Program analysis: from proving correctness to proving incorrectness

Roberto Bruni, Roberta Gori
(University of Pisa)
Lecture #09

BISS 2024
March 11-15, 2024
Pointer analysis
Why is pointer analysis important?

“If the pointer is not pointing to a valid object or function, bad things may happen”

Robert C. Seacord
Effective C: An Introduction to Professional C Programming
Example

$q := \text{nil}$;

while $p \neq \text{nil}$ do ( 
  $t := [p + 1]$;
  $[p + 1] := q$;
  $q := p$;
  $p := t$
)
Example

\[
q := \text{nil};
\]

\[
\text{while } p \neq \text{nil} \text{ do (}
\]

\[
t := [p + 1];
\]

\[
[p + 1] := q;
\]

\[
q := p;
\]

\[
p := t
\]

\)

\[
\]
Example

$q := \text{nil};$

while $p \neq \text{nil}$ do (  
  $t := [p + 1];$
  $[p + 1] := q;$
  $q := p;$
  $p := t$
)

\[ t := [p + 1]; \]
\[ [p + 1] := q; \]
\[ q := p; \]
\[ p := t \]
Example

\[ q := \text{nil}; \]
\[ \text{while } p \neq \text{nil} \text{ do (} \]
\[ t := [p + 1]; \]
\[ [p + 1] := q; \]
\[ q := p; \]
\[ p := t \]
\[ ) \]
Example

\( q := \text{nil}; \)
while \( p \neq \text{nil} \) do (\( t := [p + 1]; \)
\( [p + 1] := q; \)
\( q := p; \)
\( p := t \)
Example

\[
q := \text{nil};\\
\text{while } p \neq \text{nil} \text{ do (}\\
\quad t := [p + 1];\\
\quad [p + 1] := q;\\
\quad q := p;\\
\quad p := t\\
\text{) }
\]
Example

$q := \text{nil};$
while $p \neq \text{nil}$ do (
  $t := [p + 1];$
  $[p + 1] := q;$
  $q := p;$
  $p := t$
)
Example

$q := \text{nil};$
while $p \neq \text{nil}$ do (  
  $t := [p + 1];$
  $[p + 1] := q;$  
  $q := p;$  
  $p := t$
)
Example

\[
q := \text{nil}; \\
\text{while } p \neq \text{nil} \text{ do (} \\
\quad t := [p + 1]; \\
\quad [p + 1] := q; \\
\quad q := p; \\
\quad p := t \\
\text{)}
\]
Example

q := nil;
while p ≠ nil do ( 
  t := [p + 1];
  [p + 1] := q;
  q := p;
  p := t
)
Example

\[
\begin{align*}
q & := \text{nil}; \\
\text{while } p \neq \text{nil} \text{ do (} \\
& \quad t := [p + 1]; \\
& \quad [p + 1] := q; \\
& \quad q := p; \\
& \quad p := t
\end{align*}
\]
Example

$q := \text{nil};$
while $p \neq \text{nil}$ do (
    $t := [p + 1]$;
    $[p + 1] := q$;
    $q := p$;
    $p := t$
)
Example

\[
q := \text{nil}; \\
\text{while } p \neq \text{nil} \text{ do (}
\]
\[
t := [p + 1]; \\
[p + 1] := q; \\
q := p; \\
p := t
\]
\)
Example

\[
q := \text{nil}; \\
\text{while } p \neq \text{nil} \text{ do (} \\
\quad t := [p + 1]; \\
\quad [p + 1] := q; \\
\quad q := p; \\
\quad p := t \\
\text{)}
\]
Example

\[
q := \text{nil}; \\
\text{while } p \neq \text{nil} \text{ do (} \\
\hspace{1em} t := [p + 1]; \\
\hspace{1em} [p + 1] := q; \\
\hspace{1em} q := p; \\
\hspace{1em} p := t \\
\text{)}
\]
Example

\[
q := \text{nil}; \\
\text{while } p \neq \text{nil} \text{ do (}
\begin{align*}
t &:= [p + 1]; \\
[p + 1] &:= q; \\
q &:= p; \\
p &:= t
\end{align*}
\)
\]

\[
P \triangleq \exists \alpha, \beta . \, \text{list}(\alpha, p) \land \text{list}(\beta, q) \land \alpha^*_0 = \alpha^* \cdot \beta
\]

an inductive list predicate:
\[
\text{list}(\epsilon, p) \triangleq (p = \text{nil}) \\
\text{list}(n \cdot \alpha, p) \triangleq \exists q . \, p \mapsto \langle n, q \rangle \land \text{list}(\alpha, q)
\]
Example

$q := \text{nil};$

while $p \neq \text{nil}$ do ( 
  $t := [p + 1]$;
  $[p + 1] := q$;
  $q := p$;
  $p := t$
)

$P \triangleq \exists \alpha, \beta . \text{list}(\alpha, p) \land \text{list}(\beta, q) \land \alpha_0^\dagger = \alpha^\dagger \cdot \beta$

would fail if lists overlap!
Example

\[
q := \text{nil}; \\
\text{while } p \neq \text{nil} \text{ do (} \\
t := [p + 1]; \\
[p + 1] := q; \\
q := p; \\
p := t
\)

\[
P \triangleq \exists \alpha, \beta. \ \text{list}(\alpha, p) \land \text{list}(\beta, q) \land \alpha^\dagger_0 = \alpha^\dagger \cdot \beta \\
\land (\forall k. \ \text{reach}(p, k) \land \text{reach}(q, k) \implies k = \text{nil})
\]

an inductive reachability predicate:

\[
\text{reach}(p, q) \triangleq p = q \\
\lor \exists n, t. \ p \mapsto \langle n, t \rangle \land \text{reach}(t, q)
\]
Example

$q := \text{nil};$

while $p \neq \text{nil}$ do ( 

$t := [p + 1];$

$[p + 1] := q;$

$q := p;$

$p := t$

)

\[ P \triangleq \exists \alpha, \beta. \ list(\alpha, p) \land list(\beta, q) \land \alpha_0^\dagger = \alpha^\dagger \cdot \beta \]
\[ \land (\forall k. \ reach(p, k) \land reach(q, k) \Rightarrow k = \text{nil}) \]

*invariant?*

what if other lists are used?
Example

$q := \text{nil};$
while $p \neq \text{nil}$ do (
    $t := [p + 1];$
    $[p + 1] := q;$
    $q := p;$
    $p := t$
)
Example

\[ q := \text{nil}; \]
\[ \text{while } p \neq \text{nil do } ( P \triangleq \exists \alpha, \beta . \ (\text{list}(\alpha, p) \ast \text{list}(\beta, q)) \land \alpha^{\dagger}_0 = \alpha^{\dagger} \cdot \beta \]
\[ t := [p + 1]; \]
\[ [p + 1] := q; \]
\[ q := p; \]
\[ p := t \]
\)
Example

\[
\{ \text{true} \} \\
[x] := 1; \\
y := 2; \\
z := 3; \\
\{ z \mapsto 3 \}
\]
Example

\[
\begin{align*}
\{& \text{true} \}\; \\
[x] & := 1; \\
y & := 2; \\
z & := 3; \\
\{z \mapsto 3\} & \text{ valid!}
\end{align*}
\]
Example

\{ true \}

\[ x := 1; \]
\[ y := 2; \]
\[ z := 3; \]

\{ x \mapsto 1 \land y \mapsto 2 \land z \mapsto 3 \}

is it valid?
Example

\{ \text{true} \}

[x] := 1;
[y] := 2;
[z] := 3;

\{ x \mapsto 1 \land y \mapsto 2 \land z \mapsto 3 \} 

we must exclude aliasing!

is it valid? ❌
Example

\( \{ x \neq y \land x \neq z \land y \neq z \} \)

\([x] := 1;\)

\([y] := 2;\)

\([z] := 3;\)

\(\{ x \mapsto 1 \land y \mapsto 2 \land z \mapsto 3 \}\)

valid!
Example

\{x_1 \neq x_2 \land \ldots\}

\[x_1\] := 1;

\[x_2\] := 2;

\ldots

\[x_n\] := n;

\{x_1 \mapsto 1 \land \ldots \land x_n \mapsto n\}

\text{n(n - 1)/2 conjuncts!}
Example

\[ \{x_1 \mapsto \_ \ast \ldots \ast x_n \mapsto \_} \]

\[ [x_1] := 1; \]
\[ [x_2] := 2; \]
\[ \ldots \]
\[ [x_n] := n; \]

\[ \{x_1 \mapsto 1 \ast \ldots \ast x_n \mapsto n} \]
Heap manipulating atomic commands
Stores, heaps and states

\[ s : X \xrightarrow{\text{fin}} \mathbb{Z} \]

- Store\: variables \: \in \: \mathbb{Z}
- Set of values \: \mathbb{Z}

\[ h : \mathbb{N} \xrightarrow{\text{fin}} \mathbb{Z}_\perp \]

- Heap\: locations \: \in \: \mathbb{N}
- Set of locations \: \mathbb{N}

\[ \text{null} = -1 \]

- Special value\: (deallocated)

\[ \text{States} \triangleq \text{Stores} \times \text{Heaps} \]

\[ \sigma = \langle s, h \rangle \]

\[ \mathcal{O} \text{(States)} \]
Notation

$\text{dom}(f)$ is the domain of definition of a heap/store function

$h_1 \# h_2 \triangleq \text{dom}(h_1) \cap \text{dom}(h_2) = \emptyset$

$h_1 \cdot h_2$ is the union of functions with disjoint domains
(undefined if $\neg(h_1 \# h_2)$)

$f[x \mapsto n]$ is the partial function like $f$ except that $x$ goes to $n$
Regular commands

- \( r ::= e \)
- \( r ::= r_1 ; r_2 \)
- \( r ::= r_1 + r_2 \)
- \( r ::= r^* \)

\( e ::= \) skip
- \( b? \)
- \( x ::= a \)
- \( x ::= [a] \)  // read
- \([a_1] ::= a_2 \)  // write
- \( x ::= \text{alloc()} \)
- \( x ::= \text{free}(x) \)
- \( x ::= \text{cons}(a_0, \ldots, a_k) \)
Assertion language

\[ P ::= \text{true} \mid \text{false} \mid a_1 < a_2 \mid a_1 = a_2 \mid \ldots \]
\[ \neg P \mid P_1 \land P_2 \mid \exists x. P \mid \ldots \]
\[ \text{emp} \quad \text{empty heap} \]
\[ a_1 \leftrightarrow a_2 \quad \text{structural assertions} \]
\[ P_1 \ast P_2 \quad \text{ownership} \]
\[ \text{and separately} \]

Boolean and classical assertions
also called pure assertions
Satisfaction: classical

\[ \langle s, h \rangle \models P \]

the assertion \( P \) holds for the given state \( \langle s, h \rangle \)

\[ \langle s, h \rangle \models a_1 < a_2 \quad \text{iff} \quad s \models a_1 < a_2 \]

\[ \langle s, h \rangle \models P_1 \land P_2 \quad \text{iff} \quad \langle s, h \rangle \models P_1 \text{ and } \langle s, h \rangle \models P_2 \]

\[ \langle s, h \rangle \models \forall x. \ P \quad \text{iff} \quad \forall v \in \mathbb{Z}. \ \langle s[x \mapsto v], h \rangle \models P \]

...
Satisfaction: structural

\[
\langle s, h \rangle \models P
\]
the assertion \( P \) holds for the given state \( \langle s, h \rangle \)

\[
\langle s, h \rangle \models a_1 \mapsto a_2 \quad \text{iff} \quad \text{dom}(h) = \{ [a_1]s \} \text{ and } h([a_1]s) = [a_2]s
\]

\[
\langle s, h \rangle \models \text{emp} \quad \text{iff} \quad h \text{ is the empty map } [ ]
\]

\[
\langle s, h \rangle \models P_1 \ast P_2 \quad \text{iff} \quad \exists h_1, h_2. \, \langle s, h_1 \rangle \models P_1 \text{ and } \langle s, h_2 \rangle \models P_2 \text{ and } h = h_1 \cdot h_2
\]
splits the heap but not the store!
Example

\[ \text{dom}(h) = \{11, 42\} \]

\[ s(x) = 11 \quad h(11) = 42 \]

\[ s(y) = 42 \quad h(42) = 11 \]

\[ h = h_1 \cdot h_2 \]

\[ \text{dom}(h_1) = \{11\} \]

\[ h_1(11) = 42 \]

\[ \text{dom}(h_2) = \{42\} \]

\[ h_2(42) = 11 \]

\[ \langle s, h \rangle \models \text{emp} \quad \times \]

\[ \langle s, h \rangle \models x \mapsto y \quad \times \]

\[ \langle s, h \rangle \models x \mapsto y^* \cdot y \mapsto x \quad \checkmark \]
Example

\[ \text{dom}(h_1) = \{11\} \]
\[ s(x) = 11 \quad h_1(11) = 42 \]
\[ s(y) = 42 \]

\[ \langle s, h_1 \rangle \models \text{emp} \, ? \quad \times \]
\[ \langle s, h_1 \rangle \models x \mapsto y \, ? \quad \checkmark \]
\[ \langle s, h_1 \rangle \models x \mapsto y \ast y \mapsto x \, ? \quad \times \]
Example

\[ \text{dom}(h_2) = \{42\} \]
\[ s(x) = 11 \]
\[ s(y) = 42 \quad h_2(42) = 11 \]

\[ \langle s, h_2 \rangle \not\models \text{emp} ? \]
\[ \langle s, h_2 \rangle \models y \mapsto x ? \quad \checkmark \]
\[ \langle s, h_2 \rangle \not\models x \mapsto y \ast y \mapsto x ? \quad \times \]
Example

\begin{align*}
\text{dom}(h_2) &= \{42\} \\
s(x) &= 11 \\
s(y) &= 42 \\
h_2(42) &= 11
\end{align*}

\begin{align*}
\text{dom}(h_1) &= \{11\} \\
s(x) &= 11 \\
h_1(11) &= 42 \\
s(y) &= 42
\end{align*}

\[h = h_1 \cdot h_2\]

\[(s, h) \models x \mapsto y \ast y \mapsto x? \quad \checkmark\]
Some subtleties

\[ P \times \text{emp} \equiv P \]

\[ P \land \text{emp} \not\equiv P \]

\[ x \mapsto v \times \text{true} \not\equiv x \mapsto v \]

\[ x \mapsto v \times x \mapsto w \equiv \text{false} \]

\[ (x = y) \times (x = y) \equiv (x = y) \]

\[ P \times P \equiv P \]

\[ x \mapsto v \times y \mapsto w \not\Rightarrow x \mapsto v \]

\[ x \mapsto v \not\Rightarrow x \mapsto v \times y \mapsto w \]

\[ x \mapsto v \not\Rightarrow x \mapsto v \land y \mapsto w \]
Let us define the following inductive predicate for list segments

\[ \text{ls}(a_1, a_2, 0) \triangleq a_1 = a_2 \land \text{emp} \]
\[ \text{ls}(a_1, a_2, n + 1) \triangleq a_1 \neq a_2 \land \exists x. \ a_1 \mapsto x \ast \text{ls}(x, a_2, n) \]

and let \( \text{ls}(a_1, a_2) \triangleq \exists n. \ \text{ls}(a_1, a_2, n) \) and \( \text{list}(a) \triangleq \text{ls}(a, \text{nil}) \)

Does the model in the figure satisfy \( \text{ls}(x, x) \) ?

and \( \text{ls}(x, y) \ast \text{ls}(y, x) \) ?

and \( \text{list}(x) \) ?
Separation Logic (SL)
Local Reasoning about Programs that Alter Data Structures

Peter O'Hearn¹, John Reynolds², and Hongseok Yang³

¹ Queen Mary, University of London
² Carnegie Mellon University
³ University of Birmingham and University of Illinois at Urbana-Champaign

Abstract. We describe an extension of Hoare's logic for reasoning about programs that alter data structures. We consider a low-level storage model based on a heap with associated lookup, update, allocation and deallocation operations, and unrestricted address arithmetic. The assertion language is based on a possible worlds model of the logic of bunched implications, and includes spatial conjunction and implication connectives alongside those of classical logic. Heap operations are axiomatized using what we call the “small axioms”, each of which mentions only those cells accessed by a particular command. Through these and a number of examples we show that the formalism supports local reasoning: A specification and proof can concentrate on only those cells in memory that a program accesses.

This paper builds on earlier work by Burstall, Reynolds, Ishtiaq and O'Hearn on reasoning about data structures.

1 Introduction

Pointers have been a persistent trouble area in program proving. The main difficulty is not one of finding an in-principle adequate axiomatization of pointer operations; rather there is a mismatch between simple intuitions about the way that pointer operations work and the complexity of their axiomatic treatments. For example, pointer assignment is operationally simple, but when there is aliasing, arising from several pointers to a given cell, then an alteration to that cell may affect the values of many syntactically unrelated expressions. (See [20, 2, 4, 6] for discussion and references to the literature on reasoning about pointers.)

We suggest that the source of this mismatch is the global view of state taken in most formalisms for reasoning about pointers. In contrast, programmers reason informally in a local way. Data structure algorithms typically work by applying local surgeries that rearrange small parts of a data structure, such as rotating a small part of a tree or inserting a node into a list. Informal reasoning usually concentrates on the effects of these surgeries, without picturing the entire memory of a system. We summarize this local reasoning viewpoint as follows.

To understand how a program works, it should be possible for reasoning and specification to be confined to the cells that the program actually accesses. The value of any other cell will automatically remain unchanged.

“it should be possible for reasoning and specification to be confined to the cells that the program actually accesses. The value of any other cell will automatically remain unchanged”
Some abbreviations

\[ a \mapsto _\_ \triangleq \exists \nu. \ a \mapsto \nu \quad \text{(for } \nu \text{ fresh)} \]

\[ a_1 \dot{=} a_2 \triangleq (a_1 = a_2) \land \text{emp} \]

\[ a \mapsto \langle a_0, \ldots, a_k \rangle \triangleq (a \mapsto a_0)^* \cdots ^* (a + k \mapsto a_k) \]
Local axioms: write

\[
\{a_1 \mapsto _\} [a_1] := a_2 \{a_1 \mapsto a_2\}
\]

\[
\{x \mapsto _\} [x] := y \{x \mapsto y\}
\]
Local axioms: read

\[
\{ a \mapsto v \land x = x' \} \ x := [a] \ \{ x = v \land a[x'/x] \mapsto v \}
\]

\[
\{ y \mapsto v \} \ x := [y] \ \{ x = v \land y \mapsto v \}
\]
Local axioms: allocation

\[
\{\text{emp}\} \ x := \text{alloc()} \ \{x \mapsto _\}\]

Local axioms: dispose

\[
\{a \mapsto _\} \text{free}(a) \{\text{emp}\}
\]

\[
\{x \mapsto _\} \text{free}(x) \{\text{emp}\}
\]
Local axioms: allocation

\[
\{ x \not\!\not\!\not\!\not\!\not\!\not\not= x' \} \ x := \ \text{cons}(a_1, \ldots, a_k) \ \{ x \mapsto \langle a_1[x'/x], \ldots, a_k[x'/x] \rangle \} 
\]
Frame rule

\[
\begin{array}{c}
\{P\} \ r \ \{Q\} \\
\{P \ast R\} \ r \ \{Q \ast R\}
\end{array}
\]

\[\text{mod}(r) \cap \text{free}(R) = \emptyset\]
Example

\[
\{ x \mapsto _* y \mapsto _* z \mapsto _ \} \\
\{ x \mapsto _ \} \\
\{ x \mapsto 1 \} \\
\{ x \mapsto 1 * y \mapsto _* z \mapsto _ \} \\
\{ x \mapsto 1 * y \mapsto 2 * z \mapsto 3 \}
\]

frame rule
write axiom

\[ [x] := 1; \]
\[ [y] := 2; \]
\[ [z] := 3; \]
Example

\{ \text{list}(x) \land x \neq \text{nil} \} \equiv \{ x \mapsto v \ast \text{list}(v) \}

\{ x \mapsto v \ast \text{list}(v) \}

\begin{align*}
\text{frame} & \quad \{ x \mapsto v \} \quad t := [x]; \quad \{ x \mapsto v \land t = v \} \\
\text{frame} & \quad \{ x \mapsto t \ast \text{list}(t) \}
\end{align*}

\begin{align*}
\text{free} & \quad \{ x \mapsto t \} \quad \text{free}(x); \quad \{ \text{emp} \} \\
\text{free} & \quad \{ \text{emp} \ast \text{list}(t) \} \equiv \{ \text{list}(t) \}
\end{align*}

\{ \text{list}(t) \}

\begin{align*}
\text{ls}(a_1, a_2) & \triangleq (a_1 = a_2 \land \text{emp}) \lor (a_1 \neq a_2 \land \exists v. \ a_1 \mapsto v \ast \text{ls}(v, a_2)) \\
\text{list}(a) & \triangleq \text{ls}(a, \text{nil})
\end{align*}
Example

\{ x \mapsto v \ast \text{list}(v) \ast \text{list}(y) \}\n
t := x;
	n := [t];

while \( n \neq \text{nil} \) do ( 

t := n;
	n := [t];
)

[t] := y;

\{ \text{list}(x) \}\n
\text{list}(a) \triangleq \text{ls}(a, \text{nil})

\text{ls}(a_1, a_2) \triangleq (a_1 = a_2 \land \text{emp}) \lor (a_1 \neq a_2 \land \exists v. a_1 \mapsto v \ast \text{ls}(v, a_2))
Example

\{ x \mapsto v \ast \text{list}(v) \ast \text{list}(y) \} \\
\{ x \mapsto v \ast \text{list}(v) \} \\
\ t := x; \\
\{ \text{ls}(x, t) \ast t \mapsto v \ast \text{list}(v) \} \\
\{ t \mapsto v \} \ n := [t]; \ \{ t \mapsto v \land n = v \} \\
\{ \text{ls}(x, t) \ast t \mapsto n \ast \text{list}(n) \} \\
\ \ \ \ \text{while } n \neq \text{nil} \ \text{do (} \\
\ { \text{ls}(x, t) \ast t \mapsto n \ast \text{list}(n) \land n \neq \text{nil} } \equiv { \text{ls}(x, t) \ast t \mapsto n \ast n \mapsto w \ast \text{list}(w) } \\
\ \ \ \ \ t := n; \\
\{ \text{ls}(x, t') \ast t' \mapsto t \ast t \mapsto w \ast \text{list}(w) \land t = n \} \\
\{ t \mapsto w \land t = n \} \ n := [t]; \ \{ t \mapsto w \land n = w \} \\
\{ \text{ls}(x, t) \ast t \mapsto n \ast \text{list}(n) \} \\
\{ \text{ls}(x, t) \ast t \mapsto n \ast \text{list}(n) \land n = \text{nil} \Rightarrow { \text{ls}(x, t) \ast t \mapsto \_ } \} \\
\{ t \mapsto \_ \} \ [t] := y; \ { t \mapsto y } \\
\{ \text{ls}(x, t) \ast t \mapsto y \ast \text{list}(y) \} \equiv { \text{list}(x) } \\
\text{ls}(a_1, a_2) \triangleq (a_1 = a_2 \land \text{emp}) \lor (a_1 \neq a_2 \land \exists v. a_1 \mapsto v \ast \text{ls}(v, a_2)) \\
\text{list}(a) \triangleq \text{ls}(a, \text{nil})
Correctness and (in)completeness
Relational semantics

\[
\llbracket \text{skip} \rrbracket \triangleq \{ (\sigma, \sigma) \}
\]

\[
\llbracket b? \rrbracket \triangleq \{ (\sigma, \sigma) \mid \sigma = \langle s, h \rangle \land s \models b \}
\]

\[
\llbracket x := a \rrbracket \triangleq \{ \langle \langle s, h \rangle, \langle s[x \mapsto \llbracket a \rrbracket s] \rangle, h \rangle \} \}
\]

\[
\llbracket x := [a] \rrbracket \triangleq \{ \langle \langle s, h \rangle, \langle s[x \mapsto v], h \rangle \rangle \mid v = h([a]s) \in \mathbb{Z} \}
\]

\[
\llbracket [a_1] := a_2 \rrbracket \triangleq \{ \langle \langle s, h \rangle, \langle s, h[[a_1]s \mapsto [a_2]s] \rangle \rangle \mid h([a_1]s) \in \mathbb{Z} \}
\]

\[
\llbracket x := \text{alloc()} \rrbracket \triangleq \{ \langle \langle s, h \rangle, \langle s[x \mapsto n], h[n \mapsto v] \rangle \rangle \mid v \in \mathbb{Z} \land (n \notin \text{dom}(h) \lor h(n) = \perp) \}
\]

\[
\llbracket \text{free}(x) \rrbracket \triangleq \{ \langle \langle s, h \cdot [s(x) \mapsto v] \rangle, \langle s, h \rangle \rangle \mid s(x) \in \mathbb{N} \land v \in \mathbb{Z} \}
\]

\[
\llbracket x := \text{cons}(a_0, \ldots, a_k) \rrbracket \triangleq \{ \langle \langle s, h \rangle, \langle s[x \mapsto n], h[n \mapsto [a_0]s, \ldots, (n + k) \mapsto [a_k]s] \rangle \rangle \mid (\forall i \in [n, n + k]. i \notin \text{dom}(h) \lor h(i) = \perp) \}
\]
Correctness

Th. [correctness]
If \( \{ P \} \overset{r}{\rightarrow} \{ Q \} \) then \([r]P \subseteq Q\)

Proof. By induction on the derivation.
Incompleteness

Th. [incompleteness]
There exists valid SL triples that are not provable

Proof. Misses footprint theorem: see slides on ISL
Final considerations on SL

Simple Proofs for Simple Programs

SL address resource manipulation
separating conjunction for in-place reasoning
pre/post describe local surgeries
Coming next

Let us explore some combination of different deductive systems
Questions
Question 1

Can you find some state that satisfies the following assertions?

\((x \triangleq y) \cdot x \mapsto y\)  

\((x = y) \cdot x \mapsto y\)  

\((x = y) \land x \mapsto y\)  

\(x \mapsto x\)
Question 2

Show a state that satisfies the assertion \((x \mapsto y) \ast \neg (x \mapsto y)\)

\[
\text{dom}(h) = \{11\}
\]
\[
s(x) = 11 \quad h(11) = 42
\]
\[
s(y) = 42
\]
Consider the imprecise list segment definition below
\[ \text{ils}(a_1, a_2) \triangleq (a_1 = a_2 \land \text{emp}) \lor (\exists v . a_1 \mapsto v \star \text{ls}(v, a_2)) \]

Prove that \( \text{ils}(a_1, a_2) \not\equiv \text{ls}(a_1, a_2) \) by finding a state that distinguishes \( \text{ls}(11, 11) \) from \( \text{ils}(11, 11) \)
* Exam 14

Complete the following derivations, if possible

$$\{ P^* x \mapsto _\_ \} [x] := 11 \{ P^*?? \}$$
$$\{ \text{true} \} [x] := 11 \{ ?? \}$$

$$\{ P^* x \mapsto _\_ \} \text{ free}(x) \{ ?? \}$$
$$\{ \text{true} \} \text{ free}(x) \{ ?? \}$$