Program Analysis (CO470/97128/97146)
Abstract Interpretation

Herbert Wiklicky

Department of Computing
Imperial College London

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

Autumn 2020
Simplification, Abstraction, Approximation

Designing a Program Analysis needs to establish correctness.
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.
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 analyses can be obtained this way.
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 tool to do this. They demonstrated that numerous analyses 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

code

```plaintext
m := 1;
while n > 1 do
    m := m*n;
    n := n-1;
endwhile
```

concrete model

property

abstract model

property
Concrete Semantics vs Abstract Semantics

code

m := 1;
while n > 1 do
    m := m*n;
    n := n-1;
endwhile

concrete model

abstract model

property

\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}
\]
Concrete Semantics vs Abstract Semantics

**Code**

\[ m := 1; \]
\[ \text{while } n > 1 \text{ do} \]
\[ \quad m := m \times n; \]
\[ \quad n := n - 1; \]
\[ \text{endwhile} \]

**Concrete Model**

\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{pmatrix}
\]

**Abstract Model**

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
\end{pmatrix}
\]

\[ \alpha \]

**Property**
Concrete Semantics vs Abstract Semantics

code
m := 1;
while n > 1 do
  m := m*n;
  n := n-1;
endwhile

concrete model

abstract model

property

$\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{pmatrix}$
Concrete Semantics vs Abstract Semantics

code

```
m := 1;
while n > 1 do
    m := m*n;
    n := n-1;
endwhile
```

concrete model

```
1 3 . .
3 1 . .
. . . .
```

property

abstract model

```
\[
\begin{pmatrix}
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}
\]
```

property #
Concrete Semantics vs Abstract Semantics

code

\begin{verbatim}
m := 1;
while n > 1 do
  m := m*n;
  n := n-1;
endwhile
\end{verbatim}

concrete model

abstract model

\begin{pmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1 \\
\end{pmatrix}

property

abstract model

property

\[ \alpha \]

\[ \gamma \]
Cast-out-of-Nines

Plausibility check for arithmetic calculations, for example:

\[123 \times 457 + 76543 =?= 123654\]
Cast-out-of-Nines

Plausibility check for arithmetic calculations, for example:

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

Perform operations $n \mod 9$ (enough to consider digits’ sum)

$$6 \times 7 + 7$$

Note that there are false positives, cf also [1] and [2].
Cast-out-of-Nines

Plausibility check for arithmetic calculations, for example:

\[123 \times 457 + 76543 =?= 123654\]

Perform operations \(n \mod 9\) (enough to consider digits’ sum)

\[6 \times 7 + 7 = 42 + 7\]
Cast-out-of-Nines

Plausibility check for arithmetic calculations, for example:

\[ 123 \times 457 + 76543 =?= 123654 \]

Perform operations \( n \mod 9 \) (enough to consider digits’ sum)

\[ 6 \times 7 + 7 = 42 + 7 = 6 + 7 \]

Note that there are false positives, cf also [1] and [2].
Cast-out-of-Nines

Plausibility check for arithmetic calculations, for example:

$$123 \times 457 + 76543 = \text{=} 123654$$

Perform operations $n \mod 9$ (enough to consider digits’ sum)

$$6 \times 7 + 7 = 42 + 7 = 6 + 7 = 4 \neq 3$$
Cast-out-of-Nines

Plausibility check for arithmetic calculations, for example:

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

Perform operations $n \mod 9$ (enough to consider digits’ sum)

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

This is holds because elementary facts like:

$$(a \pm b) \mod 9 = (a \mod 9 \pm b \mod 9) \mod 9$$
Cast-out-of-Nines

Plausibility check for arithmetic calculations, for example:

\[ 123 \times 457 + 76543 = ?= 123654 \]

Perform operations \( n \mod 9 \) (enough to consider digits’ sum)

\[ 6 \times 7 + 7 = 42 + 7 = 6 + 7 = 4 \neq 3 \]

This is holds because elementary facts like:

\[
(a \pm b) \mod 9 = (a \mod 9 \pm b \mod 9) \mod 9
\]

\[
(a \times b) \mod 9 = (a \mod 9 \times b \mod 9) \mod 9
\]
Cast-out-of-Nines

Plausibility check for arithmetic calculations, for example:

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

Perform operations $n \mod 9$ (enough to consider digits’ sum)

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

This is holds because elementary facts like:

$$(a \pm b) \mod 9 = (a \mod 9 \pm b \mod 9) \mod 9$$
$$(a \times b) \mod 9 = (a \mod 9 \times b \mod 9) \mod 9$$
$$(10 \times a \pm b) \mod 9 = (a \pm b) \mod 9$$
Cast-out-of-Nines

Plausibility check for arithmetic calculations, for example:

\[ 123 \times 457 + 76543 = ? = 123654 \]

Perform operations \( n \mod 9 \) (enough to consider digits’ sum)

\[ 6 \times 7 + 7 = 42 + 7 = 6 + 7 = 4 \neq 3 \]

This is holds because elementary facts like:

\[
(a \pm b) \mod 9 \quad = \quad (a \mod 9 \pm b \mod 9) \mod 9
\]
\[
(a \times b) \mod 9 \quad = \quad (a \mod 9 \times b \mod 9) \mod 9
\]
\[
(10 \times a \pm b) \mod 9 \quad = \quad (a \pm b) \mod 9
\]

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.
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...
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
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
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
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 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 $s$?
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

\[
\begin{align*}
\text{safe: } \quad & s^* \sqsubseteq s \quad \text{or} \quad s \sqsubseteq s^* \\
\end{align*}
\]
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 $[a, b]$. 
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.
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.
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
Close Approximations

Approximate $f \in \mathbb{R}^{16}$ by “least square” simplifications in $\mathbb{R}^{8}$
Close Approximations

Approximate \( f \in \mathbb{R}^{16} \) by “least square” simplifications in \( \mathbb{R}^8 \), in \( \mathbb{R}^4 \)
Close Approximations

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

Approximate $f \in \mathbb{R}^{16}$ by over/under approximation in $\mathbb{R}^8$
Safe Approximations

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

Approximate $f \in \mathbb{R}^{16}$ by over/under approximation in $\mathbb{R}^{8}$, in $\mathbb{R}^{4}$, in $\mathbb{R}^{2}$
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.
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.
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 \( \mathcal{C} = (\mathcal{C}, \leq_C) \) and \( \mathcal{D} = (\mathcal{D}, \leq_D) \) 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 \( d \in \mathcal{D} \):

\[
c \leq_C \gamma(d) \text{ iff } \alpha(c) \leq_D d,
\]

then \((\mathcal{C}, \alpha, \gamma, \mathcal{D})\) form a Galois connection.
Relating Concrete and Abstract Properties

\[ c \]

\[ \alpha(c) \]

\[ \gamma(d) \]
Relating Concrete and Abstract Properties

$c$ e.g. $P(\{0\})$

d e.g. $P(\{-, 0, +\})$
Relating Concrete and Abstract Properties

\[ C \]

\[ \mathcal{P}(\mathbb{Z}) \]

e.g. \( \mathcal{P}(\mathbb{Z}) \)

\[ D \]

\[ \mathcal{P}(\{-, 0, +\}) \]

e.g. \( \mathcal{P}(\{-, 0, +\}) \)
Relating Concrete and Abstract Properties

\[ C \]

\[ D \]

e.g. \( \mathcal{P}(\mathbb{Z}) \)

e.g. \( \mathcal{P}(\{-, 0, +\}) \)
Relating Concrete and Abstract Properties

e.g. $\mathcal{P}(\mathbb{Z})$

e.g. $\mathcal{P}(-, 0, +)$

$c \xrightarrow{\alpha} d$

$C \quad D$

$\alpha(c)$
Relating Concrete and Abstract Properties

$C\xrightarrow{\alpha(c)} D$

- $C$: e.g. $\mathcal{P}(\mathbb{Z})$
- $D$: e.g. $\mathcal{P}(\{-, 0, +\})$
Relating Concrete and Abstract Properties

\[ \gamma(d') \quad \alpha(c) \]

\[ C \quad D \]

\[ c = \alpha(c) \]

\[ d = \gamma(d') \]

\[ \text{e.g. } \mathcal{P}(\mathbb{Z}) \quad \text{e.g. } \mathcal{P}(\{-, 0, +\}) \]
Galois Connections

Definition
Let \( C = (\mathcal{C}, \leq_{\mathcal{C}}) \) and \( D = (\mathcal{D}, \leq_{\mathcal{D}}) \) be two partially ordered sets with two order-preserving functions \( \alpha : \mathcal{C} \rightarrow \mathcal{D} \) and \( \gamma : \mathcal{D} \rightarrow \mathcal{C} \). 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. \( \forall c \in \mathcal{C}, c \leq_{\mathcal{C}} \gamma \circ \alpha(c) \).
Galois Connections

Definition
Let \( C = (C, \leq_C) \) and \( D = (D, \leq_D) \) be two partially ordered sets with two order-preserving functions \( \alpha : C \rightarrow D \) and \( \gamma : D \rightarrow C \). Then \((C, \alpha, \gamma, D)\) form a Galois connection iff

(i) \( \alpha \circ \gamma \) is reductive i.e. \( \forall d \in D, \alpha \circ \gamma(d) \leq_D d \),
(ii) \( \gamma \circ \alpha \) is extensive i.e. \( \forall c \in C, c \leq_C \gamma \circ \alpha(c) \).

Proposition
Let \((C, \alpha, \gamma, D)\) be a Galois connection. Then \( \alpha \) and \( \gamma \) are quasi-inverse, i.e.

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

Given an abstraction $\alpha$ there is a unique concretisation $\gamma$. 
Uniqueness and Duality

Given an abstraction $\alpha$ there is a unique concretisation $\gamma$.

Proposition

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

Given an abstraction $\alpha$ there is a unique concretisation $\gamma$.

Proposition

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

(i) $\alpha$ uniquely determines $\gamma$ by

$$\gamma(d) = \bigsqcup \{ c \mid \alpha(c) \leq_D d \},$$

and $\gamma$ uniquely determines $\alpha$ via

$$\alpha(c) = \bigsqcap \{ d \mid c \leq_C \gamma(d) \}.$$
Uniqueness and Duality

Given an abstraction \( \alpha \) there is a unique concretisation \( \gamma \).

Proposition

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

(i) \( \alpha \) uniquely determines \( \gamma \) by

\[
\gamma(d) = \bigsqcup \{ c \mid \alpha(c) \leq_D d \},
\]

and \( \gamma \) uniquely determines \( \alpha \) via

\[
\alpha(c) = \bigcap \{ d \mid c \leq_C \gamma(d) \}.
\]

(ii) \( \alpha \) is completely additive and \( \gamma \) is completely multiplicative, and \( \alpha(\bot) = \bot \) and \( \gamma(\top) = \top \).
Uniqueness and Duality

Given an abstraction $\alpha$ there is a unique concretisation $\gamma$.

Proposition

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

(i) $\alpha$ uniquely determines $\gamma$ by

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

and $\gamma$ uniquely determines $\alpha$ via

$$\alpha(c) = \bigcap \{ d \mid c \leq \mathcal{C} \gamma(d) \}.$$

(ii) $\alpha$ is completely additive and $\gamma$ is completely multiplicative, and $\alpha(\bot) = \bot$ 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 $\text{op} : \mathbb{Z} \to \mathbb{Z}$ on $\mathbb{Z}$ which is lifted to $\hat{\text{op}} : \mathcal{P}(\mathbb{Z}) \to \mathcal{P}(\mathbb{Z})$ via

$$\hat{\text{op}}(Z) = \{ \text{op}(x) \mid x \in X \},$$

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

$$\alpha(\hat{\text{op}}(Z)) \subseteq \text{op}^\#(\alpha(Z))$$
Proposition

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

\[
\widehat{\text{op}}(Z) = \{ \text{op}(x) \mid x \in X \},
\]

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

\[
\alpha(\widehat{\text{op}}(Z)) \subseteq \text{op}^\#(\alpha(Z))
\]

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:
General Construction

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

$$
\begin{array}{c}
\mathcal{A} & \xrightarrow{\alpha} & \mathcal{A}^# \\
\downarrow{f} & \downarrow{\gamma} & \downarrow{f^#} \\
\mathcal{B} & \xrightarrow{\alpha'} & \mathcal{B}^#
\end{array}
$$

Correct approximation:

$$
\alpha' \circ f \leq f^# \circ \alpha.
$$

Induced semantics:

$$
f^# = \alpha' \circ f \circ \gamma.
$$
General Construction

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

Correct approximation:

$$\alpha' \circ f \leq f\# \circ \alpha.$$
General Construction

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

$$
\begin{array}{c}
A & \xrightarrow{\alpha} & A^# \\
\downarrow{f} & \downarrow{\gamma} & \downarrow{f^#} \\
B & \xleftarrow{\alpha'} & B^#
\end{array}
$$

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?
Abstract Multiplication

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

<table>
<thead>
<tr>
<th>× #</th>
<th>⊥</th>
<th>even</th>
<th>odd</th>
<th>⊤</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
<td>⊥</td>
</tr>
<tr>
<td>even</td>
<td>⊥</td>
<td>even</td>
<td>even</td>
<td>even</td>
</tr>
<tr>
<td>odd</td>
<td>⊥</td>
<td>even</td>
<td>odd</td>
<td>⊤</td>
</tr>
<tr>
<td>⊤</td>
<td>⊥</td>
<td>even</td>
<td>⊤</td>
<td>⊤</td>
</tr>
</tbody>
</table>
Abstract Multiplication

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

<table>
<thead>
<tr>
<th>(\times)</th>
<th>(\bot)</th>
<th>even</th>
<th>odd</th>
<th>(\top)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
<td>(\bot)</td>
</tr>
<tr>
<td>even</td>
<td>(\bot)</td>
<td>even</td>
<td>even</td>
<td>even</td>
</tr>
<tr>
<td>odd</td>
<td>(\bot)</td>
<td>even</td>
<td>odd</td>
<td>(\top)</td>
</tr>
<tr>
<td>(\top)</td>
<td>(\bot)</td>
<td>even</td>
<td>(\top)</td>
<td>(\top)</td>
</tr>
</tbody>
</table>

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

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

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

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

$\alpha(\emptyset) = \bot = \emptyset$

$\alpha(X) = \text{even} \iff \forall x \in X \exists k : x = 2k$

$\alpha(X) = \text{odd} \iff \forall x \in X \exists k : x = 2k + 1$

$\alpha(X) = \top = \{\text{even, odd}\}$ otherwise
Parity (again)

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

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

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

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

- $\gamma(\bot) = \emptyset$
- $\gamma(\text{even}) = \{x \in \mathbb{Z} | \exists k : x = 2k\} = E$
- $\gamma(\text{odd}) = \{x \in \mathbb{Z} | \exists k : x = 2k + 1\} = O$
- $\gamma(\top) = \top = \mathbb{Z}$ otherwise
To construct $\times^#$ using $\alpha$ and $\gamma$ we need to lift $\times : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ to $\hat{\times} : \mathcal{P}(\mathbb{Z}) \times \mathcal{P}(\mathbb{Z}) \to \mathcal{P}(\mathbb{Z})$. 

Obviosly, for $X = \{x\} \subseteq \mathbb{Z}$ and $Y = \{y\} \subseteq \mathbb{Z}$:

$$X \hat{\times} Y = \{x \times y | x \in X \text{ and } y \in Y\}$$

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

$\gamma(\text{even}) = E$,

then $E \hat{\times} E = E' \subset E$, and 

$\alpha(\text{E'}) = \text{even}$,

$\gamma(\text{odd}) = O$,

then $E \hat{\times} O = E$ and $\alpha(E) = \text{even}$,

etc.

Therefore, even $\times^#$ even = even,

even $\times^#$ odd = even,

even $\times^#$ even = even,

etc.
Parity: From $\times$ to $\times^\#$

To construct $\times^\#$ using $\alpha$ and $\gamma$ we need to lift $\times : \mathbb{Z} \times \mathbb{Z} \rightarrow \mathbb{Z}$ to $\hat{\times} : \mathcal{P}(\mathbb{Z}) \times \mathcal{P}(\mathbb{Z}) \rightarrow \mathcal{P}(\mathbb{Z})$. Obviously, for $X = \{x\} \subseteq \mathbb{Z}$ and $Y = \{y\} \subseteq \mathbb{Z}$:

$$X \hat{\times} Y = \{x \times y \mid x \in X \text{ and } y \in Y\}$$
Parity: From $\times$ to $\times^#$

To construct $\times^#$ using $\alpha$ and $\gamma$ we need to lift $\times : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ to $\hat{\times} : \mathcal{P}(\mathbb{Z}) \times \mathcal{P}(\mathbb{Z}) \to \mathcal{P}(\mathbb{Z})$. Obviously, for $X = \{ x \} \subseteq \mathbb{Z}$ and $Y = \{ y \} \subseteq \mathbb{Z}$:

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

Defining the abstract multiplication $\times^# = \alpha \circ (\hat{\times}) \circ (\gamma, \gamma)$:
Parity: From $\times$ to $\times^\#$

To construct $\times^\#$ using $\alpha$ and $\gamma$ we need to lift $\times : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ to $\hat{\times} : \mathcal{P} (\mathbb{Z}) \times \mathcal{P} (\mathbb{Z}) \to \mathcal{P} (\mathbb{Z})$.

Obviosly, for $X = \{ x \} \subseteq \mathbb{Z}$ and $Y = \{ y \} \subseteq \mathbb{Z}$:

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

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

$\gamma ($even$) = E$, then $E \hat{\times} E = E' \subseteq E$, and $\alpha (E') =$even
Parity: From $\times$ to $\times^\#$

To construct $\times^\#$ using $\alpha$ and $\gamma$ we need to lift $\times: \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ to $\hat{\times}: \mathcal{P}(\mathbb{Z}) \times \mathcal{P}(\mathbb{Z}) \to \mathcal{P}(\mathbb{Z})$. Obviously, for $X = \{x\} \subseteq \mathbb{Z}$ and $Y = \{y\} \subseteq \mathbb{Z}$:

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

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

- $\gamma(\text{even}) = E$, then $E \hat{\times} E = E' \subset E$, and $\alpha(E') = \text{even}$
- $\gamma(\text{odd}) = O$, then $E \hat{\times} O = E$ and $\alpha(E) = \text{even}$
Parity: From $\times$ to $\times^\#$

To construct $\times^\#$ using $\alpha$ and $\gamma$ we need to lift $\times : \mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ to $\hat{\times} : \mathcal{P}(\mathbb{Z}) \times \mathcal{P}(\mathbb{Z}) \to \mathcal{P}(\mathbb{Z})$. Obviously, for $X = \{x\} \subseteq \mathbb{Z}$ and $Y = \{y\} \subseteq \mathbb{Z}$:

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

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

- $\gamma(\text{even}) = E$, then $E \hat{\times} E = E' \subset E$, and $\alpha(E') = \text{even}$
- $\gamma(\text{odd}) = O$, then $E \hat{\times} O = E$ and $\alpha(E) = \text{even}$
- etc.
Parity: From $\times$ to $\times\#$

To construct $\times\#$ using $\alpha$ and $\gamma$ we need to lift $\times:\mathbb{Z} \times \mathbb{Z} \to \mathbb{Z}$ to $\hat{\times}:\mathcal{P}^{}\mathbb{Z} \times \mathcal{P}^{}\mathbb{Z} \to \mathcal{P}^{}\mathbb{Z}$. Obviously, for $X = \{x\} \subseteq \mathbb{Z}$ and $Y = \{y\} \subseteq \mathbb{Z}$:

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

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

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

Therefore, $\text{even } \times\# \text{ even } = \text{even}$, $\text{even } \times\# \text{ odd } = \text{even}$, etc.
Concrete Semantics $\rightarrow$ and Abstract Semantics $\leadsto$

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 $v_2$;
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

$$S \vdash v_1 \rightarrow v_2$$
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

$$S \vdash v_1 \rightarrow v_2$$

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

$$S \vdash l_1 \leadsto l_2$$
Concrete Semantics $\rightarrow$ 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

$$S \vdash v_1 \rightarrow v_2$$

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

$$S \vdash l_1 \rightsquigarrow l_2$$

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

$$f_S(l_1) = l_2$$ 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 \( s, s' \in \text{State} = (\text{Var} \rightarrow \mathbb{Z}) \), e.g.
Situation in While

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

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

translates to just an evaluation of the state:

\[
z := 2 \times z \quad [z \mapsto 2] \rightarrow [z \mapsto 4]
\]
Situation in While

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

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

translates to just an evaluation of the state:

\[
z := 2 \times z \vdash [z \mapsto 2] \rightarrow [z \mapsto 4]
\]

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

\[
z := 2 \times z \vdash \text{even}(z) \rightsquigarrow \text{even}(z)
\]

and also \( z := 2 \times z \vdash \text{odd}(z) \rightsquigarrow \text{even}(z) \).
Correctness Relation

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

\[ \triangleright : \mathcal{V} \times \mathcal{L} \rightarrow \{\text{tt, ff}\} \quad \text{or} \quad \triangleright \subseteq \mathcal{V} \times \mathcal{L} \]
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:

\[ \triangleright : \mathcal{V} \times \mathcal{L} \rightarrow \{ \text{tt}, \text{ff} \} \text{ or } \triangleright \subseteq \mathcal{V} \times \mathcal{L} \]

The intention is that “\( \nu \triangleright l \)” formalises our claim that the value \( \nu \) is described by the property \( l \) (or \( \nu \) abstracts to \( l \)).
Preservation of Correctness

One has to prove that ▷ is preserved under computation.
Preservation of Correctness

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

\[ v_1 \rightarrow l_1 \land \]

This property is also expressed by the following diagram:

\[ S \vdash v_1 \rightarrow v_2 \]

\[ \vdash \rightarrow \]

\[ S \vdash l_1 \Rightarrow l_2 \]

\[ \rightarrow \]

\[ v_2 \rightarrow l_2 \]

\[ \vdash \rightarrow \]

\[ 22 / 31 \]
Preservation of Correctness

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

$$v_1 \triangleright I_1 \land S \vdash v_1 \rightarrow v_2 \land S \vdash I_1 \leadsto I_2$$
Preservation of Correctness

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

$$v_1 \triangleright l_1 \land S \vdash v_1 \rightarrow v_2 \land S \vdash l_1 \rightsquigarrow l_2 \Rightarrow v_2 \triangleright l_2$$
Preservation of Correctness

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

$$
\begin{align*}
\forall v_1 & \quad \triangleright \quad l_1 \\
S \vdash v_1 & \rightarrow v_2 \\
S \vdash l_1 & \rightsquigarrow l_2 \\
\Rightarrow & \\
\forall v_2 & \quad \triangleright \quad l_2
\end{align*}
$$

This property is also expressed by the following diagram:

$$
\begin{align*}
S \vdash \begin{array}{c}
\vdash \\
\vdash \\
\vdash
\end{array} \\
\Rightarrow \quad \Rightarrow
\end{align*}
\begin{align*}
S \vdash \begin{array}{c}
l_1 \\
l_2
\end{array}
\Rightarrow \rightsquigarrow
\end{align*}
$$
Correctness of Parity

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>even</th>
<th></th>
<th>odd</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>△</td>
<td>even</td>
<td>1</td>
<td>odd</td>
</tr>
<tr>
<td>2</td>
<td>△</td>
<td>even</td>
<td>3</td>
<td>odd</td>
</tr>
<tr>
<td>4</td>
<td>△</td>
<td>even</td>
<td>5</td>
<td>odd</td>
</tr>
</tbody>
</table>

... even | ... odd

Thus it is correct: "\( p \equiv z := 2 \times z \) always produces an even \( z \)."
Correctness of Parity

<table>
<thead>
<tr>
<th></th>
<th>even</th>
<th></th>
<th>odd</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>☑</td>
<td>1</td>
<td>☑</td>
</tr>
<tr>
<td>2</td>
<td>☑</td>
<td>3</td>
<td>☑</td>
</tr>
<tr>
<td>4</td>
<td>☑</td>
<td>5</td>
<td>☑</td>
</tr>
</tbody>
</table>
| ... | ... | ... | ...

\[ z := 2 \times z \vdash [z \mapsto 1] \rightarrow [z \mapsto 2] \quad \text{odd}(z) \leadsto \text{even}(z) \]
\[ z := 2 \times z \vdash [z \mapsto 2] \rightarrow [z \mapsto 4] \quad \text{even}(z) \leadsto \text{even}(z) \]
\[ z := 2 \times z \vdash [z \mapsto 3] \rightarrow [z \mapsto 6] \quad \text{odd}(z) \leadsto \text{even}(z) \]

Thus it is correct: "\( p \equiv z := 2 \times z \) always produces an even \( z \)."
Correctness of Parity

|   | 0  | 1  | 2  | 3  | 4  | 5  | ...
|---|----|----|----|----|----|----|----|
|   | ⊃  | ⊃  | ⊃  | ⊃  | ⊃  | ⊃  | ...
|   | even | odd | even | odd | even | odd | ...

\[
z := 2 \times z \vdash [z \mapsto 1] \rightarrow [z \mapsto 2] \quad \text{odd}(z) \leadsto \text{even}(z)
\]
\[
z := 2 \times z \vdash [z \mapsto 2] \rightarrow [z \mapsto 4] \quad \text{even}(z) \leadsto \text{even}(z)
\]
\[
z := 2 \times z \vdash [z \mapsto 3] \rightarrow [z \mapsto 6] \quad \text{odd}(z) \leadsto \text{even}(z)
\]

\[
1 \quad \text{odd} \land p \vdash 1 \rightarrow 2 \land p \vdash \text{odd} \leadsto \text{even} \Rightarrow 2 \quad \text{even}
\]
\[
2 \quad \text{even} \land p \vdash 2 \rightarrow 4 \land p \vdash \text{even} \leadsto \text{even} \Rightarrow 4 \quad \text{even}
\]
\[
3 \quad \text{odd} \land p \vdash 3 \rightarrow 6 \land p \vdash \text{odd} \leadsto \text{even} \Rightarrow 6 \quad \text{even}
\]

Thus it is correct: 
\[p \equiv z := 2 \times z\] always produces an even \(z\).
Correctness of Parity

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>even</td>
<td>odd</td>
<td>even</td>
<td>odd</td>
<td>even</td>
<td>odd</td>
</tr>
</tbody>
</table>

\[ z := 2 \times z \vdash [z \mapsto 1] \rightarrow [z \mapsto 2] \quad \text{odd}(z) \rightsquigarrow \text{even}(z) \]
\[ z := 2 \times z \vdash [z \mapsto 2] \rightarrow [z \mapsto 4] \quad \text{even}(z) \rightsquigarrow \text{even}(z) \]
\[ z := 2 \times z \vdash [z \mapsto 3] \rightarrow [z \mapsto 6] \quad \text{odd}(z) \rightsquigarrow \text{even}(z) \]

\[ 1 \triangleright \text{odd} \land p \vdash 1 \rightarrow 2 \land p \vdash \text{odd} \rightsquigarrow \text{even} \Rightarrow 2 \triangleright \text{even} \]
\[ 2 \triangleright \text{even} \land p \vdash 2 \rightarrow 4 \land p \vdash \text{even} \rightsquigarrow \text{even} \Rightarrow 4 \triangleright \text{even} \]
\[ 3 \triangleright \text{odd} \land p \vdash 3 \rightarrow 6 \land p \vdash \text{odd} \rightsquigarrow \text{even} \Rightarrow 6 \triangleright \text{even} \]

Thus it is correct: “\( p \equiv z := 2 \times z \) always produces an \textbf{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 $\triangleright$. 
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 $\triangleright$.

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

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

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

\begin{align*}
v \triangleright l_1 & \land l_1 \sqsubseteq l_2 \implies v \triangleright l_2 \quad (1) \\
\forall l \in \mathcal{L}' \subseteq \mathcal{L} : v \triangleright l & \implies v \triangleright \bigcap \mathcal{L}' \quad (2)
\end{align*}
Consider the first of these conditions:

\[ \nu \triangleright l_1 \land l_1 \sqsubseteq l_2 \Rightarrow \nu \triangleright l_2 \]

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.
Consider the first of these conditions:

$$\forall \triangleright l_1 \land l_1 \sqsubseteq l_2 \Rightarrow \forall \triangleright l_2$$

The condition says that the smaller the property is with respect to the partial order, the better (i.e. precise) it is.
Consider the first of these conditions:

\[ \mathbf{v} \triangleright l_1 \land l_1 \sqsubseteq l_2 \Rightarrow \mathbf{v} \triangleright l_2 \]

- 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:

\[ \forall l \in \mathcal{L}' \subseteq \mathcal{L} : v \triangleright l \Rightarrow v \triangleright \bigcap \mathcal{L}' \]
Condition (2)

Looking at the second condition describing correctness:

$$\forall l \in \mathcal{L}' \subseteq \mathcal{L} : v \triangleright l \Rightarrow v \triangleright \bigcap \mathcal{L}'$$

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).
Condition (2)

Looking at the second condition describing correctness:

$$\forall l \in \mathcal{L}' \subseteq \mathcal{L} : v \triangleright l \Rightarrow v \triangleright \bigcap \mathcal{L}'$$

- 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:

$$v \triangleright \top$$

$$v \triangleright l_1 \land v \triangleright l_2 \Rightarrow v \triangleright (l_1 \cap l_2)$$
Again: Parity Example

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

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

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

The conditions imposed on \( \triangleright \) and \( \mathcal{L} \) mean in this case:

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

\[
2 \triangleright \{\text{even}\} \wedge \{\text{even}\} \subseteq \top \Rightarrow 2 \triangleright \top
\]
Again: Parity Example

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

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

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

   $$2 \triangleright \{\text{even}\} \land \{\text{even}\} \subseteq \top \Rightarrow 2 \triangleright \top$$

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

   $$(2 \triangleright \{\text{even}\} \land 2 \triangleright \top) \Rightarrow 2 \triangleright (\{\text{even}\} \cap \top)$$

   i.e. $$(2 \triangleright \{\text{even}\} \land 2 \triangleright \top) \Rightarrow 2 \triangleright \{\text{even}\}$$
Preservation of Correctness via Abstraction

We require that correctness is preserved:
Preservation of Correctness via Abstraction

We require that correctness is preserved:

\[ v_1 \triangleright l_1 \land S \vdash v_1 \rightarrow v_2 \land S \vdash l_1 \rightsquigarrow l_2 \Rightarrow v_2 \triangleright l_2 \]
Preservation of Correctness via Abstraction

We require that correctness is preserved:

\[ v_1 \triangleright l_1 \land S \vdash v_1 \rightarrow v_2 \land S \vdash l_1 \leadsto l_2 \Rightarrow v_2 \triangleright l_2 \]

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

\[ v_1 \triangleright l_1 \land f_S(v_1) = v_2 \land f_S(l_1) = l_2 \Rightarrow v_2 \triangleright l_2 \]
Preservation of Correctness via Abstraction

We require that correctness is preserved:

\[
\nu_1 \triangleright l_1 \land S \models \nu_1 \rightarrow \nu_2 \land S \models l_1 \leadsto l_2 \Rightarrow \nu_2 \triangleright l_2
\]

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

\[
\nu_1 \triangleright l_1 \land f_S(\nu_1) = \nu_2 \land f_S^\#(l_1) = l_2 \Rightarrow \nu_2 \triangleright l_2
\]

This property is also expressed by the following diagram:

\[
\begin{tikzcd}
\mathcal{V} \arrow{r}{f_S} \arrow{d}{\alpha} & \mathcal{V} \\
\mathcal{L} \arrow{d}{\gamma} \arrow{r}{f_S^\#} & \mathcal{L} \\
\end{tikzcd}
\]
We can use a representation function $\beta : \mathcal{V} \rightarrow \mathcal{L}$ to induce a Galois connection $(\mathcal{P}(\mathcal{V}), \alpha, \gamma, \mathcal{L})$ via

$$\alpha(\mathcal{V}) = \bigsqcup \{ \beta(\mathcal{v}) \mid \mathcal{v} \in \mathcal{V} \}$$

$$\gamma(l) = \{ \mathcal{v} \in \mathcal{V} \mid \beta(\mathcal{v}) \sqsubseteq l \}$$
We can use a representation function $\beta : \mathcal{V} \rightarrow \mathcal{L}$ to induce a Galois connection $(\mathcal{P}(\mathcal{V}), \alpha, \gamma, \mathcal{L})$ via

$$\alpha(\mathcal{V}) = \bigsqcup \{ \beta(v) \mid v \in \mathcal{V} \}$$

$$\gamma(l) = \{ v \in \mathcal{V} \mid \beta(v) \sqsubseteq l \}$$

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

$$\alpha(\mathcal{V}) = \{ \eta(v) \mid v \in \mathcal{V} \}$$

$$\gamma(\mathcal{D}) = \{ v \mid \eta(v) \in \mathcal{D} \}$$

in order to construct a Galois connection.
Example: Parity

A representation function $\beta : \mathbb{Z} \rightarrow \mathcal{P}(\{\text{even, odd}\})$ is easily defined by:

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

Correctness implies that the abstract properties are dominated by the actual ones, e.g. $\beta(4) = \{\text{even}\} \subseteq \top = \{\text{even, odd}\}$ is acceptable. This means that we also could use as a representation function $\beta(n) = \top = \{\text{even, odd}\}$ for all $n \in \mathbb{Z}$. Though this would be valid it would also be rather imprecise.
Example: Parity

A representation function $\beta : \mathbb{Z} \rightarrow \mathcal{P}(\{\text{even, odd}\})$ is easily defined by:

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

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

A representation function $\beta : \mathbb{Z} \rightarrow \mathcal{P}\{\text{even, odd}\}$ is easily defined by:

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

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

This means that we also could use as a representation function

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

for all $n \in \mathbb{Z}$. Though this would be valid it would also be rather imprecise.
References Abstract Interpretation


References Abstract Interpretation


References Abstract Interpretation


