Principles of Abstract Interpretation
MIT press

Ch. 46, Points-To Analysis

Patrick Cousot

pcousot.github.io

PrAbsInt@gmail.com   github.com/PrAbsInt/

Chapter 40

Ch. 46, Points-To Analysis

Pointers in C (not considering `malloc`)

Pointers in C

- We consider static memory allocation where the memory addresses (where pointer variables of a program can point to) are determined statically by the compiler (as opposed to dynamic memory allocation by malloc in C [**DBLP:books/ph/KernighanR88**] during program execution).

Example 46.1

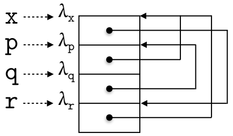
```
#include <stdio.h>
#include <stdlib.h>
int main () {
   int x,*p,**q,*r;
   x = &p;
   p = &x;
   q = &p;
   r = *q;
   *r = &r;
}
```

Where is p pointing to on program exit?

Example 46.1

The points-to analysis of the following C program

```
#include <stdio.h>
#include <stdlib.h>
int main () {
   int x,*p,**q,*r;
   x = &p;
   p = &x;
   q = &p;
   r = *q;
   *r = &r;
}
```



Memory μ:

$\mu(\lambda_x)$	λ_r
$\mu(\lambda_p)$	λ_x
$\mu(\lambda_q)$	λ_p
$\mu(\lambda_r)$	λ_x

should be able to determine that p points to r on program exit.

Pointer or points-to analysis

- Pointer analysis, or points-to analysis consists in automatically inferring where pointer variables of a program can point to.
- Points-to analysis started at the end of the seventies [**DBLP:journals/cacm/Barth77**; **DBLP:conf/popl/Weihl80**] and has been extensively studied since then [**DBLP:conf/lfp/Hudak86**; **DBLP:conf/pepm/Deutsch95**; **DBLP:conf/paste/Hind01**; **DBLP:journals/csur/KanvarK16**].
- Our objective is to explore the formal design of two classical points-to analyzes [**Andersen94-Thesis**; **DBLP:conf/popl/Steensgaard96**] as opposed to the resolution of postulated constraints [**Andersen94-Thesis**] or a postulated type inference system [**DBLP:conf/popl/Steensgaard96**].

Syntax of the pointer language, section 46.1

Pointer syntax

The syntax of programs of section 4.1 is extended as follows.

$\mathtt{p}, \mathtt{q}, \ldots, \mathtt{x}, \mathtt{y}, \ldots$	$\in$	$\mathbb{V}$	variables
$\mathtt{p}, \mathtt{q}, \ldots$	$\in$	$\vec{\mathbb{V}} \subseteq \mathbb{V}$	pointer variables
$\mathtt{A}$	$::=$	$\ldots$	arithmetic expressions
	$\mid$	$\mathtt{x}$	variable
	$\mid$	$\mathtt{NULL}$	null pointer
	$\mid$	$\&\mathtt{x}$	address of variable
	$\mid$	$\star\mathtt{p}$	dereferenced pointer variable
$\mathtt{B}$	$::=$	$\ldots$	boolean expressions
	$\mid$	$\mathtt{x} == \mathtt{y}$	equality
	$\mid$	$\mathtt{p} == \mathtt{NULL}$	pointer equality to $\mathtt{nil}$
$\mathtt{S}$	$::=$	$\ldots$	statement
	$\mid$	$\mathtt{x} = \mathtt{A}\,;$	assignment
	$\mid$	$\star\mathtt{p} = \mathtt{A}\,;$	assignment to a dereferenced pointer

Comments on the pointer syntax

- Variables are statically allocated in memory at locations/addresses determined by the compiler and loader. These locations can be values of pointer variables.
- The null pointer `NULL` designates no location at all.
- `&x` is the location of variable `x` in memory.
- `*p` dereferences pointer variable `p`. It is the location contained in the location of variable `p`.
- One can test for nullity `p == NULL` or `x == y` equality of pointer variables.
- Assignment is of an integer or pointer value to a variable location `x = A ;` or an indirect assignment `*p = A ;` to a dereferenced pointer variable.
- Some languages like Pascal [**DBLP:journals/acta/Wirth71**] do not have such primitives but use them implicitly e.g. when passing parameters by reference (also called by variable) or for array parameters (to avoid a copy of the array).

Semantics of the pointer language

Pointer semantic domains, section 46.2

Let us define

z	$\in$	$\mathbb{Z}$	integers
l	$\in$	$\mathbb{L}$	enumerable set of locations (addresses) ($\mathbb{L} \cap \mathbb{Z} = \emptyset$)
λ	$\in$	$\mathbb{L}_{\mathbb{V}} \triangleq \mathbb{V} \rightarrowtail \mathbb{L}$	locations of variables (λ injective: unique location $\lambda_{\mathtt{x}}$ of variable x in memory)
$\mathtt{nil}$	$\notin$	$\mathbb{L}$	null pointer (value denoted by `NULL`)
$\nu \in \mathbb{V}$	$\triangleq$	$\mathbb{Z} \cup \mathbb{L} \cup \{\mathtt{nil}\}$	values
Ω	$\notin$	$\mathbb{V}$	error
$\nu \in \mathbb{V}_{\Omega}$	$\triangleq$	$\mathbb{V} \cup \{\Omega\}$	values or error

Pointer semantic domains, section 46.2

- For a given program, the locations λ are fixed by the compiler, linker and loader.
- So we can restrict values to these specific locations.

$l \in \mathbb{L}_\lambda \triangleq \{\lambda_{\mathtt{x}} \in \mathbb{L} \mid \mathtt{x} \in \mathbb{V}\}$	set of locations of all variables
$\mu \in \mathbb{M}_\lambda \triangleq \mathbb{L}_\lambda \rightarrow \mathbb{V}$	memories (values of locations)
$\nu \in \mathbb{V}_\lambda \triangleq \mathbb{Z} \cup \mathbb{L}_\lambda \cup \{\mathtt{nil}\}$	values
$P \in \mathbb{P}^{\mathbb{M}}_\lambda \triangleq \wp(\mathbb{M}_\lambda)$	memory properties

TODO: A ervoir

Comments on the pointer semantic domains

- The compiler and loader initially assign a unique location $\lambda_{\mathtt{x}}$ to each program variable $\mathtt{x}$
- The memory allocation function $\boldsymbol{\lambda} \triangleq \mathtt{x} \in \mathbb{V} \mapsto \lambda_{\mathtt{x}} \in \mathbb{L}$ is therefore bijective.
- Values returned by expressions are
  - integers,
  - locations,
  - the null pointer `nil` (denoted `NULL`), or
  - an error Ω (e.g. when dereferencing a null pointer which in practice often stops the execution with a segmentation fault (raised after a memory access violation in hardware with memory protection) or bus error (e.g.in misaligned memory access that hardware cannot access).
- A memory μ maps locations $\lambda_{\mathtt{x}}$ of variables $\mathtt{x} \in \mathbb{V}$ to their content $\mu(\lambda_{\mathtt{x}})$ which is the value of $\mathtt{x}$.

Comments on the pointer semantic domains

- In case of error, execution is assumed to stop immediately.
- This is contrary to some programming languages such as C [**DBLP:books/ph/KernighanR88**] where the program behavior is undefined in case of error.
- In C, errors need not be signaled during execution so the undefined execution may have unpredictable effects like erasing data or changing/destroying the program code.
- Of course it is impossible to define the semantics of such programs.
- So, more restrictively, we consider such undefined behaviors to be an error which immediately stops execution.
- The soundness of the analysis is precisely defined in section 46.12.

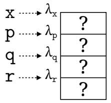
Invariant properties

- Invariant properties are sets of memories (but not an error since execution stops when the evaluation of a pointer expression is erroneous).

Reachability semantics of the pointer language, section 46.4

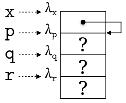
- We should extend the trace semantics and then abstract it into the reachability semantics of 19
- For brevity, we directly and postulate the reachability semantics $\widehat{\mathcal{S}}^{\vec{\varrho}}$ of the pointer manipulating statements.
- Of course it could be obtained by abstraction as shown in chapter 20, “Calculational Design of the Forward Reachability Semantics” from an extension of the prefix trace semantics of chapter 6, “Structural Deductive Stateless Prefix Trace Semantics” to include pointers.

Example 46.5 of execution trace of example 46.1



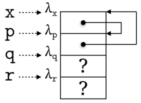
$\mu(\lambda_x)$	?
$\mu(\lambda_p)$	?
$\mu(\lambda_q)$	?
$\mu(\lambda_r)$	?

$\xrightarrow{\texttt{x = \&p;}}$



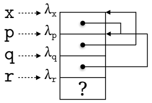
$\mu(\lambda_x)$	λ_p
$\mu(\lambda_p)$	?
$\mu(\lambda_q)$	?
$\mu(\lambda_r)$	?

$\xrightarrow{\texttt{p = \&x;}}$



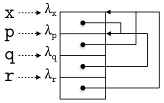
$\mu(\lambda_x)$	λ_p
$\mu(\lambda_p)$	λ_x
$\mu(\lambda_q)$	?
$\mu(\lambda_r)$	?

$\xrightarrow{\texttt{q = \&p;}}$



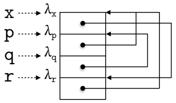
$\mu(\lambda_x)$	λ_p
$\mu(\lambda_p)$	λ_x
$\mu(\lambda_q)$	λ_p
$\mu(\lambda_r)$	?

$\xrightarrow{\texttt{r = *q;}}$



$\mu(\lambda_x)$	λ_p
$\mu(\lambda_p)$	λ_x
$\mu(\lambda_q)$	λ_p
$\mu(\lambda_r)$	λ_x

$\xrightarrow{\texttt{*r = \&r;}}$



$\mu(\lambda_x)$	λ_r
$\mu(\lambda_p)$	λ_x
$\mu(\lambda_q)$	λ_p
$\mu(\lambda_r)$	λ_x

Reachability semantics of the pointer language, section 46.4

- The reachability semantics

$$\widehat{\mathcal{S}}^{\vec{\varrho}}_{\lambda}[\![\mathsf{S}]\!]\,\mathscr{P}_0\ \ell$$

  defines the memory states reachable at program point $\ell \in \mathsf{labx}[\![\mathsf{S}]\!]$ of program component S when its execution starts in an initial memory state satisfying precondition $\mathscr{P}_0$.

- The reachability semantics $\widehat{\mathcal{S}}^{\vec{\varrho}}_{\lambda}$ is relative to a given assignment λ of a location λ_{x} to each variable $\mathsf{x} \in \mathbb{vars}[\![\mathsf{S}]\!]$ of program component S.
- The forward reachability semantics $\widehat{\mathcal{S}}^{\vec{\varrho}}[\![\mathsf{S}]\!]$ of a program component S in chapter 19 was operating on environments mapping variables to values,

$$\widehat{\mathcal{S}}^{\vec{\varrho}}[\![\mathsf{S}]\!] \quad \in \quad \wp(\mathbb{Ev}^{\varrho}) \rightarrow \mathsf{labx}[\![\mathsf{S}]\!] \rightarrow \wp(\mathbb{Ev}^{\varrho})$$

Reachability semantics of the pointer language, section 46.4

- The forward reachability semantics is now operating on memories (for a fixed allocation λ of locations to variables),

$$\widehat{\mathcal{S}}^{\vec{\varrho}}_{\lambda}[\![\mathsf{S}]\!] \quad \in \quad \wp(\mathbb{M}^{\varrho}_{\lambda}) \to \mathsf{labx}[\![\mathsf{S}]\!] \to \wp(\mathbb{M}^{\varrho}_{\lambda})$$

- So the value of a variable x which was $\rho(\mathsf{x})$ for environments $\rho \in \mathbb{E}\mathbb{v}$ is now $\mu(\lambda_{\mathsf{x}})$ for memories $\mu \in \mathbb{M}_{\lambda}$ with given memory allocation λ.
- By using symbolic addresses, the analysis is independent of the particular memory allocation λ determined at compile, linking, and loading time.
- We first extend the semantics of arithmetic and boolean expressions in section 3.6 to allow for pointer operations and comparison.

Definition of the semantics of arithmetic and pointer expressions

$$\mathscr{A} \in \lambda \in \mathbb{L}\mathbb{v} \to \mathcal{A} \to \mathbb{M}_\lambda \to \mathbb{V}_\lambda \cup \{\Omega\} \qquad (46.2)$$

$$\mathscr{A}_\lambda[\![1]\!]\,\mu \triangleq 1$$

$$\mathscr{A}_\lambda[\![\mathtt{x}]\!]\,\mu \triangleq \mu(\lambda_{\mathtt{x}})$$

$$\mathscr{A}_\lambda[\![\mathtt{A}_1 - \mathtt{A}_2]\!]\,\mu \triangleq (\!|\, \mathscr{A}_\lambda[\![\mathtt{A}_1]\!]\,\mu \in \mathbb{Z} \wedge \mathscr{A}_\lambda[\![\mathtt{A}_2]\!]\,\mu \in \mathbb{Z} \;?\; \mathscr{A}_\lambda[\![\mathtt{A}_1]\!]\,\mu - \mathscr{A}_\lambda[\![\mathtt{A}_2]\!]\,\mu \;:\; \Omega \,|\!)$$

$$\mathscr{A}_\lambda[\![\mathtt{NULL}]\!]\,\mu \triangleq \mathtt{nil}$$

$$\mathscr{A}_\lambda[\![\&\mathtt{x}]\!]\,\mu \triangleq \lambda_{\mathtt{x}}$$

$$\mathscr{A}_\lambda[\![\star\mathtt{p}]\!]\,\mu \triangleq (\!|\, \mu(\lambda_{\mathtt{p}}) \in \mathbb{Z} \cup \{\mathtt{nil}\} \;?\; \Omega \;:\; \mu(\mu(\lambda_{\mathtt{p}})) \,|\!)$$

Comments on the semantics of arithmetic and pointer expressions

- The evaluation of an expression may return an error Ω.
- `NULL` denotes the null pointer `nil`.
- `&x` denotes the location of the variable `x`.
- It is assumed that the locations λ allocated by the compiler for different variables cannot overlap and be erroneous so $\boldsymbol{\lambda} \triangleq \mathtt{x} \in \mathbb{V} \mapsto \lambda_{\mathtt{x}} \in \mathbb{L}$ is bijective.
- The value of a variable `x` is obtained by looking at its location $\lambda_{\mathtt{x}}$ and then at the value $\mu(\lambda_{\mathtt{x}})$ in the memory at that location.
- For `*p` there is one more level of indirection $\mu(\mu(\lambda_{\mathtt{p}}))$ unless `p` is not a pointer in which case this is a runtime error.
- Otherwise the definition is similar to (3.4), except that errors are propagated.

Definition of the semantics of boolean expressions

$$\mathscr{B} \in \lambda \in \mathbb{L}\mathbb{v} \to B \to \mathbb{M}_\lambda \to \mathbb{B} \cup \{\Omega\}$$

$$\mathscr{B}_\lambda[\![\mathsf{A}_1 < \mathsf{A}_2]\!]\,\mu \triangleq (\!|\, \mathscr{A}_\lambda[\![\mathsf{A}_1]\!]\,\mu \in \mathbb{Z} \wedge \mathscr{A}_\lambda[\![\mathsf{A}_2]\!]\,\mu \in \mathbb{Z} \;?\; \mathscr{A}_\lambda[\![\mathsf{A}_1]\!]\,\mu < \mathscr{A}_\lambda[\![\mathsf{A}_2]\!]\,\mu \;:\; \Omega \,|\!)$$

$$\mathscr{B}_\lambda[\![\mathsf{p} == \mathsf{NULL}]\!]\,\mu \triangleq (\mu(\lambda_{\mathsf{p}}) = \mathtt{nil})$$

$$\mathscr{B}_\lambda[\![\mathsf{x} == \mathsf{y}]\!]\,\mu \triangleq (\mu(\lambda_{\mathsf{x}}) = \mu(\lambda_{\mathsf{y}}))$$

$$\mathscr{B}_\lambda[\![\mathsf{B}_1 \text{ nand } \mathsf{B}_2]\!]\,\mu \triangleq (\!|\, \mathscr{B}_\lambda[\![\mathsf{B}_1]\!]\,\mu \neq \Omega \wedge \mathscr{B}_\lambda[\![\mathsf{B}_2]\!]\,\mu \neq \Omega \;?\; \mathscr{B}_\lambda[\![\mathsf{B}_1]\!]\,\mu \uparrow \mathscr{B}_\lambda[\![\mathsf{B}_2]\!]\,\mu \;:\; \Omega \,|\!)$$

Definition of the reachability of an arithmetic or pointer assignment statement

S ::= x = A ;

$$
\begin{array}{rcl}
\widehat{\mathcal{S}}^{\vec{\varrho}} & \in & \lambda \in \mathbb{L}\mathbb{v} \to \mathbb{S} \to \wp(\mathbb{M}^{\vec{\varrho}}_{\lambda}) \to \mathbb{L} \to \wp(\mathbb{M}^{\vec{\varrho}}_{\lambda}) \\
\widehat{\mathcal{S}}^{\vec{\varrho}}_{\lambda}[\![\mathsf{S}]\!]\, \mathscr{P}_0\, \ell & = & (\!|\ \ell = \mathsf{at}[\![\mathsf{S}]\!]\ ?\ \mathscr{P}_0 \\
 & & \|\ \ell = \mathsf{after}[\![\mathsf{S}]\!]\ ?\ \mathsf{assign}^{\vec{\varrho}}_{\lambda}[\![\mathsf{x}, \mathsf{A}]\!]\ \mathscr{P}_0 \\
 & & ⁏\ \varnothing\ |\!) \\
\mathsf{assign}^{\vec{\mathsf{r}}}_{\lambda}[\![\mathsf{x}, \mathsf{A}]\!]\ \mathscr{P}_0 & \triangleq & \{\mu[\lambda_{\mathsf{x}} \leftarrow \mathscr{A}_{\lambda}[\![\mathsf{A}]\!]\,\mu] \mid \mu \in \mathscr{P}_0 \wedge \mathscr{A}_{\lambda}[\![\mathsf{A}]\!]\,\mu \neq \Omega\} \\
\mathsf{assign}^{\vec{\mathsf{R}}}_{\lambda}[\![\mathsf{x}, \mathsf{A}]\!]\ \mathscr{P}_0 & \triangleq & \{\langle \mu_0,\ \mu[\lambda_{\mathsf{x}} \leftarrow \mathscr{A}_{\lambda}[\![\mathsf{A}]\!]\,\mu]\rangle \mid \langle \mu_0,\ \mu\rangle \in \mathscr{P}_0 \wedge \mathscr{A}_{\lambda}[\![\mathsf{A}]\!]\,\mu \neq \Omega\}
\end{array}
\qquad (46.3)
$$

Comments on the semantics of arithmetic and pointer expressions

- The semantics (19.12) of an assignment statement $\mathsf{S} ::= \mathsf{x} = \mathsf{A}\ ;$ maps the initial states to themselves on entry and to themselves as modified by the assignment on exit.
- An arithmetic or pointer assignment modifies the memory (but not the allocation of locations to variables which are chosen statically by the compiler and loader hence immutable during execution).
- Moreover, if the evaluation of the expression A returns an error Ω then the execution stops so there is no reachable successor state (so the postcondition is $\emptyset$).
- The difference with (19.12) is that the value of a variable x has to be fetched in memory μ at the location λ_{x} for that variable so $\rho(\mathsf{x}) = \mu(\lambda_{\mathsf{x}})$ and that execution stops on error (there was no error in (19.12) but e.g. integer overflow could be handled similarly by stopping execution).

Definition of the reachability of a dereferenced pointer assignment statement

$$\mathsf{S} ::= \star\mathsf{p} = \mathsf{A} \;;$$

$$\hat{\mathcal{S}}^{\vec{\varrho}} \in \lambda \in \mathbb{L}\mathbb{v} \to \mathbb{S} \to \wp(\mathbb{M}^{\vec{\varrho}}_{\lambda}) \to \mathbb{L} \to \wp(\mathbb{M}^{\vec{\varrho}}_{\lambda}) \qquad (46.4)$$

$$\begin{aligned}
\hat{\mathcal{S}}^{\vec{\varrho}}_{\lambda}[\![\mathsf{S}]\!]\,\mathcal{P}_0\,\ell &= (\!|\; \ell = \mathsf{at}[\![\mathsf{S}]\!] \;?\; \mathcal{P}_0 \\
&\quad\;\; [\!]\; \ell = \mathsf{after}[\![\mathsf{S}]\!] \;?\; \mathsf{assign}^{\vec{\varrho}}_{\lambda}[\![\star\mathsf{p}, \mathsf{A}]\!]\;\; \mathcal{P}_0 \\
&\quad\;\; \text{:}\; \varnothing \;|\!)
\end{aligned}$$

$$\mathsf{assign}^{\vec{\mathsf{r}}}_{\lambda}[\![\star\mathsf{p}, \mathsf{A}]\!]\;\;\mathcal{P}_0 \triangleq \{\mu[\mu(\lambda_{\mathsf{p}}) \leftarrow \mathcal{A}_{\lambda}[\![\mathsf{A}]\!]\,\mu] \mid \mu \in \mathcal{P}_0 \wedge \mu(\lambda_{\mathsf{p}}) \in \mathbb{L} \wedge \mathcal{A}_{\lambda}[\![\mathsf{A}]\!]\,\mu \neq \Omega\}$$

$$\begin{aligned}
\mathsf{assign}^{\vec{\mathsf{R}}}_{\lambda}[\![\star\mathsf{p}, \mathsf{A}]\!]\;\;\mathcal{P}_0 \triangleq \{\langle\mu_0,\; \mu[\mu(\lambda_{\mathsf{p}}) \leftarrow \mathcal{A}_{\lambda}[\![\mathsf{A}]\!]\,\mu]\rangle \mid \langle\mu_0,\; \mu\rangle \in \mathcal{P}_0 \wedge \mu(\lambda_{\mathsf{p}}) \in \mathbb{L} \wedge \\
\mathcal{A}_{\lambda}[\![\mathsf{A}]\!]\,\mu \neq \Omega\}
\end{aligned}$$

Comments on the reachability semantics of a dereferenced pointer assignment statement `S ::= *p = A ;`

- The assignment to a dereferenced pointer `*p = A ;` assigns the value of expression `A` to the memory pointed to by pointer `p`.
- This is an error if `p` does not contain a pointer i.e. is an integer or `nil` so that $\mu(\lambda_{\mathtt{p}}) \notin \mathbb{L}$.
- An error can also be raised by the evaluation of expression `A` (e.g. by dereferencing a null pointer).
- In case of error, the execution stops so there is no reachable successor state.

Reachability semantics of a test B

$$\begin{array}{rcl}
\mathsf{test}^{\vec{\varrho}}, \overline{\mathsf{test}}^{\vec{\varrho}} & \in & \lambda \in \mathbb{L}\mathbb{v} \longrightarrow B \longrightarrow \wp(\mathbb{M}^{\vec{\varrho}}_{\lambda}) \longrightarrow \wp(\mathbb{M}^{\vec{\varrho}}_{\lambda}) \\
\mathsf{test}^{\vec{\mathsf{r}}}_{\lambda}[\![\mathsf{B}]\!]\,\mathscr{P}_0 & \triangleq & \{\mu \in \mathscr{P}_0 \mid \boldsymbol{\mathscr{B}}_{\lambda}[\![\mathsf{B}]\!]\,\mu = \mathsf{tt}\} \\
\mathsf{test}^{\vec{\mathsf{R}}}_{\lambda}[\![\mathsf{B}]\!]\,\mathscr{P}_0 & \triangleq & \{\langle\mu_0,\ \mu\rangle \in \mathscr{P}_0 \mid \boldsymbol{\mathscr{B}}_{\lambda}[\![\mathsf{B}]\!]\,\mu = \mathsf{tt}\} \\
\overline{\mathsf{test}}^{\vec{\mathsf{r}}}_{\lambda}[\![\mathsf{B}]\!]\,\mathscr{P}_0 & \triangleq & \{\mu \in \mathscr{P}_0 \mid \boldsymbol{\mathscr{B}}_{\lambda}[\![\mathsf{B}]\!]\,\mu = \mathsf{ff}\} \\
\overline{\mathsf{test}}^{\vec{\mathsf{R}}}_{\lambda}[\![\mathsf{B}]\!]\,\mathscr{P}_0 & \triangleq & \{\langle\mu_0,\ \mu\rangle \in \mathscr{P}_0 \mid \boldsymbol{\mathscr{B}}_{\lambda}[\![\mathsf{B}]\!]\,\mu = \mathsf{ff}\}
\end{array} \tag{46.6}$$

Comments on the reachability semantics of a test B

- The reachability test transformers of (19.16) are adapted to the memory model. In case of error $\mathscr{B}[\![\mathsf{B}]\!]\,\lambda\,\mu = \Omega \notin \{\mathsf{tt}, \mathsf{ff}\}$, the execution stops so there is no reachable successor state.
- Observe that, by our definition of reachability, the fatal error Ω is never reachable by our assumption that execution stop in case of error. To improve error reporting, it would also be possible to keep errors in the reachable states (even with e.g. a program point to better track the origin of errors).

Reachability semantics of all other program components

- The reachability semantics $\widehat{\mathcal{S}}^{\vec{\varrho}}_{\lambda}\ \mathscr{P}_0\ \ell$ of all other program components in (19.12) is modified in the same way, replacing environments ρ by memories μ for the given location assignment λ and stopping execution when expressions (e.g. boolean conditions in conditional or iteration statements) return an error.
- For example the reachability semantics (19.22) and (19.23) of conditional statements and (19.16) of an iteration statement remain the same but for the definition of $\text{test}^{\vec{\varrho}}[\![\mathsf{B}]\!]$ by (46.6)
- Similarly for $\overline{\text{test}}^{\vec{\varrho}}[\![\mathsf{B}]\!]$. In case the evaluation $\mathscr{B}[\![\mathsf{B}]\!]\ \lambda\ \mu$ of the boolean expression B returns an error Ω, $\text{test}^{\vec{\varrho}}[\![\mathsf{B}]\!]X(\ell)$ is the empty set $\varnothing$, so by theorem 19.36, $\widehat{\mathcal{S}}^{\vec{\varrho}}[\![\mathsf{S}_b]\!]\ (\text{test}^{\vec{\varrho}}[\![\mathsf{B}]\!]X(\ell))\ \ell\prime$ is also the empty set $\varnothing$ i.e. there are no further reachable states since the trace execution is stopped by the error.

Abstract interpreters for the pointer language, section 46.5

Objective

- We extend the flow sensitive static analysis of section 21.2 and the flow insensitive static analysis of section 45.3 to handle the pointer statements of section 46.1.

Flow sensitive abstract interpreter for the pointer language

- The abstract interpreter of section 21.2 is designed for an abstract domain

$$\mathbb{D}^{\natural} \quad \triangleq \quad \langle \mathbb{P}^{\natural},\ \sqsubseteq^{\natural},\ \bot^{\natural},\ \sqcup^{\natural},\ \mathsf{assign}^{\natural}_{\lambda}[\![\mathsf{x}, \mathsf{A}]\!],\ \mathsf{test}^{\natural}[\![\mathsf{B}]\!],\ \overline{\mathsf{test}}^{\natural}[\![\mathsf{B}]\!]\rangle$$

  is unchanged

- but for the concretization $\gamma_{\lambda} \in \mathbb{P}^{\natural}_{\lambda} \xrightarrow{\nearrow} \wp(\mathbb{M}^{\varrho}_{\lambda})$ to the memory collecting domain $\wp(\mathbb{M}^{\varrho}_{\lambda})$ instead of the environment collecting domain $\wp(\mathbb{E}\mathbb{v}^{\varrho})$.
- and (21.7) becomes

$$\mathbb{D}^{\natural}_{\lambda} \quad \triangleq \quad \langle \mathbb{P}^{\natural}_{\lambda},\ \sqsubseteq^{\natural}_{\lambda},\ \bot^{\natural}_{\lambda},\ \sqcup^{\natural}_{\lambda},\ \mathsf{assign}^{\natural}_{\lambda}[\![\mathsf{x}, \mathsf{A}]\!],\ \mathsf{assign}^{\natural}_{\lambda}[\![\star\mathsf{p}, \mathsf{A}]\!],\ \mathsf{test}^{\natural}_{\lambda}[\![\mathsf{B}]\!],\ \overline{\mathsf{test}}^{\natural}_{\lambda}[\![\mathsf{B}]\!]\rangle$$

Flow-sensitive abstract semantics of an assignment statement S ::= x = A ;

$$\begin{array}{rcl} \widehat{\mathcal{S}}^{¤} & \in & \lambda \in \mathbb{L}\mathbb{v} \to \mathcal{S} \to \mathbb{P}^{¤}_{\lambda} \to \mathcal{L} \to \mathbb{P}^{¤}_{\lambda} \\ \widehat{\mathcal{S}}^{¤}_{\lambda}\, \overline{P}\, \ell & \triangleq & \llparenthesis\ \ell = \mathsf{at}[\![\mathsf{S}]\!] \mathrel{?} \overline{P} \\ & & [\!]\ \ell = \mathsf{after}[\![\mathsf{S}]\!] \mathrel{?} \mathsf{assign}^{¤}_{\lambda}[\![\mathsf{x}, \mathsf{A}]\!]\, \lambda\, \overline{P} \\ & & \mathrel{⁏} \bot^{¤}\ \rrparenthesis \end{array} \tag{46.8}$$

where $\mathsf{assign}^{¤}_{\lambda}[\![\mathsf{x}, \mathsf{A}]\!] \in \mathbb{L} \to \mathbb{P}^{¤} \to \mathbb{P}^{¤}$ satisfies $\mathsf{assign}^{\vec{\varrho}}_{\lambda}[\![\mathsf{x}, \mathsf{A}]\!] \circ \gamma_{\lambda} \subseteq \gamma_{\lambda} \circ \mathsf{assign}^{¤}_{\lambda}[\![\mathsf{x}, \mathsf{A}]\!]$.

Flow-sensitive abstract semantics of a dereferenced pointer assignment statement S ::= ⋆p = A ;

$$
\begin{array}{rcl}
\widehat{\mathcal{S}}^{\texttt{¤}} & \in & \lambda \in \mathbb{L}\mathbb{v} \longrightarrow \mathbb{S} \longrightarrow \mathbb{P}^{\texttt{¤}}_{\lambda} \longrightarrow \mathbb{L} \longrightarrow \mathbb{P}^{\texttt{¤}}_{\lambda} \\
\widehat{\mathcal{S}}^{\texttt{¤}}_{\lambda}[\![\mathsf{S}]\!]\,\overline{P}\,\ell & \triangleq & (\!\!|\ \ell = \mathsf{at}[\![\mathsf{S}]\!]\ ?\ \overline{P} \\
& & \|\ \ell = \mathsf{after}[\![\mathsf{S}]\!]\ ?\ \mathsf{assign}^{\texttt{¤}}_{\lambda}[\![\star\mathsf{p},\mathsf{A}]\!]\,\overline{P} \\
& & \mathbin{⁏}\ \bot^{\texttt{¤}}_{\lambda}\ |\!\!)
\end{array}
\tag{46.9}
$$

where $\mathsf{assign}^{\texttt{¤}}_{\lambda}[\![\star\mathsf{p},\mathsf{A}]\!] \in \mathbb{P}^{\texttt{¤}}_{\lambda} \longrightarrow \mathbb{P}^{\texttt{¤}}_{\lambda}$ satisfies $\mathsf{assign}^{\vec{\varrho}}_{\lambda}[\![\star\mathsf{p},\mathsf{A}]\!] \circ \gamma_{\lambda} \subseteq \gamma_{\lambda} \circ \mathsf{assign}^{\texttt{¤}}_{\lambda}[\![\star\mathsf{p},\mathsf{A}]\!]$.

- The abstract semantics of conditional and iteration statements is unchanged except for the use of (46.6) for the collecting semantics of tests.

Theorem (46.10) Well-definedness of the reachability semantics and the abstract interpreter The abstract interpreter $\widehat{\mathcal{S}}^{¤}_{\lambda}[\![\mathsf{S}]\!]$ for the abstract domain $\mathbb{D}^{¤}_{\lambda}$ assumed to be well-defined by definition 21.1 (and assuming $\mathsf{assign}^{¤}_{\lambda}[\![\mathsf{x}, \mathsf{A}]\!]$, $\mathsf{test}^{¤}_{\lambda}[\![\mathsf{B}]\!]$, and $\overline{\mathsf{test}}^{¤}_{\lambda}[\![\mathsf{B}]\!]$ to be continuous) is well-defined and continuous for any program component $r\mathsf{S} \in \mathbb{P}\mathbb{c}$.

Proof The language extension of section 46.1 preserves the well-definedness of the abstract interpreter in theorem 21.16, that of the reachability semantics in corollary **??**, as well as the soundness of the abstract interpreted theorem 27.4 (in particular for the collecting semantics (46.3) and (46.4)). □

Flow insensitive abstract interpreter for the pointer language

- As in chapter 45, “Flow-Insensitive Static Analysis,” the flow insensitive abstract semantics for the pointer language is obtained by the $\alpha_{\mathfrak{g}}$-abstraction of its flow insensitive semantics.

Flow-insensitive semantics of an assignment statement S ::= x = A ;

$$\begin{array}{rcl} \widehat{\mathcal{S}}^{\mathfrak{g}¤} & \in & \lambda \in \mathbb{L}_{\mathbb{v}} \to \mathcal{S} \to \mathbb{P}^{¤}_{\lambda} \to \mathbb{P}^{¤}_{\lambda} \\ \widehat{\mathcal{S}}^{\mathfrak{g}¤}_{\lambda}[\![\mathsf{S}]\!]\,\overline{P} & \triangleq & \overline{P} \sqcup^{¤}_{\lambda} \mathsf{assign}^{¤}_{\lambda}[\![\mathsf{x}, \mathsf{A}]\!]\,\overline{P} \end{array} \tag{46.12}$$

Proof of (46.12)

$$\begin{array}{rll} & \dot{\alpha}_{\mathfrak{g}}(\widehat{\mathcal{S}}^{¤}[\![\mathsf{x} = \mathsf{A}\ ;]\!]\ \lambda)\overline{P} & \\ = & \alpha_{\mathfrak{g}}(\widehat{\mathcal{S}}^{¤}[\![\mathsf{x} = \mathsf{A}\ ;]\!]\ \lambda\ \overline{P}) & \text{(def. (45.1) of } \dot{\alpha}_{\mathfrak{g}}\text{)} \\ = & \alpha_{\mathfrak{g}}((\!|\ \ell = \mathsf{at}[\![\mathsf{S}]\!]\ ?\ \overline{P}\ \|\ \ell = \mathsf{after}[\![\mathsf{S}]\!]\ ?\ \mathsf{assign}^{¤}_{\lambda}[\![\mathsf{x}, \mathsf{A}]\!]\ \overline{P}\ ⁏\ \bot^{¤}\ |\!)) & \text{(def. (46.8) of } \widehat{\mathcal{S}}^{¤}\text{)} \\ = & \overline{P} \sqcup^{¤}_{\lambda} \mathsf{assign}^{¤}_{\lambda}[\![\mathsf{x}, \mathsf{A}]\!]\ \overline{P} & \text{(def. (45.1) of } \alpha_{\mathfrak{g}} \text{ and } \mathsf{labx}[\![\mathsf{S}]\!] = \{\mathsf{at}[\![\mathsf{S}]\!], \mathsf{after}[\![\mathsf{S}]\!]\}\text{)} \\ \triangleq & \widehat{\mathcal{S}}^{\mathfrak{g}¤}_{\lambda}[\![\mathsf{S}]\!]\ \lambda\ \overline{P} & \text{(as defined in (46.12))} \quad \square \end{array}$$

By a similar proof using (45.1) and (46.9), we get (46.13).

Flow-insensitive semantics of a dereferenced pointer assignment statement S ::= $\star$p = A ;

$$
\begin{aligned}
\widehat{\boldsymbol{\mathcal{S}}}^{\mathfrak{g}¤} &\in \lambda \in \mathbb{L}\mathbb{v} \to \mathcal{S} \to \mathbb{P}^{¤}_{\lambda} \to \mathbb{P}^{¤}_{\lambda} \\
\widehat{\boldsymbol{\mathcal{S}}}^{\mathfrak{g}¤}_{\lambda}[\![\mathsf{S}]\!]\,\overline{P} &= \overline{P} \sqcup^{¤}_{\lambda} \mathsf{assign}^{¤}_{\lambda}[\![\star\mathsf{p}, \mathsf{A}]\!]\,\overline{P}
\end{aligned}
\tag{46.13}
$$

theorem 45.14 becomes

Theorem (46.14) The flow insensitive abstract semantics $\widehat{\boldsymbol{\mathcal{S}}}^{\mathrm{g}¤}_{\lambda}[\![\mathsf{S}]\!]$ on a well-defined abstract domain of definition 21.1 is well-defined.

Cartesian abstract interpreters for the pointer language, section 46.5

Cartesian abstract domain for the pointer language

- Most existing points-to static analyzes are cartesian.
- Cartesian points-to analyzers can express x and y may point only to the locations of u or v but not that x and y may only point to the same location either u or v.
- For such cartesian analyzes, chapter 28, “Abstract Cartesian Semantics” is most relevant and the cartesian semantics can be easily extended to handle the pointer statements of section 46.1.

Cartesian abstraction for the pointer language

- We have $\mathbb{M}_\lambda \triangleq \mathbb{L}_\lambda \to \mathbb{V}_\lambda$.
- Assume we abstract value properties by

$$\langle \wp(\mathbb{V}_\lambda),\ \subseteq\rangle \xleftarrow[\alpha^{¤}_\lambda]{\gamma^{¤}_\lambda} \langle \mathbb{P}^{¤}_\lambda,\ \sqsubseteq^{¤}_\lambda\rangle. \tag{46.16}$$

- Then the cartesian abstraction is

$$\langle \wp(\mathbb{L}_\lambda \to \mathbb{V}_\lambda),\ \subseteq\rangle \xleftarrow[\dot{\alpha}^{\times}_\lambda]{\dot{\gamma}^{\times}_\lambda} \langle \mathbb{L}_\lambda \to \wp(\mathbb{V}_\lambda),\ \dot{\subseteq}\rangle \xleftarrow[\dot{\alpha}^{¤}_\lambda]{\dot{\gamma}^{¤}_\lambda} \langle \mathbb{L}_\lambda \to \mathbb{P}^{¤}_\lambda,\ \dot{\sqsubseteq}^{¤}_\lambda\rangle$$

  where

$$\dot{\alpha}^{\times}_\lambda(P) \quad\triangleq\quad l \in \mathbb{L}_\lambda \mapsto \{\mu(l) \mid \mu \in P\} \tag{46.17}$$

  is the cartesian abstraction and $\dot{\alpha}^{¤}_\lambda(P) \triangleq l \in \mathbb{L}_\lambda \mapsto \alpha^{¤}_\lambda(P(l))$ is the pointwise extension of $\alpha^{¤}_\lambda$.
- By composition of these Galois connections, we get

$$\langle \wp(\mathbb{L}_\lambda \to \mathbb{V}_\lambda),\ \subseteq\rangle \xleftarrow[\dot{\alpha}^{\times¤}_\lambda]{\dot{\gamma}^{\times¤}_\lambda} \langle \dot{\mathbb{P}}^{¤}_\lambda,\ \dot{\sqsubseteq}^{¤}\rangle \quad \text{where} \quad \dot{\mathbb{P}}^{¤}_\lambda \triangleq \mathbb{L}_\lambda \to \mathbb{P}^{¤}_\lambda$$

  with $\dot{\alpha}^{\times¤}_\lambda(P) \triangleq \dot{\alpha}^{¤}_\lambda \circ \dot{\alpha}^{\times}_\lambda(P) = l \in \mathbb{L}_\lambda \mapsto \alpha^{¤}_\lambda(\{\mu(l) \mid \mu \in P\})$.

Cartesian abstract domain

- The cartesian abstract domain

$$\mathbb{D}^{\text{¤}} \quad \triangleq \quad \langle \mathbb{P}^{\text{¤}},\ \sqsubseteq^{\text{¤}},\ \bot^{\text{¤}},\ \top^{\text{¤}},\ \sqcup^{\text{¤}},\ \sqcap^{\text{¤}},\ 1^{\text{¤}},\ \ominus^{\text{¤}},\ \ominus^{\text{¤}-1},\ \olessthan^{\text{¤}},\ \overline{\olessthan}^{\text{¤}} \rangle \qquad (28.42)$$

is extended to include abstract pointer operations

$$\mathbb{D}^{\text{¤}}_{\lambda} \quad \triangleq \quad \langle \mathbb{P}^{\text{¤}}_{\lambda},\ \sqsubseteq^{\text{¤}}_{\lambda},\ \bot^{\text{¤}}_{\lambda},\ \top^{\text{¤}}_{\lambda},\ \sqcup^{\text{¤}}_{\lambda},\ \sqcap^{\text{¤}}_{\lambda},\ 1^{\text{¤}}_{\lambda},\ \ominus^{\text{¤}}_{\lambda},\ \ominus^{\text{¤}-1}_{\lambda},\ \star^{\text{¤}-1}_{\lambda},\ \olessthan^{\text{¤}},\ \overline{\olessthan}^{\text{¤}},\ \mathsf{NULL}^{\text{¤}}_{\lambda},\ \boldsymbol{\lambda}^{\text{¤}}_{\lambda},\ \star^{\text{¤}}_{\lambda} \rangle \qquad (46.18)$$

with well-definedness conditions $\mathsf{NULL}^{\text{¤}}_{\lambda} \in \mathbb{P}^{\text{¤}}_{\lambda}$, $\boldsymbol{\lambda}^{\text{¤}}, \star^{\text{¤}} \in \mathbb{L}\mathbb{v} \to \mathbb{V} \to \mathbb{P}^{\text{¤}}_{\lambda}$.

- We must then calculate structural definitions of $\mathscr{A}^{\text{¤}}_{\lambda}[\![\mathsf{A}]\!]\,\overline{P}$, $\mathscr{A}^{\text{¤}-1}_{\lambda}[\![\mathsf{1}]\!]\,\chi\,\overline{P}$, $\mathsf{assign}^{\text{¤}}_{\lambda}[\![\mathsf{x}, \mathsf{A}]\!]$, $\mathsf{assign}^{\text{¤}}_{\lambda}[\![\star\mathsf{p}, \mathsf{A}]\!]$, $\mathsf{test}^{\text{¤}}_{\lambda}[\![\mathsf{B}]\!]$, and $\overline{\mathsf{test}}^{\text{¤}}_{\lambda}[\![\mathsf{B}]\!]$ to extend (28.17), (28.30), and (28.38) for the pointer language of section 46.1.

Cartesian forward reachability of an arithmetic or pointer expressions

- The cartesian reachability abstraction of expressions is

$$\mathscr{A}^{\text{¤}}_{\lambda}[\![\mathsf{A}]\!] \in (\mathbb{L}_\lambda \longrightarrow \mathbb{P}^{\text{¤}}_\lambda) \longrightarrow \mathbb{P}^{\text{¤}}_\lambda$$

such that

$$\mathscr{A}^{\text{¤}}_{\lambda}[\![\mathsf{A}]\!]\,\overline{P} \quad \dot{\sqsupseteq}^{\text{¤}}_\lambda \quad \alpha^{\text{¤}}_\lambda(\{\mathscr{A}_\lambda[\![\mathsf{A}]\!]\,\mu \mid \mu \in \dot{\gamma}^{\times\text{¤}}_\lambda(\overline{P})\})$$

- To define $\mathscr{A}^{\text{¤}}_{\lambda}[\![\mathsf{A}]\!]$ structurally, we extend (28.17) as follows (in case $\gamma^{\text{¤}}_\lambda(\bot^{\text{¤}}) = \varnothing$, the smash of the cartesian property $\owedge^{\text{¤}}_\lambda$ ensures strictness by a test for the infimum $\dot{\bot}^{\text{¤}}_\lambda$)

Cartesian abstract forward reachability semantics of an arithmetic or pointer expression A

The abstract cartesian semantics of an expression A and an assignment statement x = A ; is defined as follows ($\overline{Q} \in \mathbb{P}^{¤}_{\lambda}, \overline{P} \in \mathbb{L}_{\lambda} \to \mathbb{P}^{¤}_{\lambda}$)

$$
\begin{array}{rcll}
\owedge^{¤}_{\lambda}(\overline{Q})\,\overline{P} & \triangleq & \overline{Q} & \text{if } \gamma^{¤}_{\lambda}(\bot^{¤}_{\lambda}) \neq \varnothing \quad \text{smash} \\
 & \triangleq & (\!| \overline{P} = \dot{\bot}^{¤}_{\lambda} \;?\; \bot^{¤}_{\lambda} \;:\; \overline{Q} |\!) & \text{if } \gamma^{¤}_{\lambda}(\bot^{¤}_{\lambda}) = \varnothing \\
\mathscr{A}^{¤} & \in & \lambda \in \mathbb{A} \to \mathbb{L}_{\lambda} \to \dot{\mathbb{P}}^{¤} \to \mathbb{V}_{\lambda} \quad \text{where} \quad \dot{\mathbb{P}}^{¤} \triangleq \mathbb{L}_{\lambda} \to \mathbb{P}^{¤}_{\lambda} & \qquad (46.19) \\
\mathscr{A}^{¤}_{\lambda}[\![1]\!]\,\overline{P} & \triangleq & \owedge^{¤}_{\lambda}(1^{¤}_{\lambda})\,\overline{P} & 1^{¤}_{\lambda} \dot{\sqsupseteq}^{¤}_{\lambda} \alpha^{¤}_{\lambda}(\{1\}) \\
\mathscr{A}^{¤}_{\lambda}[\![\mathsf{x}]\!]\,\overline{P} & \triangleq & \overline{P}(\lambda_{\mathsf{x}}) & \\
\mathscr{A}^{¤}_{\lambda}[\![\mathsf{A}_1 - \mathsf{A}_2]\!]\,\overline{P} & \triangleq & \owedge^{¤}_{\lambda}(\mathscr{A}^{¤}_{\lambda}[\![\mathsf{A}_1]\!]\,\overline{P} \ominus^{¤}_{\lambda} \mathscr{A}^{¤}_{\lambda}[\![\mathsf{A}_2]\!]\,\overline{P})\,\overline{P} & \\
\multicolumn{4}{c}{\text{where } \overline{P} \ominus^{¤}_{\lambda} \overline{Q} \dot{\sqsupseteq}^{¤}_{\lambda} \alpha^{¤}_{\lambda}(\{x - y \mid x \in \gamma^{¤}_{\lambda}(\overline{P}) \cap \mathbb{Z} \wedge y \in \gamma^{¤}_{\lambda}(\overline{Q}) \cap \mathbb{Z}\})}
\end{array}
$$

$$\begin{array}{rcll}
\mathscr{A}^{¤}_{\lambda}[\![\mathtt{NULL}]\!]\,\overline{P} & \triangleq & \owedge^{¤}_{\lambda}(\mathtt{NULL}^{¤}_{\lambda})\,\overline{P} & \mathtt{NULL}^{¤}_{\lambda} \mathrel{\dot{\sqsupseteq}}^{¤}_{\lambda} \alpha^{¤}_{\lambda}(\{\mathtt{nil}\}) \\
\mathscr{A}^{¤}_{\lambda}[\![\&\mathtt{x}]\!]\,\overline{P} & \triangleq & \owedge^{¤}_{\lambda}(\boldsymbol{\lambda}^{¤}_{\lambda}[\![\mathtt{x}]\!])\,\overline{P} & \boldsymbol{\lambda}^{¤}_{\lambda}[\![\mathtt{x}]\!] \mathrel{\dot{\sqsupseteq}}^{¤}_{\lambda} \alpha^{¤}_{\lambda}(\{\lambda_{\mathtt{x}}\}) \\
\mathscr{A}^{¤}_{\lambda}[\![\star\mathtt{p}]\!]\,\overline{P} & \triangleq & \owedge^{¤}_{\lambda}(\star^{¤}_{\lambda}[\![\mathtt{p}]\!]\,\overline{P})\,\overline{P} &
\end{array}$$

where $\star^{¤}_{\lambda}[\![\mathtt{p}]\!]\,\overline{P} \mathrel{\dot{\sqsupseteq}}^{¤}_{\lambda} \alpha^{¤}_{\lambda}(\{\mu(\mu(\lambda_{\mathtt{p}})) \mid \mu \in \dot{\gamma}^{\times ¤}_{\lambda}(\overline{P}) \wedge \mu(\lambda_{\mathtt{p}}) \in \mathbb{L}\})$

Remark 46.20

- By choosing $\alpha^{¤}_{\lambda}$ to be the identity, we get the definition of cartesian reachability $\mathscr{A}^{\times}_{\lambda}[\![\mathsf{A}]\!]\,\overline{P} \triangleq \{\mathscr{A}_{\lambda}[\![\mathsf{A}]\!]\,\mu \mid \mu \in \dot{\gamma}^{\times}_{\lambda}(\overline{P})\} \in (\mathbb{L}_{\lambda} \to \wp(\mathbb{V}_{\lambda}) \to \wp(\mathbb{V}_{\lambda})$ as an instance of $\mathscr{A}^{¤}_{\lambda}[\![\mathsf{A}]\!]\,\overline{P}$ with
  - $\dot{\mathbb{P}}^{¤}_{\lambda} \triangleq \wp(\mathbb{L}_{\lambda} \to \mathbb{V}_{\lambda})$,
  - $\dot{\bot}^{\times}_{\lambda} \triangleq \dot{\varnothing}$,
  - $\not{\bot}^{\times}_{\lambda} \triangleq \varnothing$,
  - $1^{\times}_{\lambda} \triangleq \{1\}$,
  - $P \ominus^{\times}_{\lambda} Q = \{x - y \mid x \in P \cap \mathbb{Z} \wedge y \in Q \cap \mathbb{Z}\}$,
  - $\mathtt{NULL}^{\times}_{\lambda} \triangleq \{\mathtt{nil}\}$,
  - $\boldsymbol{\lambda}^{\times}_{\lambda}[\![\mathsf{x}]\!] \triangleq \{\lambda_{\mathsf{x}}\}$, and
  - $\star^{\times}_{\lambda}[\![\mathsf{p}]\!]\,P \triangleq \{\mu(\mu(\lambda_{\mathsf{p}})) \mid \mu \in P \wedge \mu(\lambda_{\mathsf{p}}) \in \mathbb{L}\}$.

Then lemma 28.14 remains valid, as follows

Lemma (46.21)

$\forall P \in \wp(\mathbb{L}_\lambda \to \mathbb{V}_\lambda)\ .\ \{\mathscr{A}_\lambda[\![\mathsf{A}]\!]\,\mu \mid \mu \in P \wedge \mathscr{A}_\lambda[\![\mathsf{A}]\!]\,\mu \neq \Omega\} \subseteq \mathscr{A}^{\times}_\lambda[\![\mathsf{A}]\!](\dot{\alpha}^{\times}_\lambda(P))$.

TODO: A revoir

Proof of lemma 46.21

— $\{\mathscr{A}[\![\mathtt{NULL}]\!]\,\lambda\,\mu \mid \mu \in P \wedge \mathscr{A}[\![\mathtt{NULL}]\!]\,\lambda\,\mu \neq \Omega\}$

$= (\!|\, P = \varnothing \mathrel{?} \varnothing \mathrel{:} \{\mathtt{nil}\} \,|\!)$ $\wr$ def. (46.2) of $\mathscr{A}[\![\mathtt{NULL}]\!]\,\lambda\,\mu \triangleq \mathtt{nil} \neq \Omega$ $\wr$

$= (\!|\, \dot{\alpha}_{\times}(P) = \dot{\varnothing} \mathrel{?} \varnothing \mathrel{:} \{\mathtt{nil}\} \,|\!)$ $\wr$ def. $\dot{\alpha}_{\times}(P) \triangleq \lambda \mapsto \{\mu(\lambda) \mid \mu \in P\}$ so $P = \varnothing \Leftrightarrow \dot{\alpha}_{\times}(P) = \dot{\varnothing}$ $\wr$

$= {}^{\times}(\mathtt{NULL}^{\times})\,\dot{\alpha}_{\times}(P)$ $\wr$ def. ${}^{\times}$ and $\mathtt{NULL}^{\times} \triangleq \{\mathtt{nil}\}$ $\wr$

$= \mathscr{A}^{¤}_{\lambda}[\![\mathtt{NULL}]\!]\,\dot{\alpha}_{\times}(P)$ $\wr$ def. (46.19) of $\mathscr{A}^{¤}_{\lambda}$ $\wr$

— $\{\mathscr{A}[\![\star\mathtt{p}]\!]\,\lambda\,\mu \mid \mu \in P \wedge \mathscr{A}[\![\star\mathtt{p}]\!]\,\lambda\,\mu \neq \Omega\}$

$= \{(\!|\, \mu(\lambda_{\mathtt{p}}) \in \mathbb{Z} \cup \{\mathtt{nil}\} \mathrel{?} \Omega \mathrel{:} \mu(\mu(\lambda_{\mathtt{p}})) \,|\!) \mid \mu \in P \wedge \mu(\lambda_{\mathtt{p}}) \notin \mathbb{Z} \cup \{\mathtt{nil}\}\}$

$\wr$ def. (46.2) of $\mathscr{A}[\![\star\mathtt{p}]\!]\,\lambda\,\mu$ $\wr$

$= \{\mu(\mu(\lambda_{\mathtt{p}})) \mid \mu \in P \wedge \mu(\lambda_{\mathtt{p}}) \in \mathbb{L}\}$ $\wr$ def. values $\mathbb{V} \triangleq \mathbb{Z} \cup \mathbb{L} \cup \{\mathtt{nil}\}$ $\wr$

$\subseteq \{\mu(\mu(\lambda_{\mathtt{p}})) \mid \mu \in \dot{\gamma}_{\times}(\dot{\alpha}_{\times}(P)) \wedge \mu(\lambda_{\mathtt{p}}) \in \mathbb{L}\}$

$$\begin{aligned}
&\quad\wr\text{cartesian abstraction Galois connection } \langle\wp(\mathbb{L}\to\mathbb{V}),\ \subseteq\rangle \xleftarrow[\dot\alpha_\times]{\dot\gamma_\times}\!\!\!\rightarrow \langle\mathbb{L}\to\wp(\mathbb{V}),\ \dot\subseteq\rangle \text{ and exercise } 11.35.3\wr\\
=&\ \llparenthesis\, \dot\alpha_\times(P) = \dot\varnothing \ ?\ \varnothing : \{\mu(\mu(\lambda_{\mathsf{p}})) \mid \mu \in \dot\gamma_\times(\dot\alpha_\times(P)) \wedge \mu(\lambda_{\mathsf{p}}) \in \mathbb{L}\}\,\rrparenthesis && \wr\dot\gamma_\times(\dot\alpha_\times(\dot\varnothing)) = \varnothing\wr\\
=&\ {}^{\times}(\star^{\times}_{\lambda}[\![\mathsf{p}]\!]\ \dot\alpha_\times(P))\ \dot\alpha_\times(P) && \wr\text{def. }{}^{\times}\text{ and } \star^{\times}_{\lambda}[\![\mathsf{p}]\!]\wr\\
=&\ \mathscr{A}^{\times}_{\lambda}[\![\mathsf{A}]\!]\ (\dot\alpha_\times(P)) && \wr\text{def. (46.19) of } \mathscr{A}^{¤}_{\lambda}[\![\star\mathsf{p}]\!]\ \overline{P} \triangleq^{\times} (\star^{\times}_{\lambda}[\![\mathsf{p}]\!]\ \overline{P})\ \overline{P}\wr
\end{aligned}$$

The proof of the other cases is similar. □

Cartesian forward reachability of a pointer assignment

Cartesian abstract forward reachability semantics of assignment x = A ;

$$\text{assign}^{¤}_{\lambda}[\![\mathsf{x}, \mathsf{A}]\!]\,\overline{P} \quad \triangleq \quad \overline{P}[\lambda_{\mathsf{x}} \leftarrow \mathscr{A}^{¤}_{\lambda}[\![\mathsf{A}]\!]\,\overline{P}] \qquad (46.22)$$

Cartesian abstract forward reachability semantics of assignment $\star$p = A ;

$$\begin{aligned}
\text{assign}^{¤}_{\lambda}[\![\star\mathsf{p}, \mathsf{A}]\!]\,\overline{P} \quad &\triangleq \quad \lambda_{\mathsf{y}} \mapsto \overline{P}(\lambda_{\mathsf{y}}) \sqcup^{¤} (\!| \boldsymbol{\lambda}^{¤}_{\lambda}[\![y]\!] \sqsubseteq^{¤} \overline{P}(\lambda_{\mathsf{p}}) \;?\; \mathscr{A}^{¤}[\![\mathsf{A}]\!]\,\overline{P} \;:\; \bot^{¤} |\!) \\
&\qquad\qquad \text{if } \alpha^{¤}_{\lambda}(\{\lambda_{\mathsf{x}}\}) = \boldsymbol{\lambda}^{¤}_{\lambda}[\![\mathsf{x}]\!] \\
&\triangleq \quad \lambda_{\mathsf{y}} \mapsto \overline{P}(\lambda_{\mathsf{y}}) \sqcup^{¤} \mathscr{A}^{¤}[\![\mathsf{A}]\!]\,\overline{P} \qquad \text{otherwise}
\end{aligned}$$

The cartesian assignment theorem 28.15 becomes (by calculational design)

Theorem (cartesian assignment for the pointer language)

$$\begin{aligned}
\dot{\alpha}^{\times}_{\lambda} \circ \mathsf{assign}^{\vec{r}}_{\lambda}[\![\mathsf{x}, \mathsf{A}]\!] &\quad \dot{\subseteq} \quad \mathsf{assign}^{\times}_{\lambda}[\![\mathsf{x}, \mathsf{A}]\!] \circ \dot{\alpha}^{\times}_{\lambda}\\
\dot{\alpha}^{\times}_{\lambda} \circ \mathsf{assign}^{\vec{r}}_{\lambda}[\![\star\mathsf{p}, \mathsf{A}]\!] &\quad \dot{\subseteq} \quad \mathsf{assign}^{\times}_{\lambda}[\![\star\mathsf{p}, \mathsf{A}]\!] \circ \dot{\alpha}^{\times}_{\lambda}
\end{aligned}$$

theorem **??** becomes

Theorem (46.24, Abstract cartesian assignment)
If $1^{\times} \sqsubseteq^{\times} \gamma^{¤}_{\lambda}(1^{¤})$, $\ominus^{\times}$ is $\sqsubseteq^{\times}$-argumentwise increasing, and $\forall X, Y \in \mathbb{P}^{¤}_{\lambda} \;.\; \gamma^{¤}_{\lambda}(X) \ominus^{\times} \gamma^{¤}_{\lambda}(Y) \sqsubseteq^{\times} \gamma^{¤}_{\lambda}(X \ominus^{¤} Y)$ then

$$\begin{array}{rcl}
\mathscr{A}^{\times}_{\lambda}[\![\mathsf{A}]\!] \circ \dot{\gamma}_{¤} & \sqsubseteq^{\times} & \gamma^{¤}_{\lambda} \circ \mathscr{A}^{¤}_{\lambda}[\![\mathsf{A}]\!] \\
\mathsf{assign}^{\times}_{\lambda}[\![\mathsf{x}, \mathsf{A}]\!] \circ \dot{\gamma}_{¤} & \dot{\sqsubseteq}^{\times} & \dot{\gamma}_{¤} \circ \mathsf{assign}^{¤}_{\lambda}[\![\mathsf{x}, \mathsf{A}]\!] \\
\mathsf{assign}^{\times}_{\lambda}[\![\star\mathsf{p}, \mathsf{A}]\!] \circ \dot{\gamma}_{¤} & \dot{\sqsubseteq}^{\times} & \dot{\gamma}_{¤} \circ \mathsf{assign}^{¤}_{\lambda}[\![\star\mathsf{p}, \mathsf{A}]\!]
\end{array}$$

Cartesian backward accessibility of an arithmetic or pointer expressions

The cartesian backward accessibility abstraction of expressions is
$\mathscr{A}^{¤-1}_{\lambda}[\![\mathsf{A}]\!] \in \mathbb{P}^{¤}_{\lambda} \to (\mathbb{L}_{\lambda} \to \mathbb{P}^{¤}_{\lambda}) \to (\mathbb{L}_{\lambda} \to \mathbb{P}^{¤}_{\lambda})$,

$$\mathscr{A}^{¤-1}_{\lambda}[\![\mathsf{A}]\!]\,\chi\,P \quad\triangleq\quad \alpha^{¤}_{\lambda}(\{\mu \in \dot{\gamma}^{\times¤}_{\lambda}(\overline{P}) \mid \mathscr{A}_{\lambda}[\![\mathsf{A}]\!]\,\mu \in \dot{\gamma}^{\times¤}_{\lambda}(\chi)\}).$$

We extend (28.30)

Cartesian backward accessibility semantics of an arithmetic or pointer expression A

$$
\begin{aligned}
\mathscr{A}^{¤-1}_{\lambda}[\![\mathsf{A}]\!] &\in \mathbb{P}^{¤}_{\lambda} \to (\mathbb{L}_{\lambda} \to \mathbb{P}^{¤}_{\lambda}) \to (\mathbb{L}_{\lambda} \to \mathbb{P}^{¤}_{\lambda}) \qquad (46.25)\\
\mathscr{A}^{¤-1}_{\lambda}[\![1]\!]\ \chi\ \overline{P} &\triangleq (\!|\ 1^{¤} \sqcap^{¤} \chi \neq \bot^{¤} \mathrel{?} \overline{P} \mathrel{:} \dot{\bot}^{¤}\ |\!)\\
\mathscr{A}^{¤-1}_{\lambda}[\![\lambda_{\mathsf{x}}]\!]\ \chi\ \overline{P} &\triangleq (\!|\ \overline{P}(\lambda_{\mathsf{x}}) \sqcap^{¤} \chi \neq \bot^{¤} \mathrel{?} \overline{P}[\lambda_{\mathsf{x}} \leftarrow \overline{P}(\lambda_{\mathsf{x}}) \sqcap^{¤} \chi] \mathrel{:} \dot{\bot}^{¤}\ |\!)\\
\mathscr{A}^{¤-1}_{\lambda}[\![\mathsf{A}_1 - \mathsf{A}_2]\!]\ \chi\ \overline{P} &\triangleq \mathsf{let}\ \langle \chi_1,\ \chi_2\rangle = \ominus^{¤-1}_{\lambda}\ \chi\ \langle \mathscr{A}^{¤}_{\lambda}[\![\mathsf{A}_1]\!]\ \overline{P},\ \mathscr{A}^{¤}_{\lambda}[\![\mathsf{A}_2]\!]\ \overline{P}\rangle\ \mathsf{in}\\
&\qquad \mathscr{A}^{¤-1}_{\lambda}[\![\mathsf{A}_1]\!]\ \chi_1\ \overline{P}\ \dot{\sqcap}^{¤}\ \mathscr{A}^{¤-1}_{\lambda}[\![\mathsf{A}_2]\!]\ \chi_2\ \overline{P}
\end{aligned}
$$

where $\ominus^{¤-1}_{\lambda}\ P\ \langle Q_1,\ Q_2\rangle \mathrel{\dot{\triangleq}}^{¤}_{\lambda} \langle \alpha^{¤}_{\lambda}(\{v_1 \in \gamma^{¤}_{\lambda}(Q_1) \mid \exists v_2 \in \gamma^{¤}_{\lambda}(Q_2)\ .\ (v_1 - v_2) \in \gamma^{¤}_{\lambda}(P)\}),$
$\alpha^{¤}_{\lambda}(\{v_2 \in \gamma^{¤}_{\lambda}(Q_2) \mid \exists v_1 \in \gamma^{¤}_{\lambda}(Q_1)\ .\ (v_1 - v_2) \in \gamma^{¤}_{\lambda}(P)\})\rangle$

$$\mathscr{A}^{\underline{¤}_1}_{\lambda}[\![\mathsf{NULL}]\!]\,\chi\,\overline{P} \triangleq (\!|\, \mathsf{NULL}^{¤}_{\lambda} \sqcap^{¤} \chi \neq \bot^{¤} \;?\; \overline{P} \;:\; \dot{\bot}^{¤} \,|\!)$$

$$\mathscr{A}^{\underline{¤}_1}_{\lambda}[\![\&\mathsf{x}]\!]\,\chi\,\overline{P} \triangleq (\!|\, \boldsymbol{\lambda}^{¤}_{\lambda}[\![\lambda_{\mathsf{x}}]\!] \sqcap^{¤} \chi \neq \bot^{¤} \;?\; \overline{P}[\lambda_{\mathsf{x}} \leftarrow \boldsymbol{\lambda}^{¤}_{\lambda}[\![\mathsf{x}]\!] \sqcap^{¤} \chi] \;:\; \dot{\bot}^{¤} \,|\!)$$

$$\mathscr{A}^{\underline{¤}_1}_{\lambda}[\![\star\mathsf{p}]\!]\,\chi\,\overline{P} \triangleq \star^{\underline{¤}_1}[\![\mathsf{p}]\!]\,\lambda\,\chi\,\overline{P}$$

$$\text{where } \star^{\underline{¤}_1}[\![\mathsf{p}]\!]\,\lambda\,\chi\,\overline{P} \mathrel{\dot{\sqsupseteq}^{¤}_{\lambda}} \lambda_{\mathsf{z}} \mapsto \alpha^{¤}_{\lambda}(\{\mu(\lambda_{\mathsf{z}}) \in \gamma^{¤}_{\lambda}(\overline{P}(\lambda_{\mathsf{z}})) \mid \mu(\mu(\lambda_{\mathsf{p}})) \in \chi \cap \gamma^{¤}_{\lambda}(\overline{P}(\mu(\lambda_{\mathsf{p}})))\})$$

Remark 46.26

- By choosing $\alpha^{¤}_{\lambda}$ to be the identity, we get the definitions of
  - $\dot{\sqsubseteq}^{\times} \triangleq \dot{\subseteq}$,
  - $\ominus^{¤-1}\ P\ \langle Q_1,\ Q_2 \rangle \triangleq \langle \{v_1 \in Q_1 \mid \exists v_2 \in Q_2\ .\ (v_1 - v_2) \in P\}$, $\{v_2 \in Q_2 \mid \exists v_1 \in Q_1\ .\ (v_1 - v_2) \in P\}\rangle$ as an instance of $\ominus^{¤-1}_{\lambda}$, and
  - $\mathscr{A}^{\times-1}_{\lambda}[\![\mathsf{A}]\!]$ as an instance of $\mathscr{A}^{¤-1}_{\lambda}[\![\mathsf{A}]\!]$

  in (46.25).
- Then $\mathscr{A}^{\times-1}_{\lambda}[\![\mathsf{A}]\!]\ \chi$ is a lower closure operator on $\mathbb{V} \longrightarrow \wp(\mathbb{V}_{\lambda})$ and $\mathscr{A}^{\times-1}_{\lambda}[\![\mathsf{A}]\!]\ \chi$ is $\dot{\subseteq}$-increasing in χ.

theorem 28.29 becomes

Theorem (46.27) $\dot{\alpha}^{\times}_{\lambda} \circ \mathscr{A}^{\times_1}_{\lambda}[\![\mathsf{A}]\!]\,\chi \mathrel{\dot{\subseteq}} \mathscr{A}^{\times_1}_{\lambda}[\![\mathsf{A}]\!]\,\chi \circ \dot{\alpha}^{\times}_{\lambda}$.

For the pointer semantics, theorem 28.31 becomes

Theorem (46.29) If $\gamma^{¤}_{\lambda} \in \langle \mathbb{P}^{¤}_{\lambda}, \sqsubseteq^{¤}\rangle \xrightarrow{\sqcap} \langle \mathbb{P}^{\times}, \sqsubseteq^{\times}\rangle$ preserves finite meets, is strict ($\gamma^{¤}_{\lambda}(\bot^{¤}) = \varnothing$), $1 \in \gamma^{¤}_{\lambda}(1^{¤})$, the hypotheses of theorem 46.24 hold, $(\ominus^{\underline{\times}_1}\ \chi)$ is argumentwise increasing, and $\ominus^{\underline{\times}_1}\ \gamma^{¤}_{\lambda}(\chi)\ \langle\gamma^{¤}_{\lambda}(P_1),\ \gamma^{¤}_{\lambda}(P_2)\rangle \mathrel{\dot\subseteq} \dot\gamma_{¤}(\ominus^{\underline{¤}_1}_{\lambda}\ \chi\ \langle P_1,\ P_2\rangle)$ then

$$\mathscr{A}^{\underline{\times}_1}_{\lambda}[\![\mathsf{A}]\!]\ \gamma^{¤}_{\lambda}(\chi) \circ \dot\gamma_{¤} \quad \dot\subseteq \quad \dot\gamma_{¤} \circ \mathscr{A}^{\underline{¤}_1}_{\lambda}[\![\mathsf{A}]\!]\ \chi.$$

Cartesian forward reachability of tests

For tests, (28.38) is completed by pointer equality tests.

Cartesian abstract forward reachability semantics of an arithmetic or pointer boolean expressions

$$
\begin{aligned}
\mathsf{test}^{¤}, \overline{\mathsf{test}}^{¤} &\in \lambda \in \mathbb{L} \longrightarrow \mathbb{B} \longrightarrow \dot{\mathbb{P}}^{¤}_{\lambda} \longrightarrow \dot{\mathbb{P}}^{¤}_{\lambda} \quad \text{where} \quad \dot{\mathbb{P}}^{¤}_{\lambda} \triangleq \mathbb{L}_{\lambda} \longrightarrow \mathbb{P}^{¤}_{\lambda} && (46.30)\\
\mathsf{test}^{¤}_{\lambda}[\![\mathsf{p} == \mathsf{NULL}]\!]\,\overline{P} &\triangleq (\!|\, \overline{P}(\lambda_{\mathsf{p}}) \sqcap^{¤} \mathsf{NULL}^{¤}_{\lambda} \neq \mathsf{NULL}^{¤}_{\lambda} \;?\; \dot{\bot}^{¤} \;:\; \overline{P}[\lambda_{\mathsf{p}} \leftarrow \mathsf{NULL}^{¤}_{\lambda}] \,|\!)\\
\overline{\mathsf{test}}^{¤}_{\lambda}[\![\mathsf{p} == \mathsf{NULL}]\!]\,\overline{P} &\triangleq (\!|\, \overline{P}(\lambda_{\mathsf{p}}) \sqsubseteq^{¤} \mathsf{NULL}^{¤}_{\lambda} \;?\; \dot{\bot}^{¤} \;:\; \overline{P} \,|\!)\\
\mathsf{test}^{¤}_{\lambda}[\![\mathsf{x} == \mathsf{y}]\!]\,\overline{P} &\triangleq \mathsf{let}\; a = \overline{P}(\lambda_{\mathsf{x}}) \sqcap^{¤} \overline{P}(\lambda_{\mathsf{y}}) \;\mathsf{in}\; (\!|\, a = \bot^{¤} \;?\; \dot{\bot}^{¤} \;:\; \overline{P}[\lambda_{\mathsf{x}} \leftarrow a][\lambda_{\mathsf{y}} \leftarrow a] \,|\!)\\
\overline{\mathsf{test}}^{¤}_{\lambda}[\![\mathsf{x} == \mathsf{y}]\!]\,\overline{P} &\triangleq (\!|\, \overline{P}(\lambda_{\mathsf{x}}) \sqsubseteq^{¤} \mathsf{NULL}^{¤}_{\lambda} \wedge \overline{P}(\lambda_{\mathsf{y}}) \sqsubseteq^{¤} \mathsf{NULL}^{¤}_{\lambda} \;?\; \dot{\bot}^{¤} \;:\; \overline{P} \,|\!)
\end{aligned}
$$

Remark 46.31

Following *Remark* 46.20, by choosing $\alpha^{¤}_{\lambda}$ to be the identity, we get $\mathsf{test}^{\times} \in \mathbb{L}_{\lambda} \to \mathcal{B} \to (\mathbb{L}_{\lambda} \to \wp(\mathbb{V}_{\lambda})) \to (\mathbb{L}_{\lambda} \to \wp(\mathbb{V}_{\lambda}))$ where $\mathsf{test}^{\times}_{\lambda}[\![\mathsf{B}]\!]/\overline{\mathsf{test}}^{\times}_{\lambda}[\![\mathsf{B}]\!]$ is the instance of $\mathsf{test}^{¤}_{\lambda}[\![\mathsf{B}]\!]/\overline{\mathsf{test}}^{¤}_{\lambda}[\![\mathsf{B}]\!]$ with

- $\dot{\mathbb{P}}^{¤} \triangleq \wp(\mathbb{L} \to \mathbb{V}_{\lambda})$,
- $\perp^{\times} \triangleq \emptyset$,
- $\dot{\perp}^{\times} \triangleq \dot{\emptyset}$,
- $\sqcap^{¤} \triangleq \cap$,
- $\sqsubseteq^{¤} \triangleq \subseteq$, and
- $\mathtt{NULL}^{\times} \triangleq \{\mathtt{nil}\}$.

theorem 28.35 is extended as

Theorem (46.32, cartesian test)

$$\dot{\alpha}^{\times}_{\lambda} \circ \mathsf{test}^{\vec{r}}_{\lambda}[\![\mathsf{B}]\!] \mathrel{\dot{\subseteq}} \mathsf{test}^{\times}_{\lambda}[\![\mathsf{B}]\!] \circ \dot{\alpha}^{\times}_{\lambda}$$

and

$$\dot{\alpha}^{\times}_{\lambda} \circ \overline{\mathsf{test}}^{\vec{r}}_{\lambda}[\![\mathsf{B}]\!] \mathrel{\dot{\subseteq}} \overline{\mathsf{test}}^{\times}_{\lambda}[\![\mathsf{B}]\!] \circ \dot{\alpha}^{\times}_{\lambda}$$

With static pointers, theorem 28.39 on abstract cartesian tests becomes

Theorem (46.33, abstract cartesian test) If

- $\olessthan^{\times}$ is argumentwise increasing,
- $\olessthan^{\times} \langle \gamma_{\lambda}^{\text{¤}}(X),\ \gamma_{\lambda}^{\text{¤}}(Y) \rangle \mathrel{\dot{\sqsubseteq}^{\times}} \dot{\gamma}_{\text{¤}}(\olessthan^{\text{¤}} \langle X,\ Y \rangle))$, and
- $\gamma_{\lambda}^{\text{¤}}$ preserves finite glbs

then

$$\mathsf{test}_{\lambda}^{\times}[\![\mathsf{B}]\!] \circ \dot{\gamma}_{\text{¤}} \mathrel{\dot{\sqsubseteq}^{\times}} \dot{\gamma}_{\text{¤}} \circ \mathsf{test}_{\lambda}^{\text{¤}}[\![\mathsf{B}]\!]$$

and

$$\overline{\mathsf{test}}_{\lambda}^{\times}[\![\mathsf{B}]\!] \circ \dot{\gamma}_{\text{¤}} \mathrel{\dot{\sqsubseteq}^{\times}} \dot{\gamma}_{\text{¤}} \circ \overline{\mathsf{test}}_{\lambda}^{\text{¤}}[\![\mathsf{B}]\!]$$

The soundness proof theorem 28.44 can be extended as follows.

Theorem (46.34) Well-definedness and soundness of the cartesian static analysis
The cartesian abstract domain $\mathbb{D}^{\dot{¤}}$, pointwise extension of $\mathbb{D}^{¤}$ in (46.18) for $\mathbb{L}$ is well-defined according to definition 21.1 so the flow-sensitive abstract interpreter $\widehat{\mathcal{S}}^{\dot{¤}}[\![\mathsf{S}]\!]$ and the flow-insensitive abstract interpreter $\widehat{\mathcal{S}}^{g¤}_{\lambda}[\![\mathsf{S}]\!]$ are well-defined and a sound abstraction of the assertional forward reachability semantics $\mathcal{S}^{\vec{r}}[\![\mathsf{S}]\!]$ of section 46.4 for any program component $\mathsf{S} \in \mathbb{P}\mathbb{c}$.