Principles of Abstract Interpretation
MIT press
Ch. **4**, Syntax

Patrick Cousot

pcousot.github.io

PrAbsInt@gmail.com   github.com/PrAbsInt/

These slides are available at
http://github.com/PrAbsInt/slides/slides/slides-04--syntax-PrAbsInt.pdf

What did we learned in Ch. **3**, Syntax, semantics, properties, and static analysis of expressions?

- To design a program analysis (*e.g*. signs of expressions):
  (1) Define the syntax of programs (*e.g*. expressions)
  (2) Define the semantics of programs (*e.g*. what expressions compute)
  (3) The collecting semantics of programs is the strongest property of the program semantics
  (4) Define the abstraction of semantic program properties by a Galois connection (*e.g*. sign abstraction of value, environment, and semantic properties)
  (5) Design the abstract semantics by calculational design of an abstraction of the collecting semantics
  $\Rightarrow$ the abstract semantics specifies a static analyzer which is sound by construction
- These slides consider (1) for a mini-language (a subset of C).

Objective

The objective of this Chapter **4** (Syntax) is to introduce the syntax of the small subset of the C programming language used throughout the book to exemplify abstract interpretation

en.wikipedia.org/wiki/C_(programming_language)
en.wikipedia.org/wiki/Syntax_(programming_languages)

Chapter 4

Ch. **4**, Syntax

Syntax of programs

Syntax of statements

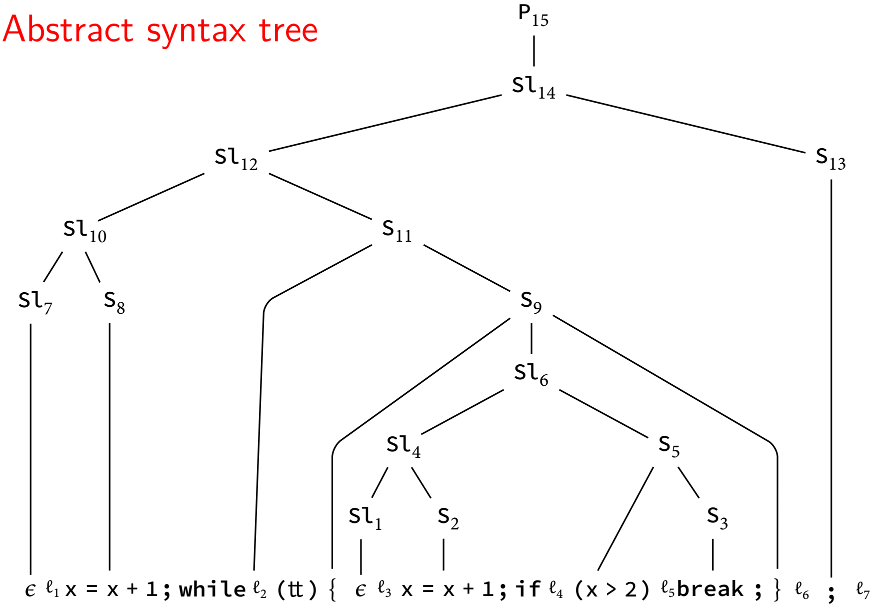
- A subset of C

$$
\begin{array}{lcll}
\texttt{S} & ::= & & \text{statement } \texttt{S} \in \mathbb{S} \\
& & \texttt{x = E ;} & \quad \text{assignment} \\
& \mid & \texttt{;} & \quad \text{skip} \\
& \mid & \texttt{if (B) S} & \quad \text{conditional} \\
& \mid & \texttt{if (B) S else S} & \\
& \mid & \texttt{while (B) S} & \quad \text{iteration} \\
& \mid & \texttt{break ;} & \quad \text{iteration break (to inner enclosing loop exit)} \\
& \mid & \texttt{\{ Sl \}} & \quad \text{compound statement} \\
\texttt{Sl} & ::= & \texttt{Sl S} \mid \epsilon & \text{statement list } \texttt{Sl} \in \mathbb{Sl}
\end{array}
$$

- Example:

$$\texttt{x = x + 1 ; while (tt) \{ x = x + 1 ;if (x > 2) break ; \};} \tag{4.1}$$

Abstract syntax tree



en.wikipedia.org/wiki/Abstract_syntax en.wikipedia.org/wiki/Abstract_syntax_tree

Syntax of programs

$$\mathtt{P} \quad ::= \quad \mathtt{Sl} \qquad\qquad \text{program } \mathtt{P} \in \mathbb{P}$$

$$\mathbb{P}c \quad \triangleq \quad \mathbb{S} \cup \mathbb{S}\mathbb{I} \cup \mathbb{P} \qquad\qquad \text{program component } \mathtt{S} \in \mathbb{P}c$$

Labels of programs

Labels

- Labels are unique and designate program points of statements
- Labels are not part of the program syntax, their syntax is free
- Example: labelling of program (4.1):

$$\ell_1\ \mathtt{x = x + 1\ ;}\quad \mathtt{while}\ \ell_2\ (\mathrm{tt})\ \{\quad \ell_3\ \mathtt{x = x + 1\ ;}\quad \mathtt{if}\ \ell_4\ (\mathtt{x > 2})\ \ell_5\ \mathtt{break\ ;}\}\ell_6\mathtt{;}\ell_7 \tag{4.5}$$

break ; statements are used to exit iteration statements only (so the **break ;** at ℓ_5 exists to ℓ_6).

`en.wikipedia.org/wiki/Control_flow#Early_exit_from_loops`
`en.wikipedia.org/wiki/Label_(computer_science)`

Labels

$\mathsf{at}[\![\mathtt{S}]\!]$	the program point at which execution of program component $\mathtt{S}$ starts;
$\mathsf{after}[\![\mathtt{S}]\!]$	the program exit point after program component $\mathtt{S}$, at which execution of $\mathtt{S}$ is supposed to normally terminate, if ever;
$\mathsf{escape}[\![\mathtt{S}]\!]$	a boolean indicating whether or not the program component $\mathtt{S}$ contains a **break ;** statement escaping out of that component $\mathtt{S}$;
$\mathsf{break\text{-}to}[\![\mathtt{S}]\!]$	the program point at which execution of the program component $\mathtt{S}$ goes to when a **break ;** statement escapes out of that component $\mathtt{S}$;
$\mathsf{breaks\text{-}of}[\![\mathtt{S}]\!]$	the set of labels of all **break ;** statements that can escape out of $\mathtt{S}$
$\mathsf{in}[\![\mathtt{S}]\!]$	the set of program points inside program component $\mathtt{S}$ (including $\mathsf{at}[\![\mathtt{S}]\!]$ but excluding $\mathsf{after}[\![\mathtt{S}]\!]$ and $\mathsf{break\text{-}to}[\![\mathtt{S}]\!]$);
$\mathsf{labx}[\![\mathtt{S}]\!]$	the potentially reachable program points while executing program component $\mathtt{S}$ either at, in, or after the program component, or resulting from a break.

Label at a statement S

- $\mathsf{at}[\![\mathtt{S}]\!]$: the program point at which execution of S starts;

$$\begin{array}{rcll}
\mathtt{P} & ::= & \mathtt{Sl} & \mathsf{at}[\![\mathtt{P}]\!] \triangleq \mathsf{at}[\![\mathtt{Sl}]\!] \\
\mathtt{Sl} & ::= & \mathtt{Sl}'\ \mathtt{S} & \mathsf{at}[\![\mathtt{Sl}]\!] \triangleq \mathsf{at}[\![\mathtt{Sl}']\!] \\
\mathtt{Sl} & ::= & \epsilon & \mathsf{at}[\![\mathtt{Sl}]\!] \triangleq \mathsf{after}[\![\mathtt{Sl}]\!] \\
\mathtt{S} & ::= & \{\ \mathtt{Sl}\ \} & \mathsf{at}[\![\mathtt{S}]\!] \triangleq \mathsf{at}[\![\mathtt{Sl}]\!]
\end{array}$$

Label after a statement S

- after⟦S⟧: the program point at which execution will continue upon termination of the statement S (unless there is a **break ;**);

$$
\begin{array}{rcll}
\mathtt{P} & ::= & \mathtt{Sl} & \mathsf{after}[\![\mathtt{P}]\!] \triangleq \mathsf{after}[\![\mathtt{Sl}]\!] \\
\mathtt{Sl} & ::= & \mathtt{Sl}'\ \mathtt{S} & \mathsf{after}[\![\mathtt{Sl}']\!] \triangleq \mathsf{at}[\![\mathtt{S}]\!], \quad \mathsf{after}[\![\mathtt{S}]\!] \triangleq \mathsf{after}[\![\mathtt{Sl}]\!] \\
\mathtt{S} & ::= & \mathbf{if}\ (\mathtt{B})\ \mathtt{S}_t & \mathsf{after}[\![\mathtt{S}_t]\!] \triangleq \mathsf{after}[\![\mathtt{S}]\!] \\
\mathtt{S} & ::= & \mathbf{if}\ (\mathtt{B})\ \mathtt{S}_t\ \mathbf{else}\ \mathtt{S}_f & \mathsf{after}[\![\mathtt{S}_t]\!] \triangleq \mathsf{after}[\![\mathtt{S}_f]\!] \triangleq \mathsf{after}[\![\mathtt{S}]\!] \\
\mathtt{S} & ::= & \mathbf{while}\ (\mathtt{B})\ \mathtt{S}_b & \mathsf{after}[\![\mathtt{S}_b]\!] \triangleq \mathsf{at}[\![\mathtt{S}]\!] \\
\mathtt{S} & ::= & \{\ \mathtt{Sl}\ \} & \mathsf{after}[\![\mathtt{Sl}]\!] \triangleq \mathsf{after}[\![\mathtt{S}]\!]
\end{array}
$$

Explicit labelling

- When explicitly decorating programs with labels, we should have

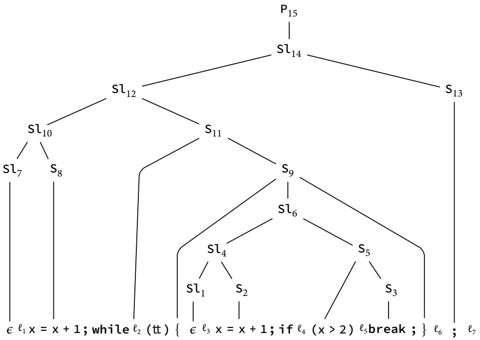
$$
\begin{array}{lcll}
\mathtt{S} & ::= & \ell\ \mathtt{x = E ;} & \mathsf{at}[\![\mathtt{S}]\!] \triangleq \ell \\
\mathtt{S} & ::= & \ell\mathtt{;} & \mathsf{at}[\![\mathtt{S}]\!] \triangleq \ell \\
\mathtt{S} & ::= & \mathtt{if}\ \ell\ \mathtt{(B)}\ \mathtt{S}_t & \mathsf{at}[\![\mathtt{S}]\!] \triangleq \ell \\
\mathtt{S} & ::= & \mathtt{if}\ \ell\ \mathtt{(B)}\ \mathtt{S}_t\ \mathtt{else}\ \mathtt{S}_f & \mathsf{at}[\![\mathtt{S}]\!] \triangleq \ell \\
\mathtt{S} & ::= & \mathtt{while}\ \ell\ \mathtt{(B)}\ \mathtt{S}_b & \mathsf{at}[\![\mathtt{S}]\!] \triangleq \ell \\
\mathtt{S} & ::= & \ell\ \mathtt{break ;} & \mathsf{at}[\![\mathtt{S}]\!] \triangleq \ell \\
\mathtt{P} & ::= & \mathtt{Sl}\ \ell & \mathsf{after}[\![\mathtt{P}]\!] \triangleq \mathsf{after}[\![\mathtt{Sl}]\!] \triangleq \ell
\end{array}
$$

- "In ℓ_1 `x = E ;`ℓ_2, execution starts at ℓ_1 and continues at ℓ_2" is a shorthand for 'In `x = E ;`, execution starts at $\mathsf{at}[\![\mathtt{x = E ;}]\!]$ and continues at $\mathsf{after}[\![\mathtt{x = E ;}]\!]$"

- The program labelling is arbitrary and can change and be generated automatically

Another choice for labels

- A label can be a path in the abstract syntax tree
- Example:



$$\ell_3 \quad = \quad \mathtt{P}_{15}\ \mathtt{Sl}_{14}\ \mathtt{Sl}_{12}\ \mathtt{S}_{11}\ \mathtt{S}_{9}\ \mathtt{Sl}_{6}\ \mathtt{Sl}_{4}\ \mathtt{S}_{2}$$

Escaping a statement S

- $\mathsf{escape}[\![\mathtt{S}]\!]$: true ($\mathsf{tt}$) if and only if the statement S contains a **break ;** that can escape out of that S;

P	::=	Sl	$\mathsf{escape}[\![\mathtt{P}]\!] \triangleq \mathsf{escape}[\![\mathtt{Sl}]\!]$	$\mathsf{escape}[\![\mathtt{P}]\!] = \mathsf{ff}$
Sl	::=	Sl′ S	$\mathsf{escape}[\![\mathtt{Sl}]\!] \triangleq \mathsf{escape}[\![\mathtt{Sl}']\!] \vee \mathsf{escape}[\![\mathtt{S}]\!]$	
Sl	::=	ϵ	$\mathsf{escape}[\![\mathtt{Sl}]\!] \triangleq \mathsf{ff}$	
S	::=	x = E ;	$\mathsf{escape}[\![\mathtt{S}]\!] \triangleq \mathsf{ff}$	
S	::=	;	$\mathsf{escape}[\![\mathtt{S}]\!] \triangleq \mathsf{ff}$	
S	::=	**if** (B) $\mathtt{S}_t$	$\mathsf{escape}[\![\mathtt{S}]\!] \triangleq \mathsf{escape}[\![\mathtt{S}_t]\!]$	
S	::=	**if** (B) $\mathtt{S}_t$ **else** $\mathtt{S}_f$	$\mathsf{escape}[\![\mathtt{S}]\!] \triangleq \mathsf{escape}[\![\mathtt{S}_t]\!] \vee \mathsf{escape}[\![\mathtt{S}_f]\!]$	
S	::=	**while** (B) $\mathtt{S}_b$	$\mathsf{escape}[\![\mathtt{S}]\!] \triangleq \mathsf{ff}$	
S	::=	**break ;**	$\mathsf{escape}[\![\mathtt{S}]\!] \triangleq \mathsf{tt}$	
S	::=	{ Sl }	$\mathsf{escape}[\![\mathtt{S}]\!] \triangleq \mathsf{escape}[\![\mathtt{Sl}]\!]$	

Break label of a statement $\mathtt{S}$

- break-to$[\![\mathtt{S}]\!]$: the program point at which execution of $\mathtt{S}$ goes to when a **break ;** statement is executed while executing $\mathtt{S}$;

$\mathtt{Sl}$	$::=$	$\mathtt{Sl}'\ \mathtt{S}$	$\text{break-to}[\![\mathtt{Sl}']\!] \triangleq \text{break-to}[\![\mathtt{S}]\!] \triangleq \text{break-to}[\![\mathtt{Sl}]\!]$
$\mathtt{S}$	$::=$	$\mathtt{if}\ (\mathtt{B})\ \mathtt{S}_t$	$\text{break-to}[\![\mathtt{S}_t]\!] \triangleq \text{break-to}[\![\mathtt{S}]\!]$
$\mathtt{S}$	$::=$	$\mathtt{if}\ (\mathtt{B})\ \mathtt{S}_t\ \mathtt{else}\ \mathtt{S}_f$	$\text{break-to}[\![\mathtt{S}_t]\!] \triangleq \text{break-to}[\![\mathtt{S}_f]\!] \triangleq \text{break-to}[\![\mathtt{S}]\!]$
$\mathtt{S}$	$::=$	$\mathtt{while}\ (\mathtt{B})\ \mathtt{S}_b$	$\text{break-to}[\![\mathtt{S}_b]\!] \triangleq \text{after}[\![\mathtt{S}]\!]$
$\mathtt{S}$	$::=$	$\{\ \mathtt{Sl}\ \}$	$\text{break-to}[\![\mathtt{Sl}]\!] \triangleq \text{break-to}[\![\mathtt{S}]\!]$

Break label of a statement S

- breaks-of⟦S⟧: collects the labels of all **break ;** statements that can escape out of S (so excluding **break ;** statements inside an iteration statement within S).
- The definition checks that **break ;** statements can only appear within loops;

P	::=	Sl	breaks-of⟦P⟧ ≜ breaks-of⟦Sl⟧	breaks-of⟦P⟧ = ∅
Sl	::=	Sl′ S	breaks-of⟦Sl⟧ ≜ breaks-of⟦Sl′⟧ ∪ breaks-of⟦S⟧	
Sl	::=	ϵ	breaks-of⟦Sl⟧ ≜ ∅	
S	::=	x = E ;	breaks-of⟦S⟧ ≜ ∅	
S	::=	;	breaks-of⟦S⟧ ≜ ∅	
S	::=	**if** (B) S_t	breaks-of⟦S⟧ ≜ breaks-of⟦S_t⟧	
S	::=	**if** (B) S_t **else** S_f	breaks-of⟦S⟧ ≜ breaks-of⟦S_t⟧ ∪ breaks-of⟦S_f⟧	
S	::=	**while** (B) S_b	breaks-of⟦S⟧ ≜ ∅	
S	::=	ℓ **break ;**	breaks-of⟦S⟧ ≜ {at⟦S⟧}	
S	::=	{ Sl }	breaks-of⟦S⟧ ≜ breaks-of⟦Sl⟧	

Labels in a statement $\texttt{S}$

- $\mathsf{in}[\![\texttt{S}]\!]$: the program points inside $\texttt{S}$ (including $\mathsf{at}[\![\texttt{S}]\!]$ but excluding $\mathsf{after}[\![\texttt{S}]\!]$ and $\mathsf{break\text{-}to}[\![\texttt{S}]\!]$)
- The definition checks that program labels are unique.

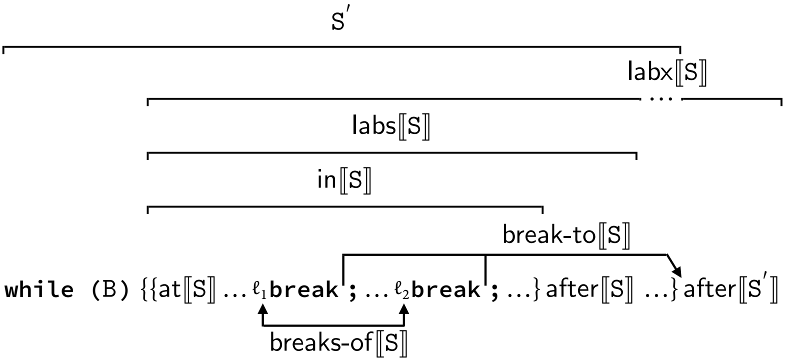
$\texttt{P}$	$::=$	$\texttt{Sl}$	$\mathsf{in}[\![\texttt{P}]\!] \triangleq \mathsf{in}[\![\texttt{Sl}]\!]$	$\mathsf{after}[\![\texttt{Sl}]\!] \notin \mathsf{in}[\![\texttt{Sl}]\!]$
$\texttt{Sl}$	$::=$	$\texttt{Sl}'\ \texttt{S}$	$\mathsf{in}[\![\texttt{Sl}]\!] \triangleq \mathsf{in}[\![\texttt{Sl}']\!] \cup \mathsf{in}[\![\texttt{S}]\!]$	$\mathsf{in}[\![\texttt{Sl}']\!] \cap \mathsf{in}[\![\texttt{S}]\!] = \varnothing$ when $\texttt{Sl}' \neq \texttt{\{} \ldots \texttt{\{}\ \epsilon\ \texttt{\}} \ldots \texttt{\}}$
$\texttt{Sl}$	$::=$	ϵ	$\mathsf{in}[\![\texttt{Sl}]\!] \triangleq \{\mathsf{at}[\![\texttt{Sl}]\!]\}$	
$\texttt{S}$	$::=$	$\texttt{x = E ;}$	$\mathsf{in}[\![\texttt{S}]\!] \triangleq \{\mathsf{at}[\![\texttt{S}]\!]\}$	
$\texttt{S}$	$::=$	$\texttt{;}$	$\mathsf{in}[\![\texttt{S}]\!] \triangleq \{\mathsf{at}[\![\texttt{S}]\!]\}$	
$\texttt{S}$	$::=$	$\texttt{if (B) S}_t$	$\mathsf{in}[\![\texttt{S}]\!] \triangleq \{\mathsf{at}[\![\texttt{S}]\!]\} \cup \mathsf{in}[\![\texttt{S}_t]\!]$	$\mathsf{at}[\![\texttt{S}]\!] \notin \mathsf{in}[\![\texttt{S}_t]\!]$
$\texttt{S}$	$::=$	$\texttt{if (B) S}_t\ \texttt{else S}_f$	$\mathsf{in}[\![\texttt{S}]\!] \triangleq \{\mathsf{at}[\![\texttt{S}]\!]\} \cup \mathsf{in}[\![\texttt{S}_t]\!] \cup \mathsf{in}[\![\texttt{S}_f]\!]$	$\mathsf{at}[\![\texttt{S}]\!] \notin \mathsf{in}[\![\texttt{S}_t]\!] \cup \mathsf{in}[\![\texttt{S}_f]\!]$ $\mathsf{in}[\![\texttt{S}_t]\!] \cap \mathsf{in}[\![\texttt{S}_f]\!] = \varnothing$
$\texttt{S}$	$::=$	$\texttt{while (B) S}_b$	$\mathsf{in}[\![\texttt{S}]\!] \triangleq \{\mathsf{at}[\![\texttt{S}]\!]\} \cup \mathsf{in}[\![\texttt{S}_b]\!]$	$\mathsf{at}[\![\texttt{S}]\!] \notin \mathsf{in}[\![\texttt{S}_b]\!]$
$\texttt{S}$	$::=$	$\texttt{break ;}$	$\mathsf{in}[\![\texttt{S}]\!] \triangleq \{\mathsf{at}[\![\texttt{S}]\!]\}$	
$\texttt{S}$	$::=$	$\texttt{\{ Sl \}}$	$\mathsf{in}[\![\texttt{S}]\!] \triangleq \mathsf{in}[\![\texttt{Sl}]\!]$	

Labels of a statement S

- labs⟦S⟧: the potentially reachable program points while executing S either in or after the statement;
- labx⟦S⟧: the potentially reachable program points while executing S either in or after the statement, or resulting from a break.

$$
\begin{array}{lcl}
\text{labs}[\![S]\!] & \triangleq & \text{in}[\![S]\!] \cup \{\text{after}[\![S]\!]\} \\
\text{labx}[\![S]\!] & \triangleq & \text{labs}[\![S]\!] \cup \left(\!\left|\ \text{escape}[\![S]\!] \ ?\ \{\text{break-to}[\![S]\!]\} \ :\ \varnothing \ \right|\!\right)
\end{array}
$$

Informal illustration of the definitions



Example of program labelling (home work)

The program $\texttt{P} = \texttt{while}\ \ell_1\ (\mathtt{tt})\ \ell_2\ \texttt{break ;}\ \ell_3\ \texttt{x = 7 ;}\ell_4$ has grammatical structure

$$\texttt{P}\left[\begin{array}{l} \texttt{Sl}_1 \left[\begin{array}{l} \texttt{Sl}_2\left[\begin{array}{l}\texttt{Sl}_3\left[\ \epsilon\right.\\ \texttt{S}_4\left[\begin{array}{l}\texttt{while}\ \ell_1\ (\mathtt{tt})\\ \texttt{S}_5\left[\ \ell_2\ \texttt{break ;}\right.\end{array}\right.\end{array}\right.\\ \texttt{S}_6\left[\ \ell_3\ \texttt{x = 7 ;}\right.\end{array}\right.\\ \ell_4\end{array}\right.$$

The labelling is

S	at⟦S⟧	in⟦S⟧	escape⟦S⟧	break-to⟦S⟧	after⟦S⟧	labs⟦S⟧	labx⟦S⟧
P	ℓ_1	ℓ_1, ℓ_2, ℓ_3	ff	—	ℓ_4	$\ell_1, \ell_2, \ell_3, \ell_4$	$\ell_1, \ell_2, \ell_3, \ell_4$
$\texttt{Sl}_1$	ℓ_1	ℓ_1, ℓ_2, ℓ_3	ff	—	ℓ_4	$\ell_1, \ell_2, \ell_3, \ell_4$	$\ell_1, \ell_2, \ell_3, \ell_4$
$\texttt{Sl}_2$	ℓ_1	ℓ_1, ℓ_2	ff	—	ℓ_3	ℓ_1, ℓ_2, ℓ_3	ℓ_1, ℓ_2, ℓ_3
$\texttt{Sl}_3$	ℓ_1	ℓ_1	ff	—	ℓ_1	ℓ_1	ℓ_1
$\texttt{S}_4$	ℓ_1	ℓ_1, ℓ_2	ff	—	ℓ_3	ℓ_1, ℓ_2, ℓ_3	ℓ_1, ℓ_2, ℓ_3
$\texttt{S}_5$	ℓ_2	ℓ_2	tt	ℓ_3	ℓ_1	ℓ_1, ℓ_2	ℓ_1, ℓ_2, ℓ_3
$\texttt{S}_6$	ℓ_3	ℓ_3	ff	—	ℓ_4	ℓ_3, ℓ_4	ℓ_3, ℓ_4

□

Properties of the program labelling

Lemma (4.15) For all program components S of a program P, $\mathsf{at}[\![\mathsf{S}]\!] \in \mathsf{in}[\![\mathsf{S}]\!]$.

Lemma (4.16) For all program non-empty components $\mathsf{S} \neq \{ \ldots\{ \ \epsilon \ \}\ldots \}$ of a program P, $\mathsf{after}[\![\mathsf{S}]\!] \notin \mathsf{in}[\![\mathsf{S}]\!]$.

Lemma (4.17) For all program components S of a program P, $\mathsf{escape}[\![\mathsf{S}]\!] \Rightarrow (\mathsf{break\text{-}to}[\![\mathsf{S}]\!] \notin \mathsf{in}[\![\mathsf{S}]\!])$.

Lemma (4.18) For all program components S of a program P, $\mathsf{escape}[\![\mathsf{S}]\!] \Rightarrow (\mathsf{break\text{-}to}[\![\mathsf{S}]\!] \neq \mathsf{after}[\![\mathsf{S}]\!])$.

Proofs in the book.

Home work

Read Ch. **4** "Syntax" of

Principles of Abstract Interpretation
Patrick Cousot
MIT Press

The End, Thank you