Collecting Semantics

Patrick Cousot
Jerome C. Hunsaker Visiting Professor
Massachusetts Institute of Technology
Department of Aeronautics and Astronautics
cousot@mit.edu
www.mit.edu/~cousot

Course 16.399: “Abstract interpretation”
http://web.mit.edu/afs/athena.mit.edu/course/16/16.399/www/

Collecting semantics

− A program static analysis determines a property of the program executions as defined by a (so-called standard) semantics
− The so-called collecting semantics defines the strongest static property of interest
− A collecting semantics therefore defines a whole class of static analyses, all the ones that abstract/approximate it

1 Used to be called static semantics in [1], it collects information about programs.

− There is no “universal” collecting semantics, since the information collected about program runtime executions can always be refined
− Examples of collecting semantics are computation traces, transitive closure of the program transition relation, set of states/predicate transformers, forward/backward reachable states, etc.

Reference

Collecting Semantics of
Arithmetic Expressions

Definition of the forward collecting semantics
of arithmetic expressions

The forward/bottom-up collecting semantics of an arithmetic expression defines the possible values that the arithmetic expression can evaluate to in a given set of environments.\(^2\)

\[
\text{Faexp} \in \text{Aexp} \mapsto \varphi(\mathbb{R}) \xrightarrow{\text{cm}} \varphi(\mathcal{I}_\Omega),
\]

\[
\text{Faexp}[A] R \triangleq \{ v : \exists \rho \in R : \rho \vdash A \Rightarrow v \}.
\quad (1)
\]

Reference

\(^2\) The forward collecting semantics \(\text{Faexp}[A] R\) specifies the strongest postcondition that values of the arithmetic expression \(A\) do satisfy when this expression is evaluated on an environment satisfying the precondition \(R\). The forward collecting semantics can therefore be understood as a predicate transformer.\(^2\)

P. Cousot, 2005

Properties of the forward collecting semantics
of arithmetic expressions

The forward/bottom-up collecting semantics is a complete join morphism (denoted with \(\text{cm}\)), that is (\(\mathcal{S}\) is an arbitrary set)

\[
\text{Faexp}[A]\left( \bigcup_{k \in \mathcal{S}} R_k \right) = \bigcup_{k \in \mathcal{S}} (\text{Faexp}[A] R_k),
\]

which implies monotony (when \(\mathcal{S} = \{1, 2\}\) and \(R_1 \subseteq R_2\)) and 0-strictness (when \(\mathcal{S} = \emptyset\))

\[
\text{Faexp}[A] 0 = 0.
\]

Structural specification of the forward collecting semantics of arithmetic expressions

\[
\begin{align*}
\text{Faexp}[n] R &= \{ n \}^3 \\
\text{Faexp}[X] R &= R(X) \\
\text{Faexp}[?] R &= \emptyset \\
\text{Faexp}[u A] R &= u^C(\text{Faexp}[A] R) \\
\text{Faexp}[A_1 \land A_2] R &= b^C(\text{Faexp}[A_1], \text{Faexp}[A_2]) R \\
\end{align*}
\]

where \(R(X) \triangleq \{ \rho(x) : \rho \in R \}\)

where \(u^C(V) \triangleq \{ u(v) : v \in V \}\)

where \(b^C(P_1, P_2) R \triangleq \{ v_1 \sqcap v_2 : \exists \rho \in R : v_1 \in P_1(\{\rho\}) \land v_2 \in P_2(\{\rho\}) \}\)

\(^3\) For short, the case \(\text{Faexp}[A] 0 = 0\) is not recalled.

P. Cousot, 2005
Proof.

\[ \text{Faexp}[n]R \]
\[ \text{def} \{ v \mid \exists \rho \in R : \rho \vdash n \Rightarrow v \} \]
\[ = \{ \text{null} \} \]

\[ \text{Faexp}[x]R \]
\[ \text{def} \{ v \mid \exists \rho \in R : \rho \vdash x \Rightarrow v \} \]
\[ = \{ \rho(x) \mid \rho \in R \} \]
\[ \text{def} \ R(x) \]

\[ \text{Faexp}[?] \]
\[ \text{def} \{ v \mid \exists \rho \in R : \rho \vdash ? \Rightarrow v \} \]
\[ = \{ v \mid v \in \text{I} \} \]
\[ = \text{I} \]

Collecting Semantics of Boolean Expressions

**Definition of the forward collecting semantics of boolean expressions**

The **collecting semantics** \( \text{Cbexp}[B]R \) of a boolean expression \( B \) defines the subset of possible environments \( \rho \in R \) for which the boolean expression may evaluate to true (hence without producing a runtime error)

\[
\text{Cbexp} \in \text{Bexp} \mapsto \varphi(\mathbb{R}) \\
\text{Cbexp}[B]R \overset{\text{def}}{=} \{ \rho \in R \mid \rho \vdash B \Rightarrow \text{tt} \}. \tag{2}
\]
Structural specification of the forward collecting semantics of boolean expressions

\[
\text{Cbexp}[\text{true}] R = R \\
\text{Cbexp}[\text{false}] R = 0 \\
\text{Cbexp}[A_1 \land A_2] = \land (\text{Faexp}[A_1], \text{Faexp}[A_2]) R
\]

where \(\land (F, G) R \equiv \{\rho \in R \mid \exists v_1 \in F(\rho) \cap I : \exists v_2 \in G(\rho) \cap I : v_1 \leq v_2 = \text{tt}\}\)

\[
\text{Cbexp}[B_1 \land B_2] R = \text{Cbexp}[B_1] R \cap \text{Cbexp}[B_2] R \\
\text{Cbexp}[B_1 \mid B_2] R = (\text{Cbexp}[B_1] R \cap (\text{Cbexp}[B_2] R \cup \text{Cbexp}[T(\neg B_2)] R)) \\
\quad \cup (\text{Cbexp}[B_1] R \cap (\text{Cbexp}[B_2] R \cup \text{Cbexp}[T(\neg B_1)] R))
\]

**Proof.**

\[
\text{Cbexp}[\text{true}] R \\
\equiv \{\rho \in R \mid \rho \vdash \text{true} \Rightarrow \text{tt}\} \\
= \{\rho \mid \rho \in R\} \\
= R
\]

\[
\text{Cbexp}[\text{false}] R \\
\equiv \{\rho \in R \mid \rho \vdash \text{false} \Rightarrow \text{tt}\} \\
= \{\rho \mid \text{ff}\}
\]

\[
\text{Cbexp}[A_1 \land A_2] \\
\equiv \{\rho \in R \mid \rho \vdash A_1 \land A_2 \Rightarrow \text{tt}\} \\
= \{\rho \in R \mid \exists v_1, v_2 \in I : \rho \vdash A_1 \Rightarrow v_1 \land \rho \vdash A_2 \Rightarrow v_2 \land v_1 \leq v_2 = \text{tt}\}
\]

\[
\text{Cbexp}[B_1 \land B_2] R \\
= \{\rho \in R \mid \rho \vdash B_1 \Rightarrow w_1 \land \rho \vdash B_2 \Rightarrow w_2 \land w_2 = \text{tt}\}
\]

\[
= \{\rho \in R \mid \rho \vdash B_1 \Rightarrow \text{tt} \cap \{\rho \in R \mid \rho \vdash B_2 \Rightarrow \text{tt}\}\}
\]

\[
\text{Cbexp}[B_1 \mid B_2] R \\
= \{\rho \in R \mid \rho \vdash B_1 \Rightarrow w_1 \land \rho \vdash B_2 \Rightarrow w_2 \land w_2 = \text{tt}\}
\]

\[
= \{\rho \in R \mid \rho \vdash B_1 \Rightarrow \text{tt} \lor \rho \vdash B_2 \Rightarrow \text{tt}\}
\]

\[
\text{Avoiding the case when } B_1 \text{ holds but } B_2 \text{ yields to a runtime error, or inversely}
\]

\[
= \{\rho \in R \mid \rho \vdash B_1 \Rightarrow \text{tt} \land (\rho \vdash B_2 \Rightarrow \text{tt} \lor \rho \vdash B_2 \Rightarrow \text{ff})\} \cup \{\rho \in R \mid \rho \vdash B_2 \Rightarrow \text{tt} \land (\rho \vdash B_1 \Rightarrow \text{tt} \lor \rho \vdash B_1 \Rightarrow \text{ff})\}
\]

\[
= (\text{Cbexp}[B_1] R \cap (\text{Cbexp}[B_2] R \cup \text{Cbexp}[T(\neg B_2)] R)) \cup (\text{Cbexp}[B_2] R \cap (\text{Cbexp}[B_1] R \cup \text{Cbexp}[T(\neg B_1)] R))
\]
Small-step operation semantics of commands

Recall that in lecture 5, we have defined the transition system of a program \( P = S \); as

\[
\langle \Sigma[P], \tau[P] \rangle
\]

where \( \Sigma[P] \) is the set of program states and \( \tau[C], C \in \text{Cmp}[P] \) is the transition relation for component \( C \) of program \( P \), defined by

\[
\tau[C] \overset{\text{def}}{=} \{ \langle \ell, \rho \rangle, \langle \ell', \rho' \rangle \mid \langle \ell, \rho \rangle \xrightarrow{[C]} \langle \ell', \rho' \rangle \}
\]

(3)

(4)

- Execution starts at the program entry point with all variables uninitialized:

\[
\text{Entry}[P] \overset{\text{def}}{=} \{ \langle \text{at}_P[P], \lambda x \in \text{Var}[P] \cdot \Omega_1 \rangle \}.
\]

(5)

- Execution ends without error when control reaches the program exit point

\[
\text{Exit}[P] \overset{\text{def}}{=} \{ \text{after}_P[P] \} \times \text{Env}[P].
\]

When the evaluation of an arithmetic or boolean expression fails with a runtime error, the program execution is blocked so that no further transition is possible.

A basic result on the program transition relation is that it is not possible to jump into or out of program components \( (C \in \text{Cmp}[P]) \)

\[
\langle \langle \ell, \rho \rangle, \langle \ell', \rho' \rangle \rangle \in \tau[C] \implies \{\ell, \ell'\} \subseteq \text{inp}[C].
\]

(6)
Big-step operational semantics of commands

- The reflexive transitive closure of the transition relation $\tau[C]$ of a program component $C \in \text{Cmp}[P]$ is $\tau^*[C] \overset{\text{def}}{=} (\tau[C])^*$.
- This is called the big-step operational semantics of commands.
- $\tau^*[P]$ can be expressed compositionally (in the sense of denotational semantics, by structural induction on the program components $C \in \text{Cmp}[P]$ of program $P$).

Observe that contrary to the classical big step operational or natural semantics [3], the effect of execution is described not only from entry to exit states but also from any (intermediate) state to any subsequently reachable state. This is better adapted to our later reachability analyses.

Reference

Auxiliary definitions

For the conditional $C = \text{if } B \text{ then } S_1 \text{ else } S_f \text{ fi}$, we define

\[
\tau_B \overset{\text{def}}{=} \{ (\text{at}_{P}[C], \rho), (\text{at}_{P}[S_1], \rho) \mid \rho \vdash B \Rightarrow \top \},
\]

\[
\tau_{\neg B} \overset{\text{def}}{=} \{ (\text{at}_{P}[C], \rho), (\text{at}_{P}[S_f], \rho) \mid \rho \vdash T(\neg B) \Rightarrow \top \},
\]

\[
\tau^{\downarrow} \overset{\text{def}}{=} \{ (\text{after}_{P}[S_1], \rho), (\text{after}_{P}[C], \rho) \mid \rho \in \text{Env}[P]\},
\]

\[
\tau^{\downarrow} \overset{\text{def}}{=} \{ (\text{after}_{P}[S_f], \rho), (\text{after}_{P}[C], \rho) \mid \rho \in \text{Env}[P]\}.
\]
Starting after the “then” branch

\[ \begin{align*}
\text{in}_P[C] &= \{ \text{at}_P[C], \text{after}_P[C] \} \cup \text{in}_P[S_i] \cup \text{in}_P[S_f], \\
\{ \text{at}_P[C], \text{after}_P[C] \} \cap (\text{in}_P[S_i] \cup \text{in}_P[S_f]) &= \emptyset, \\
\text{in}_P[S_i] \cap \text{in}_P[S_f] &= \emptyset.
\end{align*} \]  

(15)

It follows that by (9) to (14), we have

\[ \tau[C] = \tau_b[C] \cup \tau_b[C]. \]

where

\[ \tau_b[C] \overset{\text{def}}{=} \tau^B \cup \tau[S_i] \cup \tau^f, \]
\[ \tau_b[C] \overset{\text{def}}{=} \tau^B \cup \tau[S_i] \cup \tau^f. \]

By the conditions (15) and (6) on labelling of the conditional command \( C \), we have \( \tau_b[C] \circ \tau_b[C] = \tau_b[C] \circ \tau_b[C] = \emptyset \) so that

\[ \tau^*[C] = (\tau_b[C])^* \cup (\tau_b[C])^*. \]  

(16)

Intuitively the steps which are repeated in the conditional must all take place in one branch or the other since it is impossible to jump from one branch into the other.

Proof. Recall from Lecture 5, that for the \( C = \text{if } B \text{ then } S_i \text{ else } S_f \text{ fi} \) (where \( \text{at}_P[C] = \ell \) and \( \text{after}_P[C] = \ell' \)), we have:

\[ \rho \vdash B \Rightarrow \Downarrow \]

(9)

\[ (\ell, \rho) \Rightarrow [\text{if } B \text{ then } S_i \text{ else } S_f \text{ fi}] \Rightarrow (\text{at}_P[S_i], \rho)^{\prime} \]

(10)

\[ \rho \vdash \neg B \Rightarrow \Downarrow \]

(11)

\[ (\ell, \rho) \Rightarrow [\text{if } B \text{ then } S_i \text{ else } S_f \text{ fi}] \Rightarrow (\text{at}_P[S_f], \rho). \]

(12)

\[ \langle \text{after}_P[S_i], \rho \rangle \Rightarrow [\text{if } B \text{ then } S_i \text{ else } S_f \text{ fi}] \Rightarrow (\ell', \rho) \]

(13)

\[ \langle \text{after}_P[S_f], \rho \rangle \Rightarrow [\text{if } B \text{ then } S_i \text{ else } S_f \text{ fi}] \Rightarrow (\ell', \rho). \]

(14)

Recall also from Lecture 5, that the labelling scheme of a conditional command \( C = \text{if } B \text{ then } S_i \text{ else } S_f \text{ fi} \in \text{Cmp}[P] \) satisfies

Assume by induction hypothesis that

\[ (\tau_b[C])^n = \tau^B \circ \tau[S_i]^{n-2} \circ \tau^f \cup \tau^B \circ \tau[S_f]^{n-1} \circ \tau^f \cup \tau[S_i]^{n}. \]  

(17)

This holds for the basis \( n = 1 \) since \( \tau[S_i]^{-1} = \emptyset \) and \( \tau[S_i]^{0} = 1_{S_i[P]} \) is the identity. For \( n \geq 1 \), we have

\[ (\tau_b[C])^{n+1} = \tau^{n+1} \circ f_i \]

(18)

\[ (\tau_b[C])^n \circ \tau_b[C] = \text{induction hypothesis} \]

(19)

\[ (\tau^B \circ \tau[S_i]^{n-2} \circ \tau^f \cup \tau^B \circ \tau[S_f]^{n-1} \circ \tau^f \cup \tau[S_i]^{n}) \circ \tau_b[C] = \tau^{n+1} \circ \tau_b[C]. \]

(20)

\[ \text{by the labelling scheme (15), (6) and the def. (9) to (14) of the possible transitions so that } \tau^f \circ \tau_b[C] = \emptyset, \text{ etc.} \]
\[
\tau^B \circ \tau[\Sigma_1]^{n-1} \circ \tau_\Sigma[\Sigma^2] \circ \tau[\Sigma_1]^{n} \circ \tau_\Sigma[\Sigma^2]
\]

=  
\{\text{def. of } \tau_\Sigma[\Sigma^2]\} \text{ and } \circ \text{ distributes over } \cup \}
\tau^B \circ \tau[\Sigma_1]^{n-1} \circ \tau^B \circ \tau[\Sigma_1]^{n-1} \circ \tau[\Sigma_1] \circ \tau^B \circ \tau[\Sigma_1]^{n} \circ \tau^B \circ \tau[\Sigma_1]^{n} \circ \tau^B \circ \tau[\Sigma_1]^{n} 
\]

=  
\{\text{by the labelling scheme (15), (6) and the def. (9) to (14) of the possible transitions so that } \tau^B \circ \tau^B = 0, \tau[\Sigma_1] \circ \tau^B, \text{etc.}\}
\tau^B \circ \tau[\Sigma_1]^{n} \circ \tau^B \circ \tau[\Sigma_1]^{n-1} \circ \tau^B \circ \tau[\Sigma_1]^{n-1} \circ \tau^B \circ \tau[\Sigma_1]^{n} \circ \tau^B

=  
\{\cup \text{ is associative and commutative and def. (17) of } (\tau_\Sigma[\Sigma^2])^{n+1}\}
(\tau_\Sigma[\Sigma^2])^{n+1}

By recurrence, (17) holds for all } n \geq 1 \text{ so that}
(\tau_\Sigma[\Sigma^2])^n
=  
\{\text{def. } t^*\}
(\tau_\Sigma[\Sigma^2])^0 \cup \bigcup_{n \geq 1} (\tau_\Sigma[\Sigma^2])^n

=  
\{\text{def. } t^* \text{ and (17)}\}

Structural big-step operational semantics:
iteration command

\[
\tau^*[(\text{while } B \text{ do } S \text{ od})] = (1_{\Sigma[P]} \cup \tau^*[S]) \circ (\tau^B \circ \tau^*[S] \circ \tau^R)^* \circ (1_{\Sigma[P]} \cup \tau^B \circ \tau^*[S] \circ \tau^R)
\]

where:
\[
\begin{align*}
\tau^B & \triangleq \{(\text{at}_P[\text{while } B \text{ do } S \text{ od}], \rho), (\text{at}_P[S], \rho) \mid \rho \vdash B \Rightarrow \top\} \\
\tau^B & \triangleq \{(\text{at}_P[\text{while } B \text{ do } S \text{ od}], \rho), (\text{after}_P[\text{while } B \text{ do } S \text{ od}], \rho) \mid \\
& \quad \rho \vdash T(-B) \Rightarrow \top\} \\
\tau^R & \triangleq \{(\text{after}_P[S], \rho), (\text{at}_P[\text{while } B \text{ do } S \text{ od}], \rho) \mid \\
& \quad \rho \in \text{Env}[P]\}
\end{align*}
\]
Auxiliary definitions

For the iteration \( C = \text{while } B \text{ do } S \text{ od} \), we define

\[
\begin{align*}
\tau^B &= \{ \langle \text{at}_P[C], \rho \rangle, \langle \text{at}_P[S], \rho \rangle \mid \rho \vdash B \Rightarrow \text{tt} \}, \\
\tau^F &= \{ \langle \text{at}_P[C], \rho \rangle, \langle \text{after}_P[C], \rho \rangle \mid \rho \vdash T(\neg B) \Rightarrow \text{tt} \}, \\
\tau^R &= \{ \langle \text{after}_P[S], \rho \rangle, \langle \text{at}_P[C], \rho \rangle \mid \rho \in \text{Env}[P] \}.
\end{align*}
\]

Proof. Recall that for the iteration \( C = \text{while } B \text{ do } S \text{ od} \) (where at\(_P[C] = \ell \), after\(_P[C] = \ell' \) and \( \ell_1, \ell_2 \in \text{in}_P[S] \)), we have defined

\[
\begin{align*}
\rho \vdash T(\neg B) &\Rightarrow \text{tt} \\
(\ell, \rho) \Rightarrow \text{while } B \text{ do } S \text{ od} &\Rightarrow (\ell', \rho), \\
\rho \vdash B &\Rightarrow \text{tt} \\
(\ell, \rho) \Rightarrow \text{while } B \text{ do } S \text{ od} &\Rightarrow \langle \text{at}_P[S], \rho \rangle, \\
(\ell_1, \rho_1) \Rightarrow \text{in}_P[S] &\Rightarrow (\ell_2, \rho_2), \\
\langle \text{after}_P[S], \rho \rangle \Rightarrow \text{while } B \text{ do } S \text{ od} &\Rightarrow (\ell, \rho).
\end{align*}
\]

Recall also from Lecture 5 that the labelling scheme of an iteration command \( C = \text{while } B \text{ do } S \text{ od} \in \text{Cmp}[P] \) satisfies

\[
\begin{align*}
\text{in}_P[C] &= \{ \text{at}_P[C], \text{after}_P[C] \} \cup \text{in}_P[S], \\
\{ \text{at}_P[C], \text{after}_P[C] \} \cap \text{in}_P[S] &= \emptyset.
\end{align*}
\]
and the basic results on the program transition relation which are
\[ \forall C \in \text{Cmp}[P] : \text{at}_P[C] \neq \text{after}_P[C]. \] (24)
and that it is not possible to jump into or out of program components \((C \in \text{Cmp}[P])\)
\[ ((\ell, \rho), (\ell', \rho')) \in \tau[C] \implies \{\ell, \ell'\} \subseteq \text{in}_P[C]. \] (25)
It follows that by (19) to (22), we have
\[ \tau[C] = \tau^B \cup \tau[S] \cup \tau^R \cup \tau^B. \] (26)
We define the composition \(\bigcirc_{i=1}^n t_i\) of relations \(t_1, \ldots, t_n\) \(^4\):
\[ \bigcirc_{i=1}^n t_i \overset{def}{=} 0, \quad \text{when } n < 0, \]
\[ \bigcirc_{i=1}^n t_i \overset{def}{=} 1_{\Sigma[P]}, \quad \text{when } n = 0, \]
\[ \bigcirc_{i=1}^n t_i \overset{def}{=} t_1 \circ \ldots \circ t_n, \quad \text{when } n > 0. \]
In order to compute \(\tau^*[C] = \bigcup_{n \geq 0} \tau[C]^n\) for the component \(C = \text{while } B \text{ do } S \text{ od of program } P\), we first compute the \(n\)-th power \(\tau[C]^n\) for \(n \geq 0\). By recurrence \(\tau[C]^0 = 1_{\Sigma[P]}, \tau[C]^1 = \tau[C] = \tau^B \cup \tau[S] \cup \tau^R \cup \tau^B\). For \(n > 1\), we have
\[ (\tau[C])^2 = \{\text{def. } t^2 = t \circ t\} \]
\[ \tau[C] \circ \tau[C] = \{\text{def. (26) of } \tau[C]\} \]
\[ (\tau^B \cup \tau[S] \cup \tau^R \cup \tau^B) \circ (\tau^B \cup \tau[S] \cup \tau^R \cup \tau^B) = \{\circ \text{ distributes over } \cup \text{ (and } \circ \text{ has priority over } \cup)\} \]
\[ \tau^B \circ \tau^B \cup \tau[S] \circ \tau^B \cup \tau^R \circ \tau^B \circ \tau^B \cup \tau[S] \circ \tau[S] \circ \tau^R \circ \tau[S] \circ \tau^B \cup \tau[S] \circ \tau^R \circ \tau^B \circ \tau[S] \circ \tau^R \circ \tau^B \circ \tau^B \cup \tau^R \circ \tau^B \circ \tau^B = \ \]
\[ \{\text{by (20) and (24)}\} \quad \tau[S] \circ \tau^B = 0, \quad \text{by (21), (20), (25) and (23)}; \]
\[ \tau^B \circ \tau^B = 0, \quad \text{by (19), (20) and (24)}; \]
\[ \tau^R \circ \tau[S] = 0, \quad \text{by (22), (21) and (23)}; \]
\[ \tau^B \circ \tau^R = 0, \quad \text{by (22), (21) and (23)}; \]
\[ \tau^R \circ \tau^B = 0, \quad \text{by (20), (22) and (24)}; \]
\[ \tau^R \circ \tau^B = 0, \quad \text{by (21), (23) and (25)}; \]
\[ \tau^B \circ \tau^B = 0, \quad \text{by (19) and (24)}; \]
\[ \tau^R \circ \tau^B \circ \tau[S] \circ \tau^R \circ \tau^B \circ \tau^B = \ \]
\[ \text{The generalization after computing the first few iterates } n = 1, \ldots, 4 \text{ leads to the following induction hypothesis } (n \geq 1) \]
\[ (\tau[C])^n \overset{def}{=} A_n \cup B_n \cup C_n \cup D_n \cup E_n \cup F_n \cup G_n \] (27)
where
\[ A_n \overset{def}{=} \bigcup_{i=1}^{n-\frac{1}{2}(k_i+2)} \left( \tau^B \circ \tau[S]^{k_i} \circ \tau^R \right); \] (28)
(This corresponds to \(j\) loops iterations from and to the loop entry \(\text{at}_P[C]\) where the \(i\)-th execution of the loop body \(S\) exactly takes \(k_i \geq 1\) steps. \(A_n = \emptyset, \ n \leq 1\).)
\[ B_n \overset{def}{=} \bigcup_{n=\frac{1}{2}(k_i+2)+1}^{\infty} \left( \left( \bigcup_{i=1}^{\frac{1}{2}(k_i+2)} \right) \circ \tau^B \circ \tau[S]^{k_i} \circ \tau^R \right); \] (29)
\[ k_i > 0, \ i = 1, \ldots, j \] are not explicitly inserted in the formula.\(^5\)
\(C_n \stackrel{\text{def}}{=} \bigcup_{n=(\frac{1}{2}(k+2))}^{\ell + 2} \left( \left( \bigotimes_{i=1}^{\ell} (\sigma^R \circ \circ \sigma^B) \right) \circ \sigma^B \right) ; \quad (30)\)

(This corresponds to \(j\) loops iterations where the \(i\)-th execution of the loop body \(S\) has \(k_i \geq 1\) steps followed by a successful condition \(B\) and a partial execution of the loop body \(S\) for \(\ell \geq 0\) steps.

\(D_n \stackrel{\text{def}}{=} \bigcup_{n=(\frac{1}{2}(k+2))}^{\ell + 2} \left( \left( \bigotimes_{i=1}^{\ell} (\sigma^B \circ \circ \sigma^R) \right) \circ \sigma^R \right) ; \quad (31)\)

\(E_n \stackrel{\text{def}}{=} \bigcup_{n=(\frac{1}{2}(k+2))}^{\ell + 2 + m} \left( \left( \bigotimes_{i=1}^{\ell} (\sigma^B \circ \circ \sigma^R) \right) \circ \sigma^R \circ \sigma^B \right) ; \quad (32)\)

\(F_n \stackrel{\text{def}}{=} \bigcup_{n=(\frac{1}{2}(k+2))}^{\ell + 2} \left( \left( \bigotimes_{i=1}^{\ell} (\sigma^B \circ \circ \sigma^R) \right) \circ \sigma^B \right) ; \quad (33)\)

\(G_n \stackrel{\text{def}}{=} (\sigma^R)^n ; \quad (34)\)

\(F_n \) is similar to \(E_n\) except that the execution of the loop terminates with condition \(B\) false.

\(G_n \) is the condition \(C\) term.

We now prove (27) by recurrence on \(n\). Given a formula \(F_n \in \{A_n, \ldots, F_0\}\) of the form \(F_n = \bigcup_{C(n,n,m)} T(n, \ell, m, \ldots)\), where \(n, \ell, m, \ldots\) are free variables in the condition \(C\) and term \(T\), we write \(F_n \mid C' = C(n, \ell, m, \ldots)\) for the formula

\(F_n \mid C' = \bigcup_{C(n,n,m)} T(n, \ell, m, \ldots).

---

For the basis observe that for \(n = 1\), \(A_1 = \emptyset\), \(B_1 = \tau^B\), \(C_1 = \tau^B\), \(D_1 = \tau^R\), \(E_1 = \emptyset\), \(F_1 = \emptyset\) and \(G_1 = (\sigma^R)^1 = \sigma^R\) so that

\((\sigma^C)^1 = \sigma^C
\)

\(= \tau^R \circ \sigma^R \circ \sigma^B \circ \sigma^B
\)

\(= A_1 \cup B_1 \cup C_1 \cup D_1 \cup E_1 \cup F_1 \cup G_1 .

---

For \(n = 2\), observe that \(A_2 = \emptyset\), \(B_2 = \tau^B \circ \sigma^R\), \(C_2 = \emptyset\), \(D_2 = \sigma^R \circ \sigma^B\), \(E_2 = \tau^R \circ \tau^B\), \(F_2 = \tau^R \circ \tau^B\) and \(G_2 = (\sigma^R)^2\) so that

\((\sigma^C)^2 = \tau^R \circ \tau^B \circ \sigma^R \circ \sigma^B \circ \sigma^R \circ \tau^R \circ \tau^B
\)

\(= A_2 \cup B_2 \cup C_2 \cup D_2 \cup E_2 \cup F_2 \cup G_2.

---

For the induction step \(n \geq 2\), we have to consider the compositions

\(A_n \circ \sigma^C, \ldots, G_n \circ \sigma^C\) in turn.
\[\begin{align*}
\quad & A_n \circ \tau C \\
= & \quad \{ \text{def. (26) of } \tau C \} \\
& A_n \circ (\tau^B \cup \tau[S] \cup \tau^R \cup \tau^B) \\
= & \quad \{ \text{def. (27) of } \tau C \} \\
& \{ \text{def. (28) of } A_n \text{ and } \tau[S]^0 = 1 \} \cup \\
& \quad \{ \text{def. (29) of } B_{n+1} \text{ with additional constraint } \ell = 0 \text{ and def. (30) of } C_{n+1} \} \\
& B_{n+1} | \ell = 0 \cup C_{n+1}.
\end{align*}\]
\[ \tau \circ \tau^B \text{ distributes over } \cup, E_n \mid m = 0 \text{ has the form } \tau' \circ \tau^B \text{ while } E_n \mid m \geq 0 \text{ has the form } \tau'' \circ \tau[S], \tau^B \circ \tau^R = \tau^R \circ \tau^B = 0 \text{ and } \tau[S] \circ \tau^B = \tau[S] \circ \tau^R = 0 \]

\[ (E_n \mid m = 0) \circ \tau[S] \cup (E_n \mid m > 0) \circ \tau[S] \cup (E_n \mid m > 0) \circ \tau^R \]

\[ \chi \text{ def. (32) of } E_{n+1} \text{ and } k_i \geq 1 \text{ so that } \ell < n \]

\[ (E_{n+1} \mid m = 1) \cup (E_{n+1} \mid m > 1) \cup (D_{n+1} \mid \ell < n) \]

\[ \chi \text{ is associative} \]

\[ (E_{n+1} \mid m > 0) \cup (D_{n+1} \mid \ell < n) \]

\[ \tau \circ \tau^C \]

\[ \{ \text{def. (26) of } \tau[C] \} \]

\[ P_n \circ (\tau^B \cup \tau[S] \cup \tau^R \cup \tau^B) \]

\[ \chi \circ \chi \text{ distributes over } \cup, \chi \text{ by def. (33) of } F_n \text{ has the form } \tau' \circ \tau^B \text{ and } \tau^B \circ \tau^R = \tau^R \circ \tau^B = \tau^B \circ \tau^B = 0 \]

\[ \varnothing \]

\[ \chi \text{ def. (34) of } G_n \text{ and } \chi \text{ of } (\tau[C])^n \]

\[ (\tau[S])^n \circ (\tau^B \cup \tau[S] \cup \tau^R \cup \tau^B) \]

\[ \chi \circ \chi \text{ distributes over } \cup, n \geq 1, \tau[S] \circ \tau^B = \tau[S] \circ \tau^R = 0 \]

\[ (\tau[S])^n \circ \tau[S] \circ (\tau[S])^n \circ \tau^R \]

\[ \{ \text{def. } n + 1 \text{-th power and (31) of } D_{n+1} \} \]

\[ (\tau[S])^{n+1} \cup (D_{n+1} \mid \ell = n) \]

Grouping all cases together, we get

\[ (\tau[C])^{n+1} \]

\[ \{ \text{def. } n + 1 \text{-th power and (27)} \} \]

\[ (A_n \cup B_n \cup C_n \cup D_n \cup E_n \cup F_n \cup G_n) \circ (\tau[C])^n \]

\[ \chi \circ \chi \text{ distributes over } \cup, \chi \text{ def. (34) of } G_n \]
\[(\tau^B \circ \eta[S]^* \circ \tau^R)^+ .\]

By the same reasoning, we get
\[
\bigcup_{n \geq 1} B_n = (\tau^B \circ \eta[S]^* \circ \tau^R)^+ \circ \tau^B \circ \eta[S]^*,
\]
\[
\bigcup_{n \geq 1} C_n = (\tau^B \circ \eta[S]^* \circ \tau^R)^+ \circ \tau^B,
\]
\[
\bigcup_{n \geq 1} D_n = \eta[S]^* \circ \tau^R \circ (\tau^B \circ \eta[S]^* \circ \tau^R)^+ \circ \tau^B \circ \eta[S]^*,
\]
\[
\bigcup_{n \geq 1} E_n = \eta[S]^* \circ \tau^R \circ (\tau^B \circ \eta[S]^* \circ \tau^R)^+ \circ \tau^B \circ \eta[S]^* \circ \tau^R.
\]

Grouping now all cases together and using the fact that \(\circ\) distributes over \(\cup\), we finally get

\[
\tau^*[C] = \tau^*[S]^0 \cup (\tau^B \circ \eta[S]^* \circ \tau^R)^+ \cup (\tau^B \circ \eta[S]^* \circ \tau^R)^+ \circ \tau^B \circ \eta[S]^*.
\]

\[
\bigcup_{n \geq 1} F_n = \eta[S]^* \circ \tau^R \circ (\tau^B \circ \eta[S]^* \circ \tau^R)^+ \circ \tau^B \circ \eta[S]^* \circ \tau^R.
\]

Structural big-step operational semantics: sequence

\[
\tau^*[C_1 ; \ldots ; C_n] = \tau^*[C_1] \circ \ldots \circ \tau^*[C_n] \quad (35)
\]

**Proof.** Let us recall from Lecture 5 that if \(S = C_1 ; \ldots ; C_n\) where \(n \geq 1\) is a sequence of commands and \(\ell_i, \ell_{i+1} \in \text{inp}[S]\) for all \(i \in [1, n]\), then

\[
(\ell_i, \rho_i) \longmapsto_{[S]} (\ell_{i+1}, \rho_{i+1}) \quad (36)
\]

We also have the labelling scheme

\[
\text{at}_p[S] = \text{at}_p[C_1],
\]

\[
\text{after}_p[S] = \text{after}_p[C_n],
\]

\[
\text{inp}_p[S] = \bigcup_{i=1}^n \text{inp}_p[C_i],
\]

\[
\forall i \in [1, n]: \text{after}_p[C_i] = \text{at}_p[C_{i+1}] = \text{inp}_p[C_i] \cap \text{inp}_p[C_{i+1}],
\]

\[
\forall i, j \in [1, n]: (j \neq i - 1 \land j \neq i + 1) \implies (\text{inp}_p[C_i] \cap \text{inp}_p[C_j] = \emptyset). \quad (37)
\]

1 — Let \(S\) be the sequence \(C_1 ; \ldots ; C_n, n \geq 1\), we first prove a lemma.

1.1 — Let \(P\) be the program with subcommand \(S = C_1 ; \ldots ; C_n\). Successive small steps in \(S\) must be made in sequence since, by the definition (4) and (36) of \(\tau[S]\) and the labelling scheme (37), it is impossible to jump from one command into a different one

\[
\tau^{k_i}[C_i] \circ \ldots \circ \tau^{k_j}[C_j] = \quad (38)
\]

where

\[
\forall i \leq i \leq j \leq n: \exists \ell \in [1, n]: (k_\ell \neq 0 \iff \ell \in [i, j]) \implies \tau^{k_i}[C_i] \circ \ldots \circ \tau^{k_j}[C_j] = \emptyset.
\]

The proof is by recurrence on \(n\).

1.1.1 — If, for the basis, \(n = 1\) then either \(k_1 = 0\) and \(\tau^{k_1}[C_1] = 1_{\text{inp}[P]}\) or \(k_1 > 0\) and then \(\tau^{k_1}[C_1] = \tau^{k_1}[C_i] \circ \ldots \circ \tau^{k_j}[C_j]\) by choosing \(i = j = 1\).

1.1.2 — For the induction step, assuming (38), we prove that
If $k_{n+1} = 0$ then $T = \tau^k[C_1] \circ \ldots \circ \tau^k[C_n] \circ \tau^{k+1}[C_{n+1}]$ is of the form (38) with $n+1$ substituted for $n$. Two cases, with several subcases have to be considered.

1.1.2.1 — If $\forall i \in [1, n] : k_i = 0$ then we consider two subcases.

1.1.2.1.1 — If $k_{n+1} = 0$ then $\forall i \in [1, n+1] : k_i = 0$ and $T = \tau^k[C_1] \circ \ldots \circ \tau^k[C_n] \circ \tau^{k+1}[C_{n+1}] = \tau^k[C_1] \circ \ldots \circ \tau^k[C_n]$.

1.1.2.1.2 — Otherwise $k_{n+1} > 0$ and then $\forall \ell \in [1, n+1] : (k_\ell \neq 0 \iff \ell \in [n+1, n+1])$ and $T = \tau^k[C_1] \circ \ldots \circ \tau^k[C_n] \circ \tau^{k+1}[C_{n+1}] = \tau^k[C_1] \circ \ldots \circ \tau^k[C_n]$ by choosing $i = j = n+1$.

1.1.2.2 — Otherwise, $\exists i \in [1, n] : k_i \neq 0$.

1.1.2.2.1 — If $\exists 1 \leq i \leq j \leq n : \forall \ell \in [1, n] : (k_\ell \neq 0 \iff \ell \in [i, j])$ then by (38), we have

$$T = \tau^k[C_1] \circ \ldots \circ \tau^k[C_j] \circ \tau^{k+1}[C_{n+1}].$$

1.1.2.2.1.1 — If $k_{n+1} = 0$ then $\exists 1 \leq i \leq j \leq n+1 : \forall \ell \in [1, n+1] : (k_\ell \neq 0 \iff \ell \in [i,j])$ and:

$$T = \tau^k[C_1] \circ \ldots \circ \tau^k[C_j] \circ \tau^{k+1}[C_{n+1}],$$

$$= \tau^k[C_1] \circ \ldots \circ \tau^k[C_j] \circ \tau^{k+1}[C_{n+1}],$$

$$= \tau^k[C_1] \circ \ldots \circ \tau^k[C_j].$$

1.1.2.2.1.2 — Otherwise $k_{n+1} > 0$ and we distinguish two subcases.

1.1.2.2.2.2 — If $n = 2$ then $k_1 = 0$ and $k_2 > 0$ or $k_1 > 0$ and $k_2 = 0$ which corresponds to case 1.1.2.2.1, whence is impossible.

1.1.2.2.2.3 — So necessarily $n \geq 3$. Let $p \in [1, n]$ be minimal and $q \in [1, n]$ maximal such that $k_p \neq 0$ and $k_q \neq 0$. There exists $m \in [p, q]$ such that $k_m = 0$ since otherwise $k_\ell \neq 0$ and either $\ell < p$ in contradiction with the minimality of $p$ or $\ell > q$ in contradiction with the maximality of $q$. We have $p < m < q$ with $p < q$ and $k_m = 0$ and $k_1 = 0$. Assume $m$ to be minimal with that property, so that $k_{m-1} \neq 0$ and then that $q$ is the minimal $q$ with this property so that $k_{q-1} = 0$. We have $k_1 = 0, \ldots, k_{p-1} = 0, k_p = 0, \ldots, k_{m-1} = 0, k_m = 0, k_{q-1} = 0 k_q \neq 0, \ldots$. It follows, by the definition (4) and (36) of $\tau[C_j]$ and the labelling scheme (37) that $\tau^k[C_1] \circ \ldots \circ \tau^{k_1}[C_n] = 0$ that $T = 0 \circ \tau^{k+1}[C_{n+1}] = 0$.
It remains to prove that
\[ \forall 1 \leq i \leq j \leq n + 1 : \exists ! \ell \in \{1, n + 1\} : (k_\ell \neq 0 \land \ell \notin [i, j]) \lor (\ell \in [i, j] \land k_\ell = 0). \]

1.1.2.2.3.1 - If \( j < n + 1 \) then this follows from (38).

1.1.2.2.3.2 - Otherwise \( j = n + 1 \) in which case either \( k_{n+1} = 0 \) and then we choose \( \ell = j \) or \( k_{n+1} > 0 \) so that \( q' = j = n + 1 \). If \( j < m \) then for \( \ell = n \), we have \( k_\ell = k_n = 0 \). Otherwise \( m < i \leq q' \). Choosing \( \ell = p \), we have \( \ell \in [1, n] \) with \( k_\ell = k_p \neq 0 \).

1.2 — We will need a second lemma, stating that \( k \) small steps in \( C_1 ; \ldots ; C_n \) must be made in sequence with \( k_1 \) steps in \( C_1 \), followed by \( k_2 \) in \( C_2 \), \ldots, followed by \( k_n \) in \( C_n \) such that the total number \( k_1 + \ldots + k_n \) of these steps is precisely \( k \).

According to lemma (38), three cases have to be considered for 
\[ \tau^{k+1}[C_1 ; \ldots ; C_n] = \{\text{def. } \tau^{k+1} = \tau^k \circ t \text{ of powers}\} \]

\[ \tau^k[C_1 ; \ldots ; C_n] \circ \tau[C_1 ; \ldots ; C_n] \]

\[ = \{\text{def. (4) and (36) of } \tau[C_1 ; \ldots ; C_n]\} \]

\[ \tau^k[C_1 ; \ldots ; C_n] \circ \bigcup_{m=1}^{n} \tau[C_m] \]

\[ = \{\text{distributes over } \cup\} \]

\[ \bigcup_{m=1}^{n} \left( \bigcup_{k=k_1+\ldots+k_n} \tau^k[C_1] \circ \ldots \circ \tau^k[C_n] \right) \circ \tau[C_m] \]

\[ = \{\text{distributes over } \cup\} \]

The proof is by recurrence on \( k \geq 0 \).

1.2.1 — For \( k = 0 \), we get \( k_1 = \ldots = k_n = 0 \) and \( 1 \Sigma [\ell] \) on both sides of the equality.

1.2.2 — For \( k = 1 \), there must exist \( m \in [1, n] \) such that \( k_m = 1 \) while for all \( j \in [1, n] - \{m\} \), \( k_j = 0 \). By the definition (4) and (36) of \( \tau[C_1 ; \ldots ; C_n] \), we have

\[ \tau[C_1 ; \ldots ; C_n] = \bigcup_{m=1}^{n} \tau[C_m]. \]

1.2.3 — For the induction step \( k \geq 2 \), we have

\[ \tau^{k+1}[C_1 ; \ldots ; C_n] \]

\[ = \bigcup_{k=k_1+\ldots+k_n} \bigcup_{m=1}^{n} \tau^k[C_1] \circ \ldots \circ \tau^k[C_n] \circ \tau[C_m] \]

\[ \overset{\text{def}}{=} \{\text{by definition}\} \]

\[ T. \]

1.2.3.1 — We first show that 

\[ T \subseteq \bigcup_{k=k_1+\ldots+k_n} \tau^k[C_1] \circ \ldots \circ \tau^k[C_n]. \]

According to lemma (38), three cases have to be considered for

\[ t \overset{\text{def}}{=} \tau^k[C_1] \circ \ldots \circ \tau^k[C_n] \circ \tau[C_m]. \]

1.2.3.1.1 — The case \( \forall i \in [1, n] : k_i = 0 \) is impossible since then \( k = \sum_{j=1}^{n} k_j = 0 \) in contradiction with \( k \geq 2 \).
12.3.1.2 — Else if $\exists 1 \leq i \leq j \leq n : \forall \ell \in [i, n] : (k_{i} \neq 0 \iff \ell \in [i, j])$ then

$$t \mathrel{\overset{\text{def}}{=} } \tau^{k_{1}}[C_{1}] \circ \ldots \circ \tau^{k_{j}}[C_{j}] \circ \tau[C_{n}].$$

We discriminate according to the value of $m$.

12.3.1.2.1 — If $m = j$, we get

$$t = \tau^{k_{1}}[C_{1}] \circ \ldots \circ \tau^{k_{j}}[C_{j}] \circ \tau[C_{j+1}],$$

with $k + 1 = k_{1} + \ldots + k_{j}$, where $k_{i} = 0$, $\ldots$, $k_{j-1} = 0$, $k_{j} = k_{i}$, $\ldots$, $k_{j} = k_{j}$, $k_{j+1} = 1$, $k_{j+2} = 0$, $\ldots$, $k_{n} = 0$.

12.3.1.2.2 — If $m = j + 1$, we get

$$t = \tau^{k_{1}}[C_{1}] \circ \ldots \circ \tau^{k_{j}}[C_{j}] \circ \tau^{k_{j+1}}[C],$$

with $k + 1 = k_{1} + \ldots + k_{j}$, where $k_{i} = 0$, $\ldots$, $k_{j-1} = 0$, $k_{j} = k_{i}$, $\ldots$, $k_{j} = k_{j}$, $k_{j+1} = 1$, $k_{j+2} = 0$, $\ldots$, $k_{n} = 0$.

12.3.1.2.3 — Otherwise, by the definition (4) and (36) of $\tau_{*}[C]$ and the labelling scheme (37), $\tau[C_{1}] \circ \ldots \circ \tau[C_{n}] = 0$ so that $T = 0$ that is $t = \tau^{k_{1}}[C_{1}] \circ \ldots \circ \tau^{k_{j}}[C_{j}] \circ \tau^{k_{j+1}}[C_{n}]$ with $k_{i} = k_{\ell}$ for $\ell \in [1, n] - \{m\}$ and $k_{m} = k_{m} + 1$.

12.3.1.3 — Otherwise $T = 0$ so that the inclusion is trivial.

12.3.2 — Inversely, we now show that

$$\bigcup_{k+1 \leq k_{1} + \ldots + k_{n} \leq k} \tau^{k_{1}}[C_{1}] \circ \ldots \circ \tau^{k_{n}}[C_{n}] \subseteq T.$$
\begin{align*}
\{ \text{def. reflexive transitive closure} \} \\
\bigcup_{k\geq 0} \tau^k [C_1 ; \ldots ; C_n] \\
= \{ \text{lemma (39)} \} \\
\bigcup_{k_1 + \ldots + k_n \geq 0} \tau^{k_1} [C_1] \circ \ldots \circ \tau^{k_n} [C_n] \\
= \bigcup_{k_1 \geq 0} \ldots \bigcup_{k_n \geq 0} \tau^{k_1} [C_1] \circ \ldots \circ \tau^{k_n} [C_n] \\
= \{ \circ \text{ distributes over } \bigcup \} \\
\left( \bigcup_{k_1 \geq 0} \tau^{k_1} [C_1] \right) \circ \ldots \circ \left( \bigcup_{k_n \geq 0} \tau^{k_n} [C_n] \right) \\
= \{ \text{def. reflexive transitive closure} \} \\
\tau^* [C_1] \circ \ldots \circ \tau^* [C_n].
\end{align*}

\[
\bigcup_{k\geq 0} \tau^k [S] \\
= \{ \text{def. reflexive transitive closure} \} \\
\tau^* [S].
\]

---

**Classification of Program Trace Properties: Safety & Liveness**

Structural big-step operational semantics: programs

\[
\tau^* [S ;;] = \tau^* [S].
\]

**PROOF.** Let us recall from Lecture 5 that for programs \( P = S ;; \), we have:

\[
\begin{array}{c}
\langle \ell, \rho \rangle \longmapsto [S] \mapsto \rho' \\
\langle \ell, \rho \rangle \longmapsto [S ;;] \mapsto \langle \ell', \rho' \rangle.
\end{array}
\]

For programs \( P = S ;; \), we have

\[
\begin{align*}
\tau^* [S ;;] \\
= \{ \text{def. reflexive transitive closure} \} \\
\bigcup_{k\geq 0} \tau^k [S ;;] \\
= \{ \text{by the definition (4) and (41) of } [S ;;] \}.
\end{align*}
\]
Safety and Liveness, informally

- **safety** properties: informally, “bad things” cannot happen during program execution [5]
- **liveness** properties: informally, “good things” eventually do happen during program execution [5]);

Trace properties

- \(\Sigma\): set of states
- \(\Sigma^\infty\): non-empty finite or infinite traces over states in \(\Sigma\)
- A **trace property** \(P\) is the set of traces which have that property so

\[ P \in \wp(\Sigma^\infty) \]
Definition of Prefix Closure
– The prefix closure $\text{PCI}(S)$ of a set $S \in \mathcal{P}(\Sigma^\infty)$ of nonempty traces, is the set of all nonempty finite prefixes (also called left factors) of traces in $S$

$$\text{PCI}(S) \overset{\text{def}}{=} \{ \sigma \in \Sigma^+ | \exists \sigma' \in \Sigma^\infty : \sigma \cdot \sigma' \in S \}$$

Properties of the prefix closure
– For finite sequences, PCI is a topological closure operator on $\mathcal{P}(\Sigma^*)$ and $\mathcal{P}(\Sigma^+)$:
  - $\text{PCI} \in \mathcal{P}(\Sigma^*) \mapsto \mathcal{P}(\Sigma^*)$
  - $\text{PCI} \in \mathcal{P}(\Sigma^+) \mapsto \mathcal{P}(\Sigma^+)$
  - $X \subseteq \text{PCI}(X)$
  - $\text{PCI}(\text{PCI}(X)) = \text{PCI}(X)$
  - $\text{PCI}(X \cup Y) = \text{PCI}(X) \cup \text{PCI}(Y)$
  - $\text{PCI}(\emptyset) = \emptyset$

Definition of Limit Closure
– The limit $\text{Lim}(S)$ of a set $S \in \mathcal{P}(\Sigma^\infty)$ of nonempty traces, is the set $S$ augmented with all infinite traces which have infinitely many finite prefixes in $S$

$$\text{Lim}(S) \overset{\text{def}}{=} S \cup \{ \sigma \in \Sigma^\infty | \forall i : \exists j \geq i : \sigma_0 \ldots \sigma_j \in S \}$$

- For bifinitary sequences, PCI satisfies:
  - $\text{PCI} \in \mathcal{P}(\Sigma^\infty) \mapsto \mathcal{P}(\Sigma^\infty)$
  - $\text{PCI} \in \mathcal{P}(\Sigma^\infty) \mapsto \mathcal{P}(\Sigma^\infty)$
  - $X \subseteq \text{PCI}(X)$, when $X \cap \Sigma^\infty \neq \emptyset$
  - $\text{PCI}(\text{PCI}(X)) = \text{PCI}(X)$
  - $\text{PCI}(X \cup Y) = \text{PCI}(X) \cup \text{PCI}(Y)$
  - $\text{PCI}(\emptyset) = \emptyset$

This implies $X \subseteq Y \Rightarrow \text{PCI}(X) \subseteq \text{PCI}(Y)$.
Properties of the limit closure

- $\lim$ is a topological closure operator on $\varphi(\Sigma^\omega)$.

**Proof.**
- $X \subseteq \lim(X)$ extensive
- $\lim(X \cup Y) = \lim(X) \cup \lim(Y)$ additive\(^{10}\)
- $\lim(\lim(X)) = \lim(X)$ idempotent\(^{10}\)
- $\lim(0) = 0$ $\emptyset$-strict

\(^{10}\) Since any infinite sequence in $\operatorname{lim}(X \cup Y)$ not in $X \cup Y$ has infinitely many different prefixes in $X \cup Y$. If there are finitely many in $X$ then there are infinitely many in $Y$ so the limit is in $\operatorname{lim}(Y)$. Same if there are finitely many in $Y$ then there are infinitely many in $X$ so the limit is in $\operatorname{lim}(X)$. If there are infinitely many in both $X$ and $Y$ then the limits are identical and in both $\operatorname{lim}(X)$ and $\operatorname{lim}(Y)$. If there are infinitely many in both $X$ and $Y$ then there are infinitely many in $X \cup Y$.

\[\lim \circ \operatorname{PCI}(S_\varphi) = S_\varphi\]

Example of safety: invariance

- $\varphi \subseteq \Sigma \times \Sigma$ state relation
- $S_\varphi = \{\sigma \in \Sigma^\omega \mid \forall i \in [0, |\sigma|]: \langle \sigma_0, \sigma_i \rangle \in \varphi\}$ invariance of $\varphi$

\[\tau^* \subseteq S_\varphi\] safety property
- $\tau^* \subseteq S_\varphi$ if and only if all reachable states are linked to initial states by $\varphi$

Formal Definition of Safety

- $S \subseteq \Sigma^\omega$ is a safety property if and only if [6]:

\[
\text{Safe}(S) = S
\]

where:

\[
\text{Safe} \overset{\text{def}}{=} \lim \circ \operatorname{PCI}
\]

(42)

i.e. $S$ is closed by limits of prefixes

Reference


Counter-example of safety property

- $\Sigma = \{a, b\}$
- $S = \{a\}^+. \{b\}^*$
  \[
  = \{a^n b^m \mid n > 0 \land m \geq 0\}
  \]
- All traces in $S$ have a finite number of $a$'s followed by zero or more $b$'s
- The infinite trace $\sigma = aaaaa \ldots$ is thus excluded from $S$
- Its impossible to discover this fact by observing finite prefixes of traces in $S$
- So $S$ is not a safety property
Proof. \[ S = \{a\}^\prec \cdot \{b\}^\prec \]
- \( \text{PCI}(S) = \{a\}^\prec \cup \{a\}^\prec \cdot \{b\}^\prec \)
  \[ = \{a\}^\prec \cdot \{b\}^\prec \]
- \( \text{Lim}(\text{PCI}(S)) = \text{Lim}(S) \)
  \[ = S \cup \{a\}^\omega \cup \{a\}^\prec \cdot \{b\}^\omega \]
  \[ = \{a\}^\infty \cup \{a\}^\prec \cdot \{b\}^\infty \]
- \( S \neq \text{Lim}(\text{PCI}(S)) \)

\[ \Box \]

Prefix of a trace

\( \sigma \triangleright n \) is the prefix of length \( n \in \mathbb{N} \) of trace \( \sigma \)
- \( \sigma \triangleright n = \sigma \) \quad if \( |\sigma| \leq n \)
- \( \sigma \triangleright n = \sigma_0 \ldots \sigma_{n-1} \) \quad if \( |\sigma| \geq n \)

\[ ^1 \text{Recall that } |\sigma| \text{ is the length of } \sigma, \omega \text{ if infinite.} \]

Suffix of a trace

\( \sigma \nrightarrow n \) is the suffix of trace \( \sigma \) beyond \( n \in \mathbb{N} \)
- \( \sigma \nrightarrow n = \varepsilon \) \quad if \( n > |\sigma| \)
- \( \sigma \nrightarrow n = \sigma_n \ldots \sigma_{\ell-1} \) \quad if \( n \leq \ell = |\sigma| < \omega \)
- \( \sigma \nrightarrow n = \sigma_n \sigma_{n+1} \sigma_{n+2} \ldots \) \quad if \( |\sigma| = \omega \)

Characterization of safety properties

Safety properties \( S \) can be disproved by looking only at some finite partial program behavior:

\[ \forall \sigma \in \Sigma^\omega : (\sigma \not\in S) \iff (\exists i \geq 1 : \sigma \triangleright i \not\in S) \]

Proof. \[ \text{Lim} \circ \text{PCI}(S) = S \]
\[ \iff \text{Lim} \circ \text{PCI}(S) \subseteq S \]
\[ \iff \{ \sigma \in \Sigma^\omega : \forall i \geq 1 : \sigma \triangleright i \in \text{PCI}(S) \} \subseteq S \]
\[ \iff \{ \sigma \in \Sigma^\omega : \forall i \geq 1 : \exists \beta \in \Sigma^\omega : \sigma \triangleright i \cdot \beta \in S \} \subseteq S \]
\[ \iff \forall \sigma \in \Sigma^\omega : (\forall i \geq 1 : \exists \beta \in \Sigma^\omega : \sigma \triangleright i \cdot \beta \in S) \implies (\sigma \in S) \]
\[ \iff \forall \sigma \in \Sigma^\omega : (\sigma \not\in S) \implies (\exists i \geq 1 : \forall \beta \in \Sigma^\omega : \sigma \triangleright i \cdot \beta \not\in S) \]
\[ \iff \forall \sigma \in \Sigma^\infty : (\sigma \notin S) \iff (\exists i \geq 1 : \forall \beta \in \Sigma^\infty : \sigma \not\leadsto i \cdot \beta \notin S) \]

**Definition of Liveness**

- \( S \subseteq \Sigma^\infty \) is a *liveness* property if and only if [7]:

\[
\text{Lim} \circ \text{PCI}(S) = \Sigma^\infty
\]

\[
\iff \text{PCI}(S) = \Sigma^\infty
\]

\[
\iff \Sigma^\infty \subseteq \text{PCI}(S)
\]

\[\]
Characterization 1 of liveness

- Proving liveness properties \( S \) imperatively requires the consideration of infinite behaviors:

\[
\forall \alpha \in \Sigma^+: \exists \beta \in \Sigma^\omega: \alpha \cdot \beta \in S
\]

Proof. \( \text{Lim} \circ \text{PCI}(S) = \Sigma^\omega \)

\[
\iff \Sigma^\omega \subseteq \text{Lim} \circ \text{PCI}(S)
\]

\[
\iff \Sigma^\omega \subseteq \{ \sigma \in \Sigma^\omega \mid \forall i \geq 1: \sigma \downarrow i \in \text{PCI}(S) \}
\]

\[
\iff \forall \sigma \in \Sigma^\omega: \forall i \geq 1: \exists \beta \in \Sigma^\omega: \sigma \downarrow i \cdot \beta \in S
\]

Example of Liveness Property: Termination

- \( L = \Sigma^+ \) termination
- \( \text{PCI}(L) = \text{PCI}(\Sigma^+) = \Sigma^+ \) liveness property
- \( \tau^\omega \subseteq L \iff \tau^\omega \subseteq \Sigma^\omega \iff \tau^\omega = \emptyset \) termination

(there is no possible infinite execution).

A liveness property cannot be checked by a program during its execution so liveness is inobservable at execution.

Dual Limit

\[
\text{Lim}(P) \triangleq \neg(\text{Lim}(\neg P))
\]

\[
\text{Lim}(P) = \neg\{ \sigma \in \Sigma^\omega \mid \forall i \in \mathbb{N}_+ : \sigma \downarrow i \in \neg P \}
\]

\[
= \{ \sigma \in \Sigma^\omega \mid \exists i \in \mathbb{N}_+ : \sigma \downarrow i \in P \}
\]

so that \( P \subseteq \text{Lim}(P) \) since whenever \( \sigma \in P \), we have \( \sigma \downarrow |\sigma| = \sigma \in P \) proving:

\( \text{PCI} \circ \text{Lim} \) is extensive

since \( P \subseteq \text{PCI}(P) \subseteq \text{PCI}(\text{Lim}(P)) \) follows from extensivity and monotonicity of \( \text{PCI} \).
Characterization 2 of liveness

If we define:

\[ \text{Live}(P) \triangleq \neg \text{Safe}(P) \cup P \]

then

\[ P \subseteq \Sigma^\infty \]

is a liveness property if and only if \( \text{Live}(P) = P \).

**Proof.** – \( \text{Live}(P) \) is a liveness property:

\[
\begin{align*}
\Sigma^\circ & = \text{PCI}(P) \cup \neg \text{PCI}(P) \\
& \subseteq \text{PCI}(P) \cup \text{PCI} \circ \lim (\neg \text{PCI}(P)) \quad \text{PCI} \circ \lim \text{ is extensive}
\end{align*}
\]

\[ = \text{PCI}(P \cup \lim (\neg \text{PCI}(P))) \quad \text{PCI is a complete join morphism} \]

\[ = \text{PCI}(P \cup \lim (\neg \text{PCI}(P))) \quad \text{def. } \lim \]

\[ = \text{PCI}(P \cup \neg \text{PCI}(P)) \quad \text{def. } \text{PCI} \circ \lim \text{ is extensive} \]

\[ = \text{PCI}(P) \cup \neg \text{Safe}(P) \quad \text{def. } \text{Safe} \]

\[ = \text{PCI}(\text{Live}(P)) \quad \text{def. } \text{Live, Q.E.D.} \]

- \( \text{Live}(P) \) is a liveness property so if \( \text{Live}(P) = P \) then \( P \) is also a liveness property;

- Reciprocally, if \( P \) is a liveness property then \( \Sigma^\circ \subseteq \text{PCI}(P) \); hence \( \Sigma^\infty = \lim (\Sigma^\circ) \subseteq \lim \circ \text{PCI}(P) = \text{Safe}(P) \); whence \( \neg \text{Safe}(P) = \emptyset \) so that \( \text{Live}(P) = \neg \text{Safe}(P) \cup P = \emptyset \cup P = P \).

An immediate consequence is that \( \text{Live} \) is extensive and idempotent. However it is not monotonic (\( \emptyset \subseteq \Sigma^\circ \) but \( \text{Live}(\emptyset) = \neg \text{Safe}(\emptyset) \cup \emptyset = \emptyset \subseteq \Sigma^\infty \)). This also shows that \( \text{Live}(P) \) may not be the least liveness property including \( P \) (since \( \emptyset \subseteq \Sigma^\circ \) but \( \text{Live}(\emptyset) \not\subseteq \Sigma^\circ \)).

\[ \text{Live}(P) = \neg \text{Safe}(P) \cup P \]

Decomposition into Safety and Liveness

- Any program property \( P \) can be decomposed into the conjunction of a safety and a liveness property [8]:

  - \( \text{Safe}(P) \) safety property

  - \( \text{Live}(P) \) liveness property

  - \( P = \text{Safe}(P) \cap \text{Live}(P) \)

**Proof.** \( \text{Safe}(P) \cap \text{Live}(P) = (\neg \text{Safe}(P) \cup P) \cap \text{Safe}(P) = (\neg \text{Safe}(P) \cap \text{Safe}(P)) \cup (P \cap \text{Safe}(P)) = P \cap \text{Safe}(P) = P \) since \( \text{Safe} \) is extensive.

Reference


A Simple Example [9]

- $\Sigma = \{a, b, c\}$
- $S = \{a\} \cdot \{b\} \cdot \Sigma^\infty$
- $\text{Safe}(S) = \{a\}^\infty \cup \{a\} \cdot \{b\} \cdot \Sigma^\infty$
- $\text{Live}(S) = \{a\} \cdot \{b, c\} \cdot \Sigma^\infty$
- $S = \text{Safe}(S) \cap \text{Live}(S)$

---

Total correctness proof:
$$\forall \sigma \in \tau^\infty : \exists i < |\sigma| : (\sigma_0, \sigma_i) \in \Phi$$

Partial correctness proof:
$$\forall s, s' \in \Sigma : (s \in \Gamma \land (s, s') \in \tau^* \land s' \in \Xi) \implies ((s, s') \in \Psi)$$

Termination proof:
$$\forall \sigma \in \tau^\infty : (\sigma_0 \in \Gamma) \implies (\exists i < |\sigma| : \sigma_i \in \bar{\tau})$$

Example: decomposition of total correctness into partial correctness (safety) and termination (liveness)

- Total correctness specification $(\Gamma, \Psi)$:
  - $\Gamma \subseteq \Sigma$ initial states
  - $\Psi \subseteq \Gamma \times \Xi$ partial correctness relation
  - $\Xi \overset{\text{def}}{=} \bar{\tau} \subseteq \Sigma$ final/blocking states
  - $\Phi \overset{\text{def}}{=} \{(s, s') \mid (s \in \Gamma) \implies (s' \in \Xi \land (s, s') \in \Psi)\}$ total correctness relation

---

PROOF. $\forall \sigma \in \tau^\infty, (\sigma_0 \in \Gamma)$

- initial states hypothesis
  - $\sigma_0 \in \Gamma \land \sigma \in \tau^\infty \land \sigma \in \Sigma^\bar{\tau}$ by termination
  - $\sigma_0 \in \Gamma \land \sigma \in \tau^\bar{\tau}$ by def. $\tau^\infty$
  - $\sigma_0 \in \Gamma \land (\sigma_0, \sigma_{|\sigma|-1}) \in \tau^* \land \sigma_{|\sigma|-1} \in \bar{\tau}$ by def. $\tau^\bar{\tau}$ and $\tau^*$
  - $\sigma_0 \in \Gamma \land (\sigma_0, \sigma_{|\sigma|-1}) \in \tau^* \land \sigma_{|\sigma|-1} \in \Xi$ by def. $\Xi$
  - $(\sigma_0, \sigma_{|\sigma|-1}) \in \Psi$ by partial correctness
  - $\exists i < |\sigma| : (\sigma_0, \sigma_i) \in \Psi$ proving total correctness
THE END

My MIT web site is http://www.mit.edu/~cousot/
The course web site is http://web.mit.edu/afs/athena.mit.edu/course/16/16.399/www/.

Course 16.399: "Abstract interpretation", Tuesday March 15th, 2005 — 109 — @ P. Cousot, 2005