CS181u Applied Logic
& Automated Reasoning

Lecture 8

Symbolic Model Verifier (νSMV)
Next Few Weeks:

Linear Temporal Logic (LTL)
We will assign symbols for expressing temporal system requirements like *always* \((G)\), *eventually* \((F)\), *next* \((X)\), *until* \((U)\), and a few more. We will give a formal and unambiguous semantics to these symbols.

Transition Systems
We will learn a formal system of specifying transition systems (which we often depict as a transition diagram).

Concurrency Concepts
Safety, liveness, mutual exclusion, ... 

Temporal Logic Software
Symbolic Model Verifier (NuSMV)
Next Few Weeks:

Linear Temporal Logic (LTL)
We will assign symbols for expressing temporal system requirements like always \((G)\), eventually \((F)\), next \((X)\), until \((U)\), and a few more. We will give a formal and unambiguous semantics to these symbols.

Transition Systems
We will learn a formal system of specifying transition systems (which we often depict as a transition diagram).

Concurrency Concepts
Safety, liveness, mutual exclusion, …

Temporal Logic Software
Symbolic Model Verifier (NuSMV)
Reactive System
Code

satisfies

Requirements

satisfies

Temporal Logic
Formula

φ

Model Checking
Reactive System Code satisfies Requirements

Transition System satisfies Temporal Logic Formula

$M \models \phi \iff \forall \pi [\pi \models \phi]$

$vSMV$ LTL Model Checking
νSMV Syntax by Example
MODULE main

VAR
  request : boolean;
  status : {ready, busy};

ASSIGN
  init(status) := ready;
  next(status) :=
    case
      request : busy;
      TRUE      : {ready, busy};
    esac;

LTLSPEC G(request → F(status = busy))
\textbf{νSMV Syntax by Example}

\begin{verbatim}
MODULE main

VAR
    request : boolean;
    status : {ready, busy};

ASSIGN
    init(status) := ready;
    next(status) :=
        case
            request : busy;
            TRUE    : {ready, busy};
        esac;

LTLSPEC G(request -> F(status = busy))
\end{verbatim}

\textit{νSMV source code consists of one or more MODULES.}
A MODULE has a **VAR** block where variables are declared.

```
VAR
    request : boolean;
    status : {ready, busy};

ASSIGN
    init(status) := ready;
    next(status) :=
        case
            request : busy;
            TRUE : {ready, busy};
        esac;

LTLSPEC G(request -> F(status = busy))
```
A MODULE has an ASSIGN block where:

- The **init** function initializes the status to ready.
- The **next** function updates the status based on the request:
  - If there is a request, the status becomes busy.
  - If there is no request, the status remains the same.
- The LTL specification checks if the request eventually leads to the status being busy.

The code snippet is as follows:

```
MODULE main

VAR
    request : boolean;
    status : {ready, busy};

ASSIGN
    init(status) := ready;
    next(status) :=
        case
            request : busy;
            TRUE : {ready, busy};
        esac;

LTLSPEC G(request -> F(status = busy))
```
\textbf{νSMV Syntax by Example}

\begin{verbatim}
MODULE main

VAR
    request : boolean;
    status : {ready, busy};

ASSIGN
    init(status) := ready;
    next(status) :=
        case
            request : busy;
            TRUE : {ready, busy};
        esac;

LTLSPEC G(request \rightarrow F(status = busy))
\end{verbatim}

A MODULE has an \textbf{ASSIGN} block where:

Variables are \textbf{initialized}:

\[
\text{init(var-name)} := \text{value};
\]
νSMV Syntax by Example

A MODULE has an **ASSIGN** block where:

Variables are **initialized**:

\[
\text{init(var-name)} := \text{value};
\]

Uninitialized variables (request) can take on any value from their type “non-deterministically.”
A MODULE has an ASSIGN block where:

Transition relation is specified:

\[
\text{next(var-name)} := \text{expression};
\]
A MODULE has an ASSIGN block where:

Transition relation is specified:

next(var-name) := expression;

Variables without specified transition are updated “non-deterministically.”
Expressions can be case statements.

Note: the result can also be a non-deterministic choice.
νSMV Syntax by Example

MODULE main

VAR
    request : boolean;
    status  : {ready, busy};

ASSIGN
    init(status) := ready;
    next(status) :=
        case
            request : busy;
            TRUE    : {ready, busy};
            esac;

LTLSPEC G(request -> F(status = busy))

A MODULE can have a SPECIFICATION block.
A νSMV file specifies a transition system

The set of states, $S$, is the set of all possible combinations of variables.

\[ \text{request} \times \text{status} = \{ \text{TRUE, FALSE} \} \times \{ \text{ready, busy} \} \]

The initial states, $I$ are specified by init.

\[
\text{init}(\text{status}) := \text{ready};
\]

\[ I = \{( \text{request, ready}), (\neg\text{request, ready})\} \]

The transition relation, $\rightarrow$ is specified by next.
A νSMV file specifies a transition system
Getting νSMV

https://nusmv.fbk.eu/

NuSMV: a new symbolic model checker

NuSMV 2.6.0 is OUT!
Running νSMV

νSMV can typically be run from the command line

./NuSMV

νSMV can be run in batch (default) or interactive mode

./NuSMV file-name.smv

./NuSMV -int file-name.smv

interactive flag

νSMV can run a “script” of commands

./NuSMV -source cmd-file [-int] file-name.smv
Running \( \nu \text{SMV} \)

Interactive mode is good for exploring capabilities of \( \nu \text{SMV} \).

Some useful commands:

\[
\begin{align*}
\text{NuSMV} & \quad \text{> go} \quad \text{runs a bunch of setup commands} \\
\text{NuSMV} & \quad \text{> <TAB>} \quad \text{shows a list of commands} \\
\text{NuSMV} & \quad \text{> quit} \quad \text{terminates interactive session} \\
\text{NuSMV} & \quad \text{> <CMD> -h} \quad \text{shows help for <CMD>}
\end{align*}
\]
Running $\nu$SMV

Some more useful commands:

```plaintext
NuSMV > print_reachable_states -v

NuSMV > pick_state -v -a [-i | -r]

NuSMV > simulate -v [-r | -i [-a]] -k <n>

NuSMV > check_ltlspec -p <LTLSPEC STRING>

NuSMV > check_property

NuSMV > print_fair_transitions -f dot -o fsm.dot
```
Other useful stuff

Some useful example files
  Will be included in updated Docker image

Look through my-commands.smv
  NuSMV> save_ts outputs the transition system to fsm.dot

Generate a pdf of the transition system
  https://dreampuf.github.io/GraphvizOnline/

  $ circo -v -Tpdf fsm.dot -o fsm.pdf
  Also try: circo, dot, neato, twopi, fdp, sfdp
Recall our Mutual Exclusion Example

\begin{verbatim}
proc(id, other, turn)
    while(true)
        n: flag := TRUE; turn = (id + 1) % 2;
        w: wait until (!other.flag | turn = id)
        c: flag := FALSE;
    }
\end{verbatim}
Recall our Mutual Exclusion Example

MODULE p(id, turn, other)
VAR
  pc : {n, w, c};
  flag : boolean;
ASSIGN
  init(pc) := n;
  init(flag) := FALSE;
  next(pc) :=
    case
      pc = n : w;
      pc = c : n;
      pc = w & (turn = id | !other.flag) : c;
      pc = w : w;
    esac;
  next(turn) :=
    case
      pc = n : (id + 1) mod 2;
      TRUE : turn;
    esac;
  next(flag) :=
    case
      pc = n : TRUE;
      pc = c : FALSE;
      TRUE : flag;
    esac;
FAIRNESS running

MODULE main
VAR
  turn : 0..1;
  p0 : process p(0, turn, p1);
  p1 : process p(1, turn, p0);
ASSIGN
  init(turn) := 0;
LTLSPEC
  G(! (p0.pc = c & p1.pc = c))
Remember the big picture

- Reactive System Code satisfies Requirements
- Transition System satisfies Temporal Logic Formula $\phi$

Model Checking