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

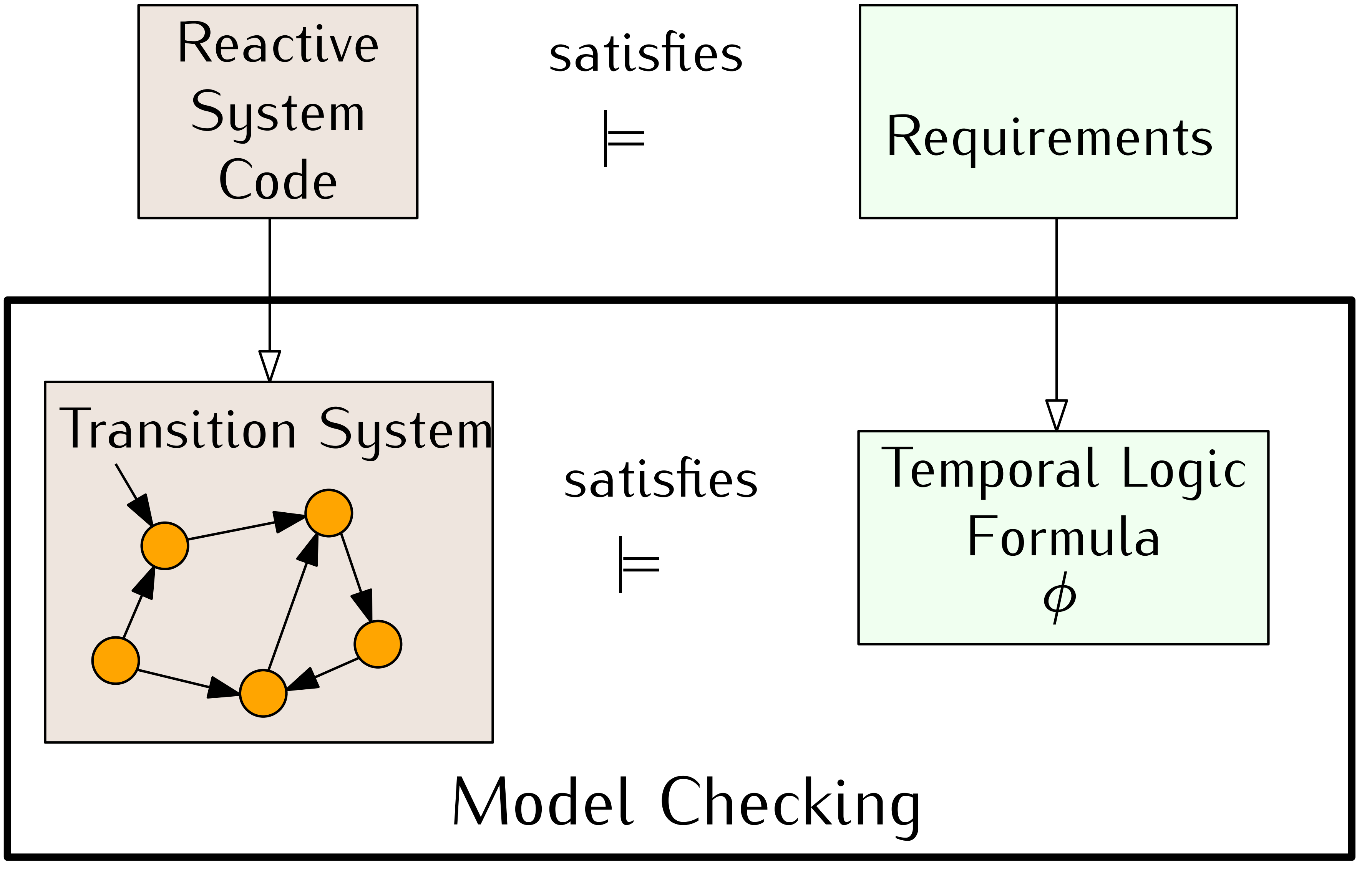
We will learn a formal system of specifying transition systems (which we often depict as a transition diagram).

Concurrency Concepts

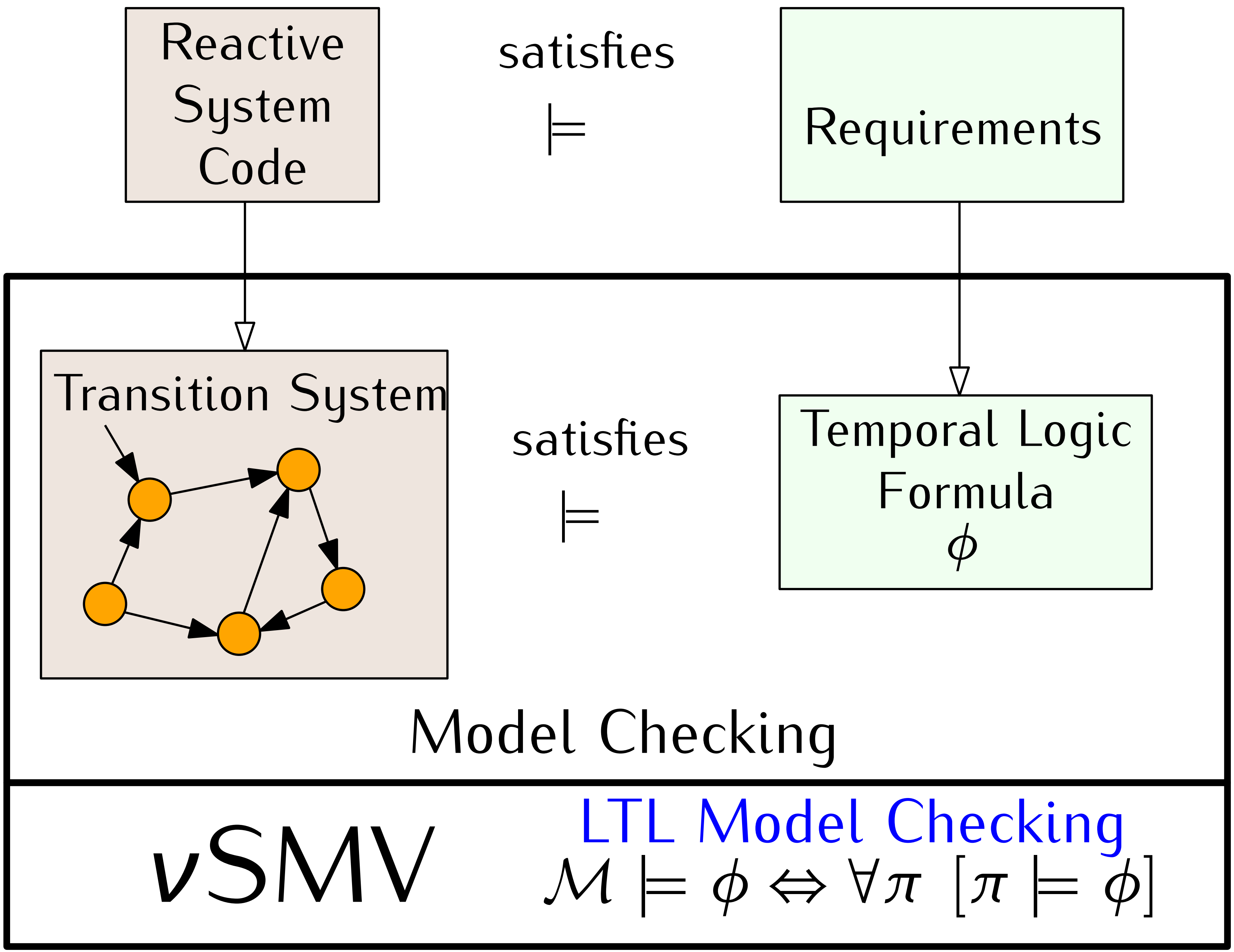
Safety, liveness, mutual exclusion, ...

Temporal Logic Software Today

Symbolic Model Verifier (NuSMV)



Reactive System Code
satisfies
$\models$
Requirements
Transition System
satisfies
$\models$
Temporal Logic Formula ϕ
Model Checking


Reactive System Code
satisfies
$\models$
Requirements
Transition System
satisfies
$\models$
Temporal Logic Formula ϕ
Model Checking
vSMV
LTL Model Checking
$\mathcal{M} \models \phi \Leftrightarrow \forall\pi\ [\pi \models \phi]$

νSMV Syntax by Example

*v*SMV 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))
```

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))
```

νSMV source code consists of one or more MODULES.

*ν*SMV 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 has a VAR block where variables are declared.

```
var-name : enum-type;
```

vSMV 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 has an ASSIGN block where:

vSMV 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 has an ASSIGN block where:

Variables are initialized:

`init(var-name) := value;`

vSMV 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 has an ASSIGN block where:

Variables are initialized:

`init(var-name) := value;`

Unitialized variables (`request`) can take on any value from their type "non-deteministically."

*v*SMV 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 has an ASSIGN block where:

Transition relation is specified:

`next(var-name) := expression;`

*ν*SMV 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 has an ASSIGN block where:

Transition relation is specified:

`next(var-name) := expression;`

Variables without specified transition are updated “non-deterministically.”

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))
```

Expressions can be case statements.

```
case
  condition1 : result1;
  condition2 : result2;
  ...
  conditionN : resultN;
  TRUE       : default;
esac;
```

Note: the `result` can also be a non-determinsitic choice.

*v*SMV 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.

$$\texttt{request} \times \texttt{status} = \{\texttt{TRUE, FALSE}\} \times \{\ \texttt{ready, busy}\ \}$$

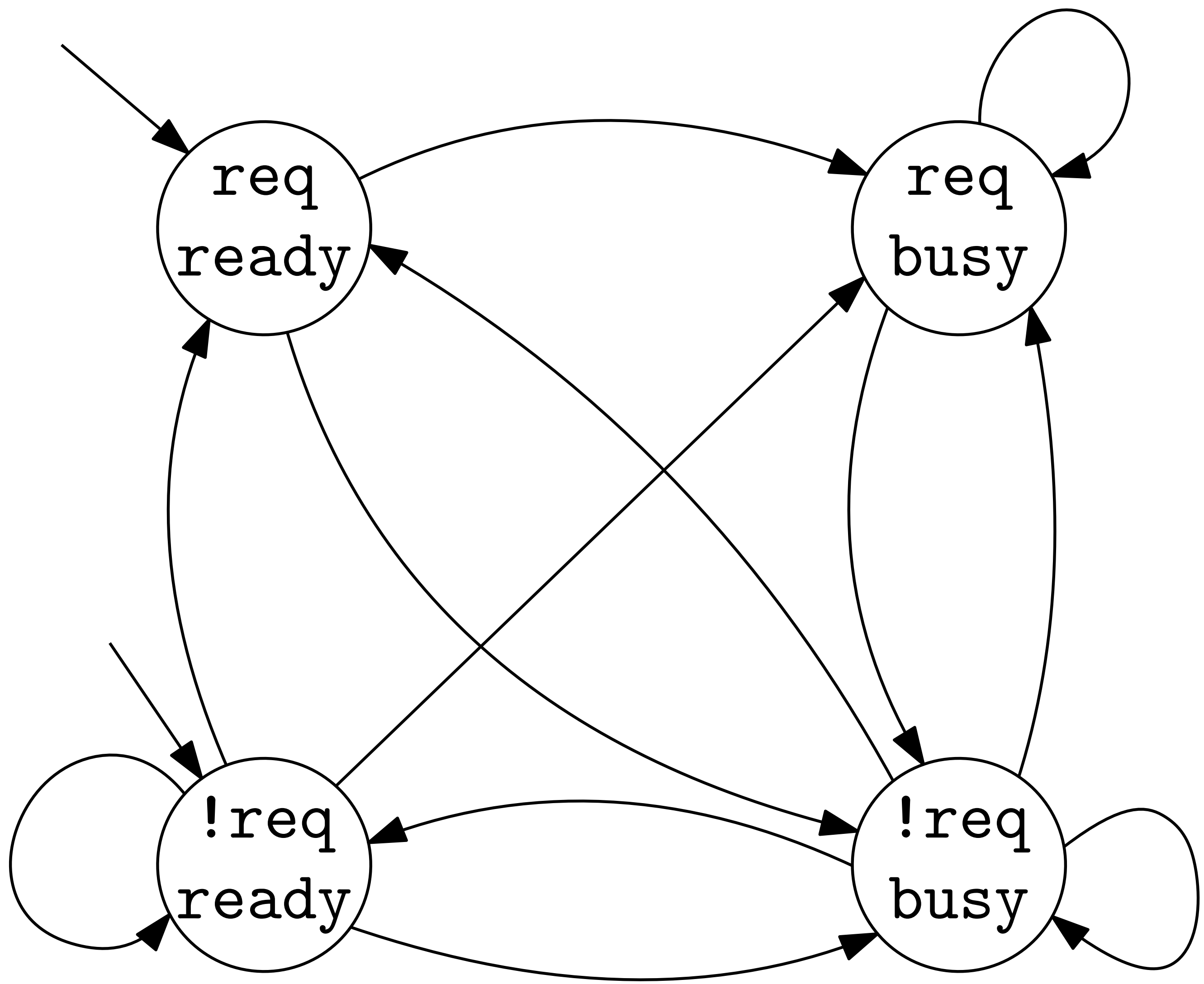
The initial states, I are specified by `init`.

```
init(status) := ready;
```

$$I = \{(\ \texttt{request, ready}), (\ \neg\texttt{request, ready}\ )\}$$

The transition relation, $\rightarrow$ is specified by `next`.

A νSMV file specifies a transition system



Running νSMV in `Docker`

νSMV is available in `Docker` as

```
smv
```

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

```
smv file-name.smv
```

smv $\boxed{\texttt{-int}}$ `file-name.smv`

interactive flag

νSMV can run a "script" of commands

smv $\boxed{\texttt{-source cmd-file}}$ `[-int] file-name.smv`

Running νSMV in Docker

Interactive mode is good for exploring capabilities of νSMV.
Some useful commands:

NuSMV > go   runs a bunch of setup commands

NuSMV > <TAB>   shows a list of commands

NuSMV > quit   terminates interactive session

NuSMV > <CMD> -h   shows help for <CMD>

Running *ν*SMV in Docker

Some more useful commands:

```
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

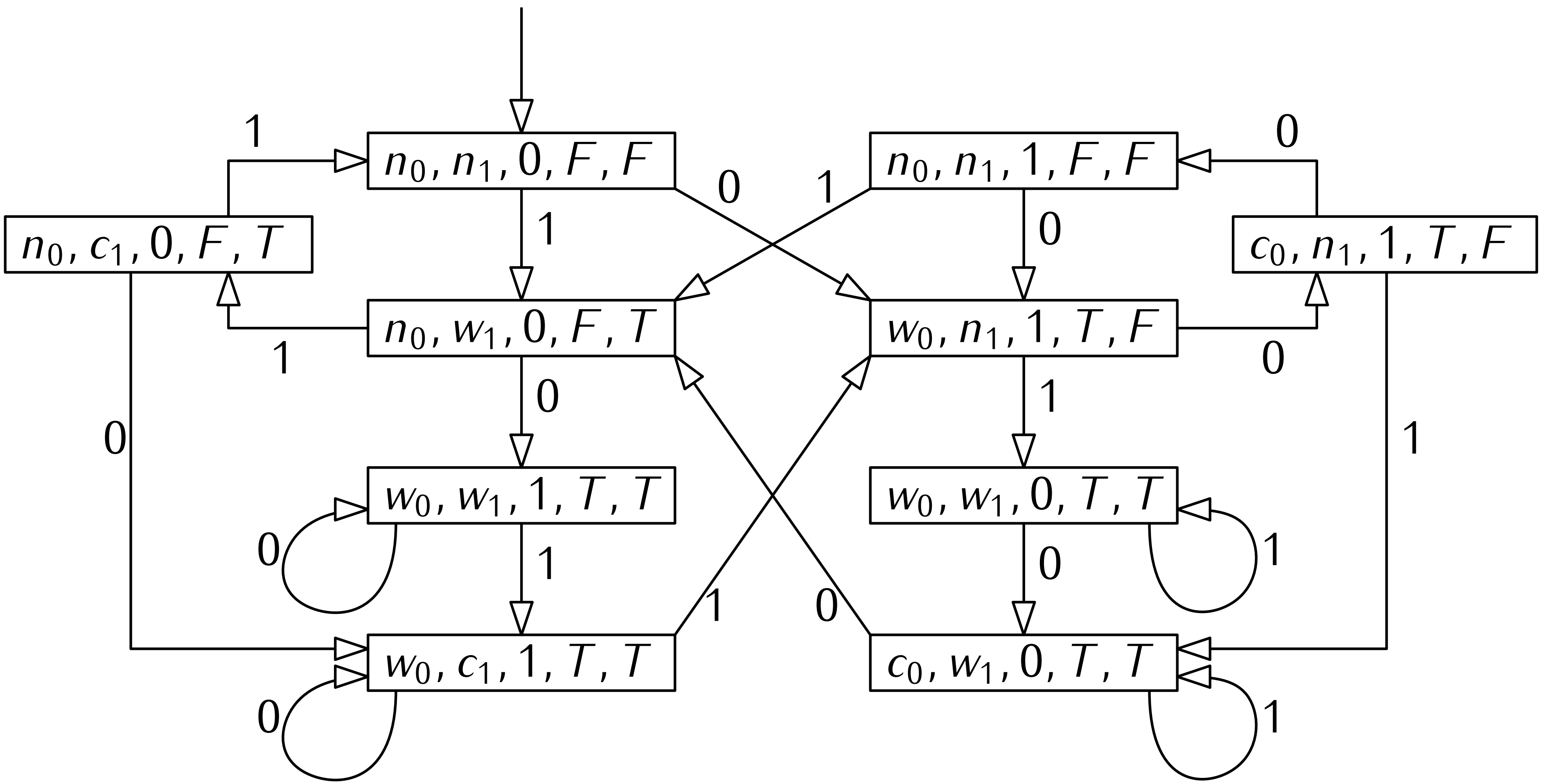
Generate a pdf of the transition system

```
$ circo -v -Tpdf fsm.dot -o fsm.pdf
```

Also try: `circo`, `dot`, `neato`, `twopi`, `fdp`, `sfdp`

Recall our Mutual Exclusion Example

```
proc(id, other, turn)
    while(true){
      n:     flag := TRUE; turn = (id + 1) % 2;
      w:     wait until (!other.flag | turn = id)
      c:     flag := FALSE;
    }
```



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