

Symbolic Execution

A 90-minute PhD Course Lecture

PhD Course

October 12, 2026

Motivation

Why Symbolic Execution?

- Hoare logic proves a contract by composing local proof obligations.
- Symbolic execution computes with expressions for unknown inputs.
- One symbolic path can describe infinitely many concrete executions.
- Can be used to prove contracts, but also to help test input generation

Motivation

Running Example: Integer Division by Subtraction

```
q := 0;  
r := x;  
while y <= r do  
  r := r - y;  
  q := q + 1
```

Pre/postconditions

$$P \equiv x \geq 0 \wedge y > 0, \quad Q \equiv x = q \cdot y + r \wedge 0 \leq r < y.$$

- Can we establish the contract?
- Which inputs take exactly two iterations?

- Define symbolic states and execution rules for the IMP language.
- Derive path constraints and concrete counterexamples.
- Prove the connection with concrete execution.
- Relate path exploration to invariants, strongest postconditions, and wlp.
- Compare underapproximation, exact expression naming, and overapproximation.

We assume familiarity with denotational and operational semantics.

Symbolic States and Rules

IMP and the Semantic Setting

$$\begin{aligned} a &::= n \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \cdot a_2 \\ b &::= \text{true} \mid \text{false} \mid a_1 = a_2 \mid a_1 \leq a_2 \mid \neg b \mid b_1 \wedge b_2 \\ c &::= \text{skip} \mid x := a \mid c_1; c_2 \\ &\quad \mid \text{if } b \text{ then } c_1 \text{ else } c_2 \mid \text{while } b \text{ do } c. \end{aligned}$$

- A concrete state $\sigma \in \Sigma$ maps variables to integers.
- Expressions are total. Arithmetic uses $\mathbb{Z}$, with no overflow.
- Fix a finite variable set V containing all program variables.

Symbolic States and Rules

Concrete Small-Step Semantics (recap)

$$\langle x := a, \sigma \rangle \longrightarrow \langle \text{skip}, \sigma[x \mapsto \llbracket a \rrbracket_\sigma] \rangle.$$

$$\frac{\langle c_1, \sigma \rangle \longrightarrow \langle c'_1, \sigma' \rangle}{\langle c_1; c_2, \sigma \rangle \longrightarrow \langle c'_1; c_2, \sigma' \rangle} \quad \langle \text{skip}; c_2, \sigma \rangle \longrightarrow \langle c_2, \sigma \rangle.$$

$$\frac{\llbracket b \rrbracket_\sigma = \text{true}}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \longrightarrow \langle c_1, \sigma \rangle} \text{ If-T} \quad \frac{\llbracket b \rrbracket_\sigma = \text{false}}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \sigma \rangle \longrightarrow \langle c_2, \sigma \rangle} \text{ If-F}$$

$$\langle \text{while } b \text{ do } d, \sigma \rangle \longrightarrow \langle \text{if } b \text{ then } (d; \text{while } b \text{ do } d) \text{ else skip}, \sigma \rangle.$$

The terminal command is skip, which has no transition.

Symbolic States and Rules

Properties of the semantics

- Semantics is deterministic
- This semantics agrees with the previous big-step semantics on terminating runs.

Formally:

$$\begin{aligned} (\exists n \in \mathbb{N}. \langle c, \sigma \rangle \longrightarrow^n \langle \text{skip}, \sigma' \rangle) \\ \Leftrightarrow \\ \langle c, \sigma \rangle \Downarrow \sigma' \end{aligned}$$

Symbolic States and Rules

Big-Step Implies Small-Step Termination

Induct on the derivation of $\langle c, \sigma \rangle \Downarrow \sigma'$.

- Base cases: skip: zero steps. Assignment: one step.
- Sequencing: Let $\langle c_1; c_2, \sigma \rangle \Downarrow \sigma'$ by $\langle c_1, \sigma \rangle \Downarrow \sigma''$ and $\langle c_2, \sigma'' \rangle \Downarrow \sigma'$.
By induction exist n_1 and n_2 such that $\langle c_1, \sigma \rangle \longrightarrow^{n_1} \langle \text{skip}, \sigma'' \rangle$ and $\langle c_2, \sigma'' \rangle \longrightarrow^{n_2} \langle \text{skip}, \sigma' \rangle$.
Therefore $\langle c_1; c_2, \sigma \rangle \longrightarrow^{n_1} \langle \text{skip}; c_2, \sigma'' \rangle \longrightarrow \langle c_2, \sigma'' \rangle \longrightarrow^{n_2} \langle \text{skip}, \sigma' \rangle$.
- A conditional takes one step to the selected branch, then uses the induction hypothesis.
- For the true loop rule, let $w \equiv \text{while } b \text{ do } d$. Its premises give executions of d and w , so

$$\begin{aligned} \langle w, \sigma \rangle &\longrightarrow \langle \text{if } b \text{ then } (d; w) \text{ else skip}, \sigma \rangle \\ &\longrightarrow \langle d; w, \sigma \rangle \longrightarrow^* \langle \text{skip}; w, \sigma_1 \rangle && \text{by ind} \\ &\longrightarrow \langle w, \sigma_1 \rangle \longrightarrow^* \langle \text{skip}, \sigma' \rangle && \text{by ind.} \end{aligned}$$

For a false guard, unfolding and branch selection give two steps to skip.

Symbolic States and Rules

Small-Step Termination Implies Big-Step

First prove a one-step expansion lemma:

$$\langle c, \sigma \rangle \longrightarrow \langle c_1, \sigma_1 \rangle \quad \wedge \quad \langle c_1, \sigma_1 \rangle \Downarrow \sigma' \quad \Longrightarrow \quad \langle c, \sigma \rangle \Downarrow \sigma'.$$

Induct on the small-step rule derivation and invert the given big-step derivation.

- Sequencing uses the induction hypothesis for the step of its first command.
- Loop unfolding uses inversion of the conditional and sequencing rules, then the corresponding big-step while rule.

Now induct on the length n of a terminating small-step execution.

- $n = 0$: the initial command is skip and $\sigma = \sigma'$.
- $n > 0$: apply the induction hypothesis to the remaining $n - 1$ steps, then the expansion lemma to the first step.

Symbolic States and Rules

Symbolic Stores and Input Symbols

Choose distinct input symbols α_x for $x \in V$, disjoint from program variables.

$\text{Term}(\{\alpha_x \mid x \in V\})$ is the set of arithmetic expressions built from integer constants and these input symbols using $+$, $-$, $\cdot$. Examples: $\alpha_x + \alpha_y$, $\alpha_x - 1$, and $2 \cdot \alpha_y$.

$$\hat{\sigma} : V \rightarrow \text{Term}(\{\alpha_x \mid x \in V\}), \quad \hat{\sigma}_0(x) = \alpha_x.$$

A symbolic configuration is

$$\langle c, \hat{\sigma}, \varphi \rangle.$$

- c : the command still to execute.
- $\hat{\sigma}$: current values as expressions over input symbols.
- φ : a formula over input symbols, the path condition.

Input symbols denote initial values and never change under assignment.

Symbolic States and Rules

Symbolic Evaluation

Evaluate syntax in a symbolic store by substitution:

$$\begin{aligned}\llbracket n \rrbracket_{\hat{\sigma}} &= n, & \llbracket x \rrbracket_{\hat{\sigma}} &= \hat{\sigma}(x), \\ \llbracket a_1 + a_2 \rrbracket_{\hat{\sigma}} &= \llbracket a_1 \rrbracket_{\hat{\sigma}} + \llbracket a_2 \rrbracket_{\hat{\sigma}}, \\ \llbracket a_1 \leq a_2 \rrbracket_{\hat{\sigma}} &= (\llbracket a_1 \rrbracket_{\hat{\sigma}} \leq \llbracket a_2 \rrbracket_{\hat{\sigma}}).\end{aligned}$$

Other operators are analogous. Write $P[\hat{\sigma}]$ for simultaneous substitution for the *free* program-variable occurrences in an assertion.

Capture avoidance keeps a quantifier from binding an input symbol introduced by substitution, which would change the assertion's meaning. Here we choose input symbols fresh for all assertion binders, so capture cannot occur.

$$\hat{\sigma}(x) = \alpha_x + 1 \quad \Rightarrow \quad (x > 5)[\hat{\sigma}] \equiv \alpha_x + 1 > 5.$$

Initialize a precondition P with $\varphi_0 = P[\hat{\sigma}_0]$.

Symbolic States and Rules

Assignment Works Forwards

$$\frac{}{\langle x := a, \hat{\sigma}, \varphi \rangle \longrightarrow_s \langle \text{skip}, \hat{\sigma}[x \mapsto \llbracket a \rrbracket_{\hat{\sigma}}], \varphi \rangle} \text{S-Assign}$$

```
x := x + 1;  
y := x
```

$$\begin{aligned}\hat{\sigma}_0 &= [x \mapsto \alpha_x, y \mapsto \alpha_y], \\ \hat{\sigma}_1 &= \hat{\sigma}_0[x \mapsto \alpha_x + 1], \\ \hat{\sigma}_2 &= \hat{\sigma}_1[y \mapsto \alpha_x + 1].\end{aligned}$$

Evaluate the right-hand side in the old store. Assignment leaves φ unchanged.

Symbolic States and Rules

Sequencing and Loop Unfolding

$$\frac{\langle c_1, \hat{\sigma}, \varphi \rangle \longrightarrow_s \langle c'_1, \hat{\sigma}', \varphi' \rangle}{\langle c_1; c_2, \hat{\sigma}, \varphi \rangle \longrightarrow_s \langle c'_1; c_2, \hat{\sigma}', \varphi' \rangle} \text{ S-Seq}$$
$$\langle \text{skip}; c_2, \hat{\sigma}, \varphi \rangle \longrightarrow_s \langle c_2, \hat{\sigma}, \varphi \rangle.$$

Let $w \equiv \text{while } b \text{ do } d$. Then

$$\langle w, \hat{\sigma}, \varphi \rangle \longrightarrow_s \langle \text{if } b \text{ then } (d; w) \text{ else skip}, \hat{\sigma}, \varphi \rangle.$$

skip has no transition. Loop execution follows by repeated unfolding.

Symbolic States and Rules

Conditionals Fork the Path Condition

Let $\beta = \llbracket b \rrbracket_{\hat{\sigma}}$.

$$\frac{}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \hat{\sigma}, \varphi \rangle \longrightarrow_s \langle c_1, \hat{\sigma}, \varphi \wedge \beta \rangle} \text{S-If-T}$$

$$\frac{}{\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \hat{\sigma}, \varphi \rangle \longrightarrow_s \langle c_2, \hat{\sigma}, \varphi \wedge \neg\beta \rangle} \text{S-If-F}$$

- The semantic rules generate both children, even if one is infeasible.
- An implementation may discard a child after proving its condition unsatisfiable.

This separation keeps the semantics independent of a solver.

Paths and Counterexamples

Worked Example: A Faulty Absolute Value

```
if x <= 0 then
  y := 0 - x
else
  y := x - 1
```

Intended postcondition:

$$Q \equiv y \geq 0 \wedge (y = x \vee y = -x).$$

Starting with $x = \alpha_x$ and $\varphi = \text{true}$:

path	φ	$\hat{\sigma}(y)$
T	$\alpha_x \leq 0$	$-\alpha_x$
F	$\alpha_x > 0$	$\alpha_x - 1$

In both leaves, $\hat{\sigma}(x) = \alpha_x$.

Paths and Counterexamples

A Counterexample Is a Model

For a terminal path $(\hat{\sigma}, \varphi)$, query

$$\text{SAT}(\varphi \wedge \neg Q[\hat{\sigma}]).$$

On the false branch:

$$\alpha_x > 0 \wedge \neg(\alpha_x - 1 \geq 0 \wedge (\alpha_x - 1 = \alpha_x \vee \alpha_x - 1 = -\alpha_x)).$$

- A model is $\alpha_x = 1$.
- Replay from $x = 1$: the program returns $y = 0$, violating Q .
- The true branch has an unsatisfiable violation condition.

A model assigns the initial inputs, not the final program variables.

Paths and Counterexamples

Exercise 1: Feasible Paths

```
if x <= 0 then y := 0 - x else y := x;  
if y < 0 then z := 1 else z := 0
```

Start with $\varphi = \text{true}$ and distinct input symbols.

- 1 Write the conditions for the four syntactic branch sequences.
- 2 Which paths are feasible over $\mathbb{Z}$?
- 3 Prove $\{\text{true}\} \subset \{z = 0\}$ using the feasible leaves.

Semantic Correspondence

Concretization of a Symbolic State

A valuation ν maps input symbols to integers. Define

$$(\nu\hat{\sigma})(x) = \llbracket \hat{\sigma}(x) \rrbracket_{\nu}.$$

The represented set of concrete stores is

$$\gamma(\hat{\sigma}, \varphi) = \{\nu\hat{\sigma} \mid \nu \models \varphi\}.$$

For $\hat{\sigma}(x) = \alpha_x$, $\hat{\sigma}(y) = \alpha_x + 1$, and $\varphi \equiv \alpha_x \geq 0$:

$$\gamma(\hat{\sigma}, \varphi) = \{\sigma \mid \sigma(x) \geq 0 \wedge \sigma(y) = \sigma(x) + 1\}$$

when $V = \{x, y\}$. The unused symbol α_y is unconstrained.

Semantic Correspondence

Evaluation Commutes with Concretization

Lemma, for every valuation ν :

$$\llbracket a \rrbracket_{\nu\hat{\sigma}} = \llbracket \llbracket a \rrbracket_{\hat{\sigma}} \rrbracket_{\nu}.$$

Likewise for Boolean expressions, and

$$P(\nu\hat{\sigma}) \iff \nu \models P[\hat{\sigma}].$$

Proof: structural induction on expressions and formulas, with capture-avoiding substitution.
Assignment case needed later:

$$\nu(\hat{\sigma}[x \mapsto \llbracket a \rrbracket_{\hat{\sigma}}]) = (\nu\hat{\sigma})[x \mapsto \llbracket a \rrbracket_{\nu\hat{\sigma}}].$$

Semantic Correspondence

One-Step Soundness

Theorem: if

$$\langle c, \hat{\sigma}, \varphi \rangle \longrightarrow_s \langle c', \hat{\sigma}', \varphi' \rangle \quad \text{and} \quad \nu \models \varphi',$$

then

$$\nu \models \varphi \quad \text{and} \quad \langle c, \nu \hat{\sigma} \rangle \longrightarrow \langle c', \nu \hat{\sigma}' \rangle.$$

- Induct on the symbolic transition derivation.
- Assignment uses the evaluation lemma.
- Sequencing uses the induction hypothesis. Unfolding is immediate.

Semantic Correspondence

If Soundness

- Let $\langle \text{if } b \text{ then } c_1 \text{ else } c_2, \hat{\sigma}, \varphi \rangle \longrightarrow_s \langle c_1, \hat{\sigma}, \varphi \wedge \llbracket b \rrbracket_{\hat{\sigma}} \rangle$ by S-If-T and $\nu \models \varphi \wedge \llbracket b \rrbracket_{\hat{\sigma}}$.
- In the true branch, $\nu \models \varphi \wedge \llbracket b \rrbracket_{\hat{\sigma}}$, so the concrete guard is true.

Semantic Correspondence

One-Step Coverage

Conversely, suppose $\nu \models \varphi$ and

$$\langle c, \nu \hat{\sigma} \rangle \longrightarrow \langle c', \sigma' \rangle.$$

There exist $\hat{\sigma}', \varphi'$ such that

$$\langle c, \hat{\sigma}, \varphi \rangle \longrightarrow_s \langle c', \hat{\sigma}', \varphi' \rangle, \quad \nu \models \varphi', \quad \sigma' = \nu \hat{\sigma}'.$$

Proof: induction on the concrete transition derivation.

At a conditional, the concrete guard selects a symbolic child whose condition ν satisfies.

Thus proving a child unsatisfiable cannot remove a concrete execution.

Semantic Correspondence

Finite-Path Correspondence

Start with $s_0 = \langle c, \hat{\sigma}_0, P[\hat{\sigma}_0] \rangle$.

Soundness: if

$$s_0 \longrightarrow_s^* \langle \text{skip}, \hat{\sigma}_f, \varphi_f \rangle, \quad \nu \models \varphi_f,$$

then $P(\nu \hat{\sigma}_0)$ and

$$\langle c, \nu \hat{\sigma}_0 \rangle \Downarrow \nu \hat{\sigma}_f.$$

Coverage: every terminating concrete execution from a P -state is represented by such a finite symbolic path and its initial valuation.

Both directions follow by induction on finite transition sequences.

Semantic Correspondence

What the Theorem Gives an Analyzer

- Exact constraints plus a satisfying model give a real execution in our IMP semantics.
- Coverage is a property of the full symbolic tree, not of an arbitrary finite search.
- To establish a postcondition, account for every terminating path.
- To establish total correctness, also prove termination.

These claims assume faithful expression semantics and sound solver answers.

An infinite symbolic tree may represent a program that terminates on every input.

Loops and Program Logic

Division: The First Iterations

Write $X = \alpha_x$, $Y = \alpha_y$, and $P_0 \equiv X \geq 0 \wedge Y > 0$. After initialization, $q = 0$ and $r = X$ symbolically.

iterations	$\hat{\sigma}(q)$	$\hat{\sigma}(r)$	condition at loop head
0	0	X	P_0
1	1	$X - Y$	$P_0 \wedge Y \leq X$
2	2	$X - 2Y$	$P_0 \wedge Y \leq X \wedge Y \leq X - Y$

Exit after exactly two iterations adds $X - 2Y < Y$, hence

$$P_0 \wedge 2Y \leq X < 3Y.$$

For example, $X = 7, Y = 3$ yields $q = 2, r = 1$.

Loops and Program Logic

Division: Every Finite Exit Path

For each fixed $k \in \mathbb{N}$, the leaf after exactly k iterations has

$$\hat{\sigma}_k(q) = k, \quad \hat{\sigma}_k(r) = X - kY,$$

$$\varphi_k \equiv P_0 \wedge kY \leq X < (k+1)Y.$$

Therefore

$$\varphi_k \Rightarrow (X = kY + (X - kY) \wedge 0 \leq X - kY < Y).$$

- Every finite exit path satisfies Q .
- There are infinitely many feasible exit paths as the input varies.
- An induction on k justifies the formula for all paths.

Loops and Program Logic

An Invariant Summarizes the Loop Heads

Reuse the invariant from the Hoare logic lecture:

$$I \equiv x = q \cdot y + r \wedge r \geq 0 \wedge y > 0.$$

The obligations are

$$\{P\} \ q := 0; \ r := x \ \{I\}, \quad \{I \wedge y \leq r\} \ r := r - y; \ q := q + 1 \ \{I\},$$

$$I \wedge \neg(y \leq r) \Rightarrow Q.$$

Symbolic execution can check the loop body obligation from an arbitrary symbolic I -state.
A ranking function $r \in \mathbb{N}$ also proves termination: under the guard, $r - y < r$.

Loops and Program Logic

Division: Symbolically Execute an Arbitrary Iteration

Choose fresh integer symbols X, Y, U, R for the values at an arbitrary loop head:

$$\hat{\sigma}_h = [x \mapsto X, y \mapsto Y, q \mapsto U, r \mapsto R].$$

Assume the invariant and the true guard:

$$\psi \equiv I[\hat{\sigma}_h] \wedge Y \leq R \equiv X = UY + R \wedge R \geq 0 \wedge Y > 0 \wedge Y \leq R.$$

Let $\hat{\sigma}_1 = \hat{\sigma}_h[r \mapsto R - Y]$ and $\hat{\sigma}_2 = \hat{\sigma}_1[q \mapsto U + 1]$. The symbolic rules give

$$\begin{aligned} \langle r := r - y; q := q + 1, \hat{\sigma}_h, \psi \rangle &\longrightarrow_s \langle \text{skip}; q := q + 1, \hat{\sigma}_1, \psi \rangle \\ &\longrightarrow_s \langle q := q + 1, \hat{\sigma}_1, \psi \rangle \\ &\longrightarrow_s \langle \text{skip}, \hat{\sigma}_2, \psi \rangle. \end{aligned}$$

The symbols denote loop-entry values, not necessarily the program's initial inputs.

Loops and Program Logic

Division: The Invariant Is Preserved

At the symbolic leaf, substitute the final store into the invariant:

$$I[\hat{\sigma}_2] \equiv X = (U + 1)Y + (R - Y) \wedge R - Y \geq 0 \wedge Y > 0.$$

Every valuation satisfying ψ satisfies all three conjuncts:

$$\begin{array}{ll} (U + 1)Y + (R - Y) = UY + R = X & \text{by } X = UY + R, \\ R - Y \geq 0 & \text{by } Y \leq R, \\ Y > 0 & \text{by } \psi. \end{array}$$

$$\psi \Rightarrow I[\hat{\sigma}_2], \quad \text{UNSAT}(\psi \wedge \neg I[\hat{\sigma}_2]).$$

Every concrete $I \wedge y \leq r$ -state is represented by $(\hat{\sigma}_h, \psi)$. By finite-path correspondence, this proves

$$\{I \wedge y \leq r\} \ r := r - y; \ q := q + 1 \ \{I\}.$$

Loops and Program Logic

Division: The Ranking Function Decreases

Take $\rho(\sigma) = \sigma(r)$. The same symbolic execution gives

$$\rho_{\text{before}} = R, \quad \rho_{\text{after}} = R - Y.$$

Under ψ , the rank stays nonnegative and strictly decreases:

$$\underbrace{0 \leq R - Y}_{Y \leq R} < \underbrace{R}_{Y > 0 \text{ gives } R - Y < R}.$$

$$\psi \Rightarrow 0 \leq \rho_{\text{after}} < \rho_{\text{before}}, \quad \text{UNSAT}(\psi \wedge \neg(0 \leq R - Y < R)).$$

The invariant gives $\rho \in \mathbb{N}$ at every loop head. Each body execution terminates and decreases ρ by at least one, since Y is a positive integer.

After initialization establishes I , preservation and well-foundedness of $<$ on $\mathbb{N}$ rule out infinitely many iterations.

Loops and Program Logic

Terminal Paths and Strongest Postconditions

Let L be all terminal symbolic paths from $(c, \hat{\sigma}_0, P[\hat{\sigma}_0])$. We have

$$\{\sigma' \mid \text{sp}(P, c)(\sigma')\} = \{\sigma' \mid P(\sigma) \wedge \langle c, \sigma \rangle \longrightarrow^* \langle \text{skip}, \sigma' \rangle\} = \bigcup_{\ell \in L} \gamma(\hat{\sigma}_\ell, \varphi_\ell).$$

For finite L , this has the formula

$$\text{sp}(P, c)(\sigma') \equiv \bigvee_{\ell \in L} \exists \nu. \varphi_\ell(\nu) \wedge \bigwedge_{x \in V} \sigma'(x) = \llbracket \hat{\sigma}_\ell(x) \rrbracket_\nu.$$

Here $\exists \nu$ abbreviates quantification over the finitely many input symbols.

Loops and Program Logic

Hoare Validity as Path Obligations

For the same full set L :

$$\models_{\text{par}} \{P\} c \{Q\} \iff \forall \ell \in L. \text{UNSAT}(\varphi_\ell \wedge \neg Q[\hat{\sigma}_\ell]).$$

This is also equivalent to

$$P \Rightarrow \text{wlp}(c, Q).$$

- Backward reasoning builds $\text{wlp}(c, Q)$.
- Forward reasoning builds stores and conditions, then checks Q .
- If $L = \emptyset$, the path obligations are vacuous, as partial correctness requires.

Enumerating some leaves does not establish the universal quantification.

Approximation and Simplification

Search and Path Explosion

n independent binary branches can generate 2^n feasible paths.

- Depth-first search uses a small frontier but may remain on an unbounded branch.
- Breadth-first search reaches every finite-depth node in a finitely branching tree, assuming each processing step finishes.
- A timeout or depth bound leaves unexplored paths.

A fair search with terminating, complete feasibility checks eventually reaches any finite counterexample.

Approximation and Simplification

Exact State Merging

At the same remaining command, suppose two path states use the same input symbols and $\varphi_1 \wedge \varphi_2$ is unsatisfiable.

$$\varphi = \varphi_1 \vee \varphi_2, \quad \hat{\sigma}(x) = \text{ite}(\varphi_1, \hat{\sigma}_1(x), \hat{\sigma}_2(x)).$$

Then

$$\gamma(\hat{\sigma}, \varphi) = \gamma(\hat{\sigma}_1, \varphi_1) \cup \gamma(\hat{\sigma}_2, \varphi_2).$$

- Merging saves explicit states but introduces conditional terms.
- More compact exploration may produce harder solver queries.

Approximation and Simplification

Simplification: Naming an Expression

Let t be a symbolic expression and β a fresh auxiliary symbol. Replace selected occurrences of t in the store by β , keeping its definition:

$$\hat{\sigma}' = \hat{\sigma}[t \rightsquigarrow \beta], \quad \varphi' = \varphi \wedge \beta = t.$$

For example,

	$\hat{\sigma}(y)$	φ
before	$(\alpha_x + 1)(\alpha_x + 1)$	$\alpha_x \geq 0$
after	$\beta \cdot \beta$	$\alpha_x \geq 0 \wedge \beta = \alpha_x + 1$

Then $\gamma(\hat{\sigma}', \varphi') = \gamma(\hat{\sigma}, \varphi)$.

Every old valuation extends uniquely with $\beta = \llbracket t \rrbracket_\nu$. The equality forces every new valuation to agree with that extension.

Approximation and Simplification

Represented States and Precision

At a fixed remaining command, let R be the exact reachable set of stores. For a collection S of symbolic states, write

$$\Gamma(S) = \bigcup_{(\hat{\sigma}, \varphi) \in S} \gamma(\hat{\sigma}, \varphi).$$

$$\underbrace{\Gamma(S_-) \subseteq R}_{\text{underapproximation}} \quad \underbrace{\Gamma(S_0) = R}_{\text{exact representation}} \quad \underbrace{R \subseteq \Gamma(S_+)}_{\text{overapproximation}} .$$

- The direction of inclusion determines which conclusions are justified.
- A transformation can be exact relative to its input states even when earlier exploration has already lost coverage.

Approximation and Simplification

Underapproximation: Forgetting Branches

For if b then c_1 else c_2 , retaining only the true child discards inputs satisfying $\varphi \wedge \neg \llbracket b \rrbracket_{\hat{\sigma}}$.

Example, with $x = \alpha_x$ and $\varphi = \text{true}$:

```
if x <= 0 then y := 0 else y := -1
```

The true path satisfies $y \geq 0$. The omitted path violates it.

- Choosing a subset of exact paths gives an underapproximation of the full executions.
- A feasible violation on a retained exact path is a real counterexample.
- No violation among retained paths does not prove the contract.

Discarding a proved-infeasible branch is exact: it removes no executions.

Approximation and Simplification

Overapproximation: Weakening the Path Condition

If $\varphi \Rightarrow \psi$, then

$$\gamma(\hat{\sigma}, \varphi) \subseteq \gamma(\hat{\sigma}, \psi).$$

Example: let $\hat{\sigma}(x) = \alpha_x$, $\hat{\sigma}(y) = \alpha_y$, and

$$\varphi \equiv \alpha_x \geq 0 \wedge \alpha_y = \alpha_x + 1, \quad \psi \equiv \alpha_x \geq 0.$$

The exact state satisfies $y > x$. The weaker state admits $x = 0, y = 0$.

- Forgetting a constraint *adds* possible stores.
- A violation in the enlarged set may be spurious.
- If the enlarged set satisfies a property, the exact set satisfies it too.

To prove a program contract this way, the enlarged set must cover *all* relevant executions.

Approximation and Simplification

Overapproximation: Replacing an Expression

Suppose $\hat{\sigma}(x) = \alpha_x$ and $\hat{\sigma}(y) = \alpha_x \cdot \alpha_x$. Replace the term for y by fresh β .

replacement	added constraint	precision
$y \mapsto \beta$	$\beta = \alpha_x \cdot \alpha_x$	exact naming
$y \mapsto \beta$	$\beta \geq 0$	overapproximation
$y \mapsto \beta$	true	unconstrained overapproximation

For a term t over existing symbols α , retaining $A(\alpha, \beta)$ is sound if

$$\varphi \wedge \beta = t \quad \Rightarrow \quad A(\alpha, \beta).$$

The condition $\beta \geq 0$ proves $y \geq 0$, but loses the relation $y = x^2$. An unconstrained β even permits negative values of y .

Concolic Execution

Revised Example: Start with Two Concrete Inputs

Execute concretely while recording symbolic expressions and branch decisions.

```
if x <= 0 then y := 0 - x else skip;  
if y < 0 then z := 1 else z := 0
```

Choose the seed $(x, y) = (-2, 1)$. Let $A = \alpha_x$ and $B = \alpha_y$.

$$\sigma(x) = -2, \quad \sigma(y) = 1, \quad \hat{\sigma}(x) = A, \quad \hat{\sigma}(y) = B, \quad \varphi_0 = \text{true}.$$

- The concrete values select the branches of the current run.
- On the true branch, $y := 0 - x$ overwrites the initial value of y .
- On the false branch, skip preserves it, so B is a relevant input.

Keep A, B symbolic: do not add the seed equalities to the path condition. The initial value of z is irrelevant because both branches assign it.

Concolic Execution

First Run: Follow the Seed $(-2, 1)$

step	operation	concrete execution	symbolic effect
1	$x \leq 0$	$-2 \leq 0 : \text{true}$	record $A \leq 0$
2	$y := 0 - x$	$y = 2$	$\hat{\sigma}(y) = -A$
3	$y < 0$	$2 < 0 : \text{false}$	record $-A \geq 0$
4	$z := 0$	$z = 0$	$\hat{\sigma}(z) = 0$

$$\varphi_{TF} = (A \leq 0) \wedge (-A \geq 0), \quad \hat{\sigma}_f = [x \mapsto A, y \mapsto -A, z \mapsto 0].$$

The trace is TF : first guard true, second guard false.

- The second constraint uses the updated symbolic value $y = -A$.
- B is unconstrained on this path: the assignment overwrites it.
- Evaluating the final store at the seed gives $(x, y, z) = (-2, 2, 0)$.

Concolic Execution

Generate the Next Input by Flipping a Decision

Try the last decision first. Keep the first decision and negate the second:

$$A \leq 0 \wedge \neg(-A \geq 0) \quad \equiv \quad A \leq 0 \wedge A > 0.$$

This is unsatisfiable: no input follows path TT .

Backtrack to the first decision. Negate it and discard the old suffix:

$$\neg(A \leq 0) \quad \equiv \quad A > 0.$$

Choose $A = 1$ and keep the unconstrained input $B = 1$. Restart with $(x, y) = (1, 1)$.

Keeping $-A \geq 0$ would wrongly exclude this path. The new branch executes skip, so the next guard will use $y = B$, not $y = -A$.

Concolic Execution

Second Run: Replay the Input (1, 1)

Reset the symbolic store to $x \mapsto A$, $y \mapsto B$ and $\varphi = \text{true}$.

step	operation	concrete execution	symbolic effect
1	$x \leq 0$	$1 \leq 0 : \text{false}$	record $A > 0$
2	skip	$y = 1$	$\hat{\sigma}(y) = B$
3	$y < 0$	$1 < 0 : \text{false}$	record $B \geq 0$
4	$z := 0$	$z = 0$	$\hat{\sigma}(z) = 0$

$$\varphi_{FF} = (A > 0) \wedge (B \geq 0), \quad \hat{\sigma}_f = [x \mapsto A, y \mapsto B, z \mapsto 0].$$

The concrete result is $(x, y, z) = (1, 1, 0)$, on path FF .

Flip the last decision, retaining its prefix: $A > 0 \wedge \neg(B \geq 0)$, or $A > 0 \wedge B < 0$. This is satisfiable. Choose $(A, B) = (1, -1)$ for the next run.

Concolic Execution

Third Run: Replay the Input $(1, -1)$

Restart with $x = 1, y = -1$, symbolic inputs A, B , and $\varphi = \text{true}$.

step	operation	concrete execution	symbolic effect
1	$x \leq 0$	$1 \leq 0 : \text{false}$	record $A > 0$
2	skip	$y = -1$	$\hat{\sigma}(y) = B$
3	$y < 0$	$-1 < 0 : \text{true}$	record $B < 0$
4	$z := 1$	$z = 1$	$\hat{\sigma}(z) = 1$

$$\varphi_{FT} = (A > 0) \wedge (B < 0), \quad \hat{\sigma}_f = [x \mapsto A, y \mapsto B, z \mapsto 1].$$

The concrete result is $(x, y, z) = (1, -1, 1)$. This is a counterexample to the candidate postcondition $z = 0$.

Flipping the last decision returns to *FF*; flipping the first returns to the already explored $A \leq 0$ subtree. The path worklist avoids duplicate exploration.

Concolic Execution

What the Three Runs Establish

path	path condition	result
TT	$A \leq 0 \wedge -A < 0$	UNSAT
TF	$A \leq 0 \wedge -A \geq 0$	$z = 0$ (B arbitrary)
FT	$A > 0 \wedge B < 0$	$z = 1$
FF	$A > 0 \wedge B \geq 0$	$z = 0$

The three feasible paths cover all integer input pairs. The violation query on FT is

$$\varphi_{FT} \wedge \neg(\hat{\sigma}_f(z) = 0) \equiv A > 0 \wedge B < 0 \wedge \neg(1 = 0),$$

satisfied by $(A, B) = (1, -1)$. Therefore

$$\not\models \{\text{true}\} \text{ c } \{z = 0\}, \quad \models \{x \leq 0 \vee y \geq 0\} \text{ c } \{z = 0\}.$$

The exact safe input region is $A \leq 0 \vee B \geq 0$. Here y in the precondition denotes its initial value.