Temporal Logic and Model Checking

- Model
  - mathematical structure extracted from hardware or software
- Temporal logic
  - provides a language for specifying functional properties
- Model checking
  - checks whether a given property holds of a model

- Model checking is a kind of static verification
  - dynamic verification is simulation (HW) or testing (SW)

Models

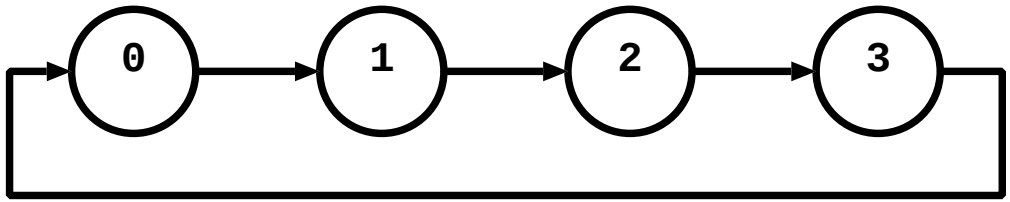
- A model is (for now) specified by a pair (S, R)
  - S is a set of *states*
  - R is a *transition relation*

- Models will get more components later
  - (S, R) also called a transition system

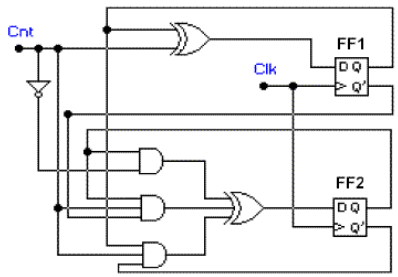
- $R\ s\ s'$ means s' can be reached from s in one step
  - here $R : S \to (S \to \mathbb{B})$ (where $\mathbb{B} = \{\mathit{true}, \mathit{false}\}$)
  - more conventional to have $R \subseteq S \times S$, which is equivalent
  - i.e. $R_{\text{(this course)}}\ s\ s' \Leftrightarrow (s, s') \in R_{\text{(some textbooks)}}$

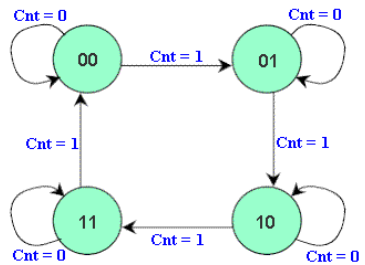
A simple example model

- A simple model: $(\underbrace{\{0,1,2,3\}}_{S}, \underbrace{\lambda n\ n'.\ n' = n{+}1 (mod\ 4)}_{R})$
  - where "$\lambda x.\ \cdots x \cdots$" is the function mapping x to $\cdots x \cdots$
  - so $R\ n\ n' = (n' = n{+}1 (mod\ 4))$
  - e.g. $R\ 0\ 1 \wedge R\ 1\ 2 \wedge R\ 2\ 3 \wedge R\ 3\ 0$



- Might be extracted from:





[Acknowledgement: http://eelab.usyd.edu.au/digital_tutorial/part3/t-diag.htm]

DIV: a software example

- Perhaps a familiar program:

```
0:  R:=X;
1:  Q:=0;
2:  WHILE Y≤R DO
3:  (R:=R-Y;
4:   Q:=Q+1)
5:
```

- State (pc, x, y, r, q)
  - $pc \in \{0, 1, 2, 3, 4, 5\}$ program counter
  - x, y, r, $q \in \mathbb{Z}$ are the values of X, Y, R, Q
- Model $(S_{\mathtt{DIV}}, R_{\mathtt{DIV}})$ where:

$$S_{\mathtt{DIV}} = [0..5] \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z} \quad \text{(where } [m..n] = \{m, m+1, \ldots, n\}\text{)}$$

$$\begin{aligned}
\forall x\ y\ r\ q.\ & R_{\mathtt{DIV}}\ (0, x, y, r, q)\ (1, x, y, x, q) \quad \wedge \\
& R_{\mathtt{DIV}}\ (1, x, y, r, q)\ (2, x, y, r, 0) \quad \wedge \\
& R_{\mathtt{DIV}}\ (2, x, y, r, q)\ ((\mathit{if}\ y \leq r\ \mathit{then}\ 3\ \mathit{else}\ 5), x, y, r, q) \quad \wedge \\
& R_{\mathtt{DIV}}\ (3, x, y, r, q)\ (4, x, y, (r-y), q) \quad \wedge \\
& R_{\mathtt{DIV}}\ (4, x, y, r, q)\ (2, x, y, r, (q+1))
\end{aligned}$$

- [Above changed from lecture to make $R_{\mathtt{DIV}}$ partial!]

Deriving a transition relation from a state machine

- State machine transition function: $\delta \ : \ \mathit{Inp} \times \mathit{Mem} \rightarrow \mathit{Mem}$
  - Inp is a set of inputs
  - Mem is a memory (set of storable values)

- Model: (S_δ, R_δ) where:

  $S_\delta = \mathit{Inp} \times \mathit{Mem}$

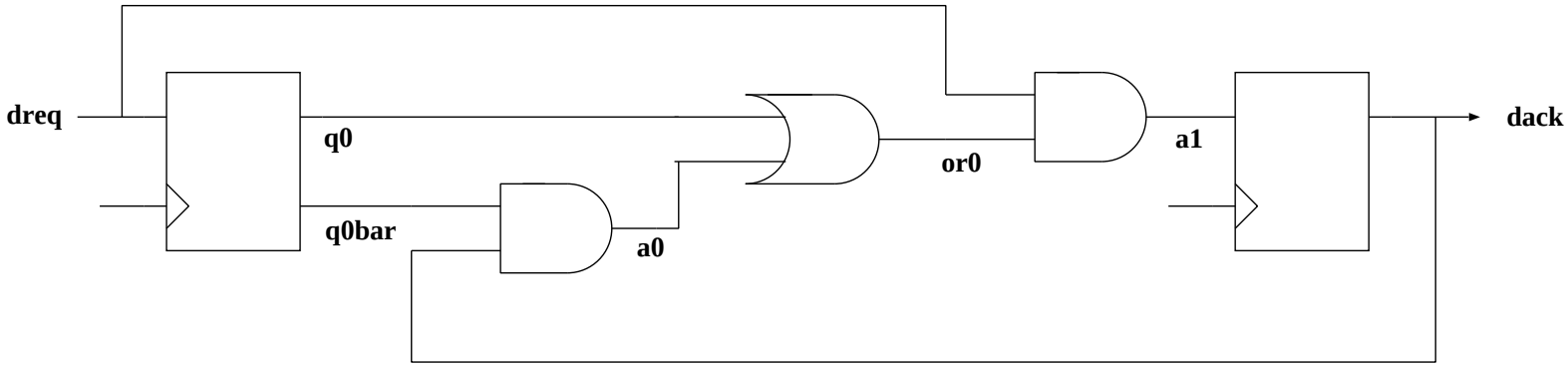
  $R_\delta \ (i, m) \ (i', m') \ = \ (m' \ = \ \delta(i, m))$

  and
  - i' arbitrary: determined by environment not by machine
  - m' determined by input and current state of machine

- Deterministic machine, non-deterministic transition relation
  - inputs unspecified (determined by environment)
  - so called “input non-determinism”

RCV: a state machine specification of a circuit

- Part of a handshake circuit:



- Input: $dreq$, Memory: $(q0, dack)$
- Relationships between Boolean values on wires:

$$\begin{array}{lcl} q0bar & = & \neg q0 \\ a0 & = & q0bar \wedge dack \\ or0 & = & q0 \vee a0 \\ a1 & = & dreq \wedge or0 \end{array}$$

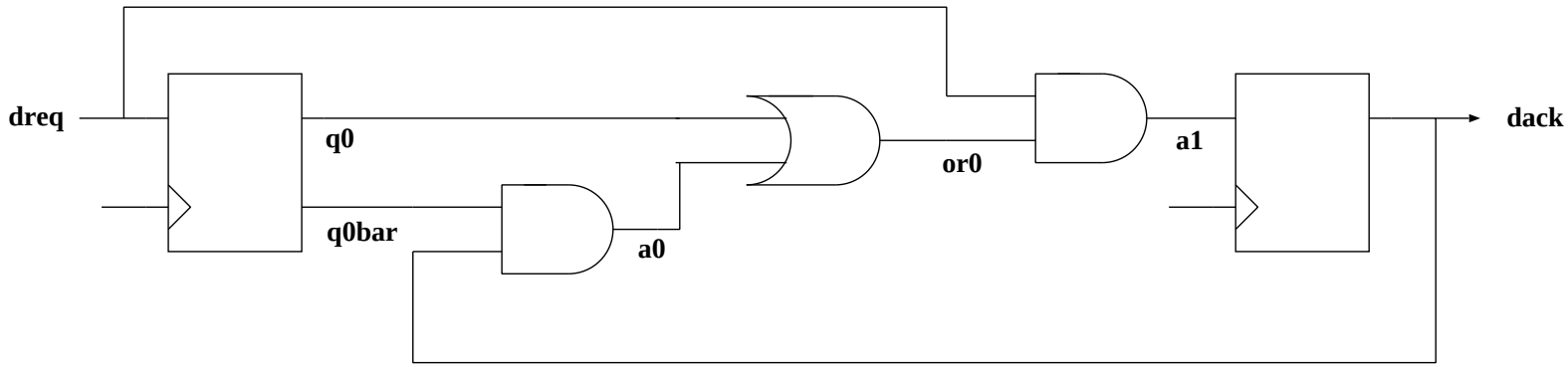
- State machine: $\delta_{\mathrm{RCV}} : \mathbb{B} \times (\mathbb{B} \times \mathbb{B}) \rightarrow (\mathbb{B} \times \mathbb{B})$

$$\delta_{\mathrm{RCV}}\ (\underbrace{dreq}_{Inp},\ \underbrace{(q0, dack)}_{Mem}) = (dreq,\ dreq \wedge (q0 \vee (\neg q0 \wedge dack)))$$

- RTL model – could have lower level model with clock edges

RCV: a model of the circuit

- Circuit from previous slide:



- State represented by a triple of Booleans $(dreq, q0, dack)$
- By De Morgan Law: $q0 \vee (\neg q0 \wedge dack) = q0 \vee dack$
- Hence δ_{RCV} corresponds to model $(S_{\mathrm{RCV}}, R_{\mathrm{RCV}})$ where:

  $S_{\mathrm{RCV}} = \mathbb{B} \times \mathbb{B} \times \mathbb{B}$

  $R_{\mathrm{RCV}}\ (dreq, q0, dack)\ (dreq', q0', dack') =$
  $(q0' = dreq) \wedge (dack' = (dreq \wedge (q0 \vee dack)))$

  [Note: we are identifying $\mathbb{B} \times \mathbb{B} \times \mathbb{B}$ with $\mathbb{B} \times (\mathbb{B} \times \mathbb{B})$]

Some comments

- $R_{\mathtt{RCV}}$ is non-deterministic and total
  - $R_{\mathtt{RCV}}\ (1,1,1)\ (0,1,1)$ and $R_{\mathtt{RCV}}\ (1,1,1)\ (1,1,1)$ (where $1 = \mathit{true}$ and $0 = \mathit{false}$)
  - $R_{\mathtt{RCV}}\ (\mathit{dreq}, q0, \mathit{dack})\ (\mathit{dreq}', \mathit{dreq}, (\mathit{dreq} \wedge (q0 \vee \mathit{dack})))$
- $R_{\mathtt{DIV}}$ is deterministic and partial
  - at most one successor state
  - no successor when $\mathit{pc} = 5$
- Non-deterministic models are very common, e.g. from:
  - asynchronous hardware
  - parallel software (more than one thread)
- Can extend any transition relation R to be total:

$$\begin{aligned} R_{\mathit{total}}\ s\ s' &= \mathit{if}\ (\exists s''.\ R\ s\ s'')\ \mathit{then}\ R\ s\ s'\ \mathit{else}\ (s' = s) \\ &= R\ s\ s' \vee (\neg(\exists s''.\ R\ s\ s'') \wedge (s' = s)) \end{aligned}$$

  - sometimes totality required (e.g. in the book *Model Checking* by Clarke et. al)

JM1: a non-deterministic software example

- From Jhala and Majumdar's tutorial:

Thread 1	Thread 2
`0: IF LOCK=0 THEN LOCK:=1;`	`0: IF LOCK=0 THEN LOCK:=1;`
`1: X:=1;`	`1: X:=2;`
`2: IF LOCK=1 THEN LOCK:=0;`	`2: IF LOCK=1 THEN LOCK:=0;`
`3:`	`3:`

- Two program counters, state: $(pc_1, pc_2, lock, x)$

$$S_{\mathtt{JM1}} = [0..3] \times [0..3] \times \mathbb{Z} \times \mathbb{Z}$$

$$\begin{aligned}\forall pc_1\ pc_2\ lock\ x.\ & R_{\mathtt{JM1}}\ (0, pc_2, 0, x)\ \ (1, pc_2, 1, x) \wedge \\ & R_{\mathtt{JM1}}\ (1, pc_2, lock, x)\ (2, pc_2, lock, 1) \wedge \\ & R_{\mathtt{JM1}}\ (2, pc_2, 1, x)\ \ (3, pc_2, 0, x) \wedge \\ & R_{\mathtt{JM1}}\ (pc_1, 0, 0, x)\ \ (pc_1, 1, 1, x) \wedge \\ & R_{\mathtt{JM1}}\ (pc_1, 1, lock, x)\ (pc_1, 2, lock, 2) \wedge \\ & R_{\mathtt{JM1}}\ (pc_1, 2, 1, x)\ \ (pc_1, 3, 0, x)\end{aligned}$$

- Not-deterministic:

$$R_{\mathtt{JM1}}\ (0, 0, 0, x)\ (1, 0, 1, x)$$
$$R_{\mathtt{JM1}}\ (0, 0, 0, x)\ (0, 1, 1, x)$$

- Not so obvious that $R_{\mathtt{JM1}}$ is a correct model

Atomic properties (properties of states)

- Atomic properties are true or false of individual states
  - an atomic property p is a function $p : S \to \mathbb{B}$
  - can also be regarded as a subset of state: $p \subseteq S$
- Example atomic properties of `RCV`
  (where $1 = \mathit{true}$ and $0 = \mathit{false}$)

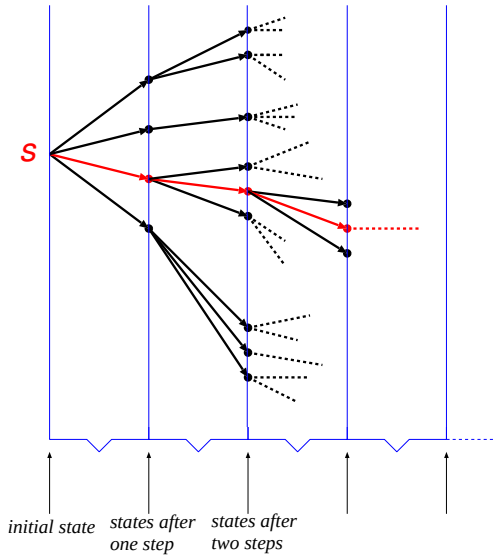
$$\begin{array}{ll}
\texttt{Dreq}(\mathit{dreq}, q0, \mathit{dack}) & = (\mathit{dreq} = 1) \\
\texttt{NotQ0}(\mathit{dreq}, q0, \mathit{dack}) & = (q0 = 0) \\
\texttt{Dack}(\mathit{dreq}, q0, \mathit{dack}) & = (\mathit{dack} = 1) \\
\texttt{NotDreqAndQ0}(\mathit{dreq}, q0, \mathit{dack}) & = (\mathit{dreq}{=}0) \wedge (q0{=}1)
\end{array}$$

- Example atomic properties of `DIV`

$$\begin{array}{ll}
\texttt{AtStart}\ (\mathit{pc}, x, y, r, q) & = (\mathit{pc} = 0) \\
\texttt{AtEnd}\ (\mathit{pc}, x, y, r, q) & = (\mathit{pc} = 5) \\
\texttt{InLoop}\ (\mathit{pc}, x, y, r, q) & = (\mathit{pc} \in \{3, 4\}) \\
\texttt{YleqR}\ (\mathit{pc}, x, y, r, q) & = (y \leq r) \\
\texttt{Invariant}\ (\mathit{pc}, x, y, r, q) & = (x = r + (y \times q))
\end{array}$$

Model behaviour viewed as a computation tree

- Atomic properties are true or false of individual states
- General properties are true or false of whole behaviour
- Behaviour of (S, R) starting from $s \in S$ as a tree:



- A path is shown in red
- Properties may look at all paths, or just a single path
  - CTL: Computation Tree Logic (all paths from a state)
  - LTL: Linear Temporal Logic (a single path)