TUESDAY, Session 2
Temporal logic and model checking, cont.

1 Branching time and CTL model checking

In a branching time temporal logics, we consider not just a single path through the Kripke model, but all possible paths emanating from a given state [1, 2].

- Path quantifiers

  A *path quantifier* indicates whether a given formula applies to all all possible paths from a given state or to some possible path:

  $$\begin{array}{lll} M, s_i \models A\phi & \text{iff} & \text{for } \textit{all} \text{ paths } \sigma = s_1, s_2, \ldots \text{: } \sigma \models \phi \\ M, s_i \models E\phi & \text{iff} & \text{for } \textit{some} \text{ paths } \sigma = s_1, s_2, \ldots \text{: } \sigma \models \phi \end{array}$$

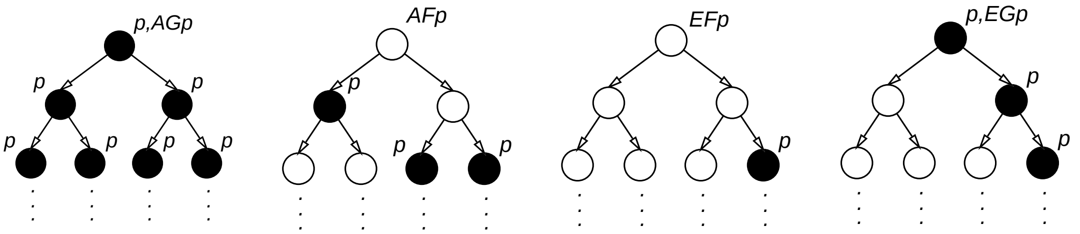
  Note that $A\phi \equiv \neg E \neg \phi$.

- The temporal logic CTL

  In the temporal logc CTL, every temporal operator F, G, X, or U is immediately preceded by a path quantifier.

  Some CTL modalities and their interpretations:

$AG\ p$	$AF\ p$	$EF\ p$	$EG\ p$
"globally p"	"inevitably p"	"possibly p"	"?"



  Note the following dualities:

  $$AGp \equiv \neg EF \neg p$$

  $$AFp \equiv \neg EG \neg p$$

Other CTL operators:

$$AXp, \quad EXp, \quad A(p\ U\ q), \quad E(p\ U\ q)$$

- Example: some specifications for the mutual exclusion protocol

$AG\neg(C_1 \wedge C_2)$	mutual exclusion
$AG(T_1 \Rightarrow AF\ C_1)$	liveness
$AG(N_1 \Rightarrow EX\ T_1)$	non-blocking

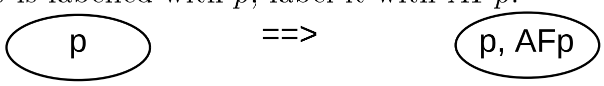
Note the last can't be stated in PLTL.

1.1 CTL model checking

- Suppose we have already labeled the set of states satisfying the proposition p.

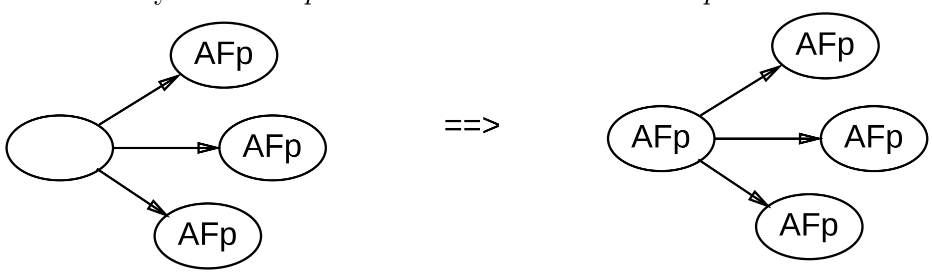
  To label the set of states satisfying AFp:

  1. If any state s is labelled with p, label it with AFp.



  2. Repeat

     label any state AFp if all successors labeled AFp



     until no change
  3. Label all states with $\neg AFp$ if not labeled AFp.

- Now the truth value of AFp in every state is known. So AFp can be treated as an atomic proposition while checking, for example $AG\ AFp$. That is, model checking procedes from smaller subformulas to larger subformulas.

- Algorithms for the other operators

$$AGp, \quad EFp, \quad EGp, \quad AXp, \quad EXp, \quad A(p\ U\ q), \quad E(p\ U\ q)$$

are similar.

- Complexity is $O(fV(V+E))$ where
  - f is the number of operators in the formula.
  - V is the number of states.
  - E is the number of transitions.

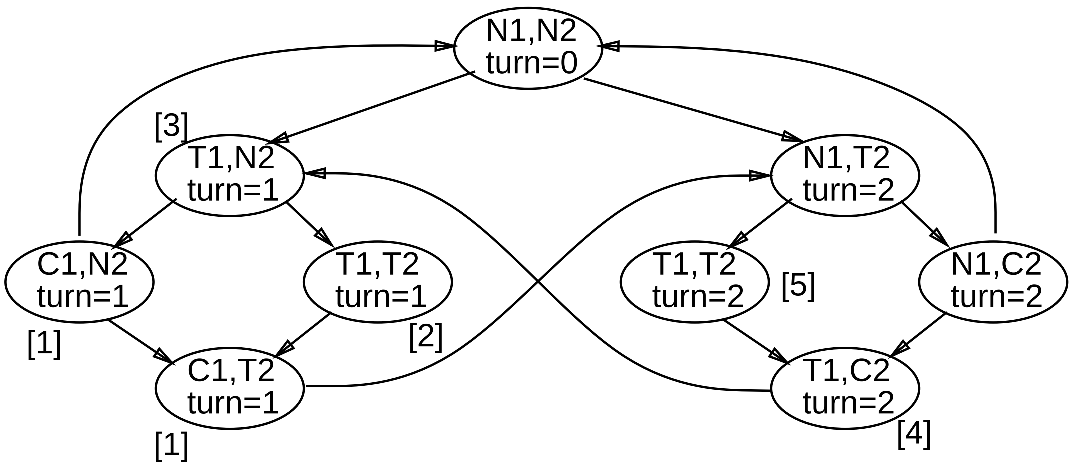
  ...since each operator terminates after at most V passes over the state graph.

- Example: checking $AG(T_1 \Rightarrow AF\ C_1)$ for the mutual exclusion protocol

  "always, if p[1] trying then inevitably p[1] critical"

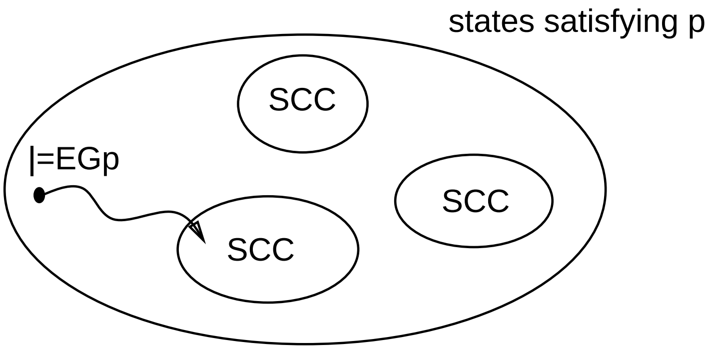
  1. label graph with $AF\ C_1$
  2. label every state $T_1 \Rightarrow AF\ C_1$ if T_1 is false or C_1 is true
  3. O.K., if all states labeled $T_1 \Rightarrow AF\ C_1$

  Result of labeling state graph with $AF\ C_1$ (numbers in [] indicate on which pass the state was labeled).



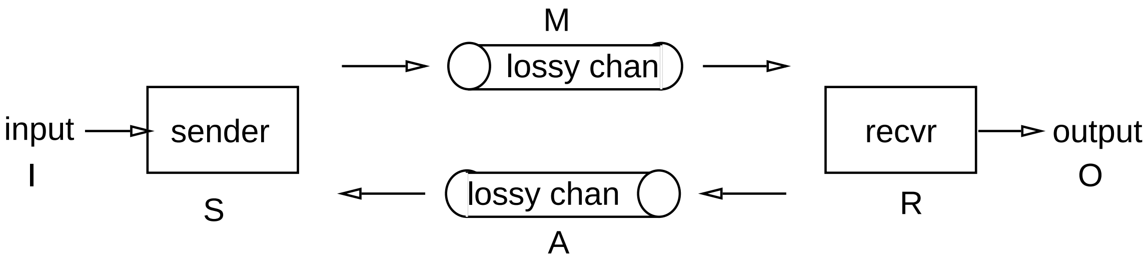
  In every state, if T_1 is true, then $AG\ C_1$ is true, hence $AG\ AF\ C_1$ is true in the initial state.

- A more efficient algorithm (Clarke/Emerson/Sistla)
  - First note, all formulas can be expressed using only EX, EU, EG. *e.g.* $AGp \equiv \neg EF \neg p$
  - $E(p\ U\ p)$ case: backward breadth-first search
  - $EG\ p$ case:
    * restrict graph to states satisfying p
    * find maximal strongly connected components
    * use BFS to find any state that can reach an SCC



This algorithm is $O(f(V+E))$ (*i.e.* linear in both formula size and model size).

1.2 Example: the ABP revisited



We construct a *very* abstract model, ignoring message data and considering only sequence numbers.

- The sender process

S ::	inp_ctr, ack_ctr : 0..1, initially 0;	
	[	inp_ctr = ack_ctr ⇒ I?data(); inp_ctr := inp_ctr + 1 mod 2
	□	inp_ctr ≠ ack_ctr ⇒ M!msg(inp_ctr)
	□	A?ack(ack_ctr)
	]*	

- the message channel (note, ack channel is similar)

M ::	ctr : 0..1, initially 0;	
	[	S?msg(ctr);
		[R!msg(ctr) □ skip]
	]*	

- the receiver process

```
R ::        rcv_ctr, out_ctr : 0..1, initially 0;
            [     M?msg(rcv_ctr);
                  [ out_ctr ≠ rcv_ctr ⇒ O!data(); out_ctr := rcv_ctr];
                  A!ack(rcv_ctr)
            ]*
```

Verifying the model

- Generate Kripke model from program text
- Express specifications in CTL

  Note: in the following, atomic propsitions like $(P_{\mathrm{msg}}Q)$ will be used to denote "P sends msg to Q". These are properly transition labels and not state labels. However, this problem is usually solved by using the "transition graph", where every transition becomes a state.

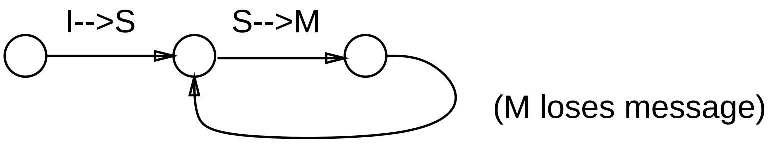
  – No duplicaiton of messages (and no buffering)

  $$\begin{aligned} \text{in\_before\_out} &\equiv (\neg(R_{\mathrm{data}}O)\ \mathcal{W}\ (I_{\mathrm{data}}S)) \\ \text{safe} &\equiv \text{in\_before\_out} \wedge AG((R_{\mathrm{data}}O) \Rightarrow AX\text{in\_before\_out}) \end{aligned}$$

  – liveness – every time a message is input one is eventually output

  $$\text{live} \equiv AG((I_{\mathrm{data}}S) \Rightarrow AF(R_{\mathrm{data}}O))$$

  When checking live, the model checker produces a counterexample like the following:

  That is, an infinite loop in the state graph, where every message is lost by the M channel.

Fairness assupmtions

- We want to verify the model *assuming* the channels do not lose messages forever. In PLTL, we could express this assumption as follows:

  $$\text{M\_fair} \equiv (GF\ (S_{\mathrm{msg}}M)\ \Rightarrow\ GF\ (M_{\mathrm{msg}}R))$$

We could then verify that

$$\text{M\_fair} \wedge \text{A\_fair} \Rightarrow \text{live}$$

As we will see, however, model checking for PLTL has exponential complexity in the formula size. Using many fairness constraints in this way would therefore be impractical.

- Suppose we try translating

  $$\text{M\_fair} \wedge \text{A\_fair} \Rightarrow \text{live}$$

  into CTL. In general if there is a CTL equivalent of an LTL formula it is obtained by adding A path quantifier to every operator.

  For example, M_fair becomes

  $$\text{M\_fair}' \equiv (AGAF\ S_{msg}M) \ \Rightarrow \ (AGAF\ M_{msg}R)$$

  This, however, is simply false in every state. Therefore

  $$\text{M\_fair} \wedge \text{A\_fair} \Rightarrow \text{live}$$

  is trivially true.

  In general, we can't express fairness constraints directly in CTL.

CTL with fairness constraints

- A *simple fairness constraint* is a formula of the form GFp, where p is a state formula.
- In a model with fairness constraints, path quantifiers apply only to paths satisfying all fairness constraints:

  $$\begin{array}{lll} M, s_i \models A_f\phi & \text{iff} & \text{for } \textit{all fair} \text{ paths } \sigma = s_1, s_2, \ldots\text{: } \sigma \models \phi \\ M, s_i \models E_f\phi & \text{iff} & \text{for } \textit{some fair} \text{ paths } \sigma = s_1, s_2, \ldots\text{: } \sigma \models \phi \end{array}$$

  where we us A_f and E_f to indicate the fair interpretation. For example, under the fairness constraint AGp,

  $$A_fF\ q \equiv A(GFp \Rightarrow Fq)$$

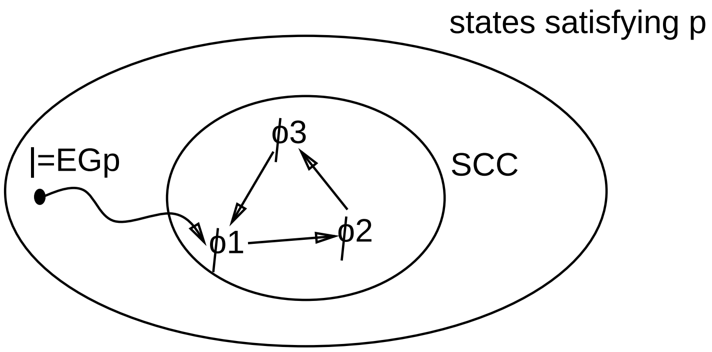
- Model checking under fairness constraints $\wedge_{i=1}^{n}\phi_n$
  - A state is fair (is the start of some fair path) iff it satisfies

    $$E_fG \text{ true}$$

  - $E_f(p\ U\ q) \equiv E(p\ U\ (q \wedge E_fG \text{ true}))$

– Algorithm for $E_f G\ p$:

* restrict the state graph to states satisfying p
* find the SCC's
* remove an SCC if it does not contain a state satisfying each ϕ_i.
* use BFS to find any state that can reach a (fair) SCC



Complexity of this algorithm: $O(f(V+E)n)$ ($i.e$, still linear).

- Fairness constraints for ABP

  – A simple fairness constraint:

$$GF((M_{\text{msg}}R))$$

  is sufficient to make the "live" specification true, but this is too strong an assumption ($i.e.$, what if the sender stops sending?).

  – A Streett fairness constraint

$$GF(S_{\text{msg}}M) \quad \Rightarrow \quad GF(M_{\text{msg}}R)$$

  is a weaker assumption (but perhaps still not justified, in case the reciever infinitely blocks reception of messages).

  CTL formulas under Streett fairness constraints can be verified in time

$$O(f(V+E)n^2)$$

  A yet weaker set of assumptions might be

$$GFEX(M_{\text{msg}}R) \quad \Rightarrow \quad GF(M_{\text{msg}}R)$$

$$GFEX(R_{\text{data}}O) \quad \Rightarrow \quad GF(R_{\text{data}}O)$$

  (the latter is to eliminate the case where the recvr receives a msg and then forever blocks further receptions while the M channel infinitely loses messages).

– A receptiveness property

$$\text{receptive} \equiv AG\ AF\ EX((I_{\text{data}}S))$$

"sender must eventually be ready to accept another message"
This requires a fairness constraint on the A as well as the M channel.

2 Expressiveness Issues

2.1 Linear *vs.* branching time

- The logic CTL* subsumes PLTL and CTL.
  - *path* formulas:
    $$p\ U\ q,\ Gp,\ Fp,\ Xp,\ \neg p,\ p \vee q$$
  - *state* formulas
    $$A\phi,\ E\phi \qquad \text{where } \phi \text{ is a path formula}$$

  An LTL formula like GFp is equivalent to the CTL* state formula $AGFp$.

- Some expressiveness results
  - "Existential" properties like $AG\ EF\ p$ not expressible in LTL,
    These are very useful for finding deadlocks in protocols.
  - "fairness" properties, like $A(GFp \Rightarrow GFp)$ not expressible in CTL.

- Complexity of model checking

CTL (with fairness)	$O(f(V+E)n^2)$
PLTL (with fairness)	$O(2^f(V+E)n^2)$ (PSPACE complete)
CTL* (with fairness)	same as PLTL

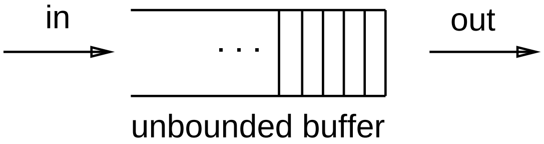
Note: LTL formulas are often small (when fairness constraints are built into the model). This means it is often practical to check them in spite of the exponential complexity.

Note: CTL* has same complexity as PLTL because we can treat state formulas as atomic propositions when checking path formulas. Because of this, it is often said that branching time is superior to linear time for model checking, since the complexity is the same or better, and it is strictly more expressive.

2.2 Data independence

- To check that ABP delivers correct data, we can add a one-bit data field to the messages and check
$$AG((I_{\text{data}(1)}S) \Rightarrow AF(R_{\text{data}(1)}O))$$
Question: Can we infer from this that protocol works for any data size?

- Suppose we want to allow arbitrary buffering of data?



e.g. allow behavior like: in(0);in(1);in(2);out(1);out(2);out(3)...

This is not expressible in propositional temporal logic.

- Data independence (Wolper)

A model is "data independent" [3] if all "data" variables occur only in assignments of the form:

x := y

or as message parameters, *e.g.* P!data(x) or Q?data(y).

- The bounded buffer property can be broken into two parts:

1. no duplication or loss of messages
2. messages delivered in order received

- Property (1) can be verified on a data-independent model with only two data values (say, 0 and 1):

$$\begin{aligned} \text{exactly\_once}(x) &\equiv (\neg x\ U\ (x \wedge XG\neg x)) \\ (1) &\equiv \text{exactly\_once}(\text{in}(1)) \Rightarrow \text{exactly\_once}(\text{out}(1)) \end{aligned}$$

The reasoning behind this is as follows: suppose a message is duplicated, *e.g.*

in(1);in(2);in(3);out(1);out(2);out(2);...

Every out() value must derive from some in() value by some sequence of assignments. So, by changing the duplicated input to 1, and all the others to 1, we a run like:

in(0);in(1);in(0);out(0);out(1);out(1);...

which violates our property (1).

- Property (2) can be verified with three data values (say, 0, 1 and 2) as follows:

$$\begin{aligned} \text{before}(x, y) &\equiv \neg y \; \mathcal{W} \; (x \wedge \neg y) \\ (2) &\equiv \text{exactly\_once}(\text{in}(1)) \wedge \text{exactly\_once}(\text{in}(2)) \\ &\qquad \wedge \text{before}(in(1), in(2)) \Rightarrow \text{before}(out(1), out(2)) \end{aligned}$$

  The reasoning is similar to the above.

3 Summary

- Reactive systems
  - Concurrency $\rightarrow$ temporal properties
  - LTL adds temporal operators to propositonal logic
    model is an infinite sequence of program states
  - Can express safety, liveness, fairness
  - Proofs are somewhat laborious
- Model checking
  - Translate model (*e.g.* in CSP) to finite state graph (Kripke model)
    * interleaving semantics for concurrency
    * model must be fairly abstract
  - Model checking algorithm for CTL
    * Naive fixed point algorithm $O(n^2)$
    * SCC based algorithm linear in formula size and model size
  - Fairness constraints
    * Simple fairness (GFp)
    * Streett fairness ($GFp \Rightarrow GFq$)
- Expressiveness issues
  - CTL* subsumes LTL and CTL
  - Tradeoff of expressivenes *vs.* complexity
  - Unbounded buffer properties
    * Cannot express directly in TL
    * Can verify using data independence arguments

References

[1] E. M. Clarke and O. Grumberg. Research on automatic verification of finite state systems. *Ann. Rev. Comput. Sci.*, 2:269–90, 1987.

[2] E. A. Emerson. Temporal and modal logic. In *Handbook of Theoretical Comp. Sci., vol. B: Formal Methods and Semantics*, chapter 16. Elsevier, 1990.

[3] P. Wolper. Expressing interesting properties of programs in propositional temporal logic. In *13th ACM POPL*, pages 184–193.