Software Testing and Validation

A.A. 2024/2025

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

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 $\mathcal{S} \models \varphi$, $\langle \mathrm{false}, c \rangle$ 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
  - graph-based: we will see that the one actually used is instead fix-point based
- Then, we will see how they can be efficiently implemented
  - LTL: SPIN and NuSMV
  - CTL: NuSMV

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., $\mathbf{AFAG}p$ has one subformula $\mathbf{AG}p$
- Since we have the full state space S, we label all states $s \in S$ with (sub)formulas holding in s
  - not only the reachable states: all of them
- Then, we use subformulas labeling to decide higher formulas labelling
- Thus, we compute $\lambda : S \to 2^{\mathrm{CTL}}$, being CTL the set of all CTL formulas
- At the end, $\mathcal{S} \models \varphi$ 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
  - $\forall s \in S.\ p \in \lambda(s) \Leftrightarrow p \in L(s)$
  - $\forall s \in S.\ \mathrm{true} \in \lambda(s)$
- Then go upwards in ϕ, using, for each node, the labeling of the sons
  - $\forall s \in S.\ \neg\Phi \in \lambda(s) \Leftrightarrow \Phi \notin \lambda(s)$
  - $\forall s \in S.\ \Phi_1 \wedge \Phi_2 \in \lambda(s) \Leftrightarrow (\Phi_1 \in \lambda(s) \wedge \Phi_2 \in \lambda(s))$
  - $\forall s \in S.\ \mathbf{EX}\Phi \in \lambda(s) \Leftrightarrow (\exists s' : (s, s') \in R \wedge \Phi \in \lambda(s'))$

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

- We already have $\lambda^{-1}(\{\Phi_1\})$ and $\lambda^{-1}(\{\Phi_2\})$
  - here and in the following, $\lambda^{-1}(\{\Phi\}) = \{s \in S \mid \Phi \in \lambda(s)\}$, for a CTL formula Φ
- All states satisfying Φ_2 are ok, let T be the set of such states
- Then, backward visit of the state space of $\mathcal{S}$, starting from T
- The backward visit stops when Φ_1 does not hold
- Complexity is $O(|S| + |R|)$

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

```
labels CheckEU(KS 𝒮, formula Φ₁EUΦ₂, labels λ)
{
  let 𝒮 = ⟨S, I, R, L⟩;
  T = {s ∈ S | Φ₂ ∈ λ(s)};
  foreach s ∈ T
    λ(s) = λ(s) ∪ {Φ₁EUΦ₂};
  while (T ≠ ∅) {
    let s be s.t. s ∈ T;
    T = T \ {s};
    foreach t ∈ {t | (t, s) ∈ R} {
      if Φ₁EUΦ₂ ∉ λ(t) ∧ Φ₁ ∈ λ(t) {
        /* Φ₁EUΦ₂ ∉ λ(t): visited states check */
        λ(t) = λ(t) ∪ {Φ₁EUΦ₂};
        T = T ∪ {t};
  } } }
  return λ;
}
```

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 $\lambda^{-1}(\{\Phi\})$: this defines a subKS $\mathcal{S}'$ of $\mathcal{S}$
  - $\lambda^{-1}(\{\Phi\})$ contains all states in which Φ holds
- Then, compute the strongly connected components (SCCs) of $\mathcal{S}'$
  - inside such components, Φ holds on all states on all paths
- Finally, label with **EG**Φ all s in such SCCs, plus all backward reachable $t \in S'$
  - so we move on states for which Φ holds forever in at least one path...
- Complexity is again $O(|S| + |R|)$

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

```
labels CheckEG(KS 𝒮, formula EGΦ, labels λ)
{
  let 𝒮 = ⟨S, I, R, L⟩;
  S′ = {s ∈ S | Φ ∈ λ(s)}; R′ = {(s, t) ∈ R | s, t ∈ S′};
  𝒜 = SCC(S′, R′); T = ∪_{A∈𝒜} A;
  foreach s ∈ T, λ(s) = λ(s) ∪ {EGΦ};
  while (T ≠ ∅) {
    let s be s.t. s ∈ T;
    T = T \ {s};
    foreach t ∈ {t | (t, s) ∈ R′} {
      if EGΦ ∉ λ(t) { /* since (t, s) ∈ R′, Φ ∈ λ(t) */
        λ(t) = λ(t) ∪ {EGΦ};
        T = T ∪ {t};
  } } }
  return λ;
}
```

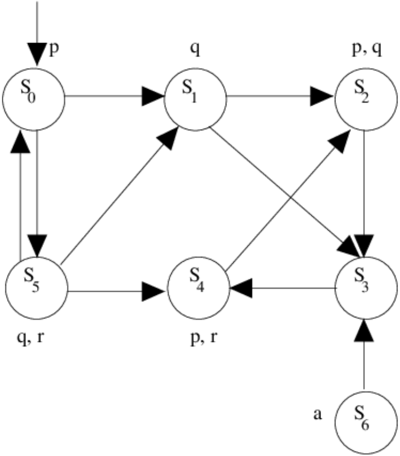
UNIVERSITÀ DEGLI STUDI DELL'AQUILA

DISIM Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

CTL Theoretic Algorithm: Complexity

- Complexity is:
  - $O(|S|)$ for boolean combinations and atomic propositions
  - $O(|S|)$ also for $\mathbf{EX}\Phi$
  - $O(|S| + |R|)$ for $\mathbf{EG}\Phi$ and Φ_1 $\mathbf{EU}$ 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

CTL Model Checking Algorithm Running Example



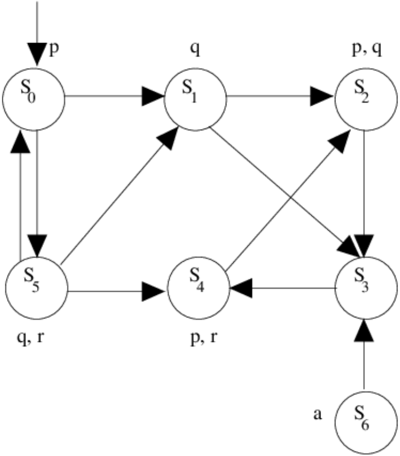
$\varphi = \mathbf{EFAF}p = \text{true}\ \mathbf{EU}(\neg(\mathbf{EG}\neg p))$

Leaves of φ AST are true and p, thus:

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

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

CTL Model Checking Algorithm Running Example

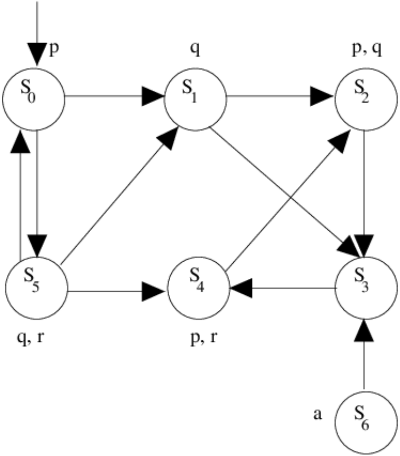


$\varphi = \text{true}\ \mathbf{EU}(\neg(\mathbf{EG}\neg p))$
Going up one level:
$\forall i \in \{0, 2, 4\}.\ \lambda(s_i) = \{\text{true}, p\}$
$\forall i \in \{1, 3, 5, 6\}.\ \lambda(s_i) = \{\text{true}, \neg p\}$,
Going up two levels:
`CheckEG(`$\mathcal{S}$`, `$\mathbf{EG}\neg p$`, `λ`)`

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

```
labels CheckEG(KS 𝒮, formula EGΦ, labels λ)
{
  let 𝒮 = ⟨S, I, R, L⟩;
  S' = {s ∈ S | Φ ∈ λ(s)}; R' = {(s, t) ∈ R | s, t ∈ S'};
  𝒜 = SCC(S', R'); T = ∪_{A∈𝒜 s.t. |A|>1} A;
  foreach s ∈ T, λ(s) = λ(s) ∪ {EGΦ};
  while (T ≠ ∅) {
    let s be s.t. s ∈ T;
    T = T \ {s};
    foreach t ∈ {t | (t, s) ∈ R'} {
      if EGΦ ∉ λ(t) {
        λ(t) = λ(t) ∪ {EGΦ};
        T = T ∪ {t};
  } } }
  return λ;
}
```

CTL Model Checking Algorithm Running Example



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

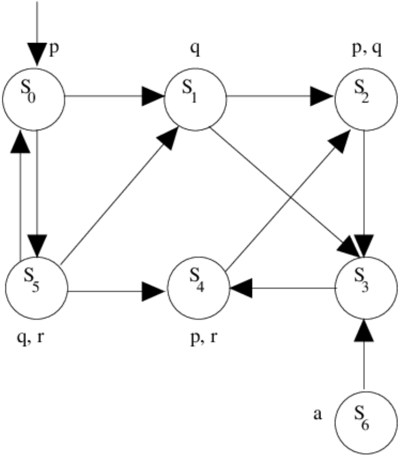
$\texttt{CheckEG}(\mathcal{S},\ \mathbf{EG}\neg p,\ \lambda)$

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

There are no non-trivial SCC on S'

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

CTL Model Checking Algorithm Running Example



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

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

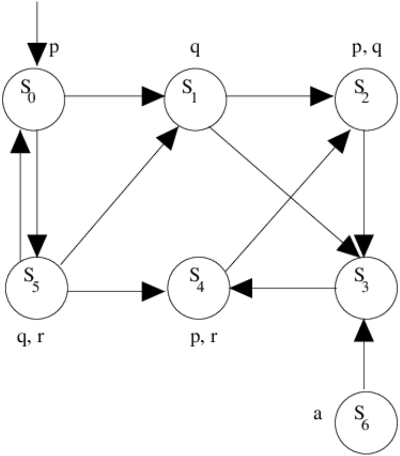
$\forall i \in \{1, 3, 5, 6\}.\ \lambda(s_i) = \{\text{true}, \neg p\}$,

Going up one more level:

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

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

CTL Model Checking Algorithm Running Example



$\varphi = \text{true}\ \mathbf{EU}(\neg(\mathbf{EG}\neg p))$
Finally, call `CheckEU(`$\mathcal{S}$`,` true $\mathbf{EU}(\neg(\mathbf{EG}\neg p)$`,` λ`)`
$T = S$, as all states are labelled with $\neg(\mathbf{EG}\neg p)$
Thus, all states must be labelled with φ

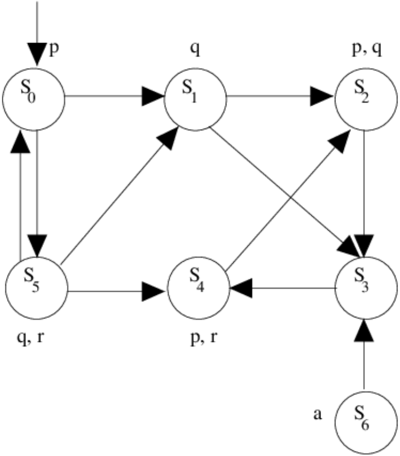
CTL Theoretic Algorithm: $\Phi_1 \mathbf{EU} \Phi_2 \in \lambda(s)$

```
labels CheckEU(KS 𝒮, formula Φ1 EU Φ2, labels λ)
{
  let 𝒮 = ⟨S, I, R, L⟩;
  T = {s ∈ S | Φ2 ∈ λ(s)};
  foreach s ∈ T
    λ(s) = λ(s) ∪ {Φ1 EU Φ2};
  while (T ≠ ∅) {
    let s be s.t. s ∈ T;
    T = T \ {s};
    foreach t ∈ {t | (t, s) ∈ R} {
      if Φ1 EU Φ2 ∉ λ(t) ∧ Φ1 ∈ λ(t) {
        λ(s) = λ(s) ∪ {Φ1 EU Φ2};
        T = T ∪ {t};
  } } }
  return λ;
}
```

UNIVERSITÀ DEGLI STUDI DELL'AQUILA

DISIM Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica

CTL Model Checking Algorithm Running Example



$\varphi = \text{true } \mathbf{EU}(\neg(\mathbf{EG}\neg p))$
$\forall i \in \{0, 2, 4\}.\ \lambda(s_i) = \{\text{true}, p, \neg(\mathbf{EG}\neg p), \varphi\}$
$\forall i \in \{1, 3, 5, 6\}.\ \lambda(s_i) = \{\text{true}, \neg p, \neg(\mathbf{EG}\neg p), \varphi\}$
Since $\varphi \in \lambda(s_0)$, we have that $\mathcal{S} \models \varphi$

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

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

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

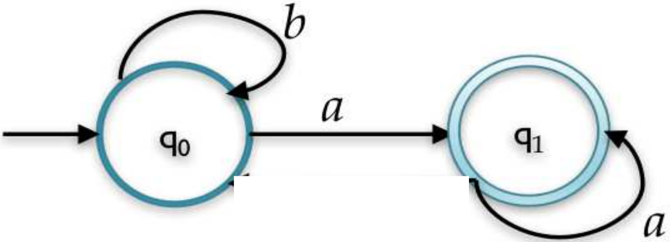
Büchi Automaton

- 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

Language Accepted by Büchi Automata

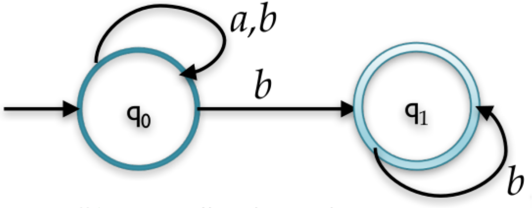
- Let $w = w_0 w_1 \dots$ be an infinite string s.t. $\forall i.\ w_i \in \Sigma$
  - $w \in \Sigma^\omega$
- The BA $\mathcal{A}$ *accepts* w iff there exists a path $\pi = q_0 w_0 q_1 w_1 \dots$ s.t.
  - $\forall i.\ q_i \in Q \wedge w_i \in w \wedge (q_i, w_i, q_{i+1}) \in \delta$
  - $q_0 \in Q_0$
  - 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

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 (infinite set of finite words) containing $\varepsilon, a, aa, aaa, \ldots$
- Note that a^ω is a single infinite word $aaaaaaa\ldots$
- Thus, $b^* a^\omega = \{a^\omega, ba^\omega, bba^\omega, \ldots\}$
- 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, $(a+b)^* b^\omega = \{b^\omega, ab^\omega, abab^\omega, abbabbbab^\omega, \ldots\}$
- That is: any finite sequence of a and b, followed by infinite b's
- Cannot be recognized by a deterministic BA!
    - instead, deterministic FSAs recognize the same languages of non-deterministic FSAs

Büchi Automata and LTL Properties

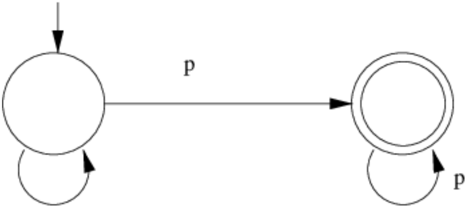
- Also LTL properties are related to infinite words
  - recall that a *model* σ is an infinite sequence of truth assignments to all $p \in AP$
  - by adapting LTL semantics about $\pi \models \varphi$, we can define whether $\sigma \models \varphi$
    - we replace a path state $\pi(i)$ with the set $P_i \subseteq AP$ s.t. $P_i = \{p \in AP \mid p \in L(\pi(i))\}$
- Thus, an LTL property recognizes a language $\mathcal{L}(\varphi) = \{\sigma \in (2^{AP})^\omega \mid \sigma \models \varphi\}$
  - sometimes, we use φ and $P = \mathcal{L}(\varphi)$ interchangeably
- Furthermore, the "infinitely often" part recalls the LTL formula $\mathbf{GF}p$
- Also the "eventually forever" $\mathbf{FG}p$ is important

Büchi Automata and LTL Properties

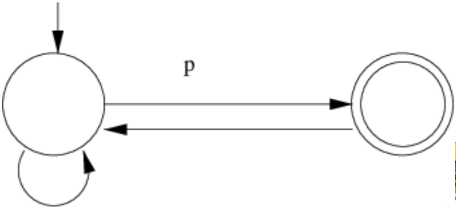
- Let φ be an LTL formula, and let $\mathcal{L}(\varphi)$ be the set of models of φ. Then, there exists a BA $\mathcal{A}_\varphi$ s.t. $\mathcal{L}(\mathcal{A}_\varphi) = \mathcal{L}(\varphi)$
  - it is easy to show that the vice versa does not hold
- We skip the proof, but:
  - of course, we have $\Sigma = 2^{AP}$
  - the size of $\mathcal{A}_\varphi$, 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`

Büchi Automata Examples

Büchi automaton for **FG**p:



Büchi automaton for **GF**p:



LTL Model Checking: Automata-Theoretic Solution

- Given $\mathcal{S}, \varphi$ 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, $\equiv \mathcal{L}(\mathcal{S}) \cap \mathcal{L}(\neg\varphi) = \varnothing$
- Finally, $\equiv \mathcal{L}(\mathcal{S} \times \mathcal{A}(\neg\varphi)) = \varnothing$
- The last step is the one which is actually computed
- Complexity is $O((|\mathcal{S}| \cdot |\mathcal{A}(\neg\varphi)|)^2) = O((|\mathcal{S}| \cdot 2^{|\varphi|})^2)$

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

- 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 $\mathcal{S}$ 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

Nested DFS for LTL Model Checking

```
DFS(KS_BA 𝒮𝒜, state (s,q), bool n, state a) {
  let 𝒮𝒜 = ⟨S_A, I_A, R_A, L_A⟩;
  foreach (s',q') ∈ S_A s.t. ((s,q),(s',q')) ∈ R_A {
    if (n ∧ (s',q') == a)
      exit reporting error;
    if ((s',q',n) ∉ T) {
      T = T ∪ {(s',q',n)};
      DFS(𝒮𝒜, (s',q'), n, a);
      if (¬n ∧ (s',q') is accepting) {
        DFS(𝒮𝒜, (s',q'), true, (s',q'));
} } } }

LTLMC(KS 𝒮, LTL φ) {
  𝒜 = BA_from_LTL(φ); T = ∅;
  let 𝒮 = ⟨S, I, R, L⟩, 𝒜 = ⟨Σ, Q, δ, Q_0, F⟩;
  foreach s ∈ I, q ∈ Q_0
    DFS(𝒮 × 𝒜, (s,q), false, null);
}
```

UNIVERSITÀ DEGLI STUDI DELL'AQUILA

DISIM Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica