CheckMate ASCII Interface

CheckMate Team

December 4, 2003

1 Introduction

1.1 Motivation

The standard input to CheckMate is a customized Simulink/Stateflow diagram. CheckMate converts the diagram (i.e., the model of a system to be verified) into a MATLAB data structure, called *polyhedral invariant hybrid automaton* (PIHA), which is a standard hybrid automaton except that *guards* are defined by hyperplanes and *resets* are affine mappings. (The continuous dynamics can be nonlinear.)

In this document, we define a syntax for an ASCII input file (i.e., `CM` file) as an alternative to the Simulink/Stateflow input to describe the model for the system to be verified. CheckMate can import and export a `CM` file automatically. This format also serves as the connection to the hybrid system interchange format (HSIF) being developed in the DARPA MoBIES project.

The following section illustrates the syntax for a simple CheckMate model. Section A defines the complete syntax for the cm file.

1.2 Features of CM File

A cm file is used to model a polyhedra-invariant hybrid automata (PIHA), a modified class of hybrid automata. A PIHA model in cm file has the following feature:

- All the guard conditions are specified using boolean combination of half-spaces. Each half-space is specified using a vector C, a relation and a scalar d. The half-space is the set $\{x|Cx\{<,>\}d\}$. Equal(=) relation is not supported in cm file.
- The continuous state variables are declared as a vector of real-valued variables.
- Concurrent hybrid automata are supported.
- Reset are specified as an linear affine mapping $x' := Ax + b$. Currently only one machine is allowed to generate resets (if the model has more than one machine).

- Differential equations in continuous state space can be : constant rate dynamics/ linear affine dynamics or nonlinear dynamics. Each of them are described in different format together with a keyword.

- Event synchronization are specified using input/output events.

A cm file consists of several parts:

- Variable declaration: declare the number of continuous state variables;

- Parameter declaration: declare the number of nondeterministic parameters;

- Half-spaces declaration: declare all the half-spaces(predicates) used in the model, including CheckMate specific half-spaces analysis_region and initial_continuous_set. (The half-spaces used in analysis_region use keyword 'AR' and those in initial_continuous_set use keyword 'ICS' so that the cm2mdl converter does not create Polyhedron Threshold blocks for them.)

- Machine list: describes a list of concurrent hybrid automata. Each machine consists the following part:
  - A list of local continuous variables
  - A list of locations, which contains continuous dynamics, invariant and a list of out-going transitions.
  - The initial location.

- Analysis_region and initial_continuous_set that are used in CheckMate.

2 Using the ASCII Interface

This document explains how to model a hybrid system given as a collection of modes with their corresponding differential equations using the ASCII interface of CheckMate.

First a brief description of a hybrid system is presented and then the mapping to the ASCII format is explained.

Hybrid systems

A Hybrid Automaton can be regarded as a tuple $H = (V, n, X_0, F, Inv, R)$ with the following components:

- V is a finite set of *modes* in which the system can operate.

- $n \geq 0$ is a non-negative integer called the *dimension* of the system (i.e the number of continuous variables in the system).

  The state-space of H is $X = V \times \Re^n$.

- $X_0 \in X$ is the set of initial states. It has both a discrete part and a continuous part.
- F assigns to each state $\in X$ a flow condition, which constrains the time derivative of the continuous part of the state. CheckMate supports linear as well as nonlinear flow conditions (integrators being a degenerate case of the linear flow).
- Inv assigns to each mode $l \in V$ an *invariant set* $Inv(l) \subseteq \Re^n$ which constrains the value of the continuous part of the state while the discrete part is l. The invariant set is specified as boolean combinations of half spaces $\hat{C} \times \hat{x} \leq \hat{d}$
- R is a relation capturing discontinuous state changes.

Hybrid Automaton are represented as finite state graphs with vertices V and edges E

$$E = \{(l, l') \in V \times V | ((l, x), (l', x')) \in R \; for \; some \; x \in Inv(l) \; and \; x' \in Inv(l')\},$$

with each edge $e \in E$ we associate a guard set

$$Guard(e) = \{x \in Inv(l) | ((l, x), (l', x')) \in R \; for \; some \; x' \in Inv(l')\}$$

and a set valued reset map

$$Reset(e, x) = \{x' \in Inv(l') | ((l, x), (l', x')) \in R\}$$

In CheckMate hybrid systems can possibly be multiple hybrid automata (they should have no continuous variables in common) which can communicate explicitly by events or implicitly by reading the continuous variables of another automaton.

The ASCII representation

A Hybrid System as described in the previous part can be mapped fairly directly to the ASCII format. For details see the example later on in the document. It might be helpful to read this section alongside the example.

- Each *Hybrid Automaton* in the system is modelled as a separate *machine* in the ASCII representation. (The example has 2 *machines* - 'pump' and 'thermostat')
- The different *modes* for each *machine* are represented by different *locations*(Each *machine* has an associated *location_list*). (e.g the 'pump' machine has 4 modes and hence 4 locations)
- n, the *dimension* of the system is equal to the total number of continuous variables in the system. (e.g the dimension of the swimming pool system is '3'). This is specified using the 'VARIABLES' field.

The variables which 'belong' to each *machine* are specified at the beginning of the declaration of the *machine*. (e.g the 'pump' manipulates variables '1' and '2' while the 'thermostat' manipulates variable '3')

- The set of initial states is specified by a boolean combination of half spaces where each halfspace is represented as $\hat{l} \times \hat{x} \leq d$. ('$\hat{l}$' is a row vector.) . In the CM file this is the 'INITIAL_SET' field. In addition, each *machine* has an 'INITIAL_LOCATION' field to specify the initial location.

- The flow conditions F are specified for each location. Each *location* has associated dynamics (which can be 'LINEAR', 'NON_LINEAR' or 'INTEGRATOR'). In the 'LINEAR' case the dynamics are specified as $xdot = \hat{A} \times \hat{x}$ while is the nonlinear case $xdot$ can be any nonlinear expression.

- The invariant can be specified for each location in the 'INVARIANT' field. This field is not mandatory [1].

- The discontinuous state changes are specified by mean of a *transition list* associated with each *location*. Each transition has a target - specified using 'TARGET' - (which is the name of a location) and a guard condition - specified using 'GUARD' - (which are boolean combinations of halfspaces[2])

- If a transition has an associated 'Reset' it is specified using the optional 'RESET' field. The reset is an affine mapping of the state variables. If a 'machine' manipulates a subset of the variables in the model then resets in that machine are specified only for those variables (leaving the other variables unchanged). For an example of this see the 'new_swim' example in the ASCII demos. Currently only one machine is allowed to generate resets (if the model has more than one machine).

Notes

- When the dynamics for a location are specified it can refer to variables which do not 'belong' to that particular machine. This is an example of implicit communication between hybrid automata (one automata can only read the other's variables)

- Explicit communication between automata (equivalently *machines*) is supported using events. Each machine has an optional input events list and output events list. Transitions can be triggered by events as well as conditions (halfspace combinations). In addition, each transition can have an output event.[3]

[1] The purpose of invariants in Hybrid Automata is to make sure that an enabled transition is taken. The CheckMate verification engine assumes 'urgent semantics' which require an enabled transition to be taken as soon as it is enabled

[2] The whole list of halfspaces is declared at the beginning of the model

[3] Models with mixed event and condition transitions may not simulate correctly in Matlab because Stateflow regards 'events' to be 'trigger inputs' and so if no trigger inputs are high then the machine is not 'executed'

- The 'ANALYSIS_REGION' field specifies the region of the continuous state space which will be considered for verification. It is specified in a manner similar to the initial set.

3 Example

The heated swimming pool example describe the dynamics of a swimming pool. There are two variables under consideration: the level of the water and temperature of the water. The level is regulated using a pump which has two modes: ON and OFF. In ON mode water is filled into the pool. In OFF mode the pump is stopped and the water is drained out at a constant rate. Two transition modes are added to model the time delay in the physical system.

To keep the temperature of the water within a constant range. A heater is used to control the temperature. It has two modes: in HEAT mode it start to heat the pool and in COOL mode it is stopped. Notice that the temperature is also changed by the dynamics of the pump automation because cold water is filled in and mixed with the water in the pool. In the heater automaton, we combine the states of pump and heater to create 8 states. More information about the model in HSIF, Charon and CheckMate can be found at [1].

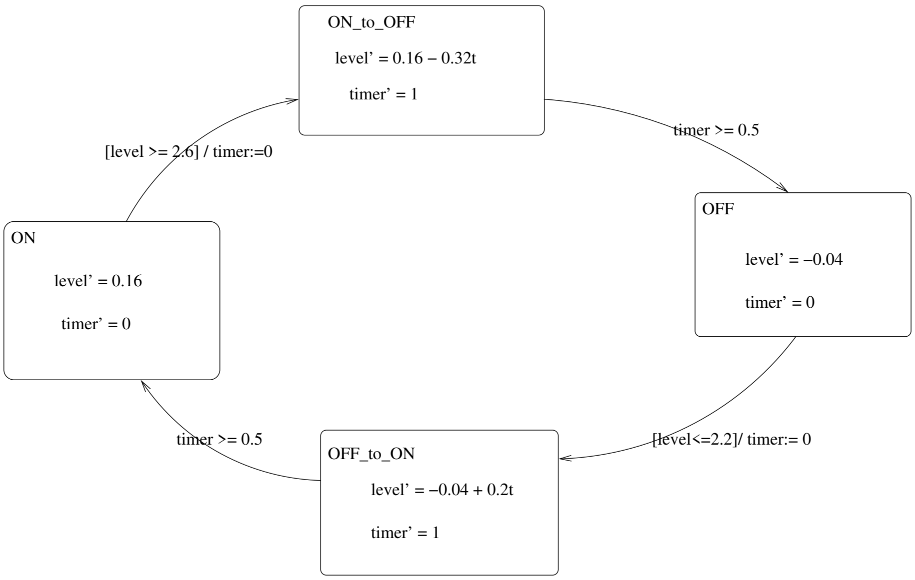

Figure 1: Heated Swimming Pool Example. Pump automaton

3.1 CM File MOdel

```
new_swim{
   VARIABLES: 3;
   HALFSPACES{
```

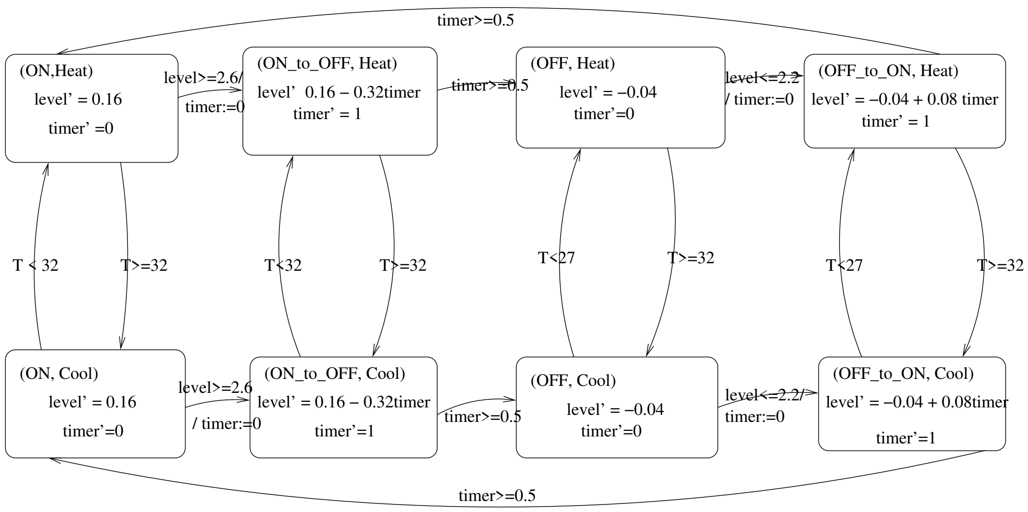

Figure 2: Heated Swimming Pool Example. Heater automaton

```
    AR_1: [1,0,0],<,15;
    AR_2: [-1,0,0],<,0;
    AR_3: [0,1,0],<,10;
    AR_4: [0,-1,0],<,0;
    AR_5: [0,0,1],<,40;
    AR_6: [0,0,-1],<,10;

    ICS_1: [1, 0, 0],<,2.55;
    ICS_2: [-1,0, 0],<,2.3;
    ICS_3: [0,1, 0],<,0;
    ICS_4: [0,-1, 0],<,0;
    ICS_5: [0,0, 1],<,30;
    ICS_6: [0,0, -1],<,0;
    empty: [1,0,0],<,2.2;
    full: [-1,0,0], <,-2.6;
    time_out:[0,-1,0],<,-0.5;
    T_low1:[0, 0, 1],<,27;
    T_low2:[0, 0, 1],<,32;
    T_high:[0, 0, -1],<,-32;
}
MACHINE{
    pump{
```

```
VARIABLE:1,2;
LOCATIONS{
   on{
      LINEAR:[0, 0, 0; 0, 0, 0],[0.16 ; 0];
      INVARIANT:~(full);
      TRANSITIONS{
         on_to_off{
            TARGET:on_to_off;
            GUARD:full;
            RESET:[1,0,0;0,1,0;0,0,1],[0;0;0];
         }
      }
   }
   on_to_off{
      LINEAR:[0, -0.32, 0; 0, 0, 0],[0.16; 1];
      INVARIANT:~(time_out);
      TRANSITIONS{
         off{
            TARGET:off;
            GUARD:time_out;
            RESET:[1,0,0;0,0,0;0,0,1],[0;0;0];
         }
      }
   }
   off{
       LINEAR:[0, 0, 0; 0, 0, 0],[-0.04 ; 0];
       INVARIANT:~(empty);
       TRANSITIONS{
         off{
            TARGET:off_to_on;
            GUARD:empty;
           RESET:[1,0,0;0,1,0;0,0,1],[0;0;0];
         }
       }
   }
   off_to_on{
```

```
      LINEAR:[0, 0.08, 0; 0, 0, 0],[-0.04; 1];
      INVARIANT:~(time_out);
      TRANSITIONS{
        on{
          TARGET:on;
          GUARD:time_out;
         RESET:[1,0,0;0,0,0;0,0,1],[0;0;0];
        }
      }
    }
  }
  INITIAL_LOCATION: on;
}
thermostat{
  VARIABLE:3;
  LOCATIONS{
    Heatmode_On{
      LINEAR:[0, 0, -0.0350],[1.0250];
      TRANSITIONS{
        HeatMode_Off{
          TARGET:HeatMode_Off;
          GUARD:full;
        }
        CoolMode_On{
          TARGET:CoolMode_On;
          GUARD:T_high;
        }
      }
    }
    HeatMode_Off{
      LINEAR:[0, 0, -0.0250],[0.8750];
      TRANSITIONS{
        Heatmode_On{
          TARGET:Heatmode_On;
          GUARD:empty;
        }
```

```
            CoolMode_Off{
                TARGET:CoolMode_Off;
                GUARD:T_high;
             }
         }
      }
      CoolMode_On{
         LINEAR:[0, 0, -0.0183],[0.3583];
         TRANSITIONS{
            Heatmode_On{
                TARGET:Heatmode_On;
                GUARD:T_low2;
             }
            CoolMode_Off{
                TARGET:CoolMode_Off;
                GUARD:full;
             }
         }
      }
      CoolMode_Off{
         LINEAR:[0, 0, -0.0083],[0.2083];
         TRANSITIONS{
            CoolMode_On{
                TARGET:CoolMode_On;
                GUARD:empty;
             }
            HeatMode_Off{
                TARGET:HeatMode_Off;
                GUARD:T_low1;
             }
         }
      }
   }
   INITIAL_LOCATION: Heatmode_On;
 }
}
```

```
   ANALYSIS_REGION:AR_1 & AR_2 & AR_3 & AR_4 & AR_5 & AR_6;
   INITIAL_SET: ICS_1 & ICS_2 & ICS_3 & ICS_4 & ICS_5 & ICS_6;
}
```

3.2 Generated CheckMate Model

The generated CheckMate model is shown as Figure 3 and Figure 4 shows the detail of the machine `pump`.

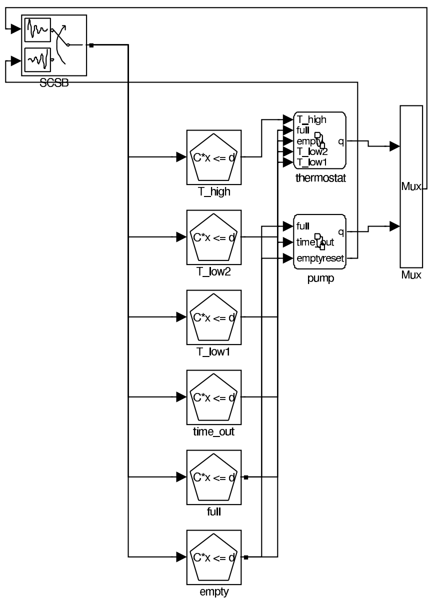

Figure 3: CheckMate model for the swimming pool example

`new_swim_dyn.m` , the generated M file that describes the continuous dynamics, is

```
 % A CheckMate generated dynamics M-file
function [sys, type, reset] = new_swim_dyn(x,u)

   type = 'linear';

   switch u(2)
      case 4
         sys_2.A = [0.0 0.08 0.0 ;0.0 0.0 0.0 ];
         sys_2.b = [-0.04 1.0 ]';
         reset_2.A = [1.0 0.0 0.0 ;0.0 1.0 0.0 ;0.0 0.0 1.0 ];
         reset_2.B = [0.0 0.0 0.0 ]';
```

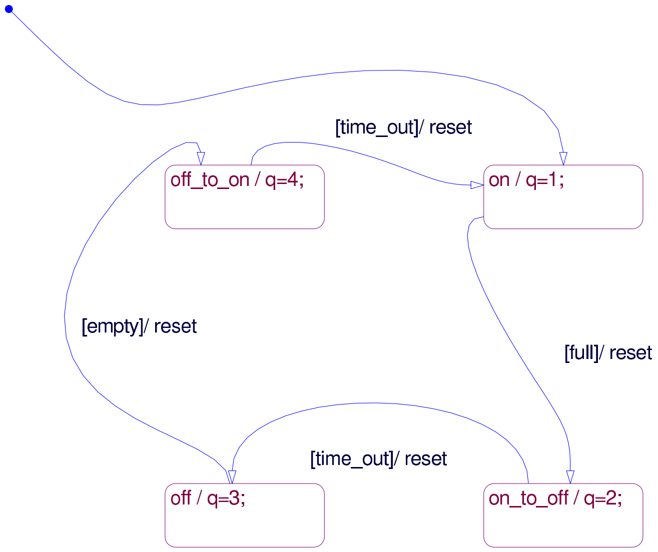

Figure 4: CheckMate model for the swimming pool example

```
case 3
   sys_2.A = [0.0 0.0 0.0 ;0.0 0.0 0.0 ];
   sys_2.b = [-0.04 0.0 ]';
   reset_2.A = [1.0 0.0 0.0 ;0.0 0.0 0.0 ;0.0 0.0 1.0 ];
   reset_2.B = [0.0 0.0 0.0 ]';
case 2
   sys_2.A = [0.0 -0.32 0.0 ;0.0 0.0 0.0 ];
   sys_2.b = [0.16 1.0 ]';
   reset_2.A = [1.0 0.0 0.0 ;0.0 1.0 0.0 ;0.0 0.0 1.0 ];
   reset_2.B = [0.0 0.0 0.0 ]';
case 1
   sys_2.A = [0.0 0.0 0.0 ;0.0 0.0 0.0 ];
   sys_2.b = [0.16 0.0 ]';
   reset_2.A = [1.0 0.0 0.0 ;0.0 0.0 0.0 ;0.0 0.0 1.0 ];
   reset_2.B = [0.0 0.0 0.0 ]';
otherwise
   sys_2.A = [0.0 0.0 0.0 ;0.0 0.0 0.0 ];
   sys_2.b = [0.16 0.0 ]';
   reset_2.A = [1.0 0.0 0.0 ;0.0 0.0 0.0 ;0.0 0.0 1.0 ];
   reset_2.B = [0.0 0.0 0.0 ]';
```

```
end
switch u(1)
   case 4
      sys_1.A = [0.0 0.0 -0.0083 ];
      sys_1.b = [0.2083 ]';
      reset_1.A = [0, 0, 0;0, 0, 0;0, 0, 0];
      reset_1.B = zeros(3.0,1);
   case 3
      sys_1.A = [0.0 0.0 -0.0183 ];
      sys_1.b = [0.3583 ]';
      reset_1.A = [0, 0, 0;0, 0, 0;0, 0, 0];
      reset_1.B = zeros(3.0,1);
   case 2
      sys_1.A = [0.0 0.0 -0.025 ];
      sys_1.b = [0.875 ]';
      reset_1.A = [0, 0, 0;0, 0, 0;0, 0, 0];
      reset_1.B = zeros(3.0,1);
   case 1
      sys_1.A = [0.0 0.0 -0.035 ];
      sys_1.b = [1.025 ]';
      reset_1.A = [0, 0, 0;0, 0, 0;0, 0, 0];
      reset_1.B = zeros(3.0,1);
   otherwise
      sys_1.A = [0.0 0.0 -0.035 ];
      sys_1.b = [1.025 ]';
      reset_1.A = [0, 0, 0;0, 0, 0;0, 0, 0];
      reset_1.B = zeros(3.0,1);
end
sys.A(3,:) = sys_1.A(1,:);
sys.b(3,:) = sys_1.b(1,:);
sys.A(2,:) = sys_2.A(2,:);
sys.b(2,:) = sys_2.b(2,:);
sys.A(1,:) = sys_2.A(1,:);
sys.b(1,:) = sys_2.b(1,:);
reset.A =  reset_1.A + reset_2.A ;
reset.B =  reset_1.B + reset_2.B ;
```

```
return
```

new_swim_setup.m, the generated setup file for this swimming pool example, is

```
 % A CheckMate generated setup M-file
evalin('base','global_var');
 % Create global variables


T_high = linearcon([],[],[0.0, 0.0, -1.0],-32.0);
assignin('base','T_high',T_high); T_low2 = linearcon([],[],[0.0,
0.0, 1.0],32.0); assignin('base','T_low2',T_low2); T_low1 =
linearcon([],[],[0.0, 0.0, 1.0],27.0);
assignin('base','T_low1',T_low1); time_out = linearcon([],[],[0.0,
-1.0, 0.0],-0.5); assignin('base','time_out',time_out); full =
linearcon([],[],[-1.0, 0.0, 0.0],-2.6);
assignin('base','full',full); empty = linearcon([],[],[1.0, 0.0,
0.0],2.2); assignin('base','empty',empty); initial_set =
{linearcon([], [], [1.0, 0.0, 0.0 ; -1.0, 0.0, 0.0 ; 0.0, 1.0, 0.0
; 0.0, -1.0, 0.0 ; 0.0, 0.0, 1.0 ; 0.0, 0.0, -1.0],
[2.55;2.3;0.0;0.0;30.0;0.0 ])};
assignin('base','initial_set',initial_set); analysis_region =
linearcon([], [], [1.0, 0.0, 0.0 ; -1.0, 0.0, 0.0 ; 0.0, 1.0, 0.0
; 0.0, -1.0, 0.0 ; 0.0, 0.0, 1.0 ; 0.0, 0.0, -1.0],
[15.0;0.0;10.0;0.0;40.0;10.0 ]);
assignin('base','analysis_region',analysis_region);

GLOBAL_SYSTEM = 'new_swim';
```

3.3 verification

A BNF for the PIHA

A.1 Terminal Symbols

List of terminal symbols(in ANTLR Grammar format).

```
tokens{
//CheckMate used terminals
MODEL   = "MODEL"   ;

VARIABLES  = "VARIABLES" ;

HALFSPACES = "HALFSPACES" ;

PARAMETERS = "PARAMETERS" ;

PAR_CONSTRAINT  = "PAR_CONSTRAINT" ;

CLOCKS  = "CLOCKS" ;

LOCATIONS  = "LOCATIONS" ;

TRANSITIONS  = "TRANSITIONS" ;

INTEGRATOR  = "INTEGRATOR" ;

LINEAR = "LINEAR" ;

NONLINEAR  = "NONLINEAR" ;

INVARIANT  = "INVARIANT" ;

ANALYSIS_REGION  = "ANALYSIS_REGION" ;

INITIAL_SET = "INITIAL_SET" ;

TARGET = "TARGET" ;
```

```
GUARD = "GUARD" ;

CLOCK = "CLOCK" ;

RESET = "RESET" ;

MACHINE = "MACHINE" ;

INITIAL_LOCATION = "INITIAL_LOCATION" ;

VARIABLE = "VARIABLE";  }

WS  :   (' '
    |   '\t'
    |   '\n'
    |   '\r')
        { _ttype = Token.SKIP; }
    ;


LEFTPAREN:'('   ;

RIGHTPAREN:')'  ;

COMMA:',';

COLON:':';

SEMI:';';

LEFTBRACE: '{';

RIGHTBRACE: '}';

LEFTBRACKET : '[';
```

```
RIGHTBRACKET : ']';

SMALLER : '<';
GREATER : '>';
NOT: '~' ;
AND: '&';
OR: '|';

FLOAT options {testLiterals = false;}
: ( INTEGER ('.' INTEGER ( ('e'|'E') ('-'|'+')? INTEGER )? )? );

protected
INTEGER
    :   ('0'..'9')+
    ;

protected
CHAR    :   'a'..'z'
    |   '_'
    |   'A'..'Z';

OP  :'+'
    | '-'
    | '*'
    |'/' ;

STRING options{ testLiterals = true;} :
(CHAR ) ( INTEGER | CHAR )* ;
```

A.2 GRAMMAR

```
model
    :   modelname LEFTBRACE variables parameters
            halfspace_list
```

```
            clock_list
            machine_list
            analysis_region
            initial_set
            RIGHTBRACE
    ;

modelname
    :   STRING
    ;

variables
        :   VARIABLES COLON nnumber SEMI
        ;

parameters
        :   PARAMETERS COLON nnumber SEMI
        |
        ;

halfspace_list
        :   HALFSPACES LEFTBRACE halfspace halfspace_list_entries
                                                     RIGHTBRACE
        ;

halfspace_list_entries
        :   halfspace halfspace_list_entries
        |
        ;

halfspace
        :   STRING COLON LEFTBRACKET vector RIGHTBRACKET
                  COMMA relation COMMA fnumber SEMI
        ;

clock_list
```

```
        :   CLOCKS LEFTBRACE clock_list_entries RIGHTBRACE
        |
        ;

clock_list_entries
        :  clock clock_list_entries
        |
        ;

clock
        :   STRING COLON fnumber COMMA fnumber COMMA fnumber
                SEMI
        ;

machine_list
        :   MACHINE LEFTBRACE machine_list_entries RIGHTBRACE
        ;

machine_list_entries
        :  machine machine_list_entries
        |
        ;

machine
        :   STRING LEFTBRACE variable_list event_list location_list
                initial_location RIGHTBRACE
        ;

event_list : EVENTS COLON input_event_list SEMI output_event_list
SEMI
        |
        ;

input_event_list : STRING event_list_entries
        |
        ;
```

```
output_event_list : STRING event_list_entries
        |
        ;

event_list_entries: COMMA STRING event_list_entries
        |
        ;

variable_list
        :   VARIABLE COLON fnumber fnumber_list_entries
                          SEMI
        |


        ;

location_list
        :   LOCATIONS LEFTBRACE location_list_entries
                          RIGHTBRACE
        ;

location_list_entries
        : location location_list_entries
        |
        ;

initial_location
        :   INITIAL_LOCATION COLON STRING SEMI
        ;

location
        :   STRING LEFTBRACE behaviour invariant
           parameter_constraint transition_list RIGHTBRACE
        ;

behaviour
```

```
        :   INTEGRATOR COLON rate_list
        |   LINEAR COLON linearmap
        | NONLINEAR LEFTBRACE nonlinearmap RIGHTBRACE
        |
        ;

rate_list
        :   fnumber fnumber_list_entries SEMI
        ;

linearmap
        :   LEFTBRACKET vector linearmap_vectors RIGHTBRACKET
            COMMA
            LEFTBRACKET fnumber linearmap_fnumbers RIGHTBRACKET
            SEMI
        ;


linearmap_vectors
        : SEMI vector linearmap_vectors
        |
        ;

linearmap_fnumbers
        :  SEMI fnumber linearmap_fnumbers
        |
        ;

vector
        :   fnumber fnumber_list_entries
        ;

fnumber_list_entries
        :  COMMA  fnumber fnumber_list_entries
        |
        ;
```

```
nonlinearmap
        :   ode_rhs SEMI nonlinearmap
        |
        ;

ode_rhs
        :  OP  expr
        |  expr
        ;

expr
        :  u_expr expr_list
        ;

expr_list
        :  OP u_expr expr_list
        |
        ;
u_expr
        :   function
        |   fnumber
        |   LEFTPAREN expr RIGHTPAREN
        ;

function
        :   STRING LEFTPAREN a_arguments RIGHTPAREN
        ;

a_arguments
        :   a_argument a_arguments_list
        ;

a_arguments_list
        :   COMMA a_argument a_arguments_list
        |
```

```
        ;

a_argument
        :   fnumber
        |   function
        ;

invariant
        :  INVARIANT COLON boolean_function SEMI
        |
        ;

boolean_function
        :   l_argument bool_func_list
        ;

bool_func_list
        :  operator l_argument bool_func_list
        |
        ;

operator
        :  AND
        |  OR
        ;

l_argument
        :   LEFTPAREN l_argument RIGHTPAREN
                  l_argument_one
        |   NOT l_argument
        |   STRING l_argument_two
        ;

l_argument_one
        :  operator l_argument l_argument_one
        |
```

```
        ;

l_argument_two
        : operator l_argument
        |
        ;

relation
        :  SMALLER
        |  GREATER
        |  EQUAL
        ;

parameter_constraint
        :  PAR_CONSTRAINT COLON boolean_function SEMI
        |
        ;

transition_list
        :   TRANSITIONS LEFTBRACE transition transition_list_entries
                  RIGHTBRACE
        |
        ;

transition_list_entries
        : transition transition_list_entries
        |
        ;

transition
        :   STRING LEFTBRACE target guard reset
                clock_flag ievent oevent RIGHTBRACE
        ;

target
        :  TARGET COLON STRING SEMI
```

```
        ;

guard
        :  GUARD COLON boolean_function SEMI
        ;

reset
        :   RESET COLON linearmap
        |
        ;

clock_flag
        :   CLOCK COLON STRING SEMI
        |
        ;

ievent : IEVENT COLON STRING SEMI
        |
        ;

oevent : OEVENT COLON STRING SEMI
        |
        ;

analysis_region
        :   ANALYSIS_REGION COLON boolean_function SEMI
        ;

initial_set
        :   INITIAL_SET COLON boolean_function SEMI
        ;

fnumber
        :   OP FLOAT
        | FLOAT
        ;
```

```
nnumber
        :  OP FLOAT
        | FLOAT
        ;
```

References

[1] HSIF discussion forum: http://micc.isis.vanderbilt.edu:8080/HSIF/Documents/HSIF-Examples/view