

Program Analysis (70020)

Correctness of an Analysis

Herbert Wiklicky

Department of Computing
Imperial College London

`herbert@doc.ic.ac.uk`
`h.wiklicky@imperial.ac.uk`

Autumn 2023

Correctness

Questions: Is a program analysis **correct**? Are the results reflecting what is really happening when the program is run?

Correctness

Questions: Is a program analysis **correct**? Are the results reflecting what is really happening when the program is run?

In other words: What is the relation between the (concrete) semantics of a program, i.e. the transition relation \Rightarrow and/or its transitive closure \Rightarrow^* , and the (solutions to) the program analysis $Analysis_{\circ}$ and $Analysis_{\bullet}$.

Correctness

Questions: Is a program analysis **correct**? Are the results reflecting what is really happening when the program is run?

In other words: What is the relation between the (concrete) semantics of a program, i.e. the transition relation \Rightarrow and/or its transitive closure \Rightarrow^* , and the (solutions to) the program analysis *Analysis*_o and *Analysis*_•.

For example: Is a variable *LV* identifies as 'live' indeed useful, or more importantly, is a 'non-live' variable really 'dead', i.e. is it safe to eliminate it (at least locally).

Syntax of WHILE

The **labelled** syntax of the language WHILE is given by the following **abstract syntax**:

a

b

S

Syntax of WHILE

The **labelled** syntax of the language WHILE is given by the following **abstract syntax**:

$$a ::= x \mid n \mid a_1 \text{ op}_a a_2$$
$$b$$
$$S$$

Syntax of WHILE

The **labelled** syntax of the language WHILE is given by the following **abstract syntax**:

$$a ::= x \mid n \mid a_1 \text{ op}_a a_2$$
$$b ::= \text{true} \mid \text{false} \mid \text{not } b \mid b_1 \text{ op}_b b_2 \mid a_1 \text{ op}_r a_2$$
$$S$$

Syntax of WHILE

The **labelled** syntax of the language WHILE is given by the following **abstract syntax**:

$$a ::= x \mid n \mid a_1 \text{ op}_a a_2$$
$$b ::= \text{true} \mid \text{false} \mid \text{not } b \mid b_1 \text{ op}_b b_2 \mid a_1 \text{ op}_r a_2$$
$$S ::= [x := a]^\ell \\ \mid [\text{skip}]^\ell \\ \mid S_1; S_2 \\ \mid \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2 \\ \mid \text{while } [b]^\ell \text{ do } S$$

Sketches of a Formal Semantics

Memory is modelled by an abstract **state**, i.e. functions of type

$$\mathbf{State} = \mathbf{Var} \rightarrow \mathbf{Z}.$$

Sketches of a Formal Semantics

Memory is modelled by an abstract **state**, i.e. functions of type

$$\mathbf{State} = \mathbf{Var} \rightarrow \mathbf{Z}.$$

For boolean and arithmetic **expressions** we assume that we know what they “evaluate to” in a state $s \in \mathbf{State}$.

Sketches of a Formal Semantics

Memory is modelled by an abstract **state**, i.e. functions of type

$$\mathbf{State} = \mathbf{Var} \rightarrow \mathbf{Z}.$$

For boolean and arithmetic **expressions** we assume that we know what they “evaluate to” in a state $s \in \mathbf{State}$. Then the semantics for **AExp** is a *total* function

$$\llbracket \cdot \rrbracket_{\mathcal{A}} \cdot : \mathbf{AExp} \rightarrow \mathbf{State} \rightarrow \mathbf{Z}$$

and the semantics of boolean expressions is given by

$$\llbracket \cdot \rrbracket_{\mathcal{B}} \cdot : \mathbf{BExp} \rightarrow \mathbf{State} \rightarrow \{\mathbf{tt}, \mathbf{ff}\}$$

Evaluating Expressions

Let us look at a program with two variables **Var** = {*x*, *y*}.

Evaluating Expressions

Let us look at a program with two variables **Var** = {*x*, *y*}.

Two possible states in this case could be for example:

$$s_0 = [x \mapsto 0, y \mapsto 1] \text{ and } s_1 = [x \mapsto 1, y \mapsto 1]$$

Evaluating Expressions

Let us look at a program with two variables $\mathbf{Var} = \{x, y\}$.
Two possible states in this case could be for example:

$$s_0 = [x \mapsto 0, y \mapsto 1] \text{ and } s_1 = [x \mapsto 1, y \mapsto 1]$$

We can evaluate an expression like $x + y \in \mathbf{AExp}$:

$$[[x + y]]_{\mathcal{A}} s_0 = 0 + 1 = 1$$

$$[[x + y]]_{\mathcal{A}} s_1 = 1 + 1 = 2$$

Evaluating Expressions

Let us look at a program with two variables $\mathbf{Var} = \{x, y\}$.
Two possible states in this case could be for example:

$$s_0 = [x \mapsto 0, y \mapsto 1] \text{ and } s_1 = [x \mapsto 1, y \mapsto 1]$$

We can evaluate an expression like $x + y \in \mathbf{AExp}$:

$$\begin{aligned} \llbracket x + y \rrbracket_{\mathcal{A}} s_0 &= 0 + 1 = 1 \\ \llbracket x + y \rrbracket_{\mathcal{A}} s_1 &= 1 + 1 = 2 \end{aligned}$$

or a Boolean expression like $x + y \leq 1 \in \mathbf{BExp}$:

$$\begin{aligned} \llbracket x + y \leq 1 \rrbracket_{\mathcal{B}} s_0 &= 1 \leq 1 = \mathbf{tt} \\ \llbracket x + y \leq 1 \rrbracket_{\mathcal{B}} s_1 &= 2 \leq 1 = \mathbf{ff} \end{aligned}$$

Execution and Transitions

The **configurations** describe the current state of the execution.

$\langle S, s \rangle$... S is to be executed in state s ,
 s ... a terminal state (i.e. $\langle \cdot, s \rangle$).

Execution and Transitions

The **configurations** describe the current state of the execution.

$\langle S, s \rangle$... S is to be executed in state s ,
 s ... a terminal state (i.e. $\langle \cdot, s \rangle$).

The **transition relation** \Rightarrow specify the (possible) computational steps during the execution starting from a certain configuration

$$\langle S, s \rangle \Rightarrow \langle S', s' \rangle$$

and at the end of the computation

$$\langle S, s \rangle \Rightarrow s'$$

Execution Rules (SOS) [Provided in Exam]

$$\text{(ass)} \quad \langle [x := a]^{\ell}, s \rangle \Rightarrow s[x \mapsto \llbracket a \rrbracket_{\mathcal{A}}]s$$

Execution Rules (SOS) [Provided in Exam]

(ass) $\langle [x := a]^\ell, s \rangle \Rightarrow s[x \mapsto \llbracket a \rrbracket_{\mathcal{A}}]s$

(skip) $\langle [\mathbf{skip}]^\ell, s \rangle \Rightarrow s$

Execution Rules (SOS) [Provided in Exam]

$$\text{(ass)} \quad \langle [x := a]^\ell, s \rangle \Rightarrow s[x \mapsto \llbracket a \rrbracket_{\mathcal{A}} s]$$

$$\text{(skip)} \quad \langle [\mathbf{skip}]^\ell, s \rangle \Rightarrow s$$

$$\text{(sq}^1\text{)} \quad \frac{\langle S_1, s \rangle \Rightarrow \langle S'_1, s' \rangle}{\langle S_1; S_2, s \rangle \Rightarrow \langle S'_1; S_2, s' \rangle}$$

Execution Rules (SOS) [Provided in Exam]

$$\text{(ass)} \quad \langle [x := a]^\ell, s \rangle \Rightarrow s[x \mapsto \llbracket a \rrbracket_{\mathcal{A}} s]$$

$$\text{(skip)} \quad \langle [\mathbf{skip}]^\ell, s \rangle \Rightarrow s$$

$$\text{(sq}^1) \quad \frac{\langle S_1, s \rangle \Rightarrow \langle S'_1, s' \rangle}{\langle S_1; S_2, s \rangle \Rightarrow \langle S'_1; S_2, s' \rangle}$$

$$\text{(sq}^T) \quad \frac{\langle S_1, s \rangle \Rightarrow s'}{\langle S_1; S_2, s \rangle \Rightarrow \langle S_2, s' \rangle}$$

Execution Rules (SOS) [Provided in Exam]

$$\text{(ass)} \quad \langle [x := a]^\ell, s \rangle \Rightarrow s[x \mapsto \llbracket a \rrbracket_{\mathcal{A}} s]$$

$$\text{(skip)} \quad \langle [\mathbf{skip}]^\ell, s \rangle \Rightarrow s$$

$$\text{(sq}^1) \quad \frac{\langle S_1, s \rangle \Rightarrow \langle S'_1, s' \rangle}{\langle S_1; S_2, s \rangle \Rightarrow \langle S'_1; S_2, s' \rangle}$$

$$\text{(sq}^T) \quad \frac{\langle S_1, s \rangle \Rightarrow s'}{\langle S_1; S_2, s \rangle \Rightarrow \langle S_2, s' \rangle}$$

Execution Rules (SOS) [Provided in Exam]

$$\text{(ass)} \quad \langle [x := a]^\ell, s \rangle \Rightarrow s[x \mapsto \llbracket a \rrbracket_{\mathcal{A}} s]$$

$$\text{(skip)} \quad \langle [\mathbf{skip}]^\ell, s \rangle \Rightarrow s$$

$$\text{(sq}^1) \quad \frac{\langle S_1, s \rangle \Rightarrow \langle S'_1, s' \rangle}{\langle S_1; S_2, s \rangle \Rightarrow \langle S'_1; S_2, s' \rangle}$$

$$\text{(sq}^T) \quad \frac{\langle S_1, s \rangle \Rightarrow s'}{\langle S_1; S_2, s \rangle \Rightarrow \langle S_2, s' \rangle}$$

$$\text{(if}^T) \quad \langle \mathbf{if} [b]^\ell \mathbf{then } S_1 \mathbf{ else } S_2, s \rangle \Rightarrow \langle S_1, s \rangle$$

$$\text{if } \llbracket b \rrbracket_{\mathcal{B}} s = \mathbf{tt}$$

Execution Rules (SOS) [Provided in Exam]

$$\text{(ass)} \quad \langle [x := a]^\ell, s \rangle \Rightarrow s[x \mapsto \llbracket a \rrbracket_{\mathcal{A}} s]$$

$$\text{(skip)} \quad \langle [\text{skip}]^\ell, s \rangle \Rightarrow s$$

$$\text{(sq}^1) \quad \frac{\langle S_1, s \rangle \Rightarrow \langle S'_1, s' \rangle}{\langle S_1; S_2, s \rangle \Rightarrow \langle S'_1; S_2, s' \rangle}$$

$$\text{(sq}^T) \quad \frac{\langle S_1, s \rangle \Rightarrow s'}{\langle S_1; S_2, s \rangle \Rightarrow \langle S_2, s' \rangle}$$

$$\text{(if}^T) \quad \langle \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2, s \rangle \Rightarrow \langle S_1, s \rangle$$

$$\text{if } \llbracket b \rrbracket_{\mathcal{B}} s = \mathbf{tt}$$

$$\text{(if}^F) \quad \langle \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2, s \rangle \Rightarrow \langle S_2, s \rangle$$

$$\text{if } \llbracket b \rrbracket_{\mathcal{B}} s = \mathbf{ff}$$

Execution Rules (SOS) [Provided in Exam]

$$\text{(ass)} \quad \langle [x := a]^\ell, s \rangle \Rightarrow s[x \mapsto \llbracket a \rrbracket_{\mathcal{A}} s]$$

$$\text{(skip)} \quad \langle [\text{skip}]^\ell, s \rangle \Rightarrow s$$

$$\text{(sq}^1) \quad \frac{\langle S_1, s \rangle \Rightarrow \langle S'_1, s' \rangle}{\langle S_1; S_2, s \rangle \Rightarrow \langle S'_1; S_2, s' \rangle}$$

$$\text{(sq}^T) \quad \frac{\langle S_1, s \rangle \Rightarrow s'}{\langle S_1; S_2, s \rangle \Rightarrow \langle S_2, s' \rangle}$$

$$\text{(if}^T) \quad \langle \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2, s \rangle \Rightarrow \langle S_1, s \rangle \quad \text{if } \llbracket b \rrbracket_{\mathcal{B}} s = \mathbf{tt}$$

$$\text{(if}^F) \quad \langle \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2, s \rangle \Rightarrow \langle S_2, s \rangle \quad \text{if } \llbracket b \rrbracket_{\mathcal{B}} s = \mathbf{ff}$$

$$\text{(wh}^T) \quad \langle \text{while } [b]^\ell \text{ do } S, s \rangle \Rightarrow \langle S; \text{while } [b]^\ell \text{ do } S, s \rangle \quad \text{if } \llbracket b \rrbracket_{\mathcal{B}} s = \mathbf{tt}$$

Execution Rules (SOS) [Provided in Exam]

$$\text{(ass)} \quad \langle [x := a]^\ell, s \rangle \Rightarrow s[x \mapsto \llbracket a \rrbracket_{\mathcal{A}} s]$$

$$\text{(skip)} \quad \langle [\text{skip}]^\ell, s \rangle \Rightarrow s$$

$$\text{(sq}^1) \quad \frac{\langle S_1, s \rangle \Rightarrow \langle S'_1, s' \rangle}{\langle S_1; S_2, s \rangle \Rightarrow \langle S'_1; S_2, s' \rangle}$$

$$\text{(sq}^T) \quad \frac{\langle S_1, s \rangle \Rightarrow s'}{\langle S_1; S_2, s \rangle \Rightarrow \langle S_2, s' \rangle}$$

$$\text{(if}^T) \quad \langle \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2, s \rangle \Rightarrow \langle S_1, s \rangle \quad \text{if } \llbracket b \rrbracket_{\mathcal{B}} s = \mathbf{tt}$$

$$\text{(if}^F) \quad \langle \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2, s \rangle \Rightarrow \langle S_2, s \rangle \quad \text{if } \llbracket b \rrbracket_{\mathcal{B}} s = \mathbf{ff}$$

$$\text{(wh}^T) \quad \langle \text{while } [b]^\ell \text{ do } S, s \rangle \Rightarrow \langle S; \text{while } [b]^\ell \text{ do } S, s \rangle \quad \text{if } \llbracket b \rrbracket_{\mathcal{B}} s = \mathbf{tt}$$

$$\text{(wh}^F) \quad \langle \text{while } [b]^\ell \text{ do } S, s \rangle \Rightarrow s \quad \text{if } \llbracket b \rrbracket_{\mathcal{B}} s = \mathbf{ff}$$

A SOS Example

Consider a (perhaps rather vacuous) program like:

$$S \equiv [z := x + y]^{\ell}; \mathbf{while} [true]^{\ell'} \mathbf{do} [skip]^{\ell''}$$

A SOS Example

Consider a (perhaps rather vacuous) program like:

$$S \equiv [z := x + y]^{\ell}; \mathbf{while} [true]^{\ell'} \mathbf{do} [\mathbf{skip}]^{\ell''}$$

$$s_0 = [x \mapsto 0, y \mapsto 1, z \mapsto 0] \text{ and } s_1 = [x \mapsto 0, y \mapsto 1, z \mapsto 1]$$

A SOS Example

Consider a (perhaps rather vacuous) program like:

$$S \equiv [z := x + y]^\ell; \mathbf{while} [true]^\ell \mathbf{do} [\mathbf{skip}]^\ell$$

$$s_0 = [x \mapsto 0, y \mapsto 1, z \mapsto 0] \text{ and } s_1 = [x \mapsto 0, y \mapsto 1, z \mapsto 1]$$

Then $\langle S, s_0 \rangle$ executes as follows:

$$\langle S, s_0 \rangle \Rightarrow$$

A SOS Example

Consider a (perhaps rather vacuous) program like:

$$S \equiv [z := x + y]^{\ell}; \mathbf{while} [true]^{\ell'} \mathbf{do} [\mathbf{skip}]^{\ell''}$$

$$s_0 = [x \mapsto 0, y \mapsto 1, z \mapsto 0] \text{ and } s_1 = [x \mapsto 0, y \mapsto 1, z \mapsto 1]$$

Then $\langle S, s_0 \rangle$ executes as follows:

$$\langle S, s_0 \rangle \Rightarrow \langle \mathbf{while} [true]^{\ell'} \mathbf{do} [\mathbf{skip}]^{\ell''}, s_1 \rangle$$

A SOS Example

Consider a (perhaps rather vacuous) program like:

$$S \equiv [z := x + y]^{\ell}; \mathbf{while} [true]^{\ell'} \mathbf{do} [\mathbf{skip}]^{\ell''}$$

$$s_0 = [x \mapsto 0, y \mapsto 1, z \mapsto 0] \text{ and } s_1 = [x \mapsto 0, y \mapsto 1, z \mapsto 1]$$

Then $\langle S, s_0 \rangle$ executes as follows:

$$\begin{aligned} \langle S, s_0 \rangle &\Rightarrow \langle \mathbf{while} [true]^{\ell'} \mathbf{do} [\mathbf{skip}]^{\ell''}, s_1 \rangle \\ &\Rightarrow \langle [\mathbf{skip}]^{\ell''}; \mathbf{while} [true]^{\ell'} \mathbf{do} [\mathbf{skip}]^{\ell''}, s_1 \rangle \end{aligned}$$

A SOS Example

Consider a (perhaps rather vacuous) program like:

$$S \equiv [z := x + y]^{\ell}; \text{while } [true]^{\ell'} \text{ do } [\text{skip}]^{\ell''}$$

$$s_0 = [x \mapsto 0, y \mapsto 1, z \mapsto 0] \text{ and } s_1 = [x \mapsto 0, y \mapsto 1, z \mapsto 1]$$

Then $\langle S, s_0 \rangle$ executes as follows:

$$\begin{aligned} \langle S, s_0 \rangle &\Rightarrow \langle \text{while } [true]^{\ell'} \text{ do } [\text{skip}]^{\ell''}, s_1 \rangle \\ &\Rightarrow \langle [\text{skip}]^{\ell''}; \text{while } [true]^{\ell'} \text{ do } [\text{skip}]^{\ell''}, s_1 \rangle \\ &\Rightarrow \langle \text{while } [true]^{\ell'} \text{ do } [\text{skip}]^{\ell''}, s_1 \rangle \end{aligned}$$

A SOS Example

Consider a (perhaps rather vacuous) program like:

$$S \equiv [z := x + y]^{\ell}; \mathbf{while} [true]^{\ell'} \mathbf{do} [skip]^{\ell''}$$

$$s_0 = [x \mapsto 0, y \mapsto 1, z \mapsto 0] \text{ and } s_1 = [x \mapsto 0, y \mapsto 1, z \mapsto 1]$$

Then $\langle S, s_0 \rangle$ executes as follows:

$$\begin{aligned} \langle S, s_0 \rangle &\Rightarrow \langle \mathbf{while} [true]^{\ell'} \mathbf{do} [skip]^{\ell''}, s_1 \rangle \\ &\Rightarrow \langle [skip]^{\ell''}; \mathbf{while} [true]^{\ell'} \mathbf{do} [skip]^{\ell''}, s_1 \rangle \\ &\Rightarrow \langle \mathbf{while} [true]^{\ell'} \mathbf{do} [skip]^{\ell''}, s_1 \rangle \\ &\Rightarrow \langle [skip]^{\ell''}; \mathbf{while} [true]^{\ell'} \mathbf{do} [skip]^{\ell''}, s_1 \rangle \end{aligned}$$

A SOS Example

Consider a (perhaps rather vacuous) program like:

$$S \equiv [z := x + y]^{\ell}; \text{while } [true]^{\ell'} \text{ do } [\text{skip}]^{\ell''}$$

$$s_0 = [x \mapsto 0, y \mapsto 1, z \mapsto 0] \text{ and } s_1 = [x \mapsto 0, y \mapsto 1, z \mapsto 1]$$

Then $\langle S, s_0 \rangle$ executes as follows:

$$\begin{aligned} \langle S, s_0 \rangle &\Rightarrow \langle \text{while } [true]^{\ell'} \text{ do } [\text{skip}]^{\ell''}, s_1 \rangle \\ &\Rightarrow \langle [\text{skip}]^{\ell''}; \text{while } [true]^{\ell'} \text{ do } [\text{skip}]^{\ell''}, s_1 \rangle \\ &\Rightarrow \langle \text{while } [true]^{\ell'} \text{ do } [\text{skip}]^{\ell''}, s_1 \rangle \\ &\Rightarrow \langle [\text{skip}]^{\ell''}; \text{while } [true]^{\ell'} \text{ do } [\text{skip}]^{\ell''}, s_1 \rangle \\ &\Rightarrow \dots \end{aligned}$$

Lemma 1

- (i) If $\langle S, s \rangle \Rightarrow s'$ then
 $final(S) = \{init(S)\}$.
- (ii) If $\langle S, s \rangle \Rightarrow \langle S', s' \rangle$ then
 $final(S) \supseteq final(S')$.
- (iii) If $\langle S, s \rangle \Rightarrow \langle S', s' \rangle$ then
 $flow(S) \supseteq flow(S')$.
- (iv) If $\langle S, s \rangle \Rightarrow \langle S', s' \rangle$ then
 $blocks(S) \supseteq blocks(S')$.
- (v) If $\langle S, s \rangle \Rightarrow \langle S', s' \rangle$ then
S label consistent implies S' label consistent.

Lemma 1 - Proof (i) [Not for Exam]

Proof.

The proof is by induction on the shape of the inference tree.
Consider the only three non-vacuous cases:

$$\text{(ass): } \langle [x := a]^\ell, s \rangle \Rightarrow s[x \mapsto \llbracket a \rrbracket s]$$

$$\text{final}([x := a]^\ell) = \{\ell\} = \{\text{init}([x := a]^\ell)\}.$$

$$\text{(skip): } \langle [\text{skip}]^\ell, s \rangle \Rightarrow s$$

$$\text{final}([\text{skip}]^\ell) = \{\ell\} = \{\text{init}([\text{skip}]^\ell)\}.$$

$$\text{(wh}^F\text{): } \langle \text{while } [b]^\ell \text{ do } S, s \rangle \Rightarrow s \text{ with } \llbracket b \rrbracket = \mathbf{false}$$

$$\text{final}(\text{while } [b]^\ell \text{ do } S) = \{\ell\} = \{\text{init}(\text{while } [b]^\ell \text{ do } S)\}.$$

Lemma 1 - Proof (ii) [Not for Exam]

Proof (cont).

(seq¹): $\langle S_1; S_2, s \rangle \Rightarrow \langle S'_1; S_2, s' \rangle$ because
 $\langle S_1, s \rangle \Rightarrow \langle S'_1, s' \rangle$:

$$\text{final}(S_1; S_2) = \text{final}(S_2) = \text{final}(S'_1; S_2).$$

(seq^T): ...

(if^T): $\langle \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2, s \rangle \Rightarrow \langle S_1, s \rangle$
with $\llbracket b \rrbracket = \mathbf{true}$:

$$\begin{aligned} \text{final}(\text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2) &= \\ \text{final}(S_1) \cup \text{final}(S_2) &\supseteq \text{final}(S_1). \end{aligned}$$

(if^F): ...

(wh^T): ...



LV Equations: $LV^{\#}$

The Live Variable Analysis is given as the solution to the following system of **equations**:

LV Equations: LV^{\leftarrow}

The Live Variable Analysis is given as the solution to the following system of **equations**:

$$LV_{exit}(\ell) = \begin{cases} \emptyset, & \text{if } \ell \in final(S_*) \\ \bigcup \{LV_{entry}(\ell') \mid (\ell', \ell) \in flow^R(S_*)\}, & \text{otherwise} \end{cases}$$

$$LV_{entry}(\ell) = (LV_{exit}(\ell) \setminus kill_{LV}([B]^\ell) \cup gen_{LV}([B]^\ell))$$

where $[B]^\ell \in blocks(S_*)$

Solutions via Iteration Operator

$$LV_{entry}(1) = \mathbf{F}_1^\bullet(LV_{entry}(1), \dots, LV_{exit}(n))$$

... ..

$$LV_{entry}(n) = \mathbf{F}_n^\bullet(LV_{entry}(1), \dots, LV_{exit}(n))$$

$$LV_{exit}(1) = \mathbf{F}_1^\circ(LV_{entry}(1), \dots, LV_{exit}(n))$$

... ..

$$LV_{exit}(n) = \mathbf{F}_n^\circ(LV_{entry}(1), \dots, LV_{exit}(n))$$

becomes a function on the lattice $\mathcal{P}(\mathbf{Var})^{2n}$

$$\mathbf{F} : \mathcal{P}(\mathbf{Var})^{2n} \rightarrow \mathcal{P}(\mathbf{Var})^{2n}$$

$$\mathbf{F}_i^\bullet(LV_{entry}(1), \dots, LV_{exit}(n)) = LV_{entry}(i)$$

$$\mathbf{F}_i^\circ(LV_{entry}(1), \dots, LV_{exit}(n)) = LV_{exit}(i)$$

LV Constraints: LV^{\Leftarrow}

The Live Variable Analysis is equivalently given as the solution to the following system of **constraints**:

$$LV_{exit}(\ell) \supseteq \begin{cases} \emptyset, & \text{if } \ell \in final(S_*) \\ \bigcup \{LV_{entry}(\ell') \mid (\ell', \ell) \in flow^R(S_*)\}, & \text{otherwise} \end{cases}$$

LV Constraints: LV^{\Leftarrow}

The Live Variable Analysis is equivalently given as the solution to the following system of **constraints**:

$$LV_{exit}(\ell) \supseteq \begin{cases} \emptyset, & \text{if } \ell \in final(S_*) \\ \bigcup \{LV_{entry}(\ell') \mid (\ell', \ell) \in flow^R(S_*)\}, & \text{otherwise} \end{cases}$$

$$LV_{entry}(\ell) \supseteq (LV_{exit}(\ell) \setminus kill_{LV}([B]^\ell)) \cup gen_{LV}([B]^\ell) \\ \text{where } [B]^\ell \in blocks(S_*)$$

LV Solutions to $LV^=$ and LV^{\leq}

Consider collections $live = (live_{\text{entry}}, live_{\text{exit}})$ of functions:

$$live_{\text{entry}} : \mathbf{Lab}_{\star} \rightarrow \mathcal{P}(\mathbf{Var}_{\star})$$

$$live_{\text{exit}} : \mathbf{Lab}_{\star} \rightarrow \mathcal{P}(\mathbf{Var}_{\star})$$

LV Solutions to $LV^=$ and LV^{\subseteq}

Consider collections $live = (live_{\text{entry}}, live_{\text{exit}})$ of functions:

$$live_{\text{entry}} : \mathbf{Lab}_{\star} \rightarrow \mathcal{P}(\mathbf{Var}_{\star})$$

$$live_{\text{exit}} : \mathbf{Lab}_{\star} \rightarrow \mathcal{P}(\mathbf{Var}_{\star})$$

If $live$ solves $LV^=$ for a statement S we write:

$$live \models LV^=(S)$$

If $live$ solves LV^{\subseteq} for a statement S we write:

$$live \models LV^{\subseteq}(S)$$

Theorem 1

Given a label consistent program S_* .

If

▶ $live \models LV^=(S_*)$

then

▶ $live \models LV^{\subseteq}(S_*)$.

That is: The least solution of $LV^=(S_*)$ coincides with the least solution to $LV^{\subseteq}(S_*)$.

Theorem 1 - Proof [Not for Exam]

Proof.

If $live \models LV^=(S_*)$ also $live \models LV^{\subseteq}(S_*)$ as “ \supseteq ” includes “ $=$ ”.

Theorem 1 - Proof [Not for Exam]

Proof.

If $live \models LV^=(S_*)$ also $live \models LV^{\subseteq}(S_*)$ as “ \supseteq ” includes “ $=$ ”.

To show that $LV^=(S_*)$ and $LV^{\subseteq}(S_*)$ have the same least solution consider the iteration operator $\mathbf{F} = \mathbf{F}_{LV^=} = \mathbf{F}_{LV^{\subseteq}}^S$

$$live \models LV^{\subseteq}(S_*) \quad \text{iff} \quad live \supseteq \mathbf{F}(live)$$

$$live \models LV^=(S_*) \quad \text{iff} \quad live = \mathbf{F}(live)$$

By Tarski's Fixed Point Theorem we have:

$$lfp(\mathbf{F}) = \bigsqcap \{live \mid live \supseteq \mathbf{F}(live)\} = \bigsqcap \{live \mid live = \mathbf{F}(live)\}.$$

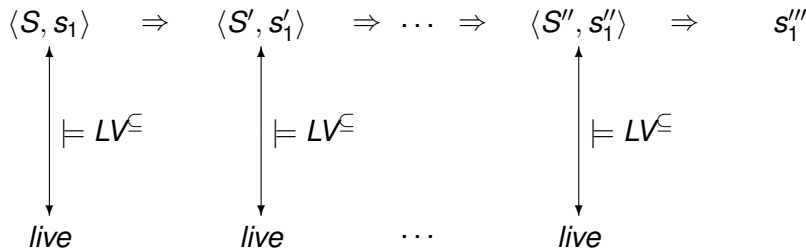
Since $lfp(\mathbf{F}) = \mathbf{F}(lfp(\mathbf{F}))$ and $lfp(\mathbf{F}) \supseteq \mathbf{F}(lfp(\mathbf{F}))$ we see that we get the same least solutions. □

Preservation of Solution

During the (actual) execution of any program S_* a solution to the Live Variable analysis $LV^{\subseteq}(S_*)$ remains a solution.

Preservation of Solution

During the (actual) execution of any program S_* a solution to the Live Variable analysis $LV^{\subseteq}(S_*)$ remains a solution.



Lemma 2

Given a label consistent program S_1 .

If

- ▶ $live \models LV^{\subseteq}(S_1)$ and
- ▶ $flow(S_1) \supseteq flow(S_2)$ and
- ▶ $blocks(S_1) \supseteq blocks(S_2)$

then

- ▶ $live \models LV^{\subseteq}(S_2)$

with S_2 being label consistent.

Lemma 2

Given a label consistent program S_1 .

If

- ▶ $live \models LV^{\subseteq}(S_1)$ and
- ▶ $flow(S_1) \supseteq flow(S_2)$ and
- ▶ $blocks(S_1) \supseteq blocks(S_2)$

then

- ▶ $live \models LV^{\subseteq}(S_2)$

with S_2 being label consistent.

Proof [Not for Exam].

If S_1 is label consistent and $blocks(S_1) \supseteq blocks(S_2)$ then S_2 is also label consistent.

If $live \models LV^{\subseteq}(S_1)$ then $live$ also satisfy each constraint in $LV^{\subseteq}(S_2)$ and hence $live \models LV^{\subseteq}(S_2)$. □

Lemma 3

Given a label consistent program S .

If

- ▶ $live \models LV^{\subseteq}(S)$ and
- ▶ $\langle S, s \rangle \Rightarrow \langle S', s' \rangle$

then

- ▶ $live \models LV^{\subseteq}(S')$.

Lemma 3

Given a label consistent program S .

If

- ▶ $live \models LV^{\subseteq}(S)$ and
- ▶ $\langle S, s \rangle \Rightarrow \langle S', s' \rangle$

then

- ▶ $live \models LV^{\subseteq}(S')$.

Proof [Not for Exam].

Follows directly from Lemma 1 and Lemma 2. □

Lemma 4

Given a label consistent program S .

If

▶ $live \models LV^{\subseteq}(S)$

then for all $(\ell, \ell') \in flow(S)$ we have:

▶ $live_{exit}(\ell) \supseteq live_{entry}(\ell')$

Lemma 4

Given a label consistent program S .

If

$$\blacktriangleright \textit{live} \models LV^{\subseteq}(S)$$

then for all $(\ell, \ell') \in \textit{flow}(S)$ we have:

$$\blacktriangleright \textit{live}_{\text{exit}}(\ell) \supseteq \textit{live}_{\text{entry}}(\ell')$$

Proof [Not for Exam].

Follows immediately from the construction of $LV^{\subseteq}(S)$. □

Correctness Relation

Assume that V is a set of *live variables*.

Define the **correctness relation** via

$$s_1 \sim_V s_2 \text{ iff } \forall x \in V : s_1(x) = s_2(x).$$

Correctness Relation

Assume that V is a set of *live variables*.

Define the **correctness relation** via

$$s_1 \sim_V s_2 \text{ iff } \forall x \in V : s_1(x) = s_2(x).$$

In other word:

Two states are equivalent iff for all live variables – i.e. all “practical purposes” – the states s_1 and s_2 agree on the variables in V .

Example

Consider $[x := y + z]^\ell$ and $V_1 = \{y, z\}$ and $V_2 = \{x\}$.

$s_1 \sim_{V_1} s_2$ means $s_1(y) = s_2(y) \wedge s_1(z) = s_2(z)$.

$s_1 \sim_{V_2} s_2$ means $s_1(x) = s_2(x)$.

Example

Consider $[x := y + z]^\ell$ and $V_1 = \{y, z\}$ and $V_2 = \{x\}$.

$s_1 \sim_{V_1} s_2$ means $s_1(y) = s_2(y) \wedge s_1(z) = s_2(z)$.

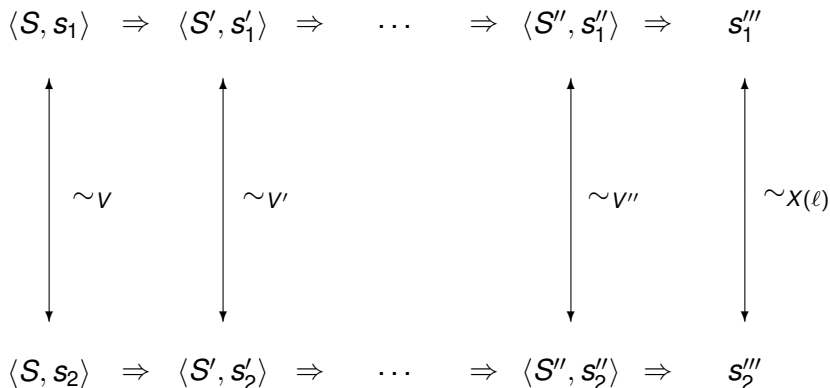
$s_1 \sim_{V_2} s_2$ means $s_1(x) = s_2(x)$.

Assume $\langle [x := y + z]^\ell, s_1 \rangle \Rightarrow s'_1$, $\langle [x := y + z]^\ell, s_2 \rangle \Rightarrow s'_2$ then

$s_1 \sim_{V_1} s_2$ ensures $s'_1 \sim_{V_2} s'_2$.

If $V_2 = \text{LV}_{\text{exit}}(\ell)$ thus is the set of live variables after $[x := y + z]^\ell$ then $V_1 = \text{LV}_{\text{entry}}(\ell)$ is the set of live variables before $[x := y + z]^\ell$.

Correctness of LV Analysis



$$V = N(\text{init}(S))$$

$$V' = N(\text{init}(S'))$$

$$V'' = N(\text{init}(S''))$$

$$\ell \in \text{final}(S)$$

Short-hand notation: $N(\ell) = \text{live}_{\text{entry}}(\ell)$ and $X(\ell) = \text{live}_{\text{exit}}(\ell)$.

Lemma 5

Given a label consistent program S .

If

▶ $live \models LV^{\subseteq}(S)$

then

▶ $s_1 \sim_{live_{exit}(l)} s_2$ implies $s_1 \sim_{live_{entry}(l')} s_2$ for all $(l, l') \in flow(S)$.

Lemma 5

Given a label consistent program S .

If

$$\blacktriangleright \textit{live} \models LV^{\subseteq}(S)$$

then

$$\blacktriangleright s_1 \sim_{\textit{live}_{\text{exit}(\ell)}} s_2 \text{ implies } s_1 \sim_{\textit{live}_{\text{entry}(\ell')}} s_2 \text{ for all } (\ell, \ell') \in \textit{flow}(S).$$

Proof [Not for Exam].

Follows directly from Lemma 4 and the definition of \sim_V . □

Theorem 2

Given a label consistent program S .

If

$$\blacktriangleright \text{live} \models LV^{\subseteq}(S)$$

then

- (i) If $\langle S, s_1 \rangle \Rightarrow \langle S', s'_1 \rangle$ and $s_1 \sim \text{live}_{\text{entry}}(\text{init}(S)) s_2$ then there exists s'_2 such that $\langle S, s_2 \rangle \Rightarrow \langle S', s'_2 \rangle$ and $s'_1 \sim \text{live}_{\text{entry}}(\text{init}(S')) s'_2$.
- (ii) If $\langle S, s_1 \rangle \Rightarrow s'_1$ and $s_1 \sim \text{live}_{\text{entry}}(\text{init}(S)) s_2$ then there exists s'_2 such that $\langle S, s_2 \rangle \Rightarrow s'_2$ and $s'_1 \sim \text{live}_{\text{exit}}(\text{init}(S)) s'_2$.

Theorem 2 - Proof [Not for Exam]

Proof.

The proof is by induction on the shape of the inference tree.

(ass): ...

(skip): ...

(seq¹): ...

(seq^T): ...

(if^T): ...

(if^F): ...

(wh^T): ...

(wh^F): ...

Theorem 2 - Proof (ass) [Not for Exam]

Proof (cont).

The proof is by induction on the shape of the inference tree.

(ass): We have $\langle [x := a]^\ell, s_1 \rangle \Rightarrow s_1[x \mapsto \llbracket a \rrbracket s_1]$ and from the specification of the constraints:

$$live_{\text{entry}}(\ell) = (live_{\text{exit}}(\ell) \setminus \{x\}) \cup FV(a)$$

and therefore

$$s_1 \sim_{live_{\text{entry}}(\ell)} s_2 \text{ implies } \llbracket a \rrbracket(s_1) = \llbracket a \rrbracket(s_2)$$

because the value of a depends only on variables in it.

Thus with $s'_2 = s_2[x \mapsto \llbracket a \rrbracket_{\mathcal{A}} s_2]$ we have $s'_1(x) = s'_2(x)$ and thus $s'_1 \sim_{live_{\text{exit}}(\ell)} s'_2$.

Theorem 2 - Proof (skip) [Not for Exam]

Proof (cont).

(skip): We have $\langle [\text{skip}]^\ell, s_1 \rangle \Rightarrow s_1$ and from the specification of the constraints we get:

$$\text{live}_{\text{entry}}(\ell) = (\text{live}_{\text{exit}}(\ell) \setminus \emptyset) \cup \emptyset = \text{live}_{\text{exit}}(\ell)$$

Thus taking s'_2 to be s_2 we get $s'_1 \sim \text{live}_{\text{exit}}(\ell) s'_2$ as required.

Theorem 2 - Proof (seq¹) [Not for Exam]

Proof (cont).

(seq¹): We have $\langle S_1; S_2, s_1 \rangle \Rightarrow \langle S'_1; S_2, s'_1 \rangle$ because of $\langle S_1, s_1 \rangle \Rightarrow \langle S'_1, s'_1 \rangle$.

By construction we have $flow(S_1; S_2) \supseteq flow(S_1)$ and also $blocks(S_1; S_2) \supseteq blocks(S_1)$, thus by Lemma 2 $live \models LV^{\subseteq}(S_1)$ and by the induction hypothesis there exists a s'_2 such that

$$\langle S_1, s_2 \rangle \Rightarrow \langle S'_1, s'_2 \rangle \text{ and } s'_1 \sim_{live_{entry}(init(s'_1))} s'_2$$

and the result follows.

Theorem 2 - Proof (seq^T) [Not for Exam]

Proof (cont).

(seq^T): We have $\langle S_1; S_2, s_1 \rangle \Rightarrow \langle S_2, s'_1 \rangle$ because of $\langle S_1, s_1 \rangle \Rightarrow s'_1$. Again by Lemma 2, $live$ is a solution to $LV^{\subseteq}(S_1)$ and thus by induction hypothesis there exists a s'_2 such that

$$\langle S_1, s_2 \rangle \Rightarrow s'_2 \text{ and } s'_1 \sim \text{live}_{\text{exit}}(\text{init}(S_1)) s'_2$$

Now we have:

$$\{(\ell, \text{init}(S_2)) \mid \ell \in \text{final}(S_1)\} \subseteq \text{flow}(S_1; S_2)$$

and by Lemma 1, $\text{final}(S_1) = \{\text{init}(S_1)\}$. Thus by Lemma 5

$$s'_1 \sim \text{live}_{\text{entry}}(\text{init}(S_2)) s'_2$$

and the result follows.

Theorem 2 - Proof (if^T) & (if^F) [Not for Exam]

Proof (cont).

(if^T): We have $\langle \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2, s_1 \rangle \Rightarrow \langle S_1, s_1 \rangle$
with $\llbracket b \rrbracket(s_1) = \mathbf{true}$.

Since $s_1 \sim \text{live}_{\text{entry}(\ell)} s_2$ and $\text{live}_{\text{entry}(\ell)} \supseteq FV(b)$ we
also have $\llbracket b \rrbracket(s_2) = \mathbf{true}$ (the value of b is only
dependent on the variables occurring in it) and
thus

$$\langle \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2, s_2 \rangle \Rightarrow \langle S_1, s_2 \rangle$$

From the constraints we get $\text{live}_{\text{entry}(\ell)} \supseteq \text{live}_{\text{exit}(\ell)}$
and hence $s_1 \sim \text{live}_{\text{exit}(\ell)} s_2$.

Since $(\ell, \text{init}(S_1)) \in \text{flow}(S)$ Lemma 5 gives

$s_1 \sim \text{live}_{\text{entry}(\text{init}(S_1))} s_2$ as required.

(if^F): similar to case (if^T).

Theorem 2 - Proof (wh^T) [Not for Exam]

Proof (cont).

(wh^T) : $\langle \text{while } [b]^\ell \text{ do } S, s_1 \rangle \Rightarrow \langle S; \text{while } [b]^\ell \text{ do } S, s_1 \rangle$
with $\llbracket b \rrbracket(s_1) = \mathbf{true}$.

Since $s_1 \sim \text{live}_{\text{entry}(\ell)} s_2$ and $\text{live}_{\text{entry}(\ell)} \supseteq FV(b)$ we also have $\llbracket b \rrbracket(s_2) = \mathbf{true}$ and thus

$\langle \text{while } [b]^\ell \text{ do } S, s_2 \rangle \Rightarrow \langle S; \text{while } [b]^\ell \text{ do } S, s_2 \rangle$

Again since $\text{live}_{\text{entry}(\ell)} \supseteq \text{live}_{\text{exit}(\ell)}$ we have $s_1 \sim \text{live}_{\text{exit}(\ell)} s_2$ and then

$$s_1 \sim \text{live}_{\text{entry}(\text{init}(S))} s_2$$

from Lemma 5 as

$(\ell, \text{init}(S)) \in \text{flow}(\text{while } [b]^\ell \text{ do } S)$.

Theorem 2 - Proof (wh^F) [Not for Exam]

Proof (cont).

(wh^F): We have $\langle \text{while } [b]^\ell \text{ do } S, s_1 \rangle \Rightarrow s_1$ with $\llbracket b \rrbracket(s_1) = \mathbf{false}$.

Since $s_1 \sim \text{live}_{\text{entry}(\ell)} s_2$ and $\text{live}_{\text{entry}(\ell)} \supseteq FV(b)$ and we also have $\llbracket b \rrbracket(s_2) = \mathbf{false}$ and thus:

$$\langle \text{while } [b]^\ell \text{ do } S, s_2 \rangle \Rightarrow s_2.$$

From the specification of LV^\subseteq we have $\text{live}_{\text{entry}(\ell)} \supseteq \text{live}_{\text{exit}(\ell)}$ and thus $s_1 \sim \text{live}_{\text{exit}(\ell)} s_2$.

□

Corollary 1

Given a label consistent program S .

If

$$\blacktriangleright \text{live} \models LV^{\subseteq}(S)$$

then

- (i) If $\langle S, s_1 \rangle \Rightarrow^* \langle S', s'_1 \rangle$ and $s_1 \sim \text{live}_{\text{entry}(\text{init}(S))} s_2$ then there exists s'_2 such that $\langle S, s_2 \rangle \Rightarrow^* \langle S', s'_2 \rangle$ and $s'_1 \sim \text{live}_{\text{entry}(\text{init}(S'))} s'_2$.
- (ii) If $\langle S, s_1 \rangle \Rightarrow^* s'_1$ and $s_1 \sim \text{live}_{\text{entry}(\text{init}(S))} s_2$ then there exists s'_2 such that $\langle S, s_2 \rangle \Rightarrow^* s'_2$ and $s'_1 \sim \text{live}_{\text{exit}(\ell)} s'_2$ for some $\ell \in \text{final}(S)$.

Corollary 1

Given a label consistent program S .

If

▶ $live \models LV^{\subseteq}(S)$

then

- (i) If $\langle S, s_1 \rangle \Rightarrow^* \langle S', s'_1 \rangle$ and $s_1 \sim_{live_{\text{entry}}(init(S))} s_2$ then there exists s'_2 such that $\langle S, s_2 \rangle \Rightarrow^* \langle S', s'_2 \rangle$ and $s'_1 \sim_{live_{\text{entry}}(init(S'))} s'_2$.
- (ii) If $\langle S, s_1 \rangle \Rightarrow^* s'_1$ and $s_1 \sim_{live_{\text{entry}}(init(S))} s_2$ then there exists s'_2 such that $\langle S, s_2 \rangle \Rightarrow^* s'_2$ and $s'_1 \sim_{live_{\text{exit}}(\ell)} s'_2$ for some $\ell \in final(S)$.

Proof [Not for Exam].

The proof is by induction on the length of the derivation sequences and uses Theorem 2. □