Logica temporale

Angelo Gargantini

March 12, 2025

5.1 Motivation

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 ϕ
    - 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.

5.2 Propositional logic

Propositional logic

The aim of logic in computer science is to develop languages to model the situations we encounter as computer science professionals, in such a way that we can reason about them formally. Reasoning about situations means constructing arguments about them; we want to do this formally, so that the arguments are valid and can be defended rigorously, or executed on a machine. Consider the following argument:

Example 1.1
If the train arrives late and there are no taxis at the station, then John is late for his meeting. John is not late for his meeting. The train did arrive late. Therefore, there were taxis at the station.

Intuitively, the argument is valid, since if we put the first sentence and the third sentence together, they tell us that if there are no taxis, then John will be late. The second sentence tells us that he was not late, so it must be the case that there were taxis.

Much of this book will be concerned with arguments that have this structure, namely, that consist of a number of sentences followed by the word 'therefore' and then another sentence. The argument is valid if the sentence after the 'therefore' logically follows from the sentences before it. Exactly what we mean by 'follows from' is the subject of this chapter and the next one. Consider another example:

Example 1.2 If it is raining and Jane does not have her umbrella with her, then she will get wet. Jane is not wet. It is raining. Therefore, Jane has her umbrella with her.

This is also a valid argument. Closer examination reveals that it actually has the same structure as the argument of the previous example! All we have done is substituted some sentence fragments for others:

Propositions

Example 1.1	Example 1.2
the train is late	it is raining
there are taxis at the station	Jane has her umbrella with her
John is late for his meeting	Jane gets wet.

The argument in each example could be stated without talking about trains and rain, as follows:

If p **and not** q, **then** r. **Not** r. p. **Therefore**, q.

In developing logics, we are not concerned with what the sentences really mean, but only in their logical structure. Of course, when we apply such reasoning, as done above, such meaning will be of great interest.

5.3 Declarative sentences

In order to make arguments rigorous, we need to develop a language in which we can express sentences in such a way that brings out their logical structure. The language we begin with is the language of propositional logic. It is based on propositions, or declarative sentences which one can, in principle, argue as being true or false. Examples of declarative sentences are:

1. The sum of the numbers 3 and 5 equals 8.
2. Jane reacted violently to Jack's accusations.
3. Every even natural number >2 is the sum of two prime numbers.
4. All Martians like pepperoni on their pizza.
5. Albert Camus´etait un´ecrivain fran¸cais.
6. Die W¨urde des Menschen ist unantastbar.

These sentences are all declarative, because they are in principle capable of being declared 'true', or 'false'. Sentence (1) can be tested by appealing to basic facts about arithmetic (and by tacitly assuming an Arabic, decimal representation of natural numbers). Sentence (2) is a bit more problematic. In order to give it a truth value, we need to know who Jane and Jack are and perhaps to have a reliable account from someone who witnessed the situation described. In principle, e.g., if we had been at the scene, we feel that we would have been able to detect Jane's violent reaction, provided that it indeed occurred in that way. Sentence (3), known as Goldbach's conjecture, seems straightforward on the face of it. Clearly, a fact about all even numbers >2 is either true or false. But to this day nobody knows whether sentence (3) expresses a truth or not. It is even not clear whether this could be shown by some finite means, even if it were true. However, in this text we will be content with sentences as soon as they can, in principle, attain some truth value regardless of whether this truth value reflects the actual state of affairs suggested by the sentence in question. Sentence (4) seems a bit silly, although we could say that if Martians exist and eat pizza, then all of them will either like pepperoni on it or not. (We have to introduce predicate logic in Chapter 2 to see that this sentence is also declarative if no Martians exist; it is then true.) Again, for the purposes of this text sentence (4) will do. Et alors, qu'est-ce qu'on pense des phrases (5) et (6)? Sentences (5) and (6) are fine if you happen to read French and German a bit. Thus, declarative statements can be made in any natural, or artificial, language.

NOT Declarative sentences

The kind of sentences we won't consider here are non-declarative ones, like

- Could you please pass me the salt?
- Ready, steady, go!

- May fortune come your way.

Primarily, we are interested in precise declarative sentences, or statements about the behaviour of computer systems, or programs. Not only do we want to specify such statements but we also want to check whether a given program, or system, fulfils a specification at hand. Thus, we need to develop a calculus of reasoning which allows us to draw conclusions from given assumptions, like initialised variables, which are reliable in the sense that they preserve truth: if all our assumptions are true, then our conclusion ought to be true as well. A much more difficult question is whether, given any true property of a computer program, we can find an argument in our calculus that has this property as its conclusion. The declarative sentence (3) above might illuminate the problematic aspect of such questions in the context of number theory. The logics we intend to design are symbolic in nature. We translate a certain sufficiently large subset of all English declarative sentences into strings of symbols. This gives us a compressed but still complete encoding of declarative sentences and allows us to concentrate on the mere mechanics of our argumentation. This is important since specifications of systems or software are sequences of such declarative sentences. It further opens up the possibility of automatic manipulation of such specifications, a job that computers just love to do[1].

Atomic sentences

Our strategy is to consider certain declarative sentences as being *atomic*, or *indecomposable*, like the sentence

'The number 5 is even.'

We assign certain distinct symbols p, q, r, . . ., or sometimes p1, p2, p3, . . . to each of these atomic sentences and we can then code up more complex sentences in a compositional way. For example, given the atomic sentences

p: 'I won the lottery last week.'

q: 'I purchased a lottery ticket.'

r: 'I won last week's sweepstakes.'

we can form more complex sentences according to the rules below:

$\neg$ The negation of p is denoted by $\neg$p and expresses 'I did not win the lottery last week,' or equivalently 'It is not true that I won the lottery last week.'

$\vee$ Given p and r we may wish to state that at least one of them is true: 'I won the lottery last week, or I won last week's sweepstakes;' we denote this declarative sentence by p $\vee$ r and call it the disjunction of p and r 2.

[1] 1 There is a certain, slightly bitter, circularity in such endeavours: in proving that a certain computer program P satisfies a given property, we might let some other computer program Q try to find a proof that P satisfies the property; but who guarantees us that Q satisfies the property of producing only correct proofs? We seem to run into an infinite regress.

$\wedge$ Dually, the formula p $\wedge$ r denotes the rather fortunate conjunction of p and r: 'Last week I won the lottery and the sweepstakes.'

$\rightarrow$ Last, but definitely not least, the sentence 'If I won the lottery last week, then I purchased a lottery ticket.' expresses an implication between p and q, suggesting that q is a logical consequence of p. We write p $\rightarrow$ q for that3 . We call p the assumption of p $\rightarrow$ q and q its conclusion.

$\leftrightarrow$ 'If and only if it is Sunday, I'm happy' expresses an equivalence between p and q, suggesting that q is a logical consequence of p and also p is a logical consequence of q. We write p $\longleftrightarrow$ q for that.

Of course, we are entitled to use these rules of constructing propositions repeatedly. For example, we are now in a position to form the proposition p $\wedge$ q $\rightarrow$ $\neg$r $\vee$ q which means that 'if p and q then not r or q'. You might have noticed a potential ambiguity in this reading. One could have argued that this sentence has the structure 'p is the case and if q then . . . ' A computer would require the insertion of brackets, as in (p $\wedge$ q) $\rightarrow$ (($\neg$r) $\vee$ q) to disambiguate this assertion. However, we humans get annoyed by a proliferation of such brackets which is why we adopt certain conventions about the binding priorities of these symbols. Convention 1.3 $\neg$ binds more tightly than $\vee$ and $\wedge$, and the latter two bind more tightly than $\rightarrow$. Implication $\rightarrow$ is right-associative: expressions of the form p $\rightarrow$ q $\rightarrow$ r denote p $\rightarrow$ (q $\rightarrow$ r).

Differenza tra $\rightarrow$ e $\leftrightarrow$

$\longleftrightarrow$ 'If and only if it is Sunday, I'm happy' expresses an equivalence between p and q, suggesting that q is a logical consequence of p and also p is a logical consequence of q. We write p $\longleftrightarrow$ q for that.

Nota che alcune volte usare $\leftrightarrow$ è più completo che usare la semplice implicazione $\rightarrow$.

Ad esempio se ho un metodo isPositive(x) che restituisce vero se x e positivo e false se non lo è, allora la completa carratterizzazione del metodo è $x > 0 \leftrightarrow$ isPositive(x). Se scrivessi solo $x > 0 \rightarrow isPositive(x)$ allora con x negativo il metodo sarebbe libero di ritornare quello che vuole (true o false).

5.4 Linear-time temporal logic

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:
  - come la logica proposizionale (1) (| significa "oppure") - in stile come grammatica BNF

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

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

LTL sintassi - (2) operatori temporali

- Inseriamo operatori temporali (2):

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

- X, F, G, U,W, R sono *connettivi temporali*
  - In particolare: X,F,G sono unari:
    - X means 'neXt state,'
    - F means 'some Future state,' and
    - G means 'all future states (Globally).'
  - The next three, U, R and W sono binari e sono 'Until,' 'Release' and 'Weak-until' respectively.

Alcuni esempi

- FG a -> corretto come F(G(a)): in futuro da un certo punto in poi varrà sempre a
- a U (b /\c) –> a vale fino a quando poi varrà b e c

Precedenza degli operatori

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 $\rightarrow$.

- **Esercizio:** alcuni esempi di LTL con e senza parentesi

Semantica per LTL

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

  A transition system M = (S, s_0,$\rightarrow$, L) is
  - a set of states S endowed
  - a state is the initial state s_0
  - 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 $\rightarrow \mathcal{P}(AP)$

  I transition system sono i nostri *modelli*.

Labelling function

- a labelling function L : S $\rightarrow \mathcal{P}(AP)$

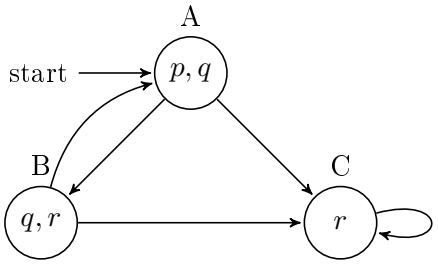
  $\mathcal{P}(AP)$ è il powerset – l'insieme delle parti – di proposizioni atomiche (**AP**)

  - 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 A, B, and C. The atomic propositions AP = {p,q,r}. The only possible transitions are A $\rightarrow$ B, A $\rightarrow$ C, B $\rightarrow$ A, B $\rightarrow$ C and C $\rightarrow$ C; and if L(A) = {p, q}, L(B) = {q, r} and L(C) = {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, $\rightarrow$, L) is an infinite sequence of states s_1, s_2 , s_3 , . . . in S such that, for each i $\geq$ 1, $s_i \rightarrow s_{i+1}$.
  - We write the path as $s_1 \rightarrow s_2 \rightarrow \ldots$.
  - We write π^i for the suffix starting at s_i, e.g., π^3 is $s_3 \rightarrow s_4 \rightarrow \ldots$.
  - Esempio

Esempio

- A -> B ->A -> B -> C ...
  - altri esempi

Validità di una formula LTL su un path (prop)

Definition 1. Let M = (S, $\rightarrow$, L) be a model and $\pi = s_1 \rightarrow \ldots$ be a **path** in M. Whether π 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. $\pi \models \varphi 1 \wedge \varphi 2$ iff $\pi \models \varphi 1$ and $\pi \models \varphi 2$
6. $\pi \models \varphi 1 \vee \varphi 2$ iff $\pi \models \varphi 1$ or $\pi \models \varphi 2$
7. $\pi \models \varphi 1 \rightarrow \varphi 2$ iff $\pi \models \varphi 2$ whenever $\pi \models \varphi 1$

Validità di una formula LTL su un path (time)

Definition 2. Let M = (S, $\rightarrow$, L) be a model and $\pi = s_1 \rightarrow \ldots$ be a path in M. Whether π satisfies an LTL formula is defined by the satisfaction relation $\models$ as follows:

8. $\pi \models$ X φ iff $\pi^2 \models \varphi$
9. $\pi \models$ G φ iff, for all i ≥ 1, $\pi^i \models \varphi$
10. $\pi \models$ F φ iff there is some i ≥ 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 ≥ 1 such that $\pi^i \models$ b and for all j = 1, . . . , i $-$ 1 we have $\pi^j \models$ a
12. **(Weak Until)** $\pi \models$ a W b iff either there is some i ≥ 1 such that $\pi^i \models$ b and for all j = 1, . . . , i $-$ 1 we have $\pi^j \models$ a; or for all k ≥ 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)

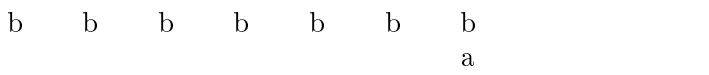
13. **(Release)** π⊨a R b iff either there is some i ≥ 1 such that $\pi^i \models$ a and for all j = 1, . . . , i we have $\pi^j \models$b, or for all k ≥ 1 we have $\pi^k \models$b.

- 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 ¬(¬a U ¬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 3. Suppose M =(S, → ,L) is a model, s ∈ S ,and φ an LTL formula. We write M ,s ⊨φ if, for every execution path π of M starting at s, we have ⊨

  Example 4. Figura 3.3 e figura 3.5, alcune formule

Formula valida con stato iniziale

- Se la macchina M ha uno stato iniziale s_0
  - Quando una formula è **valida** per una macchina M (e non solo per un path da uno stato) ?

  Definition 5. Suppose M =(S, s_0,→ ,L) is a model, s_0 ∈ S lo stato iniziale, and φ an LTL formula. We write $M \models \varphi$ if, for every execution path π of M starting at s_0, we have ⊨

Formula valida con stati iniziali

- Se la macchina M ha un insieme di stati iniziali S_0
  - Quando una formula è **valida** per una macchina M (e non solo per un path) ?

  Definition 6. Suppose M =(S, S_0,→ ,L) is a model, $S_0 \subseteq S$ gli stati iniziali, and φ an LTL formula. We write $M \models \varphi$ if per ogni $s_0 \in S_0$ vale $M, s_0 \models \varphi$

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 più complessi - 3.2

Important equivalences between LTL formulas

We say that two LTL formulas φ and ψ are semantically equivalent, or simply equivalent, writing $\varphi \equiv \psi$, if for all models M and all paths π 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).

- φ W $\psi \equiv (\varphi$ U $\psi) \vee$ G φ

  Both U and R can be defined in terms of the weak until:
  - Until and Weak until: φ U $\psi \equiv \varphi$ W $\psi \wedge$ F ψ

  Also R can be defined in terms of W
  - φ W $\psi \equiv (\varphi$ U $\psi) \vee$ G $\varphi \equiv \varphi$ U $(\psi \vee$ G $\varphi) \equiv \psi$ R $(\psi \vee \varphi)$ φ U $\psi \equiv$ F$\psi \wedge (\varphi$ W $\psi)$ φ R $\psi \equiv \psi$ W $(\psi \wedge \varphi)$

F and G duality

- F and G are duals:
  - $\neg$ G $\varphi \equiv$ F $\neg \varphi$ $\qquad\qquad$ $\neg$ 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:
    - $\neg$ (φ U ψ) $\equiv\neg \varphi$ R $\neg \psi$ $\qquad\qquad$ $\neg$ (φ R ψ) $\equiv\neg \varphi$ U $\neg \psi$

Distributive

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

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

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

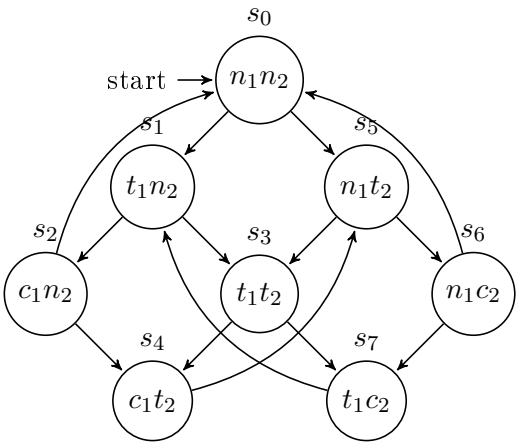
Safety Only one process is in its critical section at any time.

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 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

Limiti LTL

Ricorda la definizione:

Definition 7. Suppose M is a model, s $\in$ S ,and φ an LTL formula. We write M,s $\models\varphi$ if, for **every** execution path π of M starting at s, we have $\models$

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

5.5 Branching-time temporal logic

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

ppp

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 A, 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 8. Let M = (S, $\rightarrow$, 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 ($\wedge$, $\vee$, $\neg$, 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. $M, s \models \neg\varphi$ iff π $M, s \not\models \varphi$
5. $M, s \models \varphi 1 \wedge \varphi 2$ iff $M, s \models \varphi 1$ and $M, s \models \varphi 2$
6. $M, s \models \varphi 1 \vee \varphi 2$ iff $M, s \models \varphi 1$ or $M, s \models \varphi 2$
7. $M, s \models \varphi 1 \rightarrow \varphi 2$ iff $M, s \models \varphi 2$ whenever $M, s \models \varphi 1$

Validità di una formula CTL (time)

8. $M, s \models$AX φ iff forall $s_1$such that $s \rightarrow s_1$ we have $M, s_1 \models \varphi$
9. $M, s \models$EX φ iff some $s_1$such that $s \rightarrow s_1$ we have $M, s_1 \models \varphi$
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 $M, s \models \varphi$
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 $M, s \models \varphi$

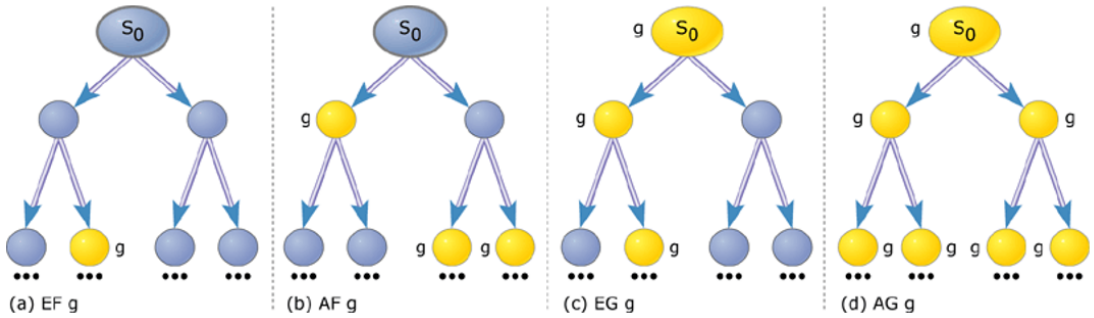
- AX: 'in every next state.'
  - EX: 'in some next state.'
  - AG: for All computation paths beginning in s the property φ holds Globally
  - EG: there Exists a path beginning in s such that φ holds Globally along the path.

Validità di una formula CTL (time 2)

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

- 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



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

Formula valida con stato iniziale

- Se la macchina M ha un insieme di stati iniziali S_0
  - Quando una formula è **valida** per una macchina M (e non solo per un path) ?

Definition 9. Suppose M =(S, S_0,→ ,L) is a model, $S_0 \subseteq S$ gli stati iniziali, and φ an CTL formula. We write $M \models \varphi$ if per ogni $s_0 \in S_0$ vale $M, s_0 \models \varphi$

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

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.
- We can derive the following equivalences:
  - $\neg$ AF $\varphi \equiv$ EG $\neg\varphi$ and EG $\varphi \equiv \neg$ AF $\neg\varphi$
  - $\neg$ EF $\varphi \equiv$ AG $\neg\varphi$ and AG $\varphi \equiv \neg$ EF $\neg\varphi$
  - $\neg$ AX $\varphi \equiv$ EX $\neg\varphi$.
  - We also have the equivalences AF $\varphi \equiv$ A[$\top$U φ] and
    - EF $\varphi \equiv$ E[$\top$U φ] which are similar to the corresponding equivalences in LTL.
  - Adequate sets of CTL connectives: not all the connectives are necessary.
  - We could (and will) use only AF, EU, EX

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.