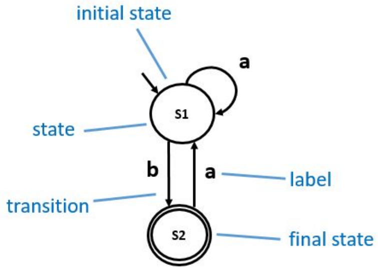
Rigorous Methods for Software Engineering (F21RS-F20RS)
Automata Based Model Checking: How It Works (Part 1)

Andrew Ireland
Department of Computer Science
School of Mathematical and Computer Sciences
Heriot-Watt University
Edinburgh

Overview

- Finite state automata and Büchi automata.
- Promela and automata.
- Constructing asynchronous and synchronous products.
- Turning finite computations into infinite computations.

Finite State Automata

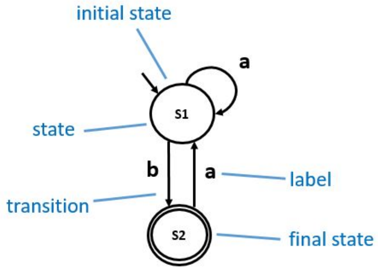


- Labels (L)
- States (S)
- Transitions (T)
- Initial states (I)
- Final states (F)

A **finite state automaton** (FSA) is a mathematical representation of a system (i.e. computation): A FSA is represented in terms of **states** and **transitions**:

- A **state** denotes a specific phase within a system.
- A **transition** defines how **states** change over time.

Finite State Automata



L: $\{a, b\}$

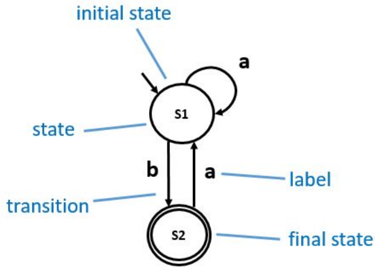
S: $\{S1, S2\}$

T: $\{(S1, a, S1), (S1, b, S2), (S2, a, S1)\}$

I: $\{S1\}$

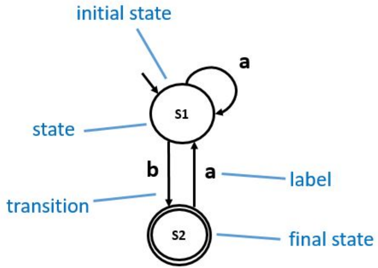
F: $\{S2\}$

Finite State Automata



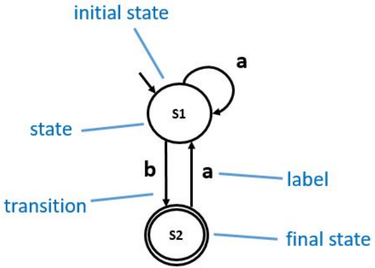
- Which of the following words will the automaton **Accept** and **Reject**?
  b
  abbaa
  bb
  a
  ab
  aabab

Finite State Automata



- **Accept** words:
  b
  ab
  aabab
- **Reject** words:
  a
  bb
  abbaa

Finite State Automata

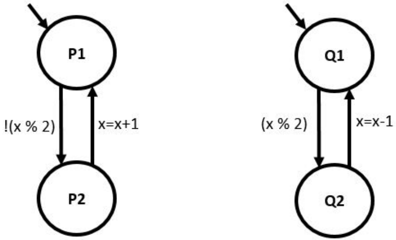


- A regular automaton **accepts** a word if and only if there is a computation that ends in a final state (i.e. an **accept** state).
- A **Büchi** automaton **accepts** word if and only if there is a computation that visits a final state (i.e. an **accept** state) infinitely often - an **acceptance cycle**.

Promela and Automata

A **Promela** process can be represented as an automaton, e.g.

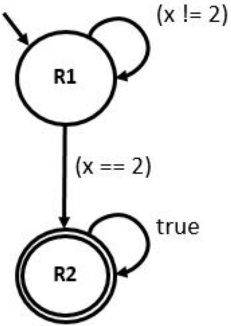
```
ltl R { <>(x == 2) }

int x = 0;

active proctype P(){
do
:: !(x % 2)  -> x = x+1;
od}

active proctype Q(){
do
:: (x % 2)  -> x = x-1;
od}
```



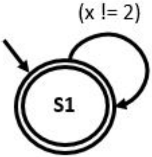
Note that % denotes the modulo operator, i.e. the statement `(X % 2)` in Promela is equivalent to stating that `X` is odd.

Promela and Automata

A **LTL** property can also be represented as an automaton, e.g.



`<>(x == 2)`



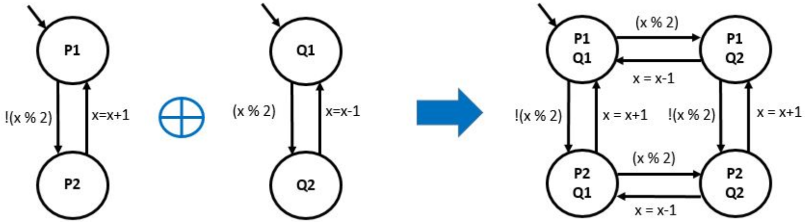
`[](x != 2)`

Constructing an Asynchronous Product

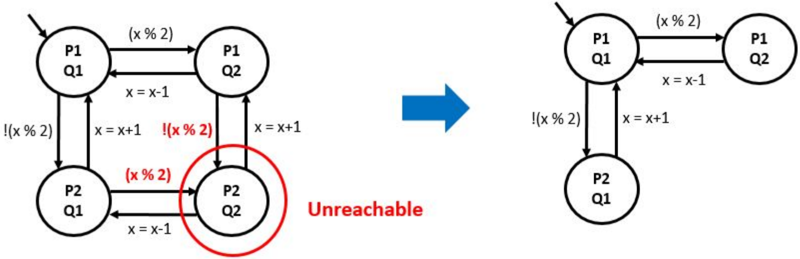
Combining two automata (i.e. processes) so that their **transitions are interleaved** gives rise to an automaton that is known as the **asynchronous product**:

- Each state within the **asynchronous product** denotes a composite state, involving one state from each of the two component automata (i.e. processes).
- Each transition within the **asynchronous product** denotes an individual transition from one of the component automata (i.e. processes).

Constructing an Asynchronous Product - An Example

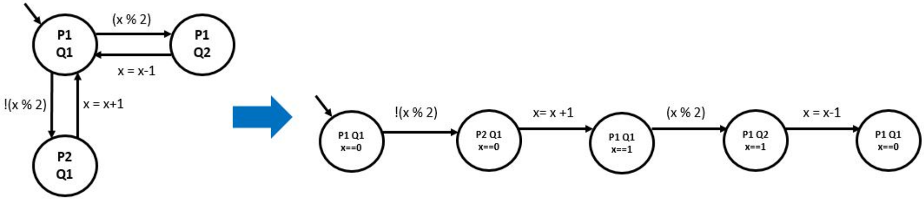


Pruning Unreachable States – An Example



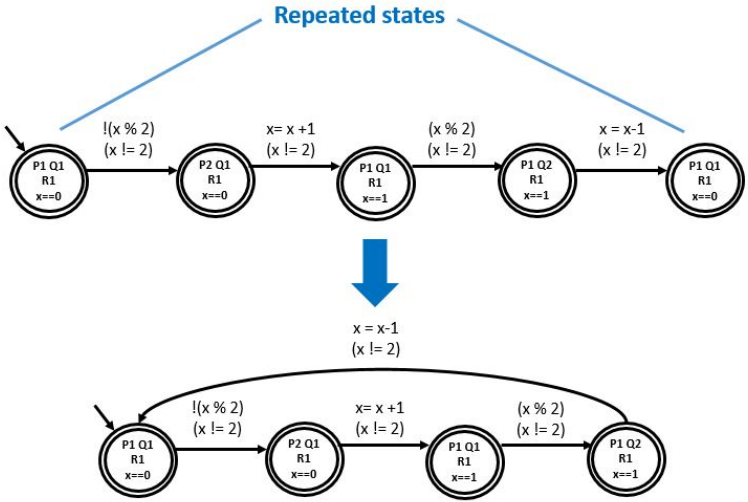
- The process of constructing a product will often give rise to **unreachable** states.
- **Unreachable** states are analogous to **unreachable** code and should be pruned.

Expanding a Finite State Automaton - An Example



- Note that the automaton on the right is known as an **expanded finite state automaton** as it includes the variable assignments associated with a state.
- Although all Promela data types are finite, an explicit expansion may lead to space issues – more later (Part 2).

Exploiting Repeated States - An Example



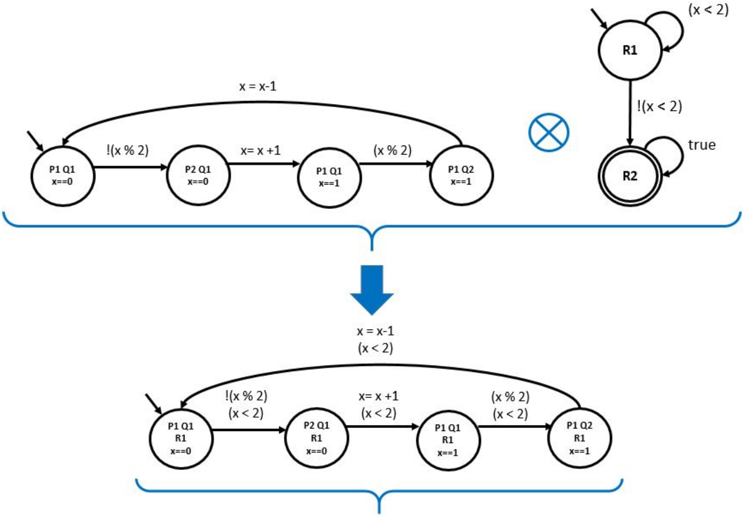
Note that when an expansion results in a repeated state then a loop is introduced.

Constructing a Synchronous Product

Combining two processes (i.e. automata) so that their **transitions are joint** gives rise to an automaton that is known as the **synchronous product**:

- Each state within the **synchronous product** denotes a composite state, involving one state from each of the two component automata (i.e. processes).
- Each transition within the **synchronous product** denotes a joint transition involving one transition from each of the two component automata (i.e. processes).

Constructing a Synchronous Product – An Example



$(P \oplus Q) \otimes R$

(Synchronous Product)

Note that the transition from state **R1** to state **R2** never occurs.

Stutter Steps: Turning the Finite into the Infinite

- Part 2 will explain how the **Büchi automaton** representation and the notion of an **acceptance cycle** provide the logical foundation for the verification technique known as model checking.
- **Problem:**
  - A **Büchi automaton** either accepts or rejects infinite inputs, i.e. it represents an **infinite computation**.
  - If a model (i.e. Promela program) represents a **finite computation**, i.e. it terminates or deadlocks, then how can it be represented as a **Büchi automaton**?
- **Solution:**
  - Add an extra transition to each terminating state S_k associated with the model, i.e. $(S_k, \ldots, S_k)$.
  - Label each of these transitions with **skip**, i.e. (S_k, **skip**, S_k).

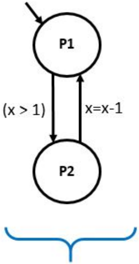
  Such extra transitions, so called **stutter steps**, turn a **finite computation** into an **infinite computation** without affecting the semantics, i.e. **skip** is equivalent to **true**.

Stutter Steps: Turning the Finite into the Infinite

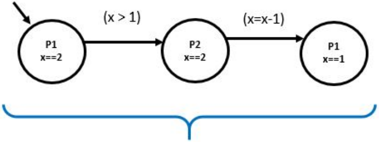
```
byte x = 2;

active proctype P()
{
   do
   :: (x > 1) -> x = x-1;
   od
}
```

Note that process **P** deadlocks with **x** equal to **1**, i.e. process **P** represents a **finite computation**.

Stutter Steps: Turning the Finite into the Infinite



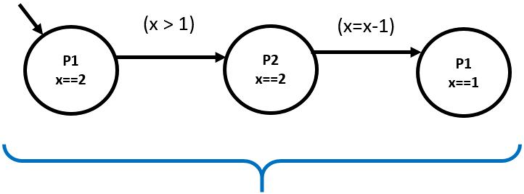
P
(Finite State Automaton)



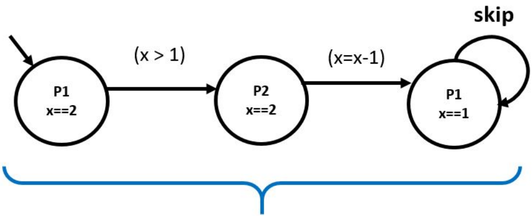
P
(Expanded Finite State Automaton)

The **expanded finite state automaton** (above right) clearly shows the finite nature of process **P**.

Stutter Steps: Turning the Finite into the Infinite



P: Finite computation



P': Infinite computation

Note that the semantics of **P** and **P'** are equivalent. Note also that typically **stutter steps** are left implicit for presentation purposes.

Summary

Learning outcomes:

- Understand the notions of a **finite state automaton** and a **Büchi automaton** and how they are used to represent **Promela** models and **LTL** properties.
- Understand the notion of a **expanded finite state automaton** and how to construct them.
- Understand notions of an **asynchronous product** and a **synchronous product** and how to construct them.
- Understand the notion of a **stutter step** and how it can be used to turn a finite computation into an infinite computation while preserving the semantics.

Recommended reading:

- "Model Checking", E.M. Clarke, O. Grumberg, D.A. Peled, MIT Press, 1999.
- "Practical Formal Methods Using Temporal Logic", M. Fisher, Wiley, 2011.
- **Büchi Store:** `http://buchi.im.ntu.edu.tw/`