

Temporal Logics^{1 2}

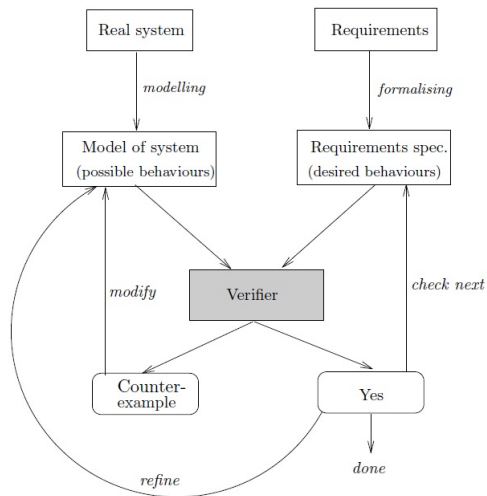
Rodica Condurache

Lecture 3

¹Bibliography : Michael Huth and Mark Ryan, "Logic in Computer Science, Modelling and reasoning about systems", Cambridge University Press 2004

²Bibliography : Christel Baier and Joost-Pieter Katoen, "Principles of Model Checking", The MIT Press

Model Checking process



Model Checking process

The formal verification requires:

A **model of the system**, typically consisting of

- **a set of states**

A state describes some information about a system at a certain moment of its behavior (eg. values of variables and program counters)

- **a transition relation**, that describes how the system can change from one state to another.

A **specification method** for expressing requirements in a formal way.

- Eg: Temporal logic formulas to specify requirements

A **set of proof rules** to determine whether the model satisfies the stated requirements.

- Eg: Algorithms implementing the semantics of the used logic

Model of Systems : Transition Systems

When transitions are made as result of actions:

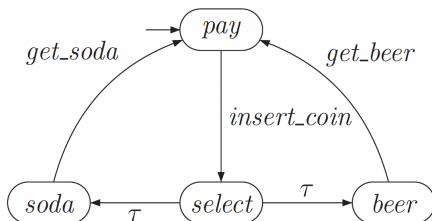
Definition (Transition System (TS))

A **transition system (TS)** over the set of atomic propositions \mathcal{AP} is a tuple $\mathcal{T} = (S, Act, \rightarrow, S_0, L)$ where

- S is a set of states,
- Act is a set of actions,
- $\rightarrow \subseteq S \times Act \times S$ is a transition relation,
- $S_0 \subseteq S$ is the set of initial states
- $L : S \rightarrow 2^{\mathcal{AP}}$ is a labeling function.

\mathcal{T} is called **finite** if S , Act , and \mathcal{AP} are finite.

Transition System : Example



- $S = \{pay, select, soda, beer\}$
- $S_0 = \{pay\}$
- $Act = \{insert_coin, get_soda, get_beer, \tau\}$.
- $L(pay) = \emptyset, L(soda) = L(beer) = \{paid, drink\}, L(select) = \{paid\}$.

Specification:

After providing a coin, the vending machine eventually delivers the drink.

Definition (executions)

Let $\mathcal{T} = (S, Act, \rightarrow, S_0, L)$ be a transition system.

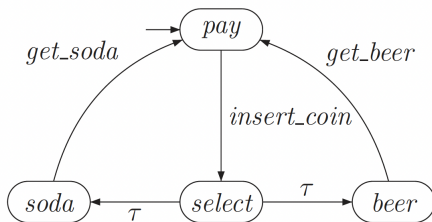
- A **finite execution** fragment is an alternating sequence of states and actions ending with a state

$$\rho = s_0\alpha_1s_1\alpha_2s_2\dots\alpha_ns_n \text{ such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \text{ for all } 0 \leq i < n, n \geq 0.$$

- An **infinite execution** fragment is an **infinite**, alternating sequence of states and actions

$$\rho = s_0\alpha_1s_1\alpha_2s_2\dots \text{ such that } s_i \xrightarrow{\alpha_{i+1}} s_{i+1} \text{ for all } 0 \leq i.$$

Executions : example



Executions in Beverage Vending Machine example:

$$\rho_1 = \text{pay} \xrightarrow{\text{coin}} \text{select} \xrightarrow{\tau} \text{soda} \xrightarrow{\text{sget}} \text{pay} \xrightarrow{\text{coin}} \text{select} \xrightarrow{\tau} \text{soda} \xrightarrow{\text{sget}} \dots$$

$$\rho_2 = \text{select} \xrightarrow{\tau} \text{soda} \xrightarrow{\text{sget}} \text{pay} \xrightarrow{\text{coin}} \text{select} \xrightarrow{\tau} \text{beer} \xrightarrow{\text{bget}} \dots$$

$$\rho = \text{pay} \xrightarrow{\text{coin}} \text{select} \xrightarrow{\tau} \text{soda} \xrightarrow{\text{sget}} \text{pay} \xrightarrow{\text{coin}} \text{select} \xrightarrow{\tau} \text{soda} .$$

Modelling Specifications : Temporal Logics

- We model specifications on Transition systems using **temporal logics**.
- elementary temporal modalities that are present in the most temporal logics include the operators:

$F(\diamond)$ “eventually” (eventually in the future)

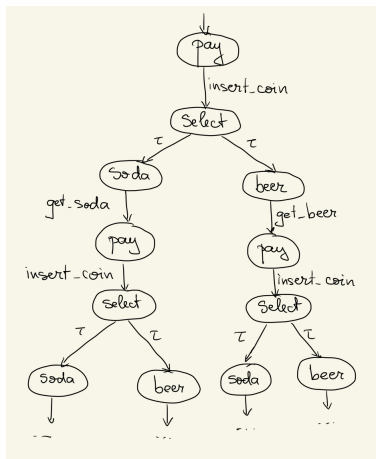
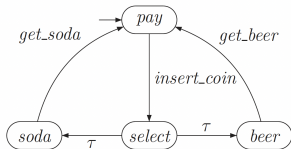
$G(\square)$ “always” (now and forever in the future)

- Different interpretations (depending on how one considers the system to change with time) ³.
 - In **linear** temporal logic, the time is linear; at each moment of time there is only one possible successor state and thus only one possible future.
 - **Branching** temporal logic is based on models where at each moment there may be several different possible futures (i.e., a state can have different possible successor states).

³Not all structures for time fall into the linear or the branching category, but these are the two most often used in the literature

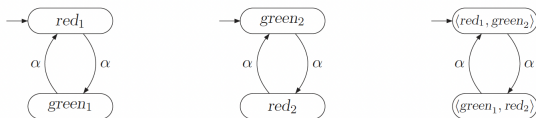
Model of Systems : possible behaviors

- The **computation tree** of a Labeled Transition System is its **acyclic unfolding**



Linear Temporal Logic : Example - Traffic Light

We consider two fully synchronized traffic lights (left and middle) and their parallel composition(right).



Desirable properties:

“The first traffic light is infinitely often green”.

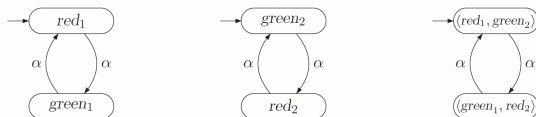
This property corresponds to the set of infinite words $s_0s_1s_2\dots$ such that $green_1 \in L(s_i)$ for infinitely many i .

$$\begin{aligned} & \{ red_1, green_2 \} \{ green_1, red_2 \} \{ red_1, green_2 \} \{ green_1, red_2 \} \dots, \\ & \emptyset \{ green_1 \} \emptyset \{ green_1 \} \emptyset \{ green_1 \} \emptyset \{ green_1 \} \emptyset \dots \\ & \{ red_1, green_1 \} \{ red_1, green_1 \} \{ red_1, green_1 \} \{ red_1, green_1 \} \dots \quad \text{and} \\ & \{ green_1, green_2 \} \{ green_1, green_2 \} \{ green_1, green_2 \} \{ green_1, green_2 \} \dots \end{aligned}$$

The word $\{ red_1, green_1 \} \{ red_1, green_1 \} \emptyset \emptyset \emptyset \dots$ does not correspond to specification.

Linear Temporal Logic : Example - Traffic Light

We consider two fully synchronized traffic lights (left and middle) and their parallel composition(right).



Desirable properties:

“The traffic lights are never both green simultaneously”

This property corresponds to the set of infinite words $s_0s_1s_2\dots$ such that $green_1 \notin L(s_i)$ or $green_2 \notin L(s_i)$ for all $i \geq 0$.

$$\begin{aligned} & \{ red_1, green_2 \} \{ green_1, red_2 \} \{ red_1, green_2 \} \{ green_1, red_2 \} \dots, \\ & \emptyset \{ green_1 \} \emptyset \{ green_1 \} \emptyset \{ green_1 \} \emptyset \{ green_1 \} \emptyset \dots \quad \text{and} \\ & \{ red_1, green_1 \} \{ red_1, green_1 \} \{ red_1, green_1 \} \{ red_1, green_1 \} \dots, \end{aligned}$$

The word $\{ red_1 green_2 \} \{ green_1, green_2 \} \dots$ does not correspond to specification.

Temporal Logics : Linear-time Temporal Logic

Definition (Linear-time Temporal Logic : LTL)

Given a set \mathcal{AP} of atomic propositions, an LTL formula over \mathcal{AP} is defined by:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid X\varphi \mid \varphi_1 \mathcal{U} \varphi_2$$

where $p \in \mathcal{AP}$.

Operator precedence		
Connective	Name	Priority
\neg	Negation	0
X	Next	0
G	Always	0
F	Eventually	0
\mathcal{U}	Until	1
\mathcal{R}	Release	1
\wedge	Conjunction	2
\vee	Disjunction	2

$$F\varphi \equiv \top \mathcal{U} \varphi$$

$$G\varphi \equiv \neg F\neg\varphi$$

$$\varphi_1 \mathcal{R} \varphi_2 \equiv \neg((\neg\varphi_2) \mathcal{U} (\neg\varphi_1))$$

Example

Mutual exclusion :

$$G\neg(\text{crit}_1 \wedge \text{crit}_2)$$

Temporal Logics : Linear-time Temporal Logic

Definition (Semantics of LTL)

Let $\pi = s_0, s_1, s_2, \dots$ be a path and φ be a LTL formula. We define the notion “ φ is true in π ”, denoted by $\pi, 0 \models \varphi$, using induction:

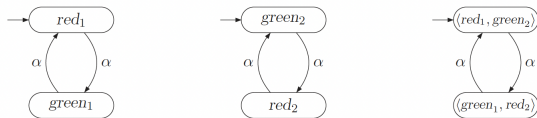
- $\pi, i \models \top$
- $\pi, i \models p$ iff $p \in L(s_i)$
- $\pi, i \models \varphi_1 \wedge \varphi_2$ iff $\pi, i \models \varphi_1$ and $\pi, i \models \varphi_2$
 $\pi, i \models \varphi_1 \vee \varphi_2$ iff $\pi, i \models \varphi_1$ or $\pi, i \models \varphi_2$
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$
- $\pi, i \models X\varphi$ iff $\pi, i+1 \models \varphi$
- $\pi, i \models \varphi_1 \mathcal{U} \varphi_2$ iff $\exists j \geq i$ such that $\pi, j \models \varphi_2$ and $\pi, k \models \varphi_1$ for all $i \leq k < j$
- $\pi, i \models \varphi_1 \mathcal{R} \varphi_2$ iff $\forall j \geq i$ such that $(\forall k \leq j) \pi, k \not\models \varphi_1$ implies $\pi, j \models \varphi_2$.

$s \models \varphi$ if and only if $\pi, 0 \models \varphi$ for all paths π starting in s

Temporal Logics : Linear-time Temporal Logic (X, U, R, F, G)

Linear Temporal Logic : Example - Traffic Light

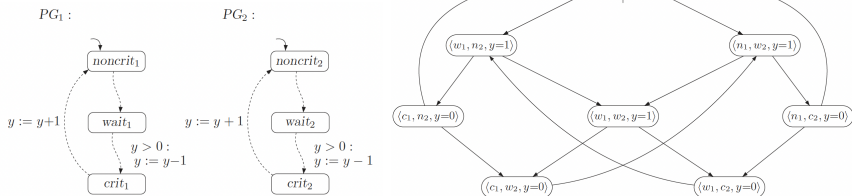
We consider two fully synchronized traffic lights (left and middle) and their parallel composition(right).



“The first traffic light is infinitely often green”. $\varphi = GFgreen_1$

“The traffic lights are never both green simultaneously” $\varphi = G(\neg green_1 \vee \neg green_2)$

Linear Temporal Logic : Example - Starvation Freedom



“The two processes are never in the critical section simultaneously”.

$$\varphi = G(\neg crit_1 \vee \neg crit_2)$$

“If a process enters waiting state, it eventually later reaches the critical state”.

$$\varphi = \bigwedge_{i \in \{1,2\}} G(wait_i \rightarrow F crit_i)$$

“Each process reaches infinitely often the critical state”.

$$\varphi = \bigwedge_{i \in \{1,2\}} GF crit_i$$

Definition (Model Checking problem for *LTL*)

input : an *LTL* formula φ , a finite Kripke model $\mathcal{M} = (S, S_0, \mathcal{AP}, \mathcal{R}, L)$ and a state $s \in S$

output : true iff $s \models \varphi$ ($\pi, 0 \models \varphi$ for all infinite paths π starting from s)

Temporal Logics : Need of branching time

- Express branching aspects of behavior: many futures are possible starting from a given state

Whenever we are in a state where p holds, it is possible to reach a state where q holds.

- cannot be expressed in LTL.

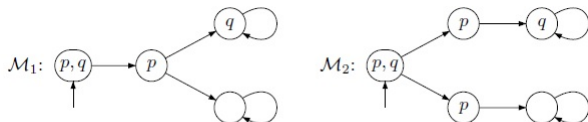


Fig. 1.1. Two models undistinguishable for LTL.

Temporal Logics : Branching-time Temporal Logic

- It supports:
 - an existential path quantifier (E)
 - an universal path quantifier (A)

Definition (Branching-time Temporal Logic : CTL*)

Given a set \mathcal{AP} of atomic propositions, an CTL* formula over \mathcal{AP} is defined by:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid E\varphi \mid A\varphi \mid X\varphi \mid \varphi_1\mathcal{U}\varphi_2$$

where $p \in \mathcal{AP}$.

Definition (Branching-time Temporal Logic : CTL)

Given a set \mathcal{AP} of atomic propositions, an CTL formula over \mathcal{AP} is defined by:

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid EX\varphi \mid AX\varphi \mid E(\varphi_1\mathcal{U}\varphi_2) \mid A(\varphi_1\mathcal{U}\varphi_2)$$

where $p \in \mathcal{AP}$.

Temporal Logics : CTL^*

Definition (Semantics of CTL and CTL^*)

Let $\pi = s_0, s_1, s_2, \dots$ be a path and φ a CTL^* formula. If π_i is the suffix of π starting from position i ,

- $\pi, i \models \top$
- $\pi, i \models p$ iff $p \in L(s_i)$
- $\pi, i \models \neg\varphi$ iff $\pi, i \not\models \varphi$
- $\pi, i \models \varphi_1 \vee \varphi_2$ iff $\pi, i \models \varphi_1$ or $\pi, i \models \varphi_2$
- $\pi, i \models X\varphi$ iff $\pi, i+1 \models \varphi$
- $\pi, i \models \varphi_1 \mathcal{U} \varphi_2$ iff $\exists j \geq 0$ such that $\pi_j \models \varphi_2$ and $\pi_k \models \varphi_1$ for all $0 \leq k < j$
- $\pi, i \models E\varphi$ iff there is an infinite path $\pi' = s'_0, s'_1, s'_2, \dots$ s.t. $s'_0 = s_i$ and $\pi', 0 \models \varphi$
- $\pi, i \models A\varphi$ iff for every infinite path $\pi' = s'_0, s'_1, s'_2, \dots$ s.t. $s'_0 = s_i$, we have $\pi', 0 \models \varphi$

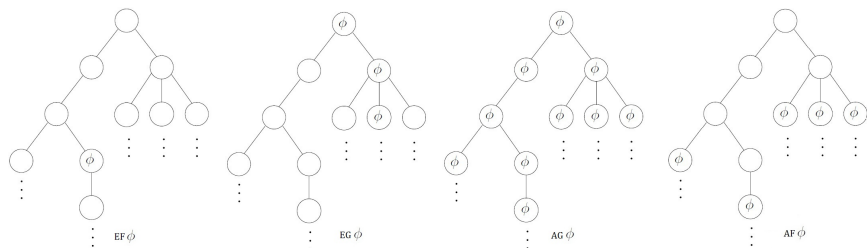
CTL formulas can be evaluated over states ($s \models \varphi$):

For any CTL formula φ , any paths π and π' s.t. $\pi(i) = \pi'(j)$, we have

$$\pi, i \models \varphi \Leftrightarrow \pi', j \models \varphi$$

Temporal Logics : CTL^*

Computation Tree Logic (CTL) - Examples



We use the equivalences:

$$AF\phi \equiv \neg EG\neg\phi$$

$$AG\phi \equiv \neg EF\neg\phi$$

Temporal Logics : *CTL*

Definition (Model Checking problem for *CTL*)

input : a *CTL* formula φ , a finite Kripke model $\mathcal{M} = (S, S_0, \mathcal{AP}, \mathcal{R}, L)$ and a state $s \in S$

output : true iff $s \models \varphi$ ($\pi, 0 \models \varphi$ for all infinite paths π starting from s)

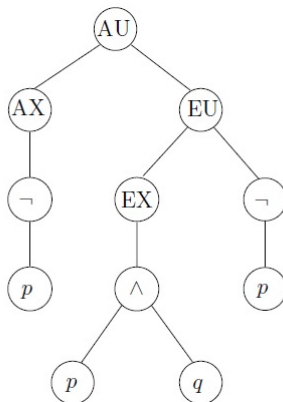
Theorem

*The Model Checking problem from *CTL* is Pspace-complete.*

Temporal Logics : Parse tree of CTL

- In order to evaluate a formula, we start from the innermost subformulas
- In fact, we build the **Parse tree** for each formula and travers it from bottom to top

Example



The Parse tree for the formula $A(AX\neg p)U E(EX(p \wedge q)U\neg p)$

(Solving Model Checking Problem)

- *Let define:*

$$pre_{\exists}(Y) = \{s \in S \mid \exists s' \in Y \text{ s.t. } (s, s') \in \mathcal{R}\}$$

$$pre_{\forall}(Y) = \{s \in S \mid \mathcal{R}(s) \subseteq Y\}$$

- *Compute* $\llbracket \varphi \rrbracket = \{s \in S \mid s \models \varphi\}$

$$\llbracket p \rrbracket = \{s \in S \mid p \in L(s)\}$$

$$\llbracket \neg \varphi \rrbracket = S \setminus \llbracket \varphi \rrbracket$$

$$\llbracket \varphi_1 \vee \varphi_2 \rrbracket = \llbracket \varphi_1 \rrbracket \cup \llbracket \varphi_2 \rrbracket$$

$$\llbracket EX\varphi \rrbracket = pre_{\exists}(\llbracket \varphi \rrbracket)$$

$$\llbracket AF\varphi \rrbracket = MC_{CTL}^{AF}(\varphi)$$

$$\llbracket E(\varphi_1 \mathcal{U} \varphi_2) \rrbracket = MC_{CTL}^{EU}(\varphi_1, \varphi_2)$$

- *Test if the input state* $s \in \llbracket \varphi \rrbracket$

- $MC_{CTL}^{EU}(\varphi_1, \varphi_2)$ is computed as:

- $Y := \emptyset; Z := \llbracket \varphi_2 \rrbracket;$
- **while** $Z \not\subseteq Y$ **do**:
 - $Y = Y \cup Z;$
 - $Z = pre_{\exists}(Y) \cap \llbracket \varphi_1 \rrbracket$
- **return** Y

Stop when cannot add nodes in Y!

- $MC_{CTL}^{AF}(\varphi)$ is computed as:

- $Y := S; Z := \llbracket \varphi \rrbracket;$
- **while** $Y \neq Z$ **do**:
 - $Y := Z$
 - $Z = Z \cup pre_{\forall}(Y)$
- **return** Y

- Also use equivalences:

$$AX\varphi \equiv \neg EX\neg\varphi$$

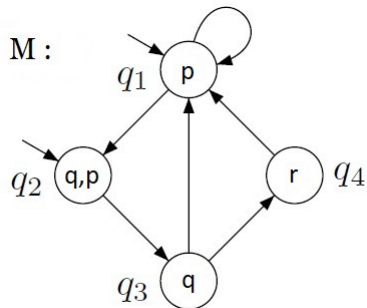
$$A(\varphi_1 \mathcal{U} \varphi_2) \equiv \neg(E(\neg\varphi_2 \mathcal{U}(\neg\varphi_1 \wedge \neg\varphi_2)) \vee EG\neg\varphi_2)$$

$$EF\varphi \equiv E(\top \mathcal{U} \varphi)$$

$$EG\varphi \equiv \neg AF\neg\varphi$$

$$AG\varphi \equiv \neg EF\neg\varphi$$

Example



Compute $\llbracket AXq \rrbracket$, $\llbracket EXp \rrbracket$ and $\llbracket E(q \cup r) \rrbracket$

Exercise 1 - LTL

Let consider a boolean circuit with input x , output y and two registers r_1 and r_2 . Translate the following properties as LTL formulas over $\mathcal{AP} = \{x, y, r_1, r_2\}$:

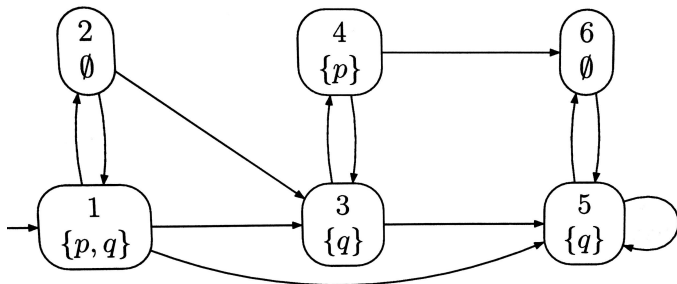
- 1 "it is impossible to get two consecutive 1 as output"
- 2 "each time the input is 1, at most two ticks later, the output will be 1"
- 3 "each time the input is 1, the register contents remain the same over the next tick."
- 4 "register r_1 is infinitely often 1"

Exercise 2 - CTL and CTL*

Are the following formulas equivalent?

- 1 $AXAG\varphi$ and $AXG\varphi$
- 2 $EXEG\varphi$ and $EXG\varphi$
- 3 $A(\varphi \wedge \psi)$ and $A\varphi \wedge A\psi$
- 4 $E(\varphi \wedge \psi)$ and $E\varphi \wedge E\psi$
- 5 $\neg A(\varphi \rightarrow \psi)$ and $E(\varphi \wedge \neg\psi)$

Exercise 3 - CTL and CTL^*



- 1 Compute $\llbracket EFp \rrbracket$
- 2 Compute $\llbracket AFq \rrbracket$
- 3 Compute $\llbracket \varphi \rrbracket$ where $\varphi = E(qU(p \wedge \neg q))$
- 4 Compute $\llbracket \varphi \rrbracket$ where $EGq \vee (EGp \wedge EFq)$