Miscellaneous Problems on Semantics and Computability

* 1. For each of the following states σ_1, σ_2, σ_3:

1. $\sigma_1 = [x \mapsto 0, y \mapsto 0]$
2. $\sigma_2 = [x \mapsto -1, y \mapsto 10]$
3. $\sigma_3 = [x \mapsto -1, z \mapsto 10]$

calculate the following:

(a) the denotation of the arithmetic expression $x + (y * 3)$;

(b) the denotation of the Boolean expression $x == y$ && $x - y \leq 3$;

(c) And, for each $i \in \{1, 2, 3\}$, a state σ such that while $x \leq y$ && $!(x = y)$ do $x \leftarrow x + z$, $\sigma_i \Downarrow \sigma$ when one exists.

Solution

(a) The arithmetic expression $x + (y * 3)$ evaluates to:
1. 0 in the state $[x \mapsto 0, y \mapsto 0]$
2. 29 in the state $[x \mapsto -1, y \mapsto 10]$
3. -1 in the state $[x \mapsto -1, z \mapsto 10]$

(b) The Boolean expression evaluates $x == y$ && $x - y \leq 3$ to:
1. $\top$ in the state $[x \mapsto 0, y \mapsto 0]$
2. $\bot$ in the state $[x \mapsto -1, y \mapsto 10]$
3. $\bot$ in the state $[x \mapsto -1, z \mapsto 0]$

(c) The statement while $x \leq y$ &&$!(x = y)$ do $x \leftarrow x + z$
1. $[x \mapsto 0, y \mapsto 0, z \mapsto 0]$
2. Does not terminate.
3. $[x \mapsto 9, y \mapsto 0, z \mapsto 10]$

* 2. Find two distinct states $\sigma, \sigma' \in$ State such that while $1 \leq x$ do $y \leftarrow y + x$, $x \leftarrow x - 1$, $\sigma \Downarrow [y \mapsto 10]$ and likewise for while $1 \leq x$ do $y \leftarrow y + x$, $x \leftarrow x - 1$, $\sigma' \Downarrow [y \mapsto 10]$.

Solution

Equivalently, while $1 \leq x$ do $y \leftarrow y - x$, $x \leftarrow x - 1$, $\sigma \Downarrow [x \mapsto 0, y \mapsto 10]$. From which we can deduce that, $\sigma(x) \geq 0$ and $\sigma(y) = 10 - \sum_{k=0}^{\sigma(x)} k$ as for each value between 0 and $\sigma(x)$ there is an iteration of the loop where y is increased by this value. The solutions are therefore:

- $[y \mapsto 10, x \mapsto 0]$
- $[y \mapsto 9, x \mapsto 1]$
- $[y \mapsto 7, x \mapsto 2]$
- $[y \mapsto 4, x \mapsto 3]$
- $[y \mapsto 0, x \mapsto 4]$
- $[y \mapsto -5, x \mapsto 5]$
- $[y \mapsto -11, x \mapsto 6]$
- etc.

** 3. Suppose $e \in \mathcal{B}$ is a Boolean expression and $S_1, S_2, S_3 \in \mathcal{S}$ are statements. Prove that the statement if e then S_1 else S_2 and the statement if e then {if e then S_1 else S_3} else {if $!e$ then S_2 else S_3} are semantically equivalently.

Solution

First, let us suppose that if e then S_1 else S_2, $\sigma \Downarrow \sigma'$ for two states $\sigma, \sigma' \in \mathsf{State}$. By inversion, one of the following must apply:

- Either $[\![e]\!]_{\mathcal{B}}(\sigma) = \top$ and $S_1, \sigma \Downarrow \sigma'$. In this case, we can construct the following derivation:

$$\dfrac{\dfrac{S_1, \sigma \Downarrow \sigma'}{\text{if } e \text{ then } S_1 \text{ else } S_3, \sigma \Downarrow \sigma'}}{\text{if } e \text{ then } \{\text{if } e \text{ then } S_1 \text{ else } S_3\} \text{ else } \{\text{if } !e \text{ then } S_2 \text{ else } S_3\}, \sigma \Downarrow \sigma'}$$

  as required.

- Or, $[\![e]\!]_{\mathcal{B}}(\sigma) = \bot$ and $S_2, \sigma \Downarrow \sigma'$ in which case we have the following derivation:

$$\dfrac{\dfrac{S_2, \sigma \Downarrow \sigma'}{\text{if } !e \text{ then } S_2 \text{ else } S_3, \sigma \Downarrow \sigma'}}{\text{if } e \text{ then } \{\text{if } e \text{ then } S_1 \text{ else } S_3\} \text{ else } \{\text{if } !e \text{ then } S_2 \text{ else } S_3\}, \sigma \Downarrow \sigma'}$$

  as $[\![!e]\!]_{\mathcal{B}}(\sigma) = \neg[\![e]\!]_{\mathcal{B}}(\sigma) = \top$.

In the converse direction, let us suppose that if e then {if e then S_1 else S_3} else {if $!e$ then S_2 else S_3}, $\sigma \Downarrow \sigma'$. Again by inversion, there are two cases to consider:

- Either $[\![e]\!]_{\mathcal{B}}(\sigma) = \top$ and if e then S_1 else S_3, $\sigma \Downarrow \sigma'$. Applying inversion again tells us that $S_1, \sigma \Downarrow \sigma'$ as the other case (where $[\![e]\!]_{\mathcal{B}}(\sigma) = \bot$) would give us a contradiction. Therefore, we can construct the following derivation:

$$\dfrac{S_1, \sigma \Downarrow \sigma'}{\text{if } e \text{ then } S_1 \text{ else } S_2, \sigma \Downarrow \sigma'}$$

  as required.

- Or $[\![e]\!]_{\mathcal{B}}(\sigma) = \bot$ and if $!e$ then S_2 else $S_3, \sigma \Downarrow \sigma'$. Applying inversion again tells us that $S_2, \sigma \Downarrow \sigma'$ as the other case (where $[\![!e]\!]_{\mathcal{B}}(\sigma) = \bot$) would give us a contradiction. Therefore, we can construct the following derivation:

$$\frac{S_2, \sigma \Downarrow \sigma'}{\text{if } e \text{ then } S_1 \text{ else } S_2, \sigma \Downarrow \sigma'}$$

  as required.

** 4. Let $\mathbb{P}$ be the three element set $\{+, -, \pm\}$. We define the function $\text{sign}_\sigma : \mathcal{A} \to \mathbb{P}$ for a given state $\sigma \in \text{State}$ by recursion as follows:

$$\begin{aligned}
\text{sign}_\sigma(x) &= \begin{cases} + & \text{if } \sigma(x) \geq 0 \\ - & \text{otherwise} \end{cases} \\
\text{sign}_\sigma(n) &= \begin{cases} + & \text{if } n \geq 0 \\ - & \text{otherwise} \end{cases} \\
\text{sign}_\sigma(e_1 + e_2) &= \begin{cases} + & \text{if } \text{sign}_\sigma(e_1) = + \text{ and } \text{sign}_\sigma(e_2) = + \\ - & \text{if } \text{sign}_\sigma(e_1) = - \text{ and } \text{sign}_\sigma(e_2) = - \\ \pm & \text{otherwise} \end{cases} \\
\text{sign}_\sigma(e_1 - e_2) &= \begin{cases} + & \text{if } \text{sign}_\sigma(e_1) = + \text{ and } \text{sign}_\sigma(e_2) = - \\ - & \text{if } \text{sign}_\sigma(e_1) = - \text{ and } \text{sign}_\sigma(e_2) = + \\ \pm & \text{otherwise} \end{cases} \\
\text{sign}_\sigma(e_1 * e_2) &= \begin{cases} \pm & \text{if } \text{sign}_\sigma(e_1) = \pm \text{ or } \text{sign}_\sigma(e_2) = \pm \\ + & \text{if } \text{sign}_\sigma(e_1) = \text{sign}_\sigma(e_2) \\ + & \text{if } \text{sign}_\sigma(e_1) \neq \text{ and } \text{sign}_\sigma(e_2) \end{cases}
\end{aligned}$$

Prove by structural induction over arithmetic expressions, for any arithmetic $e \in \mathcal{A}$ and state $\sigma \in \text{State}$, that:

- If $\text{sign}_\sigma(e) = +$, then $[\![e]\!]_{\mathcal{A}}(\sigma) \geq 0$;
- And, if $\text{sign}_\sigma(e) = -$, then $[\![e]\!]_{\mathcal{A}}(\sigma) < 0$

You should treat this as a single induction proof rather than proving each cases separately.

Solution

Let us proceed by structural induction on arithmetic expressions.

- Consider an expression of the form $n \in \mathbb{Z}$:
  - If $\text{sign}_\sigma(n) = +$, then by definition $n \geq 0$ and thus $[\![n]\!]_{\mathcal{A}}(\sigma) = n \geq 0$ as required.
  - Likewise, if $\text{sign}_\sigma(n) = -$, then by definition $n < 0$ and thus $[\![n]\!]_{\mathcal{A}}(\sigma) = n < 0$ as required.
- Consider an expression of the form $x \in \text{Var}$:
  - If $\text{sign}_\sigma(n) = +$, then by definition $\sigma(x) \geq 0$ and thus $[\![x]\!]_{\mathcal{A}}(\sigma) = \sigma(x) \geq 0$ as required.
  - Likewise, if $\text{sign}_\sigma(x) = -$, then by definition $\sigma(x) < 0$ and thus $[\![x]\!]_{\mathcal{A}}(\sigma) = \sigma(x) < 0$ as required.
- Consider an expression of the form $e_1 + e_2$:

- If $\text{sign}_\sigma(e_1 + e_2) = +$, then by definition $\text{sign}_\sigma(e_1) = +$ and $\text{sign}_\sigma(e_2) = +$. Therefore, by induction, $[\![e_1]\!]_{\mathcal{A}}(\sigma)$, $[\![e_2]\!]_{\mathcal{A}}(\sigma) \geq 0$. It thus follows that $[\![e_1 + e_2]\!]_{\mathcal{A}}(\sigma) = [\![e_1]\!]_{\mathcal{A}}(\sigma) + [\![e_2]\!]_{\mathcal{A}}(\sigma) \geq 0$ as required.
- If $\text{sign}_\sigma(e_1 + e_2) = -$, then by definition $\text{sign}_\sigma(e_1) = -$ and $\text{sign}_\sigma(e_2) = -$. Therefore, by induction, $[\![e_1]\!]_{\mathcal{A}}(\sigma)$, $[\![e_2]\!]_{\mathcal{A}}(\sigma) < 0$. It thus follows that $[\![e_1 + e_2]\!]_{\mathcal{A}}(\sigma) = [\![e_1]\!]_{\mathcal{A}}(\sigma) + [\![e_2]\!]_{\mathcal{A}}(\sigma) < 0$ as required.

- Consider an expression of the form $e_1 - e_2$:
  - If $\text{sign}_\sigma(e_1 - e_2) = +$, then by definition $\text{sign}_\sigma(e_1) = +$ and $\text{sign}_\sigma(e_2) = -$. Therefore, by induction, $[\![e_1]\!]_{\mathcal{A}}(\sigma) \geq 0$ and $[\![e_2]\!]_{\mathcal{A}}(\sigma) < 0$. It thus follows that $[\![e_1 - e_2]\!]_{\mathcal{A}}(\sigma) = [\![e_1]\!]_{\mathcal{A}}(\sigma) - [\![e_2]\!]_{\mathcal{A}}(\sigma) \geq 0$ as required.
  - If $\text{sign}_\sigma(e_1 - e_2) = -$, then by definition $\text{sign}_\sigma(e_1) = -$ and $\text{sign}_\sigma(e_2) = +$. Therefore, by induction, $[\![e_1]\!]_{\mathcal{A}}(\sigma) < 0$ and $[\![e_2]\!]_{\mathcal{A}}(\sigma) \geq 0$. It thus follows that $[\![e_1 - e_2]\!]_{\mathcal{A}}(\sigma) = [\![e_1]\!]_{\mathcal{A}}(\sigma) - [\![e_2]\!]_{\mathcal{A}}(\sigma) < 0$ as required.

- Consider an expression of the form $e_1 * e_2$:
  - If $\text{sign}_\sigma(e_1 * e_2) = +$, then by definition $\text{sign}_\sigma(e_1) = +$ and $\text{sign}_\sigma(e_1 2) = +$ or $\text{sign}_\sigma(e_1) = -$ and $\text{sign}_\sigma(e_2) = -$. If both have the sign $+$, then, by induction, $[\![e_1]\!]_{\mathcal{A}}(\sigma)$, $[\![e_2]\!]_{\mathcal{A}}(\sigma) \geq 0$ and thus it follows that $[\![e_1 * e_2]\!]_{\mathcal{A}}(\sigma) = [\![e_1]\!]_{\mathcal{A}}(\sigma) \cdot [\![e_2]\!]_{\mathcal{A}}(\sigma) \geq 0$ as required. Otherwise, if both have the sign $-$, then, by induction, $[\![e_1]\!]_{\mathcal{A}}(\sigma)$, $[\![e_2]\!]_{\mathcal{A}}(\sigma) < 0$ and thus it follows that $[\![e_1 * e_2]\!]_{\mathcal{A}}(\sigma) = [\![e_1]\!]_{\mathcal{A}}(\sigma) \cdot [\![e_2]\!]_{\mathcal{A}}(\sigma) < 0$ as required.
  - If $\text{sign}_\sigma(e_1 - e_2) = -$, then by definition $\text{sign}_\sigma(e_1) = -$ and $\text{sign}_\sigma(e_2) = +$ or vice versa. Let us suppose that $\text{sign}_\sigma(e_1) = -$ and $\text{sign}_\sigma(e_2) = +$. By induction, $[\![e_1]\!]_{\mathcal{A}}(\sigma) < 0$ and $[\![e_2]\!]_{\mathcal{A}}(\sigma) \geq 0$. It thus follows that $[\![e_1 * e_2]\!]_{\mathcal{A}}(\sigma) = [\![e_1]\!]_{\mathcal{A}}(\sigma) \cdot [\![e_2]\!]_{\mathcal{A}}(\sigma) < 0$ as required. The case where $\text{sign}_\sigma(e_1) = +$ and $\text{sign}_\sigma(e_2) = -$ is analogous.

*** 5. Let P be the following While program:

$$\begin{array}{l}\textsf{while } x + 1 \leq y \textsf{ do} \\ \quad y \leftarrow y - 1;\end{array}$$

Prove by induction, over a combination of x and y, that P terminates from any initial state $\sigma \in \text{State}$ such that $\sigma(x) \leq \sigma(y)$. That is, prove, for all $\sigma \in \text{State}$ such that $\sigma(x) \leq \sigma(y)$, that there exists some $\sigma' \in \text{State}$ such that $P, \sigma \Downarrow \sigma'$. You may *not* assume that $\sigma(x)$ or $\sigma(y)$ are greater than or equal to 0.

Solution

This can be proven by induction over $\sigma(y) - \sigma(x)$.

- In the base case, we know that $\sigma(y) = \sigma(x)$. Therefore, we have that:

$$\frac{}{P, \sigma \Downarrow \sigma}$$

  as the branch condition $x + 1 \leq y$ evaluates to $\bot$. Therefore, the program terminates in this case.

- Now let us suppose that $\sigma(y) - \sigma(x) = n + 1$ for some $n \geq 0$. Therefore, the branch condition $x + 1 \leq y$ evaluates to $\top$. Then we must construct a derivation of the form:

$$\frac{\dfrac{}{y \leftarrow y - 1, \sigma \Downarrow \sigma[y \mapsto \sigma(y) - 1]} \quad \begin{array}{c}\vdots \\ P, \sigma[y \mapsto \sigma(y) - 1] \Downarrow \sigma'\end{array}}{P, \sigma \Downarrow \sigma'}$$

We have that $(\sigma(y)-1)-\sigma(x)=n$ and, therefore, the induction hypothesis tells us that there exists some final state $\sigma' \in \mathsf{State}$ such that $P, \sigma[y \mapsto \sigma(y)-1] \Downarrow \sigma'$ as required.

The following question uses the notation $\sigma \sim_x \sigma'$ which indicates, for some variable $x \in \mathsf{Var}$ and for two states $\sigma, \sigma' \in \mathsf{State}$, that $\sigma(y) = \sigma'(y)$ for all *other* variable $y \in \mathsf{Var} \setminus \{x\}$. For example, $[x \mapsto 2, y \mapsto 3] \sim_x [x \mapsto -100, y \mapsto 3]$ but $[x \mapsto 2, y \mapsto 3] \not\sim_x [x \mapsto 2, y \mapsto 4]$.

*** 6. Let us suppose we introduce a new language construct for Boolean expressions so that the extended grammar is given as follows:

$$B \to \mathsf{true} \mid \mathsf{false} \mid B \;\&\&\; B \mid B \;||\; B \mid !B \mid \mathsf{forall}\ x.B$$

The denotational semantics for the new construct is given by the following equation:

$$[\![\mathsf{forall}\ x.e]\!]_{\mathcal{B}}(\sigma) = \begin{cases} \top & \forall \sigma' \in \mathsf{State}.\, \sigma \sim_x \sigma' \Rightarrow [\![e]\!]_{\mathcal{B}}(\sigma') \\ \bot & \text{otherwise} \end{cases}$$

with all other constructs retaining their original semantics.

(a) Evaluate the denotation of the expressions $\mathsf{forall}\ x.x = 2$ and $\mathsf{forall}\ x.x \leq y \;||\; y \leq x$ in the state $[x \mapsto 2, y \mapsto 2]$.

(b) Prove that if $e \in \mathcal{B}$ and $e' \in \mathcal{B}$ are semantically equivalent Boolean expressions, then $\mathsf{forall}\ x.e$ and $\mathsf{forall}\ x.e'$ are semantically equivalent.

(c) Suppose that $e \in \mathcal{B}$ is a Boolean expression such that $[\![e]\!]_{\mathcal{B}}(\sigma) = [\![e]\!]_{\mathcal{B}}(\sigma')$ for any two states $\sigma \sim_x \sigma'$. Prove that $\mathsf{forall}\ x.e$ is semantically equivalent to e.

Solution

(a)
- $[\![\mathsf{forall}\ x.x = 2]\!]_{\mathcal{B}}([x \mapsto 2, y \mapsto 2]) = \bot$ as $[\![x = 2]\!]_{\mathcal{B}}([x \mapsto 3, y \mapsto 2])$.
- $[\![\mathsf{forall}\ x.x \leq y \;||\; y \leq x]\!]_{\mathcal{B}}([x \mapsto 2, y \mapsto 2]) = \top$ as, for any $n \in \mathbb{Z}$, either $n \leq 2$ or else $2 \leq n$.

(b) Suppose $e \in \mathcal{B}$ and $e' \in \mathcal{B}$ are semantically equivalent Boolean expressions. That is $[\![e]\!]_{\mathcal{B}}(\sigma) = [\![e']\!]_{\mathcal{B}}(\sigma)$ for all $\sigma \in \mathsf{Staet}$. Let $\sigma \in \mathsf{State}$ be some state. By definition $[\![\mathsf{forall}\ x.e]\!]_{\mathcal{B}}(\sigma)$ is true if $[\![e]\!]_{\mathcal{B}}(\sigma') = \top$ for all $\sigma' \sim_x \sigma$. If this is the case, then $[\![e']\!]_{\mathcal{B}}(\sigma') = \top$ for all $\sigma' \sim_x \sigma$ by semantic equivalence and thus $[\![\mathsf{forall}\ x.e]\!]_{\mathcal{B}}(\sigma)$ is also true. Conversely, if $[\![\mathsf{forall}\ x.e]\!]_{\mathcal{B}}(\sigma)$ is false then $[\![e]\!]_{\mathcal{B}}(\sigma') = \bot$ for some $\sigma' \sim_x \sigma$. Therefore, $[\![e']\!]_{\mathcal{B}}(\sigma') = \bot$ for some $\sigma' \sim_x \sigma$ and $[\![\mathsf{forall}\ x.e']\!]_{\mathcal{B}}(\sigma)$ is false as required. Therefore, $\mathsf{forall}\ x.e$ and $\mathsf{forall}\ x.e'$ are semantically equivalent.

(c) Suppose that $e \in \mathcal{B}$ is a Boolean expression such that $[\![e]\!]_{\mathcal{B}}(\sigma) = [\![e]\!]_{\mathcal{B}}(\sigma')$ for any two states $\sigma \sim_x \sigma'$. Let $\sigma \in \mathsf{State}$ be some state. If $[\![\mathsf{forall}\ x.e]\!]_{\mathcal{B}}(\sigma)$ is true, then $[\![e]\!]_{\mathcal{B}}(\sigma') = \top$ for all $\sigma' \sim_x \sigma$. Therefore, $[\![e]\!]_{\mathcal{B}}(\sigma) = \top$ by assumption. Conversely, if $[\![\mathsf{forall}\ x.e]\!]_{\mathcal{B}}(\sigma)$ is false, then $[\![e]\!]_{\mathcal{B}}(\sigma') = \bot$ for some $\sigma' \sim_x \sigma$. Therefore, $[\![e]\!]_{\mathcal{B}}(\sigma) = \bot$ by assumption. Thus, e and $\forall x.e$ are semantically equivalent as required.

* 7. Show that the function $f : \mathbb{N} \rightharpoonup \mathbb{N}$ defined by

$$f(x)\begin{cases} \simeq x+1 & \text{if } x^2-1 \text{ is at least 2022} \\ \uparrow & \text{otherwise} \end{cases}$$

is computable.

Solution

The function is computed by the following code with respect to x.

```
y := x * x -  1;
if (! (y < 2022)) { x := x+1; y := 0 }
else { while (true) do skip }
```

Award 1 mark for correctly stating the input/output variable; 2 marks for a mostly correct program; 1 mark for the infinite loop when the output is undefined; and 1 mark for setting all auxiliary variables to zero at the end.

* 8. State whether each of the following statements is true or false.

(a) Every injection has an inverse.

(b) The set $\mathbb{N}$ is decidable.

(c) Some WHILE programs compute total functions.

(d) If a function has an inverse, it must be a surjection.

(e) The set of all WHILE programs is countable.

Solution

(a) False.

(b) True.

(c) True.

(d) True.

(e) True.

* 9. Write a program that demonstrates that the function

$$f : \mathbb{N} \rightharpoonup \mathbb{N}$$
$$f(n)\begin{cases} \simeq 2^n & \text{if } n \text{ is divisible by 3} \\ \uparrow & \text{otherwise} \end{cases}$$

is computable.

Solution

The following program computes f with respect to n.

```
q := 0; r := n;
while (3 <= r) do { q := q + 1; r := r - 3 };
if (r = 0) then {
  j := 0; y := 1;  // Loop invariant: y = 2^j
  while (! (j = n)) { y := 2 * y; j := j + 1 }
  n := y
}
else {
  while (true) do skip
}
q := 0; r := 0; j := 0; y := 0
```

Award 1 mark for correctly stating the input/output variable; 2 marks for a mostly correct program; 1 mark for the infinite loop when the output is undefined; and 1 mark for setting all auxiliary variables to zero at the end.

** 10. Construct a bijection $\mathbb{Z} \times \mathbb{Z} \xrightarrow{\cong} \mathbb{N}$. Prove that it is a bijection by constructing its inverse, and show that it is indeed an inverse.

Solution

We construct a bijection $g : \mathbb{Z} \times \mathbb{Z} \xrightarrow{\cong} \mathbb{N}$ as

$$\mathbb{Z} \times \mathbb{Z} \xrightarrow{\beta \times \beta} \mathbb{N} \times \mathbb{N} \xrightarrow{\phi} \mathbb{N}$$

The inverse $g^{-1} : \mathbb{N} \xrightarrow{\cong} \mathbb{Z} \times \mathbb{Z}$ is given by

$$\mathbb{N} \xrightarrow{\phi^{-1}} \mathbb{N} \times \mathbb{N} \xrightarrow{\beta^{-1} \times \beta^{-1}} \mathbb{Z} \times \mathbb{Z}$$

For any $(a, b) \in \mathbb{Z} \times \mathbb{Z}$

$$\begin{aligned}(\beta^{-1} \times \beta^{-1})(\phi^{-1}(\phi((\beta \times \beta)(a, b)))) &= (\beta^{-1} \times \beta^{-1})(\phi^{-1}(\phi(\beta(a), \beta(b)))) \\ &= (\beta^{-1} \times \beta^{-1})(\beta(a), \beta(b)) \\ &= (\beta^{-1}(\beta(a)), \beta^{-1}(\beta(b))) \\ &= (a, b)\end{aligned}$$

For any $n \in \mathbb{N}$, write $(n_1, n_2) = \phi^{-1}(n)$ and calculate

$$\begin{aligned}\phi((\beta \times \beta)((\beta^{-1} \times \beta^{-1})(\phi^{-1}(n)))) &= \phi((\beta \times \beta)((\beta^{-1} \times \beta^{-1})(\phi^{-1}(n)))) \\ &= \phi((\beta \times \beta)((\beta^{-1} \times \beta^{-1})(n_1, n_2)))) \\ &= \phi((\beta \times \beta)(\beta^{-1}(n_1), \beta^{-1}(n_2))) \\ &= \phi((\beta \times \beta)(\beta^{-1}(n_1), \beta^{-1}(n_2))) \\ &= \phi(\beta(\beta^{-1}(n_1)), \beta(\beta^{-1}(n_2))) \\ &= \phi(n_1, n_2) \\ &= n\end{aligned}$$

** 11. Show that the predicate

$$U = \{\ulcorner S_1 \urcorner \mid \text{for all } k \leq 100 \text{ it is true that } [\![S_1]\!]_{\mathtt{x}}(k) + 1 = [\![S_1]\!]_{\mathtt{x}}(k+1)\}$$

is semi-decidable.

Solution

For each $k = 0, \ldots, 100$ simulate S_1 on input k, and then S_2 on input $k+1$. If these simulations terminate, check whether the output of the first plus one is equal to the output of the second. If not, return false and halt. Otherwise, after all the simulations are over return 1. Because this is a semi-decision procedure, the simulations need not terminate.

** 12. Show that if the predicates $A \subseteq \mathbb{N}$ and $B \subseteq \mathbb{N}$ are decidable then their *symmetric difference*, i.e. the set

$$A \oplus B = \{x \in \mathbb{N} \mid (x \in A \wedge x \notin B) \vee (x \notin A \wedge x \in B)\} = (A \cup B) - (A \cap B)$$

is also decidable.

Solution

Let S decide A wrt x, and let T decide B wrt y. Assume there are no variables in common in S and T. The program

```
x := z; S;
y := z; T;
if (x = 0 /\ y = 1) { z := 1 }
else if (x = 1 /\ y = 0) { z := 1 }
else { z := 0 }
x := 0; y := 0
```

decides $A \oplus B$ wrt z.

Award 1 mark for recognising there are programs that decide A and B; 1 mark for realising that you need to run both of them; and 1 mark for a correct solution that sets all auxiliary variables to zero.

*** 13. Show that the predicate

$$V = \{\langle \ulcorner S_1 \urcorner, \ulcorner S_2 \urcorner \rangle \mid \text{there exists } k \in \mathbb{N} \text{ such that } [\![S_1]\!]_{\mathtt{x}}(k) + 1 = [\![S_2]\!]_{\mathtt{x}}(k+1)\}$$

is undecidable. (The use of $=$ above means that both sides of the equation must be defined, and equal.)

Solution

Construct the code transformation F : **Stmt** $\times \mathbb{N} \to$ **Stmt** $\times$ **Stmt** given by

$$F(\mathtt{D}, n) = (\mathtt{x := n;\ D;\ x := 0}, \mathtt{x := 1})$$

It is easy to argue that (the reflection of) this code transformation is computable. Let the first component of that pair be $S_{\mathtt{D},\,\mathtt{n}}$. We have

$$\langle \ulcorner S_{\mathtt{D},n} \urcorner, \ulcorner \mathtt{x := 1} \urcorner \rangle \in V \iff \langle \ulcorner \mathtt{D} \urcorner, n \rangle \in \mathsf{HALT}$$

As the latter is not decidable, neither is the former.

Award 1 mark for recognising that a reduction is the most appropriate proof method; 3 marks for constructing the reduction, and arguing that it is computable; and 1 mark for correctly stating the reduction property in this particular instance.