

Verifica Automatica di Sistemi

Colognese, Rossini

Giugno 2018

Transition System	2
Program Graph (PG)	2
Parallelismo e Comunicazione	3
Linear Time Properties	4
Safety	4
Liveness	4
Fairness	4
Linear Temporal Logic	5
Model Checking	6
Model Checking	6
Computation Tree Logic	7
Computation Tree	7
Equivalenza tra formule LTL e CTL	8
CTL*	9

Transition System

Dato un sistema reale, esegue un'astrazione modellandone: gli stati, le transizioni tra essi e informazioni aggiuntive (comunicazione e proprietà degli stati).

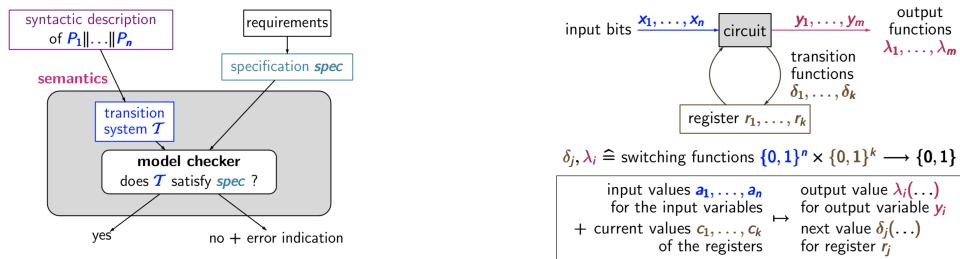
Il **Transition System** (TS) è una tupla: $\mathcal{T} = (S, Act, \rightarrow, S_0, AP, L)$, dove:

- S è l'insieme degli stati (con $S_0 \in S$);
- Act è l'insieme delle azioni;
- $\rightarrow \subseteq S \times Act \times S$ è la relazione di transizione;
- AP è l'insieme delle proposizioni atomiche;
- $L : S \rightarrow 2^{AP}$ la funzione di etichettamento.

Esecuzioni: sequenze massimali di transizioni. $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots$

Effect-function: $Act \times Eval(Var) \rightarrow Eval(Var)$

Reach(\mathcal{T}) = insieme di tutti gli stati raggiungibili da uno stato iniziale attraverso qualche esecuzione.



Program Graph (PG)

Un *Program Graph* su un'insieme di variabili tipate (Var) è una tupla: $\mathcal{P} = (Loc, Act, Effect, \hookrightarrow, Loc_0, g_0)$, dove:

- Loc è l'insieme delle locazioni (con $L_0 \in Loc$);
- Act è l'insieme delle azioni;
- $Effect: Act \times Eval(Var) \rightarrow Eval(Var)$;
- $\hookrightarrow: Loc \times Cond(Var) \times Act \times Loc$;
- $g_0 \in Cond(Var)$: condizione iniziale sulle variabili.

Il TS corrispondente ad un PG ha gli stati della forma: $\langle l, \eta \rangle$, con l locazione corrente e η la variabile valutata.

- *state space:* $S = Loc \times Eval(Var)$
- *initial state:* $S_0 = \{\langle l, \eta \rangle : l \in Loc_0, \eta \models g_0\}$
- \rightarrow è così definita: if $l \hookrightarrow^{g:\alpha} l' \wedge \eta \models g$ then $\langle l, \eta \rangle \rightarrow^\alpha \langle l', Effect(\alpha, \eta) \rangle$
- $AP = Loc \cup Cond(Var)$
- *labeling function:* $L(\langle l, \eta \rangle) = \{l\} \cup \{g \in Cond(Var) : \eta \models g\}$

Parallelismo e Comunicazione

Operatore di Interleaving ($\parallel\!\parallel$): utilizzato quando le azioni sono indipendenti (non sfruttano le stesse risorse).
Proprietà del diamante: $\text{Effect}(\alpha \parallel\!\parallel \beta) = \text{Effect}(\alpha; \beta + \beta; \alpha)$

$$\begin{aligned}\mathcal{T}_1 \parallel\!\parallel \mathcal{T}_2 &= \\ &= \{S_1 \times S_2, Act_1 \cup Act_2, \longrightarrow, S_{0,1} \times S_{0,2}, AP, L\} \\ \text{Atomic proposition: } AP_1 \uplus AP_2 \\ \text{Labeling function: } L(\langle s_1, s_2 \rangle) &= L_1(s_1) \cup L_2(s_2)\end{aligned}$$

where the transition relation \longrightarrow is given by:

$$\frac{s_1 \xrightarrow{\alpha_1} s'_1}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s'_1, s_2 \rangle} \quad \frac{s_2 \xrightarrow{\alpha_2} s'_2}{\langle s_1, s_2 \rangle \xrightarrow{\alpha} \langle s_1, s'_2 \rangle}$$

L'operatore di *Interleaving* fallisce per azioni dipendenti, portando a stati inconsistenti.
Per questo motivo viene introdotta la **Mutua Esclusione** attraverso i semafori.

Linear Time Properties

Execution fragment: seq. di transizioni infinita ($s_0 \rightarrow^{\alpha_0} s_1 \rightarrow^{\alpha_1} \dots$) o finita ($s_0 \rightarrow^{\alpha_0} s_1 \rightarrow^{\alpha_1} \dots \rightarrow^{\alpha_{n-1}} s_n$).

Path fragment: sequenza di stati infinita ($\pi = s_0 s_1 s_2 \dots$) o finita ($\pi = s_0 s_1 s_2 \dots s_n$).

Queste possono essere *massimali* se sono infinite oppure finiscono in uno stato terminale.

Paths(\mathcal{T}) = insieme dei percorsi massimali, partendo dallo stato iniziale.

Paths(s) = insieme dei percorsi massimali, partendo dallo stato s .

Paths_{fin}(s) = insieme dei percorsi finiti, partendo dallo stato s .

Traces = sequenza di insiemi di proposizioni atomiche (assumendo che non ci siano stati terminali).

Traces(\mathcal{T}) = { trace(π) : $\pi \in \text{Paths}(\mathcal{T})$ } $\subseteq (2^{AP})^\omega$

Traces_{fin}(\mathcal{T}) = { trace($\hat{\pi}$) : $\hat{\pi} \in \text{Paths}_{\text{fin}}(\mathcal{T})$ } $\subseteq (2^{AP})^*$

Safety

LT Property (senza stati terminali): è un linguaggio E di infinite parole su un alfabeto $\Sigma = 2^{AP}$ con $E \subseteq (2^{AP})^\omega$

$\mathcal{T} \models E$ iff $\text{Traces}(\mathcal{T}) \subseteq E$ $s \models E$ iff $\text{Traces}(s) \subseteq E$

Safety (Mutex): su un'insieme infinito di parole, si vogliono evitare situazioni di inconsistenza e deadlock. Abbiamo degli *invarianti* che definiscono le proprietà da non violare. Non ci devono essere *bad prefix* (pay, drink, drink).

prefix(σ) = insieme di prefissi di σ non vuoti.

prefix(E) = $\bigcup_{\sigma \in E} \text{pref}(\sigma)$.

Prefix closure: sono i prefissi che sono accettati dal linguaggio. $cl(E) = \{\sigma \in (2^{AP})^\omega : \text{pref}(\sigma) \subseteq \text{pref}(E)\}$.

Teorema: E è una safety property se e solo se $cl(E) = E$.

Liveness

Liveness (Live): si vogliono evitare situazioni di starvation (da wait possono sempre andare in crit).

E è chiamata **liveness property** se ogni parola finita può essere estesa ad una parola infinita in E . $\text{pref}(E) = (2^{AP})^\omega$

Fairness

Dato $\rho = s_0 \rightarrow^{a_0} s_1 \rightarrow^{a_1} s_2 \rightarrow^{a_2} \dots$, un frammento infinito di esecuzione:

Unconditional Fairness: $\exists^{\infty} i \geq 0. \alpha_i \in A$

ρ è *unconditional A-fair* se l'azione A viene eseguita infinitamente spesso.

Ogni processo ottiene il suo turno infinitamente spesso.

Strong Fairness: $\exists^{\infty} i \geq 0. A \cap \text{Act}(s_i) \neq \emptyset \implies \exists^{\infty} i \geq 0. \alpha_i \in A$

Ogni processo che è abilitato infinitamente spesso, ottiene il suo turno infinitamente spesso.

ρ non è *strongly A-fair* se:

- nessuna azione A viene eseguita da un certo momento;
- l'azione A è abilitata infinite volte.

Weak Fairness: $\forall^{\infty} i \geq 0. A \cap \text{Act}(s_i) \neq \emptyset \implies \exists^{\infty} i \geq 0. \alpha_i \in A$

Ogni processo che è continuamente abilitato, da un certo momento in poi, ottiene il suo turno infinitamente spesso.

ρ non è *weak A-fair* se:

- nessuna azione A viene eseguita da un certo momento;
- l'azione A è continuamente abilitata da un certo punto in poi.

Unconditionally A-fair \Rightarrow Strongly A-fair \Rightarrow Weakly A-fair

Linear Temporal Logic

Next operator: $\bigcirc a$ significa che vale a solo nell'istante successivo. Self Duality: $\bigcirc \neg \phi = \neg \bigcirc \phi$

Until operator: $a \bigcup b$ significa che continua a valere a fino ad un certo punto in cui varrà b e non più a .

$$\varphi ::= \text{true} \mid a \mid \varphi_1 \wedge \varphi_2 \mid \neg \varphi \mid \bigcirc \varphi \mid \varphi_1 \mathbf{U} \varphi_2$$

Diamond operator: $\lozenge \phi = \text{true} \bigcup \phi$ significa ad un certo punto varrà ϕ almeno una volta.

Box operator: $\square \phi = \neg \lozenge \neg \phi$ significa che varrà sempre ϕ

Weak until: $\phi \mathcal{W} \psi \equiv (\phi \bigcup \psi) \vee \square \phi$

Spesso infinite volte: $\square \lozenge \phi$

Prima o poi sempre: $\lozenge \square \phi$

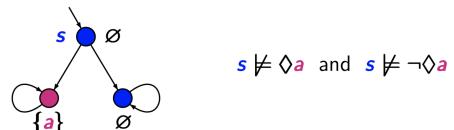
Words(ϕ) = $\{\sigma \in (2^{AP})^\omega : \sigma \models \phi\}$

Expansion Law: $\psi_1 \bigcup \psi_2 \equiv \psi_2 \vee (\psi_1 \wedge \bigcirc(\psi_1 \bigcup \psi_2))$

$$\begin{aligned} \sigma \models \text{true} & \\ \sigma \models a & \text{ iff } A_0 \models a, \text{i.e., } a \in A_0 \\ \sigma \models \varphi_1 \wedge \varphi_2 & \text{ iff } \sigma \models \varphi_1 \text{ and } \sigma \models \varphi_2 \\ \sigma \models \neg \varphi & \text{ iff } \sigma \not\models \varphi \\ \sigma \models \bigcirc \varphi & \text{ iff } \text{suffix}(\sigma, 1) = A_1 A_2 A_3 \dots \models \varphi \\ \pi = s_0 s_1 s_2 \dots \models \varphi & \text{ iff } \text{trace}(\pi) \models \varphi \\ & \text{ iff } \text{trace}(\pi) \in \text{Words}(\varphi) \end{aligned}$$

$$\begin{aligned} \sigma \models \varphi_1 \mathbf{U} \varphi_2 & \text{ iff there exists } j \geq 0 \text{ such that} \\ & A_j A_{j+1} A_{j+2} \dots \models \varphi_2 \text{ and} \\ & A_i A_{i+1} A_{i+2} \dots \models \varphi_1 \text{ for } 0 \leq i < j \\ \sigma \models \lozenge \varphi & \text{ iff there exists } j \geq 0 \text{ such that} \\ & A_j A_{j+1} A_{j+2} \dots \models \varphi \\ \sigma \models \square \varphi & \text{ iff for all } j \geq 0 \text{ we have:} \\ & A_j A_{j+1} A_{j+2} \dots \models \varphi \end{aligned}$$

$$\begin{aligned} T \models a & \text{ as } s_0 \models a \text{ and } s_2 \models a \\ T \not\models \lozenge a & \text{ as } s_0 s_1 s_2 \dots \not\models \lozenge a \\ T \models \lozenge \square b \vee \square \lozenge(\neg a \wedge \neg b) & \text{ as } s_2 \models b, s_1 \not\models a, b \\ T \models \square(a \rightarrow (\bigcirc \neg a \vee b)) & \text{ as } s_2 \models b, s_0 \models \bigcirc \neg a \end{aligned}$$



$$\begin{aligned} \lozenge(\varphi \vee \psi) &\equiv \lozenge \varphi \vee \lozenge \psi \\ \text{correct} \\ \hline \lozenge(\varphi \wedge \psi) &\equiv \lozenge \varphi \wedge \lozenge \psi \\ \text{wrong, e.g.,} & \begin{array}{c} \text{green circle} \\ \{b\} \end{array} \xrightarrow{\quad} \begin{array}{c} \text{blue circle} \\ \{a\} \end{array} \models \lozenge b \wedge \lozenge a \not\models \lozenge(b \wedge a) \\ \text{similarly: } \square(\varphi \wedge \psi) &\equiv \square \varphi \wedge \square \psi \\ \square(\varphi \vee \psi) &\not\equiv \square \varphi \vee \square \psi \end{aligned}$$

$$\begin{aligned} \text{until: } \varphi \mathbf{U} \psi &\equiv \psi \vee (\varphi \wedge \bigcirc(\varphi \mathbf{U} \psi)) \\ \text{eventually: } \lozenge \psi &\equiv \psi \vee \bigcirc \lozenge \psi \\ \text{always: } \square \psi &\equiv \psi \wedge \bigcirc \square \psi \end{aligned}$$

Positive Normal Form (PNF): proprietà per eliminare il *not* da una formula, trasformandola.

$$\begin{aligned} \neg \lozenge \phi &\text{ diventa } \square \neg \phi \\ \neg \square \phi &\text{ diventa } \lozenge \neg \phi \\ \neg \bigcirc \phi &\text{ diventa } \bigcirc \neg \phi \end{aligned}$$

$$\begin{aligned} \text{unconditional fairness } \square \lozenge \text{crit}; \\ \text{strong fairness } \square \lozenge \text{wait}; \rightarrow \square \lozenge \text{crit}; \\ \text{weak fairness } \lozenge \square \text{wait}; \rightarrow \square \lozenge \text{crit}; \end{aligned}$$

Model Checking

L'idea di base consiste nel confutare $\mathcal{T} \models \phi$ cercando un path π in \mathcal{T} tale che $\pi \models \neg\phi$

NBA (Non-Determinist Büchi Automata) $\mathcal{A} = (Q, \Sigma, \delta Q_0, F)$

- Q insieme finito di stati;
- Σ alfabeto;
- $\delta : Q \times \Sigma \rightarrow 2^Q$ relazione di transizione;
- $Q_0 \subseteq Q$ insieme di stati iniziali;
- $F \subseteq Q$ insieme di stati finali, anche chiamati stati accettanti.

Dato $\pi = q_0 q_1 q_2 \dots$ dove $q_0 \in Q_0$ e $q_{i+1} \in \delta(q_i, A_i)$ per $i \geq 0$, un path π è accettante se $\exists i \in \mathbb{N}. q_i \in F$. $\mathcal{L}_\omega(\mathcal{A})$ = insieme di parole infinite su Σ che hanno un path/run accettante in \mathcal{A} .

GNBA (Generalized Non-Determinist Büchi Automata) $\mathcal{G} = (Q, \Sigma, \delta, Q_0, \mathcal{F})$

- Q, Σ, δ, Q_0 sono come in NBA;
- $\mathcal{F} \subseteq 2^Q$ è un set di set accettanti, tali che $\mathcal{F} = \{F_{\phi_1} \cup F_{\phi_2} : \phi_1 \cup \phi_2 \in cl(\phi)\}$.

Una run $\pi = q_0 q_1 q_2 \dots$ è accettante se ogni set accettante è visitato infinitamente spesso. $\forall F \in \mathcal{F} \quad \exists i \in \mathbb{N}. q_i \in F$. $\mathcal{L}_\omega(\mathcal{G}) = \{\sigma \in \Sigma^\omega : \sigma \text{ ha una run accettante in } \mathcal{G}\}$

Per ogni GNBA \mathcal{G} esiste un NBA \mathcal{A} corrispondente con $\mathcal{L}_\omega(\mathcal{G}) = \mathcal{L}_\omega(\mathcal{A})$.

Per ogni formula LTL ϕ su AP c'è un NBA \mathcal{A} su un alfabeto 2^{AP} tale che:

- $Words(\phi) = \mathcal{L}_\omega(\mathcal{A})$
- $\text{size}(\mathcal{A}) = \mathcal{O}(\exp(|\phi|))$

Persistence checking: $\mathcal{T} \otimes \mathcal{A} \models \Diamond \Box \neg F$

From LTL to GNBA: si passa dalle $A_0 A_1 \dots \in Words(\phi)$ a $B_0 B_1 \dots$ run accettanti, con $B_i = \{\psi \in cl(\phi) : A_i A_{i+1} \dots \models \psi\}$. Sappiamo che $cl(\phi)$ sono le sottoformule di ϕ e i loro negati. Sia $B \subseteq cl(\phi)$. B è elementare se:

- B è consistente, cioè se $\psi \in B$ allora $\neg\psi \notin B$
- B è massimamente consistente, cioè se $\psi \in cl(\phi) \setminus B$ allora $\neg\psi \in B$
- B è localmente consistente rispetto all'until \bigcup , cioè se $\psi_1 \bigcup \psi_2 \in B$ e $\neg\psi_2 \in B$ allora $\neg\psi_1 \notin B$

Fairness in LTL e in CTL

LTL:

- *Unconditional fairness:* $\Box \Diamond \phi$
- *Strong fairness:* $\Box \Diamond \psi \rightarrow \Box \Diamond \phi$
- *Weak fairness:* $\Diamond \Box \psi \rightarrow \Box \Diamond \phi$

CTL:

- *Unconditional fairness:* $\forall \Box \forall \Diamond \phi$
- *Strong fairness:* $\forall \Box \forall \Diamond \psi \rightarrow \forall \Box \forall \Diamond \phi$
- *Weak fairness:* non c'è in CTL (dim. sotto).

Computation Tree Logic

$\forall \bigcirc \phi$ = per ogni stato figlio vale ϕ ;
 $\exists \bigcirc \phi$ = esiste un figlio in cui vale ϕ .
 $\forall \square \phi$ = per ogni computazione (branch) e per ogni stato della computazione vale ϕ .
 $\forall \Diamond \phi$ = per ogni computazione prima o poi vale ϕ .
 $\exists \Box \phi$ = esiste una computazione in cui, per ogni stato, vale ϕ .
 $\exists \Diamond \phi$ = esiste una computazione in cui vale in un solo stato ϕ .

Il quantificatore di esistenza (\exists) può essere ricavato attraverso delle operazioni sul quantificatore \forall :

$$\begin{aligned}\exists \Box A &:= \neg \forall \Diamond \neg A \\ \exists \Diamond A &:= \neg \forall \Box \neg A \\ \exists \bigcirc A &:= \neg \forall \bigcirc \neg A\end{aligned}$$

Un (*UB-*)frame $\langle S, N \rangle$ è un grafo dove $N \subseteq S \times S$ è totale ($\forall s \exists s' \ sN s'$).

Un (*UB-*)model $\langle F, V \rangle$ è una coppia dove F è un frame e $V : s \rightarrow 2^{prop}$ è una valutazione.

Let $M = \langle S, N, V \rangle$ a model,
the satisfiability relation $M \models \subseteq S \times WFF$

is defined as

1. $M, s \not\models \perp$
2. $M, s \models p$ iff $p \in V(s)$
3. $M, s \models A \wedge B \Leftrightarrow M, s \models A \ \& \ M, s \models B$
4. $M, s \models A \vee B \Leftrightarrow M, s \models A \text{ OR } M, s \models B$
5. $M, s \models \neg A \Leftrightarrow M, s \not\models A$,
6. $M, s \models A \rightarrow B \Leftrightarrow (M, s \models A \Rightarrow M, s \models B)$,
7. $M, s \models \forall \bigcirc A \Leftrightarrow \forall b_s \forall s' \in b_s \ M, s' \models A$
8. $M, s \models \forall \Diamond A \Leftrightarrow \forall b_s \exists s' \in b_s \ M, s' \models A$
9. $M, s \models \exists \Box A \Leftrightarrow \exists b_s \forall s' \in b_s \ M, s' \models A$
10. $M, s \models \exists \Diamond A \Leftrightarrow \exists b_s \exists s' \in b_s \ M, s' \models A$
11. $M, s \models \forall \bigcirc A \Leftrightarrow \forall s' (sNs' \Rightarrow M, s' \models A)$
12. $M, s \models \exists \bigcirc A \Leftrightarrow \exists s' (sNs' \ \& \ M, s' \models A)$

Con $s' \in b_s$ si intende un figlio di s che appartiene ad un suo branch.

Alla semantica *CTL* si aggiunge l'operatore *Until* \bigcup .

$$\begin{aligned}M, s \models B \exists \bigcup A &\Leftrightarrow \exists b_s \exists k(M, b_s[k] \models A \ \& \ \forall j \in [0, k-1] \ b_s[j] \models B) \\ M, s \models B \forall \bigcup A &\Leftrightarrow \forall b_s \exists k(M, b_s[k] \models A \ \& \ \forall j \in [0, k-1] \ b_s[j] \models B)\end{aligned}$$

$$\begin{aligned}\exists \Diamond \alpha &\equiv \text{true } \exists \bigcup A \\ \forall \Diamond \alpha &\equiv \text{true } \forall \bigcup A\end{aligned}$$

Computation Tree

Per costruire il *computation tree* di un transition system, è necessario farne l'*unfolding*. Se ci sono più stati iniziali, si rappresenta un albero per ciascuno stato s_0 .

$\exists \neg \Diamond \neg \phi$ non è una formula CTL (non c'è \neg nelle path).
 $\forall \Box \forall \Diamond \text{ crit}_1 \wedge \forall \Box \forall \Diamond \text{ crit}_2$ (per ogni branch, per ogni stato prima o poi vale crit_i ; infinitamente spesso).

Path Formulas

Let $\pi = s_0 s_1 s_2 \dots$ be an infinite path fragment.

$\pi \models \bigcirc \Phi$	iff $s_1 \models \Phi$
$\pi \models \Phi_1 \bigcup \Phi_2$	iff there exists $j \geq 0$ such that
	$s_j \models \Phi_2$
	$s_k \models \Phi_1$ for $0 \leq k < j$

semantics of derived operators:

$\pi \models \Diamond \Phi$	iff there exists $j \geq 0$ with $s_j \models \Phi$
$\pi \models \Box \Phi$	iff for all $j \geq 0$ we have: $s_j \models \Phi$

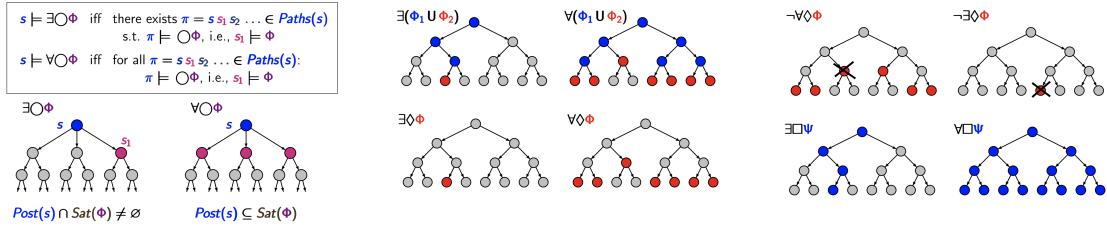
State Formulas

$s \models \text{true}$	
$s \models a$	iff $a \in L(s)$
$s \models \Phi_1 \wedge \Phi_2$	iff $s \models \Phi_1$ and $s \models \Phi_2$
$s \models \neg \Phi$	iff $s \not\models \Phi$
$s \models \exists \varphi$	iff there is a path $\pi \in \text{Paths}(s)$ s.t. $\pi \models \varphi$
$s \models \forall \varphi$	iff for each path $\pi \in \text{Paths}(s)$: $\pi \models \varphi$

satisfaction set for state formula Φ :

$$\text{Sat}(\Phi) \stackrel{\text{def}}{=} \{s \in S : s \models \Phi\}$$

$$\mathcal{T} \models \phi \Leftrightarrow S_0 \subseteq Sat(\phi) \Leftrightarrow s_0 \models \phi \forall s_0 \in \mathcal{T} \text{ con } Sat(\phi) = \{s \in S : s \models \phi\}$$



Se $\mathcal{T} \not\models \neg\phi$, allora non è necessario che $\mathcal{T} \models \phi$. Ci sono più stati iniziali (la regola deve essere verificata $\forall s_0$).

Equivalenza tra formule CTL

$\phi_1 \equiv \phi_2$ sse, per ogni TS $\mathcal{T} : \mathcal{T} \models \phi_1 \Leftrightarrow \mathcal{T} \models \phi_2$
e vale $Sat(\phi_1) = Sat(\phi_2)$.

$$\begin{aligned}\exists \Diamond(a \wedge b) &\neq \exists \Diamond a \wedge \exists \Diamond b \\ \forall \Diamond(a \wedge b) &\neq \forall \Diamond a \wedge \forall \Diamond b \\ \forall \Box(\phi_1 \wedge \phi_2) &\equiv \forall \Box \phi_1 \wedge \forall \Box \phi_2 \\ \exists \Diamond(\phi_1 \vee \phi_2) &\equiv \exists \Diamond \phi_1 \vee \exists \Diamond \phi_2\end{aligned}$$

Duality laws.

$$\begin{aligned}\forall \Box \phi &\equiv \neg \exists \Diamond \neg \phi \\ \forall \Diamond \phi &\equiv \neg \exists \Box \neg \phi \\ \forall \bigcirc \phi &\equiv \neg \exists \bigcirc \neg \phi \\ \exists \bigcirc \phi &\equiv \neg \forall \bigcirc \neg \phi \\ \forall (\phi \bigcup \psi) &\equiv \neg \exists ((\neg \psi) \bigcup (\neg \phi \wedge \neg \psi)) \wedge \neg \exists \Box \neg \psi \\ \forall \bigcup \text{ e } \forall \bigcirc &\text{ sono esprimibili attraverso } \exists \bigcup, \exists \bigcirc \text{ e } \exists \Box\end{aligned}$$

Equivalenza tra formule LTL e CTL

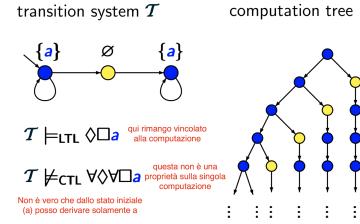
Sia ϕ una formula CTL e ψ una formula LTL: $\phi \equiv \psi$ sse $\forall \mathcal{T} \text{ e } \forall s \in \mathcal{T} : s \models_{CTL} \phi \Leftrightarrow s \models_{LTL} \psi$

CTL formula Φ	LTL formula φ
a	a
$\forall O a$	$O a$
$\forall(a \bigcup b)$	$a \bigcup b$
$\forall \Box a$	$\Box a$
$\forall \Diamond a$ dimostrata su quaderno	$\Diamond a$
$\forall(a W b)$	$a W b$ II WeakUntil non lo abbiamo fatto
$\forall \Box \forall \Diamond a$	$\Box \Diamond a$ but: $\forall \Diamond \forall \Box a \not\equiv \Diamond \Box a$ proprietà di persistenza (weak fairness)
infinitely often a	

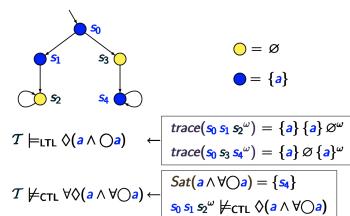
Dimostrazione: in CTL non vale weak fairness

$$\Diamond \Box a \neq \forall \Diamond \forall \Box a$$

Nessuna formula LTL è equivalente a $\forall \Diamond \forall \Box a$



$$\Diamond(a \wedge \bigcirc a) \neq \forall \Diamond(a \wedge \forall \bigcirc a)$$



The CTL formulas

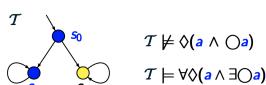
$$\begin{aligned}\forall \Diamond(a \wedge \forall O a) \\ \forall \Diamond \Box a \\ \forall \Box \Diamond a\end{aligned}$$

have no equivalent LTL formula



Does $\forall \Diamond(a \wedge \exists O a) \equiv \Diamond(a \wedge \bigcirc a)$ hold?

answer: no.



$$\begin{aligned}Sat(\exists O a) &= \{s_0, s_1\} \\ Sat(\forall \Diamond(a \wedge \exists O a)) &= \{s_0, s_1\}\end{aligned}$$

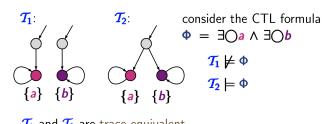
If Φ is CTL formula and φ an LTL formula such that $\Phi \equiv \varphi$ then $\neg \Phi \equiv \neg \varphi$

Se \mathcal{T}_1 e \mathcal{T}_2 sono trace equivalent NON è vero che $\mathcal{T}_1 \models \phi$ sse $\mathcal{T}_2 \models \phi$.

wrong. E.g.,

$$\Phi = \forall \Box \forall \Diamond a, \quad \varphi = \Box \Diamond a$$

- $\Phi \equiv \varphi$
- there is no CTL formula that is equivalent to $\neg \varphi \equiv \Diamond \Box \neg a$



CTL*

state formulas:
 $\Phi ::= \text{true} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg \Phi \mid \exists \varphi$

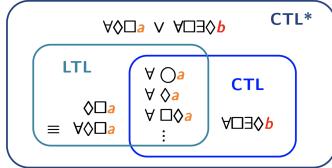
path formulas:
 $\varphi ::= \Phi \mid \varphi_1 \wedge \varphi_2 \mid \neg \varphi \mid \bigcirc \varphi \mid \varphi_1 \mathsf{U} \varphi_2$

- $s \models true$
- $s \models a \quad \text{iff} \quad a \in L(s)$
- $s \models \neg\Phi \quad \text{iff} \quad s \not\models \Phi$
- $s \models \Phi_1 \wedge \Phi_2 \quad \text{iff} \quad s \models \Phi_1 \text{ and } s \models \Phi_2$
- $s \models \exists \varphi \quad \text{iff} \quad \begin{array}{l} \text{there exists a path } \pi \in Paths(s) \\ \text{such that } \pi \models \varphi \end{array}$

$\pi \models \Phi$	iff	$s_0 \models \Phi \leftarrow$ satisfaction relation for CTL* state formulas
$\pi \models \neg\varphi$	iff	$\pi \not\models \varphi$
$\pi \models \varphi_1 \wedge \varphi_2$	iff	$\pi \models \varphi_1$ and $\pi \models \varphi_2$
$\pi \models \Box\varphi$	iff	$\text{suffix}(\pi, 1) \models \varphi$
$\pi \models \varphi_1 \cup \varphi_2$	iff	there exists $j \geq 0$ such that $\text{suffix}(\pi, j) \models \varphi_2$ $\text{suffix}(\pi, i) \models \varphi_1$ for $0 \leq i < j$

Con CTL* è possibile scrivere la proprietà di Liveness.

- **CTL** is a sublogic of **CTL***
 - **LTL** is a sublogic of **CTL***
 - **CTL*** is more expressive than **LTL** and **CTL**



$\Phi_1 \equiv \Phi_2$ iff for all transition systems \mathcal{T} :
 $\mathcal{T} \models \Phi_1 \iff \mathcal{T} \models \Phi_2$

Examples:

$$\begin{array}{rcl} \neg\exists E\Diamond a & \equiv & \forall A\Box\neg a \\ \forall A\Diamond\Box a & \equiv & \forall A\Box\forall A a \\ & \vdots & \\ \forall A\Diamond A a & \equiv & A a \\ \exists E\Diamond a & \equiv & \exists a \\ \exists E A a & \equiv & \exists a \end{array}$$

$$\exists \Diamond \exists \Box a \equiv \exists \Diamond \Box a$$

correct. $\neg \square E \Diamond A \equiv \neg \Diamond \square \neg A$

correct. Both formulas assert that an *a*-state is reachable from the current state within one or more steps.