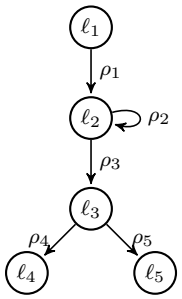
```
main(int x, int y, int z) {
  assume(y >= z);
  while (x < y) {
    x++;
  }
  assert(x >= z);
}
```

(a)



(b)

$$\rho_1 = (move(\ell_1, \ell_2) \wedge y \geq z \wedge skip(x, y, z))$$
$$\rho_2 = (move(\ell_2, \ell_2) \wedge x + 1 \leq y \wedge x' = x + 1 \wedge skip(y, z))$$
$$\rho_3 = (move(\ell_2, \ell_3) \wedge x \geq y \wedge skip(x, y, z))$$
$$\rho_4 = (move(\ell_3, \ell_4) \wedge x \geq z \wedge skip(x, y, z))$$
$$\rho_5 = (move(\ell_3, \ell_5) \wedge x + 1 \leq z \wedge skip(x, y, z))$$

(c)

Fig. 0.1 An example program (a), its control-flow graph (b), and its transition relations (c). Formally, the program is given by $Prog = (V, pc, \varphi_{init}, \mathcal{R}, \varphi_{err})$ where $V = (pc, x, y, z)$ is the tuple of program variables, pc is the program counter variable, $\mathcal{R} = \{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5\}$ is the set of ("single-statement") transition relations, $\varphi_{init} = at\_\ell_1$ is the initial condition, and $\varphi_{err} = at\_\ell_5$ is the error condition. The primed variables are $V' = (pc', x', y', z')$. We use *move* and *skip* as an abbreviation, for example $move(\ell_1, \ell_2)$ stands for $(pc = \ell_1 \wedge pc' = \ell_2)$ and $skip(x, y, z)$ represents $(x' = x \wedge y' = y \wedge z' = z)$.

0.1 Preliminaries

In this section, we describe programs, computations, and properties. To see an example of a program early, go to Figure 0.1.

0.1.1 Programs

A *program* $Prog = (V, pc, \varphi_{init}, \mathcal{R}, \varphi_{err})$ consists of

- V - a finite tuple of *program variables*,
- pc - a *program counter variable* that is included in V,
- φ_{init} - an *initiation condition* given by a formula over V,
- $\mathcal{R}$ - a finite set of ("single-statement") *transition relations*, where each transition relation $\rho \in \mathcal{R}$ is given by a formula over V and their primed versions V',
- φ_{err} - an *error condition* given by a formula over V.

Each program variable is assigned a *domain* of values. A *program state* is a function that assigns each program variable a value from its respective domain. Let Σ be the set of program states. A formula with free variables in V represents a set of program states. A formula with free variables in V and V' represents a binary relation over program states, where the first component of each pair assigns values to V and the second component of the pair assigns values to V'. We identify formulas with sets and relations that they represent. Accordingly, we identify the logical consequence relation between formulas $\models$ with the set inclusion $\subseteq$. Furthermore, we identify the satisfaction relation between valuations and formulas, which is denoted by $\models$, with the membership relation $\in$.

Example 1. For example, we consider the program shown in Figure 0.1 that has program variables $V = (pc, x, y, z)$. The program variables x, y, and z range over integers. The set of control locations is $\mathcal{L} = \{\ell_1, \ldots \ell_5\}$. A formula $y \geq z$ represents the set of program states in which the value of the variable y is greater than the value of z. Let s be a program state that assigns 1, 3, 2, and ℓ_1 to the program variables x, y, z, and pc, respectively. Then, we have $s \models y \geq z$. Furthermore, we have $y \geq z \models y + 1 \geq z$. □

Each state that satisfies the initiation condition φ_{init} is called an *initial* state. Each state that satisfies the error condition φ_{err} is called an *error* state. The program transition relation $\rho_{\mathcal{R}}$ is the union of the "single-statement" transition relations, i.e.,

$$\rho_{\mathcal{R}} = \bigvee_{\rho \in \mathcal{R}} \rho \ . \tag{0.1}$$

A pair of states (s, s') is connected by a program transition if it lies in the program transition relation $\rho_{\mathcal{R}}$, i.e., if $(s, s') \models \rho_{\mathcal{R}}$.

Let $\mathcal{L}$ be the domain of the program counter variable pc, i.e., the set of control locations of the program. To simplify the notation in the examples, we introduce the following abbreviations, where $\ell \in \mathcal{L}$ is a control location and $v_1, \ldots, v_n$ are program variables.

$$\begin{aligned} at\_\ell &= (pc = \ell) \\ at'\_\ell &= (pc' = \ell) \\ move(\ell, \ell') &= (at\_\ell \wedge at'\_\ell) \\ skip(v_1, \ldots, v_n) &= (v_1' = v_1 \wedge \ldots \wedge v_n' = v_n) \end{aligned} \tag{0.2}$$

Example 2. Our example program has an initiation condition $\varphi_{init} = (pc = at\_\ell_1)$ and an error condition $\varphi_{err} = (pc = at\_\ell_5)$. Program transitions $\mathcal{R} = \{\rho_1, \rho_2, \rho_3, \rho_4, \rho_5\}$ form a control-flow graph as shown in Figure 0.1(b). The corresponding transition relations are in Figure 0.1(c). Here, the first transition relation ρ_1 requires that the value of program counter is equal to ℓ_1 and that $y \geq z$ for the transition to be applicable. After executing the transition, the program counter value changes to ℓ_2 and the values of x, y,

and z are not modified. The transition relation of the program consists of the disjunction $\rho_{\mathcal{R}} = \rho_1 \vee \rho_2 \vee \rho_3 \vee \rho_4 \vee \rho_5$. □

A *program computation* is either a finite or an infinite sequence of program states $s_1, s_2, \ldots$ that satisfies the following three conditions.

- The first element is an initial state, i.e., $s_1 \models \varphi_{init}$.
- Each pair of consecutive states (s_i, s_{i+1}) is connected by a program transition, i.e., $(s_i, s_{i+1}) \models \rho_{\mathcal{R}}$.
- If the sequence is finite then the last element does not have any successors wrt. the program transition relation $\rho_{\mathcal{R}}$, i.e., if the last element is s_n, there is no state s such that $(s_n, s) \models \rho_{\mathcal{R}}$.

Example 3. In the example program, we consider a program computation connected by following the sequence of transitions $\rho_1, \rho_2, \rho_2, \rho_3, \rho_4$. We represent states as tuples of values of the program variables pc, x, y, and z, respectively.

$$(\ell_1, 1, 3, 2), (\ell_2, 1, 3, 2), (\ell_2, 2, 3, 2), (\ell_2, 3, 3, 2), (\ell_3, 3, 3, 2), (\ell_4, 3, 3, 2)$$

The last program state does not any successors wrt. the program transition relation. □

0.1.2 Correctness

We consider only two properties of program computations. These properties are concerned with the reachability of particular program states and the finiteness, i.e., termination, of the computation. Checking an expressive class of temporal properties can be reduced to reasoning about reachability and termination.

Safety A state is *reachable* if it occurs in a program computation. A program is *safe* if no error state is reachable.

Let φ_{reach} denote the set of reachable program states. A program is *safe* if and only if no error state lies in φ_{reach}, i.e.,

$$\varphi_{err} \wedge \varphi_{reach} \models \mathit{false} \ . \tag{0.3}$$

Example 4. In our example program, the state $(\ell_3, 3, 3, 2)$ is reachable, as witnessed by the above computation. The set of reachable program states is

$$\begin{aligned} \varphi_{reach} = (& at\_\ell_1 \vee \\ & at\_\ell_2 \wedge y \geq z \vee \\ & at\_\ell_3 \wedge y \geq z \wedge x \geq y \vee \\ & at\_\ell_4 \wedge y \geq z \wedge x \geq y) \ . \end{aligned}$$

Our program is safe, since φ_{reach} does not contain any states at the control location ℓ_5. □

Termination A program *terminates* if every computation is finite. A binary relation is *well-founded* if it does not admit any infinite chains. The restriction of the program transition relation $\rho_{\mathcal{R}}$ to the reachable program states is given by $\rho_{\mathcal{R}} \wedge \varphi_{reach}$ (the conjunction of a formula over V and V' and a formula over V). A program terminates if and only if the binary relation $\rho_{\mathcal{R}} \wedge \varphi_{reach}$ is well-founded.

Example 5. For our example, we obtain the following restriction of the program transition relation to reachable states.

$$\begin{aligned}\rho_{\mathcal{R}} \wedge \varphi_{reach} = (&\mathit{move}(\ell_1, \ell_2) \wedge y \geq z \wedge \mathit{skip}(x, y, z) \vee \\ &\mathit{move}(\ell_2, \ell_2) \wedge y \geq z \wedge x + 1 \leq y \wedge x' = x + 1 \wedge \mathit{skip}(y, z) \vee \\ &\mathit{move}(\ell_2, \ell_3) \wedge y \geq z \wedge x \geq y \wedge \mathit{skip}(x, y, z) \vee \\ &\mathit{move}(\ell_3, \ell_4) \wedge y \geq z \wedge x \geq y \wedge x \geq z \wedge \mathit{skip}(x, y, z))\end{aligned}$$

The restriction consists of four disjuncts, since the transition relation ρ_5 does not intersect with φ_{reach}. Furthermore, the restriction is well-founded, i.e., our program terminates. Any attempt to construct an infinite sequence leads to unbounded increase of the values of the variable x, which contradicts the condition that x is bounded from above by y whenever the loop execution is carried on. □

0.1.3 Inductive Safety and Termination Arguments

An *inductive invariant* φ contains the intial states and is closed under successors. Formally, an inductive invariant is a formula over the program variables that represents a superset of the initial program states and is closed under the application of the transition relation $\rho_{\mathcal{R}}$, i.e.,

$$\varphi_{init} \models \varphi \quad \text{and} \quad \varphi \wedge \rho_{\mathcal{R}} \models \varphi[V'/V]$$

A program is safe if there exists an inductive invariant φ that does not contain any error states, i.e., $\varphi \wedge \varphi_{err} \models \mathit{false}$.

Example 6. For our example program, the weakest inductive invariant consists of the set of all states and is represented by the formula *true*. The strongest inductive invariant was obtained in Example **??** and is shown below.

$$at\_\ell_1 \vee (at\_\ell_2 \wedge y \geq z) \vee (at\_\ell_3 \wedge y \geq z \wedge x \geq y) \vee (at\_\ell_4 \wedge y \geq z \wedge x \geq y)$$

The strongest inductive invariant does not contain any error states. We observe that a slightly weaker inductive invariant below also proves the safety of our examples.

$$at\_\ell_1 \vee (at\_\ell_2 \wedge y \geq z) \vee (at\_\ell_3 \wedge y \geq z \wedge x \geq y) \vee at\_\ell_4$$

□

The set W is well-founded with respect to certain ordering $\prec$ if there is no infinite sequence $w_1 \succ w_2 \succ \ldots$ starting from any $w_i \in W$. A program terminates if there exists such $(W, \prec)$ and a function r such that:

$$\varphi_{reach} \wedge \rho_{\mathcal{R}} \rightarrow r(V) \succ r(V')$$