

Software Testing and Validation

A.A. 2022/2023

Corso di Laurea in Informatica

CTL and LTL Model Checking Algorithms

Igor Melatti

Università degli Studi dell'Aquila

Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica



UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

Theoretic vs. Practical Algorithms

- Model Checking problem:
 - input: a KS $\mathcal{S} = \langle S, I, R, L \rangle$ and a formula φ
 - output: true iff $\mathcal{S} \models \varphi$, $\langle \text{false}, c \rangle$ otherwise, being c a counterexample
- Depending on φ being LTL or CTL, different algorithms must be provided
- We will first show the “theoretical” algorithm for CTL
 - classical approach: both S and R fit into RAM
- Then, we will see how they can be efficiently implemented
 - LTL: SPIN and NuSMV
 - CTL: NuSMV



UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

CTL Theoretic Algorithm

- CTL is based on *state* formulas, i.e., φ holds depending on the state we are considering
 - this also holds for subformulas of φ , e.g., **AFAG** p has one subformula **AG** p
- Since we have the full state space S , we label all states $s \in S$ with (sub)formulas holding in s
 - not just the reachable states: all of them
- Then, we use subformulas labeling to decide higher formulas labelling
- Thus, we have $\lambda : S \rightarrow 2^{\text{CTL}}$, being CTL the set of all CTL formulas
- At the end, $S \models \varphi$ iff $\forall s \in I. \varphi \in \lambda(s)$



UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

CTL Theoretic Algorithm

- Consider the abstract syntax tree of φ , call it ϕ
- Start from the leaves in ϕ , which must be an atomic proposition p or true
 - $\forall s \in S. p \in \lambda(s) \Leftrightarrow p \in L(s)$
 - $\forall s \in S. \text{true} \in \lambda(s)$
- Then go upwards in ϕ , using, for each node, the labeling of the sons
 - $\forall s \in S. \neg\Phi \in \lambda(s) \Leftrightarrow \Phi \notin \lambda(s)$
 - $\forall s \in S. \Phi_1 \wedge \Phi_2 \in \lambda(s) \Leftrightarrow (\Phi_1 \in \lambda(s) \wedge \Phi_2 \in \lambda(s))$
 - $\forall s \in S. \mathbf{EX}\Phi \in \lambda(s) \Leftrightarrow (\exists s' : (s, s') \in R \wedge \Phi \in \lambda(s'))$



UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

CTL Theoretic Algorithm: $\Phi_1 \mathbf{EU} \Phi_2 \in \lambda(s)$

- We already have $\lambda(\Phi_1)$ and $\lambda(\Phi_2)$
- All states satisfying Φ_2 are ok
- Then, backward visit of the state space of \mathcal{S}
- The backward visit stops when Φ_1 does not hold
- Complexity is $O(|S| + |R|)$



UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

CTL Theoretic Algorithm: $\Phi_1 \mathbf{EU} \Phi_2 \in \lambda(s)$

```
labels CheckEU(KS  $\mathcal{S}$ , formula  $\Phi_1 \mathbf{EU} \Phi_2$ , labels  $\lambda$ )
{
  let  $\mathcal{S} = \langle S, I, R, L \rangle$ ;
   $T = \{s \in S \mid \Phi_2 \in \lambda(s)\}$ ;
  foreach  $s \in T$ 
     $\lambda(s) = \lambda(s) \cup \{\Phi_1 \mathbf{EU} \Phi_2\}$ ;
  while ( $T \neq \emptyset$ ) {
    let  $s$  be s.t.  $s \in T$ ;
     $T = T \setminus \{s\}$ ;
    foreach  $t \in \{t \mid (t, s) \in R\}$  {
      if  $\Phi_1 \mathbf{EU} \Phi_2 \notin \lambda(t) \wedge \Phi_1 \in \lambda(t)$  {
         $\lambda(t) = \lambda(t) \cup \{\Phi_1 \mathbf{EU} \Phi_2\}$ ;
         $T = T \cup \{t\}$ ;
      } /* if */ } /* foreach */ } /* while */
  return  $\lambda$ ;
}
```



UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

CTL Theoretic Algorithm: $\mathbf{EG}\Phi \in \lambda(s)$

- We already have $\lambda(\Phi)$
- Consider all states in which Φ holds: this defines a subKS \mathcal{S}' of \mathcal{S}
- Then, compute the strongly connected components (SCCs) of \mathcal{S}'
 - inside such components, Φ holds on all states on all paths
- Finally, label with $\mathbf{EG}\Phi$ all s in such SCCs, plus all backward reachable $t \in \mathcal{S}'$
 - so we move on states for which Φ holds...
- Complexity is again $O(|S| + |R|)$



UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

CTL Theoretic Algorithm: $\mathbf{EG}\Phi \in \lambda(s)$

```
labels CheckEG(KS  $\mathcal{S}$ , formula  $\mathbf{EG}\Phi$ , labels  $\lambda$ )
{
  let  $\mathcal{S} = \langle S, I, R, L \rangle$ ;
   $S' = \{s \in S \mid \Phi \in \lambda(s)\}$ ;  $R' = \{(s, t) \in R \mid s, t \in S'\}$ ;
   $\mathcal{A} = \text{SCC}(S', R')$ ;  $T = \bigcup_{A \in \mathcal{A}} A$ ;
  foreach  $s \in T$ ,  $\lambda(s) = \lambda(s) \cup \{\mathbf{EG}\Phi\}$ ;
  while ( $T \neq \emptyset$ ) {
    let  $s$  be s.t.  $s \in T$ ;
     $T = T \setminus \{s\}$ ;
    foreach  $t \in \{t \mid (t, s) \in R'\}$  {
      if  $\mathbf{EG}\Phi \notin \lambda(t)$  { /* since  $(t, s) \in R'$ ,  $\Phi \in \lambda(t)$  */
         $\lambda(t) = \lambda(t) \cup \{\mathbf{EG}\Phi\}$ ;
         $T = T \cup \{t\}$ ;
      } /* if */ } /* foreach */ } /* while */
  return  $\lambda$ ;
}
```



CTL Theoretic Algorithm: Complexity

- Complexity is:
 - $O(|S|)$ for boolean combinations and atomic propositions
 - $O(|S|)$ also for **EX** Φ
 - $O(|S| + |R|)$ for **EG** Φ and Φ_1 **EU** Φ_2
- Since this must be done for every subformula of φ , the overall complexity is $O((|S| + |R|)|\varphi|)$
 - $|\varphi|$ is the number of nodes of the abstract syntax tree of φ
- Linear in the size of the input, if one of the two is fixed... is this as good as it seems?
- Alas no: state space explosion hits exactly in $|S|$ and $|R|$
 - $|\varphi|$ is typically low for real-world properties to be verified

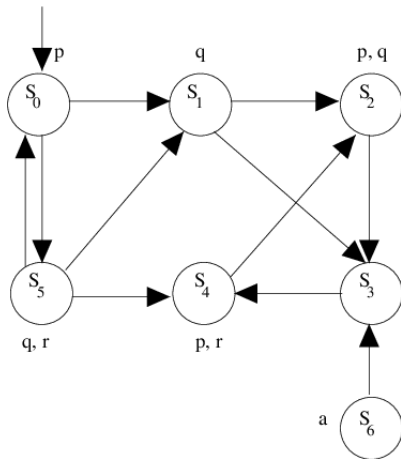


UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

CTL Model Checking Algorithm Running Example



$$\varphi = \mathbf{EFAF}p = \text{true } \mathbf{EU}(\neg(\mathbf{EG}\neg p))$$

Leaves of φ AST are true and p , thus:

$$\forall i \in \{0, 2, 4\}. \quad \lambda(s_i) = \{\text{true}, p\}$$

$$\forall i \in \{1, 3, 5, 6\}. \quad \lambda(s_i) = \{\text{true}\},$$

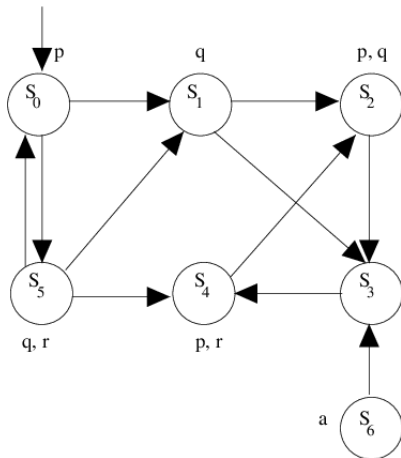


UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

CTL Model Checking Algorithm Running Example



$\varphi = \text{true } \mathbf{EU}(\neg(\mathbf{EG}\neg p))$

Going up one level:

$\forall i \in \{0, 2, 4\}. \lambda(s_i) = \{\text{true}, p\}$

$\forall i \in \{1, 3, 5, 6\}. \lambda(s_i) = \{\text{true}, \neg p\},$

Going up two levels:

$\text{CheckEG}(\mathcal{S}, \mathbf{EG}\neg p, \lambda)$



UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

CTL Theoretic Algorithm: $\mathbf{EG}\Phi \in \lambda(s)$

```
labels CheckEG(KS  $\mathcal{S}$ , formula  $\mathbf{EG}\Phi$ , labels  $\lambda$ )
{
  let  $\mathcal{S} = \langle S, I, R, L \rangle$ ;
   $S' = \{s \in S \mid \Phi \in \lambda(s)\}$ ;  $R' = \{(s, t) \in R \mid s, t \in S'\}$ ;
   $\mathcal{A} = \text{SCC}(S', R')$ ;  $T = \bigcup_{A \in \mathcal{A}} A$ ;
  foreach  $s \in T$ ,  $\lambda(s) = \lambda(s) \cup \{\mathbf{EG}\Phi\}$ ;
  while ( $T \neq \emptyset$ ) {
    let  $s$  be s.t.  $s \in T$ ;
     $T = T \setminus \{s\}$ ;
    foreach  $t \in \{t \mid (t, s) \in R'\}$  {
      if  $\mathbf{EG}\Phi \notin \lambda(t)$  {
         $\lambda(t) = \lambda(t) \cup \{\mathbf{EG}\Phi\}$ ;
         $T = T \cup \{t\}$ ;
      } /* if */ } /* foreach */ } /* while */
  return  $\lambda$ ;
}
```

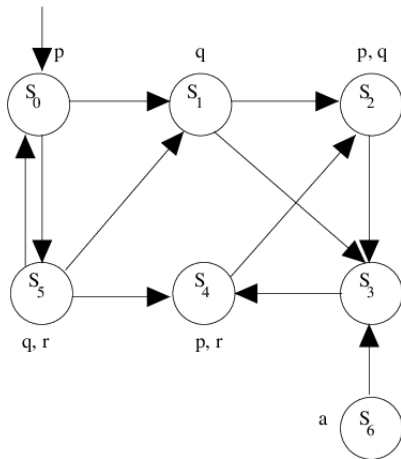


UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

CTL Model Checking Algorithm Running Example



$\varphi = \text{true } \mathbf{EU}(\neg(\mathbf{EG}\neg p))$
CheckEG(\mathcal{S} , $\mathbf{EG}\neg p$, λ)

$S' = \{s_1, s_3, s_5, s_6\}$

There are no non-trivial SCC on S'

Thus $T = \emptyset$ and λ does not change

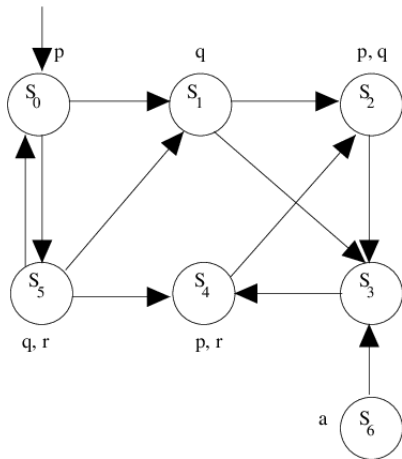


UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

CTL Model Checking Algorithm Running Example



$\varphi = \text{true } \mathbf{EU}(\neg(\mathbf{EG}\neg p))$

$\forall i \in \{0, 2, 4\}. \lambda(s_i) = \{\text{true}, p\}$

$\forall i \in \{1, 3, 5, 6\}. \lambda(s_i) = \{\text{true}, \neg p\},$

Going up one more level:

$\forall i \in \{0, 2, 4\}. \lambda(s_i) = \{\text{true}, p, \neg(\mathbf{EG}\neg p)\}$

$\forall i \in \{1, 3, 5, 6\}. \lambda(s_i) = \{\text{true}, \neg p, \neg(\mathbf{EG}\neg p)\}$

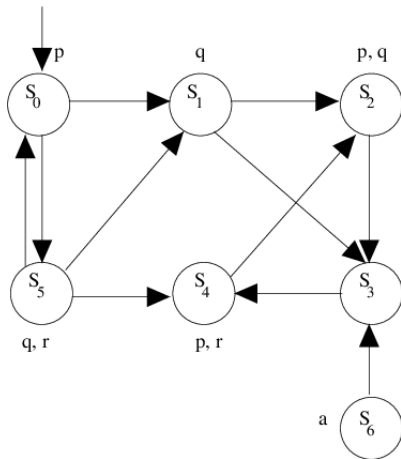


UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

CTL Model Checking Algorithm Running Example



$\varphi = \text{true EU}(\neg(\mathbf{EG}\neg p))$

Finally, call $\text{CheckEU}(S, \text{true EU}(\neg(\mathbf{EG}\neg p), \text{labels } \lambda))$

$T = S$, as all states are labelled with $\text{true EU}(\neg(\mathbf{EG}\neg p))$

Thus, all states must be labelled with φ



UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



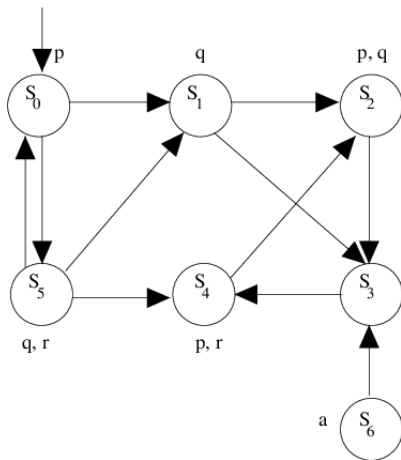
DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

CTL Theoretic Algorithm: $\Phi_1 \mathbf{EU} \Phi_2 \in \lambda(s)$

```
labels CheckEU(KS  $\mathcal{S}$ , formula  $\Phi_1 \mathbf{EU} \Phi_2$ , labels  $\lambda$ )
{
  let  $\mathcal{S} = \langle S, I, R, L \rangle$ ;
   $T = \{s \in S \mid \Phi_2 \in \lambda(s)\}$ ;
  foreach  $s \in T$ 
     $\lambda(s) = \lambda(s) \cup \{\Phi_1 \mathbf{EU} \Phi_2\}$ ;
  while ( $T \neq \emptyset$ ) {
    let  $s$  be s.t.  $s \in T$ ;
     $T = T \setminus \{s\}$ ;
    foreach  $t \in \{t \mid (t, s) \in R\}$  {
      if  $\Phi_1 \mathbf{EU} \Phi_2 \notin \lambda(t) \wedge \Phi_1 \in \lambda(t)$  {
         $\lambda(s) = \lambda(s) \cup \{\Phi_1 \mathbf{EU} \Phi_2\}$ ;
         $T = T \cup \{t\}$ ;
      } /* if */ } /* foreach */ } /* while */
  return  $\lambda$ ;
}
```



CTL Model Checking Algorithm Running Example



$\varphi = \text{true } \mathbf{EU}(\neg(\mathbf{EG}\neg p))$

$\forall i \in \{0, 2, 4\}. \lambda(s_i) = \{\text{true}, p, \neg(\mathbf{EG}\neg p), \varphi\}$

$\forall i \in \{1, 3, 5, 6\}. \lambda(s_i) = \{\text{true}, \neg p, \neg(\mathbf{EG}\neg p), \varphi\}$

Since $\varphi \in \lambda(s_0)$, we have that $\mathcal{S} \models \varphi$



UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

LTL Model Checking Algorithm

- Many LTL algorithms exist, we will directly see the most efficient one
- Surprising fact: not only LTL is not included inside CTL, it is also more difficult to check!
- Namely, whilst CTL model checking is in P, LTL model checking is PSPACE-complete
 - no, PSPACE is not “good” as P is: $NP \subseteq PSPACE$
- Efficient algorithms for LTL run in $O((|S| + |R|)2^{|\varphi|})$
- In practice, this is not much worse than CTL model checking
 - the real problem is $O(|S| + |R|)$
 - φ is usually small, it is difficult to come up with lengthy formulas



UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

LTL Model Checking Algorithm

- The idea is simple: first translate φ into a special automaton $\mathcal{A}(\varphi)$
- Then, visit both \mathcal{S} and $\mathcal{A}(\varphi)$, one step at a time
 - equivalent to verify to Cartesian product $\mathcal{S} \times \mathcal{A}(\varphi)$
- If some special node is found, we have a counterexample for φ
- Otherwise, $\mathcal{S} \models \varphi$
- Such algorithm may be implemented on-the-fly, thus instead of a KS we have an NFSS
 - no need to have S and R in memory before starting



UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

Büchi Automaton

- A (non-deterministic) Büchi Automaton (BA) is a 5-tuple $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, F \rangle$ where:
 - Σ is the *alphabet*, i.e., a finite set of symbols
 - Q is the finite set of states
 - $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation
 - $Q_0 \subseteq Q$ are the initial states
 - $F \subseteq Q$ are the final states
- With respect to a KS, we also have final states and edges are labeled with symbols from an alphabet
 - the labeling L is also missing in BAs
 - however, we will see that AP is linked to Σ



UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

Büchi Automaton

- BAs are not different from well-known automata in computational theory
 - finite state automata (FSA) are essentially equal in the definition
- The difference is in the language they accept
 - FSA: a word w is recognized if, by walking inside the FSA through symbols in w , a final state is reached
 - this implies that $|w| < \infty$
 - the set of all recognized w may be infinite, but each w is finite
- A BA recognize a(n infinite) language of *infinite* words
 - each word w has an infinite number of symbols



UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

Language Accepted by Büchi Automata

- Let $w = w_0w_1 \dots$ be an infinite string s.t. $\forall i. w_i \in \Sigma$
 - $w \in \Sigma^\omega$
- The BA \mathcal{A} *accepts* w iff there exists a path $\pi = q_0w_0q_1w_1 \dots$ s.t.
 - $\forall i. q_i \in Q \wedge w_i \in w \wedge (q_i, w_i, q_{i+1}) \in \delta$
 - $q_0 \in Q_0$
 - if $I = \{i \mid q_i \in F\}$, then $|I| = \infty$
 - otherwise stated: π goes through a final state *infinitely often* (or *almost always*)
 - this is where the definition differs from FSAs, where π is finite and its final state must be in F
- $\mathcal{L}(\mathcal{A})$ is the set of infinite words recognized by \mathcal{A}
- Languages recognized by a BA are called ω -regular
 - recall that FSA recognize *regular* languages

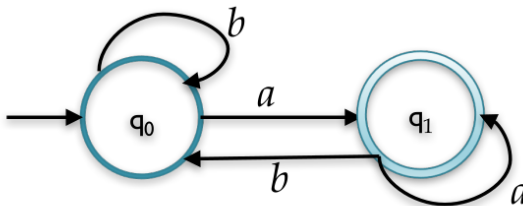


UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

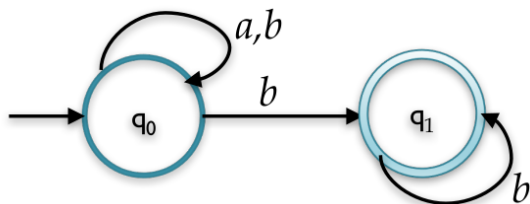
Büchi Automata Examples



- Final states are those with thicker boundaries, initial states are pointed to by an arrow
- This recognizes the language $(b^*a)^\omega$
- Note that a^* is a language (set of infinite words) containing $\varepsilon, a, aa, aaa, \dots$
- Note that a^ω is a single infinite word $aaaaaa \dots$
- Thus, $(b^*a)^\omega = \{a^\omega, ba^\omega, bba^\omega, \dots\}$
- That is: a finite number of b 's, followed by infinite a 's



Büchi Automata Examples



- This recognizes the language $(a + b)^* b^\omega$
- That is, $(a + b)^* b^\omega = \{b^\omega, ab^\omega, abab^\omega, abbabbbab^\omega, \dots\}$
- That is: any finite sequence of a and b , followed by infinite b 's
- May also be stated as: a finite number of a and infinite b
- Cannot be recognized by a deterministic BA!
 - instead, deterministic FSAs recognize the same languages of non-deterministic FSAs



UNIVERSITÀ
DELL'AQUILA



DISM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

Büchi Automata and LTL Properties

- Also LTL properties are related to infinite words
 - recall that a *model* σ is an infinite sequence of truth assignments to all $p \in AP$
 - by adapting LTL semantics about $\pi \models \varphi$, we can define whether $\sigma \models \varphi$
 - we replace a path state $\pi(i)$ with the set $P_i \subseteq AP$ s.t.
 $P_i = \{p \in AP \mid p \in L(\pi(i))\}$
- Thus, an LTL property recognizes a language
 $\mathcal{L}(\varphi) = \{\sigma \in AP^\omega \mid \sigma \models \varphi\}$
 - sometimes, we use φ and $P = \mathcal{L}(\varphi)$ interchangeably
- Furthermore, the “infinitely often” part recalls the LTL formula **GF** p
- Also the “eventually forever” **FG** p is important



UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

Büchi Automata and LTL Properties

- Let φ be an LTL formula, and let $\mathcal{L}(\varphi)$ be the set of models of φ . Then, there exists a BA \mathcal{A}_φ s.t. $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$
 - it is easy to show that the vice versa does not hold
- We skip the proof, but:
 - of course, we have $\Sigma = 2^{AP}$
 - the size of \mathcal{A}_φ , i.e., the number of states, is $2^{O(|\varphi|)}$
 - since we typically verify small properties, this is ok
- There exist tools performing such translation
 - inside SPIN model checker, using option `-f`



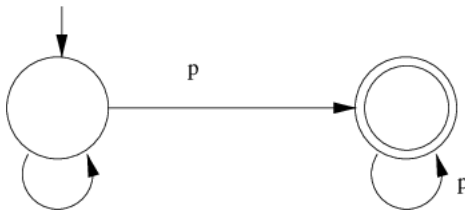
UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



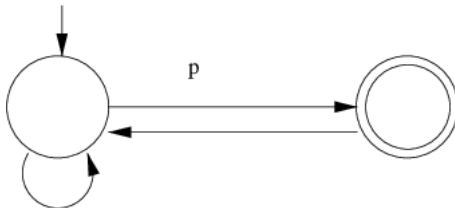
DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

Büchi Automata Examples

Büchi automaton for **FGp**:



Büchi automaton for **GFp**:



UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

LTL Model Checking: Automata-Theoretic Solution

- Given \mathcal{S}, φ decide if $\mathcal{S} \models \varphi$
- Consider \mathcal{S} as a BA where $F = S$
- Then, $\mathcal{S} \models \varphi \equiv \mathcal{L}(\mathcal{S}) \subseteq \mathcal{L}(\varphi)$
- Furthermore, $\equiv \mathcal{L}(\mathcal{S}) \cap \mathcal{L}(\neg\varphi) = \emptyset$
- Finally, $\equiv \mathcal{L}(\mathcal{S} \times \mathcal{A}(\neg\varphi)) = \emptyset$
- The last step is the one which is actually computed
- Complexity is $O(|\mathcal{S}| \cdot |\mathcal{A}(\neg\varphi)|) = O(|\mathcal{S}| \cdot 2^{|\varphi|})$



UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

On-the-Fly LTL Model Checking for $\mathcal{L}(\mathcal{S} \times \mathcal{A}(\neg\varphi)) = \emptyset$

- The graph to be visited is defined as $G = (V, E)$ where:
 - $V = S \times Q$
 - thus, each state is a pair with a state from S and a state from $\mathcal{A}(\neg\varphi)$
 - $((s, q), (s', q')) \in E$ iff $(s, s') \in R$ and $\exists p \in L(s') : \delta(q, p, q')$
 - thus, $\Sigma = AP$
- On such G , we must find *acceptance cycles*
 - an *acceptance state* is (s, q) s.t. $q \in F$
 - we have an *acceptance cycle* if (s, q) is an acceptance state and it is reachable from itself
- If an acceptance cycle is found, we have a counterexample and $\mathcal{S} \not\models \varphi$
- If the visit of G terminates without finding one, $\mathcal{S} \models \varphi$



On-the-Fly LTL Model Checking

- No need for S, Q, R, δ to be in RAM from the beginning
 - similar to Murphi: we have a `next` function directly derived from the input model
 - also $\mathcal{A}(\varphi)$ is described by a suitable language
- Depth-First Visit, easily and efficiently adaptable for finding acceptance cycles
- Namely, *Nested* Depth-First Visit: one for exploring $\mathcal{S} \times \mathcal{A}(\varphi)$, the other to detect cycles
 - the two searches are interleaved
- If an acceptance cycle is found, the DFS stack contains the counterexample



UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica

Nested DFS for LTL Model Checking

```
DFS(KS_BA  $\mathcal{SA}$ , state  $(s, q)$ , bool  $n$ , state  $a$ ) {  
  let  $\mathcal{SA} = \langle S_A, I_A, R_A, L_A \rangle$ ;  
  foreach  $(s', q') \in S_A$  s.t.  $((s, q), (s', q')) \in R_A$  {  
    if  $(n \wedge (s, q) == a)$   
      exit reporting error;  
    if  $((s', q', n) \notin T)$  {  
       $T = T \cup \{(s', q', n)\}$ ;  
      DFS( $\mathcal{SA}$ ,  $(s', q')$ ,  $n$ ,  $a$ );  
      if  $(\neg n \wedge (s', q')$  is accepting) {  
        DFS( $\mathcal{SA}$ ,  $(s', q')$ , true,  $(s', q')$ );  
      }  
    }  
  }  
}
```

```
LTLMC(KS  $S$ , LTL  $\varphi$ ) {  
   $\mathcal{A} = \text{BA\_from\_LTL}(\varphi)$ ;  $T = \emptyset$ ;  
  let  $S = \langle S, I, R, L \rangle$ ,  $\mathcal{A} = \langle \Sigma, Q, \delta, Q_0, F \rangle$ ;  
  foreach  $s \in I, q \in Q_0$   
    DFS( $S \times \mathcal{A}$ ,  $(s, q)$ , false, null);  
}
```



UNIVERSITÀ
DEGLI STUDI
DELL'AQUILA



DISIM
Dipartimento di Ingegneria
e Scienze dell'Informazione
e Matematica