Program Analysis (70020)

Abstract Interpretation

Herbert Wiklicky

Department of Computing
Imperial College London

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

Autumn 2021

Live Variable Analysis

A variable is *live* at the exit from a label if there exists a path from the label to a use of the variable that does not re-define the variable. The *Live Variables Analysis* will determine:

> *For each program point, which variables* may *be live at the exit from the point.*

This analysis might be used as the basis for *Dead Code Elimination*. If the variable is not live at the exit from a label then, if the elementary block is an assignment to the variable, the elementary block can be eliminated.

Parity Analysis

A variable has *even* or *odd* parity at a label if we can guarntee that its value is *even* (e) or *odd* (o) for any execution of this label (not necessarily the same actual value). The *Parity Analysis* will determine:

> *For each program point, what is the parity of each variable.*

This analysis might be used as the basis for . . . (saving a bit?).

LV Analysis: Property Space

$$kill_{\mathrm{LV}} : \mathbf{Block}_\star \to \mathcal{P}(\mathbf{Var}_\star)$$

$$gen_{\mathrm{LV}} : \mathbf{Block}_\star \to \mathcal{P}(\mathbf{Var}_\star)$$

$$\mathrm{LV}_{entry} : \mathbf{Lab}_\star \to \mathcal{P}(\mathbf{Var}_\star)$$

$$\mathrm{LV}_{exit} : \mathbf{Lab}_\star \to \mathcal{P}(\mathbf{Var}_\star)$$

Important fact: Information we are interested in is in $\mathcal{P}(\mathbf{Var}_\star)$.

LV Equations and Transfer Functions

$$\mathrm{LV}_{exit}(\ell) = \begin{cases} \emptyset, \text{if } \ell \in \mathit{final}(S_\star) \\ \bigcup\{\mathrm{LV}_{entry}(\ell') \mid (\ell', \ell) \in \mathit{flow}^R(S_\star)\}, \text{otherwise} \end{cases}$$

$$\mathrm{LV}_{entry}(\ell) = (\mathrm{LV}_{exit}(\ell) \backslash \mathit{kill}_{\mathrm{LV}}([B]^\ell) \cup \mathit{gen}_{\mathrm{LV}}([B]^\ell) \quad \text{where } [B]^\ell \in \mathit{blocks}(S_\star)$$

with

$$\begin{aligned} \mathit{kill}_{\mathrm{LV}}([\, x := a\,]^\ell) &= \{x\} \\ \mathit{kill}_{\mathrm{LV}}([\, \textbf{skip}\,]^\ell) &= \emptyset \\ \mathit{kill}_{\mathrm{LV}}([b]^\ell) &= \emptyset \end{aligned}$$

$$\begin{aligned} \mathit{gen}_{\mathrm{LV}}([\, x := a\,]^\ell) &= FV(a) \\ \mathit{gen}_{\mathrm{LV}}([\, \textbf{skip}\,]^\ell) &= \emptyset \\ \mathit{gen}_{\mathrm{LV}}([b]^\ell) &= FV(b) \end{aligned}$$

Parity Information

The *LV* Analysis associates to lables information – concretely the set of live variables, i.e. a set in $\mathcal{P}(\textbf{Var}_\star)$. This is modified by local *transfer functions* and *collected* globaly according to *flow*.

For Parity we have identify the abstract properties to work with.

- Sets in $\mathcal{P}(\textbf{Var}_\star \times \{e, o\})$ or maybe $\mathcal{P}(\textbf{Var}_\star \times \{e, o, ?\})$, e.g. $\{(x, e), (x, o), (y, e)\} \equiv \{(x, ?), (y, e)\}$.
- Functions in $\textbf{Var}_\star \to \{e, o\}$ or better $\textbf{Var}_\star \to \{e, o, ?\}$. e.g. $\{x \mapsto ?, y \mapsto e\}$.
- represented as value tables, e.g. $\{x \mapsto ?, y \mapsto e\} = \begin{array}{|c|c|} \hline \mathrm{x} & \mathrm{y} \\ \hline ? & e \\ \hline \end{array}$

Questions: How to modify parity information locally and how to combine it, e.g. maybe $\{(x, e), (x, o), (y, e)\} \cup \{(x, e), (y, e)\}$.

Simplification, Abstraction, Approximation

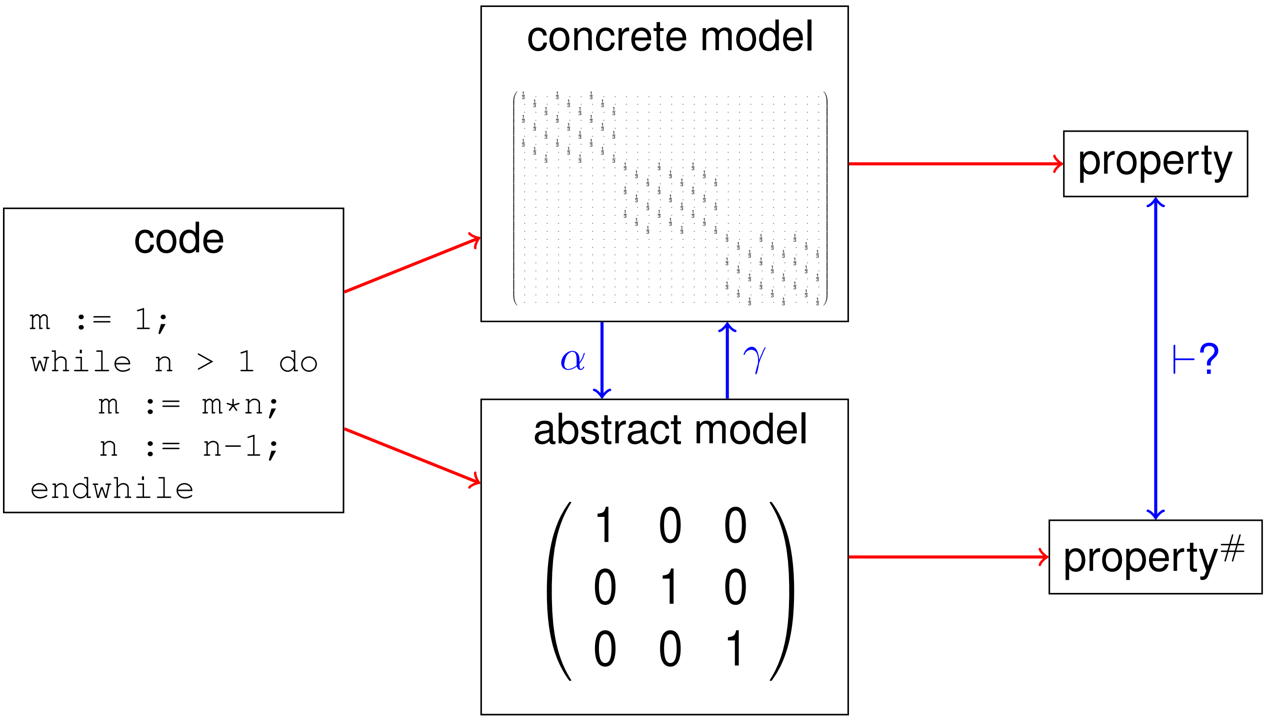
Designing a Program Analysis needs to establish correctness.

Doing this for each program property, cf. Live Variable, might be cumbersome, so we are looking for a general way to construct correct and efficient frameworks; more or less automatically.

From the 1970s the work of Cousot and Cousot on Abstract Interpretation provides a tools to do this. They demonstrated that numerous analysises can be obtained this way.

The central element is the simplification of the concrete semantics in order to obtain an abstract one as an optimal approximation.

Concrete Semantics vs Abstract Semantics



Cast-out-of-Nines

Plausibility check for arithmetic calculations, for example:

$$123 \times 457 + 76543 =?= 123654$$

Perform operations $n \bmod 9$ (enough to consider digits' sum)

$$6 \times 7 + 7 = 42 + 7 = 6 + 7 = 4 \neq 3$$

This is holds because elementary facts like:

$$\begin{aligned}
(a \pm b) \bmod 9 &= (a \bmod 9 \pm b \bmod 9) \bmod 9 \\
(a \times b) \bmod 9 &= (a \bmod 9 \times b \bmod 9) \bmod 9 \\
(10 \times a \pm b) \bmod 9 &= (a \pm b) \bmod 9
\end{aligned}$$

Note that there are false positives, cf also [1] and [2].

Approximation and Correctness

Data-flow analyses can be re-formulated in a different scenario where correctness is guaranteed by construction.

Classically, the theory of Abstract Interpretation allows us to

- construct simplified a (computable) abstract semantics
- construct approximate solutions
- obtain the correctness of the approximate solutions

Abstract Interpretation also uses other techniques, like widening/narrowing, which we will not cover here.

Notions of Approximation

Assume that we have a “solution” s to a problem.
What counts as a (good) approximation s^* to s?

In order theoretic structures we are looking for
Safe Approximations

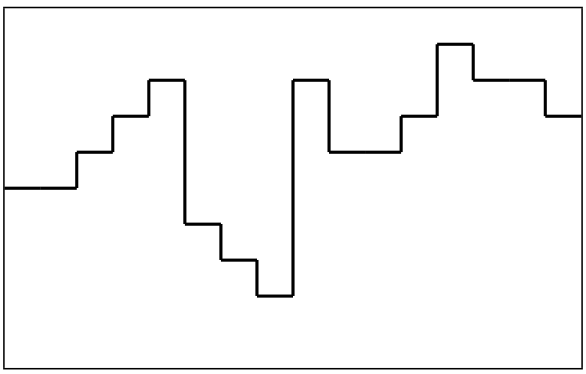
$$s^* \sqsubseteq s \quad \text{or} \quad s \sqsubseteq s^*$$

In quantitative, vector space structures we want
Close Approximations

$$\|s - s^*\| = \min_x \|s - x\|$$

Example: Function Approximation

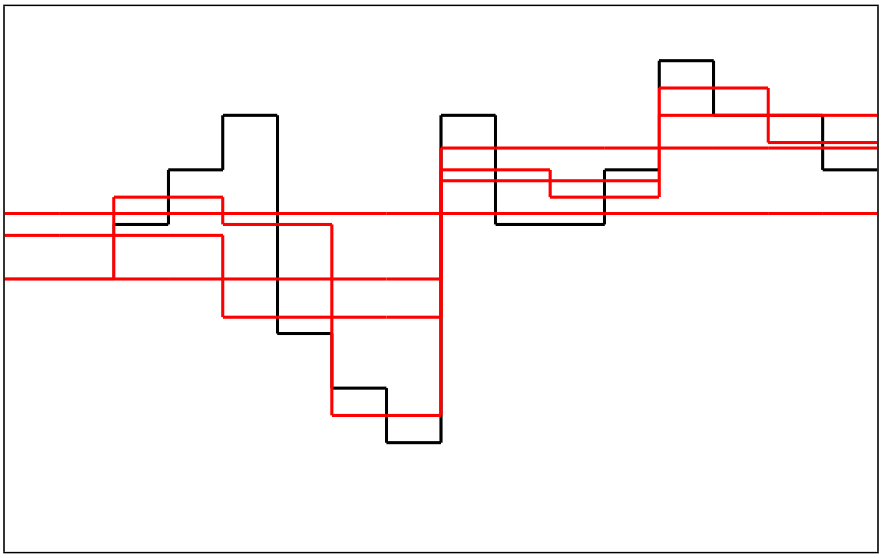
Concrete and abstract domain are step-functions on $[a, b]$.
The set of (real-valued) step-function $\mathcal{T}_n$ is based on the sub-division of the interval into n sub-intervals.



The concrete function needs n data points, its abstraction or approximation should need less, i.e. from $\mathbb{R}^n$ to $\mathbb{R}^m$ with $m < m$.

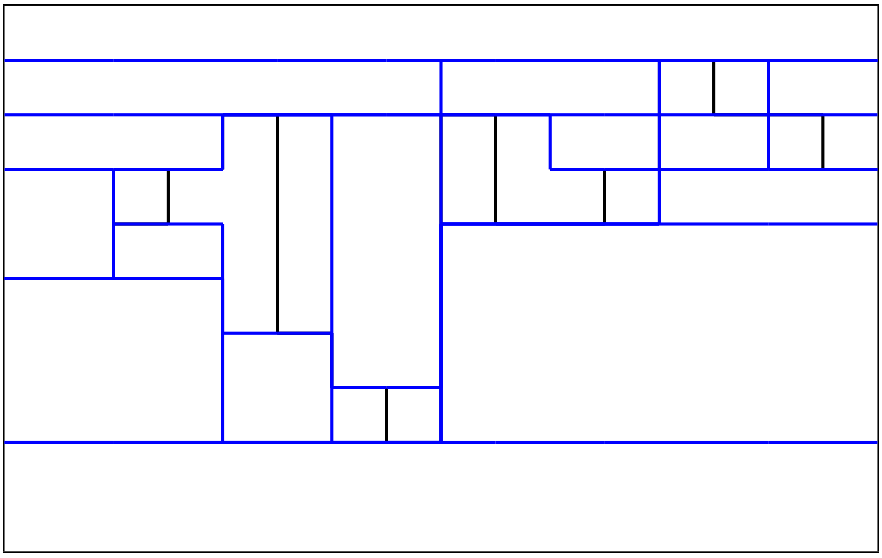
Close Approximations

Approximate $f \in \mathbb{R}^{16}$ by “least square” simplifications in $\mathbb{R}^8$, in $\mathbb{R}^4$, in $\mathbb{R}^2$ or even in $\mathbb{R}$.



Safe Approximations

Approximate $f \in \mathbb{R}^{16}$ by over/under approximation in $\mathbb{R}^8$, in $\mathbb{R}^4$, in $\mathbb{R}^2$ or even in $\mathbb{R}$.



Abstract Interpretation

In Program Analysis (cf. Monotone Frameworks) our property spaces are (complete) lattice.

Aim: Find abstract descriptions on which computations are easier; then relate the concrete and abstract solutions.

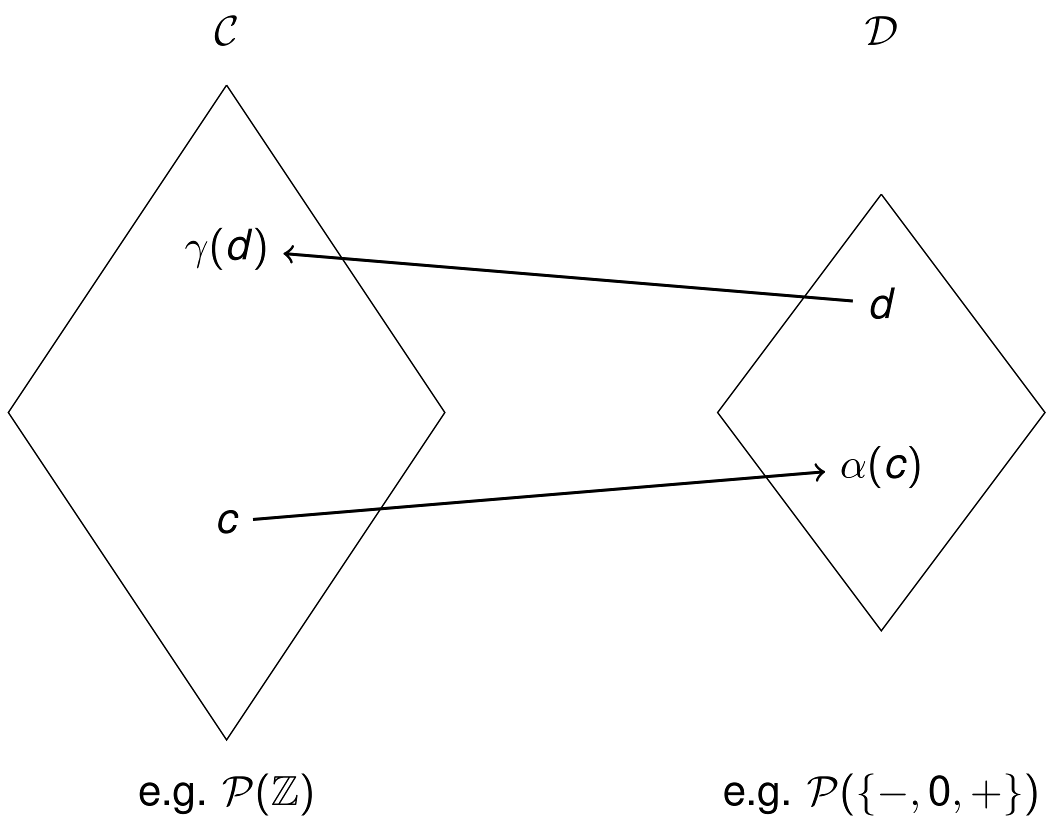
Definition

Let $\mathcal{C} = (\mathcal{C}, \leq_{\mathcal{C}})$ and $\mathcal{D} = (\mathcal{D}, \leq_{\mathcal{D}})$ be two partially ordered sets. If there are two functions $\alpha : \mathcal{C} \to \mathcal{D}$ and $\gamma : \mathcal{D} \to \mathcal{C}$ such that for all $c \in \mathcal{C}$ and all $d \in \mathcal{D}$:

$$c \leq_{\mathcal{C}} \gamma(d) \text{ iff } \alpha(c) \leq_{\mathcal{D}} d,$$

then $(\mathcal{C}, \alpha, \gamma, \mathcal{D})$ form a Galois connection.

Relating Concrete and Abstract Properties



Galois Connections

Definition

Let $\mathcal{C} = (\mathcal{C}, \leq_{\mathcal{C}})$ and $\mathcal{D} = (\mathcal{D}, \leq_{\mathcal{D}})$ be two partially ordered sets with two order-preserving functions $\alpha : \mathcal{C} \mapsto \mathcal{D}$ and $\gamma : \mathcal{D} \mapsto \mathcal{C}$. Then $(\mathcal{C}, \alpha, \gamma, \mathcal{D})$ form a Galois connection iff

(i) $\alpha \circ \gamma$ is reductive i.e. $\forall d \in \mathcal{D}$, $\alpha \circ \gamma(d) \leq_{\mathcal{D}} d$,

(ii) $\gamma \circ \alpha$ is extensive i.e. $\forall c \in \mathcal{C}$, $c \leq_{\mathcal{C}} \gamma \circ \alpha(c)$.

Proposition

Let $(\mathcal{C}, \alpha, \gamma, \mathcal{D})$ be a Galois connection. Then α and γ are quasi-inverse, i.e.

$$\text{(i) } \alpha \circ \gamma \circ \alpha = \alpha \quad \text{and} \quad \text{(ii) } \gamma \circ \alpha \circ \gamma = \gamma$$

Uniqueness and Duality

Given an abstraction α there is a unique concretisation γ.

Proposition

Let $(\mathcal{C}, \alpha, \gamma, \mathcal{D})$ be a Galois connection, then

(i) α uniquely determines γ by

$$\gamma(d) = \bigsqcup\{c \mid \alpha(c) \leq_{\mathcal{D}} d\},$$

and γ uniquely determines α via

$$\alpha(c) = \bigsqcap\{d \mid c \leq_{\mathcal{C}} \gamma(d)\}.$$

(ii) α is completely additive and γ is completely multiplicative, and $\alpha(\perp) = \perp$ and $\gamma(\top) = \top$.

For a proof see e.g. [3] Lemma 4.22.

Correctness and Optimality

Proposition

Given $\alpha : \mathcal{P}(\mathbb{Z}) \to \mathcal{D}$ and $\gamma : \mathcal{D} \to \mathcal{P}(\mathbb{Z})$ a Galois connection with $\mathcal{D}$ some property lattice. Consider an operation $op : \mathbb{Z} \to \mathbb{Z}$ on $\mathbb{Z}$ which is lifted to $\widehat{op} : \mathcal{P}(\mathbb{Z}) \to \mathcal{P}(\mathbb{Z})$ via

$$\widehat{op}(X) = \{op(x) \mid x \in X\},$$

then $op^{\#} : \mathcal{D} \to \mathcal{D}$ defined as $op^{\#} = \alpha \circ \widehat{op} \circ \gamma$ is the most precise function on $\mathcal{D}$ satisfying for all $Z \subseteq \mathbb{Z}$:

$$\alpha(\widehat{op}(Z)) \sqsubseteq op^{\#}(\alpha(Z))$$

It is enough to consider so-called Galois Insertions.
See [1] Lemma 2.3.2.

General Construction

The general construction of correct (and optimal) abstractions $f^{\#}$ of concrete function f is as follows:

$$\begin{tikzcd}
\mathcal{A} \arrow[r, shift left, "\alpha"] \arrow[d, "f"'] & \mathcal{A}^{\#} \arrow[l, shift left, "\gamma"] \arrow[d, "f^{\#}"] \\
\mathcal{B} \arrow[r, shift left, "\alpha'"] & \mathcal{B}^{\#} \arrow[l, shift left, "\gamma'"]
\end{tikzcd}$$

Correct approximation:

$$\alpha' \circ f \leq_{\#} f^{\#} \circ \alpha.$$

Induced semantics:

$$f^{\#} = \alpha' \circ f \circ \gamma.$$

Abstract Multiplication

How can we justify or obtain correct abstract versions of various operations, e.g. multiplication?

$\times^{\#}$	$\bot$	**even**	**odd**	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
even	$\bot$	**even**	**even**	**even**
odd	$\bot$	**even**	**odd**	$\top$
$\top$	$\bot$	**even**	$\top$	$\top$

Abstract Interpretation – introduced by Patrick Cousot and Radhia Cousot in 1977 – allows to “compute” abstractions which are correct by construction.

Parity (again)

Consider concrete $\mathcal{C} = \mathcal{P}(\mathbb{Z})$ and abstract $\mathcal{D} = \mathcal{P}(\{\textbf{even}, \textbf{odd}\})$.

The abstraction $\alpha : \mathcal{C} \to \mathcal{D}$ is given by for $X \subseteq \mathbb{Z}$:

$$\begin{aligned}
\alpha(\emptyset) &= \bot = \emptyset \\
\alpha(X) &= \textbf{even} \text{ iff } \forall x \in X\ \exists k : x = 2k \\
\alpha(X) &= \textbf{odd} \text{ iff } \forall x \in X\ \exists k : x = 2k + 1 \\
\alpha(X) &= \top = \{\textbf{even}, \textbf{odd}\} \text{ otherwise}
\end{aligned}$$

The concretisation $\gamma : \mathcal{D} \to \mathcal{C}$ then needs to be:

$$\begin{aligned}
\gamma(\bot) &= \emptyset \\
\gamma(\textbf{even}) &= \{x \in \mathbb{Z} \mid \exists k : x = 2k\} = E \\
\gamma(\textbf{odd}) &= \{x \in \mathbb{Z} \mid \exists k : x = 2k + 1\} = O \\
\gamma(\top) &= \top = \mathbb{Z} \text{ otherwise}
\end{aligned}$$

Parity: From $\times$ to $\times^{\#}$

To construct $\times^{\#}$ using α and γ we need to lift $. \times . : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ to $.\widehat{\times}. : \mathcal{P}(\mathbb{Z}) \times \mathcal{P}(\mathbb{Z}) \to \mathcal{P}(\mathbb{Z})$. Obviosly, for $X = \{x\} \subseteq \mathbb{Z}$ and $Y = \{y\} \subseteq \mathbb{Z}$:

$$X \widehat{\times} Y = \{x \times y \mid x \in X \text{ and } y \in Y\}$$

Defining the abstract multiplication $\times^{\#} = \alpha \circ (.\widehat{\times}.) \circ (\gamma, \gamma)$:

- $\gamma(\textbf{even}) = E$, then $E \widehat{\times} E = E' \subset E$, and $\alpha(E') = \textbf{even}$
- $\gamma(\textbf{odd}) = O$, then $E \widehat{\times} O = E$ and $\alpha(E) = \textbf{even}$
- etc.

Therefore, **even** $\times^{\#}$ **even** $=$ **even**, **even** $\times^{\#}$ **odd** $=$ **even**, etc.

Concrete Semantics $\to$ and Abstract Semantics $\rightsquigarrow$

Imagine some programming language, e.g. WHILE. Its concrete semantics identifies values in $\mathcal{V}$ (e.g. states) and specifies how a program S transforms v_1 into v_2; we may write this as

$$S \vdash v_1 \to v_2$$

A program analysis or abstract semantics identifies the set $\mathcal{L}$ of properties and how a program S transforms l_1 in to l_2

$$S \vdash l_1 \rightsquigarrow l_2$$

Unlike for general semantics, it is customary to require $\rightsquigarrow$ to be deterministic and thus define a function; this allows us to write:

$$f_S(l_1) = l_2 \text{ to mean } S \vdash l_1 \rightsquigarrow l_2.$$

Situation in While

We have SOS transitions $\langle S, s\rangle \Rightarrow \langle S', s'\rangle$ with S and S' programs and $s, s' \in \textbf{State} = (\textbf{Var} \to \textbf{Z})$, e.g.

$$\langle z := 2 \times z, [z \mapsto 2]\rangle \Rightarrow [z \mapsto 4]$$

translates to just an evaluation of the state:

$$z := 2 \times z \vdash [z \mapsto 2] \to [z \mapsto 4]$$

The fact that this also holds for the (abstract) parity means:

$$z := 2 \times z \vdash \textbf{even}(z) \rightsquigarrow \textbf{even}(z)$$

and also $z := 2 \times z \vdash \textbf{odd}(z) \rightsquigarrow \textbf{even}(z)$.

Correctness Relation

Every program analysis should be correct with respect to the semantics.

For a class of (so-called first-order) program analyses this is established by directly relating properties to values using a correctness relation:

$$\rhd : \mathcal{V} \times \mathcal{L} \to \{\textbf{tt}, \textbf{ff}\} \quad \text{or} \quad \rhd \subseteq \mathcal{V} \times \mathcal{L}$$

The intention is that "$v \ \rhd \ l$" formalises our claim that the value v is described by the property l (or v abstracts to l).

Preservation of Correctness

One has to prove that $\triangleright$ is preserved under computation. This may be formulated as the implication:

$$\begin{array}{rcl} v_1 \triangleright l_1 & \wedge & \\ S \vdash v_1 \rightarrow v_2 & \wedge & S \vdash l_1 \rightsquigarrow l_2 \\ & \Rightarrow & v_2 \triangleright l_2 \end{array}$$

This property is also expressed by the following diagram:

$$\begin{array}{ccccc} S & \vdash & v_1 & \rightarrow & v_2 \\ & & \vdots & & \vdots \\ & & \triangleright & \Rightarrow & \triangleright \\ & & \vdots & & \vdots \\ S & \vdash & l_1 & \rightsquigarrow & l_2 \end{array}$$

Correctness of Parity

$$\begin{array}{ccc|ccc} 0 & \triangleright & \textbf{even} & 1 & \triangleright & \textbf{odd} \\ 2 & \triangleright & \textbf{even} & 3 & \triangleright & \textbf{odd} \\ 4 & \triangleright & \textbf{even} & 5 & \triangleright & \textbf{odd} \\ & \dots & & & \dots & \end{array}$$

$$\begin{array}{c|c} z := 2 \times z \vdash [z \mapsto 1] \rightarrow [z \mapsto 2] & \textbf{odd}(z) \rightsquigarrow \textbf{even}(z) \\ z := 2 \times z \vdash [z \mapsto 2] \rightarrow [z \mapsto 4] & \textbf{even}(z) \rightsquigarrow \textbf{even}(z) \\ z := 2 \times z \vdash [z \mapsto 3] \rightarrow [z \mapsto 6] & \textbf{odd}(z) \rightsquigarrow \textbf{even}(z) \\ \dots & \dots \end{array}$$

$$\begin{array}{c} 1 \triangleright \textbf{odd} \ \wedge \ p \vdash 1 \rightarrow 2 \ \wedge \ p \vdash \textbf{odd} \rightsquigarrow \textbf{even} \Rightarrow 2 \triangleright \textbf{even} \\ 2 \triangleright \textbf{even} \ \wedge \ p \vdash 2 \rightarrow 4 \ \wedge \ p \vdash \textbf{even} \rightsquigarrow \textbf{even} \Rightarrow 4 \triangleright \textbf{even} \\ 3 \triangleright \textbf{odd} \ \wedge \ p \vdash 3 \rightarrow 6 \ \wedge \ p \vdash \textbf{odd} \rightsquigarrow \textbf{even} \Rightarrow 6 \triangleright \textbf{even} \\ \dots \end{array}$$

Thus it is correct: "$p \equiv z := 2 \times z$ always produces an **even** z".

Abstract Interpretation and Correctness

The theory of Abstract Interpretation comes to life when we augment the set of properties $\mathcal{L}$ with a preorder (better: lattice) structure and elate this to the correctness relation $\rhd$.

The most common scenario is when $\mathcal{L} = (\mathcal{L}, \sqsubseteq, \sqcup, \sqcap, \perp, \top)$ is a complete lattice with partial ordering $\sqsubseteq$.

We then impose the following relationship between $\rhd$ and $\mathcal{L}$:

$$v \rhd l_1 \ \wedge \ l_1 \sqsubseteq l_2 \quad \Rightarrow \quad v \rhd l_2 \tag{1}$$

$$\forall l \in \mathcal{L}' \subseteq \mathcal{L} : v \rhd l \quad \Rightarrow \quad v \rhd \bigsqcap \mathcal{L}' \tag{2}$$

Condition (1)

Consider the first of these conditions:

$$v \rhd l_1 \ \wedge \ l_1 \sqsubseteq l_2 \Rightarrow v \rhd l_2$$

- The condition says that the smaller the property is with respect to the partial order, the better (i.e. precise) it is.
- This is an "arbitrary" decision in the sense that we could instead have decided that the larger the property is, the better it is, as is indeed the case in much of the literature on Data Flow Analysis; luckily the principle of duality from lattice theory tells us that this difference is only cosmetic.

Condition (2)

Looking at the second condition describing correctness:

$$\forall l \in \mathcal{L}' \subseteq \mathcal{L} : v \rhd l \Rightarrow v \rhd \bigsqcap \mathcal{L}'$$

- The second condition says that there is always a best property for describing a value. This is important for having to perform only one analysis (using the best property, i.e. the greatest lower bound of the candidates) instead of several analyses (one for each of the candidates).
- The condition has two immediate consequences:

$$v \rhd \top$$

$$v \rhd l_1 \wedge v \rhd l_2 \Rightarrow v \rhd (l_1 \sqcap l_2)$$

Again: Parity Example

The abstract properties **even** and **odd** do themselves not form a lattice $\mathcal{L}$, but we can use – as usual: $\mathcal{L} = \mathcal{P}(\{\textbf{even}, \textbf{odd}\})$, where $\{\textbf{even}\}$ represents the definitive fact **even** and $\{\textbf{odd}\}$ the precise property **odd**; while the empty set $\bot = \emptyset$ represents an undefined parity and $\top = \{\textbf{even}, \textbf{odd}\}$ stands for any parity.

The conditions imposed on $\rhd$ and $\mathcal{L}$ mean in this case:

(1) Any parity is always a valid description, e.g.

$$2 \rhd \{\textbf{even}\} \wedge \{\textbf{even}\} \sqsubseteq \top \Rightarrow 2 \rhd \top$$

(2) The most precise parity is valid, e.g.

$$(2 \rhd \{\textbf{even}\} \wedge 2 \rhd \top) \Rightarrow 2 \rhd (\{\textbf{even}\} \sqcap \top)$$

$$\text{i.e. } (2 \rhd \{\textbf{even}\} \wedge 2 \rhd \top) \Rightarrow 2 \rhd \{\textbf{even}\}$$

Preservation of Correctness via Abstraction

We require that corectness is preserved:

$$v_1 \rhd l_1 \;\wedge S \vdash v_1 \to v_2 \wedge S \vdash l_1 \rightsquigarrow l_2 \Rightarrow v_2 \rhd l_2$$

With a (semantical transfer) function function f_S we have:

$$v_1 \rhd l_1 \;\wedge f_S(v_1) = v_2 \wedge f_S^{\#}(l_1) = l_2 \Rightarrow v_2 \rhd l_2$$

This property is also expressed by the following diagram:

$$\begin{tikzcd}
\mathcal{V} \arrow[rr, "f_S"] \arrow[d, shift right, "\alpha"'] & & \mathcal{V} \arrow[d, shift left, "\alpha"] \\
\mathcal{L} \arrow[rr, "f_S^{\#}"'] \arrow[u, shift right, "\gamma"'] & & \mathcal{L} \arrow[u, shift left, "\gamma"]
\end{tikzcd}$$

Representation and Extraction Functions

We can use a representation function $\beta : \mathcal{V} \to \mathcal{L}$ to induce a Galois connection $(\mathcal{P}(\mathcal{V}), \alpha, \gamma, \mathcal{L})$ via

$$\begin{aligned}
\alpha(V) &= \bigsqcup \{\beta(v) \mid v \in V\} \\
\gamma(l) &= \{v \in V \mid \beta(v) \sqsubseteq l\}
\end{aligned}$$

For $\mathcal{L} = \mathcal{P}(\mathcal{D})$ with $\mathcal{D}$ being some set of “abstract values” we can also use an extraction function, $\eta : \mathcal{V} \to \mathcal{D}$ defined as

$$\begin{aligned}
\alpha(V) &= \{\eta(v) \mid v \in V\} \\
\gamma(D) &= \{v \mid \eta(v) \in D\}
\end{aligned}$$

in order to construct a Galois connection.

Example: Parity

A repesentation function $\beta : \mathbf{Z} \to \mathcal{P}(\{\mathbf{even}, \mathbf{odd}\})$ is easily defined by:

$$\beta(n) = \begin{cases} \{\mathbf{even}\} & \text{if } \exists k \in \mathbf{Z} \text{ s.t. } n = 2k \\ \{\mathbf{odd}\} & \text{otherwise} \end{cases}$$

Correctness implies that the abstract properties are dominated by the actual ones, e.g. $\beta(4) = \{\mathbf{even}\} \sqsubseteq \top = \{\mathbf{even}, \mathbf{odd}\}$ is acceptable.

This means that we also could use as a representation function

$$\beta(n) = \top = \{\mathbf{even}, \mathbf{odd}\}$$

for all $n \in \mathbf{Z}$. Though this would be valid it would also be rather imprecise.

References Abstract Interpretation

[1] Neil D. Jones and Flemming Nielson: *Abstract Interpretation: A semantics-based tool for program analysis*. in: Handbook of Logic in Computer Science (Vol. 4), pp 527–636, Oxford University Press,1995.

[2] Patrick Cousot and Radhia Cousot: *Abstract Interpretation and application to logic programs*. The Journal of Logic Programming, Vol. 13, pp 103–179, 1992.

[3] Flemming Nielson, Hanne Riis Nielson and Chris Hankin: *Principles of Program Analysis*. Chapter 4, Springer Verlag, 1999/2005.

[4] Patrick Cousot: *Abstract Interpretation*. MIT Course, 2005. `http://web.mit.edu/16.399/www/`