Building a Deductive Verifier for Probabilistic Programs

CIRM 2686 - Logic of Probabilistic Programming

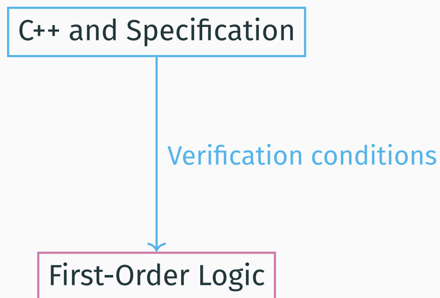
Philipp Schröer

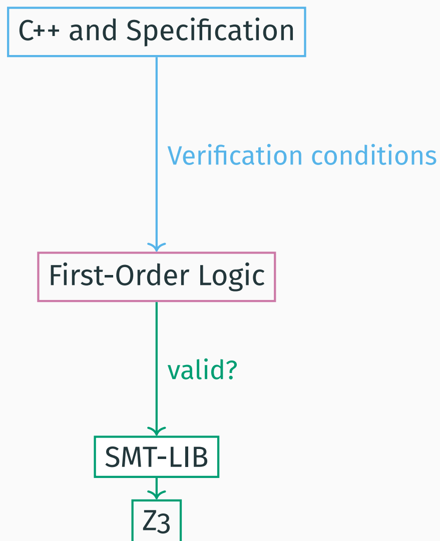
Supervisors: Kevin Batz, Benjamin Kaminski, Christoph Matheja

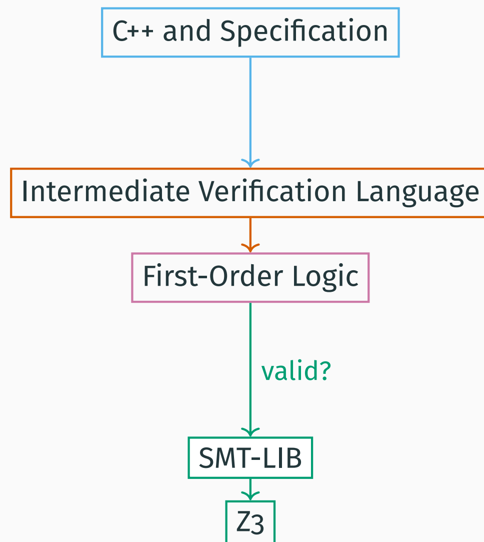
January 31, 2022

RWTH Aachen University

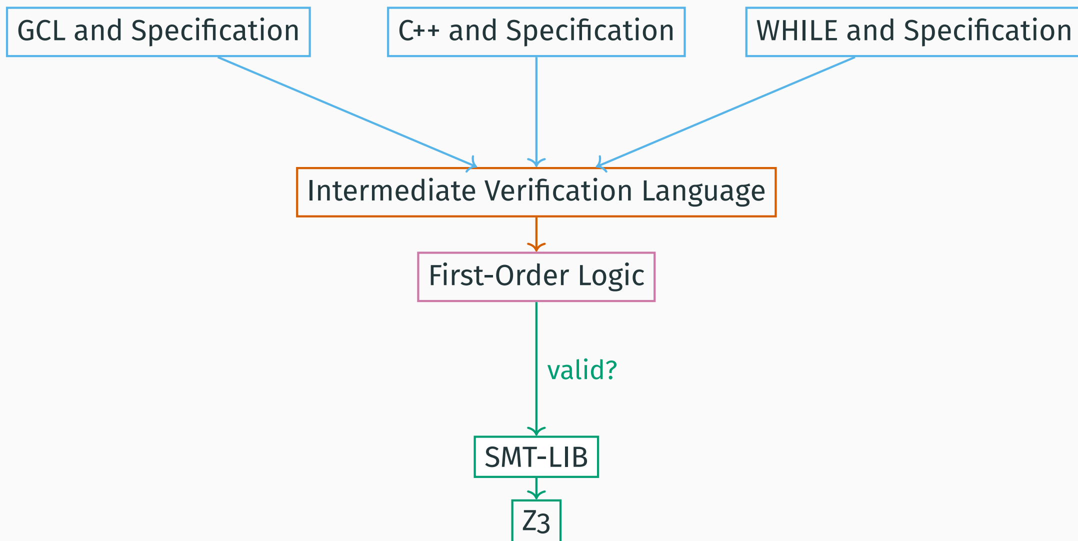
Classical Deductive Verifiers




C++ and Specification
Verification conditions
First-Order Logic
valid?
SMT-LIB
Z3


C++ and Specification
Intermediate Verification Language
First-Order Logic
valid?
SMT-LIB
Z3

Classical Deductive Verifiers



Contributions

Summary: *Design and implementation of a deductive verification infrastructure for probabilistic programs.*

Contributions

Summary: *Design and implementation of a deductive verification infrastructure for probabilistic programs.*

Architecture:

- A quantitative intermediate verification language – *QVL*

QVL

Contributions

Summary: *Design and implementation of a deductive verification infrastructure for probabilistic programs.*

Architecture:

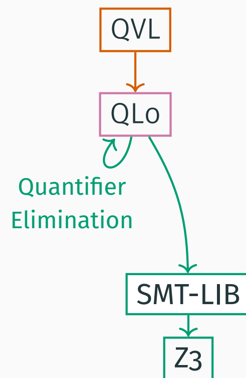
- A quantitative intermediate verification language – *QVL*
- A quantitative assertion language – *QLo*

Contributions

Summary: *Design and implementation of a deductive verification infrastructure for probabilistic programs.*

Architecture:

- A quantitative intermediate verification language – *QVL*
- A quantitative assertion language – *QLo*
- QLo validity checking
    - SMT-LIB encoding of QLo validity
    - Quantifier elimination for QLo



Contributions

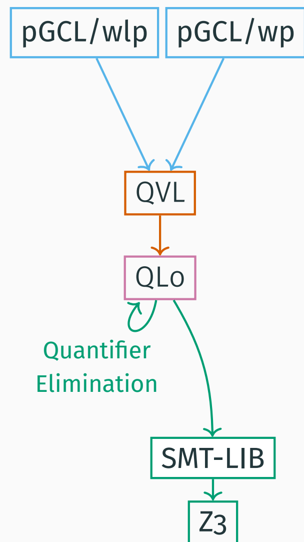
Summary: *Design and implementation of a deductive verification infrastructure for probabilistic programs.*

Architecture:

- A quantitative intermediate verification language – *QVL*
- A quantitative assertion language – *QLo*
- QLo validity checking
  - SMT-LIB encoding of QLo validity
  - Quantifier elimination for QLo

Case study:

- Encodings of pGCL (w.r.t. wlp and wp) into QVL



Contributions

Summary: *Design and implementation of a deductive verification infrastructure for probabilistic programs.*

Architecture:

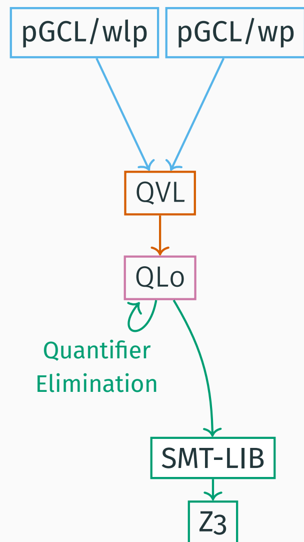
- A quantitative intermediate verification language – *QVL*
- A quantitative assertion language – *QLo*
- QLo validity checking
  - SMT-LIB encoding of QLo validity
  - Quantifier elimination for QLo

Case study:

- Encodings of pGCL (w.r.t. wlp and wp) into QVL

Implementation:

- ≈ 7000 LOC Rust & Python



Probabilistic Programs

Probabilistic Programs, Briefly

Probabilistic programs = Programs + probabilistic choices

The *pGCL* program C_{geo} (*geometric loop*):

$$
\begin{array}{l}
c := 0;\ x := 1; \\
\texttt{while}\ (x = 1)\ \{ \\
\quad \{x := 0\}\ [0.5]\ \{c := c + 1\} \\
\}
\end{array}
$$

Probabilistic programs = Programs + probabilistic choices

The *pGCL* program C_{geo} (*geometric loop*):

```
c := 0; x := 1;
while (x = 1) {
  {x := 0} [0.5] {c := c + 1}
}
```

probabilistic choice

Probabilistic programs = Programs + probabilistic choices

The *pGCL* program C_{geo} (*geometric loop*):

$$c := 0;\ x := 1;$$
$$\texttt{while}\ (x = 1)\ \{$$
$$\quad \{x := 0\}\ [0.5]\ \{c := c + 1\}$$
$$\}$$

1. Distribution of final states:

$$\text{wp}[\![C_{\text{geo}}]\!]([c = 0]) = 0.5$$
$$\text{wp}[\![C_{\text{geo}}]\!]([c = 1]) = 0.25$$
$$\text{wp}[\![C_{\text{geo}}]\!]([c = 2]) = 0.125$$
$$\ldots$$

Probabilistic programs = Programs + probabilistic choices

The *pGCL* program C_{geo} (*geometric loop*):

$$
\begin{aligned}
&c := 0;\ x := 1;\\
&\texttt{while}\ (x = 1)\ \{\\
&\quad \{x := 0\}\ [0.5]\ \{c := c + 1\}\\
&\}
\end{aligned}
$$

1. Distribution of final states:

$$\mathrm{wp}[\![C_{\text{geo}}]\!]([c = 0]) = 0.5$$
$$\mathrm{wp}[\![C_{\text{geo}}]\!]([c = 1]) = 0.25$$
$$\mathrm{wp}[\![C_{\text{geo}}]\!]([c = 2]) = 0.125$$
$$\ldots$$

2. Expected position after termination?

$$\mathrm{wp}[\![C_{\text{geo}}]\!](c) = 1$$

Probabilistic programs = Programs + probabilistic choices

The *pGCL* program C_{geo} (*geometric loop*):

```
c := 0; x := 1;
while (x = 1) {
  {x := 0} [0.5] {c := c + 1}
}
```

1. Distribution of final states:

$$\text{wp}[\![C_{\text{geo}}]\!]([c = 0]) = 0.5$$
$$\text{wp}[\![C_{\text{geo}}]\!]([c = 1]) = 0.25$$
$$\text{wp}[\![C_{\text{geo}}]\!]([c = 2]) = 0.125$$
$$\ldots$$

2. Expected position after termination?

$$\text{wp}[\![C_{\text{geo}}]\!](c) = 1$$

3. Probability of termination?

$$\text{wp}[\![C_{\text{geo}}]\!](1) = 1$$

Expected Values

Expected values are a central concept in the verification of probabilistic programs.

Lower bound $\leq$ Expected value $\leq$ Upper bound

Expected Values

Expected values are a central concept in the verification of probabilistic programs.

Lower bound ≤ Expected value ≤ Upper bound

Existing calculi based on expected values:

- **Total correctness:** *Weakest pre-expectation calculus* (wp) (McIver & Morgan, 2005)
- **Partial correctness:** *Weakest liberal pre-expectation calculus* (wlp) (McIver & Morgan, 2005)
- **Runtimes:** *Expected runtime calculus* (ert) (Kaminski et al., 2016)
- **Probabilistic sensitivity** (Aguirre et al., 2021)
- **Conditional expected values:** *Conditional weakest pre-expectation calculus* (Jansen et al., 2015)
- ...

Expectations

(McIver & Morgan, 2005)

We use *expectations* to talk about expected values.

Expectations: $\mathbb{E} = \{ \quad X : \Sigma \to \mathbb{R}_{\geq 0} \cup \{ \infty \} \quad \}$

We use *expectations* to talk about expected values.

Expectations: $\mathbb{E} = \{ \quad X \colon \Sigma \to \mathbb{R}_{\geq 0} \cup \{ \infty \} \quad \}$

Program states: $\Sigma = \{ \quad \sigma \colon \mathsf{Vars} \to \mathbb{Q}_{\geq 0} \quad \}$

We use *expectations* to talk about expected values.

Expectations: $\mathbb{E} = \{ \quad X\colon \Sigma \to \mathbb{R}_{\geq 0} \cup \{\infty\} \quad \}$

Program states: $\Sigma = \{ \quad \sigma\colon \mathsf{Vars} \to \mathbb{Q}_{\geq 0} \quad \}$

Definition. Weakest Pre-Expectations

Let C be a pGCL program. We define

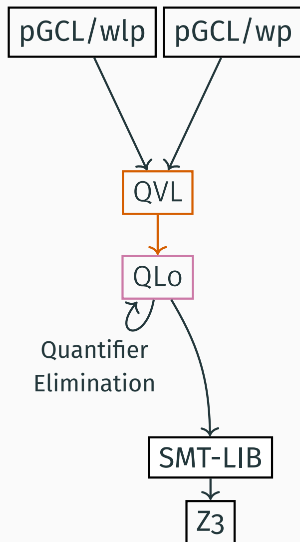
$$\mathrm{wp}[\![C]\!]\colon \mathbb{E} \to \mathbb{E}$$

such that

$$\mathrm{wp}[\![C]\!](X) = \lambda\sigma.\ \text{expected value of } X \text{ on termination of } C\text{, starting in state } \sigma$$

Intermediate Verification Languages

Architecture



A Classical IVL – Inspiration

Predicates: $\mathbb{P} = \{ P : \Sigma \to \{ 0, 1 \} \}$

Idea: An imperative language as a domain-specific language to write *verification conditions*.

A Classical IVL – Inspiration

Predicates: $\mathbb{P} = \{ P : \Sigma \to \{0,1\} \}$

Idea: An imperative language as a domain-specific language to write *verification conditions*.

Semantics of the simple IVL: $\mathbf{vc}[\![S]\!] : \mathsf{ASN} \to \mathsf{ASN}$

S	$\mathbf{vc}[\![S]\!](P)$
$x := a$	$P[x \mapsto a]$
havoc x	$\forall x.\ P$
assume Q	$Q \Rightarrow P$
assert Q	$Q \land P$
$S_1;\ S_2$	$\mathbf{vc}[\![S_1]\!](\mathbf{vc}[\![S_2]\!](P))$
if $(*)$ $\{S_1\}$ else $\{S_2\}$	$\mathbf{vc}[\![S_1]\!](P) \land \mathbf{vc}[\![S_2]\!](P)$

A Classical IVL – Inspiration

Predicates: $\mathbb{P} = \{ P : \Sigma \to \{0,1\} \}$

Idea: An imperative language as a domain-specific language to write *verification conditions*.

Semantics of the simple IVL: $\textbf{vc}[\![S]\!] : \text{ASN} \to \text{ASN}$

S	$\textbf{vc}[\![S]\!](P)$
x `:=` a	$P[x \mapsto a]$
`havoc` x	$\forall x.\ P$
`assume` Q	$Q \Rightarrow P$
`assert` Q	$Q \land P$
S_1`;` S_2	$\textbf{vc}[\![S_1]\!](\textbf{vc}[\![S_2]\!](P))$
`if (*)` $\{S_1\}$ `else` $\{S_2\}$	$\textbf{vc}[\![S_1]\!](P) \land \textbf{vc}[\![S_2]\!](P)$

Definition

S *verifies* iff $\quad \text{vc}[\![S]\!](\text{true}) = \text{true}.$

A Classical IVL – Inspiration

Predicates: $\mathbb{P} = \{ P : \Sigma \to \{ 0,1 \} \}$

Idea: An imperative language as a domain-specific language to write *verification conditions*.

Semantics of the simple IVL: $\mathbf{vc}[\![S]\!] : \text{ASN} \to \text{ASN}$

S	$\mathbf{vc}[\![S]\!](P)$
x `:=` a	$P[x \mapsto a]$
`havoc` x	$\forall x.\ P$
`assume` Q	$Q \Rightarrow P$
`assert` Q	$Q \land P$
S_1`;` S_2	$\mathbf{vc}[\![S_1]\!](\mathbf{vc}[\![S_2]\!](P))$
`if (*)` $\{S_1\}$ `else` $\{S_2\}$	$\mathbf{vc}[\![S_1]\!](P) \land \mathbf{vc}[\![S_2]\!](P)$

Definition

S *verifies* iff $\mathbf{vc}[\![S]\!](\text{true}) = \text{true}$.

Example

`assume` P`;`

S`;`

`assert` Q

A Classical IVL – Inspiration

Predicates: $\mathbb{P} = \{ P : \Sigma \to \{0,1\} \}$

Idea: An imperative language as a domain-specific language to write *verification conditions*.

Semantics of the simple IVL: $\mathbf{vc}[\![S]\!] : \text{ASN} \to \text{ASN}$

S	$\text{vc}[\![S]\!](P)$
$x := a$	$P[x \mapsto a]$
havoc x	$\forall x.\ P$
assume Q	$Q \Rightarrow P$
assert Q	$Q \wedge P$
$S_1;\ S_2$	$\text{vc}[\![S_1]\!](\text{vc}[\![S_2]\!](P))$
if $(*)$ $\{S_1\}$ else $\{S_2\}$	$\text{vc}[\![S_1]\!](P) \wedge \text{vc}[\![S_2]\!](P)$

Definition

S verifies iff $\text{vc}[\![S]\!](\text{true}) = \text{true}$.

Example

```
assume P;
S;
assert Q
// true
```

Predicates: $\mathbb{P} = \{ P : \Sigma \to \{0,1\} \}$

Idea: An imperative language as a domain-specific language to write *verification conditions*.

Semantics of the simple IVL: $\mathbf{vc}[\![S]\!] : \mathrm{ASN} \to \mathrm{ASN}$

S	$\mathsf{vc}[\![S]\!](P)$
$x := a$	$P[x \mapsto a]$
havoc x	$\forall x.\ P$
assume Q	$Q \Rightarrow P$
assert Q	$Q \wedge P$
$S_1;\ S_2$	$\mathsf{vc}[\![S_1]\!](\mathsf{vc}[\![S_2]\!](P))$
if $(*)$ $\{S_1\}$ else $\{S_2\}$	$\mathsf{vc}[\![S_1]\!](P) \wedge \mathsf{vc}[\![S_2]\!](P)$

Definition

S verifies iff $\mathsf{vc}[\![S]\!](\mathrm{true}) = \mathrm{true}$.

Example

```
assume P;

S;
// Q ∧ true
assert Q
// true
```

Predicates: $\mathbb{P} = \{ P : \Sigma \to \{0,1\} \}$

Idea: An imperative language as a domain-specific language to write *verification conditions*.

Semantics of the simple IVL: $\mathbf{vc}[\![S]\!] : \text{ASN} \to \text{ASN}$

S	$\text{vc}[\![S]\!](P)$
`x := a`	$P[x \mapsto a]$
`havoc x`	$\forall x.\ P$
`assume Q`	$Q \Rightarrow P$
`assert Q`	$Q \wedge P$
$S_1 ;\ S_2$	$\text{vc}[\![S_1]\!](\text{vc}[\![S_2]\!](P))$
`if (*) {`S_1`} else {`S_2`}`	$\text{vc}[\![S_1]\!](P) \wedge \text{vc}[\![S_2]\!](P)$

Definition

S *verifies* iff $\quad \text{vc}[\![S]\!](\text{true}) = \text{true}$.

Example

```
assume P;
// vc[[S]](Q)
S;
// Q ∧ true
assert Q
// true
```

Predicates: $\mathbb{P} = \{\, P : \Sigma \to \{0,1\} \,\}$

Idea: An imperative language as a domain-specific language to write *verification conditions*.

Semantics of the simple IVL: $\mathbf{vc}[\![S]\!] : \mathsf{ASN} \to \mathsf{ASN}$

S	$\mathsf{vc}[\![S]\!](P)$
`x := a`	$P[x \mapsto a]$
`havoc x`	$\forall x.\, P$
`assume Q`	$Q \Rightarrow P$
`assert Q`	$Q \wedge P$
$S_1;\, S_2$	$\mathsf{vc}[\![S_1]\!](\mathsf{vc}[\![S_2]\!](P))$
`if (*)` $\{S_1\}$ `else` $\{S_2\}$	$\mathsf{vc}[\![S_1]\!](P) \wedge \mathsf{vc}[\![S_2]\!](P)$

Definition

S *verifies* iff $\mathsf{vc}[\![S]\!](\mathsf{true}) = \mathsf{true}$.

Example

```
// P ⇒ vc⟦S⟧(Q)
assume P;
// vc⟦S⟧(Q)
S;
// Q ∧ true
assert Q
// true
```

A Classical IVL – Inspiration

Predicates: $\mathbb{P} = \{ P : \Sigma \to \{0,1\} \}$

Idea: An imperative language as a domain-specific language to write *verification conditions*.

Semantics of the simple IVL: $\mathbf{vc}[\![S]\!] : \mathrm{ASN} \to \mathrm{ASN}$

S	$\mathrm{vc}[\![S]\!](P)$
x := a	$P[x \mapsto a]$
havoc x	$\forall x.\ P$
assume Q	$Q \Rightarrow P$
assert Q	$Q \wedge P$
S_1; S_2	$\mathrm{vc}[\![S_1]\!](\mathrm{vc}[\![S_2]\!](P))$
if (*) {S_1} else {S_2}	$\mathrm{vc}[\![S_1]\!](P) \wedge \mathrm{vc}[\![S_2]\!](P)$

Definition

S *verifies* iff $\mathrm{vc}[\![S]\!](\mathrm{true}) = \mathrm{true}$.

Example

```
// {P} S {Q}
// P ⇒ vc⟦S⟧(Q)
assume P;
// vc⟦S⟧(Q)
S;
// Q ∧ true
assert Q
// true
```

More Encodings For The Simple IVL

$\texttt{if}\ (b)\ \{S_1\}\ \texttt{else}\ \{S_2\}$

$\texttt{while}\ (b)\ \texttt{invariant}\ I\ \{S\}$

More Encodings For The Simple IVL

$\texttt{if}\ (b)\ \{S_1\}\ \texttt{else}\ \{S_2\}$

$\texttt{while}\ (b)\ \texttt{invariant}\ I\ \{S\}$

```
if (*) {
  assume b;
  S₁
} else {
  assume ¬b;
  S₂
}
```

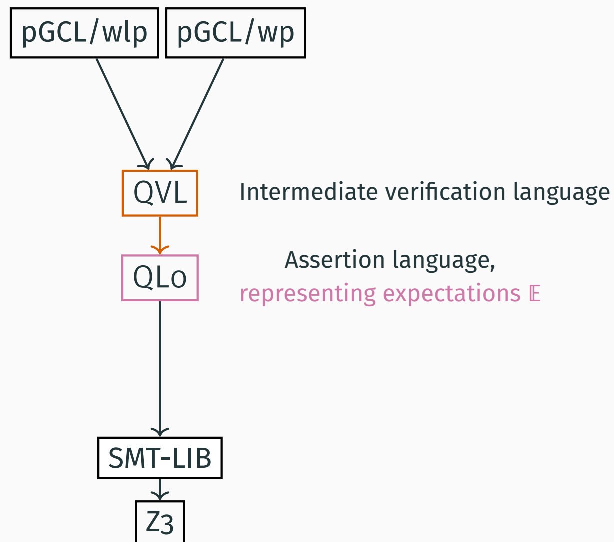
More Encodings For The Simple IVL

`if` (b) $\{S_1\}$ `else` $\{S_2\}$	`while` (b) `invariant` I $\{S\}$

`if` $(*)$ {
  `assume` b;
  S_1
} `else` {
  `assume` $\neg b$;
  S_2
}
	`assert` I;
	`havoc` Vars;
	`assume` I;
	`if` $(*)$ {
	  `assume` b;
	  S
	  `assert` I; `assume` false
	} `else` {
	  `assume` $\neg b$;
	}

QLo and QVL

Architecture



QLo: A Quantitative Assertion Language

Expectations: $\mathbb{E} = \left\{ X : \Sigma \to \mathbb{R}_{\geq 0} \cup \{ \infty \} \right\}$

QLo = Relatively Complete Assertion Language for Probabilistic Programs (Batz et al., 2021) + Logical Operators

QLo: A Quantitative Assertion Language

Expectations: $\mathbb{E} = \left\{ X : \Sigma \to \mathbb{R}_{\geq 0} \cup \{ \infty \} \right\}$

QLo = Relatively Complete Assertion Language for Probabilistic Programs + Logical Operators

representing $\mathbb{E}$

(Batz et al., 2021)

writing $\mathbb{E}$

comparing $\mathbb{E}$

A *syntactic expectation* $\varphi \in \mathsf{Exp}$ represents a (semantic) expectation $[\![\varphi]\!] \in \mathbb{E}$.

A *syntactic expectation* $\varphi \in \mathsf{Exp}$ represents a (semantic) expectation $[\![\varphi]\!] \in \mathbb{E}$.

φ	$[\![\varphi]\!]$
$[x = 3] \cdot \infty + [x \neq 3] \cdot x$	$\lambda \sigma . \begin{cases} \infty, & \text{if } \sigma(x) = 3 \\ \sigma(x), & \text{else} \end{cases}$

A *syntactic expectation* $\varphi \in \mathsf{Exp}$ represents a (semantic) expectation $[\![\varphi]\!] \in \mathbb{E}$.

φ	$[\![\varphi]\!]$
$[x = 3] \cdot \infty + [x \neq 3] \cdot x$	$\lambda\sigma.\ \begin{cases} \infty, & \text{if } \sigma(x) = 3 \\ \sigma(x), & \text{else} \end{cases}$
$\bigsqcup_y [y \cdot y < x] \cdot y$	$\lambda\sigma.\ \sqrt{\sigma(x)}$

A *syntactic expectation* $\varphi \in \mathrm{Exp}$ represents a (semantic) expectation $[\![\varphi]\!] \in \mathbb{E}$.

φ	$[\![\varphi]\!]$
$[x = 3] \cdot \infty + [x \neq 3] \cdot x$	$\lambda\sigma. \begin{cases} \infty, & \text{if } \sigma(x) = 3 \\ \sigma(x), & \text{else} \end{cases}$
$\bigsqcup_y [y \cdot y < x] \cdot y$	$\lambda\sigma. \sqrt{\sigma(x)}$

Why?

- Can express interesting expectations
- *Expressiveness*: If we can write $\varphi \in \mathrm{Exp}$, then we can write $\mathrm{wp}[\![C]\!](\varphi) \in \mathrm{Exp}$. Verification – $\mathrm{wp}[\![S]\!](\varphi) \sqsubseteq \psi$ – is *complete relative to* checking $\sqsubseteq$.

Logical Operators for QLo

Let $\varphi, \psi \in \mathrm{QLo}$.

How to express $\varphi \sqsubseteq \psi$ as a (syntactic) expectation?

We add operators based on *Gödel logic* (Gödel, 1932):

Logical Operators for QLo

Let $\varphi, \psi \in \mathrm{QLo}$.

How to express $\varphi \sqsubseteq \psi$ as a (syntactic) expectation?

We add operators based on *Gödel logic* (Gödel, 1932):

φ	$[\![\varphi]\!]$
$\varphi \to \psi$	$\lambda \sigma. \begin{cases} \infty, & \text{if } [\![\varphi]\!](\sigma) \leq [\![\psi]\!](\sigma) \\ [\![\psi]\!](\sigma), & \text{else} \end{cases}$

Logical Operators for QLo

Let $\varphi, \psi \in \mathrm{QLo}$.

How to express $\varphi \sqsubseteq \psi$ as a (syntactic) expectation?

We add operators based on *Gödel logic* (Gödel, 1932):

φ	$[\![\varphi]\!]$
$\varphi \to \psi$	$\lambda\sigma. \begin{cases} \infty, & \text{if } [\![\varphi]\!](\sigma) \leq [\![\psi]\!](\sigma) \\ [\![\psi]\!](\sigma), & \text{else} \end{cases}$

Example

$$2 \to 3 \equiv \infty$$
$$3 \to 2 \equiv 2$$
$$(x+1) \to x \equiv x$$

Logical Operators for QLo

Let $\varphi, \psi \in \mathrm{QLo}$.

How to express $\varphi \sqsubseteq \psi$ as a (syntactic) expectation?

We add operators based on *Gödel logic* (Gödel, 1932):

φ	$[\![\varphi]\!]$
$\varphi \to \psi$	$\lambda\sigma. \begin{cases} \infty, & \text{if } [\![\varphi]\!](\sigma) \leq [\![\psi]\!](\sigma) \\ [\![\psi]\!](\sigma), & \text{else} \end{cases}$

Example

$$\begin{aligned} 2 \to 3 &\equiv \infty \\ 3 \to 2 &\equiv 2 \\ (x+1) \to x &\equiv x \end{aligned}$$

Theorem

$\varphi \to \psi \equiv \infty$ iff $\varphi \sqsubseteq \psi$

Logical Operators for QLo

Let $\varphi, \psi \in \mathrm{QLo}$.
How to express $\varphi \sqsubseteq \psi$ as a (syntactic) expectation?

We add operators based on *Gödel logic* (Gödel, 1932):

φ	$[\![\varphi]\!]$
$\varphi \to \psi$	$\lambda\sigma.\begin{cases}\infty, & \text{if } [\![\varphi]\!](\sigma) \leq [\![\psi]\!](\sigma) \\ [\![\psi]\!](\sigma), & \text{else}\end{cases}$

Example

$$\begin{aligned} 2 \to 3 &\equiv \infty \\ 3 \to 2 &\equiv 2 \\ (x+1) \to x &\equiv x \end{aligned}$$

Theorem

$$\varphi \to \psi \equiv \infty \quad \text{iff} \quad \varphi \sqsubseteq \psi$$

Definition

φ is *valid* iff $\varphi \equiv \infty$.

Let $\varphi, \psi \in \mathrm{QLo}$.
How to express $\varphi \sqsubseteq \psi$ as a (syntactic) expectation?

We add operators based on *Gödel logic* (Gödel, 1932):

φ	$[\![\varphi]\!]$
$\varphi \rightarrow \psi$	$\lambda\sigma.\begin{cases}\infty, & \text{if } [\![\varphi]\!](\sigma) \le [\![\psi]\!](\sigma)\\ [\![\psi]\!](\sigma), & \text{else}\end{cases}$
$\varphi \searrow \psi$	$\lambda\sigma.\begin{cases}\infty, & \text{if } [\![\varphi]\!](\sigma) \le [\![\psi]\!](\sigma)\\ 0, & \text{else}\end{cases}$

Example

$$2 \rightarrow 3 \quad\equiv\quad \infty$$
$$3 \rightarrow 2 \quad\equiv\quad 2$$
$$(x+1) \rightarrow x \quad\equiv\quad x$$

Theorem

$\varphi \rightarrow \psi \equiv \infty$ iff $\varphi \sqsubseteq \psi$

Definition

φ is *valid* iff $\varphi \equiv \infty$.

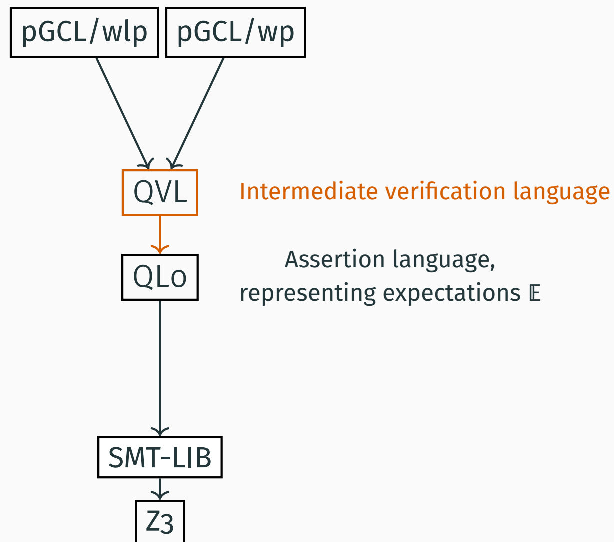
The Quantitative Assertion Language QLo

A QLo formula φ represents an expectation $[\![\varphi]\!] \in \mathbb{E}$.

$$
\begin{aligned}
\varphi \quad ::= \quad & a && \mid \neg\varphi \\
& \mid \varphi + \varphi && \mid \sim\varphi \\
& \mid \varphi \cdot \varphi && \\
& \mid [b] && \\
& \mid ?(b) && \\
& \mid \prod_{x} \varphi && \mid \coprod_{x} \varphi \\
& \mid \varphi \sqcap \varphi && \mid \varphi \sqcup \varphi \\
& \mid \varphi \rightarrow \varphi && \mid \varphi \leftarrow \varphi \\
& \mid \varphi \searrow \varphi && \mid \varphi \nwarrow \varphi
\end{aligned}
$$

a is an arithmetic expression and b is a Boolean expression.

Architecture



Intermediate verification language

Assertion language,
representing expectations $\mathbb{E}$

A Quantitative IVL (for Lower Bounds)

Semantics: $\mathbf{vc}[\![S]\!] : \mathrm{QLo} \to \mathrm{QLo}$

S	$\mathbf{vc}[\![S]\!](\psi)$
x `:=` a	$\psi[x \mapsto a]$
`havoc` x	$\prod_y \psi$
`assume` φ	$\varphi \to \psi$
`compare` φ	$\varphi \searrow \psi$
`assert` φ	$\varphi \sqcap \psi$
S_1`;` S_2	$\mathbf{vc}[\![S_1]\!](\mathbf{vc}[\![S_2]\!](\psi))$
`if` $(\sqcap)$ $\{S_1\}$ `else` $\{S_2\}$	$\mathbf{vc}[\![S_1]\!](\psi) \sqcap \mathbf{vc}[\![S_2]\!](\psi)$
$\{S_1\}$ $[p]$ $\{S_2\}$	$p \cdot \mathbf{vc}[\![S_1]\!](\psi) + (1-p) \cdot \mathbf{vc}[\![S_2]\!](\psi)$

Semantics: $\mathbf{vc}[\![S]\!] : \mathrm{QLo} \to \mathrm{QLo}$

S	$\mathbf{vc}[\![S]\!](\psi)$
`x := a`	$\psi[x \mapsto a]$
`havoc x`	$\bigsqcap_y \psi$
`assume φ`	$\varphi \to \psi$
`compare φ`	$\varphi \searrow \psi$
`assert φ`	$\varphi \sqcap \psi$
S_1 `;` S_2	$\mathbf{vc}[\![S_1]\!](\mathbf{vc}[\![S_2]\!](\psi))$
`if` $(\sqcap)$ $\{S_1\}$ `else` $\{S_2\}$	$\mathbf{vc}[\![S_1]\!](\psi) \sqcap \mathbf{vc}[\![S_2]\!](\psi)$
$\{S_1\}$ $[p]$ $\{S_2\}$	$p \cdot \mathbf{vc}[\![S_1]\!](\psi) + (1-p) \cdot \mathbf{vc}[\![S_2]\!](\psi)$

A Quantitative IVL (for Lower Bounds)

Semantics: $\mathbf{vc}[\![S]\!] : \mathrm{QLo} \to \mathrm{QLo}$

S	$\mathbf{vc}[\![S]\!](\psi)$
$x := a$	$\psi[x \mapsto a]$
$\mathtt{havoc}\ x$	$\prod_y \psi$
$\mathtt{assume}\ \varphi$	$\varphi \to \psi$
$\mathtt{compare}\ \varphi$	$\varphi \searrow \psi$
$\mathtt{assert}\ \varphi$	$\varphi \sqcap \psi$
$S_1;\ S_2$	$\mathbf{vc}[\![S_1]\!](\mathbf{vc}[\![S_2]\!](\psi))$
$\mathtt{if}\ (\sqcap)\ \{S_1\}\ \mathtt{else}\ \{S_2\}$	$\mathbf{vc}[\![S_1]\!](\psi) \sqcap \mathbf{vc}[\![S_2]\!](\psi)$
$\{S_1\}\ [p]\ \{S_2\}$	$p \cdot \mathbf{vc}[\![S_1]\!](\psi) + (1-p) \cdot \mathbf{vc}[\![S_2]\!](\psi)$

Definition

S verifies iff $\quad \mathbf{vc}[\![S]\!](\infty) \equiv \infty$.

A Quantitative IVL (for Lower Bounds)

Semantics: $\mathbf{vc}[\![S]\!] : \mathrm{QLo} \to \mathrm{QLo}$

S	$\mathbf{vc}[\![S]\!](\psi)$
$x := a$	$\psi[x \mapsto a]$
$\mathtt{havoc}\ x$	$\prod_y \psi$
$\mathtt{assume}\ \varphi$	$\varphi \to \psi$
$\mathtt{compare}\ \varphi$	$\varphi \searrow \psi$
$\mathtt{assert}\ \varphi$	$\varphi \sqcap \psi$
$S_1;\ S_2$	$\mathbf{vc}[\![S_1]\!](\mathbf{vc}[\![S_2]\!](\psi))$
$\mathtt{if}\ (\sqcap)\ \{S_1\}\ \mathtt{else}\ \{S_2\}$	$\mathbf{vc}[\![S_1]\!](\psi) \sqcap \mathbf{vc}[\![S_2]\!](\psi)$
$\{S_1\}\ [p]\ \{S_2\}$	$p \cdot \mathbf{vc}[\![S_1]\!](\psi) + (1-p) \cdot \mathbf{vc}[\![S_2]\!](\psi)$

Definition

S verifies iff $\quad \mathbf{vc}[\![S]\!](\infty) \equiv \infty.$

Example

```
assume φ;
S;
assert ψ
```

Semantics: $\mathbf{vc}[\![S]\!] : \mathrm{QLo} \to \mathrm{QLo}$

S	$\mathbf{vc}[\![S]\!](\psi)$
$x := a$	$\psi[x \mapsto a]$
$\mathtt{havoc}\ x$	$\bigsqcap_y \psi$
$\mathtt{assume}\ \varphi$	$\varphi \to \psi$
$\mathtt{compare}\ \varphi$	$\varphi \searrow \psi$
$\mathtt{assert}\ \varphi$	$\varphi \sqcap \psi$
$S_1;\ S_2$	$\mathbf{vc}[\![S_1]\!](\mathbf{vc}[\![S_2]\!](\psi))$
$\mathtt{if}\ (\sqcap)\ \{S_1\}\ \mathtt{else}\ \{S_2\}$	$\mathbf{vc}[\![S_1]\!](\psi) \sqcap \mathbf{vc}[\![S_2]\!](\psi)$
$\{S_1\}\ [p]\ \{S_2\}$	$p \cdot \mathbf{vc}[\![S_1]\!](\psi) + (1-p) \cdot \mathbf{vc}[\![S_2]\!](\psi)$

Definition

S verifies iff $\quad \mathbf{vc}[\![S]\!](\infty) \equiv \infty$.

Example

```
assume φ;

S;

assert ψ
// ∞
```

A Quantitative IVL (for Lower Bounds)

Semantics: $\mathbf{vc}[\![S]\!] : \mathrm{QLo} \to \mathrm{QLo}$

S	$\mathbf{vc}[\![S]\!](\psi)$
`x := a`	$\psi[x \mapsto a]$
`havoc x`	$\prod_y \psi$
`assume` φ	$\varphi \to \psi$
`compare` φ	$\varphi \searrow \psi$
`assert` φ	$\varphi \sqcap \psi$
S_1; S_2	$\mathbf{vc}[\![S_1]\!](\mathbf{vc}[\![S_2]\!](\psi))$
`if` $(\sqcap)$ $\{S_1\}$ `else` $\{S_2\}$	$\mathbf{vc}[\![S_1]\!](\psi) \sqcap \mathbf{vc}[\![S_2]\!](\psi)$
$\{S_1\}$ $[p]$ $\{S_2\}$	$p \cdot \mathbf{vc}[\![S_1]\!](\psi) + (1-p) \cdot \mathbf{vc}[\![S_2]\!](\psi)$

Definition

S verifies iff $\mathbf{vc}[\![S]\!](\infty) \equiv \infty$.

Example

```
assume φ;

S;
// ψ ⊓ ∞
assert ψ
// ∞
```

A Quantitative IVL (for Lower Bounds)

Semantics: $\mathbf{vc}[\![S]\!] : \mathrm{QLo} \to \mathrm{QLo}$

S	$\mathbf{vc}[\![S]\!](\psi)$
$x := a$	$\psi[x \mapsto a]$
$\mathtt{havoc}\ x$	$\prod_y \psi$
$\mathtt{assume}\ \varphi$	$\varphi \to \psi$
$\mathtt{compare}\ \varphi$	$\varphi \searrow \psi$
$\mathtt{assert}\ \varphi$	$\varphi \sqcap \psi$
$S_1 ; S_2$	$\mathbf{vc}[\![S_1]\!](\mathbf{vc}[\![S_2]\!](\psi))$
$\mathtt{if}\ (\sqcap)\ \{S_1\}\ \mathtt{else}\ \{S_2\}$	$\mathbf{vc}[\![S_1]\!](\psi) \sqcap \mathbf{vc}[\![S_2]\!](\psi)$
$\{S_1\}\ [p]\ \{S_2\}$	$p \cdot \mathbf{vc}[\![S_1]\!](\psi) + (1-p) \cdot \mathbf{vc}[\![S_2]\!](\psi)$

Definition

S verifies iff $\mathbf{vc}[\![S]\!](\infty) \equiv \infty$.

Example

```
assume φ;
// vc⟦S⟧(ψ)
S;
// ψ ⊓ ∞
assert ψ
// ∞
```

Semantics: $\mathsf{vc}[\![S]\!] : \mathrm{QLo} \to \mathrm{QLo}$

S	$\mathsf{vc}[\![S]\!](\psi)$
`x := a`	$\psi[x \mapsto a]$
`havoc x`	$\bigsqcap_y \psi$
`assume φ`	$\varphi \to \psi$
`compare φ`	$\varphi \searrow \psi$
`assert φ`	$\varphi \sqcap \psi$
S_1; S_2	$\mathsf{vc}[\![S_1]\!](\mathsf{vc}[\![S_2]\!](\psi))$
`if` $(\sqcap)$ $\{S_1\}$ `else` $\{S_2\}$	$\mathsf{vc}[\![S_1]\!](\psi) \sqcap \mathsf{vc}[\![S_2]\!](\psi)$
$\{S_1\}$ $[p]$ $\{S_2\}$	$p \cdot \mathsf{vc}[\![S_1]\!](\psi) + (1-p) \cdot \mathsf{vc}[\![S_2]\!](\psi)$

Definition

S verifies iff $\mathsf{vc}[\![S]\!](\infty) \equiv \infty$.

Example

```
// φ → vc⟦S⟧(ψ)
assume φ;
// vc⟦S⟧(ψ)
S;
// ψ ⊓ ∞
assert ψ
// ∞
```

A Quantitative IVL (for Lower Bounds)

Semantics: $\mathbf{vc}[\![S]\!] : \mathrm{QLo} \to \mathrm{QLo}$

S	$\mathbf{vc}[\![S]\!](\psi)$
$x := a$	$\psi[x \mapsto a]$
$\mathtt{havoc}\ x$	$\prod_y \psi$
$\mathtt{assume}\ \varphi$	$\varphi \to \psi$
$\mathtt{compare}\ \varphi$	$\varphi \searrow \psi$
$\mathtt{assert}\ \varphi$	$\varphi \sqcap \psi$
$S_1;\ S_2$	$\mathbf{vc}[\![S_1]\!](\mathbf{vc}[\![S_2]\!](\psi))$
$\mathtt{if}\ (\sqcap)\ \{S_1\}\ \mathtt{else}\ \{S_2\}$	$\mathbf{vc}[\![S_1]\!](\psi) \sqcap \mathbf{vc}[\![S_2]\!](\psi)$
$\{S_1\}\ [p]\ \{S_2\}$	$p \cdot \mathbf{vc}[\![S_1]\!](\psi) + (1-p) \cdot \mathbf{vc}[\![S_2]\!](\psi)$

Definition

S verifies iff $\quad \mathbf{vc}[\![S]\!](\infty) \equiv \infty.$

Example

Verifies iff $\varphi \sqsubseteq \mathbf{vc}[\![S]\!](\psi)$:

```
// φ → vc⟦S⟧(ψ)
assume φ;
// vc⟦S⟧(ψ)
S;
// ψ ⊓ ∞
assert ψ
// ∞
```

QVL Syntax

Insight: We generalizated from Boolean case to lower bounds of expectations, but upper bounds require lattice-theoretic duals.

Syntax of a QVL program S**:**

$$
\begin{aligned}
S \;::=\;& \texttt{skip} \\
&\mid x := a \\
&\mid S;\, S \\
&\mid \{S\}\,[p]\,\{S\} \\
&\mid \texttt{down havoc}\; x \\
&\mid \texttt{down assert}\; \varphi \\
&\mid \texttt{down assume}\; \varphi \\
&\mid \texttt{down compare}\; \varphi \\
&\mid \texttt{if}\;(\sqcap)\;\{S\}\;\texttt{else}\;\{S\} \\
&\mid \texttt{down negate} \\
&\mid \texttt{up negate} \\
&\mid \texttt{up havoc}\; x \\
&\mid \texttt{up assert}\; \varphi \\
&\mid \texttt{up assume}\; \varphi \\
&\mid \texttt{up compare}\; \varphi \\
&\mid \texttt{if}\;(\sqcup)\;\{S\}\;\texttt{else}\;\{S\}
\end{aligned}
$$

QVL: Lower and Upper Bounds

$$\varphi \sqsubseteq \mathsf{vc}[\![S]\!](\psi)$$

$$\mathsf{vc}[\![S]\!](\psi) \sqsubseteq \varphi$$

```
down assume φ;
S;
down assert ψ
```

QVL: Lower and Upper Bounds

$\varphi \sqsubseteq \mathsf{vc}[\![S]\!](\psi)$

$\mathsf{vc}[\![S]\!](\psi) \sqsubseteq \varphi$

```
down assume φ;
S;
down assert ψ
```

```
down negate;
up assume φ;
S;
up assert ψ;
up negate
```

QVL: Lower and Upper Bounds

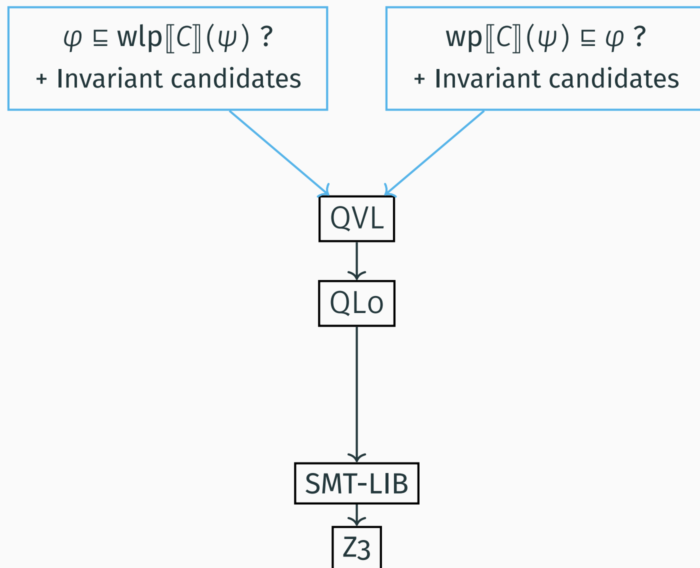
$\varphi \sqsubseteq \mathsf{vc}[\![S]\!](\psi)$	$\mathsf{vc}[\![S]\!](\psi) \sqsubseteq \varphi$
down assume φ; S; down assert ψ	down negate; up assume φ; S; up assert ψ; up negate

All of our pGCL encodings are dual (modulo some negations)

Encoding pGCL

Two big questions:

1. When is an encoding correct?
2. How do we encode loops?



Definition. Soundness (lower bounds of wlp)

Let $E(C) \in \mathsf{QVL}$ be an encoding of $C \in \mathsf{pGCL}$.

For all $\varphi \in \mathrm{QLo}$ and $\psi \in \mathrm{QLo}$, we require

$$\varphi \sqsubseteq \mathsf{vc}[\![E(C)]\!](\psi) \quad \Longrightarrow \quad \varphi \sqsubseteq \mathsf{wlp}[\![C]\!](\psi)\ .$$

Definition. Soundness (lower bounds of wlp)

Let $E(C) \in \mathrm{QVL}$ be an encoding of $C \in$ pGCL.

For all $\varphi \in \mathrm{QLo}$ and $\psi \in \mathrm{QLo}$, we require

$$\varphi \sqsubseteq \mathrm{vc}[\![E(C)]\!](\psi) \quad \Longrightarrow \quad \varphi \sqsubseteq \mathrm{wlp}[\![C]\!](\psi)\ .$$

This condition is equivalent to: $\mathrm{vc}[\![E(C)]\!](\psi) \sqsubseteq \mathrm{wlp}[\![C]\!](\psi)\ .$

Soundness: $\mathrm{vc}[\![E(C)]\!](\psi) \sqsubseteq \mathrm{wlp}[\![C]\!](\psi)$.

Theorem

For any loop $C = \texttt{while}\ (b)\ \{C'\}$,

$$0 = \mathrm{vc}[\![\ \texttt{down assert 0}\ ]\!](\psi) \sqsubseteq \mathrm{wlp}[\![C]\!](\psi)\ .$$

Encoding Loops 1

Soundness: $\mathrm{vc}[\![E(C)]\!](\psi) \sqsubseteq \mathrm{wlp}[\![C]\!](\psi)$.

Theorem. Park Co-Induction **(McIver & Morgan, 2005)**

Let $C = \texttt{while}\ (b)\ \{C'\}$ and $I \in \mathbb{E}_{\leq 1}$.

Encoding Loops 1

Soundness: $\mathrm{vc}[\![E(C)]\!](\psi) \sqsubseteq \mathrm{wlp}[\![C]\!](\psi)$.

Theorem. Park Co-Induction **(McIver & Morgan, 2005)**

Let $C = \texttt{while}\ (b)\ \{C'\}$ and $I \in \mathbb{E}_{\leq 1}$.

I is a valid sub-invariant w.r.t. ψ when $I \sqsubseteq [b] \cdot \mathrm{wlp}[\![C']\!](I) + [\neg b] \cdot \psi$.

Encoding Loops 1

Soundness: $\mathrm{vc}[\![E(C)]\!](\psi) \sqsubseteq \mathrm{wlp}[\![C]\!](\psi)$.

Theorem. Park Co-Induction **(McIver & Morgan, 2005)**

Let $C = \texttt{while}\ (b)\ \{C'\}$ and $I \in \mathbb{E}_{\leq 1}$.

I is a valid sub-invariant w.r.t. ψ when $I \sqsubseteq [b] \cdot \mathrm{wlp}[\![C']\!](I) + [\neg b] \cdot \psi$.

Encoding Loops 1

Soundness: $\mathrm{vc}[\![E(C)]\!](\psi) \sqsubseteq \mathrm{wlp}[\![C]\!](\psi)$.

Theorem. Park Co-Induction **(McIver & Morgan, 2005)**

Let $C = \mathtt{while}\ (b)\ \{C'\}$ and $I \in \mathbb{E}_{\leq 1}$.

I is a valid sub-invariant w.r.t. ψ when $I \sqsubseteq [b] \cdot \mathrm{wlp}[\![C']\!](I) + [\neg b] \cdot \psi$.

Soundness: $\mathrm{vc}[\![E(C)]\!](\psi) \sqsubseteq \mathrm{wlp}[\![C]\!](\psi)$.

Theorem. Park Co-Induction **(McIver & Morgan, 2005)**

Let $C = \texttt{while}\ (b)\ \{C'\}$ and $I \in \mathbb{E}_{\leq 1}$.

I is a valid sub-invariant w.r.t. ψ when $I \sqsubseteq [b] \cdot \mathrm{wlp}[\![C']\!](I) + [\neg b] \cdot \psi$.

If that is the case, then $I \sqsubseteq \mathrm{wlp}[\![C]\!](\psi)$.

Soundness: $\mathrm{vc}[\![E(C)]\!](\psi) \sqsubseteq \mathrm{wlp}[\![C]\!](\psi)$.

Theorem. Park Co-Induction **(McIver & Morgan, 2005)**

Let $C = \texttt{while}\ (b)\ \{C'\}$ and $I \in \mathbb{E}_{\leq 1}$.

I is a valid sub-invariant w.r.t. ψ when $\quad I \sqsubseteq [b] \cdot \mathrm{wlp}[\![C']\!](I) + [\neg b] \cdot \psi$.

If that is the case, then $\quad I \sqsubseteq \mathrm{wlp}[\![C]\!](\psi)$.

We want an encoding $E(C) \in \mathrm{QVL}$ such that

$$\mathrm{vc}[\![E(C)]\!](\psi) = \lambda\sigma. \begin{cases} [\![I]\!](\sigma), & \text{if } I \text{ is a valid sub-invariant w.r.t. } \psi \\ 0, & \text{else} \end{cases}$$

Encoding Loops 2

$$\mathrm{vc}[\![S_1]\!](\rho)(\sigma) = \begin{cases} I(\sigma), & \text{if } I \sqsubseteq \rho \\ 0, & \text{else} \end{cases}$$

Encoding Loops 2

$S_1 =$ `down assert` I;
`down havoc` Vars;
`down compare` I;

$$\mathrm{vc}[\![S_1]\!](\rho)(\sigma) = \begin{cases} I(\sigma), & \text{if } I \sqsubseteq \rho \\ 0, & \text{else} \end{cases}$$

$S_1 =$
```
down assert I;
down havoc Vars;
down compare I;
```

$$\mathrm{vc}[\![S_1]\!](\rho)(\sigma) = \begin{cases} I(\sigma), & \text{if } I \sqsubseteq \rho \\ 0, & \text{else} \end{cases}$$

Encoding Loops 2

$S_1 =$

```
down assert I;
down havoc Vars;
down compare I;
```

$$\mathrm{vc}[\![S_1]\!](\rho)(\sigma) = \begin{cases} I(\sigma), & \text{if } I \sqsubseteq \rho \\ 0, & \text{else} \end{cases}$$

Encoding Loops 2

```
S_1 =   down assert I;
        down havoc Vars;
        down compare I;
```

$$\mathrm{vc}[\![S_1]\!](\rho)(\sigma) = \begin{cases} I(\sigma), & \text{if } I \sqsubseteq \rho \\ 0, & \text{else} \end{cases}$$

$$\mathrm{vc}[\![S_2]\!](\psi) = [b] \cdot \mathrm{vc}[\![S']\!](I) + [\neg b] \cdot \psi$$

Encoding Loops 2

$S_1 =$

```
down assert I;
down havoc Vars;
down compare I;
```

$$\mathrm{vc}[\![S_1]\!](\rho)(\sigma) = \begin{cases} I(\sigma), & \text{if } I \sqsubseteq \rho \\ 0, & \text{else} \end{cases}$$

$S_2 =$

```
if (⊓) {
  down assume [b];
  S′;
  down assert I;
  down assume 0
} else {
  down assume [¬b]
}
```

$$\mathrm{vc}[\![S_2]\!](\psi) \quad = [b] \cdot \mathrm{vc}[\![S']\!](I) + [\neg b] \cdot \psi$$

Encoding Loops 2

$S_1 =$

```
down assert I;
down havoc Vars;
down compare I;
```

$$\mathrm{vc}[\![S_1]\!](\rho)(\sigma) = \begin{cases} I(\sigma), & \text{if } I \sqsubseteq \rho \\ 0, & \text{else} \end{cases}$$

$S_2 =$

```
if (⊓) {
  down assume [b];
  S';
  down assert I;
  down assume 0
} else {
  down assume [¬b]
}
```

$$\mathrm{vc}[\![S_2]\!](\psi) = [b] \cdot \mathrm{vc}[\![S']\!](I) + [\neg b] \cdot \psi$$

Encoding Loops 2

$S_1 =$

```
down assert I;
down havoc Vars;
down compare I;
```

$$\mathrm{vc}[\![S_1]\!](\rho)(\sigma) = \begin{cases} I(\sigma), & \text{if } I \sqsubseteq \rho \\ 0, & \text{else} \end{cases}$$

$S_2 =$

```
if (⊓) {
  down assume [b];
  S';
  down assert I;
  down assume 0
} else {
  down assume [¬b]
}
```

$$\mathrm{vc}[\![S_2]\!](\psi) = [b] \cdot \mathrm{vc}[\![S']\!](I) + [\neg b] \cdot \psi$$

Encoding Loops 2

$S_1 =$

```
down assert I;
down havoc Vars;
down compare I;
```

$S_2 =$

```
if (⊓) {
  down assume [b];
  S';
  down assert I;
  down assume 0
} else {
  down assume [¬b]
}
```

$$\mathrm{vc}[\![S_1;\ S_2]\!](\psi) = \lambda\sigma.\begin{cases}[\![I]\!](\sigma), & \text{if } I \text{ is a valid sub-invariant w.r.t. } \psi \\ 0, & \text{else}\end{cases}$$

pGCL Encodings

For **wlp**, we have encodings S of C such that

$$\forall \varphi \sqsubseteq 1. \qquad \mathrm{vc}[\![S]\!](\varphi) \sqsubseteq \mathrm{wlp}[\![C]\!](\varphi) \; .$$

For **wp**, we have encodings S of C such that

$$\forall \varphi. \qquad \mathrm{wp}[\![C]\!](\varphi) \sqsubseteq \mathrm{vc}[\![S]\!](\varphi) \; .$$

pGCL Encodings

For **wlp**, we have encodings S of C such that

$$\forall \varphi \sqsubseteq 1. \qquad \mathsf{vc}[\![S]\!](\varphi) \sqsubseteq \mathsf{wlp}[\![C]\!](\varphi)\ .$$

For **wp**, we have encodings S of C such that

$$\forall \varphi. \qquad \mathsf{wp}[\![C]\!](\varphi) \sqsubseteq \mathsf{vc}[\![S]\!](\varphi)\ .$$

Also: we support k-*induction*, a generalization of the loop rule from before.

Implementation

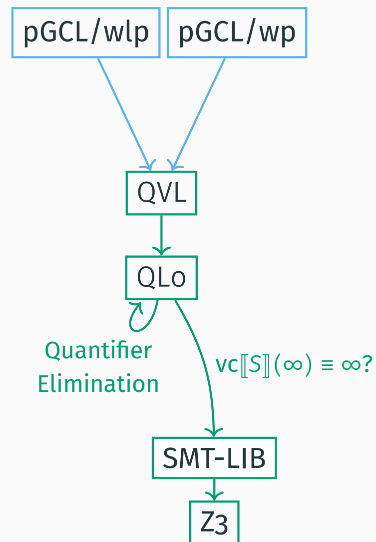
Implementation

pgcl2heyvl (Python):

- Recursive translation from pGCL to QVL
- Supports wlp and wp

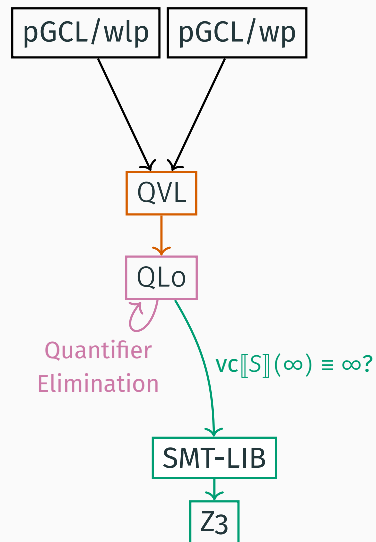
Caesar (Rust): *"veni, vidi, vc?"*

- Generates verification conditions from QVL
- Optimizes QLo (e.g. quantifier elimination)
- Validity checking of QLo using Z3



Generation of verification conditions $\mathrm{vc}[\![S]\!](\infty)$:

- Strict generation by definition
- Lazy generation by interspersing SAT checks with generation



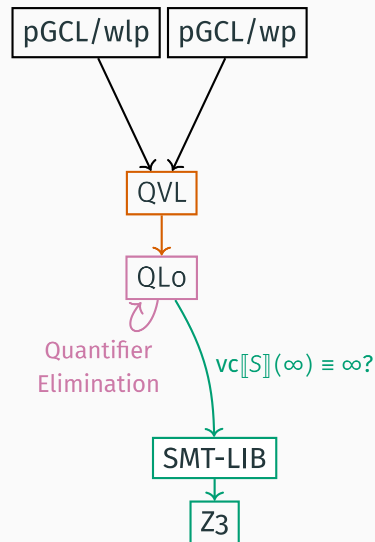
Generation of verification conditions $\mathrm{vc}[\![S]\!](\infty)$:

- Strict generation by definition
- Lazy generation by interspersing SAT checks with generation

Optimization of QLo:

- Quantifier elimination:

  Transform φ into quantifier-free φ' such that φ is valid iff φ' is valid.
- Detect qualitative sub-formulas
- Expression simplification



Generation of verification conditions $\mathrm{vc}[\![S]\!](\infty)$:

- Strict generation by definition
- Lazy generation by interspersing SAT checks with generation

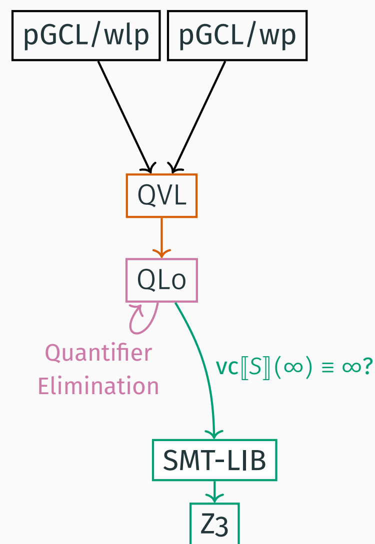
Optimization of QLo:

- Quantifier elimination:

  Transform φ into quantifier-free φ' such that φ is valid iff φ' is valid.
- Detect qualitative sub-formulas
- Expression simplification

Validity checking of QLo using Z3:

- Encoding using Z3 datatypes
- Encoding using built-in datatypes



Experiments

Tested against *kipro2*, a tool for probabilistic k-induction (Batz et al. 2021).

- Programs that consist of only one loop
- Competetive on benchmark set for wp

  (e.g. geometric loop, uniform number generation, Rabin's mutual exclusion)

Experiments

Tested against *kipro2*, a tool for probabilistic k-induction (Batz et al. 2021).

- Programs that consist of only one loop
- Competetive on benchmark set for wp

  (e.g. geometric loop, uniform number generation, Rabin's mutual exclusion)

Verified Rabin's mutual exclusion algorithm with nested loops (based on Hurd, 2005):

```
while (1 < i) invariant I_1 {
  n := i;
  while (0 < n) invariant I_2 {
    {d := 0} [0.5] {d := 1};
    i := i − d;
    n := n − 1
  }
}
```

Experiments

Tested against *kipro2*, a tool for probabilistic k-induction (Batz et al. 2021).

- Programs that consist of only one loop
- Competetive on benchmark set for wp

  (e.g. geometric loop, uniform number generation, Rabin's mutual exclusion)

Verified Rabin's mutual exclusion algorithm with nested loops (based on Hurd, 2005):

```
while (1 < i) invariant I_1 {
  n := i;
  while (0 < n) invariant I_2 {
    {d := 0} [0.5] {d := 1};
    i := i - d;
    n := n - 1
  }
}
```

$$B = n \leq 5 \wedge i \leq 5$$

$$I_1 = [B] \cdot ([1 = i] + [1 < i] \cdot \frac{2}{3})$$

$$I_2 = [B] \cdot [0 \leq n \wedge n \leq i] \cdot (\frac{2}{3} \cdot I_3 + I_4)$$

$$I_3 = 1 - ([i = n] \cdot (n+1) \cdot \frac{1}{2^n} + [i = n+1] \cdot \frac{1}{2^n})$$

$$I_4 = [i = n] \cdot n \cdot \frac{1}{2^n} + [i = n+1] \cdot \frac{1}{2^n}$$

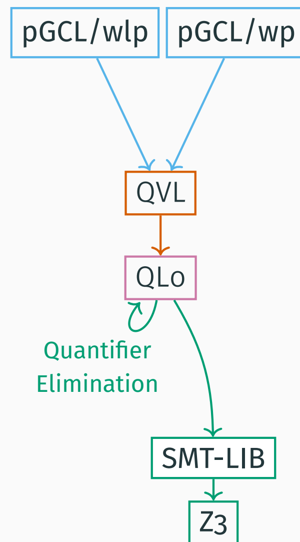
$$\frac{1}{2^n} = [n = 0] \cdot 1 + [n = 2] \cdot \frac{1}{4} + \ldots + [n = 4] \cdot \frac{1}{32}$$

Conclusion

Conclusion

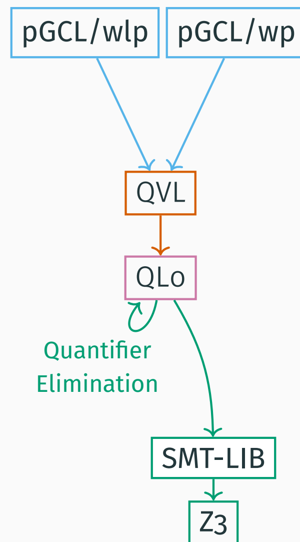
Design and implementation of a deductive verification infrastructure for probabilistic programs.

- Architecture:
  - A quantitative intermediate verification language – *QVL*
  - A quantitative assertion language – *QLo*
  - QLo validity checking
- Case study: Encodings of pGCL (w.r.t. wlp and wp) into QVL
- Implementation: ≈ 7000 LOC Rust & Python



Design and implementation of a deductive verification infrastructure for probabilistic programs.

- Architecture:
    - A quantitative intermediate verification language – *QVL*
    - A quantitative assertion language – *QLo*
    - QLo validity checking
- Case study: Encodings of pGCL (w.r.t. wlp and wp) into QVL
- Implementation: ≈ 7000 LOC Rust & Python

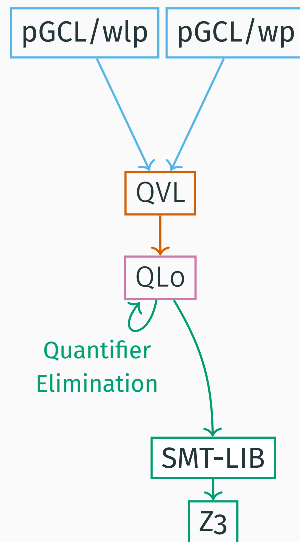


Future work:

- More encodings

Design and implementation of a deductive verification infrastructure for probabilistic programs.

- Architecture:
  - A quantitative intermediate verification language – *QVL*
  - A quantitative assertion language – *QLo*
  - QLo validity checking
- Case study: Encodings of pGCL (w.r.t. wlp and wp) into QVL
- Implementation: ≈ 7000 LOC Rust & Python



Future work:

- More encodings
- User-defined datatypes, axioms, procedures...

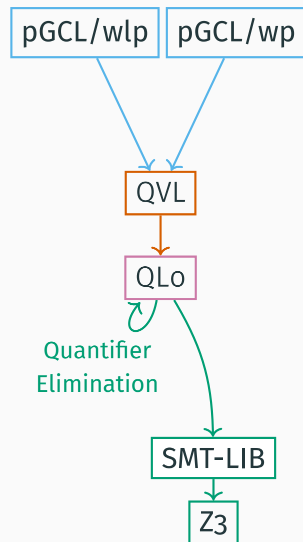
Conclusion

Design and implementation of a deductive verification infrastructure for probabilistic programs.

- Architecture:
  - A quantitative intermediate verification language – *QVL*
  - A quantitative assertion language – *QLo*
  - QLo validity checking
- Case study: Encodings of pGCL (w.r.t. wlp and wp) into QVL
- Implementation: ≈ 7000 LOC Rust & Python

Future work:

- More encodings
- User-defined datatypes, axioms, procedures...
- Generalization to other domains



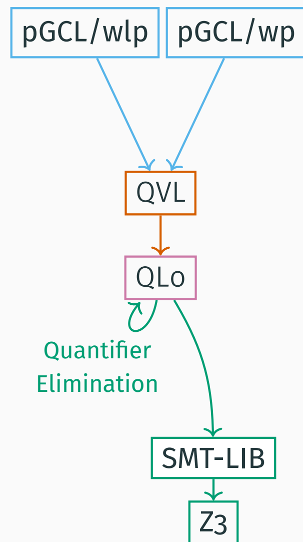
Conclusion

Design and implementation of a deductive verification infrastructure for probabilistic programs.

- Architecture:
  - A quantitative intermediate verification language – *QVL*
  - A quantitative assertion language – *QLo*
  - QLo validity checking
- Case study: Encodings of pGCL (w.r.t. wlp and wp) into QVL
- Implementation: ≈ 7000 LOC Rust & Python

Future work:

- More encodings
- User-defined datatypes, axioms, procedures...
- Generalization to other domains
- Error reporting



Design and implementation of a deductive verification infrastructure for probabilistic programs.

- Architecture:
  - A quantitative intermediate verification language – *QVL*
  - A quantitative assertion language – *QLo*
  - QLo validity checking
- Case study: Encodings of pGCL (w.r.t. wlp and wp) into QVL
- Implementation: ≈ 7000 LOC Rust & Python

Future work:

- More encodings
- User-defined datatypes, axioms, procedures...
- Generalization to other domains
- Error reporting
- Optimizations

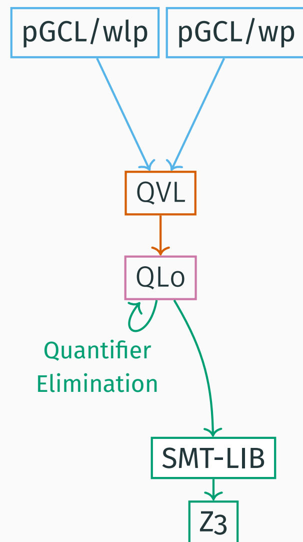


Table of Contents

Probabilistic Programs

Intermediate Verification Languages

QLo and QVL

Encoding pGCL

Implementation

Conclusion

Appendix

Gödel Implication vs. Hard Implication

$$X \to Y = \lambda\sigma. \begin{cases} \infty, & \text{if } X(\sigma) \leq Y(\sigma) \\ Y(\sigma), & \text{else} \end{cases} \qquad X \searrow Y = \lambda\sigma. \begin{cases} \infty, & \text{if } X(\sigma) \leq Y(\sigma) \\ 0, & \text{else} \end{cases}$$

All our pGCL encodings can be defined with `compare` (hard implications) as well.

Hard implication is syntactic sugar: $X \searrow Y \equiv \neg\neg(X \to Y)$ $\quad$ $(\neg X \equiv X \to 0)$

Only the Gödel implication makes QLo a proper intuitionistic logic.

Verifies iff $x+1 \sqsubseteq x+2$**:**

`down assume` $x+1$;

`down assert` $x+2$

Verifies iff $x+1 \sqsubseteq x+2$**:**

`down compare` $x+1$;

`down assert` $x+2$

Gödel Implication vs. Hard Implication

$$X \rightarrow Y = \lambda\sigma. \begin{cases} \infty, & \text{if } X(\sigma) \leq Y(\sigma) \\ Y(\sigma), & \text{else} \end{cases}$$

$$X \searrow Y = \lambda\sigma. \begin{cases} \infty, & \text{if } X(\sigma) \leq Y(\sigma) \\ 0, & \text{else} \end{cases}$$

All our pGCL encodings can be defined with `compare` (hard implications) as well.

Hard implication is syntactic sugar: $X \searrow Y \equiv \neg\neg(X \rightarrow Y)$ $\quad (\neg X \equiv X \rightarrow 0)$

Only the Gödel implication makes QLo a proper intuitionistic logic.

Verifies iff $x + 1 \sqsubseteq x + 2$:

```
down assume x + 1;
down assume ∞;
down assert x + 2
```

Verifies iff $x + 1 \sqsubseteq \infty \wedge \infty \leq x + 2$:

```
down compare x + 1;
down compare ∞;
down assert x + 2
```

Dual Operators

Let $X, Y \in \mathbb{E}$.

Definition

Gödel implication:

$$X \rightarrow Y = \lambda \sigma . \begin{cases} \infty, & \text{if } X(\sigma) \leq Y(\sigma) \\ Y(\sigma), & \text{else} \end{cases}$$

Dual Operators

Let $X, Y \in \mathbb{E}$.

Definition

Gödel implication:

$$X \to Y = \lambda\sigma.\begin{cases}\infty, & \text{if } X(\sigma) \leq Y(\sigma)\\ Y(\sigma), & \text{else}\end{cases}$$

Definition

Gödel co-implication:

$$X \leftarrow Y = \lambda\sigma.\begin{cases}0, & \text{if } X(\sigma) \geq Y(\sigma)\\ Y(\sigma), & \text{else}\end{cases}$$

Dual Operators

Let $X, Y \in \mathbb{E}$.

Definition

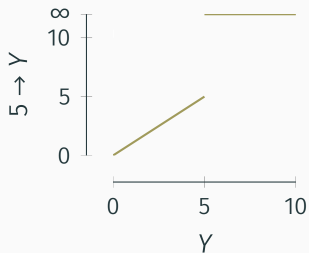
Gödel implication:

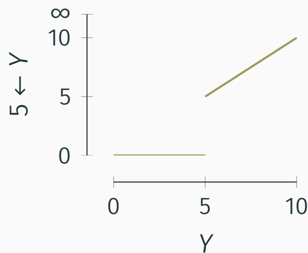
$$X \rightarrow Y = \lambda\sigma.\begin{cases}\infty, & \text{if } X(\sigma) \leq Y(\sigma)\\ Y(\sigma), & \text{else}\end{cases}$$

Definition

Gödel co-implication:

$$X \leftarrow Y = \lambda\sigma.\begin{cases}0, & \text{if } X(\sigma) \geq Y(\sigma)\\ Y(\sigma), & \text{else}\end{cases}$$





QVL Semantics

S	$\mathrm{vc}[\![S]\!](\varphi)$
`skip`	φ
x `:≈` μ	$E(x,\mu,\varphi)$
S_1`;` S_2	$\mathrm{vc}[\![S_1]\!](\mathrm{vc}[\![S_2]\!](\varphi))$
`down havoc` x	$\bigsqcap_x \ \varphi$
`down assert` ψ	$\psi \sqcap \varphi$
`down assume` ψ	$\psi \to \varphi$
`down compare` ψ	$\psi \searrow \varphi$
`if` $(\sqcap)$ $\{S_1\}$ `else` $\{S_2\}$	$\mathrm{vc}[\![S_1]\!](\varphi) \sqcap \mathrm{vc}[\![S_2]\!](\varphi)$
`up havoc` x	$\bigsqcup_x \ \varphi$
`up assert` ψ	$\psi \sqcup \varphi$
`up assume` ψ	$\psi \leftarrow \varphi$
`up compare` ψ	$\psi \nwarrow \varphi$
`if` $(\sqcup)$ $\{S_1\}$ `else` $\{S_2\}$	$\mathrm{vc}[\![S_1]\!](\varphi) \sqcup \mathrm{vc}[\![S_2]\!](\varphi)$
`down negate`	$\neg\varphi$
`up negate`	$\sim\varphi$

$$(X \to Y)(\sigma) = \begin{cases} \infty, & \text{if } X(\sigma) \sqsubseteq Y(\sigma) \\ Y(\sigma), & \text{else} \end{cases}$$

$$(X \searrow Y)(\sigma) = \begin{cases} \infty, & \text{if } X(\sigma) \sqsubseteq Y(\sigma) \\ 0, & \text{else} \end{cases}$$

$$(\neg X)(\sigma) = \begin{cases} \infty, & \text{if } X(\sigma) = 0 \\ 0, & \text{else} \end{cases}$$

$$(X \leftarrow Y)(\sigma) = \begin{cases} 0, & \text{if } X(\sigma) \sqsupseteq Y(\sigma) \\ Y(\sigma), & \text{else} \end{cases}$$

$$(X \nwarrow Y)(\sigma) = \begin{cases} 0, & \text{if } X(\sigma) \sqsupseteq Y(\sigma) \\ \infty, & \text{else} \end{cases}$$

$$(\sim X)(\sigma) = \begin{cases} 0, & \text{if } X(\sigma) = \infty \\ \infty, & \text{else} \end{cases}$$

QLo Semantics

$\rho \in \mathrm{QLo}$	$[\![\rho]\!]$
$a \in \mathrm{ArithExp}$	$[\![a]\!]$
$\varphi + \psi$	$[\![\varphi]\!] + [\![\psi]\!]$
$\varphi \cdot \psi$	$[\![\varphi]\!] \cdot [\![\psi]\!]$
$[b]$	$[b]$
$?(b)$	$?(b)$
$\prod_x \varphi$	$\inf \left\{ [\![\varphi[x \mapsto v]]\!] \;\middle|\; v \in \mathbb{Q}_{\geq 0} \right\}$
$\varphi \sqcap \psi$	$[\![\varphi]\!] \sqcap [\![\psi]\!]$
$\varphi \rightarrow \psi$	$[\![\varphi]\!] \rightarrow [\![\psi]\!]$
$\varphi \searrow \psi$	$[\![\varphi]\!] \searrow [\![\psi]\!]$
$\coprod_x \varphi$	$\sup \left\{ [\![\varphi[x \mapsto v]]\!] \;\middle|\; v \in \mathbb{Q}_{\geq 0} \right\}$
$\varphi \sqcup \psi$	$[\![\varphi]\!] \sqcup [\![\psi]\!]$
$\varphi \leftarrow \psi$	$[\![\varphi]\!] \leftarrow [\![\psi]\!]$
$\varphi \nwarrow \psi$	$[\![\varphi]\!] \nwarrow [\![\psi]\!]$
$\neg\varphi$	$\neg[\![\varphi]\!]$
$\sim\varphi$	$\sim[\![\phi]\!]$

$$(X \rightarrow Y)(\sigma) = \begin{cases} \infty, & \text{if } X(\sigma) \sqsubseteq Y(\sigma) \\ Y(\sigma), & \text{else} \end{cases}$$

$$(X \searrow Y)(\sigma) = \begin{cases} \infty, & \text{if } X(\sigma) \sqsubseteq Y(\sigma) \\ 0, & \text{else} \end{cases}$$

$$(\neg X)(\sigma) = \begin{cases} \infty, & \text{if } X(\sigma) = 0 \\ 0, & \text{else} \end{cases}$$

$$(X \leftarrow Y)(\sigma) = \begin{cases} 0, & \text{if } X(\sigma) \sqsupseteq Y(\sigma) \\ Y(\sigma), & \text{else} \end{cases}$$

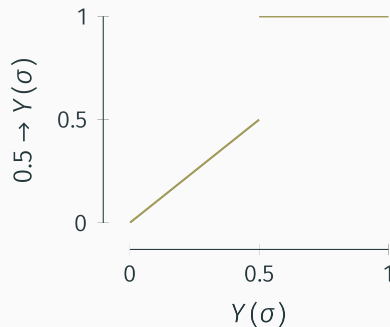
$$(X \nwarrow Y)(\sigma) = \begin{cases} 0, & \text{if } X(\sigma) \sqsupseteq Y(\sigma) \\ \infty, & \text{else} \end{cases}$$

$$(\sim X)(\sigma) = \begin{cases} 0, & \text{if } X(\sigma) = \infty \\ \infty, & \text{else} \end{cases}$$

Gödel Logic

Gödel implication:

$$X \to Y = \lambda \sigma . \begin{cases} 1, & \text{if } X(\sigma) \leq Y(\sigma) \\ Y(\sigma), & \text{else} \end{cases}$$



$(\mathbb{E}_{\leq 1}, \sqsubseteq, \sqcap, \to)$ forms a *Heyting algebra*.

Theorem

For all $X, Y, Z \in \mathbb{E}_{\leq 1}$.

1. $X \to X = 1$,
2. $X \sqcap (X \to Y) = X \sqcap Y$,
3. $Y \sqcap (X \to Y) = Y$,
4. $X \to (Y \sqcap Z) = (X \to Y) \sqcap (X \to Z)$.