Basic notions of programming - statespace, problem, program

1 Statespace

Problems are about data, programs also deal with data. The typevalue-set of a data is a set containing all possible values of the given data.

Definition: *Let $A_1, \ldots, A_n$ (where $n \in \mathbb{N}^+$) be typevalue-sets and let $v_1, \ldots, v_n$ be unique labels (more precisely: variables) identifying the typevalue-sets. The set $\{v_1\colon a_1, \ldots, v_n\colon a_n\}$ (where $\forall i \in [1..n] : a_i \in A_i$) constructed from the variables and sets mentioned beforehand, is called* state.

In case a statespace contains only one component, the notation a_1 can be used instead of $\{v_1 : a_1\}$ to denote a state.

Definition: *Let $A_1, \ldots, A_n$ (where $n \in \mathbb{N}^+$) be typevalue-sets and let $v_1, \ldots, v_n$ be unique labels (more precisely: variables) identifying the typevalue-sets. The set of all possible $\{v_1\colon a_1, \ldots, v_n\colon a_n\}$ variables (where $\forall i \in [1..n] : a_i \in A_i$) constructed from the variables and sets mentioned beforehand, is called statespace and denoted by $(v_1\colon A_1, \ldots, v_n\colon A_n)$.*

$$(v_1\colon A_1, \ldots, v_n\colon A_n) ::= \left\{ \{v_1\colon a_1, \ldots, v_n\colon a_n\} \,|\forall i \in [1..n] : a_i \in A_i \right\}$$

Definition: *The labels (variables) of the statespace $A = (v_1{:}A_1, \ldots, v_n{:}A_n)$ are considered as $v_i\colon A \to A_i$ functions, where $v_i(a) = a_i$ for any $a = \{v_1{:}a_1, \ldots, v_n{:}a_n\}$ state.*

Definition: *Let $A = (v_1\colon A_1, \ldots, v_n\colon A_n)$ and $B = (u_1\colon B_1, \ldots, u_n\colon B_m)$ any statespaces ($n, m \in \mathbb{N}^+$ and $m \leqslant n$). We say that statespace B is a subspace of statespace A ($B \leq A$), if there exist an injective function $\varphi\colon [1..m] \to [1..n]$, where $\forall i \in [1..m] : B_i = A_{\varphi(i)}$.*

2 Problem

Definition: *Let A by an arbitrary statespace. Any $F \subseteq A \times A$ relation is called problem.*

The definition of problem is based on our approach, where we consider the problem as a mapping from the statespace to the same statespace. In this relation, for every element of the statespace we determine the goal states we intend to get starting from the given element of the statespace.

3 Program

Definition: *Let A be the so-called base state space and $\bar{A}$ be the set of all states which belong to the state spaces B whose subspace is A, i.e. $\bar{A} = \bigcup_{A \leq B} B$. $\bar{A}$ does not contain the state* ***fail****. The relation $S \subseteq A \times (\bar{A} \cup \{\textbf{fail}\})^{**}$ is called* ***program*** *over A, if*

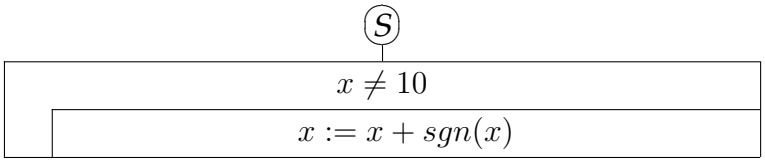
1. $\mathcal{D}_S = A$
2. $\forall a \in A$: $\forall \alpha \in S(a) : |\alpha| \geq 1$ *and* $\alpha_1 = a$
3. $\forall \alpha \in \mathcal{R}_S : (\forall i \in \mathbb{N}^+ : i < |\alpha| \rightarrow \alpha_i \neq fail)$
4. $\forall \alpha \in \mathcal{R}_S : (|\alpha| < \infty \rightarrow \alpha_{|\alpha|} \in A \cup \{fail\})$

We defined the program as a relation that maps from the statespace A to the set of finite and infinite sequences containing elements of $\bar{A} \cup \{fail\}$ such that

- The program assigns at least one sequence to every state in the statespace A.
- Every sequence that is assigned to a state a by the program, begins with the state a (the same state to which it is assigned).
- In case there is a $fail$ state in a sequence (where the sequence represents a possible execution of the program, i.e. it is an element of $\mathcal{R}_S$), it has to be the last element of the sequence (indicating a failure).
- In case an execution of a program is finite, the last element of the sequence that represents the execution has to be a normal state (element of the statespace A, not an element of $\bar{A}$) or the special $fail$ state.

In the following examples we consider programs in order to analyse the sequences assigned to the states of the statespace by the programs.

Exercise 1: *Let* $H = \{a \in \mathbb{Z} \mid a \geqslant -5\}$
$A = (x\colon H)$



In the program, sgn denotes the signum function. For simplicity, let us denote a state of the statespace by a, instead of using the official notation $\{x{:}a\}$.
As we know, any sequence assigned to the state a by the program S begins with the same state it is assigned to, a. We also know, that the loop will stop if its loopcondition becomes $false$, that is when the value of variable x is 10 (in other words, when we are in the state $\{x{:}10\}$).
Our program is a loop. Whenever the loopcondition holds, we execute the body of the loop.

The loopbody is the assignment $x := x + sgn(x)$. Since $sgn(x) = -1$ for any x negative number, we decrease the value of x by 1 in the loopbody in case the value of x is less than zero. Notice, that -5 is the smallest possible value of x. The program $x := x + sgn(x)$ will abort executing from the state where $x = -5$.

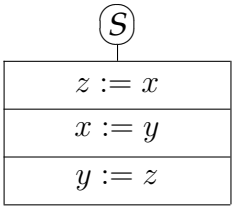
- *Write down the sequences assigned to the states 4, 13, -2, 0 and 10 by the program S.*

The finite sequence $< 4, 5, 6, 7, 8, 9, 10 >$ assigned to the state 4. The program assigns a finite sequence to the state -2 as well: $< -2, -3, -4, -5, fail >$, this sequence aims to express that the program terminates starting from the state -2, but decreasing the value of x in the state -5 causes a failure.

$S(13) = \{< 13, 14, 15, 16, \dots >\}$
$S(0) = \{< 0, 0, 0, 0, \dots >\}$
$S(10) = \{< 10 >\}$

Notice that it does matter what the base-statespace of a program is. Also remember, that due to the definition of program, every sequence assigned to a state begins with the same state it is assigned to, and (in case the sequence is finite) the last element of a sequence is a state of the base-statespace A, or the special $fail$ state. So, the definition of program allows for sequences to contain intermediate states from any statespace B where A is a subspace of B. In practice it means, that during the execution of the program, using of auxiliary variables is allowed; these variables are created and destroyed during the execution of the program, and their corresponding values are recorded while they are existing.

Exercise 2: *Consider the following program and let us suppose that the base-statespece of S is $A = (x{:}\mathbb{Z}, y{:}\mathbb{Z})$. It makes sense, as usually we use this program to solve the problem where we want to swap to integers x and y, this is why variable z (its type is also integer) can be considered as an auxiliary variable.*



Let us write down the sequence that is assigned to the state $\{x{:}3, y{:}8\}$ by this program. Remember, that the sequence has to begin with the same state it is assigned to, and the last element of the sequence (in case it is a finite sequence) is from statespace A or (in case the program terminates due to a failure) is the $fail$ state. During the execution of the program we record the actual values that belong to the variable z, which means the intermediate states of the sequence are elements of the statespace $(x{:}\mathbb{Z}, y{:}\ \mathbb{Z}, z{:}\mathbb{Z})$. As we know, the program $y := z$ sets the value of y to the value that belongs to z and does not change the value of x and z. Our S program is a sequence of three assignments and the following sequence is assigned to the state $\{x{:}3, y{:}8\}$:

$$< \{x{:}3, y{:}8\}, \{x{:}3, y{:}8, z{:}3\}, \{x{:}8, y{:}8, z{:}3\}, \{x{:}8, y{:}3\} >$$

We will give the formal definition of solution in the next chapter. The aim of the following example is to illustrate, when we want to say that a program solves a given problem.

***Example* 1:** *Consider the following problem: given a natural number n, our task is to find one of its positive divisors, d. The statespace of the problem is $A = (n{:}\mathbb{N}, d{:}\mathbb{N})$*

For example, in the case of the "input" state $\{n{:}10, d{:}7\}$, the only good states are the elements of the statespace where the value of n is not changed and d is a divisor of n. So the problem (remember: a problem is a relation) assigns four states to the state $\{n{:}10, d{:}7\}$, namely $\{n{:}10, d{:}1\}, \{n{:}10, d{:}2\}, \{n{:}10, d{:}5\}$ and $\{n{:}10, d{:}10\}$.
What do we expect from a program to solve this given problem? If we consider the state $\{n{:}10, d{:}7\}$, we want our program not to abort but surely terminate and end up in any of the listed four states starting its execution from the state $\{n{:}10, d{:}7\}$. In general, we expect the program to surely terminate and end up in a "good" state (that is assigned to the given state by the problem) starting its execution from any state of the domain of the problem. In practice, we say that our program finds a divisor of 10 if the value of d is 1, 2, 5 or 10 after the execution of the program. In case our program is nondeterministic, and it ends up in a state where the value of d is either 2 or 10, but never is 1 or 5, we can say the program found a divisor of 10 (dispite the fact, that there is no execution of the program that yields the divisor 5). However, if there is an infinite execution assigned to the state $\{n{:}10, d{:}7\}$ by the program, or the program can terminate in a state where d is not a divisor of n, then the program does not solve the problem.

Informally, we say that a program solves the problem if

- for any state in the domain of the problem, it is guaranteed that the program terminates faultlessly starting its execution from the given state
- for any state in the domain of the problem, staring its execution there, the program ends up in "good" states (that are assigned to the given state by the problem)