System Validation

Lecture 5: Computation Tree Logic

Joost-Pieter Katoen

Formal Methods and Tools Group

E-mail: `katoen@cs.utwente.nl`

URL: `fmt.cs.utwente.nl/courses/systemvalidation/`

January 21, 2004

Overview of lecture

⇒ *Introduction*

- Computation tree logic
  - Syntax and semantics
  - Some formulas express the same
- Model-checking CTL
- Fairness
- The difference between PLTL and CTL
- Practical use of CTL

Linear and branching temporal logic

- *Linear* temporal logic:

  the initial

  "statements about (all) paths starting in ~~a~~ state"

  – $s \models G\,(x \leqslant 20)$ iff for all possible paths starting in s always $x \leqslant 20$

  LTL

- *Branching* temporal logic:

  "statements about all or some paths starting in a state"

  – $s \models \mathbf{AG}\,(x \leqslant 20)$ iff for **all** paths starting in s always $x \leqslant 20$

  – $s \models \mathbf{EG}\,(x \leqslant 20)$ iff for **some** path starting in s always $x \leqslant 20$

  CTL

Why branching temporal logic?

- Expressiveness of linear and most branching temporal logics is incomparable:
  - there are properties that can be expressed in linear, but not in most branching TL
  - there are properties that can be expressed in most branching, but not in linear TL

- The model-checking algorithms are different, and so are their time and space complexities

model checking was originally developed for a branching temporal logic [Emerson & Clarke 1981]

Branching temporal logics

There are various branching temporal logics:

- Hennessy-Milner logic
- Computation Tree Logic (CTL)
- Extended Computation Tree Logic (CTL*)
  - combines PLTL and CTL into a single framework
- Alternation-free modal μ-calculus
- Modal μ-calculus
- Propositional dynamic logic

Overview of lecture

- Introduction

⇒ *Computation tree logic*
  - *Syntax and semantics*
  - *Some formulas express the same*

- Model-checking CTL

- Fairness

- The difference between PLTL and CTL

- Practical use of CTL

Propositional linear temporal logic

Is the smallest set of formulas generated by the rules:

1. each atomic proposition p is a formula
2. if Φ and Ψ are (path) formulas, then $\neg\Phi$ and $\Phi \vee \Psi$ are formulas
3. if Φ and Ψ are formulas, then $\mathbf{X}\,\Phi$ ("next") and $\Phi\,\mathbf{U}\,\Psi$ ("until") are formulas.

derived operators $\mathbf{G}$ (always) and $\mathbf{F}$ (eventually)

how to specify that for every computation it is always possible to return to the initial state? $\mathbf{G}\,\mathbf{F}$ *start?*

?

Propositional **branching** temporal logic

Global idea.

- Extend PLTL with *path quantifiers*:
  - $\mathbf{A}$, where $\mathbf{A}\,\varphi$ denotes that φ holds over all paths
  - $\mathbf{E}$, where $\mathbf{E}\,\varphi$ denotes that there exists some path satisfying φ
- $\mathbf{A}\,\varphi$ and $\mathbf{E}\,\varphi$ are called *state*-formulas
- PLTL-formula φ is called a *path*-formula

how to specify that for every computation it is always possible to return to the initial state? $\mathbf{AG}\,\mathbf{EF}$ *start!*

Computation tree logic

CTL is the smallest set of formulas generated by the rules:

1. State-formulas:

   (a) each atomic proposition p is a state-formula
   (b) if Φ and Ψ are state-formulas, then $\neg\,\Phi$ and $\Phi \vee \Psi$ are state-formulas
   (c) if φ is a path-formula, then $\mathbf{E}\,\varphi$ and $\mathbf{A}\,\varphi$ are state-formulas

2. Path-formulas:

   (a) if Φ and Ψ are state-formulas, then $\mathbf{X}\,\Phi$ and $\Phi\,\mathbf{U}\,\Psi$ are path-formulas.

$\mathbf{X}$ and $\mathbf{U}$ are always directly preceded by $\mathbf{E}$ or $\mathbf{A}$

Derived operators

$$\mathbf{F}\,\Phi \equiv \text{true}\,\underset{\smile}{\mathbf{U}}\,\Phi$$ — path formula

$$\mathbf{G}\,\Phi \equiv \neg\boxed{\mathbf{F}\,\neg\,\Phi}$$ — global idea.

$$\mathbf{EF}\,\Phi \equiv \mathbf{E}\,(\text{true}\,\mathbf{U}\,\Phi)$$ "potentially Φ"

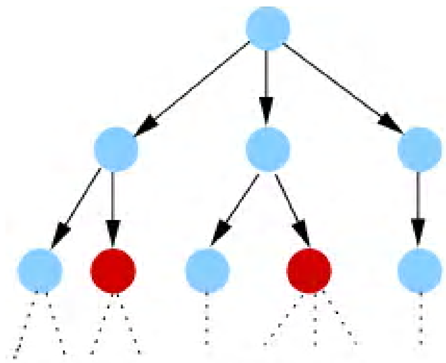
$$\mathbf{AG}\,\Phi \equiv \neg\,\mathbf{EF}\,\neg\,\Phi$$ "invariantly Φ"

$$\mathbf{AF}\,\Phi \equiv \mathbf{A}\,(\text{true}\,\mathbf{U}\,\Phi)$$ "inevitably Φ"

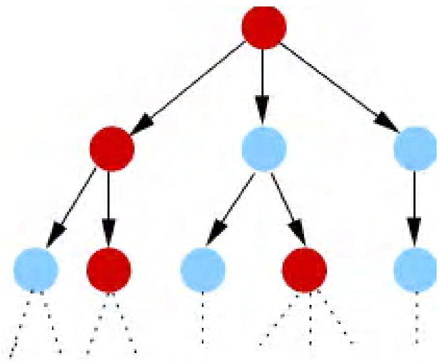
$$\mathbf{EG}\,\Phi \equiv \neg\,\mathbf{AF}\,\neg\,\Phi$$ "potentially always Φ"

the boolean connectives are derived as usual

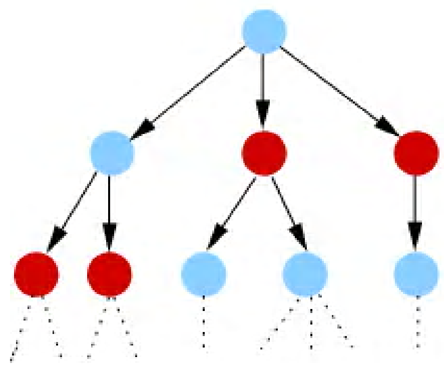
Derived operators



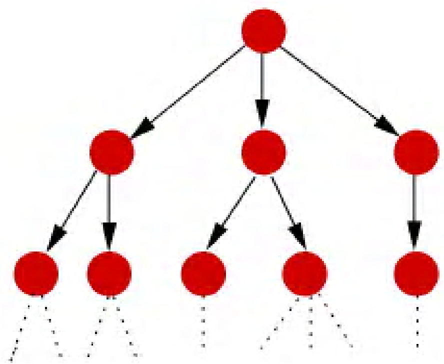
EF *red*



EG *red*



AF *red*



AG *red*

Some example CTL-formulas

let AP be the set of atomic propositions over variable x, boolean operators $<$, $\geqslant$ and $=$, and function $x + c$ for constant c

- the following formulas are *legal* CTL-formulas over AP:
  - $\neg\,(x + 7 < 21) \;\vee\; (x = 64)$
  - $\mathbf{AF}\,(x + 12 \geqslant 10)$
  - $\mathbf{EG}\,(x \geqslant 0 \;\wedge\; x < 200)$
  - $x = 10 \Rightarrow \mathbf{AX}\,\mathbf{E}\,(x \geqslant 10\,\mathbf{U}\,x = 0)$

- the following formulas are *illegal* CTL-formulas over AP:
  - $\neg\,(x + x < 21) \;\vee\; (x^3 = 64)$
  - $\mathbf{E}\,(\mathbf{F}\,(x \geqslant 10) \wedge \mathbf{G}\,(x \geqslant 0))$
  - $\mathbf{E}\,(x \geqslant 20) \wedge \mathbf{X}\,x = 20)$

Interpretation of CTL

Formal interpretation of CTL-formulas is defined in terms of a Kripke structure $\mathcal{M} = (S, I, R, Label)$ where

- S is a countable set of states,
- $I \subseteq S$ is a set of initial states,
- $R \subseteq S \times S$ is a transition relation with $\forall s \in S. (\exists s' \in S. (s, s') \in R)$
- $Label : S \longrightarrow 2^{AP}$ is an interpretation function on S.

$Label(s)$ is the set of the atomic propositions that are valid in s

Semantics of CTL: state-formulas

Defined by a relation $\models$ such that

$\boxed{\mathcal{M}, s \models \Phi \text{ if and only if formula } \Phi \text{ holds in state } s \text{ of structure } \mathcal{M}}$

$s \models p$	iff	$p \in Label(s)$
$s \models \neg\, \Phi$	iff	$\neg\, (s \models \Phi)$
$s \models \Phi \vee \Psi$	iff	$(s \models \Phi) \vee (s \models \Psi)$
$s \models \mathbf{E}\, \varphi$	iff	$\sigma \models \varphi$ for *some* path σ that starts in s
$s \models \mathbf{A}\, \varphi$	iff	$\sigma \models \varphi$ for *all* paths σ that start in s

Semantics of CTL: path-formulas

A *path* in $\mathcal{M}$ is an infinite sequence of states $s_0\, s_1\, s_2 \ldots$ such that $(s_i, s_{i+1}) \in R$ for all $i \geqslant 0$

Define a relation $\models$ such that

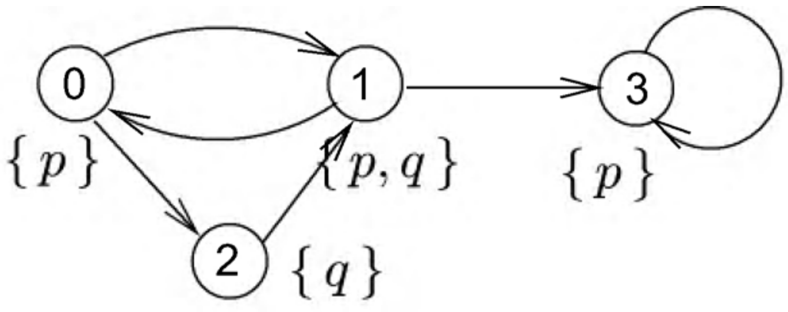
$\boxed{\mathcal{M}, \sigma \models \varphi \text{ if and only if path } \sigma \text{ in model } \mathcal{M} \text{ satisfies formula } \varphi}$

$$\sigma \models \mathbf{X}\,\Phi \quad \text{iff } \sigma[1] \models \Phi$$

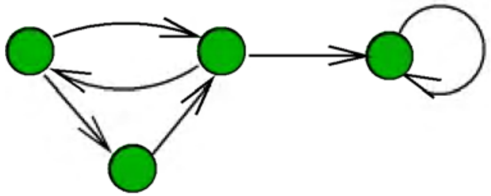
$$\sigma \models \Phi\,\mathbf{U}\,\Psi \quad \text{iff } (\exists\, j \geqslant 0.\, \sigma[j] \models \Psi \;\wedge\; (\forall\, 0 \leqslant k < j.\, \sigma[k] \models \Phi))$$

where $\sigma[i]$ denotes the $(i{+}1)$-th state in the path σ

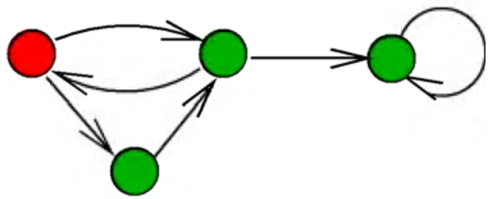
Example of semantics of CTL



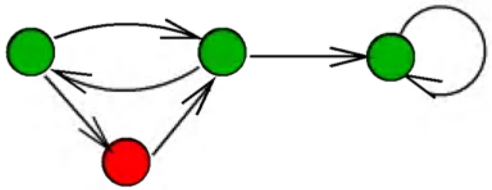
$\mathbf{EX}\, p$



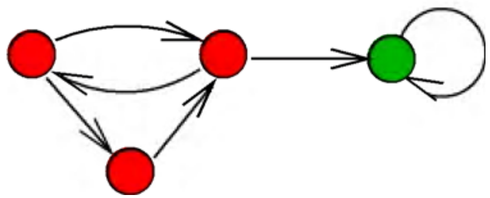
$\mathbf{AX}\, p$



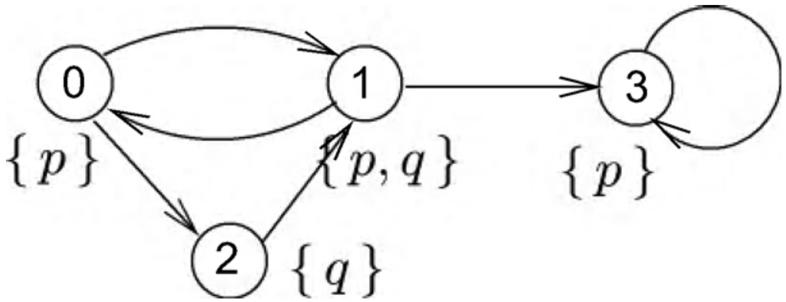
$\mathbf{EG}\, p$



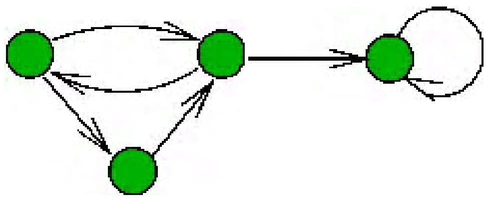
$\mathbf{AG}\, p$



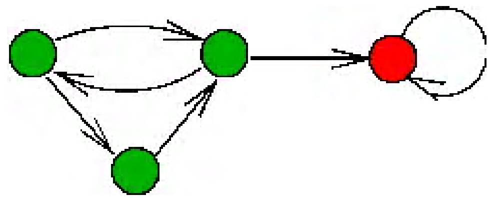
Example of semantics of CTL (cont'd)



$\mathbf{EF}\,\mathbf{EG}\,p$



$\mathbf{A}\,(p\,\mathbf{U}\,q)$



Some important validities for CTL

PLTL expansion rules:

(last lecture)

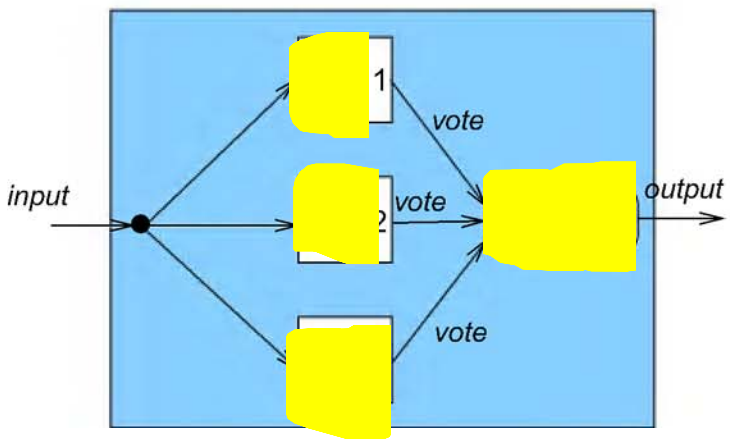
$$
\begin{aligned}
\Phi\,\mathbf{U}\,\Psi &\equiv \Psi \lor (\Phi \land \mathbf{X}(\Phi\,\mathbf{U}\,\Psi)) \\
\mathbf{F}\,\Phi &\equiv \Phi \lor \mathbf{X}\,\mathbf{F}\,\Phi \\
\mathbf{G}\,\Phi &\equiv \Phi \land \mathbf{X}\,\mathbf{G}\,\Phi
\end{aligned}
$$

CTL expansion rules:

$$
\begin{aligned}
(\Phi\,\mathbf{U}\,\Psi) &\equiv \Psi \lor (\Phi \land \mathbf{EX}\,\mathbf{E}(\Phi\,\mathbf{U}\,\Psi)) \\
(\Phi\,\mathbf{U}\,\Psi) &\equiv \Psi \lor (\Phi \land \mathbf{AX}\,\mathbf{A}(\Phi\,\mathbf{U}\,\Psi)) \\
\mathbf{EF}\,\Phi &\equiv \Phi \lor \mathbf{EX}\,\mathbf{EF}\,\Phi \\
\mathbf{AF}\,\Phi &\equiv \Phi \lor \mathbf{AX}\,\mathbf{AF}\,\Phi \\
\mathbf{EG}\,\Phi &\equiv \Phi \land \mathbf{EX}\,\mathbf{EG}\,\Phi \\
\mathbf{AG}\,\Phi &\equiv \Phi \land \mathbf{AX}\,\mathbf{AG}\,\Phi
\end{aligned}
$$

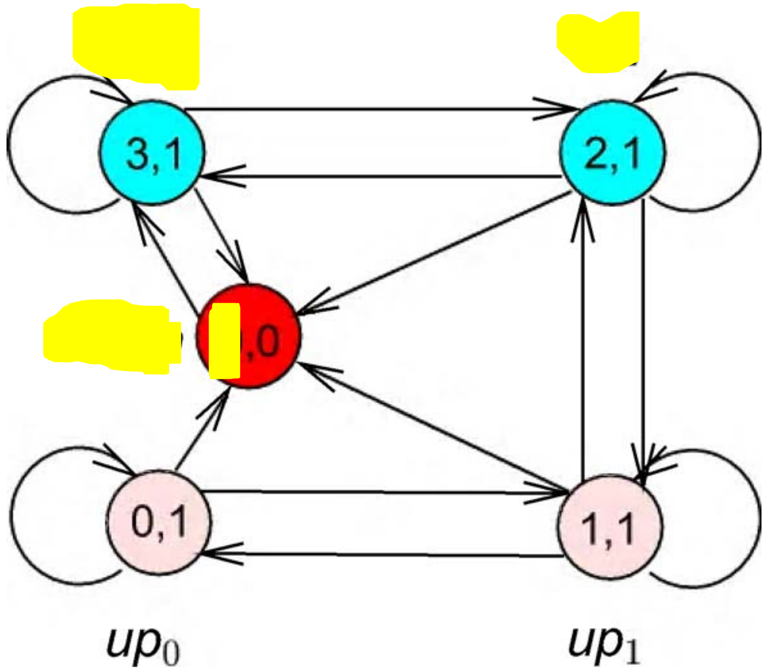
Specifying properties in CTL

- Triple Modular Redundant system: 3 processors and a single voter
  - processors run same program; voter takes a majority vote
  - each component (processor and voter) is failure-prone
  - there is a single repairman for repairing processors and voter



- Modelling assumptions:
  - if voter fails, whole system goes down
  - after repair of voter, system starts “as new”
  - state = (#processors, #voters)

Specifying properties in CTL



- Possibly, the system never goes down: $\mathbf{EG} \neg \mathit{down}$

- Inevitably, the system never goes down: $\mathbf{AG} \neg \mathit{down}$

- It is always possible to start as new: $\mathbf{AG}\,\mathbf{EF}\, \mathit{up}_3$ (not $\mathbf{AF}\, \mathit{up}_3$)

- The system only goes down while being operational:

$$\mathbf{A}\,((\mathit{up}_3 \vee \mathit{up}_2)\,\mathbf{U}\,\mathit{down})$$

Overview of lecture

- Introduction
- Computation tree logic
  - Syntax and semantics
  - Some formulas express the same

$\Rightarrow$ *Model-checking CTL*

- Fairness
- The difference between PLTL and CTL
- Practical use of CTL

Model checking CTL

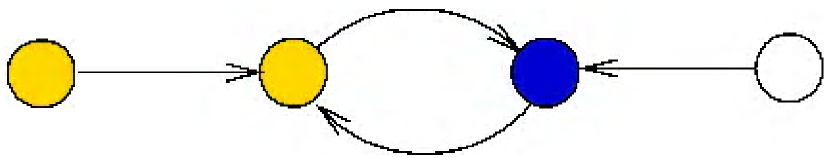
- how to check whether state s satisfies Φ?
  - compute *recursively* the set $Sat(\Phi)$ of states that satisfy Φ
  - check whether state s belongs to $Sat(\Phi)$

- recursive computation:
  - determine the sub-formulas of Φ
  - start to compute $Sat(p)$, for all atomic propositions p in Φ
  - then check the smallest sub-formulas that contain p
  - check the formulas that contain these sub-formulas
  - and so on....... until formula Φ is checked

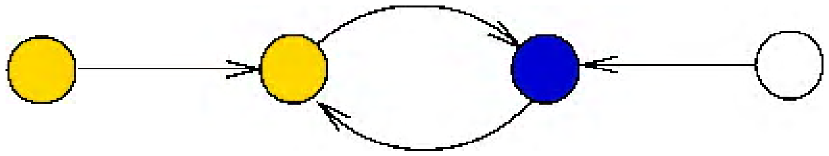
Model checking CTL: pseudo-algorithm

- $Sat(p)$ is the set of states labelled with atomic proposition p
- $Sat(\Phi \vee \Psi)$ is $Sat(\Phi) \cup Sat(\Psi)$
- $Sat(\neg\Phi)$ equals $S - Sat(\Phi)$
- $Sat(\mathbf{EX}\,\Phi)$ is the set of states that can directly move to $Sat(\Phi)$
- $Sat(\mathbf{AX}\,\Phi)$ is the set of states that can directly only move to $Sat(\Phi)$
- $Sat(\mathbf{E}\,(\Phi\,\mathbf{U}\,\Psi))$ is computed iteratively:
  - $S^0 = Sat(\Psi)$
  - $S^1 = S^0 \cup \Phi$-states that can directly move to S^0
  - $S^2 = S^1 \cup \Phi$-states that can directly move to S^1
  - . . . . until $S^{k+1} = S^k$

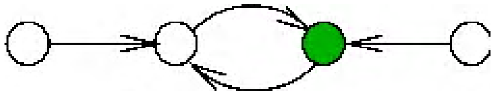
Computing $Sat(\mathbf{E}\,(yellow\;\mathbf{U}\;blue))$



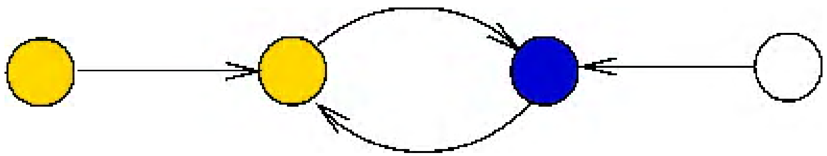
Computing $Sat(\mathbf{E}\,(yellow\,\mathbf{U}\,blue))$



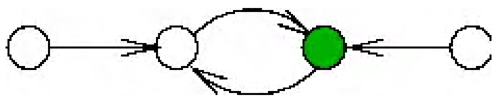
first iteration



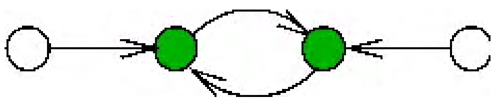
Computing $Sat(\mathbf{E}\,(yellow\,\mathbf{U}\,blue))$



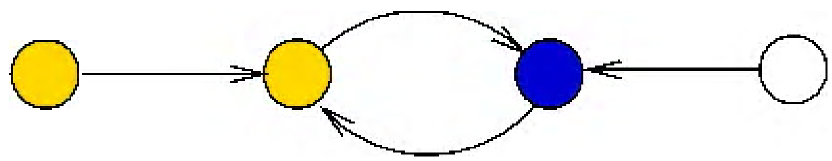
first iteration



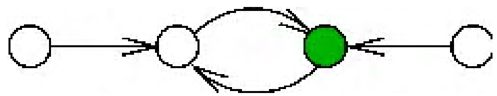
second iteration



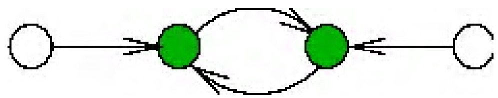
Computing $Sat(\mathbf{E}\,(yellow\;\mathbf{U}\;blue))$



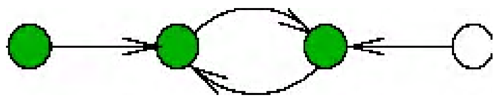
first iteration



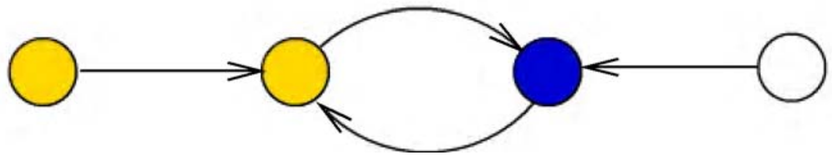
second iteration

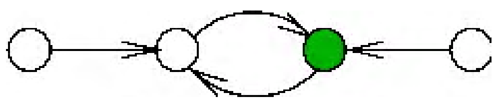


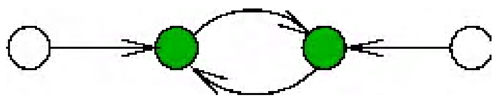
third iteration

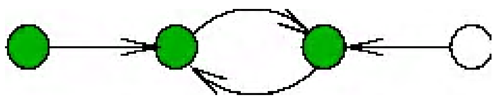


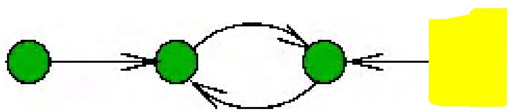
Computing $Sat(\mathbf{E}\,(yellow\;\mathbf{U}\;blue))$



first iteration 

second iteration 

third iteration 

fourth iteration  done!

Overview of model-checking CTL

- Algorithm: bottom-up traversal of the parse tree of the formula
- For until-formulas: a fixed-point computation
- For $\mathbf{EG}$-formulas: a more efficient algorithm using detection of strongly connected components
- Special attention has to be devoted to fairness issues
- Worst case time-complexity is $\mathcal{O}(|\Phi| \cdot N^2)$

  where $|\Phi|$ is the length of Φ and N is the number of states in the system model
- Tools: NuSMV, Cadence SMV, UPPAAL, CADP, ...

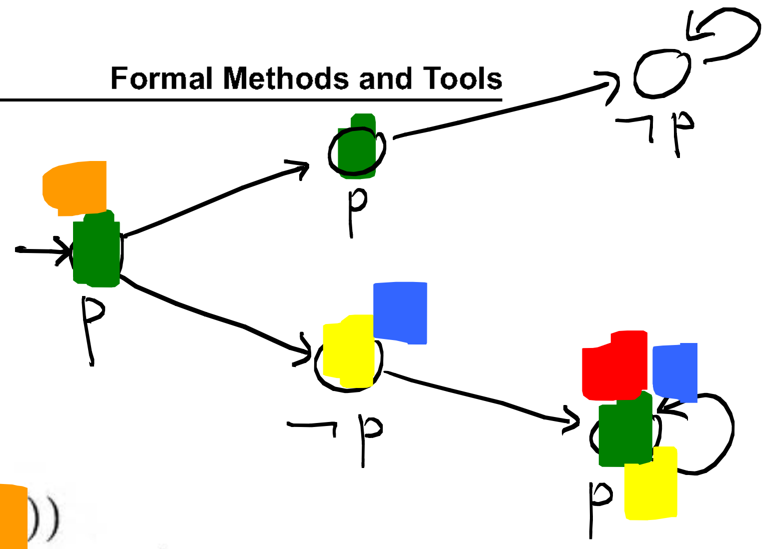
Overview of lecture

- Introduction
- Computation tree logic
  - Syntax and semantics
  - Some formulas express the same
- Model-checking CTL

$\Rightarrow$ *The difference between PLTL and CTL*

- Fairness
- Practical use of CTL

PLTL versus CTL



- Their expressiveness is incomparable:
  - there is no equivalent PLTL-formula for $\mathbf{AG}\,\mathbf{EF}\,p$
  - there is no equivalent CTL-formula for $\mathbf{A}$ ([illegible]))
    - each path reaches a point at which p holds for two consecutive moments
    - [illegible] $\wedge$ [illegible]) and $\mathbf{AF}\,(p \;\wedge$ [illegible] $p)$ do not express the same
  - but [illegible] common formulas like $\mathbf{A}\,(p\,\mathbf{U}\,q)$ and $\mathbf{AG}\,p$
- Complexity of model checking is different:
  - model checking PLTL is PSPACE-complete: $\mathcal{O}(\mathit{System}^2 \cdot 2^{\mathit{Formula}})$
  - model checking CTL is in polynomial time: $\mathcal{O}(\mathit{System}^2 \cdot \mathit{Formula})$

don't think that CTL model checking is more efficient
as CTL-formulas are sometimes much longer than PLTL-formulas!

Overview of lecture

- Introduction
- Computation tree logic
  - Syntax and semantics
  - Some formulas express the same
- Model-checking CTL
- The difference between PLTL and CTL

⇒ *Fairness*

- Practical use of CTL

Fairness: modelling concurrency

Consider the parallel execution of two processes: (initially $x = 0$)

process $P =$ **while** $\langle\ (x \geqslant 0)$ **do** $x := x + 1\ \rangle$ **od**

process $Q = x := -1$

- Does this parallel program ever terminate?
- Expected runs: $P\,Q\,P\,Q\,P\,Q \ldots$ or $P\,P\,Q\,P\,Q\,Q\,P\,P \ldots$ or the like
- But not: $P\,P\,P\,P\,P\ \ldots$ (no Q) or $Q\,Q\,Q\ \ldots$ (no P)
- *Fairness* is modeled by fair scheduling assumptions – described as temporal logic-formulas – over the processes

Typical fairness assumptions (in PLTL)

- *Unconditional fairness*: property *running* is true infinitely often:

$$\mathbf{G}\,\mathbf{F}\,\mathit{running}$$

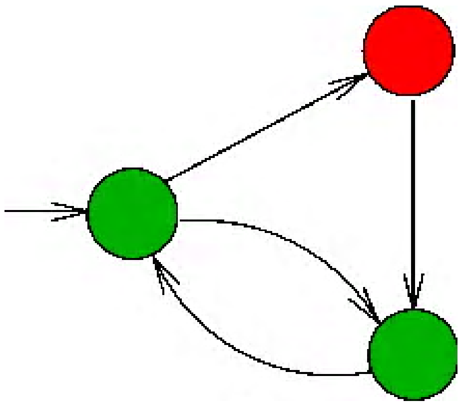
- *Weak fairness*: if *enabled* is eventually continuously true, *running* holds infinitely often:

$$\mathbf{F}\,\mathbf{G}\,\mathit{enabled} \;\Rightarrow\; \mathbf{G}\,\mathbf{F}\,\mathit{running}$$

- *Strong fairness*: if *enabled* holds infinitely often, *running* does so too:

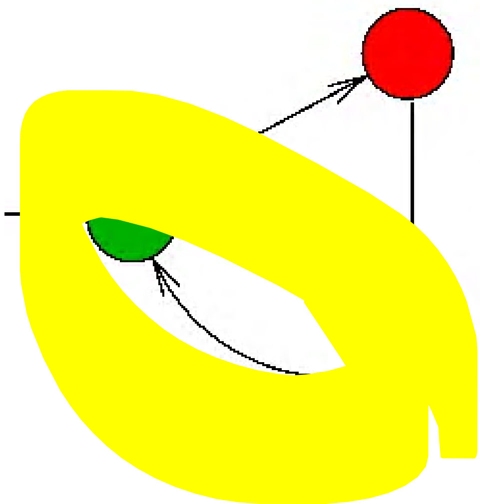
$$\mathbf{G}\,\mathbf{F}\,\mathit{enabled} \;\Rightarrow\; \mathbf{G}\,\mathbf{F}\,\mathit{running}$$

Fair versus unfair computations



do we have $\mathbf{AG}\,(green \;\Rightarrow\; \mathbf{AF}\,red)$?

Fair versus unfair computations



- no, since there exists an entirely *green* path!
- but, is this a “fair” path?
- no, as becoming *red* is possible infinitely often
- how to exclude these *unfair* computations?
- add a fairness assumption, e.g., $\mathbf{AG}\,\mathbf{AF}\,red$!
- then $\mathbf{AG}\,(green \Rightarrow \mathbf{AF}\,red)$ is valid as the unfair computations are ignored

$\Rightarrow$ fairness assumptions rule out “unrealistic” runs

Overview of lecture

- Introduction
- Computation tree logic
  - Syntax and semantics
  - Some formulas express the same
- Model-checking CTL
- The difference between PLTL and CTL
- Fairness

$\Rightarrow$ *Practical use of CTL*

Practical properties in CTL

- Reachability
  - simple reachability $\mathbf{EF}\,\Psi$
  - conditional reachability $\mathbf{E}\,(\Phi\,\mathbf{U}\,\Psi)$ also LTL
  - reachability from any state $\mathbf{AG}\,(\mathbf{EF}\,\Phi)$ — CTL only
- Safety ("something bad never happens")
  - simple safety $\mathbf{AG}\,\neg\Phi$
  - conditional safety $\mathbf{A}\,(\Phi\,\mathbf{U}\,\Psi) \;\vee\; \mathbf{AF}\,\Phi$
- Liveness $\mathbf{AG}\,(\Phi \;\Rightarrow\; \mathbf{AF}\,\Psi)$
- Fairness $\mathbf{AG}\,(\mathbf{AF}\,\Phi)$

Most commonly used specification patterns for CTL

Investigation of 555 requirement specifications reveals that the following patterns are most widely used for state-formulas P, Q and R: (Dwyer et al, 1998)

pattern	*scope*	*PLTL-formula*	*frequency*
response	global	$\mathbf{AG}\,(P \Rightarrow \mathbf{AF}\,Q)$	43.4 %
universality	global	$\mathbf{AG}\,P$	19.8 %
absence	global	$\mathbf{AG}\,\neg P$	7.4 %
precedence	global	$\mathbf{AG}\,\neg P \vee \mathbf{A}\,(\neg P\,\mathbf{U}\,Q)$	4.5 %
absence	between	[illegible]	3.2 %
absence	after	$\mathbf{AG}\,(Q \Rightarrow \mathbf{AG}\,\neg P)$	2.1 %
existence	global	$\mathbf{AF}\,P$	2.1 %
			$\approx$ 80 %

more info at: `www.cis.ksu.edu/santos/spec-patterns/`