G52CON:
Concepts of Concurrency

Lecture 17 Model Checking I

Brian Logan
School of Computer Science
bsl@cs.nott.ac.uk

Outline of this lecture

- model checking
- transition systems and properties
- example: simple transition system
- SMV description and specification languages
- truth of CTL formulas

Exercise 5

```
// Process 1

init1;
while(true) {
  c1 = 0;    // entry protocol
  while (c2 == 0) {};
  crit1;
  c1 = 1;    // exit protocol
  rem1;
}
```

```
// Process 2

init2;
while(true) {
    c2 = 0;    // entry protocol
    while (c1 == 0) {};
    crit2;
    c2 = 1;    // exit protocol
    rem2;
}
```

```
//shared variables
integer c1 == 1 c2 == 1;
```

Exercise 5a

```
// Process 1
init1;
while(true) {
   c1 = 0;   // entry protocol
   while (c2 == 0) {
       if (turn == 2) {
             c1 = 1;
             while (turn == 2) {};
             c1 = 0;
       }
   }
   crit1;
   turn = 2; // exit protocol
   c1 = 1;
   rem1;
}
```

```
// Process 2
init2;
while(true) {
      c2 = 0;   // entry protocol
      while (c1 == 0) {
            if (turn == 1) {
            c2 = 1;
            while (turn == 1) {};
            c2 = 0;
            }
      }
      crit2;
      turn = 1; // exit protocol
      c2 = 1;
      rem2;
}
```

`c1 == 1 c2 == 1 turn == 1`

Formal verification

Formal verification consists of three parts:

- a *description language* for describing the system to be verified;
- a *specification language* for describing the properties to be verified; and
- a *verification method* to establish whether the description of the system satisfies the specification

Proof-based approaches to verification

In a proof-based approach

- the system description is a set of formulas Γ in some logic
- the specification is another formula ϕ in the same logic
- the verification method consists of trying to find a proof that $\Gamma \vdash \phi$

This is time consuming and requires expertise on the part of the user.

Model-based approaches to verification

In a model-based approach

- the system is represented by a finite model M for an appropriate logic;
- the specification is a formula ϕ in the same logic; and
- the verification method consists of computing whether M satisfies ϕ ($M \models \phi$)

This process can be *automated* (model checking).

Model checking

- automatic, model-based, property verification approach, i.e., the specification describes a single property of the system rather than its complete behaviour;

- intended for concurrent, reactive systems, e.g., concurrent programs, embedded systems and computer hardware;

- post-development methodology.

Verifying properties by model checking

To verify that a program or system satisfies a property, we:

- describe the system using the description language of the model-checker;

- express the property to be verified using the specification language of the model checker; and

- run the model checker with the system description and property to be verified as inputs.

Model checking and temporal logic

Model checking is based on *temporal logic*

- in classical (propositional) logic, a model is an assignment of truth values to atomic propositions
- the models of temporal logic contain several states and a formula can be true in some states and false in others
- truth is *dynamic* in that formulas can change their truth values as the system evolves from state to state

In model checking, the models M are *transition systems* and the properties ϕ are formulas of temporal logic

How it works

When the model checker is run

- it generates a model (transition system), M, from the system description;
- converts the property to be verified into a temporal logic formula ϕ and;
- for every state s in M, checks whether s satisfies ϕ ($M, s \models \phi$)

If the model doesn't satisfy the formula most model checkers also output a trace of the system behaviour that causes the failure.

Transition systems

A transition system consists of a set of states and the transitions between them (a directed graph)

- the *states* are the states of the system being modelled
- states are labelled by a set of atomic propositions which are true in that state, e.g., “variable x has value 1”, “process 1 is in its critical section” etc.
- the *transitions* correspond to the atomic transitions of the system, e.g., atomic instructions or synchronized methods
- there may be many transitions from each state—one for each process that could go next in an interleaving

Example: simple transition system

```
// shared variables
integer x = 0; y = 0;
```

```
// Process 1
while(true) {
     x = 1;
     y = 100;
}
```

```
// Process 2
while(true) {
     x = y;
}
```

Atomic propositions:

p_0 true when `x == 0`

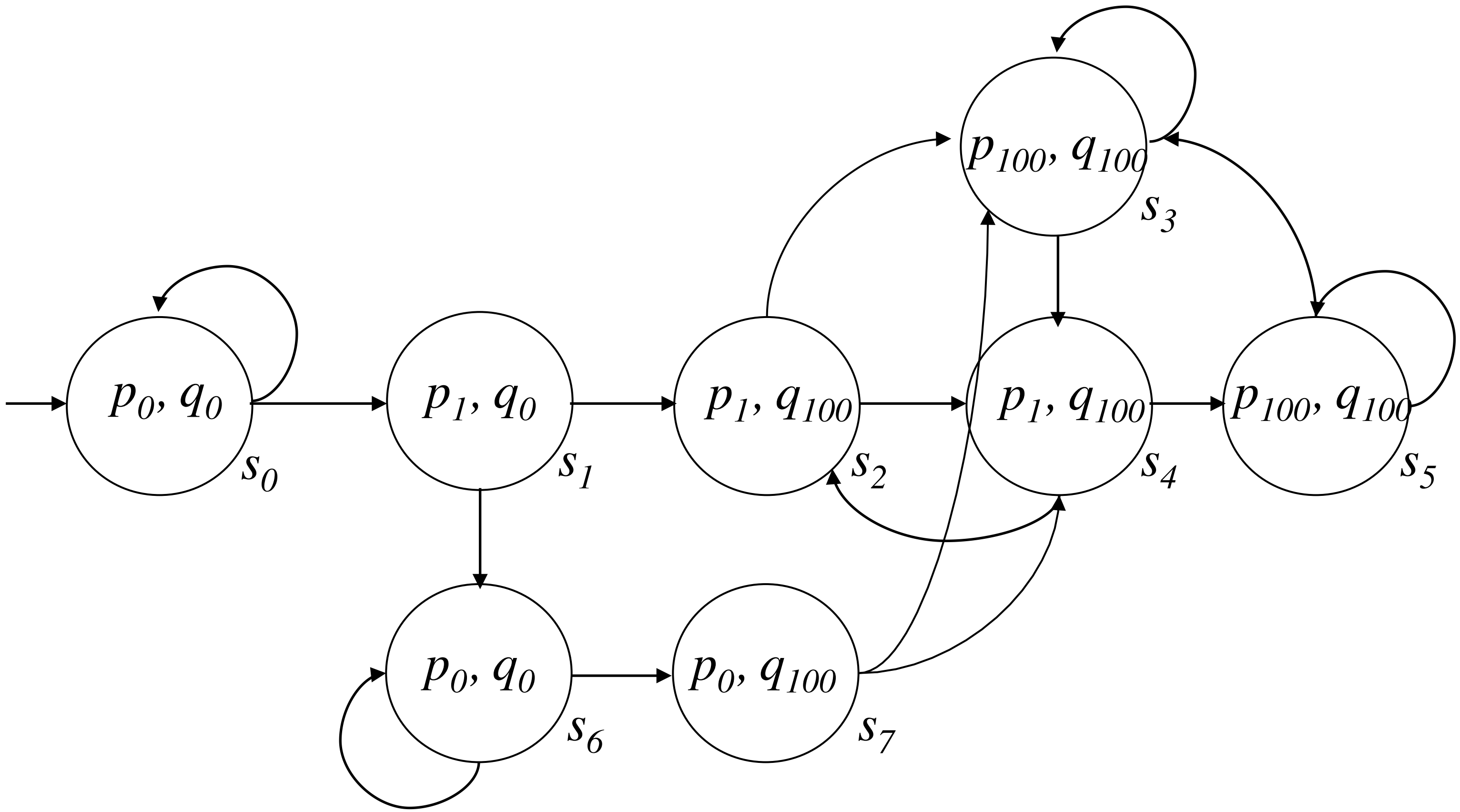
p_1 true when `x == 1`

p_{100} true when `x == 100`

q_0 true when `y == 0`

q_{100} true when `y == 100`

Example: simple transition system 2



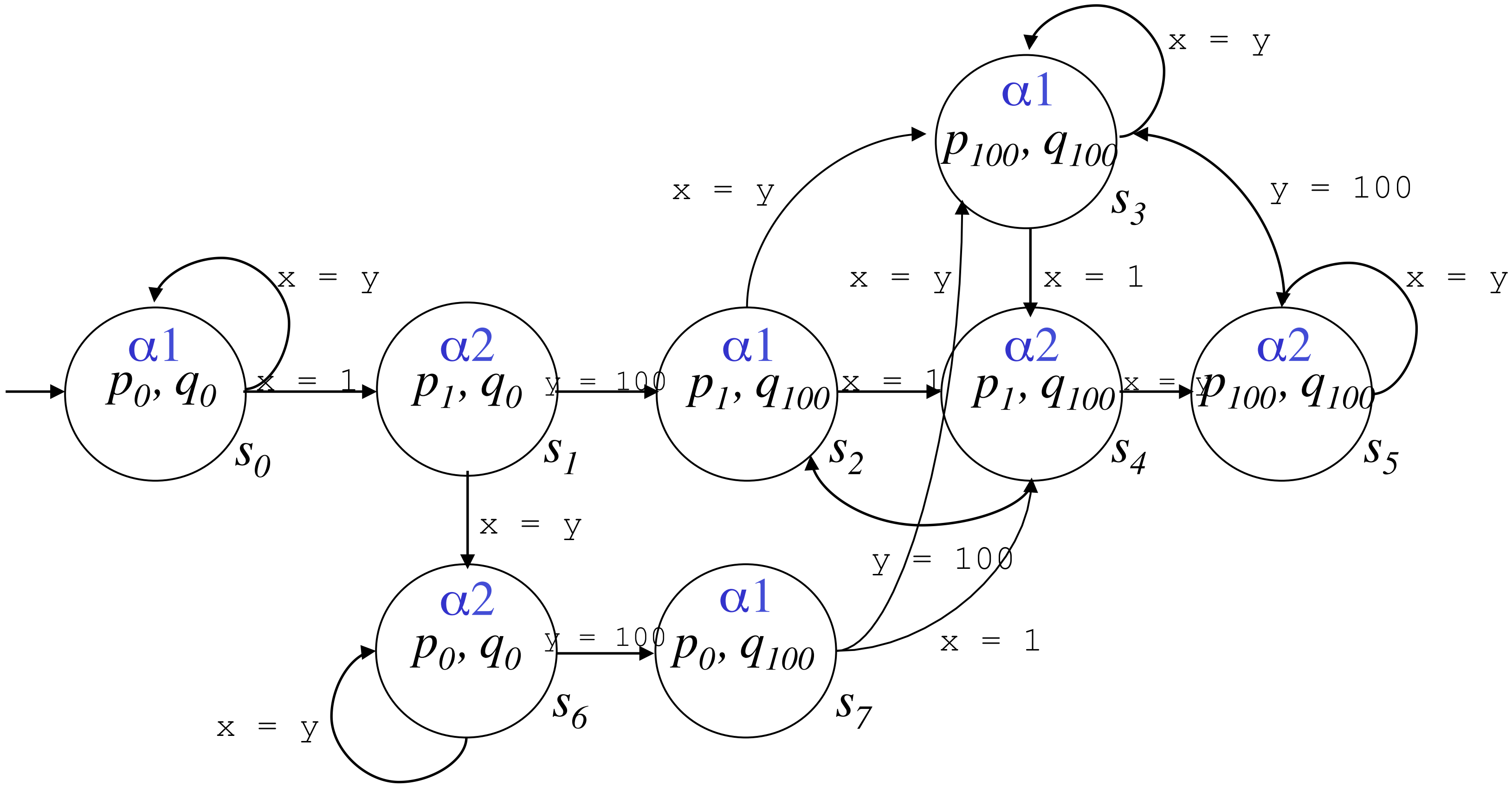
Example: simple transition system

```
// shared variables
integer x = 0; y = 0;
```

```
// Process 1
while(true) {
  α1
    x = 1;
  α2
    y = 100;
}
```

```
// Process 2
while(true) {
    x = y;
}
```

Example: simple transition system 2



The system description

Model checkers don't usually take program text as input:

- a system description at the program statement level may be too fine grained for the properties to be checked

- model checkers are also used to verify hardware systems, communication protocols, etc.

Instead, each model checker has its own description language and specification language.

Example: SMV model checker

```
MODULE main
VAR
  request: boolean;
  status : {ready, busy};
ASSIGN
  init(status) := ready;
  next(status) := case
                    request : busy;
                    1 : {ready, busy};
                  esac;
SPEC
  AG(request -> AF status = busy)
```

Specifying properties

The property of the system to be verified is expressed in the model checker's specification language

- many model checkers allow properties to be expressed directly in temporal logic (often using a simplified syntax)
- for example, the SMV model checker uses *Computation Tree Logic* (CTL) as its specification language

Syntax of CTL

CTL is a *branching-time temporal logic*

- a set of atomic propositions $p, q, r, \ldots$
- standard logical connectives: $\neg, \wedge, \vee, \rightarrow$
- **temporal connectives**: AX, EX, AF, EF, AG, EG, AU and EU
- formulas: $\phi = p \mid \neg\phi \mid \phi_1 \wedge \phi_2$ ... AX ϕ ... A[ϕ U φ] ...

Temporal connectives

- AX ϕ : on **A**ll paths, ϕ is true in the ne**X**t state
- EX ϕ : on som**E** path, ϕ is true in the ne**X**t state

- AF ϕ : on **A**ll paths, in some **F**uture state ϕ is true
- EF ϕ : on som**E** path, in some **F**uture state ϕ is true

- AG ϕ : on **A**ll paths, in all future states (**G**lobally) ϕ is true
- EG ϕ : on som**E** path, in all future states (**G**lobally) ϕ is true

- A[ϕ U φ] : on **A**ll paths, ϕ is true **U**ntil φ is true
- E[ϕ U φ] : on som**E** path, ϕ is true **U**ntil φ is true

Specifying properties of systems

Given some atomic propositions expressing properties of interest such as *ready*, *started*, *requested*, *acknowledged*, *enabled*, *deadlock* etc., we can express properties such as:

- there exits some state where *started* holds, but *ready* does not:

$$\mathrm{EF}\ (started \wedge \neg ready)$$

- a request for a resource will eventually be acknowledged:

$$\mathrm{AG}(requested \rightarrow \mathrm{AF}\ acknowledged)$$

- a process will eventually be permanently deadlocked:

$$\mathrm{AF}(\mathrm{AG}\ deadlock)$$

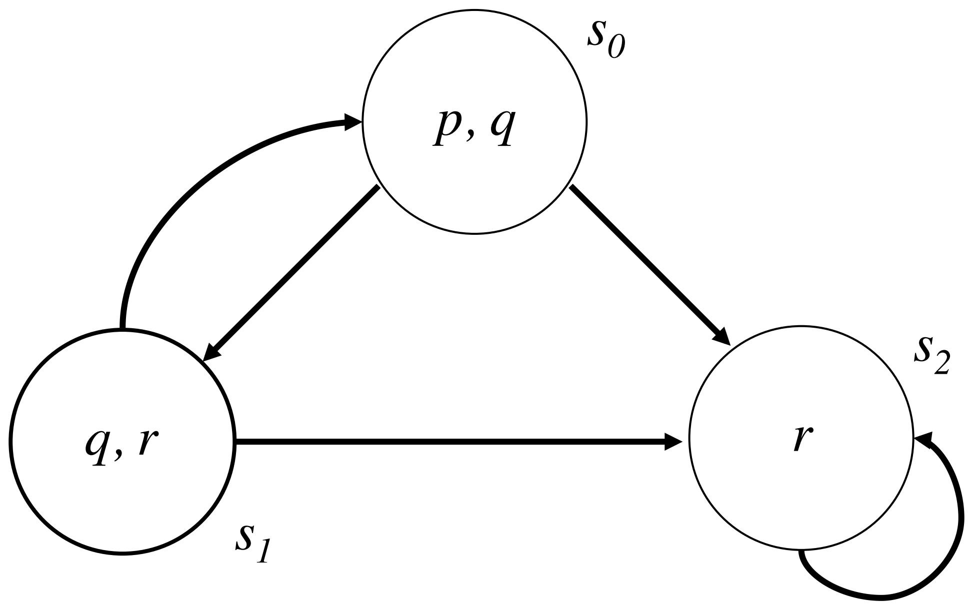
- from any state it is possible to get to a restart state:

$$\mathrm{AG}(\mathrm{AF}\ restart)$$

Semantics of CTL

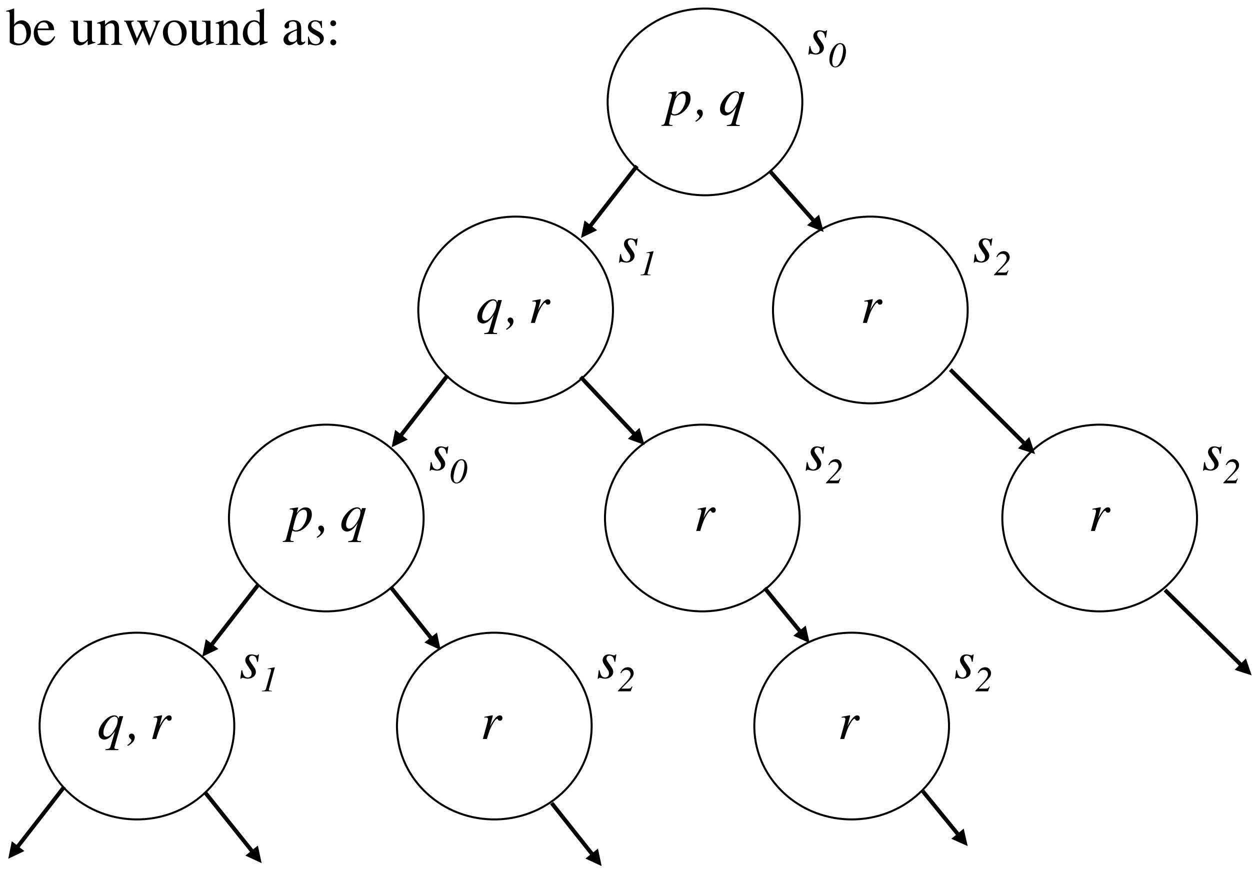
CTL formulas can be evaluated relative to the computation tree which is the unwinding of the transition system describing the system.

For example, the graph:



Unwinding the graph

Can be unwound as:



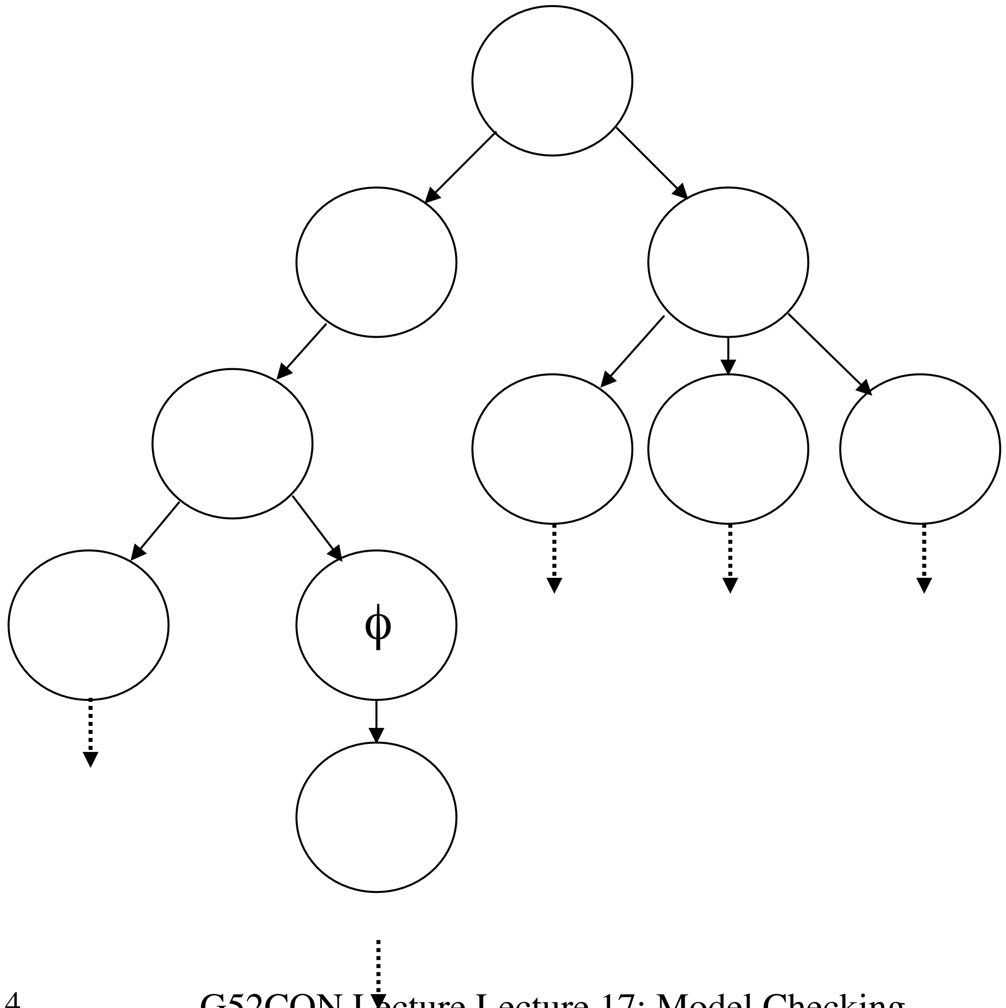
Interpreting temporal connectives

- M, *s* |= AX ϕ : in every next state starting in *s* ϕ holds
- M, *s* |= EX ϕ : in some next state starting in *s* ϕ holds
- M, *s* |= AF ϕ : for all computation paths starting in *s* there is some future state where ϕ holds
- M, *s* |= EF ϕ : there exits a computation path starting in *s* such that ϕ holds in some future state

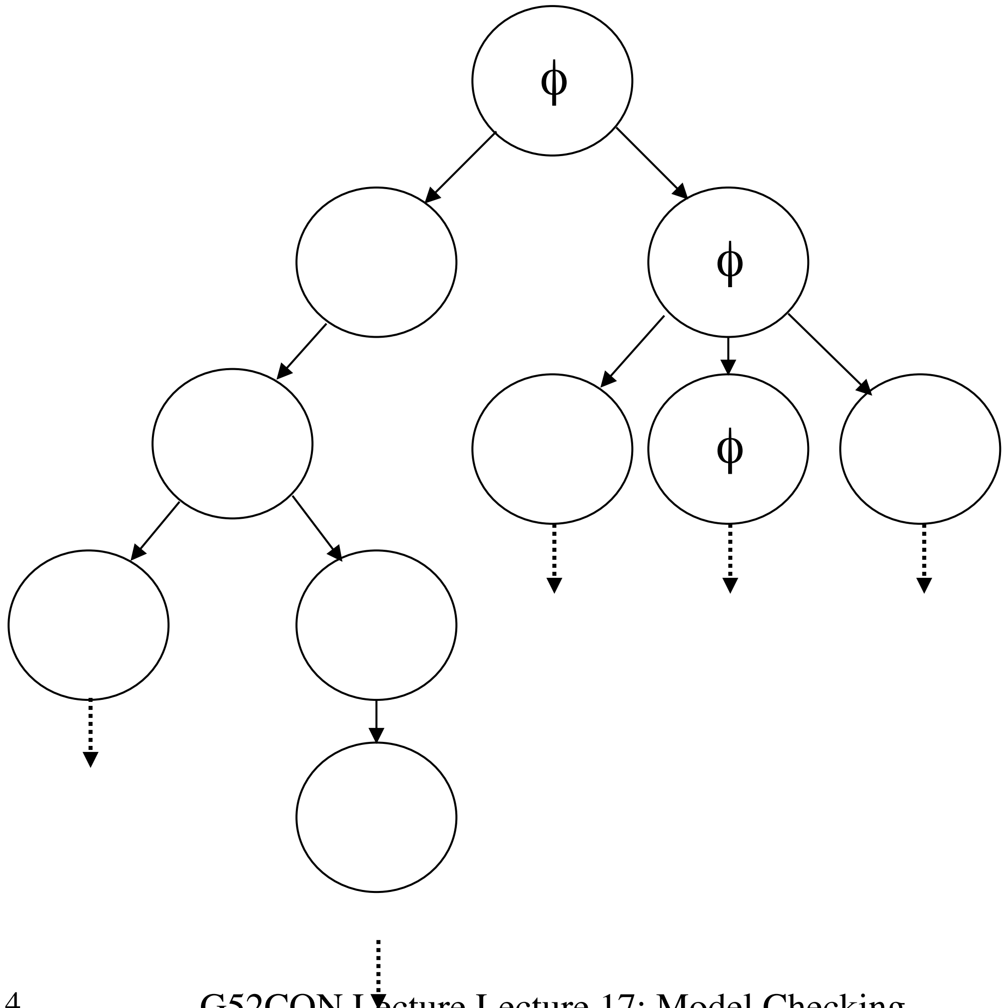
Interpreting temporal connectives 2

- M, $s \models AG\ \phi$: for all computation paths starting in s the property ϕ holds globally (in every state along the path including s)
- M, $s \models EG\ \phi$: there exists a computation path starting in s such that ϕ holds globally (in every state along the path including s)
- M, $s \models A[\phi_1\ U\ \phi_2]$: for all computation paths starting in s the property ϕ_1 holds in every state along the path (including s) until ϕ_2 holds
- M, $s \models E[\phi_1\ U\ \phi_2]$: there exists a computation path starting in s such that the property ϕ_1 holds in every state along the path (including s) until ϕ_2 holds

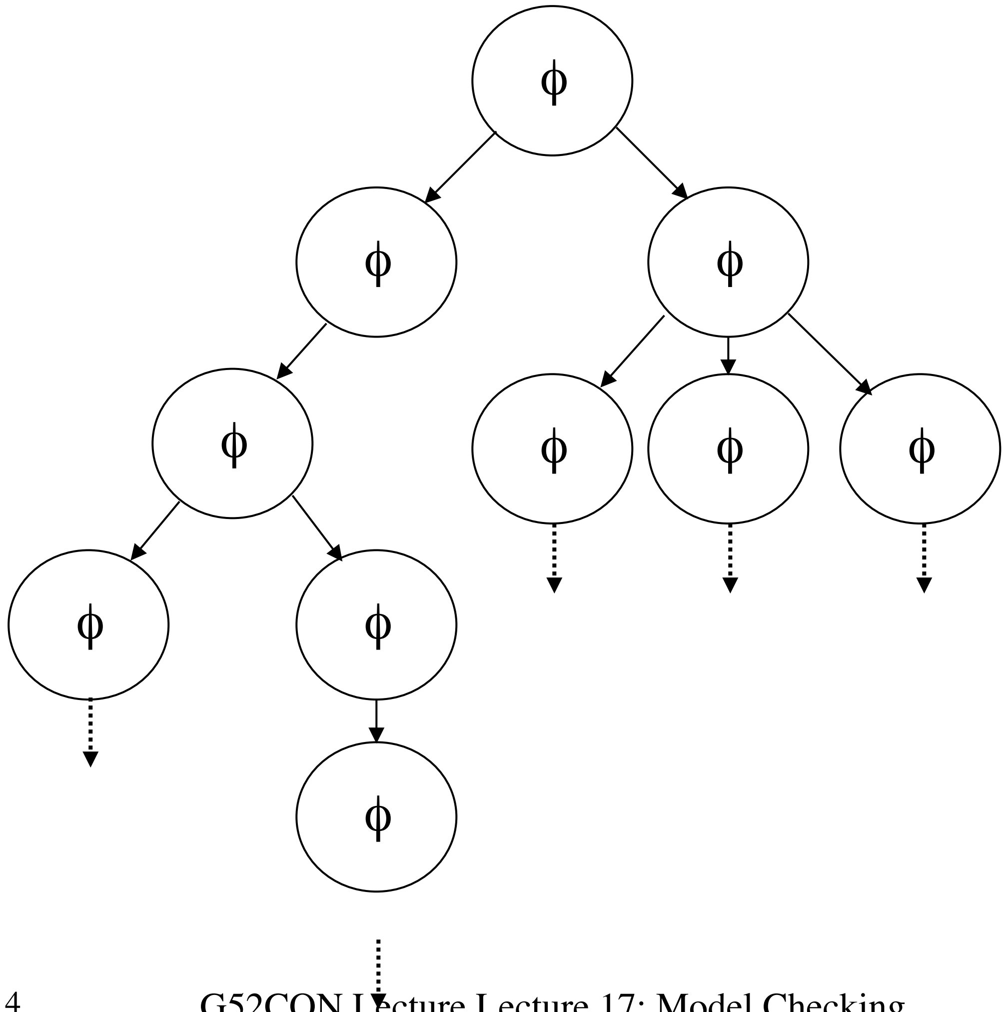
Example: a system which satisfies EF ϕ



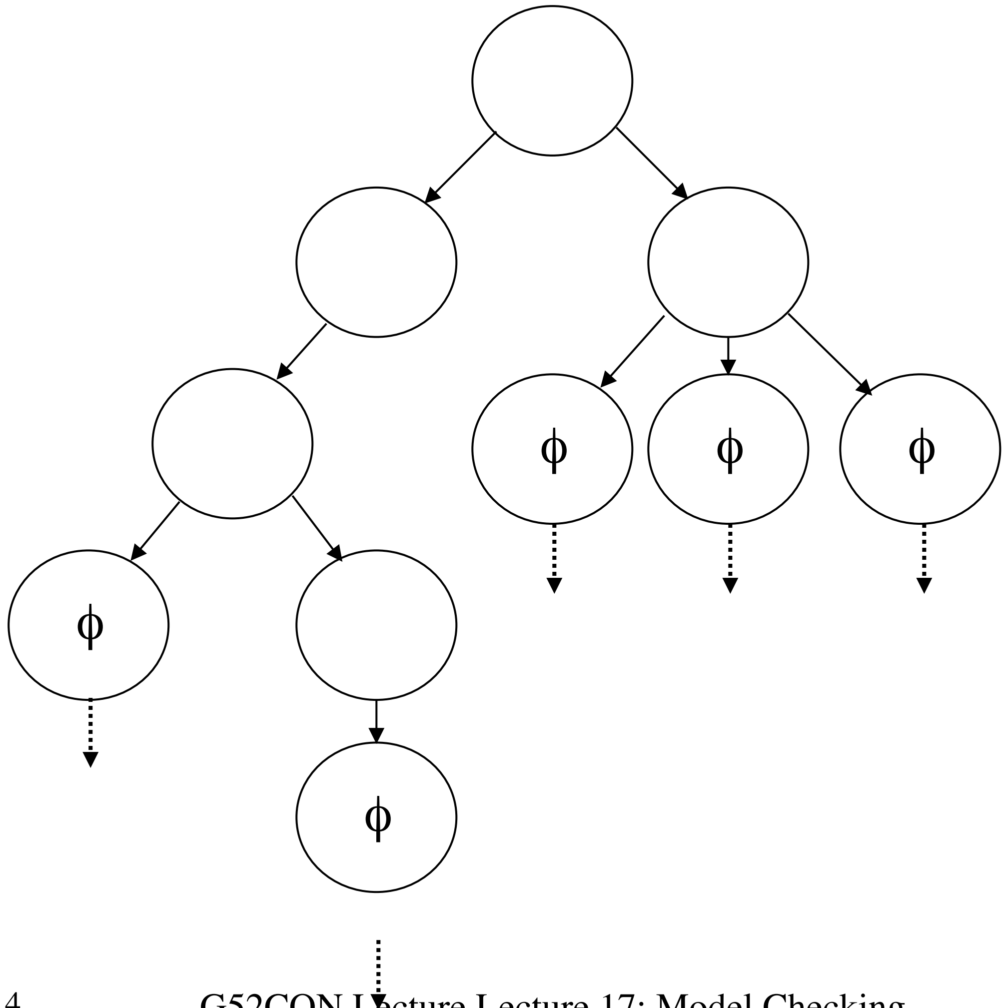
Example: a system which satisfies EG ϕ



Example: a system which satisfies AG ϕ



Example: a system which satisfies AF ϕ



Models of CTL

A model M = (S, →, L) for CTL is given by:

- a set of states S
- a transition relation → on S, such that for every $s \in S$ there exists an s' $\in S$ such that $s \rightarrow s'$
- if there are no transitions possible from *s*, e.g., *s* is a termination state or a deadlock state, we add transition from *s* to a special state with a transition to itself, representing termination or deadlock.
- a labelling function L(*s*) specifying the set of atomic propositions which are true at *s*.

Definition of truth for CTL formulas

Let M = (S, →, L) be a model of CTL. For any state $s \in$ S, a CTL formula ϕ holds at s iff:

$$\mathrm{M}, s \models \phi$$

1. M, $s \models p$ iff $p \in \mathrm{L}(s)$

2. M, $s \models \neg\phi$ iff M, $s \not\models \phi$

3. M, $s \models \phi_1 \wedge \phi_2$ iff M, $s \models \phi_1$ and M, $s \models \phi_2$

4. M, $s \models \phi_1 \vee \phi_2$ iff M, $s \models \phi_1$ or M, $s \models \phi_2$

5. M, $s \models \phi_1 \rightarrow \phi_2$ iff M, $s \not\models \phi_1$ or M, $s \models \phi_2$

Definition of truth for CTL formulas 2

6. M, s |= AX ϕ iff for all s_1 such that $s \rightarrow s_1$, we have M, s_1 |= ϕ

7. M, s |= EX ϕ iff for some s_1 such that $s \rightarrow s_1$, we have M, s_1 |= ϕ

8. M, s |= AF ϕ iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s, there is some s_i such that M, s_i |= ϕ

9. M, s |= EF ϕ iff there exists a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s and there is some s_i such that M, s_i |= ϕ

10. M, s |= AG ϕ iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s, all s_i along the path we have M, s_i |= ϕ

11. M, s |= EG ϕ iff there exists a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s and all s_i along the path we have M, s_i |= ϕ

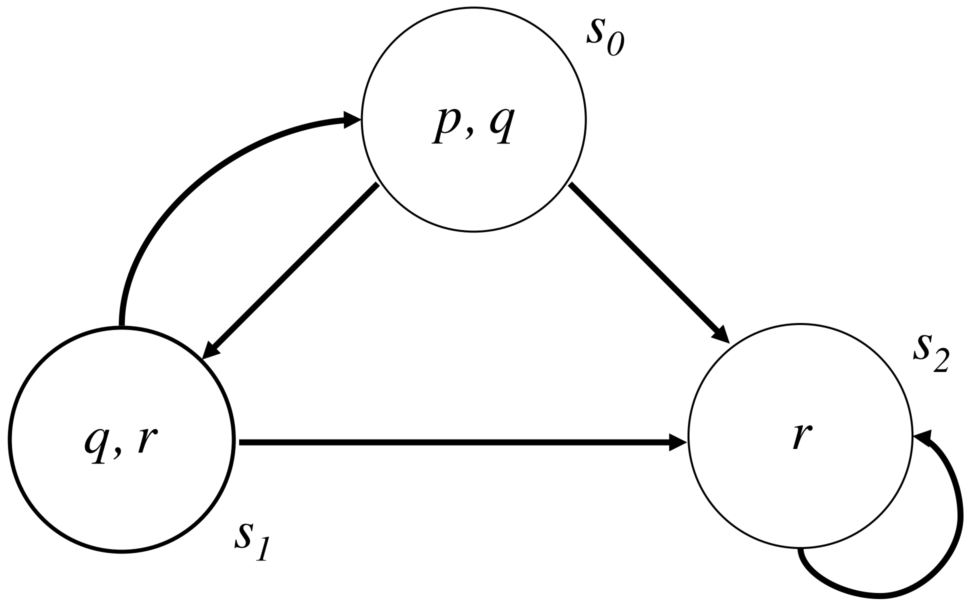
Definition of truth for CTL formulas 3

12. M, s |= A[ϕ_1 U ϕ_2] iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s and that path satisfies ϕ_1 U ϕ_2, i.e., there is some s_i along the path such that M, s_i |= ϕ_2 and for each $j < i$, we have M, s_j |= ϕ_1

13. M, s |= E[ϕ_1 U ϕ_2] iff there exists a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s and that path satisfies ϕ_1 U ϕ_2, i.e., there is some s_i along the path such that M, s_i |= ϕ_2 and for each $j < i$, we have M, s_j |= ϕ_1

Exercise: evaluating CTL formulas

Given the following transition system:



Questions

- is the CTL formula AF r true at s_0?

- is the CTL formula AG r true at s_0?

- is the CTL formula AG AF r true at s_0?

The next lecture

Model Checking II

Suggested reading:

- Huth & Ryan (2000), chapter 3.