Verifikation nebenläufiger Programme
Wintersemester 2004/05

Ulrich Hannemann   Jan Bredereke

1 Sequential Transition Diagrams and Systems

1.1 Sequential Transition Diagrams

Transition diagrams describe the control structure of a program in terms of locations and transitions. A location represents the program counter which indicates the next instruction to be executed. A transition describes the effect of the execution of an instruction in terms of the new value of the program counter. The execution of an instruction itself is described in terms of a state transformation, where a state represents the contents of the memory. A state transformation consists of assigning to some memory cells the results of some operation performed on values read from the memory.

Intuitively the control structure of a program can be pictured as a labelled directed graph. The nodes in the graph are referred to as *locations*. Directed edges connect these nodes. *The entry node is a distinguished location where computation starts.* The similarly distinguished *exit* node has no outgoing edges. Each edge is labelled by an instruction of the form $c \to f$, where c denotes a total boolean condition or state function, also called *predicate*, and f denotes a total state transformation (such boolean functions will also be denoted by b):

$$\begin{tikzcd} l \arrow[rr, "c \to f"] & & l' \end{tikzcd}$$

The intuitive meaning of a transition is that execution may proceed from a location l only if the current state at l satisfies the boolean condition c. In such a case we say that the transition is *enabled*. The execution of a transition then consists of applying the state transformation f to the current state. Subsequently, the execution moves to l'.

We proceed with giving a more formalised account of transition diagrams and their semantics. First we introduce formally the semantical notions of predicates and state transformations.

Definition 1.1 (Predicate, state transformation) Given a set of states Σ, with typical element σ, a predicate is a total (boolean) function assigning truth values to states. The set of predicates, with typical element φ, will be denoted by $\Phi \overset{\text{def}}{=} \Sigma \to \mathit{Bool}$, where Bool denotes the domain of truth values $\{\mathit{tt}, \mathit{ff}\}$. A state transformation f is simply an element of $\Sigma \to \Sigma$, i.e., the set of total functions from Σ to Σ. $\square$

For a predicate φ we introduce the notation $\models \varphi(\sigma)$ to indicate *satisfaction* of φ in σ, defined by $\varphi(\sigma) = \mathit{tt}$ (sometimes also the notation $\sigma \models \varphi$ will be used). Similarly, we use the notation $\models \varphi$ to express *validity* of φ, defined by: $\models \varphi(\sigma)$ for all states σ. We often use the phrases "$\varphi(\sigma)$ *holds*" and "φ *is valid*" to express, respectively, that φ is satisfied in σ and that $\models \varphi$ holds.

We have the usual lifting of the boolean operations to predicates: For example the conjunction of two predicates φ and ψ, denoted by $\varphi \wedge \psi$, is defined by $\models (\varphi \wedge \psi)(\sigma)$ if $\models \varphi(\sigma)$ and $\models \psi(\sigma)$.

Transition diagrams are mathematically defined as follows:

Definition 1.2 (Transition diagram) A transition diagram is a tuple (L, T, s, t), where L is a finite set of locations, with typical element l, T is a finite set of triples $(l, c \to f, l')$ called *transitions* with $l, l' \in L$, $c : \Sigma \to \mathit{Bool}$ and $f : \Sigma \to \Sigma$ total functions where Σ denotes a set of states, $s \in L$ is a distinguished location called *entry* location, and $t \in L$ is a distinguished location called *exit* location such that for no label l and instruction $c \to f$ we have that $(t, c \to f, l) \in T$. $\square$

A transition $(l, a, l') \in T$, with $a = c \to f$, we will also denote by $l \xrightarrow{a} l'$ or just $l \to l'$. Simpler variants of the general form for instructions will often be used such as c standing for a test with no state transformation, or f standing for $\mathit{true} \to f$, where true denotes the predicate which assigns the truth value tt to every state.

In concrete examples of transition diagrams a state will be a total function assigning values to variables, i.e., $\Sigma \overset{\text{def}}{=} \mathit{VAR} \to \mathit{VAL}$, where VAR, with typical elements $x, y, z, \ldots$, denotes a set of variables, and VAL denotes the domain of values, i.e., for $y \in \mathit{VAR}$ one has that $\sigma(y)$ denotes the value of variable y in state σ, and this value belongs to VAL. A state transformation will then denote an assignment of the form $(y_1, \ldots, y_n) := (g_1, \ldots, g_n)$, or $\bar{y} := \bar{g}$, for short, where $g_i : \Sigma \to \mathit{VAL}$ denotes a total semantic function for computing a value from VAL, called *value* expression, y_i is a variable belonging to VAR, $i = 1, \ldots, n$, and the variables y_i in $\bar{y}$ are different from each other. Note that in general it is not the case that $\mathit{VAR} = \{y_1, \ldots, y_n\}$; usually only a few variables of VAR are changed in such a state transformation.

The semantics of an assignment is given by the following definition:

Definition 1.3 (Semantic assignment) First we define the *variant* of a state σ with respect to a sequence of distinct program variables $\bar{x} = (x_1, \ldots, x_n)$ and a corresponding sequence of values $\bar{d} = (d_1, \ldots, d_n)$, denoted by $(\sigma : \bar{x} \mapsto \bar{d})$,

$$(\sigma : \bar{x} \mapsto \bar{d})(y) = \begin{cases} d_i & \text{if } y \equiv x_i, \\ \sigma(y) & \text{if } y \not\equiv x_i, \text{ for } i = 1, \ldots, n, \end{cases}$$

where $\equiv$ denotes syntactic equality. Semantically an assignment $\bar{y} := \bar{g}$ denotes the state transformation: $(\bar{y} := \bar{g})(\sigma) = (\sigma : \bar{y} \mapsto \bar{d})$, where $\bar{d} = (g_1(\sigma), \ldots, g_n(\sigma))$. $\square$

Since f and g are total functions, this definition does not model the effect of executing undefined or partially defined operations such as $x := 1/0$ or $x := 0/y$.

Now we are able to define formally the semantics $\mathit{Comp}\ [\![P]\!]$ of transition diagrams: For a transition diagram P, an *execution sequence (or computation) η starting in σ_0* is a sequence of *configurations*

$$\eta : \langle l_0; \sigma_0 \rangle \longrightarrow \langle l_1; \sigma_1 \rangle \longrightarrow \langle l_2; \sigma_2 \rangle \longrightarrow \ldots$$

such that $l_0 \equiv s$, and:

- For each *computation step* $\langle l_i; \sigma_i \rangle \longrightarrow \langle l_{i+1}; \sigma_{i+1} \rangle$ in this sequence there is a transition of P of the form

$$\begin{tikzcd} l_i \arrow[rr, "c \to f"] & & l_{i+1} \end{tikzcd}$$

  such that $c(\sigma_i) = tt$ and $\sigma_{i+1} = f(\sigma_i)$. Note that our transition diagrams may be nondeterministic since two different transitions may depart from a single location with conditions that are not necessarily disjoint.

- This sequence cannot be extended; i.e., if the sequence is finite, its last configuration has no possible successors.

Consequently, such an execution sequence satisfies one of the following cases:

1. The sequence is finite, and its last configuration is of the form $\langle t; \sigma \rangle$. Such a sequence is called a *terminating* sequence, and we define the value of the sequence η as:
$$val(\eta) \overset{def}{=} \sigma, \text{ with } \sigma \in \Sigma.$$

2. The sequence is finite, but its last configuration is of the form $\langle l; \sigma \rangle$, $l \not\equiv t$. This must therefore be a deadlock state such that for no transition departing from l there exists a condition c with $c(\sigma) = tt$. Such a sequence is called a *failing* sequence, and we define its value as:
$$val(\eta) \overset{def}{=} fail.$$

3. The sequence is infinite. Such a sequence is called *divergent*, and we define its value as:
$$val(\eta) \overset{def}{=} \perp.$$

The values *fail* and $\perp$ are special symbols used to denote failure and divergence, respectively, and are not contained in Σ.

Denote by $Comp\ [\![P]\!]\,\sigma$ the set of all computations of P starting with the initial state σ. We define the *meaning* of the transition diagram P as a function $\mathcal{M}\,[\![P]\!]$:
$$\mathcal{M}\,[\![P]\!]\,\sigma \overset{def}{=} \{val(\eta) \mid \eta \in Comp\ [\![P]\!]\,\sigma\}.$$
Thus the meaning of a transition diagram P is a function which for a given initial state σ gives the set of all possible outcomes, including the possibility of failing and of divergent computations ($fail \in \mathcal{M}\,[\![P]\!]\,\sigma$ or $\perp \in \mathcal{M}\,[\![P]\!]\,\sigma$).

It is customary to write $\mathcal{M}\,[\![P]\!]\,(\sigma)$ in order to emphasise that $\mathcal{M}$ is a mapping which for each transition system P yields a function $\mathcal{M}\,[\![P]\!]$ describing the initial-final state, failure and divergent behaviour of the computations of P:
$$\mathcal{M}\,[\![P]\!] : \Sigma \to 2^{\Sigma \cup \{\perp, fail\}}.$$

1.2 Transition Systems

The *val* function used to define the $\mathcal{M}[\![P]\!]$ semantics ignores the actual locations of execution sequences. The only aspect that remains is whether the last location is the exit location or not, and whether the computation is infinite. Therefore the meaning $\mathcal{M}[\![P]\!]$ of a transition diagram P does not depend on the particular names of the locations in a transition diagram. Informally, two transition diagrams are equivalent if they are the same up to a renaming of the locations, where the renaming should respect the start and exit locations. Assume that we have two transition diagrams $P \equiv (L, T, s, t)$ and $P' \equiv (L', T', s', t')$, with the same state space Σ. A renaming that transforms the names of the nodes of P into those of P' can be formalised as a *bijection* ϕ between L and L' that respects the entry and exit locations and the T relation:

Definition 1.4 (Equivalence of transition diagrams) We call two transition diagrams, $P \overset{\text{def}}{=} (L, T, s, t)$ and $P' \overset{\text{def}}{=} (L', T', s', t')$, *equivalent* , if there exists a bijection $\phi : L \to L'$ such that

- $s' = \phi(s)$,
- $t' = \phi(t)$, and
- $(l, a, m) \in T$ if and only if $(\phi(l), a, \phi(m)) \in T'$. □

Observe that renaming is an equivalence relation.

Definition 1.5 (Transition systems and programs) A *transition system* is an equivalence class of transition diagrams. Such an equivalence class is also called a *program.* □

Let P be a program, i.e., an equivalence class of transition diagrams. As remarked above, all transition diagrams in this class have the same meaning. Hence we can define the meaning of the program by taking the meaning of an arbitrary element from the equivalence class.

In the sequel, when dealing with transition systems we will often use transition diagrams that represent these equivalence classes instead. We say for instance: "let program P be represented by (L, T, s, t)", or even "let program $P \overset{\text{def}}{=} (L, T, s, t)$". One should check in such cases that the results are independent from the choice of the representatives. This is usually left to the reader.

Example 1.6 Consider program $P = (L, T, s, t)$ with

- $L = \{s, l, t\}$ and
- $T = \{(s, \mathit{true} \to f_0, l), (l, c_1 \to f_1, l), (l, c_2 \to f_2, t)\}$, where

  $c_1(\sigma) = \mathit{tt}$ iff $\sigma(y) > 0$, $c_2(\sigma) = \mathit{tt}$ iff $\sigma(y) = 0$,

  $f_0(\sigma) = (\sigma : y, z \mapsto \sigma(x), 0)$,

  $f_1(\sigma) = (\sigma : y, z \mapsto \sigma(y) - 1, \sigma(y) + \sigma(z))$, and

  $f_2(\sigma) = \sigma$,

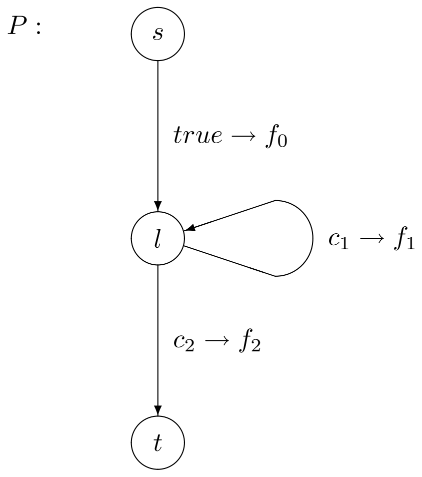

Figure 1: A simple transition diagram.

where $\sigma \in \Sigma$, with $\Sigma \overset{\text{def}}{=} \{x, y, z\} \to \mathcal{Z}$, and $\mathcal{Z}$ denoting the set of integers.

Graphically, P can be represented as the transition diagram in Figure 1.

It is easy to check that, if $\sigma(x) \geq 0$, then $\mathcal{M} [\![P]\!] \, \sigma = \{(\sigma : y, z \mapsto 0, \sum_{i=0}^{\sigma(x)} i)\}$ and that, if $\sigma(x) < 0$, then $\mathcal{M} [\![P]\!] \, \sigma = \{fail\}$. $\square$

These assignments f_1 and f_2 we abbreviate, respectively, to $(y, z) := (x, 0)$ and $(y, z) := (y-1, y+z)$. Similarly, condition c_1 will be abbreviated to $(y > 0)$ and c_2 to $(y = 0)$. Parentheses are dropped whenever this does not lead to confusion.

Now P can be represented as in Figure 2 below, using the above abbreviations and the additional ones introduced in Section 1.1. Whenever this is convenient, we will stick to these abbreviations.

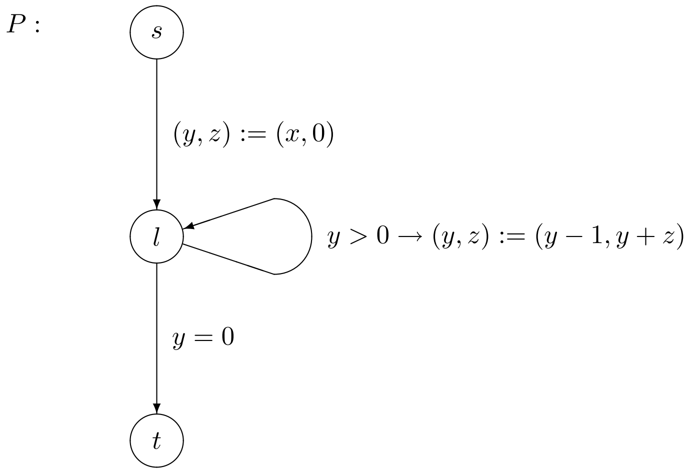

Figure 2: A more convenient notation for the simple transition diagram.