Semantics of Programming Languages
Solutions to Assignment 12

Patrick Maier, Jan Schwinghammer
`http://www.ps.uni-sb.de/courses/sem-ws01/`

INFORMATIK

We are terribly sorry, there is a typo in program P. The condition in the if-statement should be negated so that the assert-statement is never reached. This does not affect exercise 12.1 but 12.2 does not make sense with the original P. Here is the corrected version, which we will use throughout the solutions; x and y are both initialized to 0.

```
l1: while (*) {
l2:   x++;
l3:   y++;
    }
l4: while (*) {
l5:   x--;
l6:   y--;
    }
l7: if (x != y) {
l8:   assert(0);
    }
```

Exercise 12.1 We use the notation of [1].

1. The set of states is $\mathsf{States} = \mathit{Val}^{\mathit{Var}}$ where $\mathit{Var} = \{x, y\}$ and $\mathit{Val} = \mathbb{Z}$. We have just one predicate, so the set of abstract states $\mathsf{States}^{\#} = \{0, 1, *\}$ is the set of trivectors with one component. Therefore the abstract program will have just one variable v_1.

   - Consider the assignment $x := x + 1$.

     The corresponding abstract statement is $v_1 := \mathsf{H}(\mathit{cond}_1, \mathit{cond}_0)$, where the function H takes two boolean expressions (with free variable v_1) to a value in $\{0, 1, *\}$ and can be expressed by a nested conditional $\mathsf{H}(\mathit{cond}_1, \mathit{cond}_0) = \mathit{cond}_1 \; ? \; 1 : \mathit{cond}_0 \; ? \; 0 : *$.

     We compute cond_1 and cond_0 by under-approximating the weakest precondition; $\mathit{cond}_1 \equiv \mathit{false}$ because

     $$\mathsf{F}(\widetilde{\mathsf{pre}}(\{s \in \mathsf{States} \mid s \models p_1\})) = \mathsf{F}(\{s \in \mathsf{States} \mid s \models x + 1 = y\}) = \mathit{false}.$$

     On the other hand, $\mathit{cond}_0 \equiv v_1 = 1$ since

     $$\mathsf{F}(\widetilde{\mathsf{pre}}(\{s \in \mathsf{States} \mid s \models \neg p_1\})) = \mathsf{F}(\{s \in \mathsf{States} \mid s \models x + 1 \neq y\}) = p_1.$$

     Hence the abstract statement is $v_1 := v_1 = 1 \; ? \; 0 : *$.

- Consider the assignment $y := y + 1$.

  Due to the symmetry of p_1, the abstract statement for incrementing y is the same than for incrementing x, i. e., $v_1 := v_1 = 1 \,?\, 0 : *$.

- Consider the assignment $x := x - 1$.

  $v_1 := \mathit{cond}_1 \,?\, 1 : \mathit{cond}_0 \,?\, 0 : *$ where $\mathit{cond}_1 \equiv \mathit{false}$ and $\mathit{cond}_0 \equiv v_1 = 1$ because

  $$\mathsf{F}(\widetilde{\mathsf{pre}}(\{s \in \mathsf{States} \mid s \models p_1\})) = \mathsf{F}(\{s \in \mathsf{States} \mid s \models x - 1 = y\}) = \mathit{false}$$
  $$\mathsf{F}(\widetilde{\mathsf{pre}}(\{s \in \mathsf{States} \mid s \models \neg p_1\})) = \mathsf{F}(\{s \in \mathsf{States} \mid s \models x - 1 \neq y\}) = p_1.$$

  Hence the abstract statement is $v_1 := v_1 = 1 \,?\, 0 : *$ again.

- Consider the assignment $y := y - 1$.

  Due to symmetry, the abstract statement is $v_1 := v_1 = 1 \,?\, 0 : *$.

2. To construct the abstract program $P^\#$, we still need to abstract the conditions in P. This is trivial for the while-loops ($l1$ and $l4$), $*$ remains $*$. For the if-statement ($l7$), the then-branch must be taken if $v_1 = 0$ and must not be taken if $v_1 = 1$. If $v_1 = *$ then the branch can be taken or not, there is a non-deterministic choice. We may consider $*$ as a numeric constant having the value 0.5 so we can express the abstract condition as $1 - v_1$, because $1 - 0 = 1$, $1 - 1 = 0$ and $1 - * = *$.[1]

   Here is the abstract program $P^\#$ (in C syntax) whose post operator equals the $\mathsf{post}^\#_{\mathsf{b\cdot c}}$ operator of P; the boolean variable $v1$ must be initialized to 1 because the initial state of P satisfies p_1.

```
l1: while (*) {
l2:   v1 = (v1 == 1) ? 0 : *;
l3:   v1 = (v1 == 1) ? 0 : *;
    }
l4: while (*) {
l5:   v1 = (v1 == 1) ? 0 : *;
l6:   v1 = (v1 == 1) ? 0 : *;
    }
l7: if (1 - v1) {
l8:   assert(0);
    }
```

3. The set of abstract states $\mathsf{States}^\# = \{0, 1, *\}^{\{v1\}}$ which we identify with the set of one-component trivectors $\{0, 1, *\}$. Thus the set of initial abstract states $\mathit{init}^\# = \{1\}$.

[1] Actually, we abuse the arithmetic function $x \mapsto 1-x$ on $\{0, 0.5, 1\}$ to express negation in a three-valued logic. Unfortunately such a hack does not work for conjunction and disjunction; the binary three-valued junctors have to be defined explicitly.

The equation system for the collecting semantics of $P^\#$ is

$$
\begin{aligned}
&acc^\#(l1) = init^\# \cup acc^\#(l3)\\
&acc^\#(l2) = \{v' \mid \exists v \in acc^\#(l1) : v = 1 \wedge v' = 0 \vee v \neq 1 \wedge v' = *\}\\
&acc^\#(l3) = \{v' \mid \exists v \in acc^\#(l2) : v = 1 \wedge v' = 0 \vee v \neq 1 \wedge v' = *\}\\
&acc^\#(l4) = acc^\#(l1) \cup acc^\#(l6)\\
&acc^\#(l5) = \{v' \mid \exists v \in acc^\#(l4) : v = 1 \wedge v' = 0 \vee v \neq 1 \wedge v' = *\}\\
&acc^\#(l6) = \{v' \mid \exists v \in acc^\#(l5) : v = 1 \wedge v' = 0 \vee v \neq 1 \wedge v' = *\}\\
&acc^\#(l7) = acc^\#(l4)\\
&acc^\#(l8) = \{v \in acc^\#(l7) \mid v = 0 \vee v = *\}
\end{aligned}
$$

Least solution:

$$
\begin{array}{lll}
acc^\#(l1) = \{1, *\} & acc^\#(l4) = \{1, *\} & acc^\#(l7) = \{1, *\}\\
acc^\#(l2) = \{0, *\} & acc^\#(l5) = \{0, *\} & acc^\#(l8) = \{*\}\\
acc^\#(l3) = \{*\} & acc^\#(l6) = \{*\} &
\end{array}
$$

As $acc^\#(l8) \neq \emptyset$, label $l8$ is reachable in program $P^\#$.

Exercise 12.2 We define two additional predicates $p_2 \equiv x + 1 = y$ and $p_3 \equiv x - 1 = y$. Note that $p_2 \equiv x = y - 1$ and $p_3 \equiv x = y + 1$.

To construct the abstract statement for the assignment $x := x+1$, we have to compute six under-approximations of weakest preconditions:

$$
\begin{aligned}
\mathsf{F}(\widetilde{\mathsf{pre}}(\{s \in \mathsf{States} \mid s \models p_1\})) &= \mathsf{F}(\{s \in \mathsf{States} \mid s \models x + 1 = y\}) = p_2\\
\mathsf{F}(\widetilde{\mathsf{pre}}(\{s \in \mathsf{States} \mid s \models \neg p_1\})) &= \mathsf{F}(\{s \in \mathsf{States} \mid s \models x + 1 \neq y\}) = \neg p_2\\
\mathsf{F}(\widetilde{\mathsf{pre}}(\{s \in \mathsf{States} \mid s \models p_2\})) &= \mathsf{F}(\{s \in \mathsf{States} \mid s \models x + 2 = y\}) = \mathit{false}\\
\mathsf{F}(\widetilde{\mathsf{pre}}(\{s \in \mathsf{States} \mid s \models \neg p_2\})) &= \mathsf{F}(\{s \in \mathsf{States} \mid s \models x + 2 \neq y\}) = p_1 \vee p_2 \vee p_3\\
\mathsf{F}(\widetilde{\mathsf{pre}}(\{s \in \mathsf{States} \mid s \models p_3\})) &= \mathsf{F}(\{s \in \mathsf{States} \mid s \models x = y\}) = p_1\\
\mathsf{F}(\widetilde{\mathsf{pre}}(\{s \in \mathsf{States} \mid s \models \neg p_3\})) &= \mathsf{F}(\{s \in \mathsf{States} \mid s \models x \neq y\}) = \neg p_1
\end{aligned}
$$

This yields the abstract assignment

$$
\begin{aligned}
\langle v_1, v_2, v_3\rangle := \langle & v_2 = 1 \ ?\ 1 : v_2 = 0 \ ?\ 0 : *,\\
& \mathit{false} \ ?\ 1 : v_1 = 1 \vee v_2 = 1 \vee v_3 = 1 \ ?\ 0 : *,\\
& v_1 = 1 \ ?\ 1 : v_1 = 0 \ ?\ 0 : *\rangle
\end{aligned}
$$

which can be written shorter as $\langle v_1, v_2, v_3\rangle := \langle v_2, v_1 = 1 \vee v_2 = 1 \vee v_3 = 1 \ ?\ 0 : *, v_1\rangle$.

As the condition of the if-statement in P is exactly $\neg p_1$, the abstract condition remains $1 - v_1$.

Here is the refined boolean program $P^\#$; the boolean vector $\langle v1, v2, v3\rangle$ must be initialized to $\langle 1, 0, 0\rangle$.

```
l1: while (*) {
l2:   <v1, v2, v3> = <v2, (v1 = 1 || v2 = 1 || v3 = 1) ? 0 : *, v1>;
l3:   <v1, v2, v3> = <v3, v1, (v1 = 1 || v2 = 1 || v3 = 1) ? 0 : *>;
    }
l4: while (*) {
l5:   <v1, v2, v3> = <v3, v1, (v1 = 1 || v2 = 1 || v3 = 1) ? 0 : *>;
l6:   <v1, v2, v3> = <v2, (v1 = 1 || v2 = 1 || v3 = 1) ? 0 : *, v1>;
    }
l7: if (1 - v1) {
l8:   assert(0);
    }
```

Here is an informal argument that $l8$ is unreachable in $P^{\#}$.

- Initially, the value of $v1$ is 1.
- The first loop preserves the value of $v1$ because its first assignment stores the value of $v1$ in $v3$, then the second assignment restores it from there.
- Similarly, the second loop preserves the value of $v1$ because it stores $v1$ to $v2$ and then restores it from there.
- Therefore, at label $l7$ the value of $v1$ must be 1, so the condition $1 - v1$ evaluates to 0 and the then-branch is not taken.

Exercise 12.3 Given a program with k variables $\{x_1, \dots, x_k\}$, all of type $\mathbb{Z}$, we should define the set of predicates $\mathcal{P} = \{x_i = z \mid 1 \le i \le k, z \in \mathbb{Z}\}$. This induces a predicate abstraction in the canonical way, however the bitvectors are of infinite length, more precisely, α_{bool} maps sets of states to $2^{\{0,1\}^{\{1,\dots,k\}\times\mathbb{Z}}}$. And a cartesian abstraction on top of α_{bool} would still yield an infinite trivector in $\{0,1,*\}^{\{1,\dots,k\}\times\mathbb{Z}}$. As we can not handle infinite objects in real computers, this analysis is not implementable.

However, we might restrict to predicates that compare the x_i only to those integers which appear as literal constants in the program text. There can only be finitely many constants, let's say l, then the size of $\mathcal{P}$ would be kl, so bitvectors and trivectors are of length kl. This analysis can not achieve full constant propagation, for instance, it will not detect that the expression $x_i + 1$ is constant when x_i is constant. However, it will still detect x_j as constant after the assignment $x_j := x_i$ where x_i is constant.

Does a cartesian abstraction on top of the boolean abstraction cause a loss of precision for constant propagation? Let S be a set of states and assume that the variable x_i is constant in S, i.e., $\exists z \in \mathbb{Z}\ \forall s \in S : s(x_i) = z$. Then for all bitvectors $v \in \alpha_{bool}(S)$, $v(x_i = z) = 1$ and $v(x_i = z') = 0$ for all $z' \in \mathbb{Z} \setminus \{z\}$. Hence the trivector that comes out of $\alpha_{cart}(\alpha_{bool}(S))$ will have a 1 at position $x_i = z$ and 0 at all positions $x_i = z'$ for $z' \in \mathbb{Z} \setminus \{z\}$. This means that the information detected by the boolean abstraction is not destroyed by a further cartesian abstraction; constant propagation can be implemented using trivectors rather than sets of bitvectors.

Exercise 12.4 Let $\mathbf{D} = \langle D, \sqsubseteq \rangle$ and $\mathbf{D}^\# = \langle D^\#, \sqsubseteq^\# \rangle$ be posets. A function $f : D \to D^\#$ is called *completely additive* if for all $X \subseteq D$, the existence of $\bigsqcup X$ implies that $\bigsqcup^\# f(X)$ exists and $f(\bigsqcup X) = \bigsqcup^\# f(X)$. Obviously, a completely additive function is additive.

Let $\mathbf{D} \underset{\alpha}{\overset{\gamma}{\rightleftarrows}} \mathbf{D}^\#$ be a Galois connection. To prove that α is completely additive, we pick a set $X \subseteq D$ such that $\bigsqcup X$ exists.

- As $\bigsqcup X$ is an upper bound of X, $\forall x \in X : x \sqsubseteq \bigsqcup X$, so by monotonicity of α, $\forall x \in X : \alpha(x) \sqsubseteq^\# \alpha(\bigsqcup X)$. Hence $\alpha(\bigsqcup X)$ is an upper bound of $\alpha(X)$.

- Let $y \in D^\#$ be an upper bound of $\alpha(X)$, i. e., $\forall x \in X : \alpha(x) \sqsubseteq^\# y$. As α and γ form a Galois connection, $\forall x \in X : x \sqsubseteq \gamma(y)$, i. e., $\gamma(y)$ is an upper bound of X, so $\bigsqcup X \sqsubseteq \gamma(y)$, for $\bigsqcup X$ is the least upper bound of X. Again because of the Galois connection, $\alpha(\bigsqcup X) \sqsubseteq^\# y$.

To summarize, $\alpha(\bigsqcup X)$ is the least upper bound of $\alpha(X)$, i. e., $\bigsqcup^\# \alpha(X)$ exists and $\alpha(\bigsqcup X) = \bigsqcup^\# \alpha(X)$.

By duality, γ is *completely multiplicative*, i. e., for all $Y \subseteq D^\#$, if the greatest lower bound of Y exists in $\mathbf{D}^\#$ then the greatest lower bound of $\gamma(Y)$ exists in $\mathbf{D}$ is equal to the image of the greatest lower of Y under γ.

To see that γ need not be additive, let $\mathbf{D} = \langle D, \sqsubseteq \rangle$ with $D = \{a, b\} \cup \{z \in \mathbb{Z} \mid z \leq 0\}$ and $a, b \sqsubseteq z$ for all $z \in \mathbb{Z}$ and $z \sqsubseteq z'$ iff $z \leq z'$ for all $z, z' \in \mathbb{Z}$. Let $\mathbf{D}^\# = \langle D^\#, \sqsubseteq^\# \rangle$ where $D^\# = \{a, b, 0\} \subseteq D$ and $\sqsubseteq^\#$ is the order induced by $\sqsubseteq$ on $D^\#$. We define $\alpha : D \to D^\#$ by $\alpha(a) = a$, $\alpha(b) = b$ and $\alpha(z) = 0$ for all $z \in \mathbb{Z}$; $\gamma : D^\# \to D$ is the identity. Then $\mathbf{D} \underset{\alpha}{\overset{\gamma}{\rightleftarrows}} \mathbf{D}^\#$ is a Galois connection. Furthermore, $a \sqcup^\# b$ exists in $D^\#$ (it is 0) but $\gamma(a) \sqcup \gamma(b)$ does not exist in $\mathbf{D}$ (in $\mathbf{D}$, all $z \in \mathbb{Z}$ are upper bounds of $\gamma(\{a, b\}) = \{a, b\}$).

References

[1] Thomas Ball, Andreas Podelski and Sriram K. Rajamani. Boolean and Cartesian Abstraction for Model Checking C Programs. In *Proceedings of TACAS*, 2001.