Construction of FSMDAs from Behavioural Descriptions

Kunal Banerjee
kunalb@cse.iitkgp.ernet.in

October 11, 2014

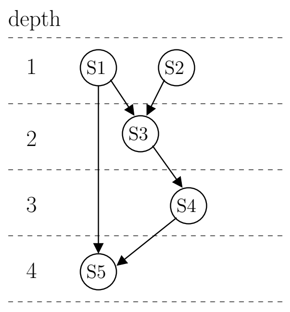
This manual gives a brief elucidation for obtaining FSMDAs from behavioural descriptions (high-level languages), such as C, and how to represent them textually so that they can be fed as inputs to our FSMDA equivalence checker. Note that the content of section 1 of this manuscript is largely borrowed from [2].

1 How to represent behavioural descriptions as FSMDAs conceptually

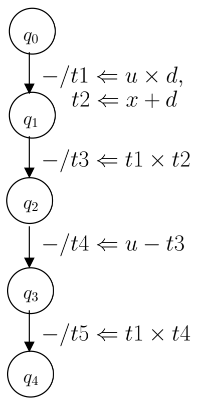
Any (sequential) behaviour consists of a combination of the following three basic constructs: (i) sequences of statements without any bifurcation of control flow, i.e., Basic Blocks (BBs), (ii) if-else constructs, i.e., Control Blocks (CBs), and (iii) loops. Therefore, without any loss of generality, capturing these three constructs in an FSMDA model enables us to effectively represent any sequential behaviour as an FSMDA.

S1: $t1 \Leftarrow u \times d$;
S2: $t2 \Leftarrow x + d$;
S3: $t3 \Leftarrow t1 \times t2$;
S4: $t4 \Leftarrow u - t3$;
S5: $t5 \Leftarrow t1 \times t4$;

(a) A sequence of statements.



(b) Its dependence graph.


q_0
$-/t1 \Leftarrow u \times d,$
$t2 \Leftarrow x + d$
q_1
$-/t3 \Leftarrow t1 \times t2$
q_2
$-/t4 \Leftarrow u - t3$
q_3
$-/t5 \Leftarrow t1 \times t4$
q_4

(c) Corresponding FSMDA.

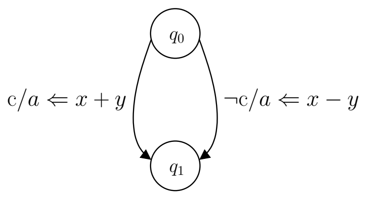
Figure 1: Construction of FSMDA corresponding to a basic block.

A BB consisting of a sequence of n statements $S_1, \ldots, S_n$, can be represented as a sequence of $n+1$ states $q_0, \ldots, q_n$, say and n edges of the form $q_{j-1} \xrightarrow{-/S_j} q_j, 1 \leq j \leq n$, in the corresponding FSMDA. The number of states in the FSMDA, however, can be reduced in the following way (although it is not mandatory).
For a BB, we first construct a dependence graph. The graph consists of a node corresponding to each statement of the BB. There is a directed edge in the dependence graph from the statement $S1$ to the statement $S2$ iff there is one of *read-after-write*, *write-after-read*, and *write-after-write* dependences between $S1$ and $S2$. Naturally, the constructed graph is a directed acyclic graph. Let the height of the dependence graph be h. We now construct an FSMDA of $h+1$ states $q_0, \ldots, q_h$, say and h edges of the form $q_j \twoheadrightarrow q_{j+1}, 0 \leq j \leq h-1$. We place the operations at depth $k, 1 \leq k \leq h$, of the dependence graph in the transition $q_{k-1} \twoheadrightarrow q_k$ of the FSMDA. The condition associated with each transition is *true*. For example, the dependence graph for the BB in Figure 1(a) is depicted in Figure 1(b) and the corresponding FSMDA is given in Figure 1(c).

A CB is of the form: `if(c) then BB1 else BB2 endif`, where `c` is a conditional statement and `BB1` and `BB2` are two basic blocks which execute when `c` is *true* and *false*, respectively. In this case, we

```
if ( c )
  $a \Leftarrow x + y$;
else
  $a \Leftarrow x - y$;
endif
```

(a) An if-else block.



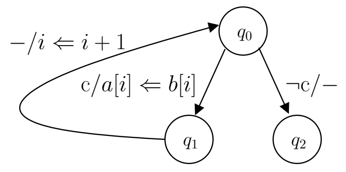
(b) Corresponding FSMDA.

Figure 2: Construction of FSMDA corresponding to a control block.

construct the FSMDAs corresponding to `BB1` and `BB2` first. The FSMDA of the CB is obtained from these two FSMDAs by: (i) merging the start states of two FSMDAs into one start state and the end states of two FSMDAs into one end state, and (ii) the condition `c` is placed as the condition of the first transition of the FSMDA corresponding to `BB1` and the condition ¬`c` is placed in the first transition of the FSMDA corresponding to `BB2`. A sample if-else block is shown in Figure 2(a) and its corresponding FSMDA is shown in Figure 2(b).

```
while ( c )
  $a[i] \Leftarrow b[i]$;
  $i \Leftarrow i + 1$;
endwhile
```

(a) A while block.



(b) Corresponding FSMDA.

Figure 3

For the loop constructs, let us consider only *while* loops without any loss of generality. A while loop is of the form `while(c) BB1 endwhile`. In this case, we first construct the FSMDA M of `BB1`. The FSMDA corresponding to the while loop is obtained by merging the start and the end state of M into one state, q_0 say, and placing the condition `c` in the transitions from q_0. We also need to add a transition from q_0 with condition ¬`c` as the exit path from the loop in the FSMDA as shown in Figure 3.

2 How to represent FSMDAs textually

Now that we have described how behavioural descriptions can be represented as FSMDAs, we explain how FSMDAs are represented textually so that they can be parsed by our equivalence checker.

The first line of the text file describing the FSMDA must contain its name (a string) within double quotes, for example, `"My_FSMDA"`.
Each transition in the FSMDA is represented using the following rule:

```
source_state number_of_transtions condition_1 | data_transformations_1 destination_state_1
                                [ condition_2 | data_transformations_2 destination_state_2 ] ;
```

Thus, the FSMDA shown in Figure 1(c) is represented as follows:

```
q0 1 - | t1 = u * d, t2 = x + d  q1 ;
q1 1 - | t3 = t1 * t2            q2 ;
q2 1 - | t4 = u - t3             q3 ;
q3 1 - | t5 = t1 * t4            q4 ;
```

Note that the operations that occur in the same transition are separated by commas. Moreover, we also refrain from using special symbols, such as '$\Leftarrow$' and '$\times$', which generally appear in our published papers since they are not readily available in standard keyboards; instead we use symbols, such as '`=`' and '`*`' for representing assignment and multiplication, respectively (we use '`==`' for checking equality, similar to C). Furthermore, we have also replaced '`/`' by '`|`' to separate the condition of execution and data transformation of a transition since the symbol '`/`' is also used to represent the division operation and therefore, introduced *reduce/reduce* conflict [1] during parsing.

The FSMDA shown in Figure 2(b) is represented as given below:

```
q0 2  c | a = x + y  q1
     !c | a = x - y  q1 ;
```

While the FSMDA shown in Figure 3(b) is represented as follows:

```
q0 2  c | a[i] = b[i]  q1
     !c | -            q2 ;
q1 1  - | i = i + 1    q0 ;
```

Thus, the symbol '`!`' is used (instead of ¬) to denote negation of a condition; the symbol '`-`' represents *true* when it appears as the condition of execution of a transition whereas, the same symbol represents *no operation* when it appears as the data transformation of a transition.

It is important to note that there can be any number of transitions occurring from a state in an FSMDA, see Figure 1 of reference [3] for example, as long as the condition of execution of each of the transitions is mutually exclusive from each of the other transitions.
The following operation depicts that the value read from port `P` is stored in the variable `v`:
`read( v, P )`
Similarly, the following operation depicts that the expression `e` is written into port `P`:
`write( P, e )`
The users are requested not to confuse these `read` and `write` operations with those of McCarthy's read/write operations [4] that are associated with arrays; the internal representation of array expressions using McCarthy's read/write operations by our equivalence checker is handled in a different manner than the `read` and `write` operations involving ports. Lastly, a final state (with no outgoing edge), `qL` say, in an FSMDA is represented as follows:
`qL 0 ;`
The users are requested to see the sample FSMDAs given in the "Benchmarks" directory (supplied with the FSMDA equivalence checker) for further clarification; they may also contact the author of this manual via email to clarify doubts.

References

[1] Bison: Reduce/Reduce Conflicts. `https://www.gnu.org/software/bison/manual/html_node/Reduce_002fReduce.html`.

[2] C. Karfa, C. Mandal, and D. Sarkar. Formal verification of code motion techniques using data-flow-driven equivalence checking. *ACM Trans. Design Autom. Electr. Syst.*, 17(3):30, 2012.

[3] C. Karfa, D. Sarkar, C. Mandal, and P. Kumar. An equivalence-checking method for scheduling verification in high-level synthesis. *IEEE Trans on CAD of ICS*, 27:556–569, 2008.

[4] J. McCarthy. Towards a mathematical science of computation. In *IFIP Congress*, pages 21–28, 1962.