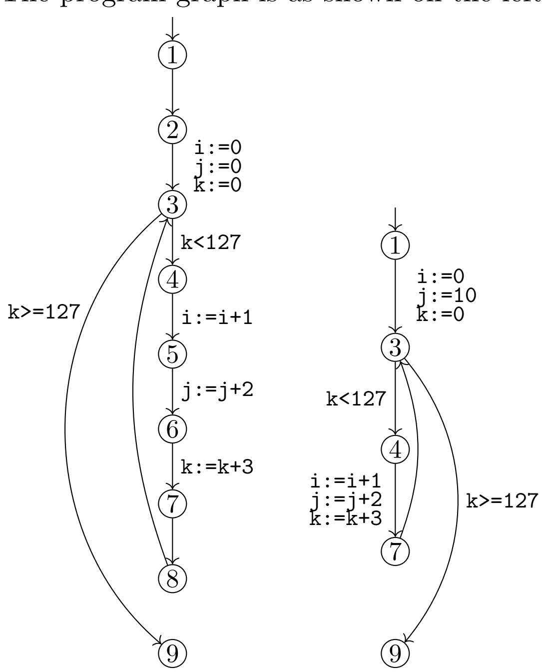
SAT and SMT Solving | WS 2022 | LVA 703147

All Solutions 12 | January 27, 2023

1 We add line numbers to the given program:

```
1  int main() {
2    char i = 0, j = 0, k = 0;
3    while (k < 127) {
4      i = i + 1;
5      j = j + 2;
6      k = k + 3;
7      // assert(k == (i + j));
8    }
9    return 0;
10 }
```

(a) The program graph is as shown on the left; on the right a simplified, equivalent version.



(b) The following are the initial predicate, and the transition predicate for the simplified graph.

$$
\begin{aligned}
I(\langle pc, i, j, k\rangle) &= (pc = 1)\\
T(\langle pc, i, j, k\rangle, \langle pc', i', j', k'\rangle) &= (pc = 1 \land pc' = 3 \land i' = 0 \land j' = 0 \land k' = 0) \lor\\
&\quad (pc = 3 \land pc' = 4 \land i' = i \land j' = j \land k' = k \land k < 127) \lor\\
&\quad (pc = 4 \land pc' = 7 \land i' = i + 1 \land j' = j + 2 \land k' = k + 3) \lor\\
&\quad (pc = 7 \land pc' = 3 \land i' = i \land j' = j \land k' = k) \lor\\
&\quad (pc = 3 \land pc' = 9 \land i' = i \land j' = j \land k' = k \land k \geq 127)
\end{aligned}
$$

Consider the following two candidate predicates to express a violation of the assertion.

$$P(\langle pc, i, j, k\rangle) = (pc = 7 \wedge k \neq i + j)$$
$$P'(\langle pc, i, j, k\rangle) = (pc = 7 \wedge \mathsf{SignExt}(k, 64) \neq \mathsf{SignExt}(i, 64) + \mathsf{SignExt}(j, 64))$$

(c) When representing i, j, and k as bitvectors of 8 bits (corresponding to the `char` type), and checking the condition P above, it is never violated. However, when checking the respective C program bmctest.c the assertion actually *does* get violated. Whaaaat?!

It turns out that property P does not reflect how the C program is actually executed. The reason is that when performing the comparison $\mathtt{i} + \mathtt{j} == \mathtt{k}$ in C, before the addition, `i` and `j` are implicitly type-converted to an `int` according to the C standard, and to perform a type-correct comparison, also `k` is promoted to an `int`. Since the types are signed, sign-based extension happens. So in order to check this in an SMT formula, one has to use the above property P' (assuming that an integer has 64 bits).

Using bounded model checking, indeed P' is violated for $\mathtt{i} = \mathtt{43}$, $\mathtt{j} = \mathtt{86}$, and $\mathtt{k} = -\mathtt{127}$ (after the last addition `k=k+1` overflowed): then $\mathtt{i} + \mathtt{j}$ evaluates to `129` (which can be represented without an overflow, due to the type conversion), while `k` still equals to $-\mathtt{127}$ also after the type conversion.

See bmc.py

$\boxed{2}$ The produced model contains two trees, six monkeys, and twelve bananas, see more_monkeys.smt2. This model is minimal (because Z3 searches for small models).

$\boxed{3}$ One can take the following instantiations:

$\mathsf{lives}(\mathsf{agatha}) \wedge \mathsf{lives}(\mathsf{butler}) \wedge \mathsf{lives}(\mathsf{charles})$	
$\forall x\, y.\ \mathsf{steal\_from}(x, y) \rightarrow \mathsf{hates}(x, y)$	$\{x \mapsto \mathsf{charles}, y \mapsto \mathsf{agatha}\}$
$\forall x\, y.\ \mathsf{steal\_from}(x, y) \rightarrow \neg\mathsf{richer}(x, y)$	$\{x \mapsto \mathsf{butler}, y \mapsto \mathsf{agatha}\}$
$\forall x.\ \mathsf{hates}(\mathsf{agatha}, x) \rightarrow \neg\mathsf{hates}(\mathsf{charles}, x)$	$\{x \mapsto \mathsf{agatha}\}$
$\mathsf{hates}(\mathsf{agatha}, \mathsf{charles})$	
$\mathsf{hates}(\mathsf{agatha}, \mathsf{agatha})$	
$\forall x.\ \mathsf{hates}(\mathsf{agatha}, x) \rightarrow \mathsf{hates}(\mathsf{butler}, x)$	$\{x \mapsto \mathsf{agatha}\}$ $\{x \mapsto \mathsf{charles}\}$
$\forall x.\ (\mathsf{lives}(x) \wedge \neg\mathsf{richer}(x, \mathsf{agatha})) \rightarrow \mathsf{hates}(\mathsf{butler}, x)$	$\{x \mapsto \mathsf{butler}\}$
$\forall x.\ \neg\mathsf{hates}(x, \mathsf{agatha}) \vee \neg\mathsf{hates}(x, \mathsf{butler}) \vee \neg\mathsf{hates}(x, \mathsf{charles})$	$\{x \mapsto \mathsf{butler}\}$
$\mathsf{steal\_from}(\mathsf{butler}, \mathsf{agatha}) \vee \mathsf{steal\_from}(\mathsf{charles}, \mathsf{agatha})$	

The respective set of *ground* formulas (i.e., formulas without variables) is unsatisfiable, see dreadbury_mansion_ground.smt2.

However, in this case Z3 even recognizes unsatisfiability of the quantified problem.

$\boxed{4}$ See `rushhour.py`.