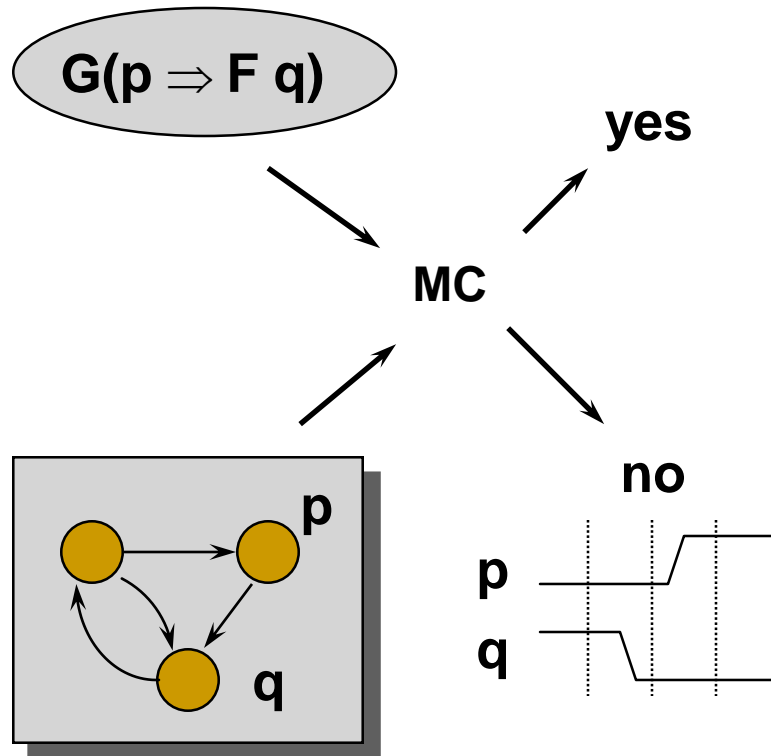

Fondamenti di Model-Checking

Formulazione classica Model Checking



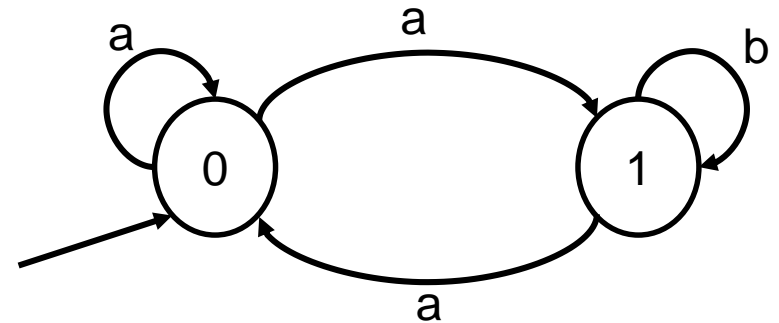
- input:
 - temporal logic formula
 - finite-state model (Kripke)
- output
 - yes
 - no + counterexample

Macchina a stati finiti (FSM)

$$M = (\Sigma, Q, I, \delta)$$

- Σ insieme finito di **simboli**
- Q insieme finito di **stati**
- $I \subseteq Q$ insieme di stati **iniziali**
- $\delta \subseteq Q \times \Sigma \times Q$ relazione di **transizione**

- Es. $(\{a,b\}, \{0,1\}, \{0\}, \{(0,a,0), (0,a,1), (1,a,0), (1,b,1)\})$



- Esecuzione: $r = (q_0, a_1, q_1) (q_1, a_2, q_2) \dots (q_{n-1}, a_n, q_n), \quad (q_{i-1}, a_i, q_i) \in \delta$

Struttura di Kripke

(Macchina a stati finiti con stati etichettati)

$K = (AP, \Sigma, Q, I, \lambda, \delta)$

- AP insieme (finito) di **proposizioni atomiche**
 - λ funzione di **etichettamento**
per ogni $q \in Q$, $\lambda(q) \subseteq AP$
 - $(\Sigma, Q, Q_{in}, \delta)$ macchina a stati finiti
-
- traccia: $\lambda(q_0) \lambda(q_1) \dots \lambda(q_n) \dots$ dove $q_0 \in Q_{in}$ e $(q_0, a_1, q_1) (q_1, a_2, q_2) \dots (q_{n-1}, a_n, q_n) \dots$ è un ω -esecuzione
 - $L(K)$ insieme di tracce di K

LTL model-checking

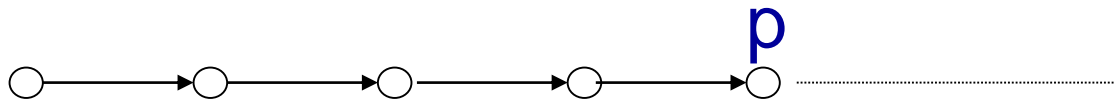
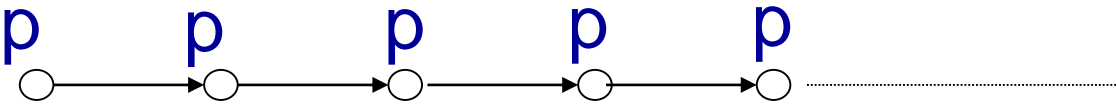
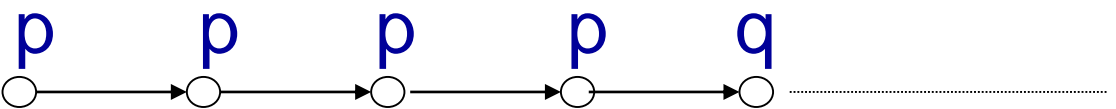
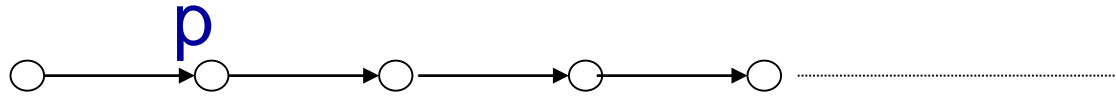
- parola infinita etichettata con proposizioni atomiche:
 - $w: \mathbb{N} \rightarrow 2^{AP}$
(ogni intero rappresenta un istante e per ogni istante i le proposizioni atomiche vere ad i sono $w(i)$)
- soddisfacimento di una formula definito rispetto ad parola infinita (prossima slide)
- data una formula φ , se φ è soddisfatta su w allora w è un modello di φ
- **Model checking**: dato un sistema di transizione T etichettato con proposizioni atomiche (**struttura di Kripke**) e una formula LTL φ , è vero che tutte le computazioni di T sono modelli di φ ?

LTL (Linear Temporal Logic)

■ Sintassi

$$\varphi := p \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \bigcirc \varphi \mid \square \varphi \mid \diamond \varphi \mid \varphi \mathbf{U} \varphi$$

■ Semantica

- $\diamond p$: 
- $\square p$: 
- $p \mathbf{U} q$: 
- $\bigcirc p$: 

LTL model checking

- Problema di decisione (risposta SI/NO)
- Soluzione efficiente si basa su applicazione teoria degli automi
- Riduzione al problema del vuoto per gli automi di Büchi

LTL model-checking (LTL-MC)

- AP: proposizioni atomiche K: struttura di Kripke su AP
 φ : formula LTL su AP
- **Def.** K è un modello di φ se φ è soddisfatta su ogni traccia di K
- **Model-checking:** K è un modello della formula φ ?
 - *K è un modello di φ* se e solo se $L(K) \subseteq L(\varphi)$
se e solo se $L(K) \cap \overline{L(\varphi)} = \emptyset$
se e solo se $L(K) \cap L(\neg \varphi) = \emptyset$
 - Se costruiamo un automa di Büchi che accetta tutte le ω -parole su AP che soddisfano φ abbiamo soluzione a LTL model-checking (automata-theoretic approach)
 - si sfrutta chiusura rispetto all'intersezione e decidibilità del vuoto degli automi di Büchi

Automa finito

$A = (\Sigma, Q, Q_{in}, \delta, Q_{fin})$

- $(\Sigma, Q, Q_{in}, \delta)$ macchina a stati finiti
- $Q_{fin} \subseteq Q$ insieme di stati **finali**
- esecuzione: $r = (q_0, a_1, q_1) (q_1, a_2, q_2) \dots (q_{n-1}, a_n, q_n), (q_{i-1}, a_i, q_i) \in \delta$
- parola associata a esecuzione: $a_1 a_2 \dots a_n$ associata a **r**
- esecuzione di accettazione: $q_0 \in Q_{in}$ e $q_n \in Q_{fin}$
- linguaggio accettato (denotato $L(A)$): insieme di parole associate ad esecuzioni di accettazione
- Nota:
 - determinare se $L(A)$ è non vuoto corrisponde a verificare se Q_{fin} è raggiungibile
 - l'automa che accetta $L(A) \cap L(A')$ è dato dal prodotto delle macchine a stati finiti sottostanti e scegliendo $Q_{fin} \times Q'_{fin}$ come insieme di stati finali

Model checking con automi finiti

Dati gli automi finiti M (modello) e S (specifica),

$$L(M) \subseteq L(S) ?$$

- Risolvibile in spazio polinomiale (Pspace):

$$L(M) \subseteq L(S) \text{ sse } L(M) \cap \overline{L(S)} = \emptyset$$

- automa \bar{S} che accetta complemento di $L(S)$ ha taglia esponenziale in $|S|$ (determinizzazione di S)
- automa che accetta intersezione ha taglia lineare in $|M| \times |\bar{S}|$, e quindi proporzionale a $|M| \times 2^{|S|}$
- vuoto può essere testato in tempo lineare
- spazio polinomiale deriva dal fatto che S può essere costruito on-the-fly durante il test del vuoto

Automi su parole infinite: automi di Büchi

$A = (\Sigma, Q, Q_{in}, \delta, Q_{fin})$ ---- come per automi finiti

- ω -esecuzione: $r = (q_0, a_1, q_1) (q_1, a_2, q_2) \dots (q_{i-1}, a_i, q_i) \dots$ dove
 $(q_{i-1}, a_i, q_i) \in \delta$ per ogni $i \in \mathbb{N}$
- ω -parola associata a ω -esecuzione:
 $a_1 a_2 \dots a_i \dots$ associata a r
- accettazione alla Büchi: $q_0 \in Q_{in}$ e esistono infiniti indici $n \in \mathbb{N}$ tali che $q_n \in Q_{fin}$ (Q_{fin} è visitato infinite volte)
- linguaggio accettato alla Büchi da A : insieme di ω -parole associate ad esecuzioni di accettazione
 - usiamo $L(A)$ per denotare questo linguaggio

Automata-theoretic approach

- Proviamo i seguenti risultati:
 - **Teorema 1.** La classe degli automi di Büchi è chiusa rispetto all'intersezione. L'automa che accetta il linguaggio intersezione si può effettivamente costruire e ha taglia proporzionale al prodotto della taglia degli automi di partenza.
 - **Teorema 2.** Il problema del vuoto per gli automi di Büchi è decidibile in tempo lineare nella taglia dell'automa.
 - **Teorema 3.** Data una formula LTL è possibile costruire un automa di Büchi che accetta esattamente tutti i suoi modelli.

Chiusura rispetto a intersezione

- Automi di Büchi: $A_i = (\Sigma, Q_i, Q_{in_i}, \delta_i, Q_{fin_i})$, $i=1,2$
- Sia $A = (\Sigma, Q, Q_{in}, \delta, Q_{fin})$ dove:
 - $Q = Q_1 \times Q_2 \times \{0,1,2\}$, $Q_{in} = Q_{in_1} \times Q_{in_2} \times \{1\}$
 - $Q_{fin} = Q_1 \times Q_2 \times \{0\}$
 - δ contiene regole del tipo $(q1, q2, i) \xrightarrow{a} (q1', q2', i')$ t.c.:
 - $q1 \xrightarrow{a} q1' \in \delta_1$, $q2 \xrightarrow{a} q2' \in \delta_2$ e
 - $i' = i + 1 \bmod 3$ se $i = 0$ oppure $q_i \in Q_{fin}$
 - $i' = i$, altrimenti
- A simula ogni A_i nella componente i del suo stato
- la terza componente diventa infinite volte 0 sse gli stati finali di entrambi A_1 e A_2 vengono visitati infinite volte
 - se ad es. l'automa A_j da un certo punto non visita più uno stato finale, allora la terza componente rimane bloccata su i

Decidere il vuoto per gli automi di Büchi

- Algoritmo 1 ---componenti fortemente connesse:
 - computa componenti fortemente connesse del grafo di transizione ---tempo $O(|Q|+|\delta|)$
 - denota come finali le componenti che contengono uno stato finale ---contestualmente a step precedente
 - risolvi raggiungibilità con insieme target gli stati delle componenti fortemente connesse finali ---tempo $O(|Q|+|\delta|)$
 - output risultato raggiungibilità
- Si può verificare che Algoritmo 1 da una risposta affermativa sse l'automa di Büchi in input accetta un linguaggio non vuoto

Nested depth-first search

- Idea: eseguire due DFS interfogliate
 - una esterna per visitare tutti gli stati finali raggiungibili
 - una interna per individuare cicli su stati finali
- Algoritmo 2 ---nested DFS:
 - non appena terminata la visita DFS esterna da tutti i successori di s , se s è *finale* parte la DFS interna da s
 - nella DFS interna, vengono esplorati tutti gli stati raggiungibili da s *non ancora visitati* nella DFS interna (la DFS interna è fatta a pezzi, e si interfoglia con quella esterna)
 - nessun ciclo su s ? continua la DFS esterna

Nested depth-first search

- *Generazione del controesempio*: concatena gli stack DFS
 - stack U usato per la DFS esterna = cammino da $s_0 \in I$ a s (dal bottom al top)
 - stack V usato per la DFS interna = ciclo da s a s (dal bottom al top)
- Tempo di esecuzione $O(|Q| + |\delta|)$
- Si può verificare che Algoritmo 2 da una risposta affermativa sse l'automa di Büchi in input accetta un linguaggio non vuoto
- Inoltre, in caso di risposta affermativa viene generato un cammino da uno stato iniziale a uno stato di accettazione seguito da un ciclo su questo stato
 - se automa di Büchi esprime negazione specifica, output è controesempio

Pseudo-codice nested DFS

```
set of states  $R := \emptyset$ ; //stati visitati DFS est.  
stack of states  $U := \varepsilon$ ; // stack DFS est.  
set of states  $T := \emptyset$ ; //stati visitati DFS int.  
stack of states  $V := \varepsilon$ ; //stack DFS int.  
  
boolean cycle found := false;  
  
while ( $Q\_in \setminus R \neq \emptyset \wedge !\text{cycle found}$ ) do  
    let  $s \in Q\_in \setminus R$ ; // explore the reachable  
        reachable cycle( $s$ ); // outer DFS  
od  
if !cycle found then return ("yes")  
else return ("no", reverse( $V.U$ ))  
fi
```

```
procedure reachable cycle (state  $s$ )  
    push( $s, U$ );  $R := R \cup \{s\}$ ;  
    repeat  
         $s := \text{top}(U)$ ;  
        if  $\text{Post}(s) \setminus R \neq \emptyset$  then  
            let  $s \in \text{Post}(s) \setminus R$ ;  
            push( $s, U$ );  $R := R \cup \{s\}$ ;  
        else  
            pop( $U$ ); // DFS est. finita per  $s$   
            if  $s$  è finale then  
                cycle found := cycle_check( $s$ ); fi  
        fi  
    until ( $(U = \varepsilon) \vee \text{cycle found}$ )  
endproc
```

Pseudo-codice nested DFS: cycle_check

/ esegue DFS interna */*

procedure boolean cycle_check(state s)

boolean cycle found := false;

 push(s, V); *// V è uno stack*

$T := T \cup \{s\}$;

repeat

$s' := \text{top}(V)$;

if $s \in \text{Post}(s')$ **then**

 cycle found := true;

 push(s, V);

else

if $\text{Post}(s') \setminus T \neq \emptyset$ **then**

// stati già visitati in DFS interna

// non vengono rivisitati

let $s'' \in \text{Post}(s') \setminus T$;

 push(s'' , V);

$T := T \cup \{s''\}$;

else

 pop(V);

fi

fi

until $((V = \varepsilon) \vee \text{cycle found})$

return cycle found

endproc

Correttezza nested DFS

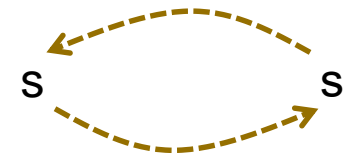
- Supponiamo che un ciclo su stato finale **s** non viene scoperto
 - DFS interna è in carico di scoprire i cicli su stati finali
 - possibile errore: non rivisitiamo stato cruciale per ciclo

- Questo accade quando l'algoritmo:

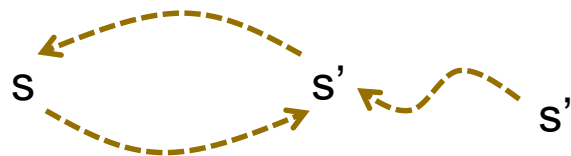
(1) da **s** scopre **s'** nella DFS interna, ma

(2) **s'** è già stato visitato e

(3) c'è un ciclo **c** su **s** che passa da **s'**



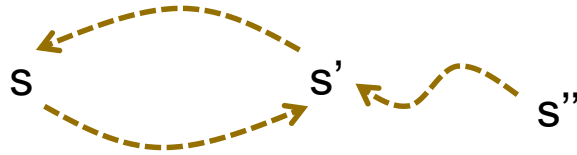
- Da (2), esiste stato finale **s''** da cui raggiungiamo **s'** nella DFS interna (e quindi tutto il ciclo **c**, ed in particolare **s**, è già visitato nella DFS interna)



Correttezza nested DFS

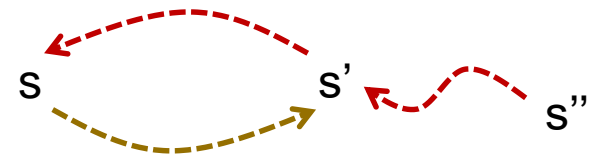
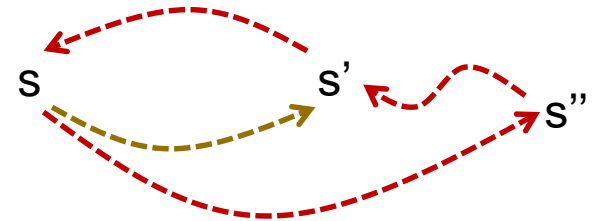
- Ipotesi per assurdo:
 - Siano s , s' e s'' i primi stati che soddisfano la condizione precedente durante la computazione
 - cioè:
 - (1) s' è scoperto da s nella DFS interna
 - (2) s' è già stato visitato nella DFS interna da s''
 - (3) c'è un ciclo c su s che passa da s'

Correttezza nested DFS



■ Due casi possibili:

- s'' scoperto da s in DFS esterna, allora esiste ciclo su s'' che dovrebbe essere scoperto prima e terminare l'algoritmo (assurdo)
- altrimenti, DFS esterna avrebbe dovuto scoprire s da s'' tramite s' e quindi s' non potrebbe essere stato già visitato quando viene scoperto in DFS interna da s (assurdo)
 - DFS interna partirebbe prima da s e poi da s''



Algoritmo 1 vs Algoritmo 2

- per applicazioni in model-checking è preferibile utilizzare la nested DFS
 - principali vantaggi:
 - ha un'implementazione on-the-fly (richiede meno spazio)
 - consente di generare un controesempio facilmente
 - stessa complessità asintotica
-

Automa di Büchi generalizzato (GBA)

$A = (\Sigma, Q, Q_{in}, \delta, F)$ dove:

- $\Sigma, Q, Q_{in}, \delta$ come in automa di Büchi
- $F = \{F_1, \dots, F_k\}$ famiglia di insiemi di stati finali, $F_i \subseteq Q$ per ogni i
- accettazione alla Büchi generalizzata:
 - data un' ω -esecuzione
$$r = (q_0, a_1, q_1) (q_1, a_2, q_2) \dots (q_{i-1}, a_i, q_i) \dots$$
 - per ogni $F_h \in F$: $q_i \in F_h$ per un numero infinito di indici i
- utile per costruire automa equivalente a formula LTL
- Esercizio: provare che dato un automa di Büchi generalizzato A come sopra, esiste un automa di Büchi B tale che $L(A) = L(B)$ e $|B| = O(k |A|)$

Chiusura e insiemi elementari

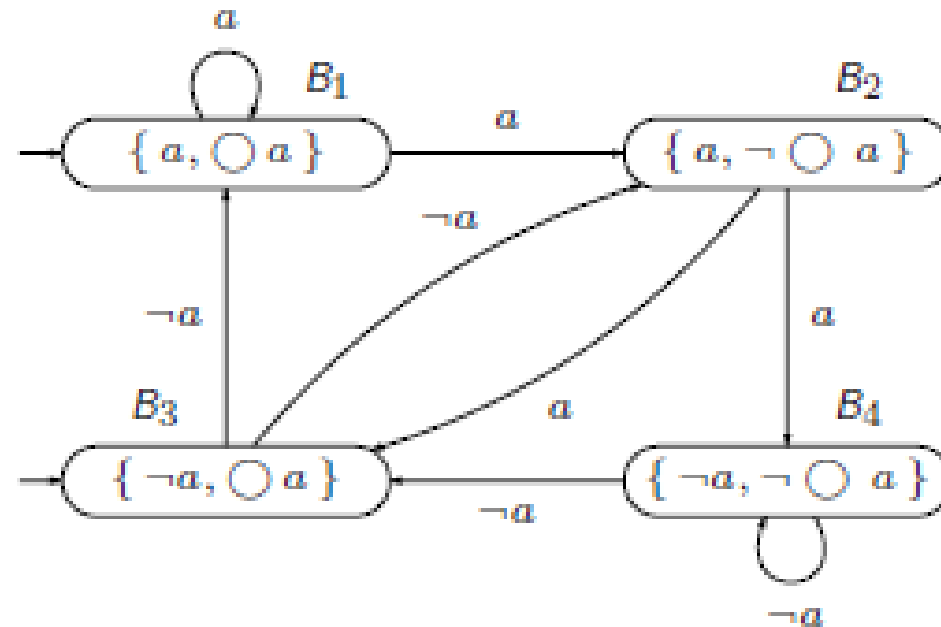
- Sia ϕ una formula LTL, l'insieme **closure**(ϕ) è l'insieme di tutte e sole le sottoformule ψ di ϕ e della loro negazione $\neg\psi$ (dove ψ e $\neg\neg\psi$ sono identificate)
- $B \subseteq \text{closure}(\phi)$ è **elementare** se
 - B è **logically consistent**: per ogni $\phi_1 \wedge \phi_2, \psi \in \text{closure}(\phi)$
 - $\phi_1 \wedge \phi_2 \in B \Leftrightarrow \phi_1 \in B \text{ e } \phi_2 \in B$
 - $\psi \in B \Rightarrow \neg\psi \notin B$
 - $\text{true} \in \text{closure}(\phi) \Rightarrow \text{true} \in B$
 - B è **locally consistent**: per ogni $\phi_1 \cup \phi_2 \in \text{closure}(\phi)$
 - $\phi_2 \in B \Rightarrow \phi_1 \cup \phi_2 \in B$
 - $\phi_1 \cup \phi_2 \in B \text{ and } \phi_2 \notin B \Rightarrow \phi_1 \in B$
 - B è **massimale**: per ogni $\psi \in \text{closure}(\phi)$
 - $\psi \notin B \Rightarrow \neg\psi \in B$

GBA equivalente a LTL formula

Per una LTL-formula ϕ , sia $G_\phi = (2^{AP}, Q, Q_{in}, \delta, F)$ dove

- $Q =$ tutti gli insiemi elementari $B \subseteq \text{closure}(\phi)$
- $Q_{in} = \{B \in Q \mid \phi \in B\}$
- $F = \{ \{B \in Q \mid \phi_1 \cup \phi_2 \notin B \text{ or } \phi_2 \in B\} \mid \phi_1 \cup \phi_2 \in \text{closure}(\phi) \}$
- La relazione di transizione $\delta : Q \times 2^{AP} \rightarrow 2^Q$ è data da:
 - Se $A \neq B \cap AP$, allora $\delta(B, A) = \emptyset$
 - $\delta(B, B \cap AP)$ è l'insieme B' di tutti gli insiemi elementari di formule che soddisfano:
 - (i) Per ogni $\bigcirc \psi \in \text{closure}(\phi)$: $\bigcirc \psi \in B \Leftrightarrow \psi \in B'$, and
 - (ii) For every $\phi_1 \cup \phi_2 \in \text{closure}(\phi)$:
$$\phi_1 \cup \phi_2 \in B \Leftrightarrow \phi_2 \in B \vee (\phi_1 \in B \wedge \phi_1 \cup \phi_2 \in B')$$

GNBA for LTL-formula $\bigcirc a$



$Q_0 = \{ B_1, B_3 \}$ since $\bigcirc a \in B_1$ and $\bigcirc a \in B_3$

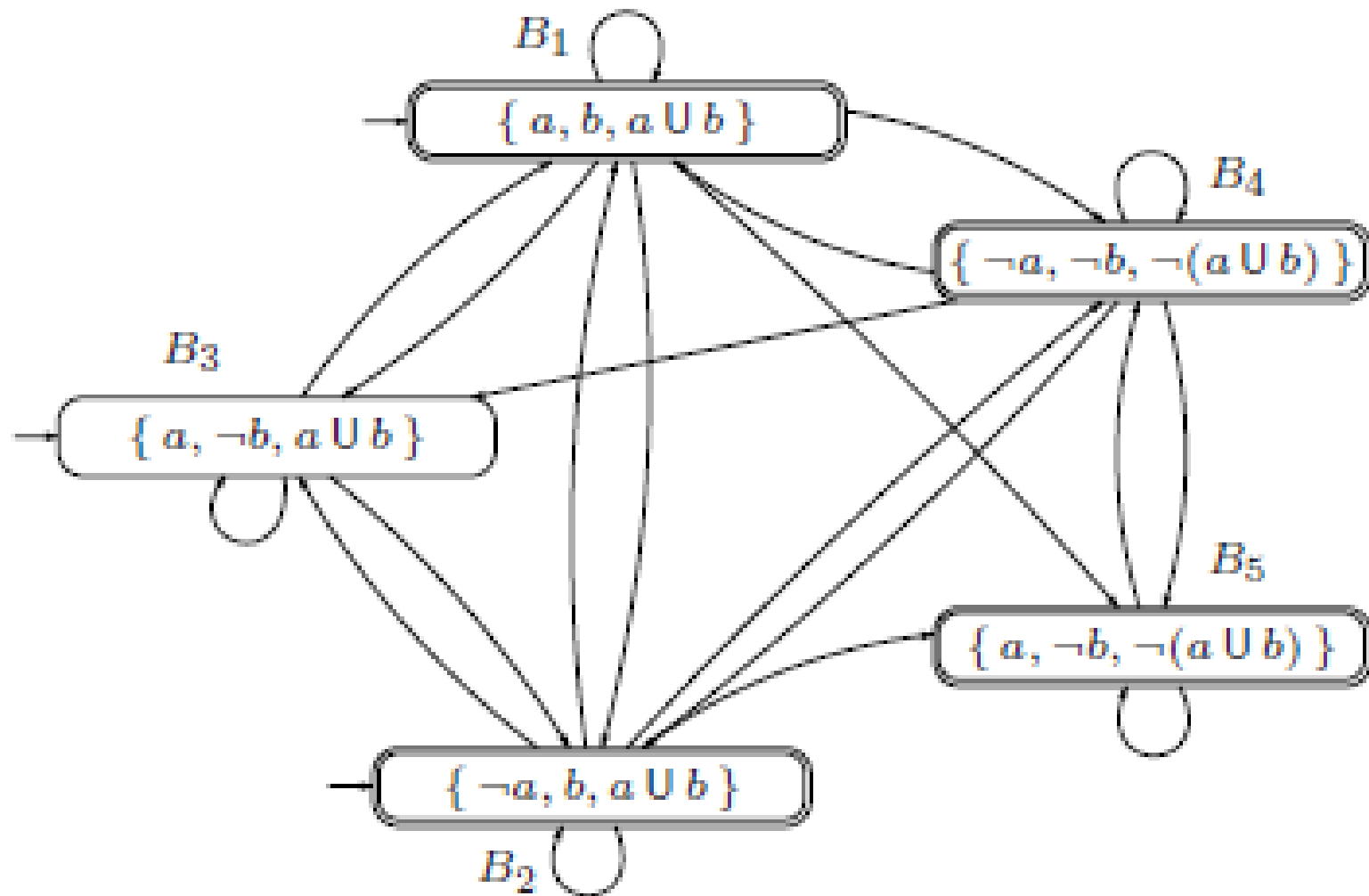
$\delta(B_2, \{ a \}) = \{ B_3, B_4 \}$ as $B_2 \cap \{ a \} = \{ a \}$, $\neg \bigcirc a = \bigcirc \neg a \in B_2$, and $\neg a \in B_3, B_4$

$\delta(B_1, \{ a \}) = \{ B_1, B_2 \}$ as $B_1 \cap \{ a \} = \{ a \}$, $\bigcirc a \in B_1$ and $a \in B_1, B_2$

$\delta(B_4, \{ a \}) = \emptyset$ since $B_4 \cap \{ a \} = \emptyset \neq \{ a \}$

The set \mathcal{F} is empty, since $\varphi = \bigcirc a$ does not contain an until-operator

GNBA for LTL-formula $a \cup b$



Automi di Büchi più espressivi di LTL

- $AP = \{p\}$
- Sia L il linguaggio di tutte le sequenze sull'alfabeto $2^{\{p\}}$ tali che $\{p\}$ sia il simbolo ad ogni posizione pari:
$$L = \{A_1 A_2 A_3 \dots / A_{2i} = \{p\} \text{ per ogni } i \geq 0\}$$
- Non esiste alcuna formula LTL φ su AP tale che $L(\varphi) = L$
- Tuttavia esiste un automa di Büchi B tale che $L(B) = L$
(Esercizio)

Complessità LTL model checking

- Gli stati di G_ϕ sono insiemi elementari di formule in $\text{closure}(\phi)$
 - ogni insieme B può essere rappresentato con un vettore di bit, con un bit in corrispondenza di ogni sottoformula ψ di ϕ
- Il numero di stati di G_ϕ è al più $2^{|\text{subf}(\phi)|}$
 - dove $\text{subf}(\phi)$ è l'insieme di tutte le sottoformule di ϕ
 - siccome $|\text{subf}(\phi)| \leq 2 \cdot |\phi|$, il numero di stati di G_ϕ è $2^{O(|\phi|)}$
- Il numero di insiemi di accettazione di G_ϕ è $O(|\phi|)$
- Il numero di stati nell'automa di Büchi equivalente A_ϕ è $2^{O(|\phi|)} O(|\phi|)$
- $2^{O(|\phi|)} O(|\phi|) = 2^{O(|\phi| + \log |\phi|)} = 2^{O(|\phi|)}$

Complessità LTL model checking

- Dato che
 - è possibile costruire in tempo lineare un automa A_K che accetta $L(K)$ (tracce di una struttura di Kripke K) di taglia $O(|K|)$
 - è possibile costruire un automa A che accetta $L(K) \cap L(A_\phi)$ in tempo $O(|K| |A_\phi|)$ e tale che $|A| = O(|K| |A_\phi|)$
 - $|A_\phi| = 2^{O(|\phi|)}$
 - È possibile testare il vuoto di un automa di Büchi B in tempo $O(|B|)$ e spazio logaritmico (NLOGSPACE)
- LTL model-checking può essere risolto in tempo $|K| 2^{O(|\phi|)}$ e spazio polinomiale (PSPACE)
- Inoltre, si può verificare che il problema è anche PSPACE-hard, dunque LTL model-checking è PSPACE-complete

Osservazioni sulla complessità

- Le specifiche possono essere spezzettate in formule di taglia piccola (tempo esponenziale nella taglia della formula non è un problema)
- La taglia del modello è solitamente esponenziale nella taglia della sua descrizione (state-space explosion)
 - trasformazione **variabili** → **stati**:
esponenziale in numero di bit (per var)
 - trasformazione **componenti** → **modello**:
esponenziale in numero componenti

FSM con memoria condivisa e canali

- Comunicazione con altre FSM può avvenire tramite memoria condivisa o canali
- FSM con memoria condivisa
 - Stati: (g, l) dove g è condiviso e l è locale
 - Transizioni: $(g, l) \xrightarrow{a} (g', l')$
- FSM con canali (code):
 - Transizioni:
 - $(q) \xrightarrow{\text{snd}(\alpha, i)} (q')$ invia α sul canale i (inserito in coda)
 - $(q) \xrightarrow{\text{rec}(\alpha, i)} (q')$ ricevi α sul canale i (prelevato dal front)

Composizione parallela di FSM

- Dato un sistema S composto di n FSM M_1, M_2, \dots, M_n che interagiscono attraverso:
 - memoria condivisa finita G
 - m canali di lunghezza finita l_1, \dots, l_m su alfabeto messaggi Γ
- E' possibile costruire una FSM equivalente M

Composizione parallela di FSM

- FSM M : $(Q_i$ stati locali e I_i stati iniziali di M_i)
 - Alfabeto: $\Sigma_1 \cup \dots \cup \Sigma_n$ (unione alfabeti M_1, M_2, \dots, M_n)
 - Stati: $(g, q_1, \dots, q_n, w_1, \dots, w_m) \in G \times Q_1 \times \dots \times Q_n \times \Gamma^{l_1} \times \dots \times \Gamma^{l_m}$
 - Stati iniziali: $G_0 \times I_1 \times \dots \times I_n \times \{\varepsilon\} \times \dots \times \{\varepsilon\}$
 - Transizioni:
 - solo una componente muove
ad es. se M_i muove senza usare canali abbiamo:
 $(g, q_1, \dots, q_i, \dots, q_n, w_1, \dots, w_m) \xrightarrow{a} (f, q_1, \dots, p_i, \dots, q_n, w_1, \dots, w_m)$
per una transizione $(g, q_i) \xrightarrow{a} (f, p_i)$ di M_i
 - eccetto se send/receive su canale di dimensione 0 (rendez-vous),
in questo caso il send e il corrispondente receive sono eseguiti in coppia

Raggiungibilità: problema di verifica base

- Dato una macchina a stati finiti FSM determinare se un insieme di stati target **T** è raggiungibile
- Soluzione: DFS (o BFS) a partire da stati iniziali, $O(|Q|+|\delta|)$

DFS (q)

Add q to Set_of_Visited_States;

for each q' such that $(q, a, q') \in \delta$

do if q' is in T,

 print “Target found” ; halt.

else if q' is not in Set_of_Visited_States

 DFS(q')

Estensioni

- Classi di modelli

- Modelli con stack (chiamate ricorsive programmi)
- Modelli con stack multipli (programmi multithreaded)
- Modelli con variabili continue (sistemi embedded real-time e ibridi)
- Altri modelli a stati infiniti (reti di Petri, higher-order PDA,...)

- Classi di specifiche

- Logica temporale branching-time (CTL,CTL*)
- mu-calculus

.....