Introduction to Formal Methods
Chapter 02: Modeling Transition Systems

Roberto Sebastiani

DISI, Università di Trento, Italy – roberto.sebastiani@unitn.it
URL: http://disi.unitn.it/rseba/DIDATTICA/fm2020/
Teaching assistant: Enrico Magnago – enrico.magnago@unitn.it

CDLM in Informatica, academic year 2019-2020

last update: Monday 18th May, 2020, 14:48

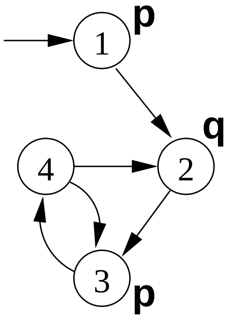
Outline

Modeling the system: Kripke models

- Kripke models are used to describe reactive systems:
  - nonterminating systems with infinite behaviors (e.g. communication protocols, hardware circuits);
  - represent the dynamic evolution of modeled systems;
  - a state includes values to state variables, program counters, content of communication channels.
  - can be animated and validated before their actual implementation

Kripke model: formal definition

- A Kripke model $\langle S, I, R, AP, L \rangle$ consists of
  - a finite set of states S;
  - a set of initial states $I \subseteq S$;
  - a set of transitions $R \subseteq S \times S$;
  - a set of atomic propositions (Boolean variables) AP;
  - a labeling function $L : S \longmapsto 2^{AP}$.
- We assume R Total: for every state s, there exists (at least) one state s' s.t. $(s, s') \in R$
- Sometimes we use variables with discrete bounded values $v_i \in \{d_1, ..., d_k\}$ (can be encoded with $\lceil log(k) \rceil$ Boolean variables)

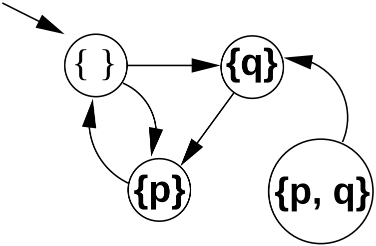


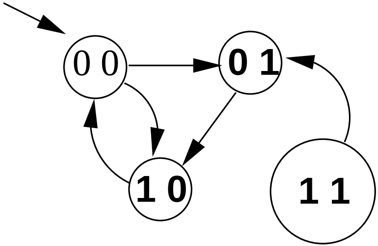
Remark

Unlike with other types of Automata (e.g., Buechi), in Kripke structures the value of every variable is always assigned in each state.

Kripke Structures: two alternative representations:

- each state identifies univocally the values of the atomic propositions which hold there
- each state is labeled by a bit vector

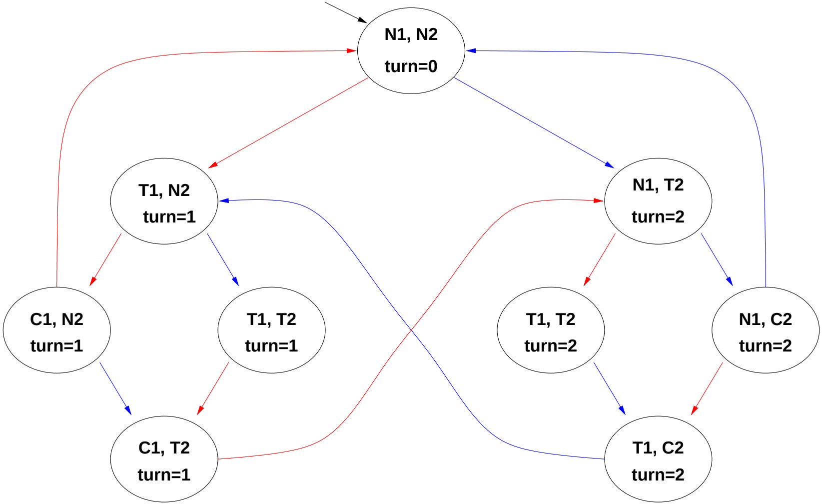




Other representations of finite state machines

- Moore machines
- Mealy machines
- Finite automata
- Büchi automata
- ...

Example: a Kripke model for mutual exclusion



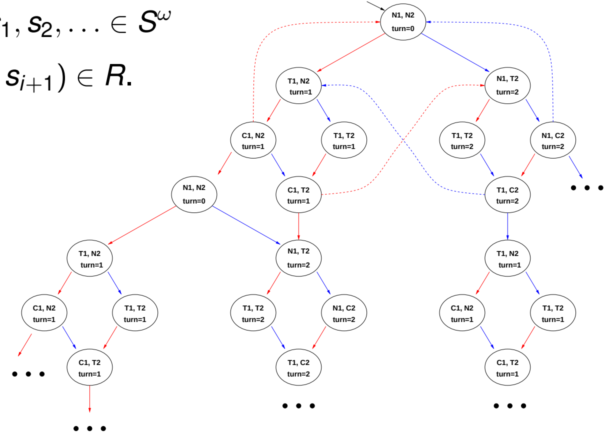
N = noncritical, T = trying, C = critical **User 1** **User 2**

Path in a Kripke Model

A path in a Kripke model M is an infinite sequence of states

$$\pi = s_0, s_1, s_2, \ldots \in S^{\omega}$$

such that $s_0 \in I$ and $(s_i, s_{i+1}) \in R$.



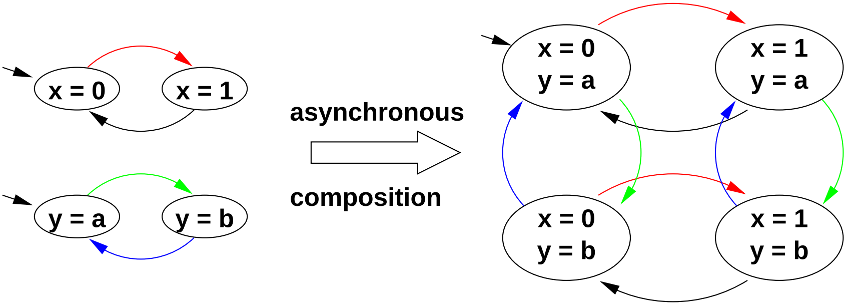
A state s is reachable in M if there is a path from the initial states to s.

Composing Kripke Models

- Complex Kripke Models are tipically obtained by composition of smaller ones
- Components can be combined via
  - asynchronous composition.
  - synchronous composition,

Asynchronous Composition

- Interleaving of evolution of components.
- At each time instant, one component is selected to perform a transition.



- Typical example: communication protocols.

Asynchronous Composition/Product: formal definition

Asynchronous product of Kripke models

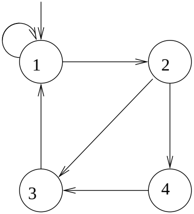
Let $M_1 \stackrel{\text{def}}{=} \langle S_1, I_1, R_1, AP_1, L_1\rangle$, $M_2 \stackrel{\text{def}}{=} \langle S_2, I_2, R_2, AP_2, L_2\rangle$. Then the asynchronous product $M \stackrel{\text{def}}{=} M_1||M_2$ is $M \stackrel{\text{def}}{=} \langle S, I, R, AP, L\rangle$, where

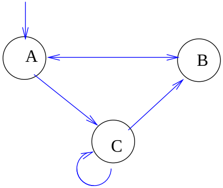
- $S \subseteq S_1 \times S_2$ s.t.,
  $\forall\langle s_1, s_2\rangle \in S,\ \forall l \in AP_1 \cap AP_2, l \in L_1(s_1)$ *iff* $l \in L_2(s_2)$
- $I \subseteq I_1 \times I_2$ s.t. $I \subseteq S$
- $R(\langle s_1, s_2\rangle, \langle t_1, t_2\rangle)$ *iff* $(R_1(s_1, t_1)$ **and** $s_2 = t_2)$ **or** $(s_1 = t_1$ **and** $R_2(s_2, t_2))$
- $AP = AP_1 \cup AP_2$
- $L : S \longmapsto 2^{AP}$ s.t. $L(\langle s_1, s_2\rangle) \stackrel{\text{def}}{=} L_1(s_1) \cup L_2(s_2)$.

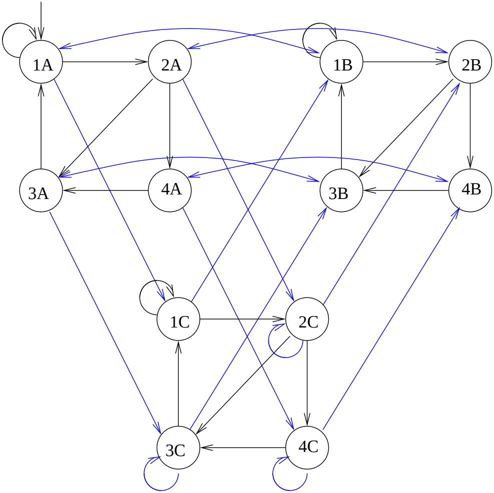
Note: combined states must agree on the values of Boolean variables.

Asynchronous composition is associative:
$(...(M_1||M_2)||...)||M_n) = (M1||(M_2||(...||M_n)...) = M_1||M_2||...||M_n$

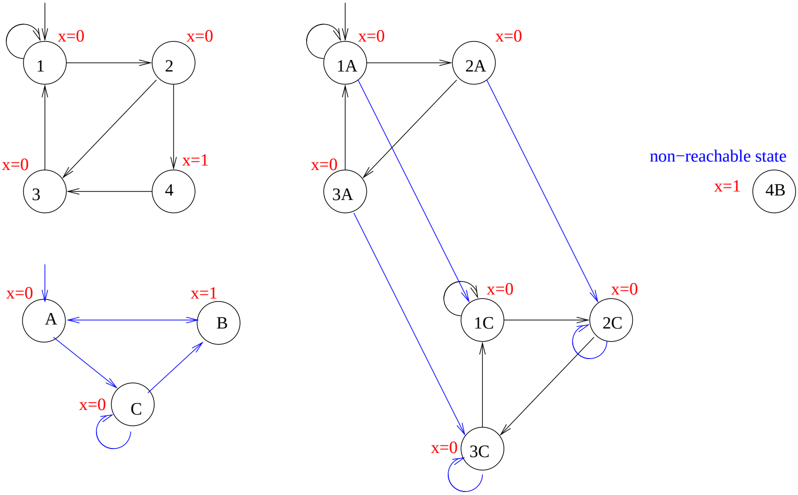
Asynchronous Composition: Example 1



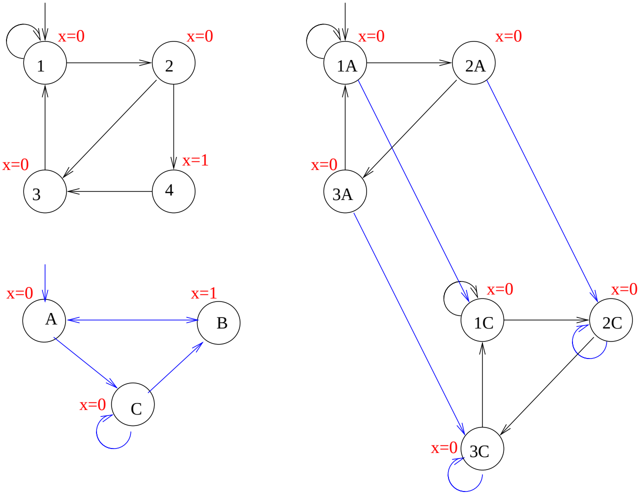




Asynchronous Composition: Example 2

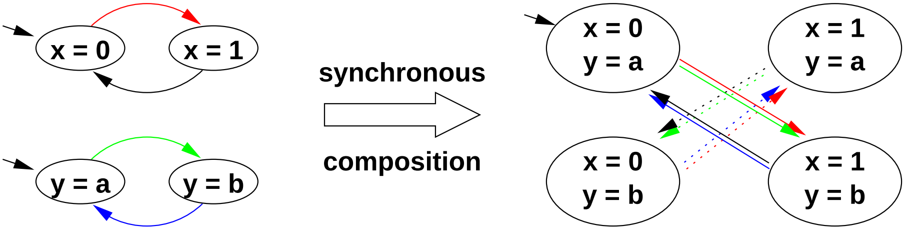


Asynchronous Composition: Example 2



Synchronous Composition

- Components evolve in parallel.
- At each time instant, every component performs a transition.



- Typical example: sequential hardware circuits.

Synchronous Composition/Product: formal definition

Synchronous product of Kripke models

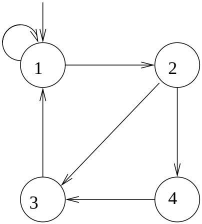
Let $M_1 \stackrel{\text{def}}{=} \langle S_1, I_1, R_1, AP_1, L_1 \rangle$, $M_2 \stackrel{\text{def}}{=} \langle S_2, I_2, R_2, AP_2, L_2 \rangle$. Then the synchronous product $M \stackrel{\text{def}}{=} M_1 \times M_2$ is $M \stackrel{\text{def}}{=} \langle S, I, R, AP, L \rangle$, where

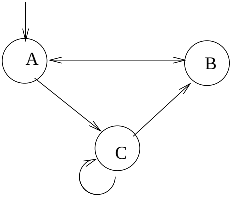
- $S \subseteq S_1 \times S_2$ s.t.,
  $\forall \langle s_1, s_2 \rangle \in S,\ \forall l \in AP_1 \cap AP_2, l \in L_1(s_1)$ *iff* $l \in L_2(s_2)$
- $I \subseteq I_1 \times I_2$ s.t. $I \subseteq S$
- $R(\langle s_1, s_2 \rangle, \langle t_1, t_2 \rangle)$ *iff* $(R_1(s_1, t_1)$ **and** $R_2(s_2, t_2))$
- $AP = AP_1 \cup AP_2$
- $L : S \longmapsto 2^{AP}$ s.t. $L(\langle s_1, s_2 \rangle) \stackrel{\text{def}}{=} L_1(s_1) \cup L_2(s_2)$.

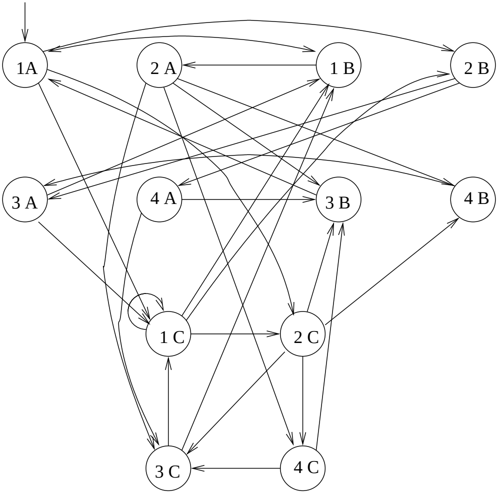
Note: combined states must agree on the values of Boolean variables.

Synchronous composition is associative:
$(...(M_1 \times M_2) \times ...) \times M_n) = (M1 \times (M_2 \times (... \times M_n)...) = M_1 \times M_2 \times ... \times M_n$

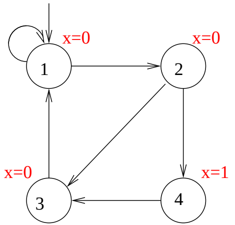
Synchronous Composition: Example 1

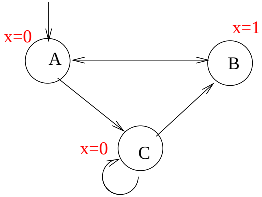


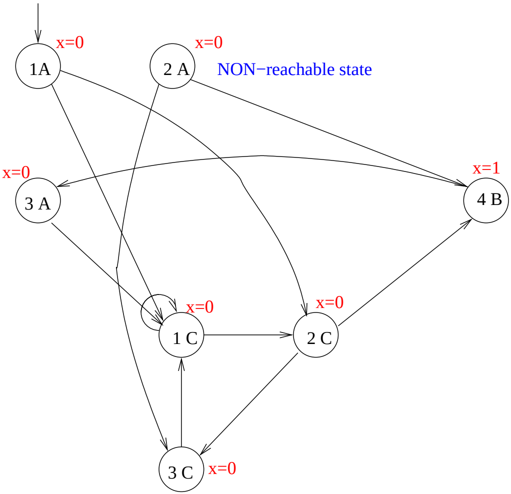




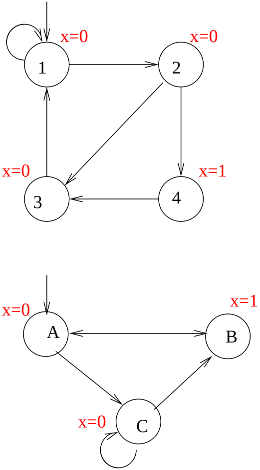
Synchronous Composition: Example 2

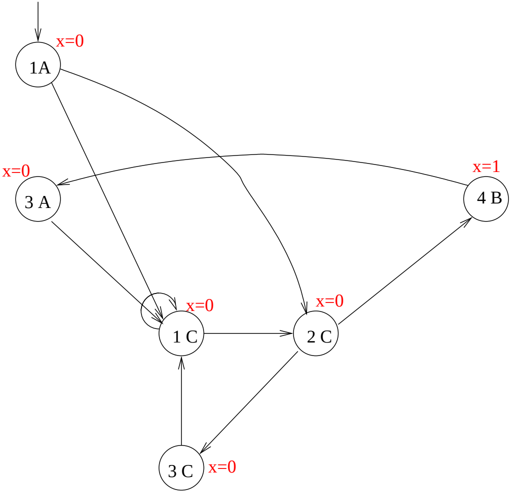






Synchronous Composition: Example 2 (cont.)





Description languages for Kripke Model

- Tipically a Kripke model is not given explicitly, rather it is usually presented in a structured language
  (e.g., SMV, SDL, PROMELA, StateCharts, VHDL, ...)
- Each component is presented by specifying
  - state variables: determine the set of atomic propositions AP, the state space S and the labeling L.
  - initial values for state variables: determine the set of initial states I.
  - instructions: determine the transition relation R.

Remark

tipically these description are much more compact (and intuitive) than the explicit representation of the Kripke model.

The SMV language

- The input language of the SMV M.C. (and NuSMV)
- Booleans, enumerative and bounded integers as data types
- now enriched with other constructs, e.g. in NuXMV language
- An SMV program consists of:
  - Declarations of the state variables (e.g., `b0`);
  - Assignments that define the valid initial states (e.g., `init(b0) := 0`).
  - Assignments that define the transition relation (e.g., `next(b0) := !b0`).
- Allows for both synchronous and asyncronous composition of modules (though synchronous interaction more natural)

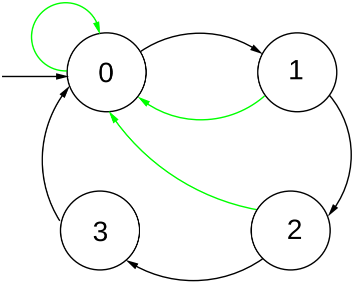
The SMV language: example

Example: The modulo 4 counter with reset

```
MODULE main
 VAR
   b0     : boolean;
   b1     : boolean;
   reset  : boolean;
   out    : 0..3;

 ASSIGN
   init(b0)  := 0;
   next(b0)  := case
                  reset = 1 : 0;
                  reset = 0 : !b0;
                esac;

   init(b1)  := 0;
   next(b1)  := case
                  reset = 1 : 0;
                  reset = 0 : (b0 xor b1);
                esac;
   out  := toint(b0) + 2*toint(b1);
```



The PROMELA language

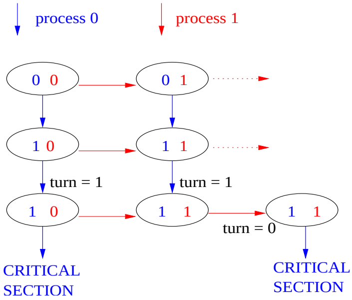
- PROMELA (Process Meta Language) is the modeling language of the M.C. SPIN
- The syntax is C-like
- A system in PROMELA consists of a set of *processes* that interact by means of:
  - shared variables
  - communication channels
    - rendez-vous communications
    - buffered communications
- Processes can be created dynamically
- Allows for both synchronous and asyncronous composition of processes (though asynchronous interaction more natural)

The PROMELA language: example

Example: A Mutual Exclusion Algorithm

```
bool turn;
bool flag[2];

proctype User(bool pid) {
  flag[pid] = 1;
  turn = 1-pid;
  (flag[1-pid] == 0 || turn == pid);
  /* Begin of critical section */
  ...
  /* End of critical section   */
  flag[pid] = 0;
}
init { run User(0); run User(1) }
```

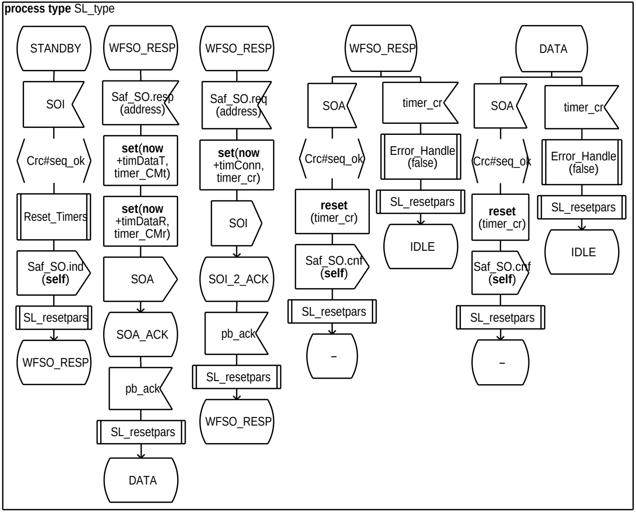


The SDL language

- An ITU standard
- Allows for booleans, enumerative and bounded integers as data types
- Allows for representing TIME (time elapse, clocks, ...)
- represents states, message I/O, conditions, clock operations, subroutines
- Allows for both synchronous and asyncronous composition of processes (though asynchronous interaction more natural)

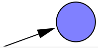
The SDL Language: example

Example: the Safety Layer protocol



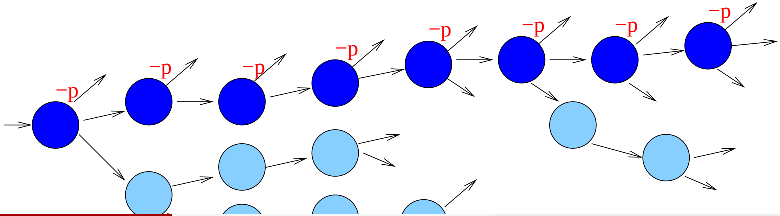
Safety properties

- bad events never happen
  - deadlock: two processes waiting for input from each other, the system is unable to perform a transition.
  - no reachable state satisfies a “bad” condition, e.g. never two processes in critical section at the same time
- can be refuted by a finite behaviour
- Ex.: it is never the case that p.



Liveness properties

- Something desirable will eventually happen
  - sooner or later this will happen
- can be refuted by infinite behaviour



Fairness properties

- Something desirable will happen infinitely often
  - important subcase of liveness
  - whenever a subroutine takes control, it will always return it (sooner or later)
- can be refuted by infinite behaviour
  - a subroutine takes control and never returns it