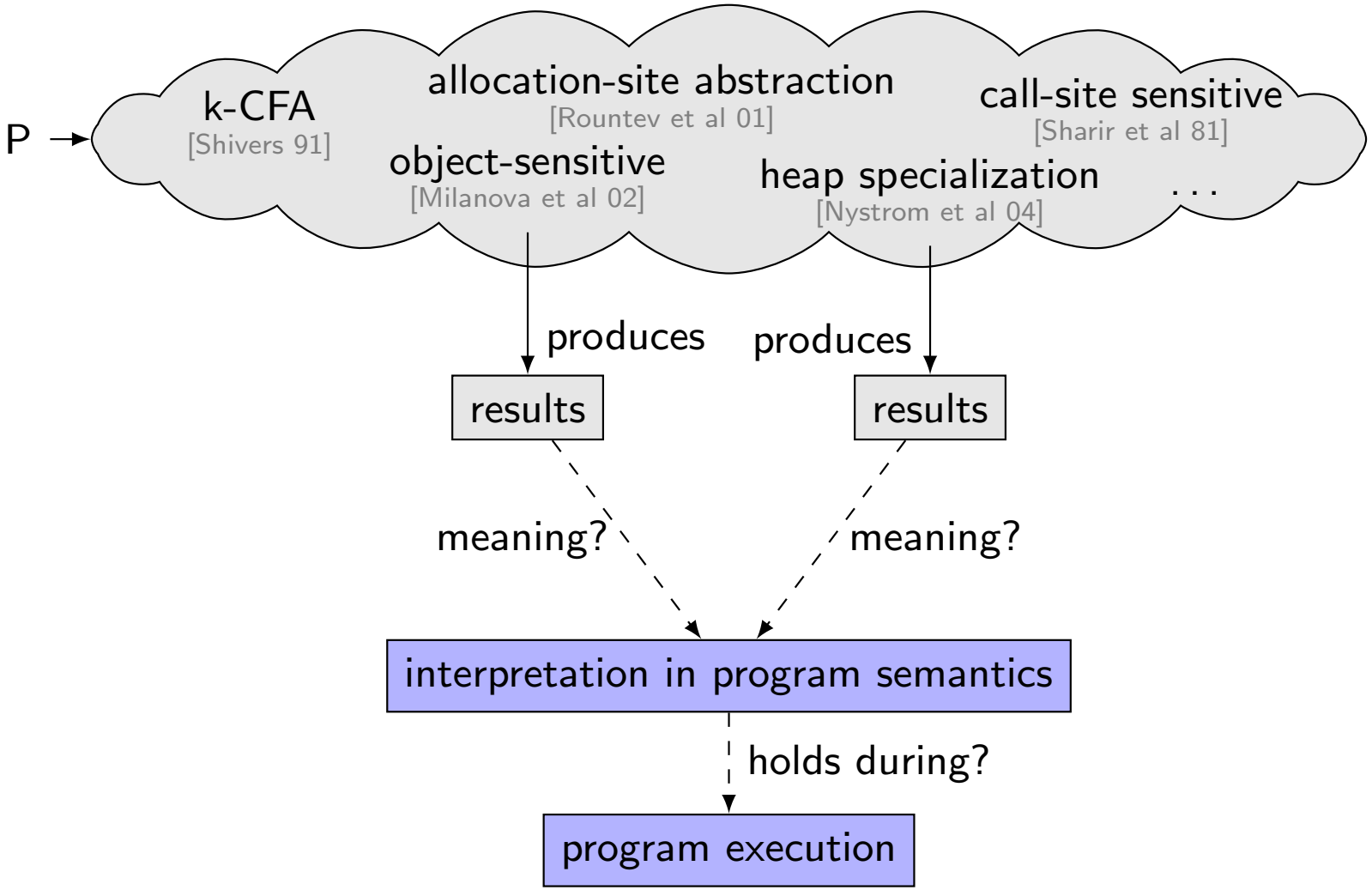
Pointer analysis

analysis algorithms for Java programs obtaining non-alias relations:



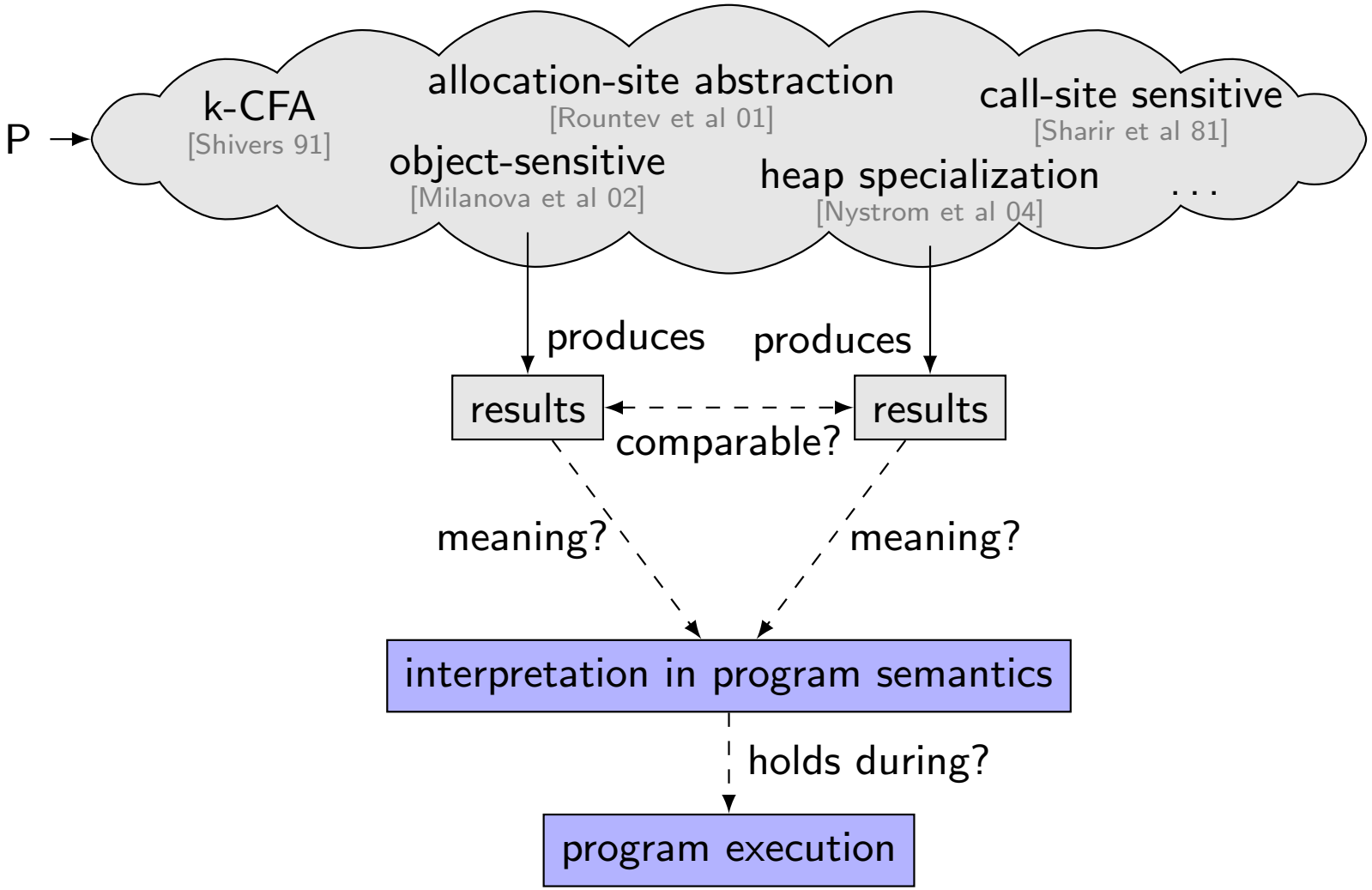
Pointer analysis

analysis algorithms for Java programs obtaining non-alias relations:



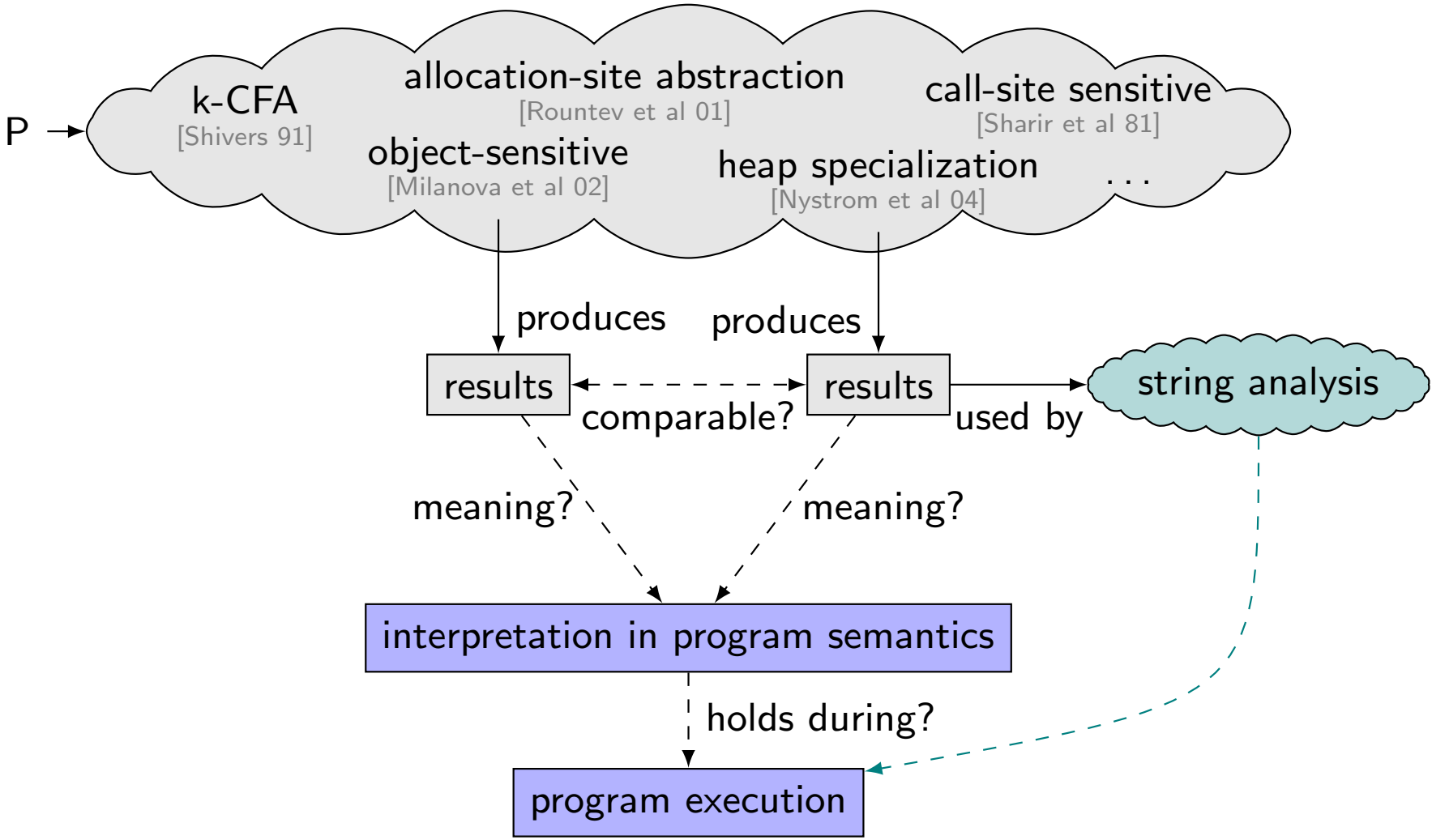
Pointer analysis

analysis algorithms for Java programs obtaining non-alias relations:



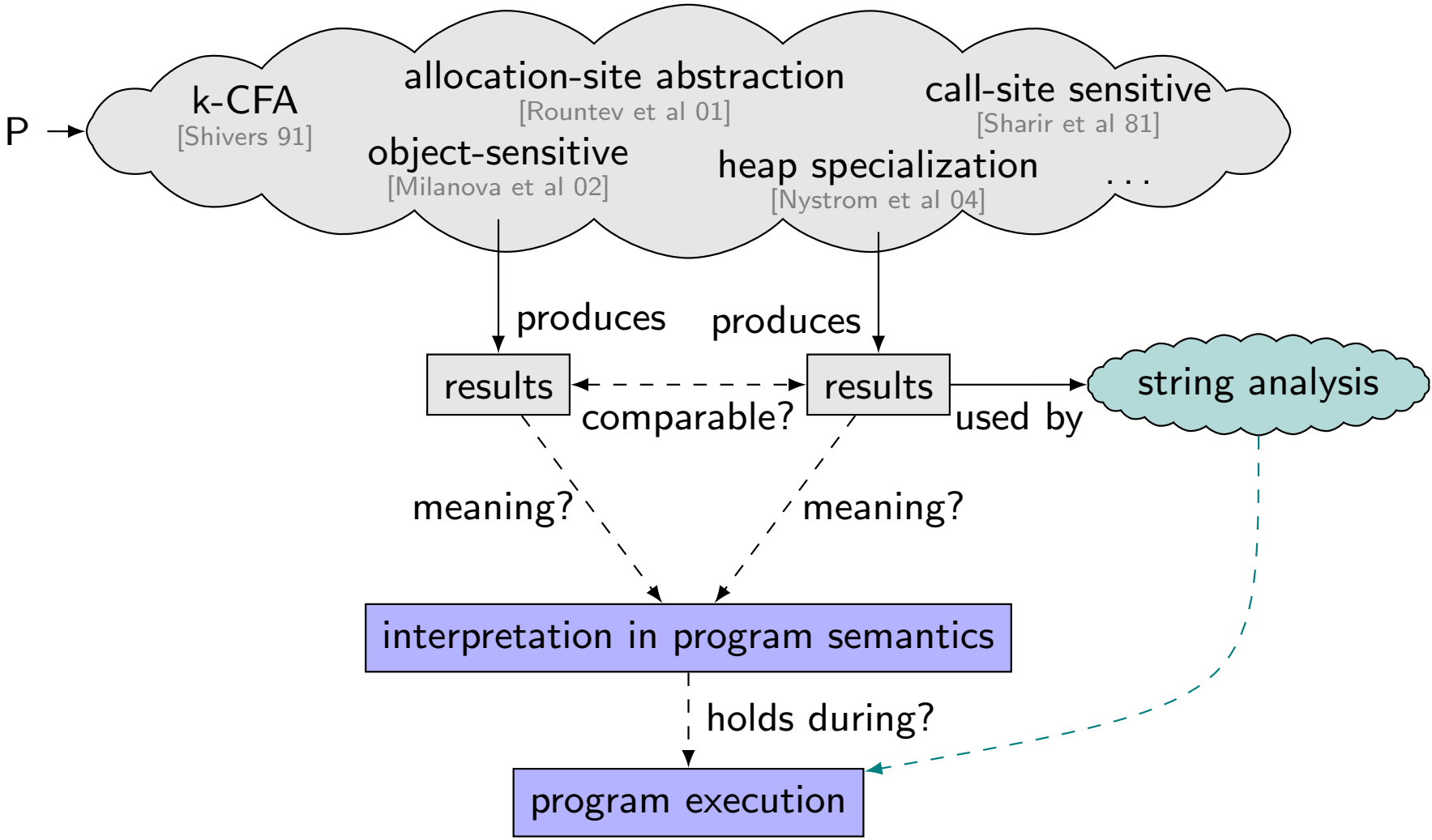
Pointer analysis

analysis algorithms for Java programs obtaining non-alias relations:

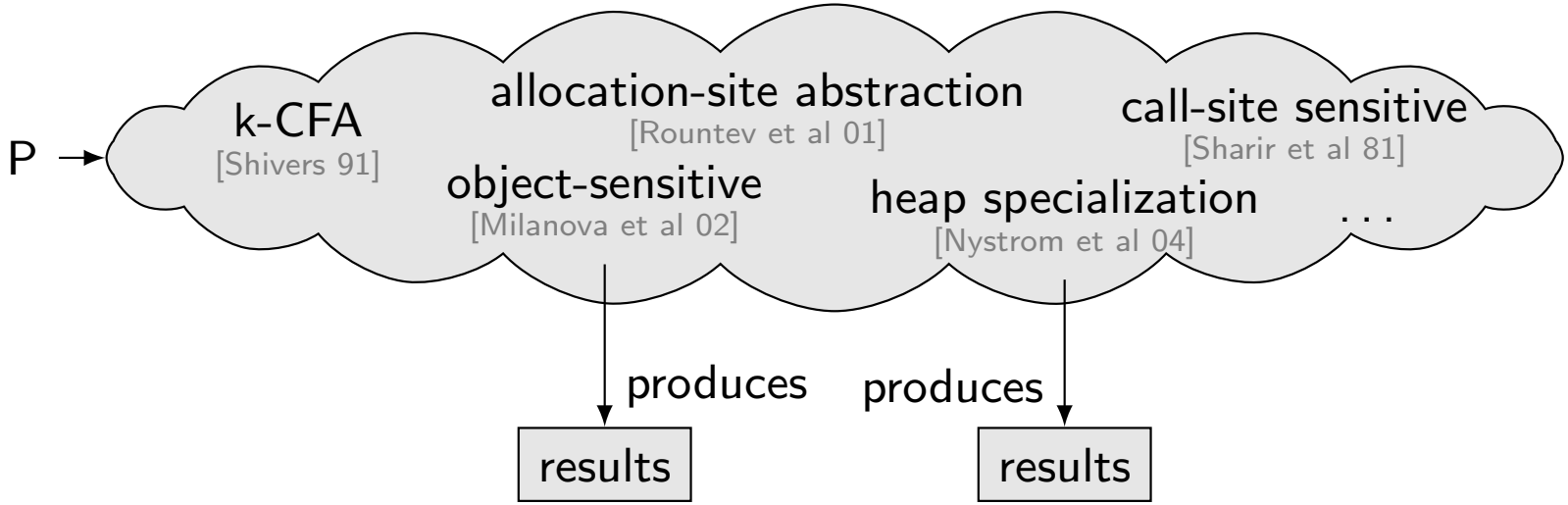


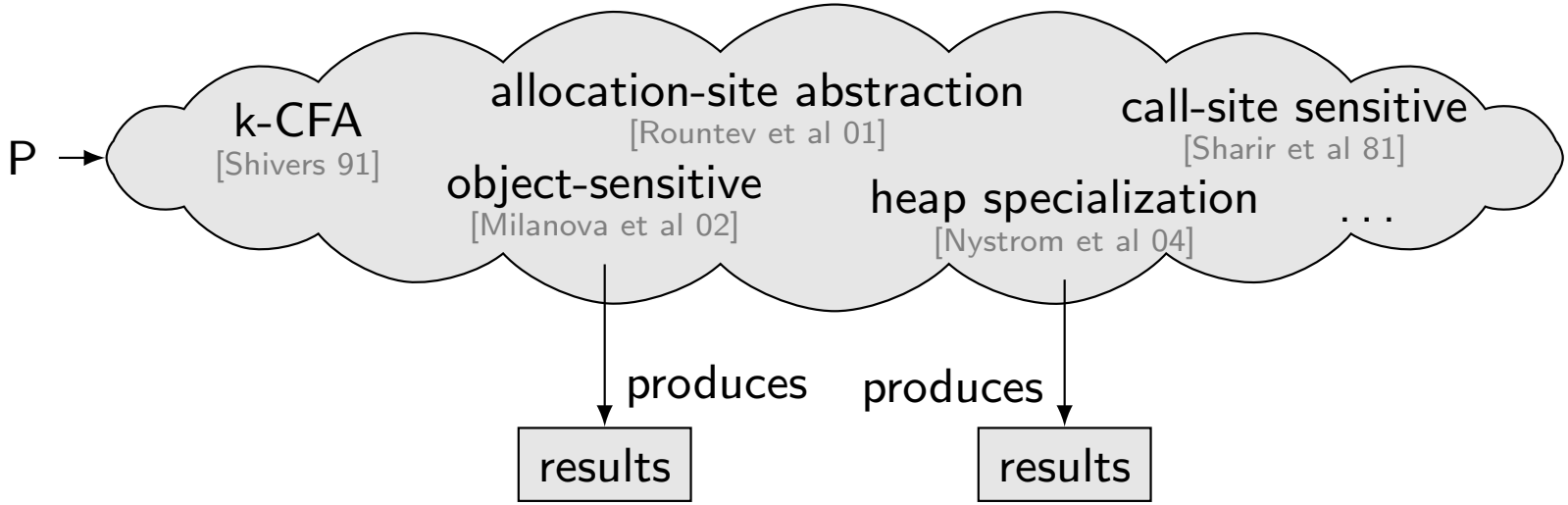
Pointer analysis

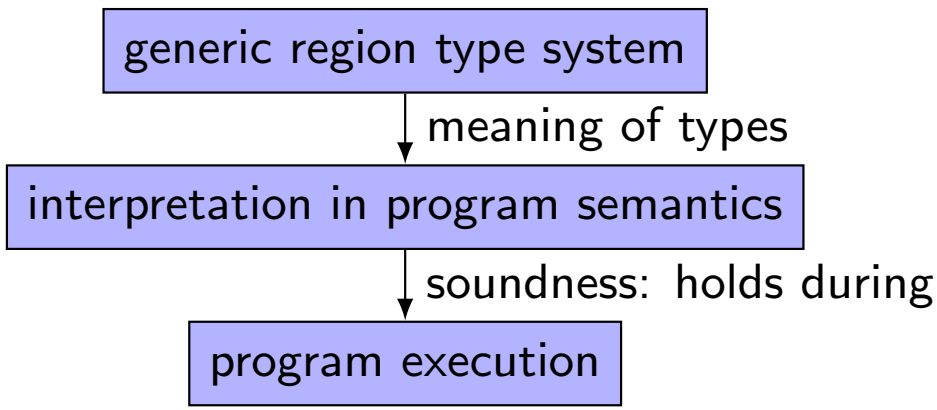
analysis algorithms for Java programs obtaining non-alias relations:

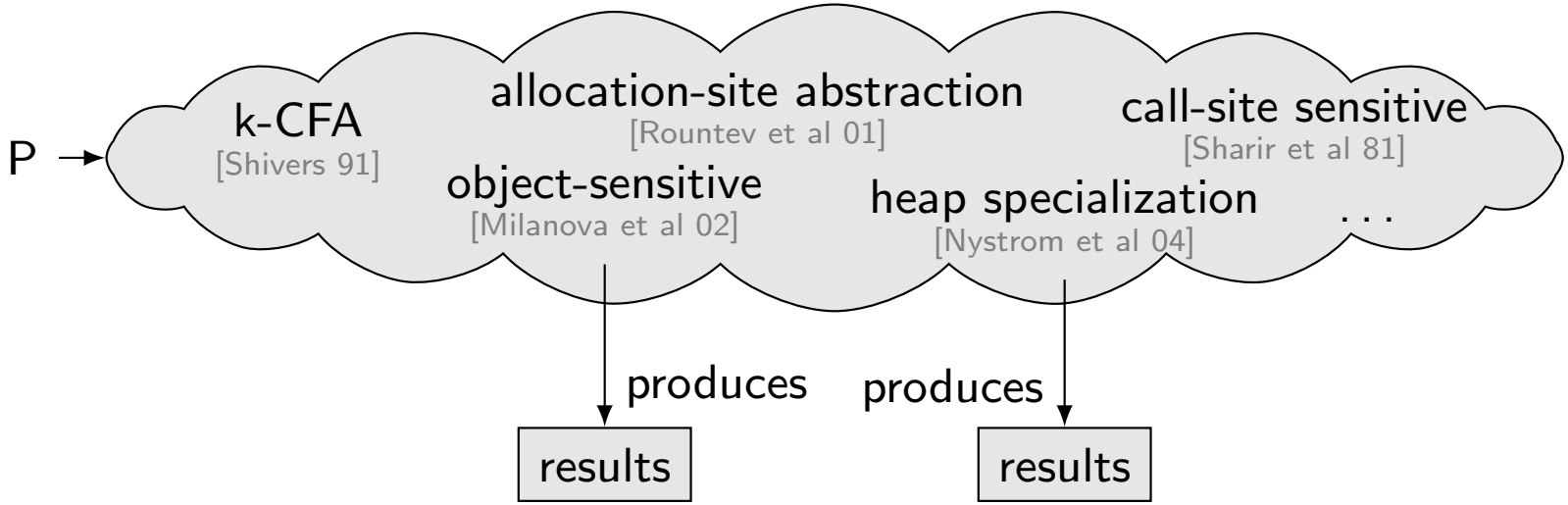


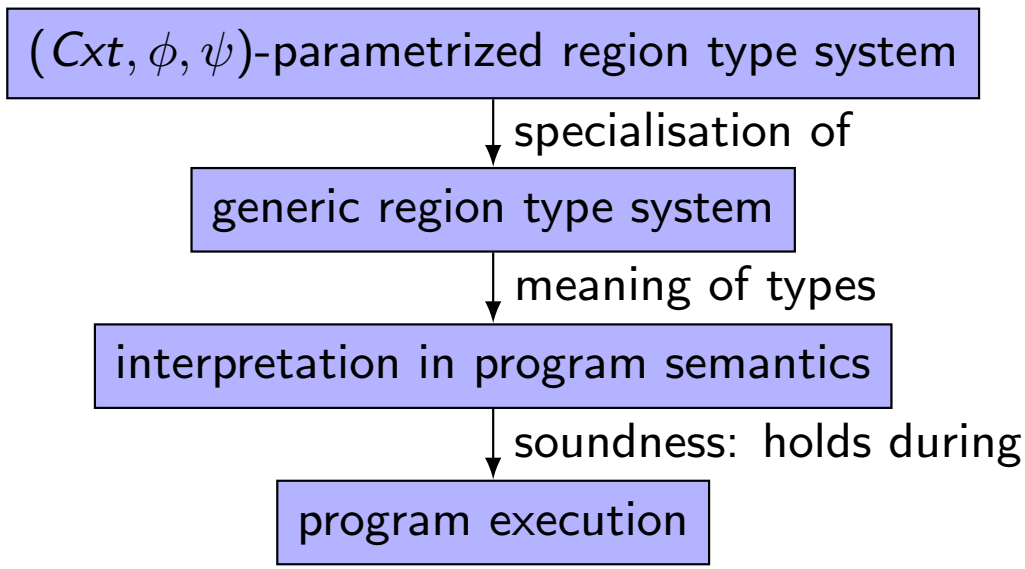
our goal: verify results of pointer analysis algorithms using region type system

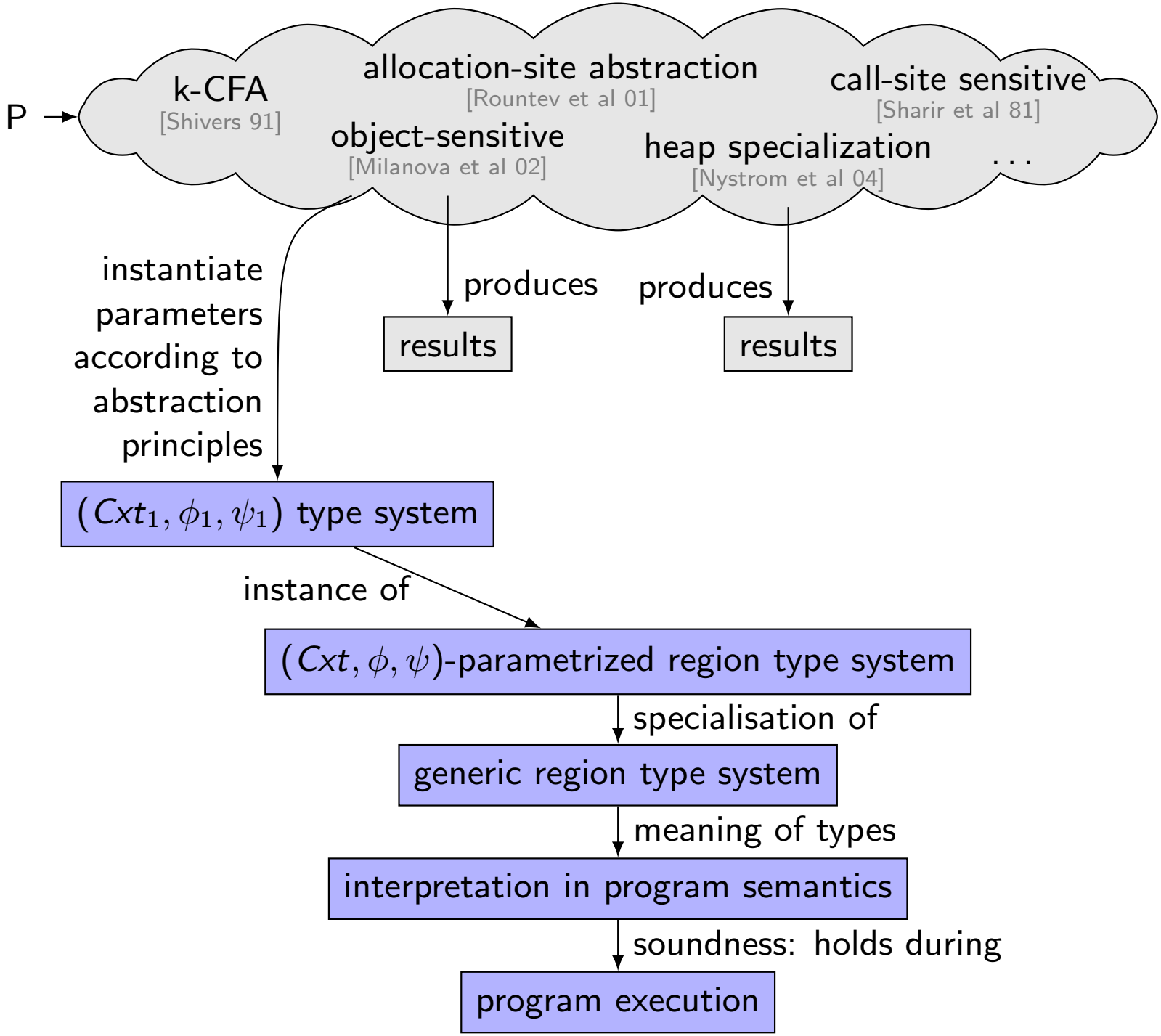

P
k-CFA
[Shivers 91]
allocation-site abstraction
[Rountev et al 01]
object-sensitive
[Milanova et al 02]
heap specialization
[Nystrom et al 04]
call-site sensitive
[Sharir et al 81]
. . .
produces
produces
results
results

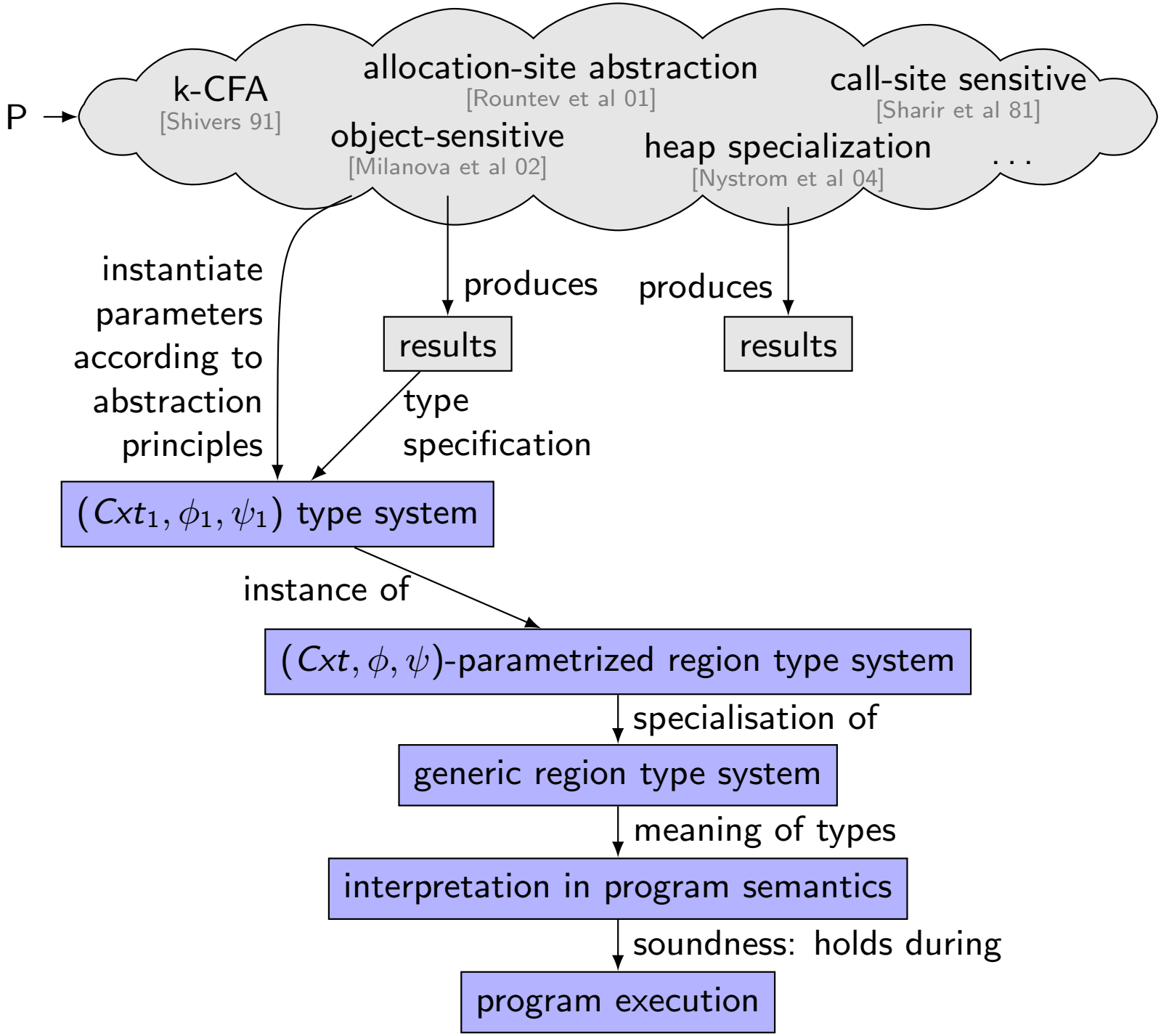

P
k-CFA
[Shivers 91]
allocation-site abstraction
[Rountev et al 01]
object-sensitive
[Milanova et al 02]
heap specialization
[Nystrom et al 04]
call-site sensitive
[Sharir et al 81]
. . .
produces
produces
results
results

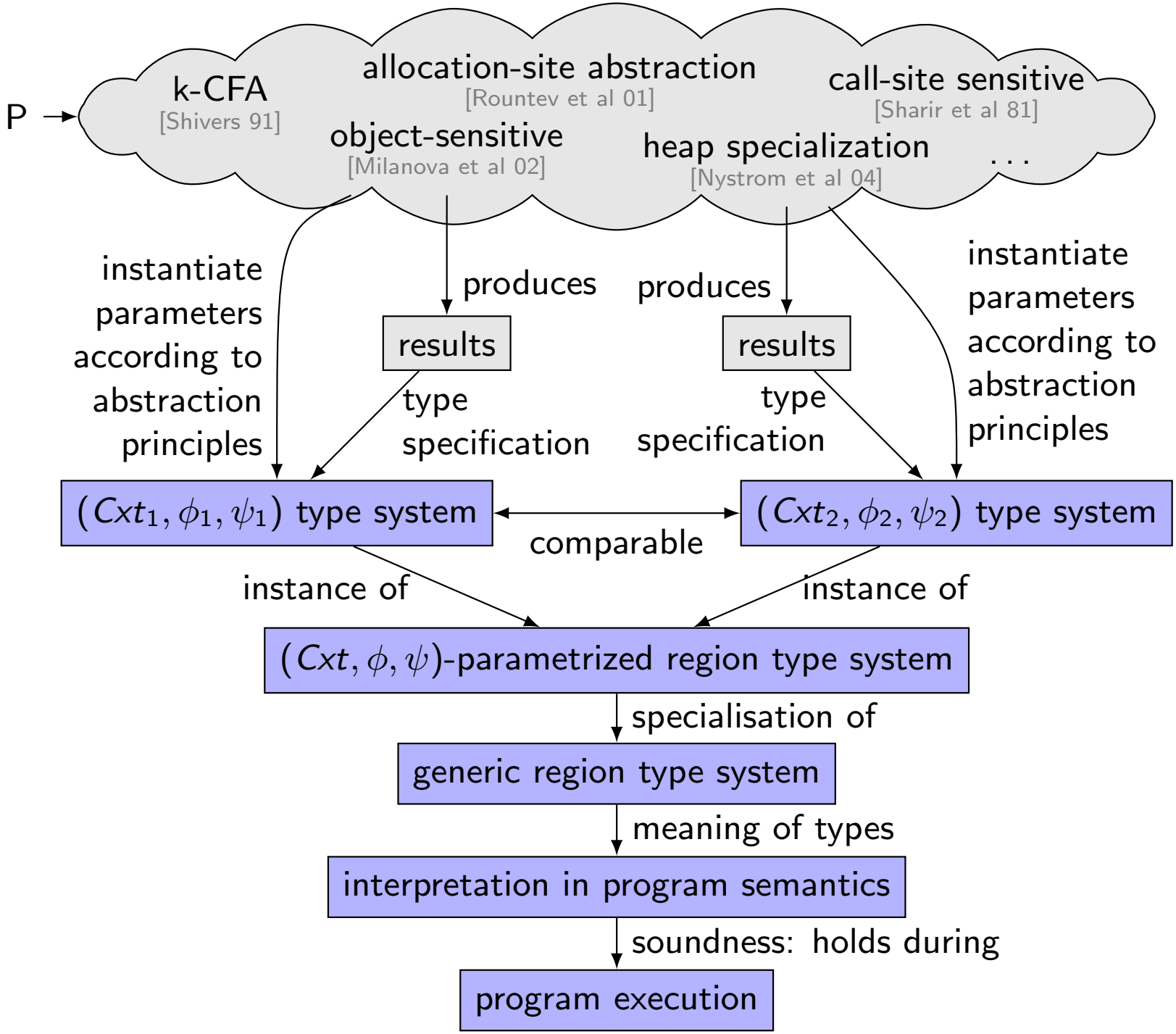

generic region type system
meaning of types
interpretation in program semantics
soundness: holds during
program execution

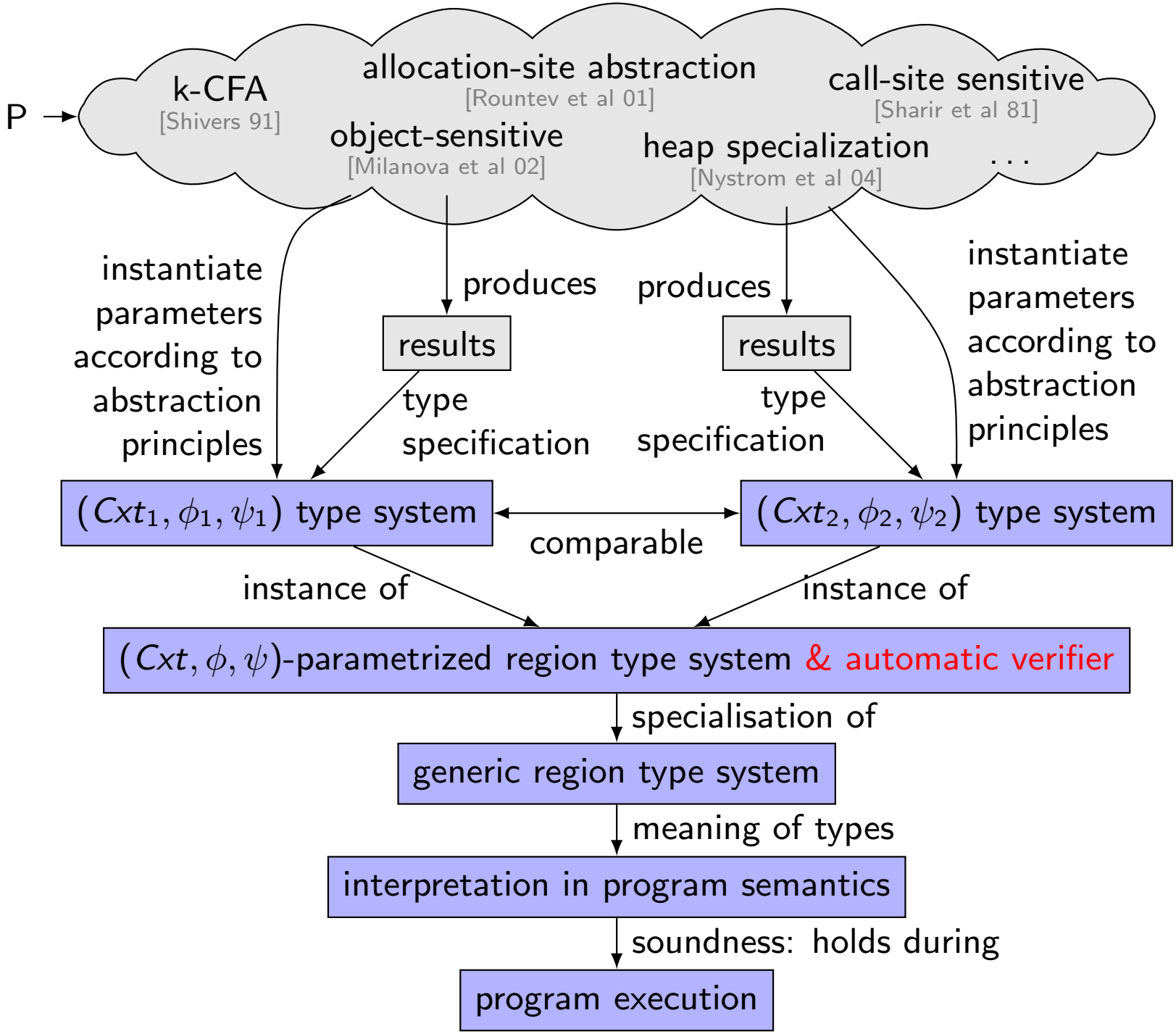

P
k-CFA
[Shivers 91]
allocation-site abstraction
[Rountev et al 01]
object-sensitive
[Milanova et al 02]
heap specialization
[Nystrom et al 04]
call-site sensitive
[Sharir et al 81]
. . .
produces
produces
results
results

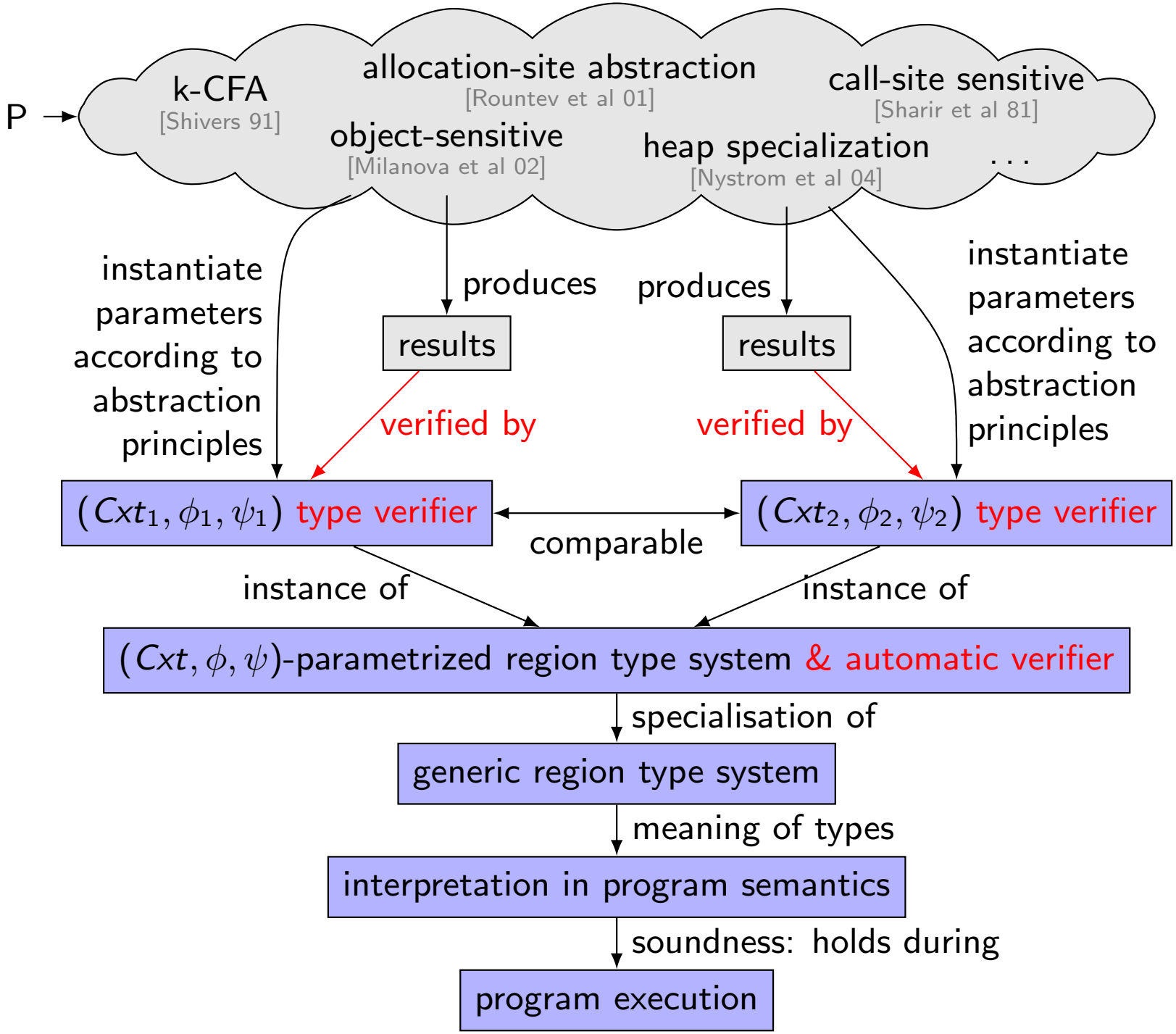

(Cxt, ϕ, ψ)-parametrized region type system
specialisation of
generic region type system
meaning of types
interpretation in program semantics
soundness: holds during
program execution

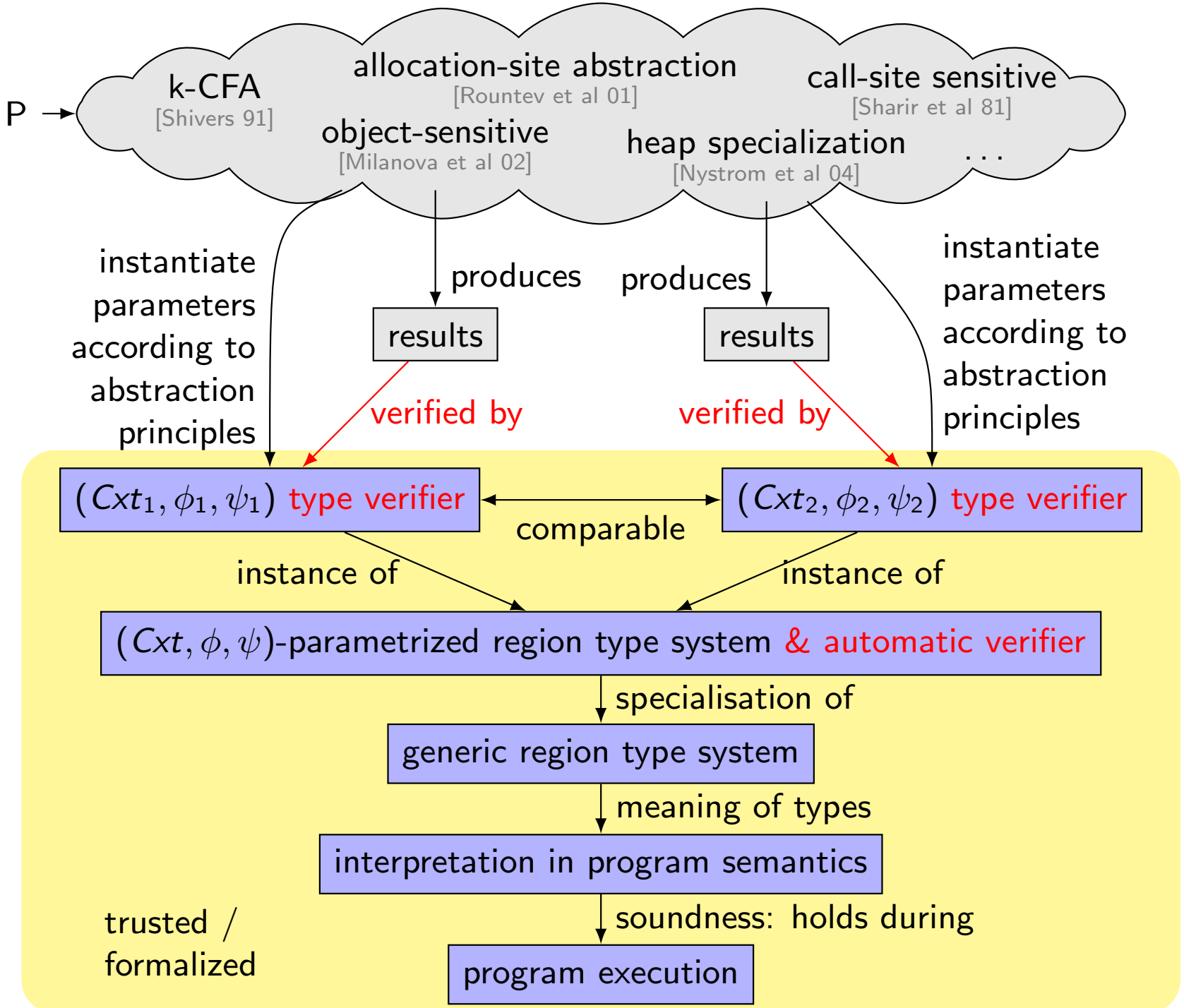

P
k-CFA
[Shivers 91]
allocation-site abstraction
[Rountev et al 01]
object-sensitive
[Milanova et al 02]
heap specialization
[Nystrom et al 04]
call-site sensitive
[Sharir et al 81]
. . .
instantiate parameters according to abstraction principles
produces
results
produces
results
(Cxt_1, ϕ_1, ψ_1) type system
instance of
(Cxt, ϕ, ψ)-parametrized region type system
specialisation of
generic region type system
meaning of types
interpretation in program semantics
soundness: holds during
program execution

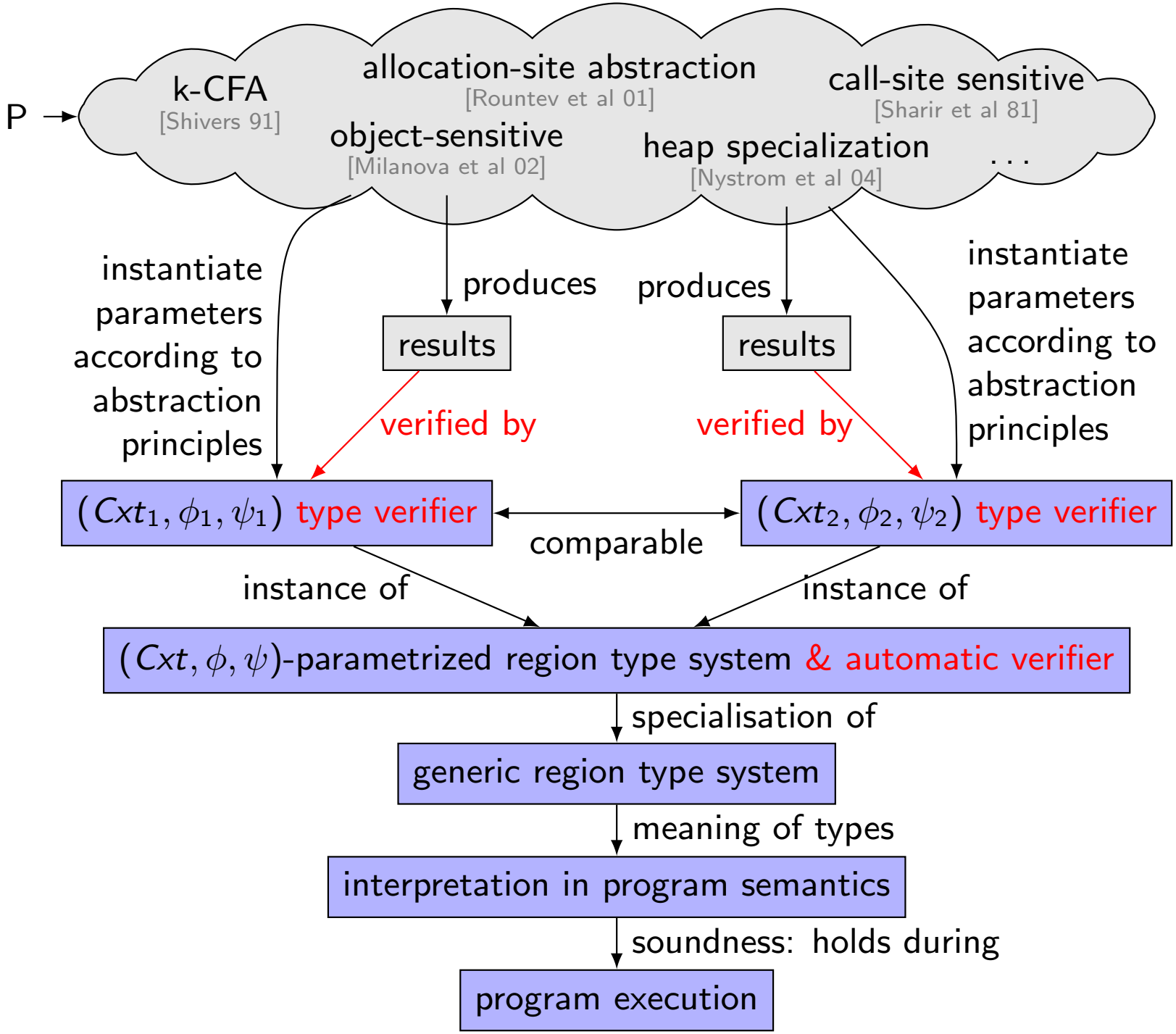

P
k-CFA
[Shivers 91]
allocation-site abstraction
[Rountev et al 01]
object-sensitive
[Milanova et al 02]
heap specialization
[Nystrom et al 04]
call-site sensitive
[Sharir et al 81]
. . .
instantiate parameters according to abstraction principles
produces
results
produces
results
type specification
(Cxt_1, ϕ_1, ψ_1) type system
instance of
(Cxt, ϕ, ψ)-parametrized region type system
specialisation of
generic region type system
meaning of types
interpretation in program semantics
soundness: holds during
program execution

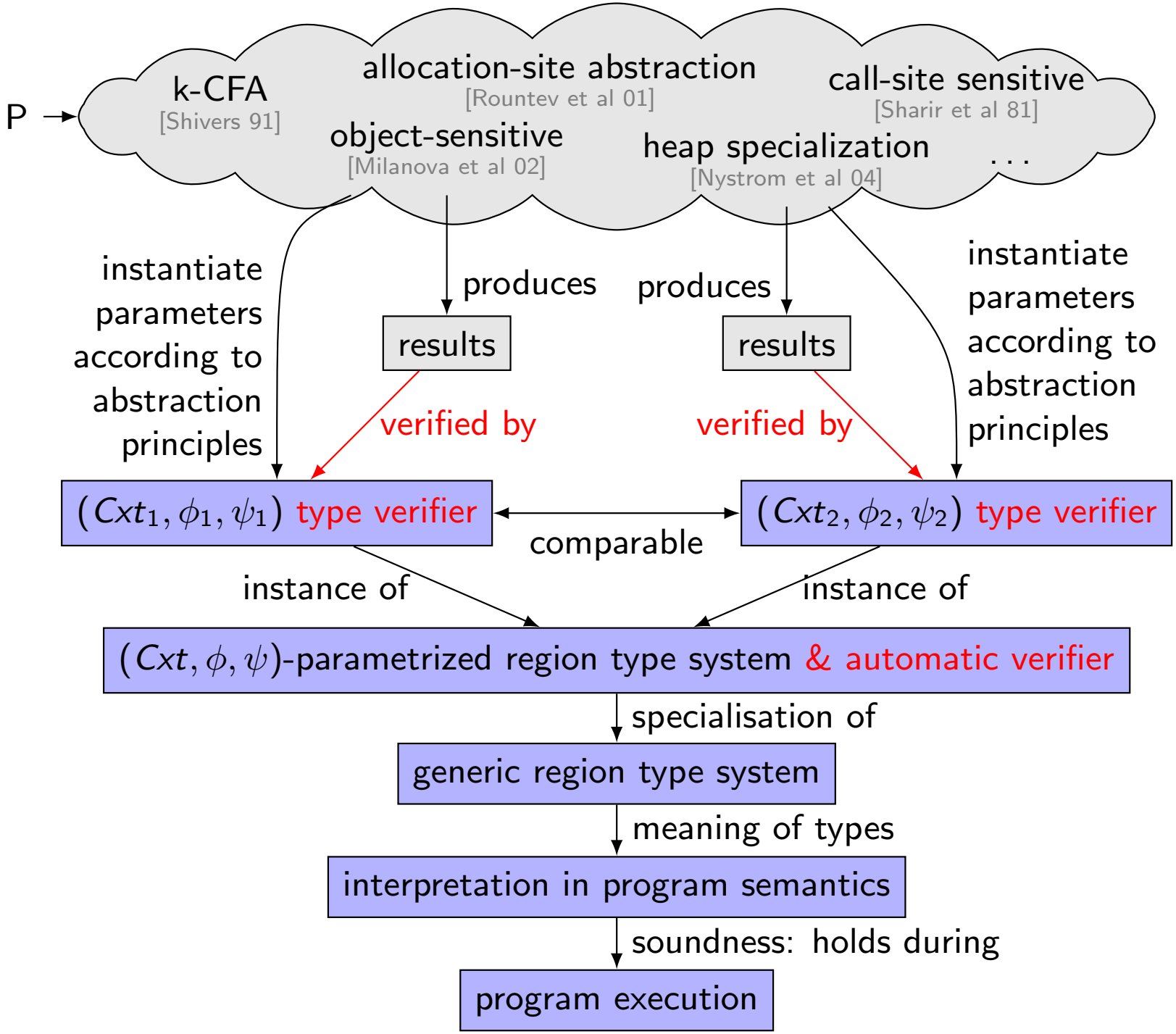

P
k-CFA
[Shivers 91]
allocation-site abstraction
[Rountev et al 01]
object-sensitive
[Milanova et al 02]
heap specialization
[Nystrom et al 04]
call-site sensitive
[Sharir et al 81]
. . .
instantiate parameters according to abstraction principles
produces
results
type specification
produces
results
type specification
instantiate parameters according to abstraction principles
(Cxt_1, ϕ_1, ψ_1) type system
comparable
(Cxt_2, ϕ_2, ψ_2) type system
instance of
instance of
(Cxt, ϕ, ψ)-parametrized region type system
specialisation of
generic region type system
meaning of types
interpretation in program semantics
soundness: holds during
program execution


P
k-CFA
[Shivers 91]
allocation-site abstraction
[Rountev et al 01]
object-sensitive
[Milanova et al 02]
heap specialization
[Nystrom et al 04]
call-site sensitive
[Sharir et al 81]
. . .
instantiate parameters according to abstraction principles
produces
results
type specification
produces
results
type specification
instantiate parameters according to abstraction principles
(Cxt_1, ϕ_1, ψ_1) type system
comparable
(Cxt_2, ϕ_2, ψ_2) type system
instance of
instance of
(Cxt, ϕ, ψ)-parametrized region type system & automatic verifier
specialisation of
generic region type system
meaning of types
interpretation in program semantics
soundness: holds during
program execution


P
k-CFA
[Shivers 91]
allocation-site abstraction
[Rountev et al 01]
object-sensitive
[Milanova et al 02]
heap specialization
[Nystrom et al 04]
call-site sensitive
[Sharir et al 81]
. . .
instantiate parameters according to abstraction principles
produces
results
verified by
produces
results
verified by
instantiate parameters according to abstraction principles
(Cxt_1, ϕ_1, ψ_1) type verifier
comparable
(Cxt_2, ϕ_2, ψ_2) type verifier
instance of
instance of
(Cxt, ϕ, ψ)-parametrized region type system & automatic verifier
specialisation of
generic region type system
meaning of types
interpretation in program semantics
soundness: holds during
program execution


P
k-CFA
[Shivers 91]
allocation-site abstraction
[Rountev et al 01]
object-sensitive
[Milanova et al 02]
heap specialization
[Nystrom et al 04]
call-site sensitive
[Sharir et al 81]
. . .
instantiate parameters according to abstraction principles
produces
results
verified by
produces
results
verified by
instantiate parameters according to abstraction principles
(Cxt_1, ϕ_1, ψ_1) type verifier
comparable
(Cxt_2, ϕ_2, ψ_2) type verifier
instance of
instance of
(Cxt, ϕ, ψ)-parametrized region type system & automatic verifier
specialisation of
generic region type system
meaning of types
interpretation in program semantics
soundness: holds during
program execution
trusted / formalized


P
k-CFA
[Shivers 91]
allocation-site abstraction
[Rountev et al 01]
object-sensitive
[Milanova et al 02]
heap specialization
[Nystrom et al 04]
call-site sensitive
[Sharir et al 81]
. . .
instantiate parameters according to abstraction principles
produces
results
verified by
produces
results
verified by
instantiate parameters according to abstraction principles
(Cxt_1, ϕ_1, ψ_1) type verifier
comparable
(Cxt_2, ϕ_2, ψ_2) type verifier
instance of
instance of
(Cxt, ϕ, ψ)-parametrized region type system & automatic verifier
specialisation of
generic region type system
meaning of types
interpretation in program semantics
soundness: holds during
program execution


P
k-CFA
[Shivers 91]
allocation-site abstraction
[Rountev et al 01]
object-sensitive
[Milanova et al 02]
heap specialization
[Nystrom et al 04]
call-site sensitive
[Sharir et al 81]
. . .
instantiate parameters according to abstraction principles
produces
results
verified by
produces
results
verified by
instantiate parameters according to abstraction principles
(Cxt_1, ϕ_1, ψ_1) type verifier
comparable
(Cxt_2, ϕ_2, ψ_2) type verifier
instance of
instance of
(Cxt, ϕ, ψ)-parametrized region type system & automatic verifier
specialisation of
generic region type system
meaning of types
interpretation in program semantics
soundness: holds during
program execution

Java class type system

formalization of program:

- sets of identifiers, field and method typing (types = classes)

$$
\begin{aligned}
\mathit{Cls} &= \{\texttt{List}, \texttt{Elem}\} \\
\mathit{Fld} &= \{\texttt{elem}, \texttt{next}\} \\
\mathit{Mtd} &= \{\texttt{copy}\}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{FType}(\texttt{List}, \texttt{elem}) &= \texttt{Elem} \\
\mathit{FType}(\texttt{List}, \texttt{next}) &= \texttt{List} \\
\mathit{MType}(\texttt{List}, \texttt{copy}) &= () \to \texttt{List}
\end{aligned}
$$

Java class type system

formalization of program:

- sets of identifiers, field and method typing (types = classes)

$$
\begin{aligned}
\mathit{Cls} &= \{\texttt{List}, \texttt{Elem}\} \\
\mathit{Fld} &= \{\texttt{elem}, \texttt{next}\} \\
\mathit{Mtd} &= \{\texttt{copy}\}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{FType}(\texttt{List}, \texttt{elem}) &= \texttt{Elem} \\
\mathit{FType}(\texttt{List}, \texttt{next}) &= \texttt{List} \\
\mathit{MType}(\texttt{List}, \texttt{copy}) &= () \to \texttt{List}
\end{aligned}
$$

- method implementations in *Featherweight Java Extended with Updates*

Java class type system

formalization of program:

- sets of identifiers, field and method typing (types = classes)

$$
\begin{aligned}
\mathit{Cls} &= \{\texttt{List}, \texttt{Elem}\} \\
\mathit{Fld} &= \{\texttt{elem}, \texttt{next}\} \\
\mathit{Mtd} &= \{\texttt{copy}\}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{FType}(\texttt{List}, \texttt{elem}) &= \texttt{Elem} \\
\mathit{FType}(\texttt{List}, \texttt{next}) &= \texttt{List} \\
\mathit{MType}(\texttt{List}, \texttt{copy}) &= () \to \texttt{List}
\end{aligned}
$$

- method implementations in *Featherweight Java Extended with Updates*

type system assigns judgements: $\Gamma \vdash e : C$

Region types

Region type system assigns a *refined* type:

$$\Gamma \vdash e : C_R$$

Region types

Region type system assigns a *refined* type:

$$\Gamma \vdash e : C_R$$

regions:

- intuition: disjoint classification of objects
- represent disjoint sets of possible concrete memory locations
- infinitely many regions $r \in Reg$ available
- in type: location is in one of regions in R
- different usage: scoping & garbage collection [Tofte et al. 97]

Region type system

annotated class specification:

- types = classes with region sets
- differentiate field and method types for same class by region
- have (possibly infinite) set of types for each method

$$
\begin{aligned}
\mathit{FType}(\texttt{List}, r, \texttt{elem}) &= \texttt{Elem}_{\{t\}} \\
\mathit{FType}(\texttt{List}, r, \texttt{next}) &= \texttt{List}_{\{r\}} \\
\mathit{MType}(\texttt{List}, r, \texttt{copy}) &= \{() \longrightarrow \texttt{List}_{\{s\}}, \\
&\qquad\qquad () \longrightarrow \texttt{List}_{\{r\}}\} \\
\\
\mathit{FType}(\texttt{List}, s, \texttt{elem}) &= \texttt{Elem}_{\{t\}} \\
\mathit{FType}(\texttt{List}, s, \texttt{next}) &= \texttt{List}_{\{s\}} \\
\mathit{MType}(\texttt{List}, s, \texttt{copy}) &= \{() \longrightarrow \texttt{List}_{\{s\}}\}
\end{aligned}
$$

Region type system

annotated class specification:

- types = classes with region sets
- differentiate field and method types for same class by region
- have (possibly infinite) set of types for each method

$$
\begin{aligned}
\mathit{FType}(\texttt{List}, r, \texttt{elem}) &= \texttt{Elem}_{\{t\}} \\
\mathit{FType}(\texttt{List}, r, \texttt{next}) &= \texttt{List}_{\{r\}} \\
\mathit{MType}(\texttt{List}, r, \texttt{copy}) &= \{() \longrightarrow \texttt{List}_{\{s\}}, \\
&\qquad\qquad () \longrightarrow \texttt{List}_{\{r\}}\} \\
\\
\mathit{FType}(\texttt{List}, s, \texttt{elem}) &= \texttt{Elem}_{\{t\}} \\
\mathit{FType}(\texttt{List}, s, \texttt{next}) &= \texttt{List}_{\{s\}} \\
\mathit{MType}(\texttt{List}, s, \texttt{copy}) &= \{() \longrightarrow \texttt{List}_{\{s\}}\}
\end{aligned}
$$

given *FType* and *MType*:

1. when are they legal?
2. how to determine it automatically?
3. relation to pointer analysis results?

Typing rules (extract)

Object creation: can pick an arbitrary region for new objects

$$\frac{}{\Gamma \vdash \texttt{new}\ C\ :\ C_{\{r\}}}$$

Typing rules (extract)

Object creation: can pick an arbitrary region for new objects

$$\frac{}{\Gamma \vdash \texttt{new}\ C\ :\ C_{\{r\}}}$$

Method call: for all regions that may contain x, there must be a suitable method type:

$$\frac{\forall r \in R.\ \exists(\overline{\tau_{arg}}', \tau'_{res}) \in MType(C, r, m).\ \overline{\tau_{arg}} <: \overline{\tau_{arg}}' \wedge \tau'_{res} <: \tau_{res}}{x : C_R, \overline{y} : \overline{\tau_{arg}} \vdash x.m(\overline{y})\ :\ \tau_{res}}$$

Typing rules (extract)

Object creation: can pick an arbitrary region for new objects

$$\frac{}{\Gamma \vdash \texttt{new}\ C\ :\ C_{\{r\}}}$$

Method call: for all regions that may contain x, there must be a suitable method type:

$$\frac{\forall r \in R.\ \exists(\overline{\tau_{arg}}', \tau'_{res}) \in \mathit{MType}(C, r, m).\ \overline{\tau_{arg}} <: \overline{\tau_{arg}}' \wedge \tau'_{res} <: \tau_{res}}{x : C_R, \overline{y} : \overline{\tau_{arg}} \vdash x.m(\overline{y})\ :\ \tau_{res}}$$

Well-typed program: for all announced method types $\mathit{MType}(C, r, m)$, a corresponding type for the method body can be derived.

Soundness

Given annotated tables $FType$ and $MType$. If

- $\Gamma \vdash e_1 : C_R$ and
- $\Gamma \vdash e_2 : D_S$ and
- $R \cap S = \emptyset$

then

- e_1 and e_2 do not alias, i.e. are different objects

Soundness

Given annotated tables *FType* and *MType*. If

- $\Gamma \vdash e_1 : C_R$ and
- $\Gamma \vdash e_2 : D_S$ and
- $R \cap S = \emptyset$

then

- e_1 and e_2 do not alias, i.e. are different objects
  - more precisely: there is a mapping from locations to regions

Soundness

Given annotated tables *FType* and *MType*. If

- $\Gamma \vdash e_1 : C_R$ and
- $\Gamma \vdash e_2 : D_S$ and
- $R \cap S = \emptyset$

then

- e_1 and e_2 do not alias, i.e. are different objects
  - more precisely: there is a mapping from locations to regions
  - really precise: Isabelle formalization of proofs

Towards verification of pointer analysis results

2. how to determine type derivation automatically?
3. relation to pointer analysis results?

Solution: parametrized version of the type system
Obeys principles used by algorithm with respect to

- label choice at object creation
- behavior at call sites

Abstraction principles

Call-graph abstraction

Context-sensitive pointer analysis distinguishes different calls to a method via its calling *context*, which parametrizes obtained pointer information of each method:

- invocation site
- region of receiver object
- (finite) stack of these

Object abstraction

Variations exist how labels are chosen to distinguish objects:

- object allocation site
- context during allocation
- combination of both

Parametrized type system

We capture these principles by four parameters PP, Cxt, ψ and ϕ.

$PP \ni i$ is a distinct labeling of program points:

$$e^i$$

$Cxt \ni z$ is a finite set of contexts. They are used to parametrize typing judgements:

$$\Gamma\,;\, z \;\vdash\; e^i \;:\; \tau$$

The function ψ models object abstraction:

$$\psi : Cxt \times PP \to Reg$$

It determines the region of the object in the rule for `new` C.

Parametrized type system

We assign just one method type $MType(C, r, m, z)$ per $z \in \mathcal{Z}$:

$$MType : Cls \times Reg \times Mtd \times Cxt \rightharpoonup \overline{Typ} \times Typ$$

The transfer function ϕ models context switch:

$$\phi : Cxt \times Cls \times Reg \times Mtd \times PP \to Cxt$$

The function selects the method type in the rule for $x.m(\overline{y})$.

Instantiation of parameters

Reg=	*object abstraction*	*principle*
PP	$\psi(z, i) = i$	allocation site abstraction
$PP \times PP = Cxt$	$\psi((i_1, i_2), i_0) = (i_0, i_1)$	object-sensitive allocation site abstraction
$PP = Cxt$	$\psi(i_c, i) = i_c$	heap specialization

Cxt=	*context transfer*	*principle*
$\{z_0\}$	$\phi(z, C, r, m, i) = z_0$	context-insensitivity
Reg	$\phi(z, C, r, m, i) = r$	object-sensitive 1-CFA
$\bigcup_{n\in\{1,\ldots,k\}} Mtd^n$	$\phi(z, C, r, m, i)$ $= (m :: z)\|_k$	method identifier k-CFA

Instantiation of parameters

Reg=	*object abstraction*	*principle*
PP	$\psi(z, i) = i$	allocation site abstraction
$PP \times PP = Cxt$	$\psi((i_1, i_2), i_0) = (i_0, i_1)$	object-sensitive allocation site abstraction
$PP = Cxt$	$\psi(i_c, i) = i_c$	heap specialization

Cxt=	*context transfer*	*principle*
$\{z_0\}$	$\phi(z, C, r, m, i) = z_0$	context-insensitivity
Reg	$\phi(z, C, r, m, i) = r$	object-sensitive 1-CFA
$\bigcup_{n\in\{1,\ldots,k\}} Mtd^n$	$\phi(z, C, r, m, i)$ $= (m :: z)\|_k$	method identifier k-CFA

stack restricted to k entries

Soundness and automatic verification

Soundness

parametrized type system is more restrictive than generic region type system

- bounded number of regions and method types
- ψ and ϕ constrain choice of region and method type
- also simplifies finding type derivation

Automatic verification

in the paper: syntax-directed algorithmic version

- derivable in parametrized system
- suggests an type checker
- results of pointer analysis can be automatically verified once abstraction principles have been suitably transformed into parameter instantiations

Verifying string analysis

String analysis:

- determine possible values of string variables
- used to enforce a security policy, i.e. prevention of code injection and cross-site scripting attacks

Can be built on top of pointer analysis [Cregut et al. '05]:

- use regions to track objects of class *String*
- produce string value assertions in terms of regions, e.g. "all strings in region r only contain numbers"

We have extended language and type system for string analysis:

- extensional formalization of grammar-based string analysis, similar to Java String Analyzer [Christensen et al. '03]

Summary

Contribution:

- framework for semantic interpretation of results of common pointer analysis algorithms
- region type system to declaratively describe pointer assertions
- transformation into algorithmic version that enables embedding and automatic verification of results of a specific analysis

Future work:

- more powerful type system: flow-/path-sensitivity
- sound enforcement of string-based security policies and "best practices" using type systems

Backup slide: Featherweight Java Extended with Updates

- FJEU [Hofmann et al. '06]; based on FJ [Igarashi et al. '99]
- idealized object-oriented language, models Java's core

Syntax

$$\begin{array}{rcl} e & ::= & \texttt{null} \mid x \mid \texttt{let}\ x = e_1\ \texttt{in}\ e_2 \mid \texttt{if}\ x = y\ \texttt{then}\ e_1\ \texttt{else}\ e_2 \mid \\ & & \texttt{new}\ C \mid x.f \mid x.f := y \mid x.m(\overline{y}) \end{array}$$

A program consists of a class hierarchy $(\mathit{Cls}, \preceq)$, for each class $C \in \mathit{Cls}$ field names $f \in \mathit{Fld}$ and method names $m \in \mathit{Mtd}$, and for each method a body e.

$$\begin{array}{rcl} \mathit{Cls} & = & \{\texttt{List}, \texttt{Elem}\} \\ \mathit{fields}(\texttt{List}) & = & \{\texttt{elem}, \texttt{next}\} \\ \mathit{methods}(\texttt{List}) & = & \{\texttt{copy}\} \end{array}$$

$$\begin{array}{l} \mathit{mbody}(\texttt{List}, \texttt{copy}) = \\ \quad \texttt{let l} = \texttt{new List in} \\ \quad \texttt{let \_} = \texttt{l.elem:=elem in} \\ \quad \texttt{let \_} = \texttt{l.next:=next.copy() in l} \end{array}$$

Operational semantics

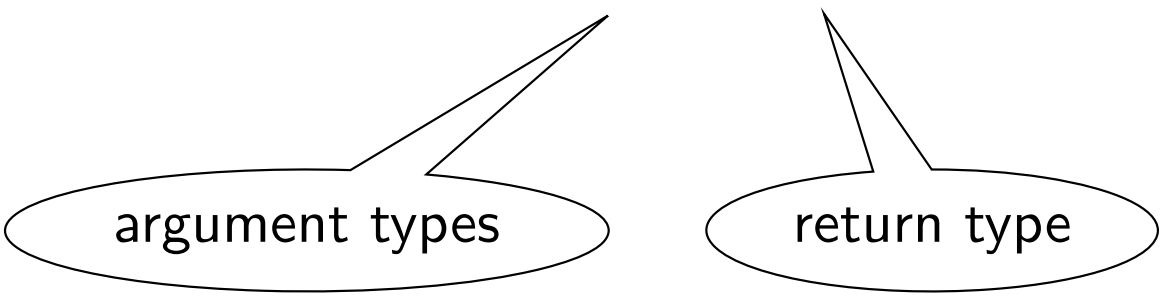
$$(s, h) \vdash e \Downarrow v, h'$$

Backup slide: Region types

$$
\begin{array}{rrcl}
\text{region:} & r, s, t, \ldots & \in & \mathit{Reg} \\
\text{sets of region:} & R, S, T, \ldots & \in & \mathfrak{P}(\mathit{Reg}) \\
\text{types:} & \tau & \in & \mathit{Typ} = \mathit{Cls} \times \mathfrak{P}(\mathit{Reg})
\end{array}
$$

For a type (C, R), we write C_R.

$$
\begin{aligned}
\Gamma &: \mathit{Var} \rightharpoonup \mathit{Typ} \\
\mathit{FType} &: \mathit{Cls} \times \mathit{Reg} \times \mathit{Fld} \rightharpoonup \mathit{Typ} \\
\mathit{MType} &: \mathit{Cls} \times \mathit{Reg} \times \mathit{Mtd} \rightharpoonup \mathfrak{P}(\overline{\mathit{Typ}} \times \mathit{Typ})
\end{aligned}
$$



Backup slide: Region type system

$$\frac{\Gamma \vdash e : \sigma \qquad \sigma <: \tau}{\Gamma \vdash e : \tau} \qquad \frac{\Gamma \vdash e_1 : \sigma \qquad \Gamma, x : \sigma \vdash e_2 : \tau}{\Gamma \vdash \texttt{let}\ x = e_1\ \texttt{in}\ e_2 : \tau}$$

$$\frac{}{\Gamma \vdash \texttt{new}\ C : C_{\{r\}}} \qquad \frac{}{\Gamma, x : \tau \vdash x : \tau} \qquad \frac{}{\Gamma \vdash \texttt{null} : \tau}$$

$$\frac{\Gamma, x : C_{R \cap S}, y : D_{R \cap S} \vdash e_1 : \tau \qquad \Gamma, x : C_R, y : D_S \vdash e_2 : \tau}{\Gamma, x : C_R, y : D_S \vdash \texttt{if}\ x = y\ \texttt{then}\ e_1\ \texttt{else}\ e_2 : \tau}$$

$$\frac{\forall r \in R.\ \exists (\overline{\sigma'}, \tau') \in \mathit{MType}(C, r, m).\ \overline{\sigma} <: \overline{\sigma}' \wedge \tau' <: \tau}{\Gamma, x : C_R, \overline{y} : \overline{\sigma} \vdash x.m(\overline{y}) : \tau}$$

$$\frac{\forall r \in R.\ \mathit{FType}(C, r, f) <: \tau}{\Gamma, x : C_R \vdash x.f : \tau} \qquad \frac{\forall r \in R.\ \tau <: \mathit{FType}(C, r, f)}{\Gamma, x : C_R, y : \tau \vdash x.f := y : \tau}$$

Backup slide: Soundness result

$$\text{Locations:} \quad l \in Loc$$
$$\text{Heap typings:} \quad \Sigma \in Loc \rightharpoonup Reg \times Cls$$

$$\frac{}{\Sigma \vdash \texttt{null} : \tau} \qquad \frac{\Sigma(l) = (C, r)}{\Sigma \vdash l : C_{\{r\}}} \qquad \frac{\Sigma \vdash v : \sigma \qquad \sigma <: \tau}{\Sigma \vdash v : \tau}$$

$$\Sigma \vdash s : \Gamma \iff \forall x \in dom(\Gamma).\ \Sigma \vdash s(x) : \Gamma(x)$$
$$h \models \Sigma \iff \forall l \in dom(\Sigma).\ l \in dom(h) \wedge \Sigma \models h(l) : \Sigma(l)$$

where

$$\Sigma \models (C, F) : (D, r) \iff C \preceq D \wedge dom(F) = fields(C) \wedge \forall f \in fields(C).\ \Sigma \vdash F(f) : FType(C, r, f)$$

Soundness Theorem

Fix a well-typed program P. For all $\Sigma, \Sigma', \tau, s, h, e, v, h'$ with $\Gamma \vdash e : \tau$ and $\Sigma \vdash s : \Gamma$ and $(s, h) \vdash e \Downarrow v, h'$ and $h \models \Sigma$ there exists some $\Sigma' \supseteq \Sigma$ such that $\Sigma' \vdash v : \tau$ and $h' \models \Sigma'$.

Backup slide: String analysis

Extended syntax

$$\begin{array}{rcl} w & \in & \Sigma^* \\ e & ::= & \ldots \mid \texttt{new String}(w) \mid x.concat(y) \end{array}$$

String operation contexts

Ω: set over string operations:

$$\omega \quad ::= \quad \texttt{newstr}\ w\ r \mid \texttt{concat}\ r\ s\ t \mid \texttt{unknown}\ W\ r$$

Types: C_R or $\texttt{String}_R$

$$\frac{\texttt{newstr}\ w\ r \in \Omega}{\Gamma \vdash \texttt{new String}(w) : \texttt{String}_{\{r\}}}$$

$$\frac{\forall r \in R, s \in S.\ \texttt{concat}\ r\ s\ t \in \Omega}{\Gamma, x : \texttt{String}_R, y : \texttt{String}_S \vdash x.concat(y) : \texttt{String}_{\{t\}}}$$