A Formal Semantics of Statestep Tables

Michael Breen

17 August 2005

1 Introduction

What follows is a formal semantics of the Statestep table formats [1], with which it is assumed the reader is already familiar. The constraint table and the transition table are first treated separately; we then define the combination of the two as a finite state machine.

The formalization is based on that of Janicki and Khedri [2], who propose a general semantics for all tables used in software engineering. However, assumptions in their treatment are violated by both the constraint and transition tables. We therefore present a separate semantics here, while employing their idea of composing the information provided by individual table cells.

2 Definitions and Notational Conventions

The cardinality of a set A is written $|A|$ and $\mathcal{P}(A)$ denotes its powerset, i.e., the set of all subsets of A (including the empty set $\emptyset$). The notation $F : A \leftrightarrow B$ introduces a binary relation F, over sets A, B, i.e., $F \subseteq A \times B$. Its domain and range are defined, respectively, as

$$dom(F) = \{a \mid (\exists b \in B \bullet (a, b) \in F)\}$$

$$ran(F) = \{b \mid (\exists a \in A \bullet (a, b) \in F)\}$$

Given $A' \subseteq A$ and $B' \subseteq B$, we also define domain restriction and range restriction:

$$A' \lhd F = \{(a, b) \mid (a, b) \in F \wedge a \in A'\}$$

$$F \rhd B' = \{(a, b) \mid (a, b) \in F \wedge b \in B'\}$$

$F : A \nrightarrow B$ distinguishes a binary relation F as a partial function, i.e., for each $a \in dom(f)$ there is a unique $b \in B$ such that $(a, b) \in F$. A (total) function, denoted $F : A \rightarrow B$, is a partial function with the additional requirement that $dom(F) = A$. This function F is said to be surjective if $ran(F) = B$ and injective if $(a_1, b), (a_2, b) \in F \Rightarrow a_1 = a_2$. A bijection is a function which is both injective and surjective. An example of a bijection is the identity relation, defined for a set A as:

$$1_A = \{(a, a) \mid a \in A\}$$

We define a set, D, of non-null sets of interest (these sets are not necessarily disjoint), called attribute sets, indexed by a set of attributes T, e.g., $T = \{Tray, Mode, \dots\}$; $D_{Tray} = \{open, closing, \dots\}$. If $W = \{W_1, W_2, \dots, W_{|W|}\} \subseteq T$ then D_W denotes

the Cartesian product of the attribute sets indexed by W, i.e., $D_W = D_{W_1} \times \ldots \times D_{W_{|W|}}$.

In general, an n-ary relation $F \subseteq D_W$ is a set of tuples. A tuple $(w_1, w_2, \ldots, w_{|W|})$ may be regarded as a function $f : \{W_1, \ldots, W_{|W|}\} \rightarrow W_1 \cup \ldots \cup W_{|W|}$ with $f(W_i) = w_i \in D_{W_i}$, $i = 1, \ldots, |W|$. For tuples, a further restriction operator is defined: if $W' \subseteq W$, $f \in F \subseteq D_W$ then $f|_{W'} = g \in D_{W'}$ where $g(W_i) = f(W_i)$ for all $W_i \in W'$. Similarly, for relations, the projection operation is defined:

$$\pi_{W'}(F) = \{g \mid (\exists f \in F \bullet g = f|_{W'})\} \subseteq D_{W'}$$

For any $E \subseteq D_U$,$F \subseteq D_V$, $(U, V \subseteq T)$, the natural join operation is defined:

$$E \otimes F = \{a \mid a \in D_{U \cup V} \wedge a|_U \in E \wedge a|_V \in F\}$$

(which for $U = V$ reduces to intersection). For the formalization of the transition table below, it is convenient to define relations over an arbitrary number of attributes but which are structured as binary relations, e.g., $E : D_U \leftrightarrow D_V, F : D_X \leftrightarrow D_Y$ – where neither U and V nor X and Y are necessarily disjoint. For binary relations, we therefore refine the definition of natural join:

$$\begin{array}{rcl} E \otimes F & = & \{(a, b) \mid a \in D_{U \cup X} \wedge b \in D_{V \cup Y} \\ & & \wedge (a|_U, b|_V) \in E \wedge (a|_X, b|_Y) \in F\} \end{array}$$

However, all operands of the natural join operation (including relations defined over two attributes) are treated as general, n-ary relations unless explicitly typed as binary, i.e., unless introduced using '$\leftrightarrow$' or another arrow symbol.

3 Constraint Table

Take $L \subseteq T$ (where T is our set of attributes and D the corresponding set of attribute sets, as above) and $I = \{1, \ldots, |L|\}$. A constraint table comprises a header $H : I \rightarrow L$ (H is a bijection) and a grid, G, indexed by $I \times K$ for some $K = \{1, \ldots, |K|\}$. The contents of the grid are given by a set of functions $G_i, i \in I$ where $G_i : K \rightarrow \mathcal{P}(D_{H(i)}) - \{\emptyset\}$ (the "don't care" symbol ('*') is of course syntactically equivalent to enumerating all the values in the appropriate set). We distinguish a set of grid cells, satisfying the following conditions:

$$(1, 1) \in P$$

$$\begin{array}{c} \text{if } (i, k) \in P \wedge k < |K| \text{ then either} \\ (i, k + 1) \in P \text{ or } (i + 1, k + 1) \in P \end{array}$$

$$\forall i \in I \bullet \bigcup_{k|(i.k) \in P} G_i(k) = D_{H(i)} \qquad (1)$$

$$\forall (i, j), (i, k) \in P, j \neq k \bullet G_i(j) \cap G_i(k) = \emptyset$$

The constraint table is valid only if it allows the definition of a set P as above. Since we do not allow empty cells in the grid, it is easily seen that P is uniquely defined for any given table (so that shading the cells in P is merely an aid to readability).

We now show how a constraint table is used to describe a relation $S : D_L$. First, we define $S_{i,j} \subseteq D_{H(i)} \times D_{H(j)}$:

$$S_{i,j} = \pi_{\{H(i) \cup H(j)\}}(S)$$

Now, for each $i, j, k \mid (i,k) \in P \wedge j \in I \wedge j > i$, we define a relation

$$S_{i,j,k} = G_i(k) \times G_j(k) \tag{2}$$

and we construct our grid so that

$$S_{i,j,k} = G_i(k) \lhd S_{i,j} \tag{3}$$

Provided each cell in P contains only one value, it follows (from (1) and the definition of domain restriction) that

$$\bigcup_{k|(i,k) \in P} S_{i,j,k} = S_{i,j} \tag{4}$$

The table then defines a relation

$$S' = \bigotimes_{i,j \in I \wedge j > i} S_{i,j}$$

We have taken every two-attribute projection of S and rejoined the resulting relations. This does not necessarily reproduce S but rather $S' \supseteq S$. A simple way of seeing this is to observe that each relation $S_{i,j,k}$ may alternatively be written using a predicate which restricts S to those elements of D_L which satisfy it, i.e., each $S_{i,j,k}$ requires S to be a subset of

$$\{s \mid s = (s_1, \ldots, s_{|I|}) \in D_L \wedge (s_i \in G_i(k) \Rightarrow s_j \in G_j(k))\}$$

However, this does not allow us to specify more complex predicates which cannot be expressed as a composition of binary constraints, e.g., those of the form

$$s_x \in X \wedge s_y \in Y \Rightarrow s_z \in Z$$

$$s_x \in X \Rightarrow s_y \in Y \vee s_z \in Z$$

(where $x, y, z \in I, X \subset D_{H(x)}$, etc.) The set $Q = \{q_1, \ldots, q_{|Q|}\}$ of such additional predicates required to describe S may be enumerated separately from the table. Treating these as relations, i.e., $Q_m = \{s \mid s = (s_1, \ldots, s_{|I|}) \in D_L \wedge q_m\}$, we have

$$S = S' \otimes \bigotimes_{m=1,\ldots,|Q|} Q_m$$

Finally, note that our definition of S above uses only the cells in the grid above the diagonal. Symmetrically, each $S_{i,j}$ may also be derived from the lower left half of the table, i.e., instead of (3), (4) we use the following: for each $i, j, k \mid (j,k) \in P \wedge i \in I \wedge j > i$, we define a relation $S_{i,j,k}$ as in (2) and we construct our grid so that

$$S_{i,j,k} = S_{i,j} \rhd G_j(k)$$

and consequently,

$$\bigcup_{k|(j,k) \in P} S_{i,j,k} = S_{i,j}$$

Obviously, consistency requires that each $S_{i,j}$ given by the upper right part of the table is the same as that given by the lower left part.

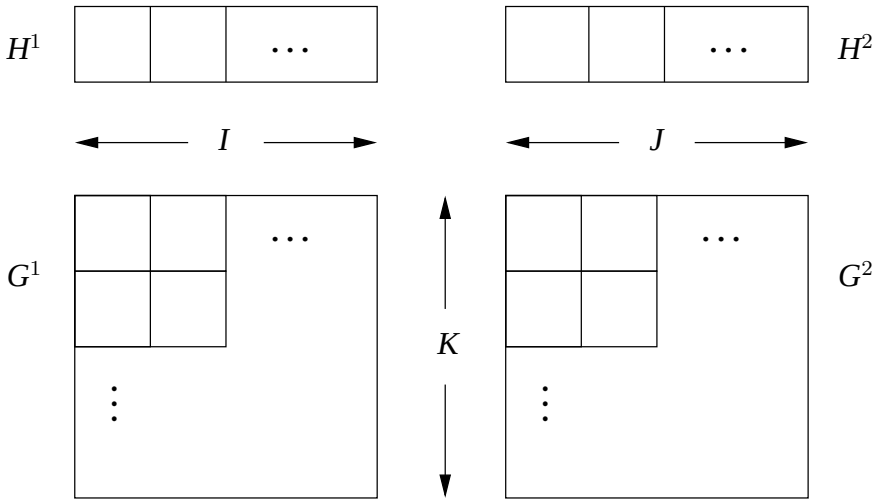

Figure 1: General transition table (alternative layout)

4 Transition Table

To simplify and generalize the formalization of the transition table, we restructure it into the form shown in Figure 1. In this diagram, each row of grid G^2 contains the target row of a transition rule. The corresponding row in grid G^1 contains the source row of the same transtion rule and also the event for this rule (G^1 is thus one column wider than G^2). The two headers, H^1 and H^2, identify the variables of the corresponding grid columns.

We take $N \subseteq O \subseteq T$ where, as before, T and D are the attributes and attribute sets respectively, and we define index sets $I = \{1, \ldots, |O|\}$ and $J = \{1, \ldots, |N|\}$. The table comprises two headers, $H^1 : I \to O$ and $H^2 : J \to N$, both of which are bijections, and two grids G^1 and G^2, indexed by $I \times K$ and $J \times K$ respectively, for some $K = \{1, \ldots, |K|\}$. The contents of G^1 are given by a set of functions, $G^1_i, i \in I$ and those of G^2 by $G^2_j, j \in J$ where

$$G^1_i : K \to \mathcal{P}(D_{H^1(i)}) - \{\emptyset\}$$

$$G^2_j : K \to \mathcal{P}(D_{H^2(j)}) - \{\emptyset\} \cup \varphi$$

The symbol φ corresponds to "no change" (represented by '=' in the tables).

A relation, $R : D_O \leftrightarrow D_N$, is then derived from the table as follows. For all $i \in I, j \in J, k \in K$, we define $R_{i,j,k} : D_{H^1(i)} \leftrightarrow D_{H^2(j)}$ as

$$R_{i,j,k} = \begin{cases} G^1_i(k) \times G^2_j(k) & \text{if } G^2_j(k) \neq \varphi \\ G^1_i(k) \times D_{H^2(j)} & \text{if } G^2_j(k) = \varphi \wedge H^1(i) \neq H^2(j) \\ 1_{G^1_i(k)} & \text{if } G^2_j(k) = \varphi \wedge H^1(i) = H^2(j) \end{cases}$$

Now, we can express the relation described by each transition table rule (i.e., each row k in our reformatted table) as

$$R_k = \bigotimes_{j \in J} \bigotimes_{i \in I} R_{i,j,k}$$

and our table relation R is simply the union of these, i.e.,

$$R = \bigcup_{k \in K} R_k$$

We can relax our earlier, simplifying requirement that $N \subseteq O$: for "no change" to be meaningful, we require only that wherever it occurs in G^2, the corresponding attribute must also appear on the other side, i.e.,

$$\forall j \in J \bullet (\varphi \in ran(G^2_j) \Rightarrow \exists i \in I \bullet H^1(i) = H^2(j))$$

Given that, in the extreme case, each row across the two grids of the table can be used to specify a single tuple of the relation, it is easily seen that our general model allows the specification of any $R : D_O \leftrightarrow D_N$ for any N, O. Of course, if $N \cap O = \emptyset$ then it makes little sense to refer to a "transition table" or to use that format of table.

5 State Machine as Constraint and Transition Tables

A finite state machine may be defined as a tuple, $(\Lambda, \Sigma, \delta)$, where Λ is the set of input events, Σ is the set of states, and δ is the transition relation. In the case of a deterministic machine, this will typically be a partial function $\delta : \Lambda \times \Sigma \nrightarrow \Sigma$ (partial because some events will be impossible in some states). This FSM is described using a constraint table (with additional constraints if necessary) to specify a relation $S \subseteq D_L$ and a transition table giving $R : D_O \leftrightarrow D_N$. We take $events \in T$, $events \notin N$, $L = O = N \cup \{events\}$. Then:

$$\Lambda = D_{events}$$

$$\Sigma = \pi_N(S)$$

$$\delta = R \cap (S \times D_N)$$

It is convenient to allow the source states of each transition rule to include some impossible states so that $dom(R) \supseteq S$. As transitions from impossible states may (conceivably) conflict with each other, R itself is not necessarily a partial function, even for a deterministic machine.

Since invalid states must not be reachable, we require $ran(\delta) \subseteq \pi_N(S)$ (a consistency criterion with respect to the constraints). With not only the valid states but also the valid state-event combinations being defined by S, we can therefore write $\delta : S \leftrightarrow \pi_N(S)$. For completeness with respect to the constraints this relation must be total, i.e., $dom(\delta) = S$. For a deterministic state machine, the transition relation is therefore a function $\delta : S \rightarrow \pi_N(S)$.

References

[1] M. Breen, "Experience of using a lightweight formal specification method for a commercial embedded system product line," *Requirements Engineering Journal*, vol. 10, pp. 161–172, 2005.

[2] R. Janicki and R. Khedri, "On a formal semantics of tabular expressions," *Science of Computer Programming*, vol. 39, pp. 189–213, 2001.