Systems Modeling

(Based on [Clarke et al. 1999])

Yih-Kuen Tsay

Dept. of Information Management
National Taiwan University

Introduction

- First two steps in correctness verification:
  1. Specify the desired *properties*
  2. Construct a *formal model* (with the desired properties in mind)
     - Capture the necessary properties and leave out the irrelevant
     - Example: gates and boolean values vs. voltage levels
     - Example: exchange of messages vs. contents of messages
- Description of a formal model
  - Graphs (state-transition diagrams)
  - Logic formulae

Concurrent Reactive Systems

- Interact frequently with the environment and *may not terminate*
- Arise from digital circuits, communication protocols, etc.
- *Temporal* (not just input-output) behaviors are most important
- Modeling elements:
  - State: a snapshot of the system at a particular instance
  - Transition:
    - how the system changes its state as a result of some action
    - described by a pair of the state before and the state after the action
  - Computation: an infinite sequence of states resulted from transitions

Kripke Structures

- Kripke structures are one of the most popular types of formal models for concurrent systems.
- Let AP be a set of atomic propositions (representing things you want to observe).
- A *Kripke structure* M over AP is a tuple $\langle S, S_0, R, L \rangle$:
  - S is a finite set of states,
  - $S_0 \subseteq S$ is the set of initial states,
  - $R \subseteq S \times S$ is a *total* transition relation, and
  - $L : S \rightarrow 2^{AP}$ is a function labeling each state with a subset of propositions (which are true in that state).
- A *computation* or *path* of M from a state s is an infinite sequence of states $\sigma = s_0, s_1, s_2, \cdots$ such that $s_0 \in S_0$ and $(s_i, s_{i+1}) \in R$, for all $i \geq 0$.

First-Order Representations

- First-order formulae serve as a unifying formalism for describing concurrent systems.
- Elements of first-order logic:
  - Logical connectives ($\wedge$, $\vee$, $\neg$, $\rightarrow$, etc.) and quantifiers ($\forall$ and $\exists$)
  - Predicate and function symbols (with predefined meanings)
- Variables range over a finite domain D.
- A *valuation* for a set V of variables is a map from the variables in V to the values in the domain D.
- A *state* of a system is a valuation for the system variables.
- A *set of states* can be described by a *first-order formula*.
- The set of initial states of a system will typically be described by $\mathcal{S}_0(V)$.

First-Order Representations (cont.)

- To describe transitions by logic formulae, we create a second copy of variables V'.
- Each variables v in V has a corresponding primed version v' in V'.
- The variables in V are *present state* variables, while the variables in V' are *next state* variables.
- A valuation for V and V' can be seen as designating a pair of states or a transition.
- A *set of transitions* or *transition relation* R can then be described by a *first-order formula* $\mathcal{R}(V, V')$.
- Be careful about the issue of granularity.

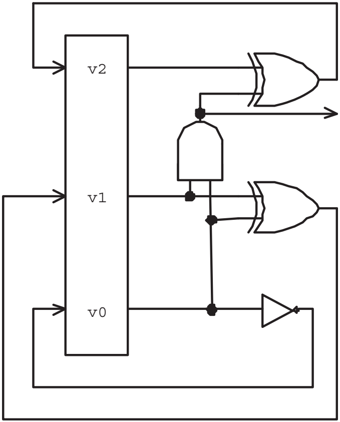
From Formulae to Kripke Structures

- Given $\mathcal{S}_0(V)$ and $\mathcal{R}(V, V')$ that represent a concurrent system, a Kripke structure $M = \langle S, S_0, R, L \rangle$ may be derived:
  - S is the set of all valuations for V.
  - The set of initial states S_0 is the set of all valuations for V satisfying $\mathcal{S}_0$.
  - $R(s, s')$ holds if $\mathcal{R}$ evaluates to *true* when each $v \in V$ is assigned the value $s(v)$ and each $v' \in V'$ is assigned the value $s'(v)$.
  - L is defined such that $L(s)$ is the set of atomic propositions true in s.
- To make R *total*, for every state s that does not have a successor, (s, s) is added into R.

Varieties of Concurrent Systems

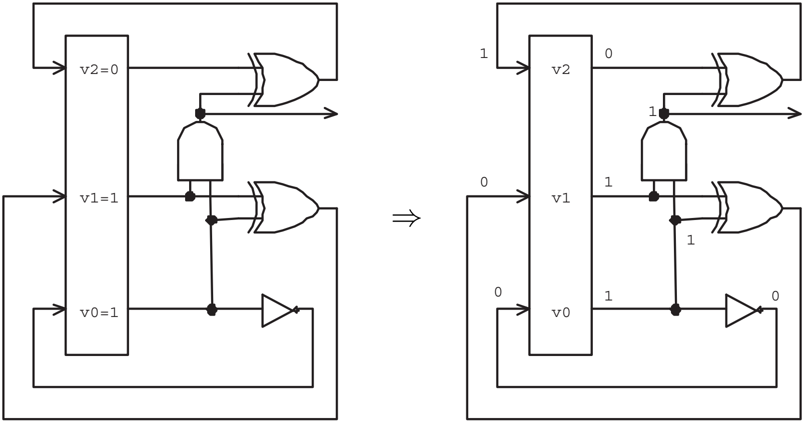
- A concurrent system consists of a set of components that execute together.
- Modes of execution:
  - Asynchronous
  - Synchronous
- Modes of communication:
  - Shared variables
  - Message-passing
  - Handshaking (or joint events)

A Synchronous Modulo 8 Counter



Source: redrawn from [Clarke et al. 1999, Fig 2.1]

A Synchronous Modulo 8 Counter (cont.)



First-Order Representations (Circuit)

- Let V be $\{v_0, v_1, v_2\}$.
- The transitions of the modulo 8 counter are
  - $v_0' = \neg v_0$
  - $v_1' = v_0 \oplus v_1$
  - $v_2' = (v_0 \wedge v_1) \oplus v_2$
- In terms of formulae, they are
  - $\mathcal{R}_0(V, V') \triangleq v_0' \Leftrightarrow \neg v_0$
  - $\mathcal{R}_1(V, V') \triangleq v_1' \Leftrightarrow v_0 \oplus v_1$
  - $\mathcal{R}_2(V, V') \triangleq v_2' \Leftrightarrow (v_0 \wedge v_1) \oplus v_2$
- Conjoining the formulae, we obtain

$$\mathcal{R}(V, V') \triangleq \mathcal{R}_0(V, V') \wedge \mathcal{R}_1(V, V') \wedge \mathcal{R}_2(V, V')$$

Programs

- Concurrent programs are composed of sequential programs/statements.
- A sequential program consists of statements sequentially composed with each other.
- We assume that all statements of a program have a unique *entry point* and a unique *exit point* (they are *structured*).
- To obtain a first-order representation of a program, it is convenient to *label* each statement of the program.

Labeling a Sequential Statement

- Given a sequential statement P, the labeled statement P^L is defined as follows, assuming all labels are unique:
  - If P is not composite, then $P^L = P$.
  - If $P = P_1; P_2$, then $P^L = P_1^L;\ l : P_2^L$.
  - If $P = \textbf{if}\ b\ \textbf{then}\ P_1\ \textbf{else}\ P_2\ \textbf{fi}$, then $P^L = \textbf{if}\ b\ \textbf{then}\ l_1 : P_1^L\ \textbf{else}\ l_2 : P_2^L\ \textbf{fi}$.
  - If $P = \textbf{while}\ b\ \textbf{do}\ P_1\ \textbf{od}$, then $P^L = \textbf{while}\ b\ \textbf{do}\ l_1 : P_1^L\ \textbf{od}$.
- The above labeling procedure may be extended to treat other statement types.

First-Order Representations (Sequential)

- Consider a labeled program P, with the entry labeled m and exit labeled m'.
- Let V denote the set of program variables.
- We postulate a special variable pc called the *program counter* that ranges over the set of program labels plus the *undefined value* $\perp$ (bottom).
- Let $same(Y)$ abbreviate $\bigwedge_{y \in Y} (y' = y)$.
- Given some condition $pre(V)$ on the initial values, the set of initial states is

$$\mathcal{S}_0(V, pc) \triangleq pre(V) \wedge pc = m.$$

First-Order Representations (cont.)

The transition relation $C(l, P, l')$ for a statement P with entry l and exit l' is defined recursively as follows:

- Assignment:
  $C(l, v := e, l') \triangleq pc = l \wedge pc' = l' \wedge v' = e \wedge \mathit{same}(V \setminus \{v\})$.
- Skip:
  $C(l, \mathit{skip}, l') \triangleq pc = l \wedge pc' = l' \wedge \mathit{same}(V)$.
- Sequential Composition:
  $C(l, P_1;\ l'' : P_2, l') \triangleq C(l, P_1, l'') \vee C(l'', P_2, l')$.

First-Order Representations (cont.)

- Conditional:
  $C(l, \textbf{if } b \textbf{ then } l_1 : P_1 \textbf{ else } l_2 : P_2 \textbf{ fi}, l')$ is the disjunction of the following:
  - $pc = l \wedge pc' = l_1 \wedge b \wedge same(V)$
  - $pc = l \wedge pc' = l_2 \wedge \neg b \wedge same(V)$
  - $C(l_1, P_1, l')$
  - $C(l_2, P_2, l')$
- While:
  $C(l, \textbf{while } b \textbf{ do } l_1 : P_1 \textbf{ od}, l')$ is the disjunction of the following:
  - $pc = l \wedge pc' = l_1 \wedge b \wedge same(V)$
  - $pc = l \wedge pc' = l' \wedge \neg b \wedge same(V)$
  - $C(l_1, P_1, l)$

Concurrent Programs

- Concurrent programs are composed of sequential processes (programs/statements).
- We consider only *asynchronous* concurrent programs, where exactly one process can make a transition at any time.
- A concurrent program P has the following form:

  $$\textbf{cobegin } P_1 \parallel P_2 \parallel \cdots \parallel P_n \textbf{ coend}$$

  where P_i's are processes.
- Let V be the set of all program variables and V_i the set of variables that *can be changed by* P_i.
- Let pc be the program counter of P and pc_i that of P_i; let PC be the set of all program counters.

Labeling Concurrent Programs

- Given $P = \textbf{cobegin}\ P_1 \parallel P_2 \parallel \cdots \parallel P_n\ \textbf{coend}$, then

  $$P^L = \textbf{cobegin}\ l_1 : P_1^L\ l_1' \parallel l_2 : P_2^L\ l_2' \parallel \cdots \parallel l_n : P_n^L\ l_n'\ \textbf{coend}.$$

- Note that each process P_i has a unique exit label l_i'.

First-Order Representations (Concurrent)

- Assume the entry is labeled m and exit labeled m'.
- Given some condition $pre(V)$ on the initial values, the set of initial states is

$$\mathcal{S}_0(V, PC) \triangleq pre(V) \wedge pc = m \wedge \bigwedge_{i=1}^{n} (pc_i = \bot)$$

  where $pc_i = \bot$ indicates that P_i is *not active*.
- $C(l, \textbf{cobegin } l_1 : P_1\ l_1' \parallel l_2 : P_2\ l_2' \parallel \cdots \parallel l_n : P_n\ l_n' \textbf{ coend}, l')$ is the disjunction of the following:
  - $pc = l \wedge pc_1' = l_1 \wedge \cdots \wedge pc_n' = l_n \wedge pc' = \bot$ (initialization)
  - $pc = \bot \wedge pc_1 = l_1' \wedge \cdots \wedge pc_n = l_n' \wedge pc' = l' \bigwedge_{i=1}^{n}(pc_i' = \bot)$ (termination)
  - $\bigvee_{i=1}^{n}(C(l_i, P_i, l_i') \wedge same(V \setminus V_i) \wedge same(PC \setminus \{pc_i\})$ (interleaving)

Synchronization Statements

- Assume the statement belongs to P_i.
- Wait (or await):
  $C(l, \mathbf{wait}(b), l')$ is the disjunction of the following:
  - $pc_i = l \wedge pc_i' = l \wedge \neg b \wedge same(V_i)$
  - $pc_i = l \wedge pc_i' = l' \wedge b \wedge same(V_i)$
- Lock (or test-and-set):
  $C(l, \mathbf{lock}(v), l')$ is the disjunction of the following:
  - $pc_i = l \wedge pc_i' = l \wedge v = 1 \wedge same(V_i)$
  - $pc_i = l \wedge pc_i' = l' \wedge v = 0 \wedge v' = 1 \wedge same(V_i \setminus \{v\})$
- Unlock:
  $C(l, \mathbf{unlock}(v), l') \stackrel{\Delta}{=} pc_i = l \wedge pc_i' = l' \wedge v' = 0 \wedge same(V_i \setminus \{v\})$.

A Mutual Exclusion Program

$$P_{MX} = m : \textbf{cobegin } P_0 \parallel P_1 \textbf{ coend } m'$$

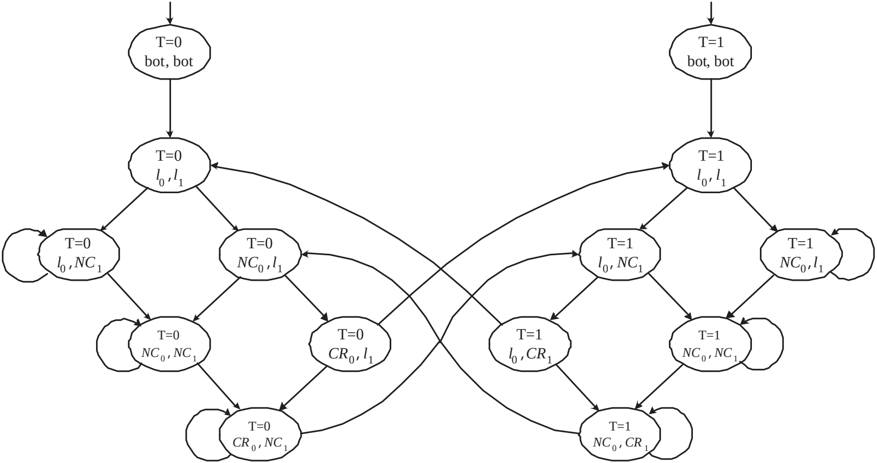
```
P_0 =                                P_1 =
l_0 : while true do                  l_1 : while true do
        NC_0 : wait T = 0;                   NC_1 : wait T = 1;
        CR_0 : T := 1;                       CR_1 : T := 0;
      od;                                  od;
l'_0                                 l'_1
```

- $V = V_0 = V_1 = \{T\}$; $PC = \{pc, pc_0, pc_1\}$.
- The pc of P_{MX} may take m, $\perp$, or m'.
- The pc_0 of P_0: $\perp$, l_0, NC_0, CR_0, or l'_0.
- The pc_1 of P_1: $\perp$, l_1, NC_1, CR_1, or l'_1.

First-Order Representation of P_{MX}

- Initial states $\mathcal{S}_0(V, PC)$: $pc = m \wedge pc_0 = \bot \wedge pc_1 = \bot$.
- Transition relation $\mathcal{R}(V, PC, V', PC')$ is the disjunction of
  - $pc = m \wedge pc_0' = l_0 \wedge pc_1' = l_1 \wedge pc' = \bot$
  - $pc_0 = l_0' \wedge pc_1 = l_1' \wedge pc' = m' \wedge pc_0' = \bot \wedge pc_1' = \bot$
  - $C(l_0, P_0, l_0') \wedge same(V \setminus V_0) \wedge same(PC \setminus \{pc_0\})$
  - $C(l_1, P_1, l_1') \wedge same(V \setminus V_1) \wedge same(PC \setminus \{pc_1\})$
- For each P_i, $C(l_i, P_i, l_i')$ is the disjunction of
  - $pc_i = l_i \wedge pc_i' = NC_i \wedge true \wedge same(T)$
  - $pc_i = NC_i \wedge pc_i' = CR_i \wedge T = i \wedge same(T)$
  - $pc_i = CR_i \wedge pc_i' = l_i \wedge T = (1 - i)$
  - $pc_i = NC_i \wedge pc_i' = NC_i \wedge T \neq i \wedge same(T)$
  - $pc_i = l_i \wedge pc_i' = l_i' \wedge false \wedge same(T)$

A Kripke Structure for P_{MX}



Source: redrawn from [Clarke et al. 1999, Fig 2.2]