Embedded Systems Design and Modeling

Chapter 14
Equivalence and Refinement

Motivation

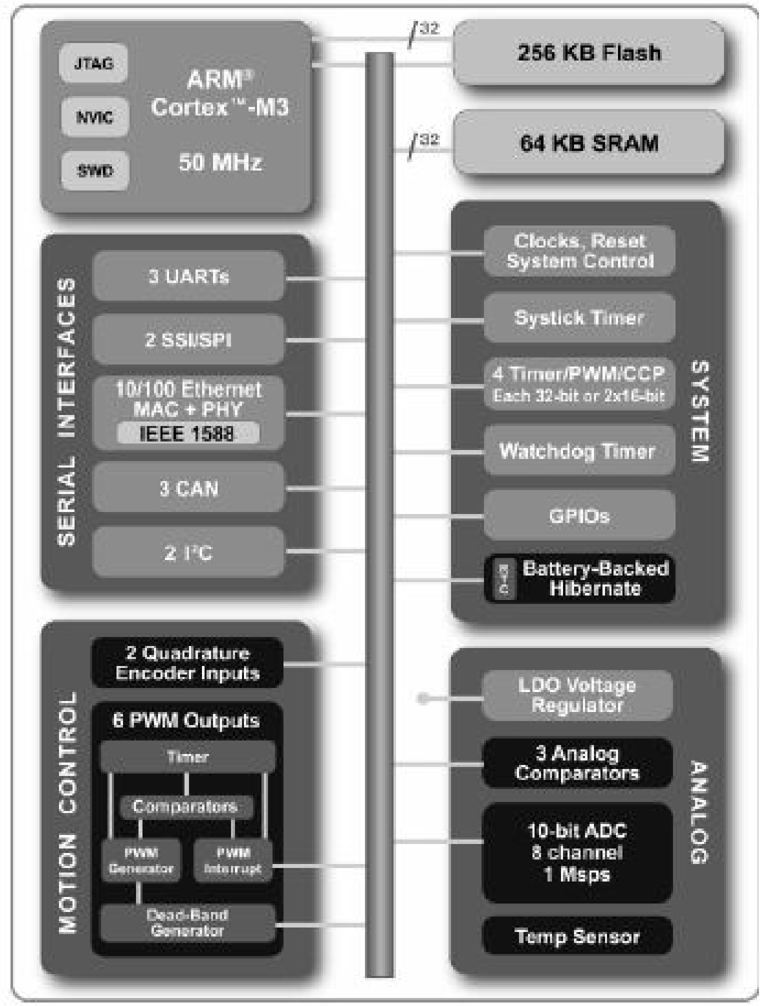
- Why do we need to compare models and systems?
  - Move up and down the abstraction levels
  - Verify the correctness of design as we go down in the synthesis path
  - Check conformance with a specification
  - Optimize a model by reducing complexity
  - Check if component substitution is OK
  - Anything else?

Component Substitution

Component Substitution

Can we replace one component in a system by another and be assured that it will continue to work correctly?

What if we replace the Cortex-M3 core by a Cortex-M4?

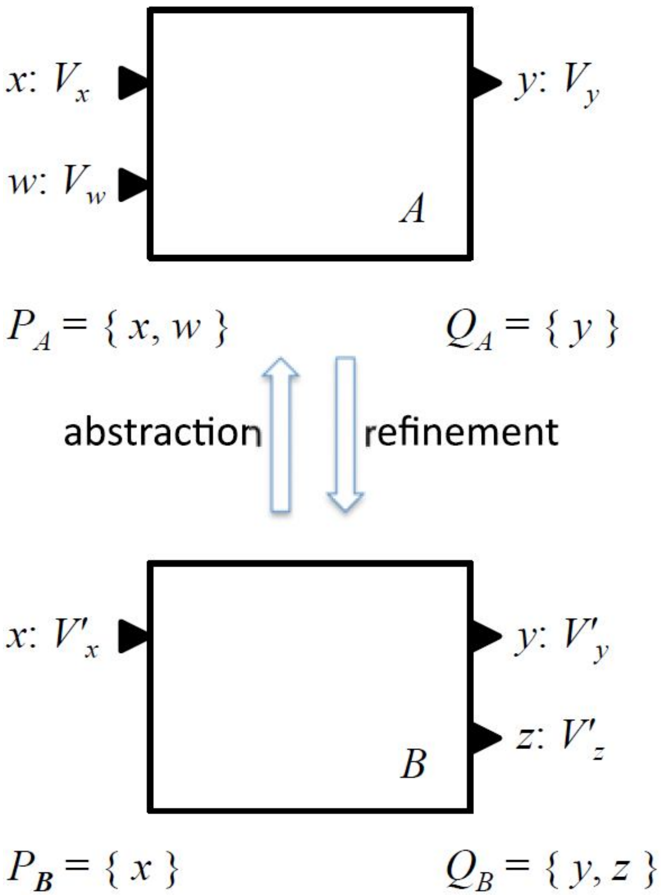


Main Questions

- How can we compare two models, e.g., state machines?
  - Are they “equivalent”?
  - What is the definition of “equivalent”?
  - Does one do “more” than the other? (e.g., exhibit different behaviors? Produce different outputs?)
  - Can one represent ALL behavior of the other?
  - What is the effect of the environment on the equivalence?

Type Refinement

- If we want to replace A by B in some environment, the ports and their types impose four constraints:



Embed

(1) $P_B \subseteq P_A$

(2) $Q_A \subseteq Q_B$

(3) $\forall\, p \in P_B, \quad V_p \subseteq V'_p$

(4) $\forall\, q \in Q_A, \quad V'_q \subseteq V_q$

4 Constraints of Type Refinement

1. B should not require some input signal that the environment does not provide.
2. B should produce all the output signals that the environment may require.
3. If the environment provides a value v on an input port p that is acceptable to A, then if p is also an input port of B, then the value is also acceptable to B.
4. If B produces a value v on an output port q, then if q is also an output port of A, then the value must be acceptable to any environment in which A can operate.

Type Equivalence

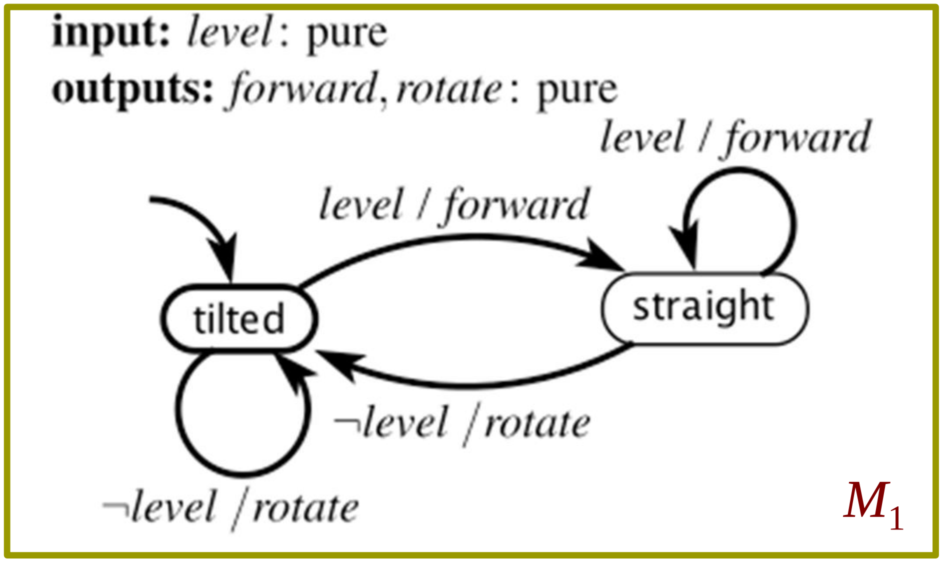
- If B is a type refinement of A, and A is a type refinement of B, then we say that A and B are type equivalent:
  - They have the same input and output ports, and the types of the ports are the same.
- Type equivalence is necessary but not sufficient to replace one machine with another:
  - If A is spec and B is implementation, A imposes more constraints than just data types
  - Functional conformity is also required

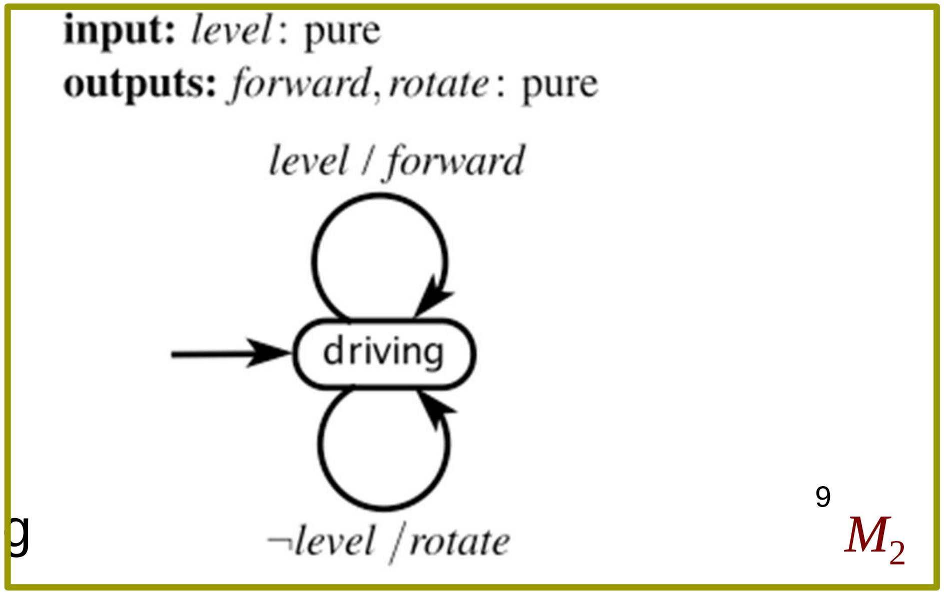
Language Equivalence

- Language L(M) of a state machine M:
  - The set of all behaviors for that state machine
- Two machines are language equivalent if they have the same language.
  - For every input sequence, the two machines must produce the same output sequence.
- Example in the next slides

Language Equivalence 1st Example

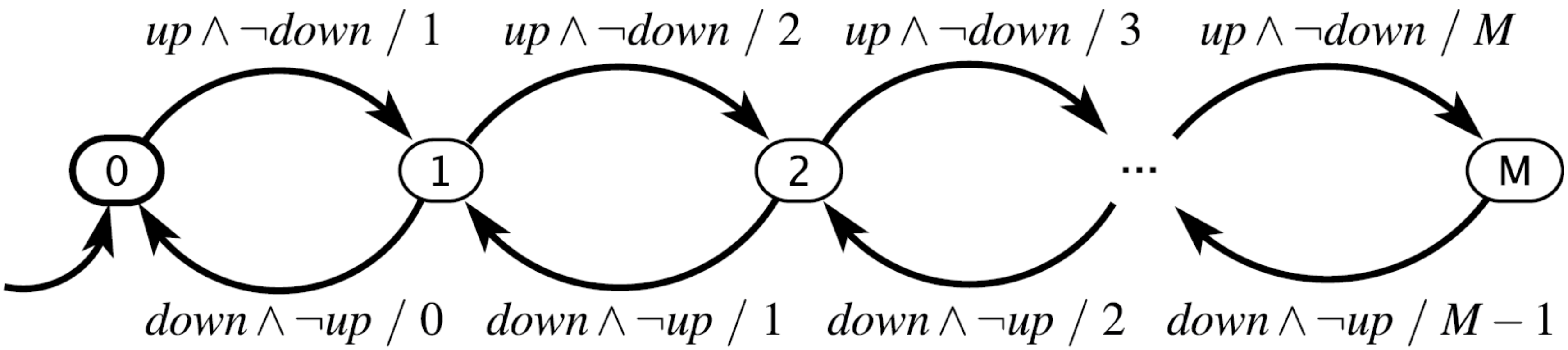
- Consider machines M1 and M2:
- Type equivalence?
  - Actor models have the same input ports and the same output ports.
  - The ports have the same types.
- Language equivalence?
  - For every input sequence, the two machines produce the same output sequence.



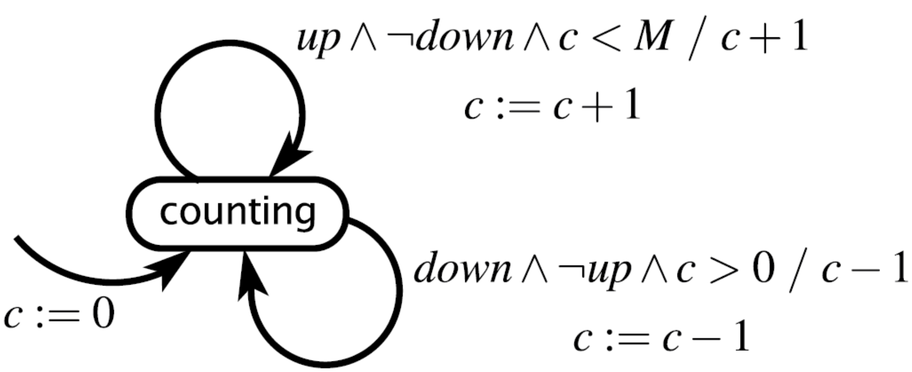


Language Equivalence 2nd Example

inputs: *up*, *down* : pure
output: *count* : $\{0, \cdots, M\}$

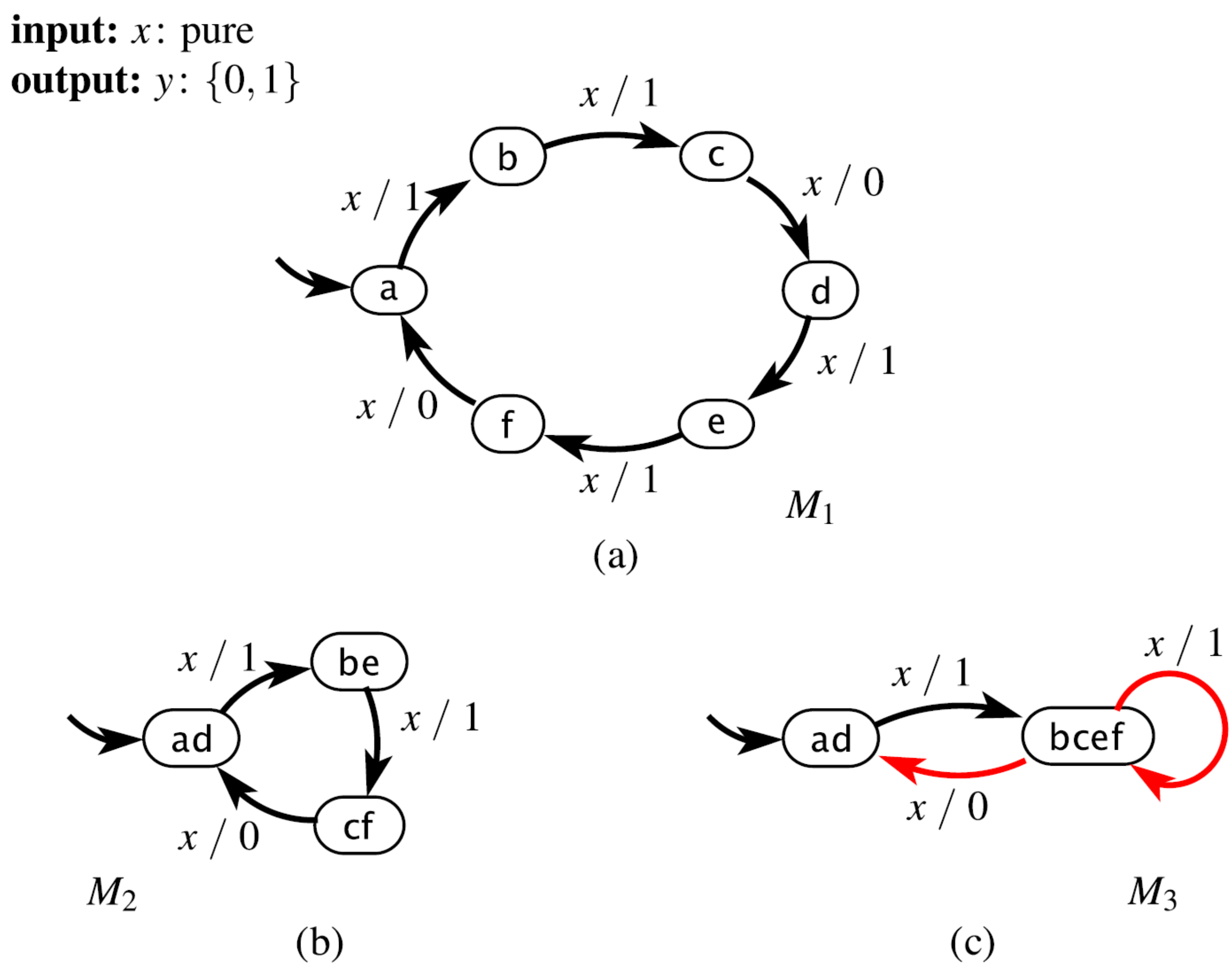


variable: c: $\{0, \cdots, M\}$
inputs: *up*, *down*: pure
output: *count*: $\{0, \cdots, M\}$



Language Equivalence 3rd Example

- M1 and M2 produce the output sequence for any input sequence
- M1 and M2 are language equivalent



Language Containment/Refinement

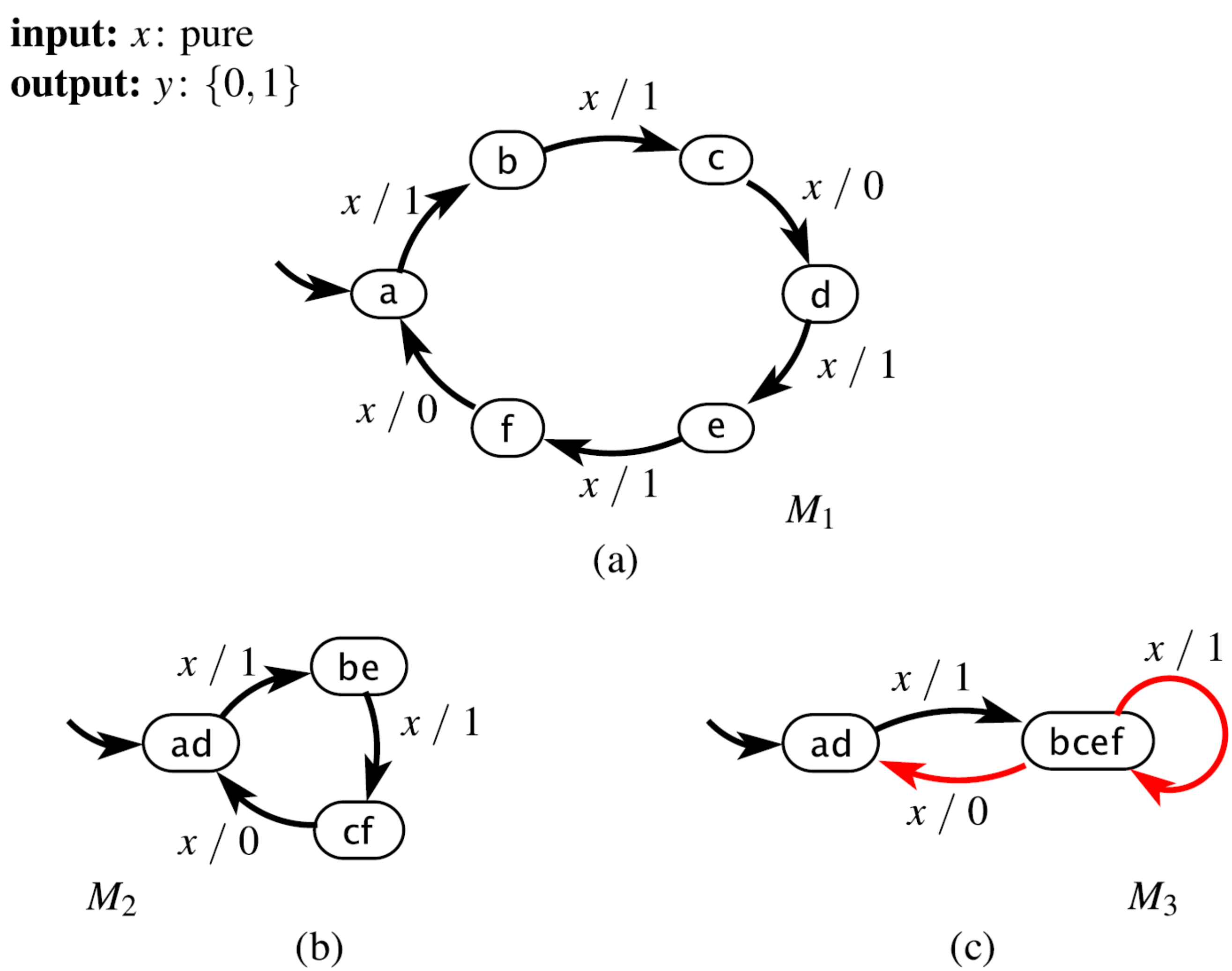
- If for two state machines A and B, L(A) is a subset of L(B), then:
  - All behaviors of A are the same as B
  - But B has behaviors that A does not
  - This is called language containment
  - A is a language refinement of B
  - B is a language containment of A

Language Containment/Refinement

- Language refinement shows the suitability of A as a replacement for B:
  - Every behavior of B is acceptable to an environment =>
  - Every behavior of A is acceptable to that environment =>
  - A can substitute for B in that environment
  - Any LTL formula about inputs, outputs, and behavior (but not states) that holds for B also holds for A
  - B may be a spec/higher level model, A may be an implementation/lower level model

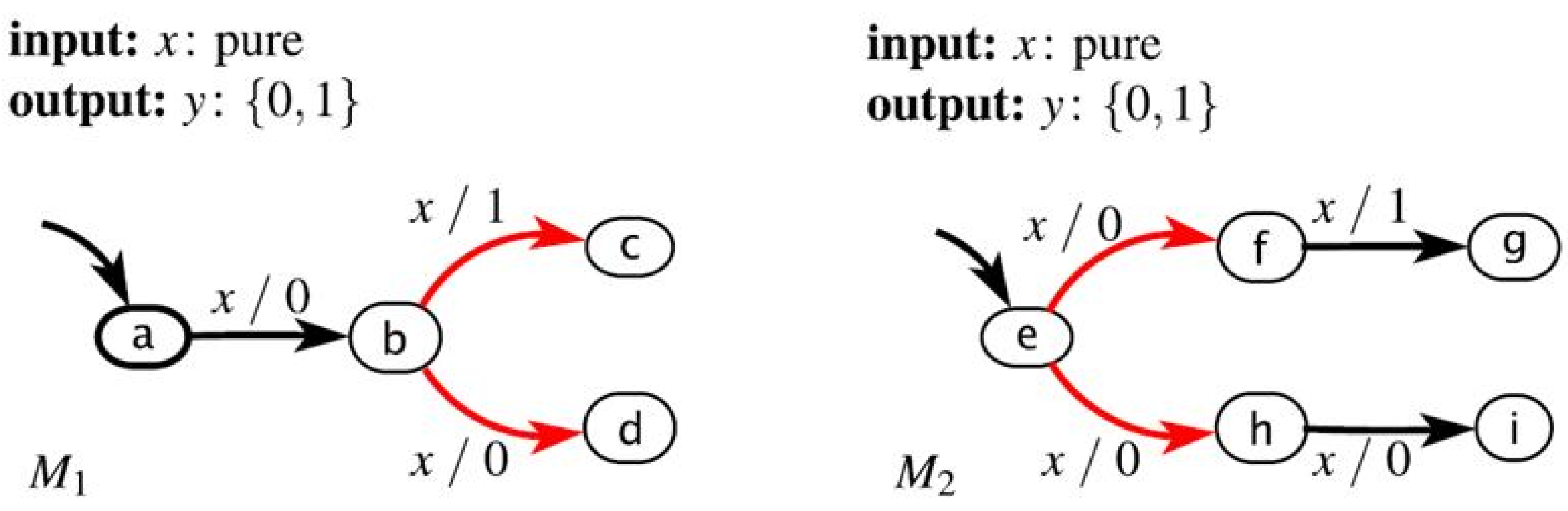
Language Containment Example

- M3 can produce any output sequence that M1 and M2 can
- But can also produce other outputs
- M1 and M2 are language refinements of M3



Simulation

- Language equivalence is not enough in general:
  - These two machines are language equivalent but have different state structures
  - In 2nd transition: M1 can do something that M2 can never match => M1 simulates M2, M1 cannot replace M2
  - In 2nd transition: M2 can do everything that M1 can => M2 can replace M1



Simulation: Matching Game

- M1 simulates M2?
- Play a game where:
  - M2 gets to move first in each round
  - Both machines in their initial states
  - M2 moves first by reacting to an input valuation
  - If nondeterministic choice, then it is allowed to make any choice, output valuation is created
  - M1 has to react to the same input valuation that M2 reacted to
  - If nondeterministic choice, it must make a choice that matches the output of M2
  - If there are multiple such choices, it must select one without knowledge of the future inputs or future moves of M2
  - Its strategy should be to choose one that enables it to continue to match M2, regardless of what future inputs arrive or future decisions M2 makes.

Matching Game Result

- M1 wins this game (M1 simulates M2) if it can always match the output symbol of M2 for all possible input sequences
- If in any reaction M2 can produce an output symbol that M1 cannot match, then M1 does not simulate M2.
- A simulation relation is complete if it includes all possible plays of the game.
- It must account for ALL reachable states of M2 (the machine that moves first) because M2’s moves are unconstrained
- M1’s moves are constrained by the need to match M2
- It is not necessary to account for all of its reachable states

Formal Model

- M1 simulates M2 if there is a subset S of States2×States1 such that:

1. $(\mathit{initialState}_2, \mathit{initialState}_1) \in S$, and

2. If $(s_2, s_1) \in S$, then $\forall\, x \in \mathit{Inputs}$, and
   $\forall\, (s_2', y_2) \in \mathit{possibleUpdates}_2(s_2, x)$,
   there is a $(s_1', y_1) \in \mathit{possibleUpdates}_1(s_1, x)$ such that:

   (a) $(s_2', s_1') \in S$, and

   (b) $y_2 = y_1$.

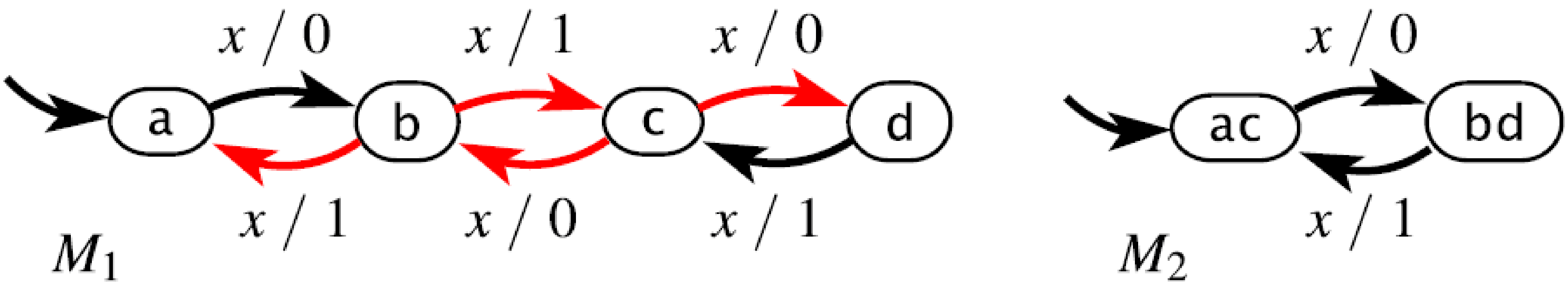
Properties

- Simulation is transitive:
  - If M1 simulates M2 and M2 simulates M3, then M1 simulates M3
- Simulation relation is non-unique:
  - When a machine M1 simulates another machine M2, there may be more than one simulation relation
- Example in the next slide

Non-Uniqueness Example

- M1 simulates M2 in 3 ways:

input: x: pure
output: y: $\{0, 1\}$



$$S_{2,1} = \{(\mathsf{ac}, \mathsf{a}), (\mathsf{bd}, \mathsf{b})\}$$

$$S_{2,1} = \{(\mathsf{ac}, \mathsf{a}), (\mathsf{bd}, \mathsf{b}), (\mathsf{ac}, \mathsf{c})\}$$

Embedded $S_{2,1} = \{(\mathsf{ac}, \mathsf{a}), (\mathsf{bd}, \mathsf{b}), (\mathsf{ac}, \mathsf{c}), (\mathsf{bd}, \mathsf{d})\}$

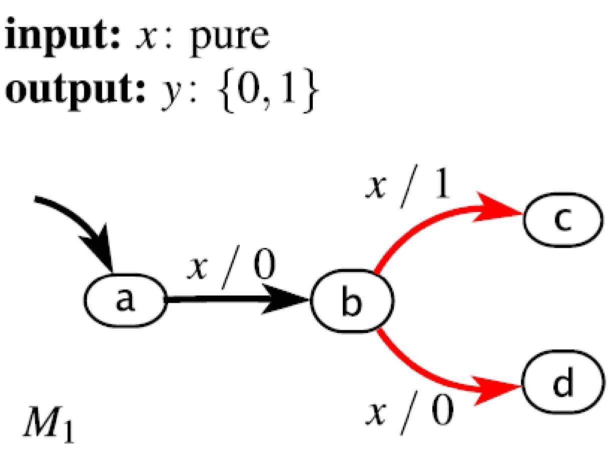
Simulation vs. Language Containment

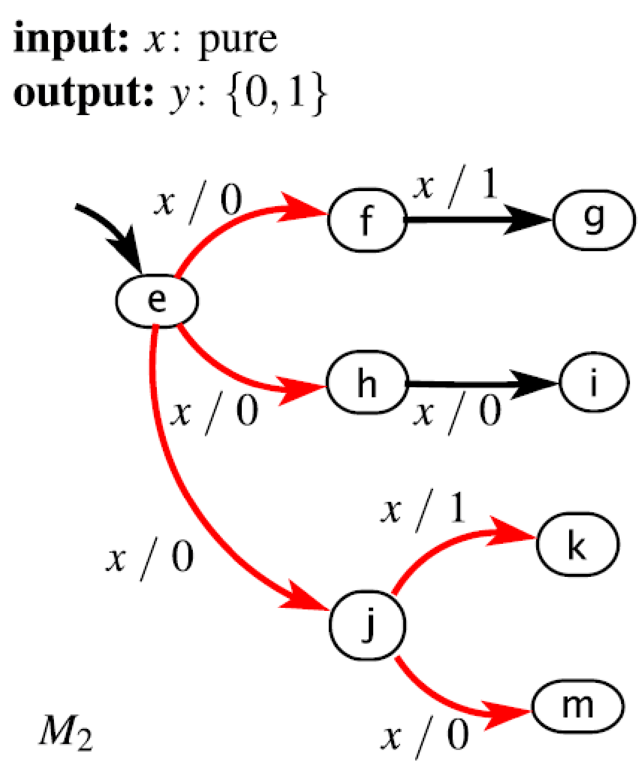
- Simulation is typically used to relate a simpler specification M1 to a more complicated realization M2
- When M1 simulates M2, then the language of M1 contains the language of M2
- Theorem: if M1 simulates M2, then L(M2) is a subset of L(M1)
  - Note: The opposite is NOT true!
- Simulation relation differs from language containment only for nondeterministic FSMs!

Bisimulation

- Is it possible to have two machines M1 and M2 where M1 simulates M2 and M2 simulates M1, and yet the machines are observably different?
  - Yes, even though by previous theorem their languages must be identical
- Example in the next slide:
  - M1 simulates M2
  - M2 simulates M1
  - But they may act differently!

Bisimulation Example





Note that the trick is in ability to alternate which machine moves first!

$$S_{2,1} = \{(\mathsf{e},\mathsf{a}), (\mathsf{f},\mathsf{b}), (\mathsf{h},\mathsf{b}), (\mathsf{j},\mathsf{b}), (\mathsf{g},\mathsf{c}), (\mathsf{i},\mathsf{d}), (\mathsf{k},\mathsf{c}), (\mathsf{m},\mathsf{d})\}$$

$$S_{1,2} = \{(\mathsf{a},\mathsf{e}), (\mathsf{b},\mathsf{j}), (\mathsf{c},\mathsf{k}), (\mathsf{d},\mathsf{m})\}$$

Bisimulation Definition

- M1 is bisimilar to M2 (or M1 bisimulates M2) if they are type equivalent and we can play the matching game so that in each round either machine can move first

Summary

- M2 is a type refinement of M1:
  - M2 can replace M1 without causing a type conflict.
- M2 is a language refinement of M1:
  - M2 can produce only output sequences that M1 can produce, given the same input sequences.
- M2 is a simulation refinement of M1 (equivalently, M1 simulates M2):
  - At every reaction, M2 can produce only outputs that M1 can produce.
- M2 is bisimilar to M1:
  - At every reaction either machine can produce only outputs that the other can produce.
- In all cases, if M1 is “valid” in a system, then so is M2, where only the meaning of “valid” varies.
- Alternative terminology: M2 implements M1 (here, M1 is taken to be a specification).

Homework Assignments

- Chapter 14: your choice!
- Due date: Tuesday 1404/3/6