Lecture 3
Property specification in Temporal Logic CTL*

J.Vain
18.02.2015

Model Checking

$$M \models P\ ?$$

Given

- *M* – model
- *P* – property to be checked

Check if *M* *satisfies* *P*

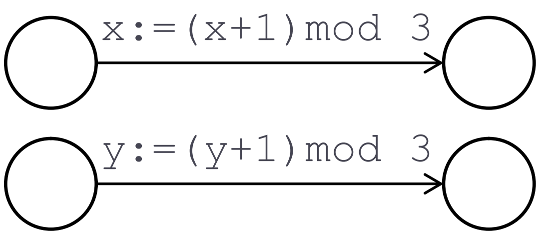
Model: Kripke Structure (revisited I)

- KS is a state-transition system that captures
  - what is true in a state
  - what can be viewed as an atomic move
  - the succession of states
- KS is a static representation that can be unrolled to a *tree of execution traces*, on which temporal properties are verified.

Representing transition

- In Kripke structure, $(s, s') \in R$ corresponds to one step of execution of the program.
- Suppose a program has two steps
  - `x := (x+1) mod 3;`
  - `y := (y+1) mod 3.`

  Then $R = \{R_1, R_2\}$
  - $R_1 : (x' = (x+1)\ mod\ 3) \wedge (y' = y)$
  - $R_2 : (y' = (y+1)\ mod\ 3) \wedge (x' = x)$



Consecutive States

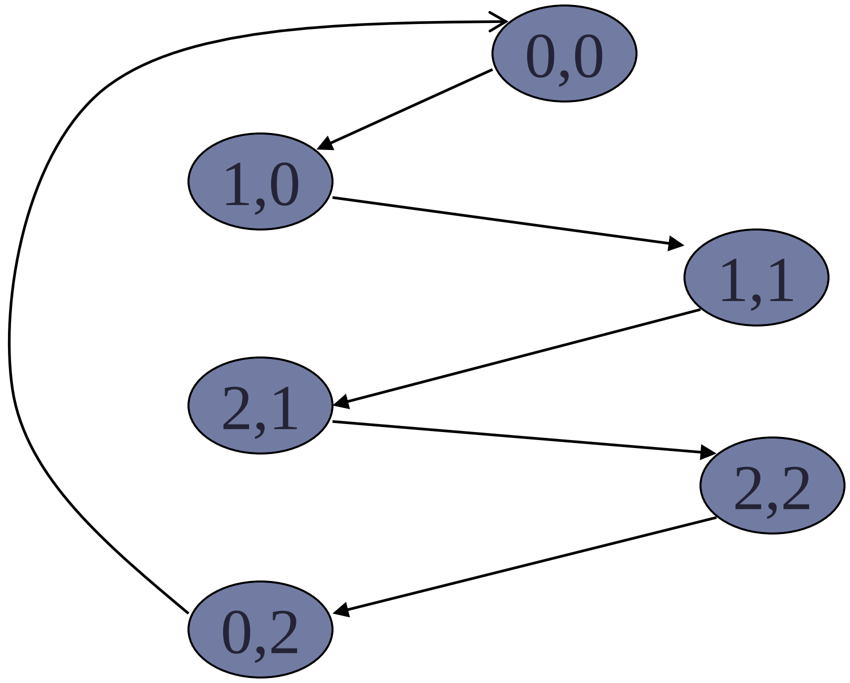
- State space: we can restrict our attention to

$$\{0, 1, 2\} \times \{0, 1, 2\}$$

- Question: which logic formula describes the relation between <u>any</u> two consecutive states?

- Consecutive states can be related by R_1 or R_2.

Consecutive states represented by $R_1 \vee R_2$



Representing transition (revisited II)

- In Kripke structure, a transition $(s, s') \in R$ corresponds to one step of execution of the program
- Suppose a program *P* has two steps
  - `x := (x+1) mod 3;`
  - `y := (y+1) mod 3;`
- For the whole program we have
  $R = ((x' = x+1 \; mod \; 3) \wedge y' = y) \vee ((y' = y+1 \; mod \; 3) \wedge x'=x)$
- (s, s') that satisfies *R* means "from *s* we can get to *s*' by any step of execution of *P*"

A giant *R*

- We can compute *R* for the whole program
  - then we will know whether two states are one-step reachable
- Convenient, but globally we loose information:
  e.g., the order in which the statements are executed
- Comment:
  - without order, the disjuncts have <u>no precedence</u>!

Introducing program counter

- In a real machine, the order of execution is managed by a *program counters*
- We introduce a virtual variable pc, and assume the program is everywhere labeled:
  - In the program: l_0`: x := x+1;` l_1`: y := x+1;` l_2`: …`

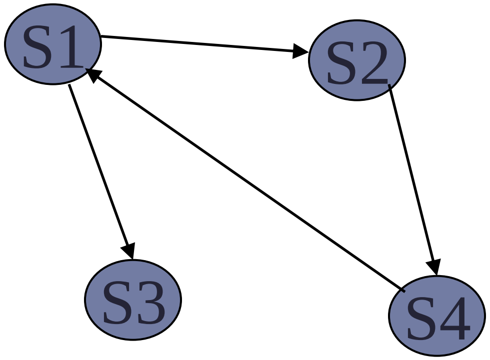
    $\Downarrow$
  - In the logic: $R_1 : x' = x+1 \wedge y' = y \wedge pc = l_0 \wedge pc' = l_1$

! Now we have complete logic representation of program executions in our model *M*!

Temporal logic CTL*

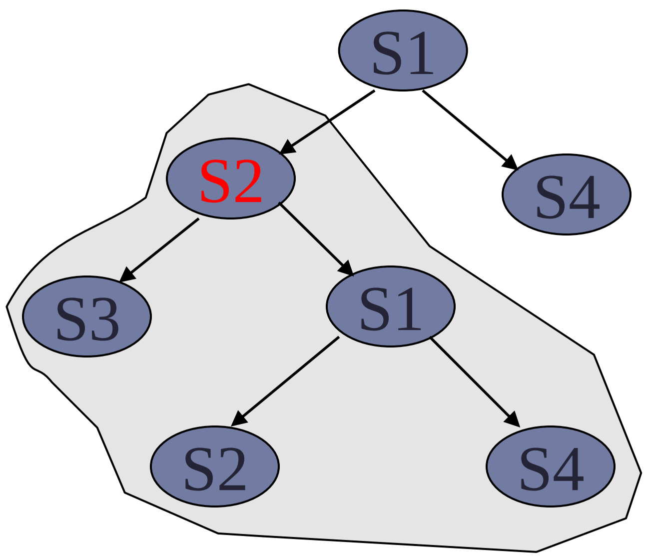
- Semantics

  KS is static model of program execution



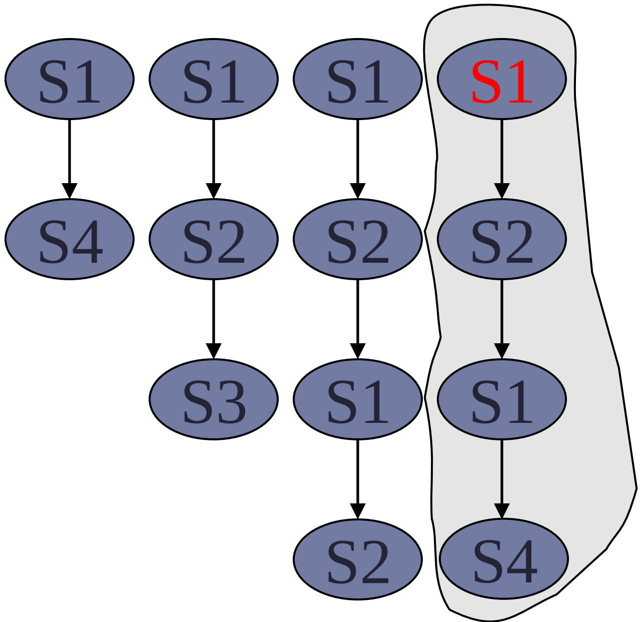
Dynamic model of program execution = unfolding of the static model

Tree structure: branching time



Is a formula valid at a given node, which represents a subtree?

Traces: linear time



Is a formula valid along a given path?

CTL* (Computational Tree Logic)

- Combines branching time and linear time
- Basic Operators
  - X: neXt
  - F: Future (〈〉)
  - G: Global ([])
  - U: Until
  - R: Release

CTL*

- State formulas
  - Express a propery of a state
  - Path quantifiers:
    - **A** – for all paths, **E** – for some paths
- Path formulas
  - Expess a property of a path
  - State quantifiers:
    - **G** – for all states (of the path)
    - **F** – for some state (of the path)

State Formulas (1)

- Atomic properties
  - $p \in$ AP, then p is a state formula
  - Examples: $x > 0$, $odd(y)$
- Propositional combinations of state formulas
  - $\neg\ \varphi,\quad \varphi \vee \psi,\quad \varphi \wedge \psi \ldots$
  - Examples: $x > 0 \vee odd(y)$, $req \Rightarrow$ (AF ack)
    - “A” is path quantifier
    - “F *ack*” is a path formula
    - “AF *ack*” is a state formula

State Formulas (2)

- Quantifiers A and E construct a state formula from a path formula
- Eφ , where φ is a path formula, which expresses property of a path
  - E means "there exists"
  - E φ - on some path from this state on φ is true.
- Dual: A φ
  - φ is true on all paths starting from this state.

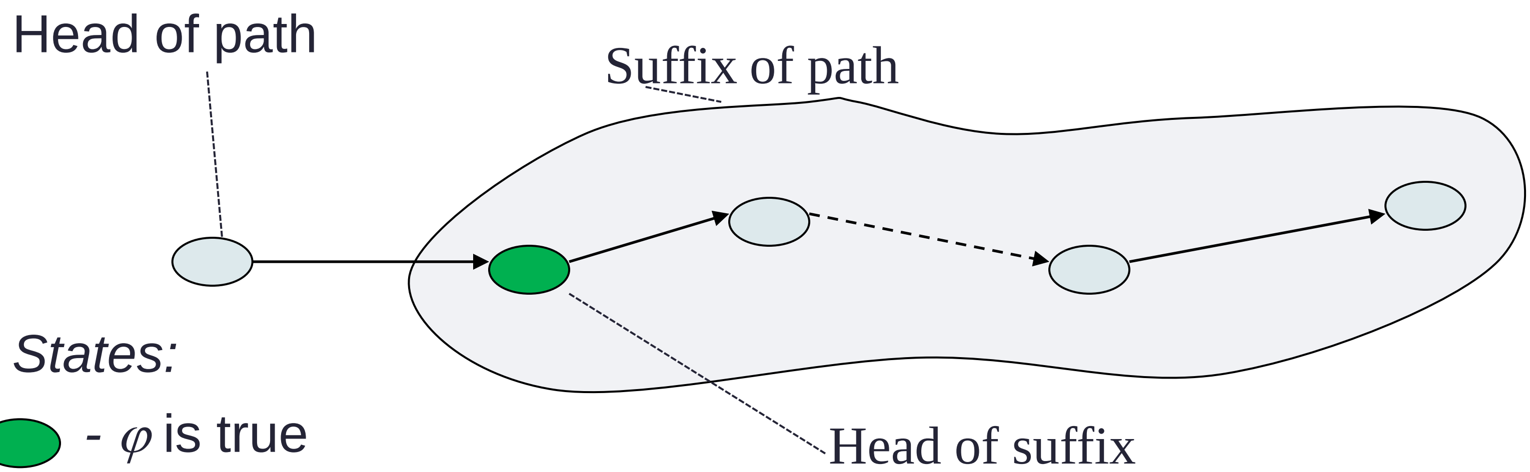
Forms of Path Formulas

- A state formula φ
  - φ is true for the first state of this path
- For path formulas φ and ψ, the path formulas are:
  - $\neg \varphi$, $\varphi \vee \psi$, $\varphi \wedge \psi$
  - X φ, Fφ, G φ, φ U ψ, φ R ψ

Path Formulas (I): *Next*-operator

X φ, where φ is a path formula

- φ is valid for the suffix of this path (path minus the first state)



States:

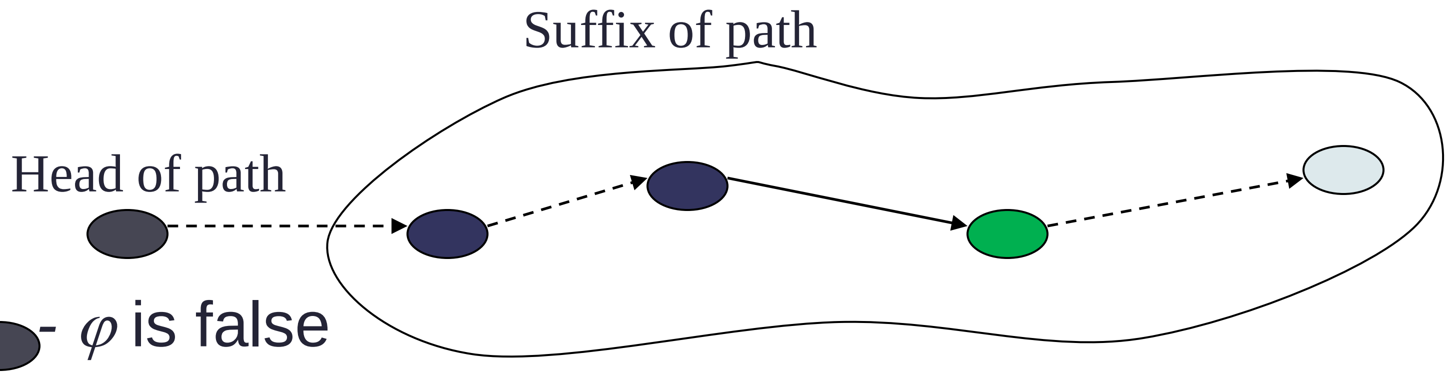
- φ is true

- φ can be either true or false

Path Formulas II: *Finally*-operator

F φ

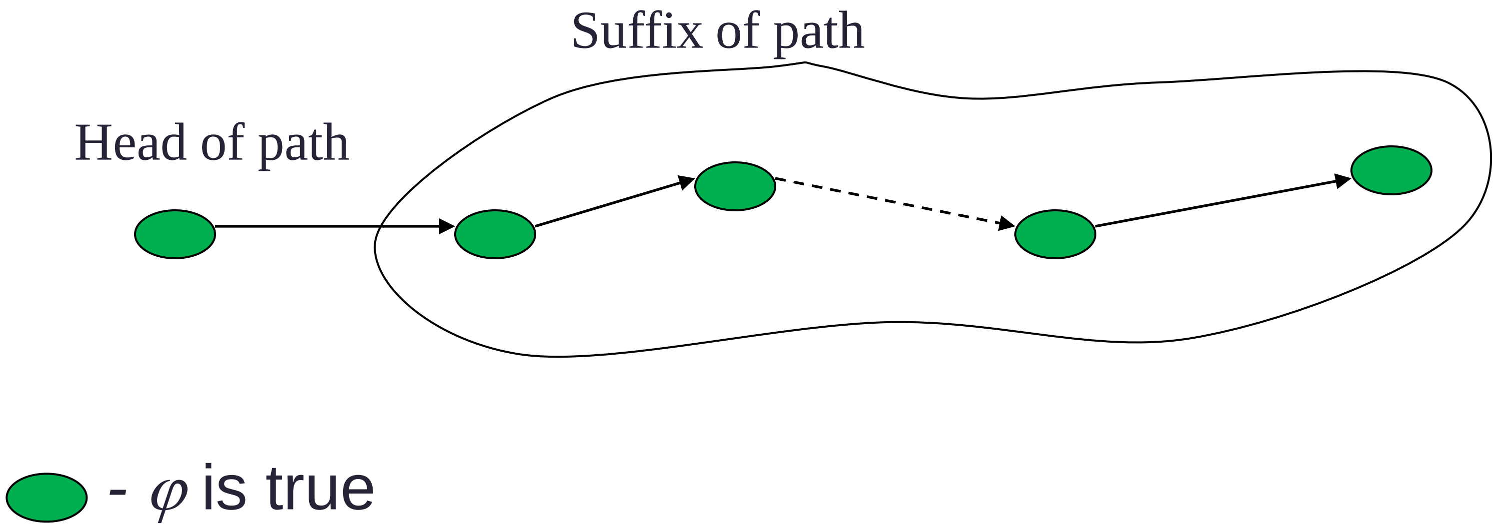
φ is valid for a suffix of this path (path minus first k nodes for some $k \geq 0$)



- φ is false
- φ is true
- φ can be either true or false

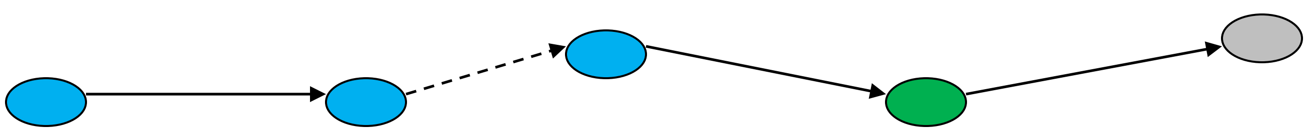
Path Formulas (III): *Globally*-operator

- G φ
  - φ is valid for head and every suffix of this path



Path Formulas IV: *Until*-operator

- φ U ψ
  - ψ is valid on a suffix of the path, before the first node of which φ is valid on every suffix thereon



- φ is true

- ψ is true

- φ and ψ are either true or false

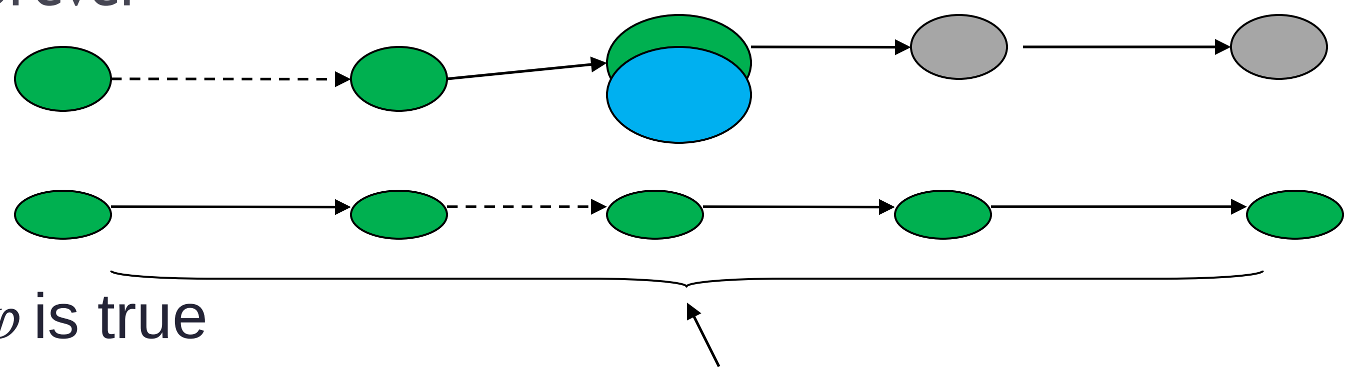
Path Formulas (V): *Release*-operator

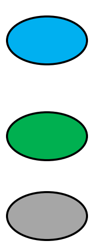
$\varphi \, \mathrm{R} \, \psi$

- ψ has to be true until and including the point where φ becomes true; if never becomes true, must remain true forever

1)

2)





- φ is true

- ψ is true

- ψ can be either true or false

φ never gets true

Formal semantics of CTL* (1)

- Notations
  - $M, s \models \varphi$ iff φ holds in state s of model M
  - $M, \pi \models \varphi$ iff φ holds along the path π in M
  - π^i : i-th suffix of π
    - $\pi = s_0, s_1, \ldots$, then $\pi^1 = s_1, \ldots$

Semantics of CTL* (2)

- *Path formulas are interpreted over a path*:
  - $M, \pi \models \varphi$
  - $M, \pi \models \mathrm{X}\ \varphi$
  - $M, \pi \models \mathrm{F}\ \varphi$
  - $M, \pi \models \varphi\ \mathrm{U}\ \psi$

Semantics of CTL* (3)

- *State formulas are interpreted over a set of states (of a path)*
  - $M, s \models p$
  - $M, s \models \neg \varphi$
  - $M, s \models \mathrm{E}\, \varphi$
  - $M, s \models \mathrm{A}\, \varphi$

CTL vs. CTL*

- CTL*, CTL and LTL have different expressive powers:
- Example:
  - In CTL there is no formula being equivalent to LTL formula **A**(**FG** *p*).
  - In LTL there is no formula equivalent to CTL formula **AG**(**EF** *p*).
  - **A**(**FG** *p*) $\vee$ **AG**(**EF** *p*) is a CTL* formula that cannot be expressed neither in CTL nor in LTL.

CTL

- Quantifiers over paths
  - **A** – **A**ll: has to hold on all paths starting from the current state.
  - **E** – **E**xists: there exists at least one path starting from the current state where holds.

- In CTL, path formulas can occur only when paired with an *A* or *E* , *i.e.* one path operator followed by a state operator.

  if φ and ψ are path formulas, then
  - X φ,
  - F φ,
  - G φ,
  - φ U ψ,
  - φ R ψ

  are path formulas

LTL (contains only path formulas)

Form of path formulas:

- If $p \in$ AP, then p is a path formula
- If φ and ψ are path formulas, then
  - $\neg\varphi$
  - $\varphi \vee \psi$
  - $\varphi \wedge \psi$
  - X φ
  - F φ
  - G φ
  - φ U ψ
  - φ R ψ

  are path formulas.

Minimal set of CTL temporal operators

- Transformations used for temporal operators :
  - **EF** φ == **E**[true **U**(φ)] (because **F** φ == [true **U**(φ)])
  - **AX** φ == ¬ **EX**(¬ φ)
  - **AG** φ == ¬ **EF**(¬ φ) == ¬ **E**[true **U**(φ)]
  - **AF** φ == **A**[true **U** φ] == ¬ **EG**(¬ φ)
  - **A**[φ **U** ψ] == ¬(**E**[(¬ ψ) **U** ¬(φ ∨ ψ)] ∨ **EG**(¬ ψ))

Summary

- CTL* is a general temporal logic that offers strong expressive power, more than CTL and LTL separately.
- CTL and LTL are practically useful enough; CTL* helps us to understand the relations between LTL and CTL.
- Next we will show how to model check CTL formuli on Kripke structures