# Verifica formale di proprietà mediante model checking

Angelo Gargantini

June 1, 2016

## Outline

TESTO DI RIFERIMENTO: M.R.A. Ruth, M.D. Ryan Logic in Computer Science Modelling and Reasoning about systems - Capitolo 3 - allegato a questi appunti

## Motivation

- There is a great advantage in being able to verify the correctness of computer systems, whether they are hardware, software, or a combination. This is most obvious in the case of safety-critical systems, but also applies to those that are commercially critical, such as mass-produced chips, mission critical, etc.
- Formal verification methods have quite recently become usable by industry and there is a growing demand for professionals able to apply them.
- We study a fully automatic way to perform formal verification
  - not rule-based
  - called **model checking**

## Formal verification by model checking

- Le tecniche di verifica formale sono generalmente viste come la somma di tre componenti:
  - Un framework in cui modellare il sistema che vogliamo analizzare
  - Un linguaggio di specifica delle proprietà da verificare
  - Un metodo per verificare che il sistema soddisfi le proprietà specificate.
- Solitamente il Model Checking si basa sull'utilizzo di una logica temporale. Quindi, le tre componenti possono essere costituite come segue:
  - Si costruisce un modello M che descrive il comportamento del sistema
  - Si codifica la proprietà da verificare in una formula temporale $\phi$
  - Si chiede al model checker di verificare che $M \models \phi$

## Formal verification by model checking

- Le tecniche di verifica formale sono generalmente viste come la somma di tre componenti:
  - Un framework in cui modellare il sistema che vogliamo analizzare
  - Un linguaggio di specifica delle proprietà da verificare
  - Un metodo per verificare che il sistema soddisfi le proprietà specificate.
- Solitamente il Model Checking si basa sull'utilizzo di una logica temporale. Quindi, le tre componenti possono essere costituite come segue:
  - Si costruisce un modello M che descrive il comportamento del sistema
  - Si codifica la proprietà da verificare in una formula temporale $\phi$
  - Si chiede al model checker di verificare che $M \models \phi$

## Logiche temporali

- Esistono diverse logiche temporali che possono essere divise in due clasi fondamentali:
  - le linear-time logics (LTL) e le branching-time logics (CTL).
- LTL considera il tempo come un insieme di cammini, dove cammino é una sequenza di istanti di tempo
- CTL rappresenta il tempo come un albero, con radice l'istante corrente
- Un'altra classificazione divide tra tempo continuo e discreto. Noi studieremo solo logiche discrete e senza metrica.

## LTL sintassi

- La logica è costruita su di un insieme di formule atomiche AP {p, q, r, ...} che rappresentano descrizioni atomiche del sistema
- Definiamo in maniera ricorsiva le formule LTL:
- (1) come la logica proposizionale

$$\phi ::= \quad \top | \bot | p \in AP | \quad \neg\phi | \phi \wedge \phi | \phi \vee \phi | \phi \rightarrow \phi |$$

- $\top, \bot, \neg, \wedge, \vee, \rightarrow$ sono connettivi logici classici

## LTL sintassi - (2) operatori temporali

$$\phi ::= \begin{array}{cc} \top|\bot|p \in AP| & \neg\phi|\phi \land \phi|\phi \lor \phi|\phi \rightarrow \phi| \\ X\phi|F\phi|G\phi| & \phi U\phi|\phi W\phi|\phi R\phi \end{array}$$

- X, F, G, U,W, R sono connettivi temporali
  - In particolare:
  - X means ‘neXt state,’
  - F means ‘some Future state,’ and
  - G means ‘all future states (Globally).’
- The next three, U, R and W are called ‘Until,’ ‘Release’ and ‘Weak-until’ respectively.

## Precedenza degli operatori

Convention 3.2 The unary connectives (consisting of $\neg$ and the temporal connectives X, F and G) bind most tightly. Next in the order come U, R and W; then come $\wedge$ and $\vee$; and after that comes $\rightarrow$.

- alcuni esempi di LTL con e senza parentesi

## Semantica per LTL

- The kinds of systems we are interested in verifying using LTL may be modelled as transition systems. A transition system models a system by means of states (static structure) and transitions (dynamic structure).

A transition system $M = (S, \rightarrow, L)$ is

- a set of states S endowed
- with a transition relation $\rightarrow$ (a binary relation on S), such that every $s \in S$ has some $s' \in S$ with $s \rightarrow s'$, and
- a labelling function $L : S \rightarrow \mathcal{P}(Atoms)$

I transition system sono i nostri *modelli*.

## Labelling function

- a labelling function $L : S \to \mathcal{P}(Atoms)$

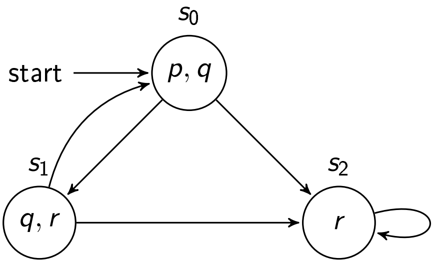
$\mathcal{P}(Atoms)$ è il powerset – l'insieme delle parti – di Atoms

- L is that it is just an assignment of truth values to all the propositional atoms, as it was the case for propositional logic (we called that a valution)
- The difference now is that we have more than one state, so this assignment depends on which state s the system is in: L(s) contains all atoms which are true in state s.

## Graphical representation

- all the information about a (finite) transition system M can be expressed using directed graphs whose nodes (which we call states) contain all propositional atoms that are true in that state.

**Example:** M has only three states s0, s1, and s2. The only possible transitions are s0 $\rightarrow$ s1, s0 $\rightarrow$ s2, s1 $\rightarrow$ s0, s1 $\rightarrow$ s2 and s2 $\rightarrow$ s2; and if L(s0) = {p, q}, L(s1) = {q, r} and L(s2 ) = {r}:



## No deadlock

- The requirement in Definition that for every s $\in$ S there is at least one s' $\in$ S such that s $\rightarrow$ s' means that no state of the system can 'deadlock.'
  - This is a technical convenience, and in fact it does not represent any real restriction on the systems we can model. If a system did deadlock, we could always add an extra state sd representing deadlock,

- un esempio di deadlock

## Path

- A **path** in a model M = (S, →, L) is an infinite sequence of states s1, s2 , s3 , . . . . in S such that, for each i ≥ 1, si → si+1.
- We write the path as s1 → s2 → . . . .
- We write $\pi^i$ for the suffix starting at $s_i$, e.g., $\pi^3$ is s3 → s4 → . . . .
- Esempio

## Validità di una formula LTL su un path (prop)

**Definition**

Let M = (S, $\rightarrow$, L) be a model and $\pi = s1 \rightarrow \ldots$ be a **path** in M. Whether $\pi$ satisfies an LTL formula is defined by the satisfaction relation $\models$ as follows:

1. $\pi \models \top$
2. $\pi \not\models \bot$
3. $\pi \models p$ iff $p \in L(s_1)$
4. $\pi \models \neg\varphi$ iff $\pi \not\models \varphi$
5. $\pi \models \varphi1 \wedge \varphi2$ iff $\pi \models \varphi1$ and $\pi \models \varphi2$
6. $\pi \models \varphi1 \vee \varphi2$ iff $\pi \models \varphi1$ or $\pi \models \varphi2$
7. $\pi \models \varphi1 \rightarrow \varphi2$ iff $\pi \models \varphi2$ whenever $\pi \models \varphi1$

## Validità di una formula LTL in un path (time)

**Definition**

Let M = (S, →, L) be a model and π = s1 → . . . be a path in M. Whether π satisfies an LTL formula is defined by the satisfaction relation ⊨ as follows:

8. $\pi \models X\ \varphi$ iff $\pi^2 \models \varphi$
9. $\pi \models G\ \varphi$ iff, for all $i \geq 1$, $\pi^i \models \varphi$
10. $\pi \models F\ \varphi$ iff there is some $i \geq 1$ such that $\pi^i \models \varphi$

## Validità di una formula LTL (time 2)

11. **(Until)** $\pi \models a \; U \; b$ iff there is some $i \geq 1$ such that $\pi^i \models b$ and for all $j = 1, \ldots, i-1$ we have $\pi^j \models a$
12. **(Weak Until)** $\pi \models a \; W \; b$ iff either there is some $i \geq 1$ such that $\pi^i \models b$ and for all $j = 1, \ldots, i-1$ we have $\pi^j \models a$; or for all $k \geq 1$ we have $\pi^k \models a$

- U, which stands for 'Until,' is the most commonly encountered one of these. The formula *a U b* holds on a path if it is the case that a holds continuously until b holds. Moreover, a U b actually demands that b does hold in some future state.
- Weak-until is just like U, except that *a W b* does not require that b is eventually satisfied along the path in question, which is required by *a U b*.

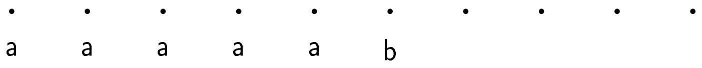
## Validità di una formula LTL (time 3)

13. **(Release)** $\pi \models a\ R\ b$ iff either there is some $i \geq 1$ such that $\pi^i \models a$ and for all $j = 1, \ldots, i$ we have $\pi^j \models b$, or for all $k \geq 1$ we have $\pi^k \models b$.

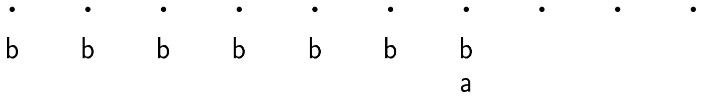
- It is called ‘Release’ because its definition determines that b must remain true up to and including the moment when a becomes true (if there is one); a ‘releases’ b.
- Release R is the dual of U; that is, a R b is equivalent to $\neg(\neg a\ U\ \neg b)$.

## Rappresentazione grafica

- Until: a is true until b become true, a U b



- Release: a releases b: a R b



aggiungere grafica per weak until

## Formula valida

- Quando una formula è **valida** per una macchina M (e non solo per un path) ?

**Definition**

Suppose $M = (S, \rightarrow, L)$ is a model, $s \in S$, and $\varphi$ an LTL formula. We write $M, s \models \varphi$ if, for every execution path $\pi$ of M starting at s, we have $\pi \models \varphi$

**Example**

Figura 3.3 e figura 3.5, alcune formule

## Practical Pattern of specifications

- *Safety* properties:
  - something is always true $G\phi$
  - something bad never happens $G\neg\phi$,
- *Liveness* properties:
  - something will happen $F\phi$
  - something good keeps happening ($GF\psi$ or $G(\phi \rightarrow F\psi)$)
- Esempi piu' complessi - 3.2

## Important equivalences between LTL formulas

We say that two LTL formulas $\varphi$ and $\psi$ are semantically equivalent, or simply equivalent, writing $\varphi \equiv \psi$, if for all models M and all paths $\pi$ in M: $\pi \models \varphi$ iff $\pi \models \psi$.

- solite equivalenze di and, or, not ....

## Until e weak until

A weak until binary operator, denoted W, with semantics similar to that of the until operator but the stop condition is not required to occur (similar to release).

- $\varphi \, W \, \psi \equiv (\varphi \, U \, \psi) \lor G \, \varphi$

Both U and R can be defined in terms of the weak until:

- Until and Weak until: $\varphi \, U \, \psi \equiv \varphi \, W \, \psi \land F \, \psi$

Also R can be defined in terms of W

- $\varphi \, W \, \psi \equiv (\varphi \, U \, \psi) \lor G \, \varphi \equiv \varphi \, U \, (\psi \lor G \, \varphi) \equiv \psi \, R \, (\psi \lor \varphi)$ $\varphi \, U \, \psi \equiv F\psi \land (\varphi \, W \, \psi)$ $\varphi \, R \, \psi \equiv \psi \, W \, (\psi \land \varphi)$

## Until e weak until

A weak until binary operator, denoted W, with semantics similar to that of the until operator but the stop condition is not required to occur (similar to release).

- $\varphi \; W \; \psi \equiv (\varphi \; U \; \psi) \lor G \; \varphi$

Both U and R can be defined in terms of the weak until:

- Until and Weak until: $\varphi \; U \; \psi \equiv \varphi \; W \; \psi \land F \; \psi$

Also R can be defined in terms of W

- $\varphi \; W \; \psi \equiv (\varphi \; U \; \psi) \lor G \; \varphi \equiv \varphi \; U \; (\psi \lor G \; \varphi) \equiv \psi \; R \; (\psi \lor \varphi) \; \varphi \; U \; \psi \equiv F\psi \land (\varphi \; W \; \psi) \; \varphi \; R \; \psi \equiv \psi \; W \; (\psi \land \varphi)$

## F and G duality

- F and G are duals:
  - $\neg G \varphi \equiv F \neg \varphi$ $\neg F \varphi \equiv G \neg \varphi$
- X is dual of itself: $\neg X \varphi \equiv X \neg \varphi$
- U and R are duals of each other:
  - $\neg(\varphi U \psi) \equiv \neg \varphi R \neg \psi$ $\neg(\varphi R \psi) \equiv \neg \varphi U \neg \psi$

## F and G duality

- F and G are duals:
  - $\neg G \varphi \equiv F \neg \varphi$ $\qquad \neg F \varphi \equiv G \neg \varphi$
- X is dual of itself: $\neg X \varphi \equiv X \neg \varphi$
- U and R are duals of each other:
  - $\neg(\varphi U \psi) \equiv \neg \varphi R \neg \psi$ $\qquad \neg(\varphi R \psi) \equiv \neg \varphi U \neg \psi$

## F and G duality

- F and G are duals:
  - $\neg G \varphi \equiv F \neg \varphi$  $\neg F \varphi \equiv G \neg \varphi$
- X is dual of itself: $\neg X \varphi \equiv X \neg \varphi$
- U and R are duals of each other:
  - $\neg(\varphi U \psi) \equiv \neg \varphi R \neg \psi$  $\neg(\varphi R \psi) \equiv \neg \varphi U \neg \psi$

## Distributive

- It's also the case that F distributes over $\vee$ and G over $\wedge$ , i.e.,
  - $F(\varphi \vee \psi) \equiv F\varphi \vee F\psi$ $\quad$ $G(\varphi \wedge \psi) \equiv G\varphi \wedge G\psi$
- But F does not distribute over $\wedge$ and G does not over $\vee$.
- F and G can be written as follows using U
  - $F\varphi \equiv \top U \varphi$ $\quad$ $G\varphi \equiv \bot R \varphi$

## Distributive

- It's also the case that F distributes over $\vee$ and G over $\wedge$ , i.e.,
  - $F(\varphi \vee \psi) \equiv F\varphi \vee F\psi$ $\qquad$ $G(\varphi \wedge \psi) \equiv G\varphi \wedge G\psi$
- But F does not distribute over $\wedge$ and G does not over $\vee$.
- F and G can be written as follows using U
  - $F\varphi \equiv \top U \varphi$ $\qquad$ $G\varphi \equiv \bot R \varphi$

## Distributive

- It's also the case that F distributes over $\vee$ and G over $\wedge$ , i.e.,
  - $F(\varphi \vee \psi) \equiv F\varphi \vee F\psi$ $\quad G(\varphi \wedge \psi) \equiv G\varphi \wedge G\psi$
- But F does not distribute over $\wedge$ and G does not over $\vee$.
- F and G can be written as follows using U
  - $F\varphi \equiv \top U \varphi$ $\quad G\varphi \equiv \bot R \varphi$

## Adequate sets of connectives for LTL

Non tutti i connettivi sono necessari. Basterebbero di meno, ma per facilità nelle scritture delle formule li usiamo tutti.

## Pattern of LTL properties

Esistono dei pattern pratici per la specifica mediante LTL di proprietà comuni:
`http://patterns.projects.cis.ksu.edu/documentation/patterns/ltl.shtml`
Alcune volte gli operatori si indicano così: G anche [] $\Box$, F anche <>$\Diamond$

| **Absence** – P is false: | |
|---|---|
| Globally | **G** (!P) |
| Before R | **F** R -> (!P **U** R) |
| After Q | **G** (Q -> G (!P)) |
| Between Q and R | **G** ((Q & !R & **F** R) -> (!P **U** R)) |

## Pattern (**Existence**)

| **Existence** P becomes true : | |
|---|---|
| Globally | F (P) |
| (*) Before R | !R W (P & !R) |
| After Q | G (!Q) \| F (Q & F P)) |
| (*) Between Q and R | G (Q & !R -> (!R W (P & !R))) |
| (*) After Q until R | G (Q & !R -> (!R U (P & !R))) |

## Pattern (Universality)

| Universality P is true : | |
|---|---|
| Globally | G (P) |
| Before R | F R -> (P U R) |
| After Q | G (Q -> G (P)) |
| Between Q and R | G ((Q & !R & F R) -> (P U R)) |
| (*) After Q until R | G (Q & !R -> (P W R)) |

## Altri Pattern

- **Precedence** S precedes P
- **Response** S responds to P :
- **Precedence** Chain ...

## Example: mutual exclusion

**mutual exclusion**

When concurrent processes share a resource (such as a file on a disk or a database entry), it may be necessary to ensure that they do not have access to it at the same time. Several processes simultaneously editing the same file would not be desirable

a process to access a critical resource must be in critical section

## mutual exclusion desired properties

Safety Only one process is in its critical section at any time.

Liveness: Whenever any process requests to enter its critical section, it will eventually be permitted to do so.

Non-blocking: A process can always request to enter its critical section.

No strict sequencing: Processes need not enter their critical section in strict sequence.

## mutual exclusion desired properties

Safety Only one process is in its critical section at any time.

Liveness: Whenever any process requests to enter its critical section, it will eventually be permitted to do so.

Non-blocking: A process can always request to enter its critical section.

No strict sequencing: Processes need not enter their critical section in strict sequence.

## mutual exclusion desired properties

Safety Only one process is in its critical section at any time.

Liveness: Whenever any process requests to enter its critical section, it will eventually be permitted to do so.

Non-blocking: A process can always request to enter its critical section.

No strict sequencing: Processes need not enter their critical section in strict sequence.

## mutual exclusion desired properties

Safety Only one process is in its critical section at any time.

Liveness: Whenever any process requests to enter its critical section, it will eventually be permitted to do so.

Non-blocking: A process can always request to enter its critical section.

No strict sequencing: Processes need not enter their critical section in strict sequence.

## mutual exclusion desired properties

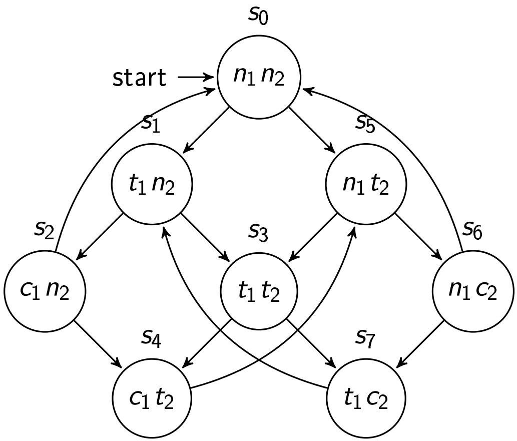
Safety Only one process is in its critical section at any time.

Liveness: Whenever any process requests to enter its critical section, it will eventually be permitted to do so.

Non-blocking: A process can always request to enter its critical section.

No strict sequencing: Processes need not enter their critical section in strict sequence.

# mutual exclusion first model



Every process can be in state: {non critical (n), trying to enter (t), critical state (c)}.

## mutual exclusion properties

**Safety** $G \neg ( c1 \wedge c2 )$. OK

**Liveness:** $G ( t1 \rightarrow F\ c1 )$. This is FALSE

**Non-blocking:** ... non riesco ad esprimerla in LTL

**No strict sequencing:** trovo un path in cui non c'è strict sequencing

## mutual exclusion properties

Safety $G \neg ( c1 \wedge c2 )$. OK

Liveness: $G ( t1 \rightarrow F\ c1 )$. This is FALSE

Non-blocking: ... non riesco ad esprimerla in LTL

No strict sequencing: trovo un path in cui non c'è strict sequencing

## mutual exclusion properties

Safety $G \neg ( c1 \wedge c2 )$. OK

Liveness: $G ( t1 \rightarrow F\ c1 )$. This is FALSE

Non-blocking: ... non riesco ad esprimerla in LTL

No strict sequencing: trovo un path in cui non c'è strict sequencing

## mutual exclusion properties

Safety $G \neg ( c1 \wedge c2 )$. OK

Liveness: $G ( t1 \rightarrow F\ c1 )$. This is FALSE

Non-blocking: ... non riesco ad esprimerla in LTL

No strict sequencing: trovo un path in cui non c'è strict sequencing

## mutual exclusion properties

Safety $G \neg ( c1 \wedge c2 )$. OK

Liveness: $G ( t1 \rightarrow F\ c1 )$. This is FALSE

Non-blocking: ... non riesco ad esprimerla in LTL

No strict sequencing: trovo un path in cui non c'è strict sequencing

## Limiti LTL

Ricorda la definizione:

**Definition**

Suppose M is a model, s $\in$ S ,and $\varphi$ an LTL formula. We write M ,s $\models \varphi$ if, for every execution path $\pi$ of M starting at s, we have $\pi \models \varphi$

- Qundi M,s $\models$**F***a* vuol dire per ogni path a partire da s *a* accade
- Come faccio a dire che non sempre accade in futuro ma può accadere?

## Outline

## The NuSMV model checker

Cenni su come

- come scaricarlo `http://nusmv.fbk.eu/`
- come installarlo
- come eseguirlo
- usare il plugin nuseen (sviluppato da Unibg)

## primo esempio

A system can be ready or busy. When arrives a request, it becomes busy (but it may become busy even without any request).

**Figura**

- MODULE
- VAR
- ASSIGN (machine)
- LTLSPEC (property)

## Esempi SMV

- vedi lucidi per NuSMV

## CTL

COMPUTATION TREE LOGIC - CTL La CTL è una logica con connettivi che ci permette di specificare proprietà temporali.

- Essendo una logica branching-time, i suoi *modelli* sono rappresentabili mediante una struttura ad albero in cui il futuro non è deterministico: esistono differenti computazioni o paths nel futuro e uno di questi sarà il percorso realizzato.

**Cosa è un modello per una logica proposizionale ???**

- Un assegnamento di un valore di verità ad ogni proposizione
- che rende vera la formula

- $a \vee b \wedge c$: trova un modello

## CTL sintassi

- La logica è costruita su di un insieme di formule atomiche AP {p, q, r, ...} che rappresentano descrizioni atomiche del sistema
- Definiamo in maniera induttiva le formule CTL:

$$\phi ::= \top | \bot | p \in AP | \neg\phi | \phi \wedge \phi | \phi \vee \phi | \phi \rightarrow \phi |$$

- $\top, \bot, \neg, \wedge, \vee, \rightarrow$ sono connettivi logici classici

## CTL sintassi

- Operatori temporali:

$$\phi ::= \begin{array}{cc} AX\phi|EX\phi & AF\phi|EF\phi \\ |A[\phi U\phi]|E[\phi U\phi] & AG\phi|EG\phi| \end{array}$$

- $\top, \bot, \neg, \wedge, \vee, \rightarrow$ sono connettivi logici classici
- AX, EX, AG, EG, AU, EU, AF e EF sono connettivi temporali
- In particolare: A sta per "along All paths" (inevitably) E sta per "along at least (there Exists) one path" (possibly)
- X, F, G e U sono gli operatori della logica temporale lineare
  - Nota Bene: AU e EU sono operatori binari e i simboli X, F, G e U non possono occorrere se non preceduti da A o E e viceversa.

## Priorità degli operatori

- Convenzione sull' ordinamento: gli operatori unary (AG, EG, AF, EF, AX, EX) legano con priorità più elevata, seguono gli operatori binary Λ, V, e dopo ancora —>, AU ed EU.
- Esempi di formule CTL ben-formate
  - AG (q —> EG r)
  - EF E(r U q)
  - A[p U EF r]
  - EF EG p —> AF r

## Attenzione

- Esempi di formule CTL non ben-formate
  - EF G r
  - A!G!p
  - F[r U q]
  - EF(r U q)
  - AEF r
  - A[(r U q) /\ (p U r)]

## Semantica per CTL (brief)

**Definition**

Let M = (S, →, L) be a model for CTL, s in S, φ a CTL formula. The relation $M, s \models \varphi$ is defined by structural induction on φ.

- If φ is atomic, satisfaction is determined by L.
- If the top-level connective of φ is a boolean connective ( ∧ , ∨ , ¬ , etc.) then the satisfaction question is answered by the usual truth-table definition and further recursion down φ.
- If the top level connective is an operator beginning A, then satisfaction holds if all paths from s satisfy the 'LTL formula' resulting from removing the A symbol.
- Similarly, if the top level connective begins with E, then satisfaction holds if some path from s satisfy the 'LTL formula' resulting from removing the E.

## Semantic of CTL

Non temporal formula are treated as usual

1. $M, s \models \top$
2. $M, s \not\models \bot$
3. $M, s \models$p iff p $\in$ L($s$)
4. $M, s \models \neg$φ iff π $M, s \not\models$φ
5. $M, s \models$φ1 $\wedge$ φ2 iff $M, s \models$φ1 and $M, s \models$φ2
6. $M, s \models$φ1 $\vee$ φ2 iff $M, s \models$φ1 or $M, s \models$φ2
7. $M, s \models$φ1 $\rightarrow$ φ2 iff $M, s \models$φ2 whenever $M, s \models$φ1

## Validità di una formula CTL (time)

8. $M, s \models$AX φ iff forall $s_1$such that $s \rightarrow s_1$ we have $M, s_1 \models$φ
9. $M, s \models$EX φ iff some $s_1$such that $s \rightarrow s_1$ we have $M, s_1 \models$φ
10. $M, s \models$AG φ iff, for all paths $s \rightarrow s_1 \rightarrow s_2 \ldots$ and all $s_i$ along the path, we have $M, s \models$φ
11. $M, s \models$EG φ iff, there is a path $s \rightarrow s_1 \rightarrow s_2 \ldots$ and all $s_i$ along the path, we have $M, s \models$φ

- AX: 'in every next state.'
- EX: 'in some next state.'
- AG: for All computation paths beginning in s the property φ holds Globally
- EG: there Exists a path beginning in s such that φ holds Globally along the path.

## Validità di una formula CTL (time 2)

12. $M, s \models$AF φ iff, for all paths $s \to s_1 \to s_2 \dots$ there exists some $s_i$ along the path, we have $M, s \models$φ
13. $M, s \models$EF φ iff, there is a path $s \to s_1 \to s_2 \dots$ and for some $s_i$ along the path, we have $M, s \models$φ

- AF: for All computation paths beginning in s there will be some Future state where φ holds.
- EF: there Exists a computation path beginning in s such that φ holds in some Future state;

## Validità di una formula CTL (time 3)

11. $M, s \models A[\phi_1 U \phi_2]$ iff, for all paths $s \to s_1 \to s_2 \ldots$ , that path satisfies $\phi_1 U \phi_2$i.e., there is some $s_i$ along the path, such that $M, s \models \phi_2$, and, for each j < i, we have $M, s \models \phi_1$.
12. $M, s \models E[\phi_1 U \phi_2]$ iff, there exists a path $s \to s_1 \to s_2 \ldots$ , that path satisfies $\phi_1 U \phi_2$.

- A U All computation paths beginning in s satisfy that φ1 Until φ2 holds on it.
- E U there Exists a computation path beginning in s such that φ1 Until φ2 holds on it.

## Esempio

Figura 3.3 e computation tree 3.5

## Pattern of CTL properties

Esistono dei pattern pratici per la specifica mediante CTL di proprietà comuni:
`http://patterns.projects.cis.ksu.edu/documentation/patterns/ctl.shtml`

| **Absence** – P is false: | |
|---|---|
| Globally | AG(!P) |
| Before R | A[(!P \| AG(!R)) W R] |
| After Q | AG(Q -> AG(!P)) |

Many of the mappings use the weak until operator (W) which is related to the strong until operator (U) by the following equivalences:
A[x W y] = !E[!y U (!x & !y)]
E[x U y] = !A[!y W (!x & !y)]

## Pattern (**Existence**)

| **Existence** P becomes true : | |
|---|---|
| Globally | AF(P) |
| (*) Before R | A[!R W (P & !R)] |
| After Q | A[!Q W (Q & AF(P))] |
| (*) Between Q and R | AG(Q & !R -> A[!R W (P & !R)]) |
| (*) After Q until R | AG(Q & !R -> A[!R U (P & !R)]) |

## Pattern (Universality)

| Universality P is true : | |
|---|---|
| Globally | AG(P) |
| (*) Before R | A[(P \| AG(!R)) W R] |
| After Q | AG(Q -> AG(P)) |
| (*) Between Q and R | AG(Q & !R -> A[(P \| AG(!R)) W R]) |
| (*) After Q until R | AG(Q & !R -> A[P W R]) |

## Practical patterns of specifications

- It is possible to get to a state where **started** holds, but **ready** doesn't: EF ( **started** $\wedge$ $\neg$**ready** ). To express impossibility, we simply negate the formula.
- For any state, if a request (of some resource) occurs, then it will eventually be acknowledged: AG ( **requested** $\rightarrow$ AF **acknowledged** ).
- A certain process is enabled infinitely often on every computation path: AG (AF enabled ).
- From any state it is possible to get to a restart state: AG (EF restart ).
- Altri esempi

## Practical patterns of specifications

- It is possible to get to a state where **started** holds, but **ready** doesn't: EF ( **started** $\wedge$ $\neg$**ready** ). To express impossibility, we simply negate the formula.
- For any state, if a request (of some resource) occurs, then it will eventually be acknowledged: AG ( **requested** $\rightarrow$ AF **acknowledged** ).
- A certain process is enabled infinitely often on every computation path: AG (AF enabled ).
- From any state it is possible to get to a restart state: AG (EF restart ).
- Altri esempi

## Practical patterns of specifications

- It is possible to get to a state where **started** holds, but **ready** doesn't: EF ( **started** $\wedge$ $\neg$**ready** ). To express impossibility, we simply negate the formula.
- For any state, if a request (of some resource) occurs, then it will eventually be acknowledged: AG ( **requested** $\rightarrow$ AF **acknowledged** ).
- A certain process is enabled infinitely often on every computation path: AG (AF enabled ).
- From any state it is possible to get to a restart state: AG (EF restart ).
- Altri esempi

## Practical patterns of specifications

- It is possible to get to a state where **started** holds, but **ready** doesn't: EF ( **started** $\wedge$ $\neg$**ready** ). To express impossibility, we simply negate the formula.
- For any state, if a request (of some resource) occurs, then it will eventually be acknowledged: AG ( **requested** $\rightarrow$ AF **acknowledged** ).
- A certain process is enabled infinitely often on every computation path: AG (AF enabled ).
- From any state it is possible to get to a restart state: AG (EF restart ).
- Altri esempi

## Practical patterns of specifications

- It is possible to get to a state where **started** holds, but **ready** doesn't: EF ( **started** $\wedge$ $\neg$**ready** ). To express impossibility, we simply negate the formula.
- For any state, if a request (of some resource) occurs, then it will eventually be acknowledged: AG ( **requested** $\rightarrow$ AF **acknowledged** ).
- A certain process is enabled infinitely often on every computation path: AG (AF enabled ).
- From any state it is possible to get to a restart state: AG (EF restart ).
- Altri esempi

## Important equivalences between CTL formulas

- We have already noticed that A is a universal quantifier on paths and E is the corresponding existential quantifier. Moreover, G and F are also universal and existential quantifiers, ranging over the states along a particular path.
  - $\neg$ AF $\varphi \equiv$ EG $\neg\varphi$
  - $\neg$ EF $\varphi \equiv$ AG $\neg\varphi$
  - $\neg$ AX $\varphi \equiv$ EX $\neg\varphi$.
- We also have the equivalences AF $\varphi \equiv$ A[ $\top$U $\varphi$ ] EF $\varphi \equiv$ E[ $\top$U $\varphi$ ] which are similar to the corresponding equivalences in LTL.
- Adequate sets of CTL connectives: not all the connectives are necessary

## CTL* and the expressive powers of LTL and CTL

- CTL allows explicit quantification over paths, and in this respect it is more expressive than LTL, as we have seen.
- However, it does not allow one to select a range of paths by describing them with a formula, as LTL does. In that respect, LTL is more expressive. For example, in LTL we can say 'all paths which have a p along them also have a q along them,' by writing $F\ p \rightarrow F\ q$ . It is not possible to write this in CTL because of the constraint that every F has an associated A or E.
- CTL* is a logic which combines the expressive powers of LTL and CTL, by dropping the CTL constraint that every temporal operator (X, U, F, G) has to be associated with a unique path quantifier (A, E).
- Past operators in LTL can be added.

## Model checking algorithm

- We want to solve this problem $M, s \overset{???}{\models} \phi$
- model checking
- by a labelling algorithm:
  - INPUT: a CTL model M = ( S, $\rightarrow$, L ) and a CTL formula φ .
  - OUTPUT: the set of states of M which satisfy φ.

1. First, change φ in terms of the connectives AF, EU, EX, $\wedge$ , $\neg$ and $\perp$ using the equivalences given earlier.
2. Next, label the states of M in which φ holds

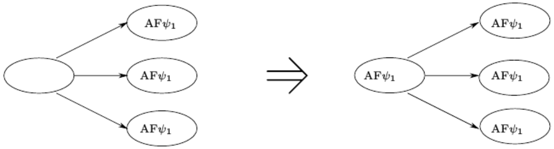
## Labeling algorithm

Case analysis over $\psi$. If $\psi$ is

- $\perp$: then no states are labelled with $\perp$
- p: then label every s such that p $\in L(s)$
- $\psi 1 \wedge \psi 2$:
  - do labelling with $\psi 1$ and with $\psi 2$
  - label s with $\psi 1 \wedge \psi 2$ if s is already labelled both with $\psi 1$ and with $\psi 2$
- $\neg\psi$ :
  - do labelling with $\psi$
  - label s with $\neg\psi 1$ if s is not labelled with $\psi$.

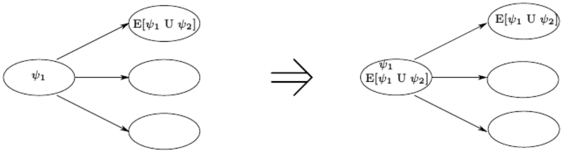
## Labeling algorithm - AF

- AF ψ :
  - do labeling with ψ
  - If any state s is labelled with ψ , label it with AF ψ.
  - Repeat: label any state with AF ψ if all successor states are labelled with AF ψ , until there is no change. See picture



## Labeling algorithm - E U

- E[ψ1 U ψ2]
  - do labeling for ψ1 and ψ2
  - If any state s is labelled with ψ2, label it with E[ψ1 U ψ2]
  - Repeat: label any state with E[ψ1 U ψ2 ] if it is labelled with ψ1 and at least one of its successors is labelled with E[ψ1 U ψ2], until there is no change.


$\mathrm{E}[\psi_1\ \mathrm{U}\ \psi_2]$
$\psi_1$
$\Rightarrow$
$\mathrm{E}[\psi_1\ \mathrm{U}\ \psi_2]$
$\psi_1$
$\mathrm{E}[\psi_1\ \mathrm{U}\ \psi_2]$

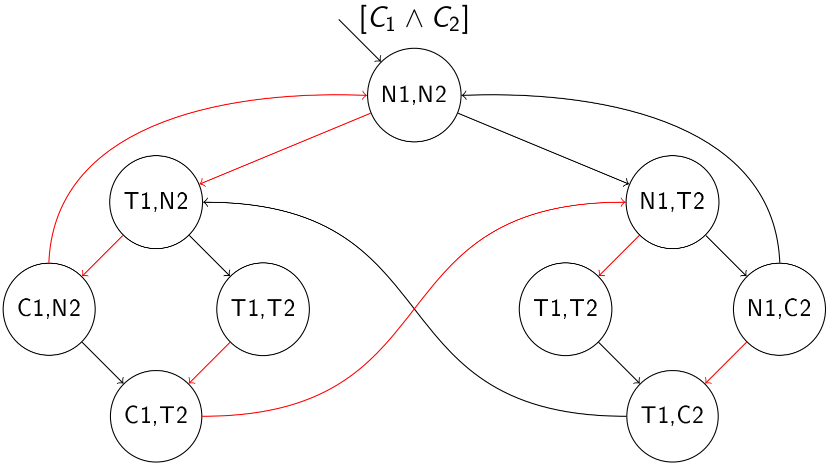
## Labeling algorithm - EX

- EX $\psi$ :
  - do labeling for $\psi$
  - label any state with EX$\psi$ if one of its successors is labelled with $\psi$

## Complessità

- The complexity of this algorithm is $O(f \cdot V \cdot (V + E))$, where f is the number of connectives in the formula, V is the number of states and E is the number of transitions; the algorithm is linear in the size of the formula and quadratic in the size of the model.
- Some improvements
  - Handling EG directly
- LTL is treated differently (skip)

## Esempio



## State Explosion problem

The ‘state explosion’ problem Although the labelling algorithm (with the clever way of handling EG) is linear in the size of the model, unfortunately the size of the model is itself more often than not exponential in the number of variables and the number of components of the system which execute in parallel. This means that, for example, adding a boolean variable to your program will double the complexity of verifying a property of it. The tendency of state spaces to become very large is known as the state explosion problem. A lot of research has gone into finding ways of overcoming it, including the use of:

- efficient data structure BDDs
- ....