CS392F: Automatic Software Design

P4: PrologViolet State Diagrams to Prolog Tables

Jianyu Huang Xiaohui Chen

jianyu@cs.utexas.edu xhchen0328@utexas.edu

Clarification

According to Prof. Don Batory in Piazza, we are not required to generate "a jar file of the executables".

Running instructions

Please execute the bash-script in Cygwin.

1. Make the the script executable
chmod u+x run.script.sh
2. Run the script
bash run.script.sh

Directory Structure

```
├──── Main.java                    //XML parser for Part 1
├──── Part1Files                   //Part1 testing files
│    ├──── xxx.pl
│    └──── xxx.state.violet
├──── Part2Files                   //Part2 testing files
│    ├──── chol_verify             //any other fsm database created by hand. (Please refer to our P1)
│    │    ├──── Chol_verify.pl
│    │    └──── chol_verify.state.violet
│    ├──── eatinghabit             //my eating habit
│    │    ├──── eatinghabit.pl
│    │    └──── eatinghabit.state.violet
│    ├──── fsm20                   //fsm20
│    │    ├──── fsm20.pl
│    │    └──── fsm20.state.violet
│    ├──── fsm30                   //fsm30
│    │    ├──── fsm30.pl
│    │    └──── fsm30.state.violet
│    └──── simpleloop              //simple
│        ├──── simpleloop.pl
│        └──── simpleloop.state.violet
└──── run.script.sh                //an idiot-proof shell script that runs your FSM tool
└──── error.txt                    //error message file generated by XML parser
```

Part 1: XML Parser

Our program is based on the Java program that parses a violet state program and interprets it as a category in “Hint” part.

1. We delete “extractNotes” function, since we don’t need the information of “NoteNode” in this assignment.

2. We modify the outputs so as to match the table definitions as the follows:

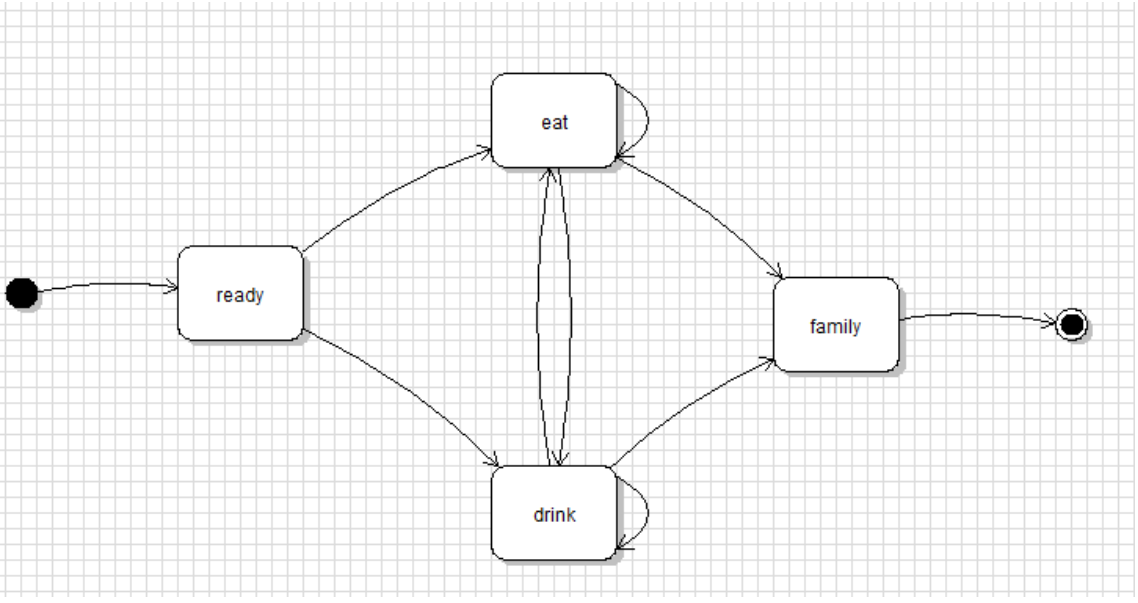
```
dbase(fsm,[node,transition]).

table(node,[nodeid,name,type]).
table(transition,[transid,startsAt,endsAt]).

tuple(node,L):-node(A,B,C),L=[A,B,C].
tuple(transition,L):-transition(A,B,C),L=[A,B,C].
```

3. Since the nodeName for start node and stop node is always “x” in the xml file, we parse the node information and assign the nodeName and state to the start and stop node, apart from the state nodes.

Part 2: Draw the state diagrams

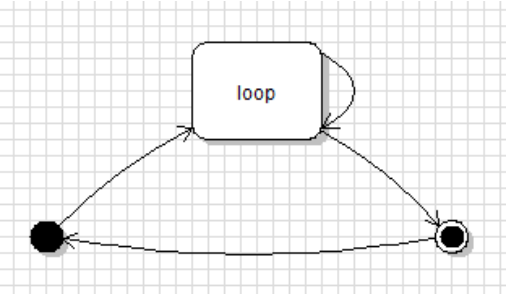
In Part 2, we draw five state diagrams in violet according to the five prolog databases. Each state diagram is in the corresponding folder in Part2File directory. Along with each diagram, there is a corresponding prolog database which is generated by the Main java class we modified in part 1.

1. My eating habit



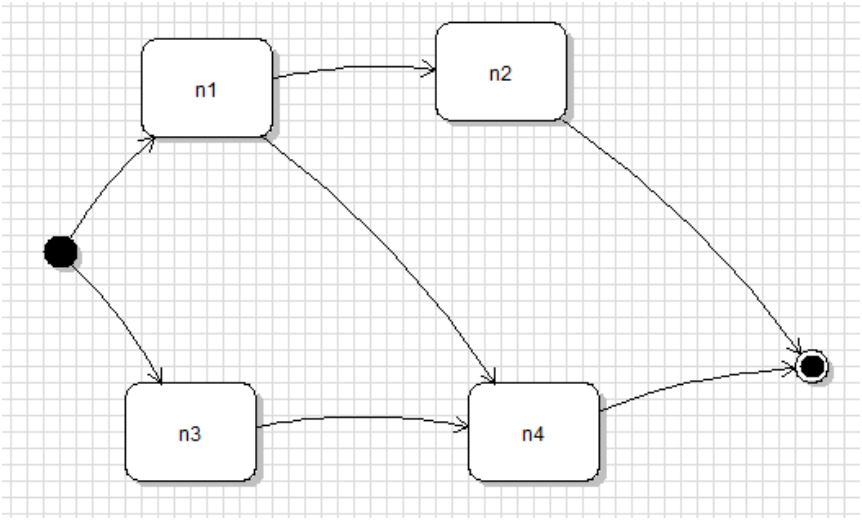
This diagram is straightforward. It is generated according to the prolog database online. It is stored in Part2Files/eatinghabit. The generated prolog database is equivalent to the prolog database.

2. Simple loop



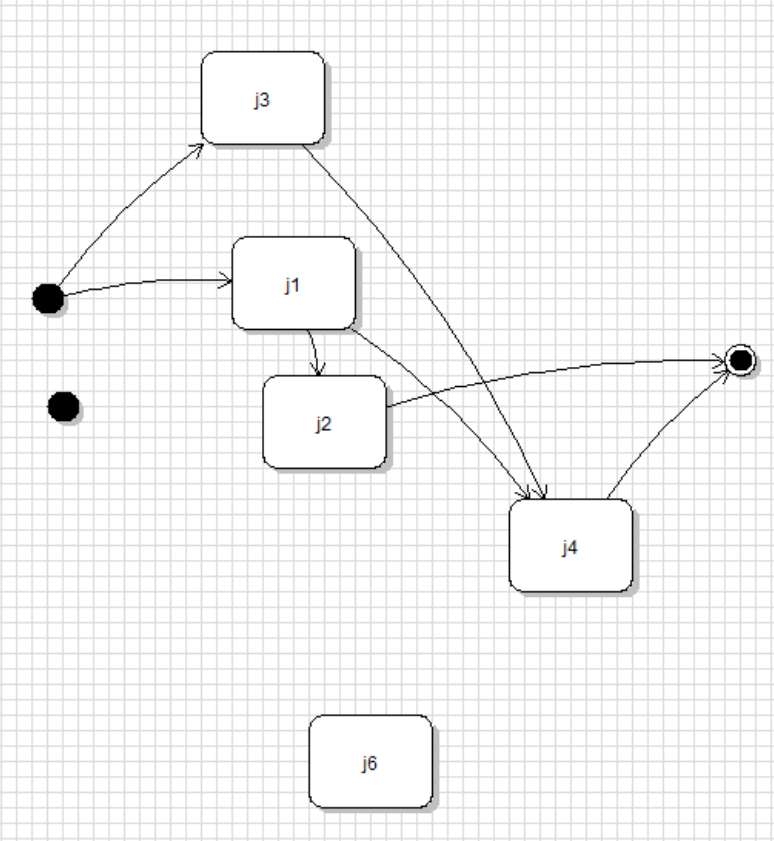
The simple loop is also straightforward. The diagram and the generated prolog database are in Part2Files/simpleloop

3. FSM20



The diagram and the generated prolog database is in Part2Files/fsm20. The generated prolog database is equivalent to the given one.

4. FSM30

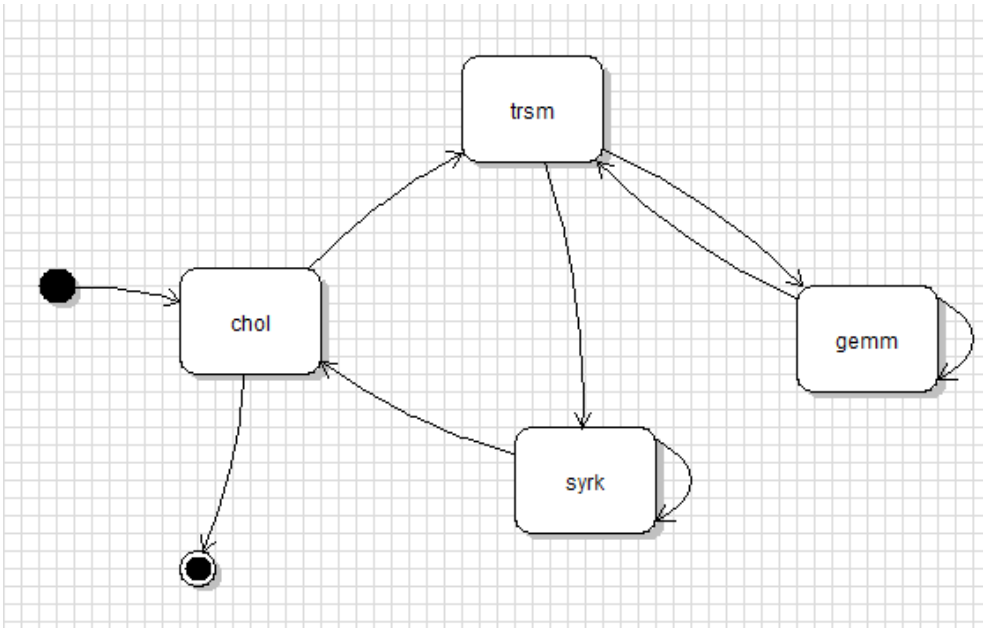


The preconditions for FSM are not satisfied. There are three obvious errors. First, there are two start states and one of them is standalone. Second, node with id n6 is a standalone state. Third, the transition from n3 to n5 is not valid since there is no node with id n5.

In fact, we are technically not able to draw a diagram. The diagram above is the diagram without the last transition. However, violet does not support two start states and so the standalone start state above has type “CircularStateNode”. The Main java class generates the following line for this node: node(, x, state)

Therefore, the Main java class could not generate the correct prolog database.

5. Other fsm database created by hand



In Part2Files/chol_verify folder, Chol_verify.pl is the expected prolog database and the diagram above is the correct state diagram. Therefore, the prolog database generated (chol_vefity.pl) is equivalent to Chol_verify.pl