

Algorithms for Model Checking (2IW55)

Lecture 2

Fairness & Basic Model Checking Algorithm for CTL and fair CTL
– based on strongly connected components –
Chapter 4.1, 4.2 + SIAM Journal of Computing 1(2), 1972

Tim Willemse

(timw@win.tue.nl)

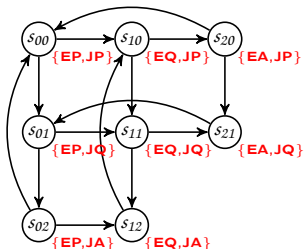
<http://www.win.tue.nl/~timw>

HG 6.81

Outline

- 1 Fairness for CTL
- 2 Strongly Connected Components
- 3 CTL Model Checking Algorithm
- 4 Example: demanding children
- 5 CTL Model Checking with Fairness
- 6 Summary
- 7 Exercise

Temporal Logics: Fairness



- Atomic Propositions: EP, EQ, EA, JP, JQ, JA
- Intended meaning: John or Ella is either Playing, posing Questions, getting Answers
- To exclude that one child gets all attention, we want that both $\neg EQ$ as well as $\neg JQ$ hold infinitely often
- fairness constraints ensuring this: $\mathcal{F} = \{\{s_{00}, s_{01}, s_{02}, s_{20}, s_{21}\}, \{s_{00}, s_{10}, s_{20}, s_{02}, s_{12}\}\}$

Temporal Logics: Fairness

Sometimes properties are violated by “unrealistic” paths only, for instance due to a scheduler. In this case, one may restrict to **fair** paths.

A Kripke Structure over AP **with fairness constraints** is a structure $\mathcal{M} = \langle S, \mathcal{R}, \mathcal{L}, \mathcal{F} \rangle$, where:

- $\langle S, \mathcal{R}, \mathcal{L} \rangle$ is an “ordinary” Kripke Structure as before
- $\mathcal{F} \subseteq 2^S$ is a set of fairness constraints

A **path is fair** if it “hits” each fairness constraint infinitely often:

$\text{fair}(\pi)$ iff $\forall C \in \mathcal{F}. \{i \mid \pi(i) \in C\}$ is an infinite set

Temporal Logics: Fairness

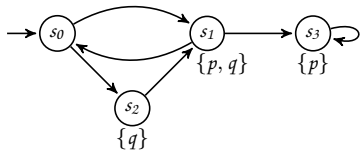
In CTL* with fairness semantics ($\models_{\mathcal{F}}$), only fair paths will be considered.

Given a fixed Kripke Structure with fairness constraints $\mathcal{M} = \langle S, \mathcal{R}, \mathcal{L}, \mathcal{F} \rangle$, $s \models_{\mathcal{F}} f$ means: formula f holds in state s in the fair CTL* semantics.

The definition of $\models_{\mathcal{F}}$ coincides with \models except for the following four clauses:

- $s \models_{\mathcal{F}} \text{true}$ iff there is some fair path starting in s
- $s \models_{\mathcal{F}} p$ iff $p \in \mathcal{L}(s)$ and there is some fair path starting in s
- $s \models_{\mathcal{F}} \mathbf{A} f$ iff for all **fair** paths π starting in s , we have $\pi \models_{\mathcal{F}} f$
- $s \models_{\mathcal{F}} \mathbf{E} f$ iff for some **fair** path π starting in s , we have $\pi \models_{\mathcal{F}} f$

Temporal Logics: Fairness



Note that $s_0 \models \text{E F G } p$, but $s_0 \not\models \text{A F G } p$

- First, consider as Fairness constraint: $\mathcal{F} = \{ \{s_3\} \}$
 - then all fair paths contain s_3 infinitely often
 - we have $s_0 \models_{\mathcal{F}} \text{A F G } p$
- Next, consider as Fairness constraint: $\mathcal{F} = \{ \{s_2\} \}$
 - then all fair paths contain s_2 infinitely often
 - in particular, fair paths cannot contain s_3
 - so $s_0 \not\models_{\mathcal{F}} \text{E F G } p$

Outline

- 1 Fairness for CTL
- 2 Strongly Connected Components**
- 3 CTL Model Checking Algorithm
- 4 Example: demanding children
- 5 CTL Model Checking with Fairness
- 6 Summary
- 7 Exercise

Strongly Connected Components

Given a directed graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$

- let $s \rightarrow_{\mathcal{G}}^* t$ mean that there is a path from node s to t in \mathcal{G}
- a **strongly connected component** (SCC) is a **maximal** subgraph S of \mathcal{G} , such that for all $s, t \in S$, $s \rightarrow_{\mathcal{G}}^* t$ and $t \rightarrow_{\mathcal{G}}^* s$
- an SCC is **non-trivial** if it contains at least one edge

The SCCs of a graph (e.g. a Kripke Structure) can be computed in $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$ time with an algorithm based on depth-first search:

- Text book version (see Introduction to Algorithms, Corben *et al*)
- Tarjan's original algorithm (see SIAM Journal on Computing 1(2), 1972)

The second algorithm is useful in model checking contexts

Strongly Connected Components

Idea behind Tarjan's SCC algorithm

Given is a directed graph $G = \langle \mathcal{V}, \mathcal{E} \rangle$

- compute **spanning trees** by depth-first search; **number** the nodes in the order they are visited
- the other, non-tree edges are either:
 - **forward** edges (can be ignored)
 - **backward** edges (to an ancestor)
 - **cross** edges (to another subtree)

backward and cross edges lead to nodes with **smaller** numbers

- nodes are kept on a **stack**; the nodes of a discovered SCC will be popped immediately from this stack
- compute $root[v]$: the smallest node which is:
 - reachable from v by a sequence of tree-edges followed by at most **one non-tree edge**; and
 - if $root[v] = v$, the root of a new SCC is found, and the whole SCC is popped from the stack

Strongly Connected Components

Procedure `find_scc` applies a repeated depth-first search on yet unprocessed nodes of the input graph $\mathcal{G} = \langle \mathcal{V}, \mathcal{E} \rangle$

The depth-first search is delegated to the procedure `dfs_scc`.

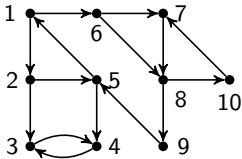
```
procedure find_scc
   $i := 0$ ;
  empty the stack;
  leave all nodes unnumbered;
  for vertices  $w \in \mathcal{V}$  do
    if  $w$  is not yet numbered then
      dfs_scc( $w$ );
    end if
  end for
end procedure
```

Strongly Connected Components

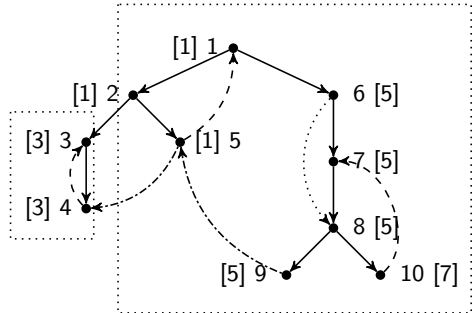
```
procedure dfs_scc(v)  
  root[v] := number[v] := i := i + 1;  
  push v on the stack;  
  for successor w of v do  
    if w is not yet numbered then                                     {tree edge}  
      dfs_scc(w);  
      root[v] := min(root[v], root[w]);  
    else if number[w] < number[v] and w on the stack then      {cross/back edge}  
      root[v] := min(root[v], number[w]);  
    end if  
  end for  
  if root[v] = number[v] then                                       {start new SCC}  
    while top of stack satisfies number(w) ≥ number(v) do  
      pop w from stack;  
    end while  
  end if  
end procedure
```

Strongly Connected Components

Example: SCC algorithm



A possible run of the SCC algorithm, with DFS node numbers, final root-values (in square brackets), tree edges (plain arrow), forward edges (dotted), back edges (dashed), cross edges (dash/dot). **Two SCCs are found: number and root value are equal**



Strongly Connected Components

We analyse the space and time requirements for running `find_scc` on a graph $G = \langle \mathcal{V}, \mathcal{E} \rangle$:

- for every node:
 - `dfs_scc` is called exactly once
 - all its outgoing edges are explored exactly once
- each node is pushed and popped from the stack exactly once
- checking whether a node is on the stack can be done in constant time, for instance by maintaining a Boolean array

Conclusion: Tarjan's algorithm for finding strongly connected components runs in time and space $\mathcal{O}(|\mathcal{V}| + |\mathcal{E}|)$

Outline

- 1 Fairness for CTL
- 2 Strongly Connected Components
- 3 CTL Model Checking Algorithm**
- 4 Example: demanding children
- 5 CTL Model Checking with Fairness
- 6 Summary
- 7 Exercise

CTL Model Checking Algorithm

Recall that CTL has the following ten temporal operators:

- $A X$ and $E X$: for all/some next state
- $A F$ and $E F$: inevitably and potentially
- $A G$ and $E G$: invariantly and potentially always
- $A [U]$ and $E [U]$: for all/some paths, until
- $A [R]$ and $E [R]$: for all/some paths, releases

Besides atomic propositions (AP), the constant true and the Boolean connectives (\neg, \vee), the following temporal operators are sufficient: $E X, E G, E [U]$.

Hence: only algorithms for computing formulae of the above form are needed.

CTL Model Checking Algorithm

Main loop of model checking CTL: check formula f on a Kripke Structure $\langle S, \mathcal{R}, \mathcal{L} \rangle$.

By recursion on f , algorithm $\text{mc_ctl}(f)$ computes $\text{label}(s)$ for all states $s \in S$, where $\text{label}(s)$ shall contain those subformulae of f that hold in s .

Algorithm $\text{mc_ctl}(f)$ employs a case distinction on the structure of f :

$f = p$	add p to $\text{label}(s)$ for those states s with $p \in \mathcal{L}(s)$
$f = g_0 \vee g_1$	$\text{mc_ctl}(g_0)$; $\text{mc_ctl}(g_1)$; add f to all states labelled with g_0 or g_1
$f = \neg g$	$\text{mc_ctl}(g)$; add f to all states not labelled with g
$f = E X g$	$\text{mc_ctl}(g)$; add f to all states with an \mathcal{R} -successor labelled by g
$f = E [g_0 U g_1]$	$\text{mc_ctl}(g_0)$; $\text{mc_ctl}(g_1)$; check_eu(g_0, g_1)
$f = E G g$	$\text{mc_ctl}(g)$; check_eg(g)

Upon termination, $s \models f$ if and only if $f \in \text{label}(s)$

CTL Model Checking Algorithm

```
procedure check_eu(f,g)
   $\mathcal{T} := \{s \mid g \in \text{label}(s)\};$ 
  for all  $s \in \mathcal{T}$  do  $\text{label}(s) := \text{label}(s) \cup \{E[f \text{ U } g]\};$ 
  end for
  while  $\mathcal{T} \neq \emptyset$  do
    choose  $s \in \mathcal{T}$ ;
     $\mathcal{T} := \mathcal{T} \setminus \{s\}$ ;
    for all  $t$  satisfying  $t \mathcal{R} s$  do
      if  $E[f \text{ U } g] \notin \text{label}(t)$  and  $f \in \text{label}(t)$  then
         $\text{label}(t) := \text{label}(t) \cup E[f \text{ U } g]$ ;
         $\mathcal{T} := \mathcal{T} \cup \{t\}$ ;
      end if
    end for
  end while
end procedure
```

Observations:

- label all states where g holds
- search backwards over states where f holds

CTL Model Checking Algorithm

```
procedure check_eg(f)
  S' := {s | f ∈ label(s)};
  SCC := {C | C is a nontrivial SCC of S'};
  T := ⋃_{C ∈ SCC} {s | s ∈ C};
  for all s ∈ T do label(s) := label(s) ∪ {E G f};
  end for
  while T ≠ ∅ do
    choose s ∈ T;
    T := T \ {s};
    for all t satisfying t ∈ S' and t R s do
      if E G f ∉ label(t) then
        label(t) := label(t) ∪ {E G f};
        T := T ∪ {t};
      end if
    end for
  end while
end procedure
```

Observations:

- restrict attention to subgraph where f holds
- an infinite path in a finite graph eventually reaches a non-trivial SCC

CTL Model Checking Algorithm

We analyse the time complexity for the standard CTL model checking algorithm of formula f (with $|f|$ the number of subformulae) on Kripke Structure $\mathcal{M} = \langle \mathcal{S}, \mathcal{R}, \mathcal{L} \rangle$.

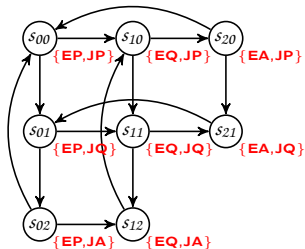
- There are at most $|f|$ calls to `mc_ctl`
- Backward reachability and detecting strongly connected components can be done in time linear to the Kripke Structure: $\mathcal{O}(|\mathcal{S}| + |\mathcal{R}|)$
- Hence, each recursive call takes at most $\mathcal{O}(|\mathcal{S}| + |\mathcal{R}|)$ time

So, the complexity of this CTL model checking algorithm is $\mathcal{O}(|f| \cdot (|\mathcal{S}| + |\mathcal{R}|))$, which is **linear** in both the formula and the state space.

Outline

- 1 Fairness for CTL
- 2 Strongly Connected Components
- 3 CTL Model Checking Algorithm
- 4 Example: demanding children**
- 5 CTL Model Checking with Fairness
- 6 Summary
- 7 Exercise

Example: demanding children

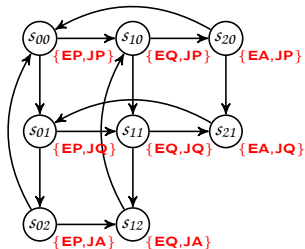


- Atomic Propositions: EP, EQ, EA, JP, JQ, JA
- Intended meaning: John or Ella is either Playing, posing Questions, getting Answers

Requirement: Whenever John asks a question, he eventually gets an answer

Formula: $A G (JQ \rightarrow A F JA)$

Example: demanding children



- Atomic Propositions: EP, EQ, EA, JP, JQ, JA
- Intended meaning: John or Ella is either Playing, posing Questions, getting Answers

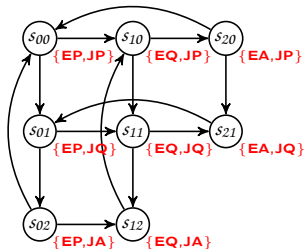
- Step 1: express using basic operators

$$A G (JQ \rightarrow A F JA)$$

$$\equiv$$

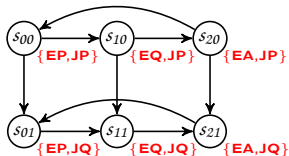
$$\neg E [true U \neg(\neg JQ \vee \neg E G \neg JA)]$$

Example: demanding children



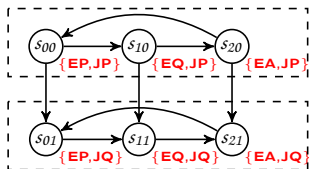
- Step 2: treat $E \ G \ \neg \mathcal{A}$
 - Restrict to the subgraph where $\neg \mathcal{A}$ holds
 - Find non-trivial SCCs
 - Backward reachability

Example: demanding children



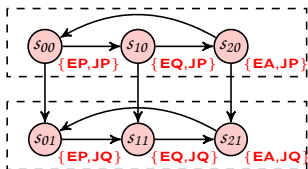
- Step 2: treat $E \ G \ \neg \mathcal{A}$
 - Restrict to the subgraph where $\neg \mathcal{A}$ holds
 - Find non-trivial SCCs
 - Backward reachability

Example: demanding children



- Step 2: treat $E \ G \ \neg \mathcal{A}$
 - Restrict to the subgraph where $\neg \mathcal{A}$ holds
 - Find non-trivial SCCs
 - Backward reachability

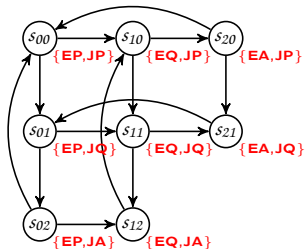
Example: demanding children



- Step 2: treat $E G \neg \mathcal{A}$
 - Restrict to the subgraph where $\neg \mathcal{A}$ holds
 - Find non-trivial SCCs
 - Backward reachability

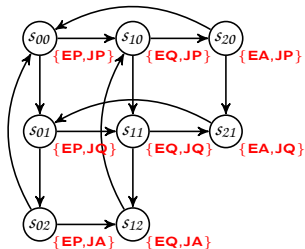
No new states are found. So, $E G \neg \mathcal{A}$ holds in the states $\{s_{00}, s_{10}, s_{20}, s_{01}, s_{11}, s_{21}\}$;

Example: demanding children



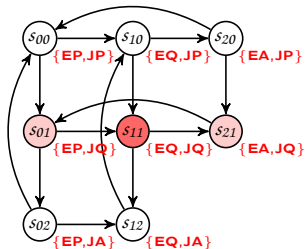
- Step 3: treat $\neg E \text{ G } \neg \mathcal{J}\mathcal{A}$
 - $E \text{ G } \neg \mathcal{J}\mathcal{A}$ holds in $\{s_{00}, s_{10}, s_{20}, s_{01}, s_{11}, s_{21}\}$, so $\neg E \text{ G } \neg \mathcal{J}\mathcal{A}$ holds in $\{s_{02}, s_{12}\}$
- Step 4: treat $\neg \mathcal{J}Q$
 - $\neg \mathcal{J}Q$ holds in $\{s_{00}, s_{10}, s_{20}, s_{02}, s_{12}\}$
- Step 5: treat $\neg \mathcal{J}Q \vee \neg E \text{ G } \neg \mathcal{J}\mathcal{A}$
 - $\neg \mathcal{J}Q \vee \neg E \text{ G } \neg \mathcal{J}\mathcal{A}$ holds in $\{s_{00}, s_{10}, s_{20}, s_{02}, s_{12}\} \cup \{s_{02}, s_{12}\} = \{s_{00}, s_{10}, s_{20}, s_{02}, s_{12}\}$

Example: demanding children



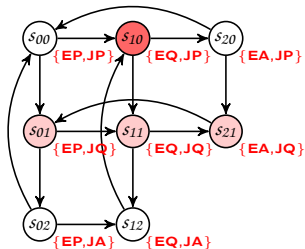
- **Step 6:** treat $\neg(\neg JQ \vee \neg E G \neg JA)$
 - $\neg JQ \vee \neg E G \neg JA$ holds in $\{s_{00}, s_{10}, s_{20}, s_{02}, s_{12}\}$,
so $\neg(\neg JQ \vee \neg E G \neg JA)$ holds in $\{s_{01}, s_{11}, s_{21}\}$
- **Step 7:** compute $E[\text{true} \cup \neg(\neg JQ \vee \neg E G \neg JA)]$
 - Start in $\{s_{01}, s_{11}, s_{21}\}$
 - Perform a backward reachability analysis over states for which true holds

Example: demanding children



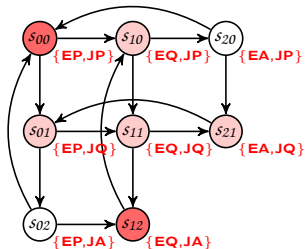
- **Step 6:** treat $\neg(\neg JQ \vee \neg E G \neg JA)$
 - $\neg JQ \vee \neg E G \neg JA$ holds in $\{s_{00}, s_{10}, s_{20}, s_{02}, s_{12}\}$,
so $\neg(\neg JQ \vee \neg E G \neg JA)$ holds in $\{s_{01}, s_{11}, s_{21}\}$
- **Step 7:** compute $E[\text{true} \cup \neg(\neg JQ \vee \neg E G \neg JA)]$
 - Start in $\{s_{01}, s_{11}, s_{21}\}$
 - Perform a backward reachability analysis over states for which true holds

Example: demanding children



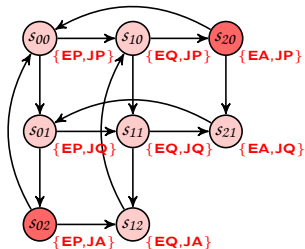
- **Step 6:** treat $\neg(\neg JQ \vee \neg E G \neg JA)$
 - $\neg JQ \vee \neg E G \neg JA$ holds in $\{s_{00}, s_{10}, s_{20}, s_{02}, s_{12}\}$,
so $\neg(\neg JQ \vee \neg E G \neg JA)$ holds in $\{s_{01}, s_{11}, s_{21}\}$
- **Step 7:** compute $E[\text{true} \cup \neg(\neg JQ \vee \neg E G \neg JA)]$
 - Start in $\{s_{01}, s_{11}, s_{21}\}$
 - Perform a backward reachability analysis over states for which true holds

Example: demanding children



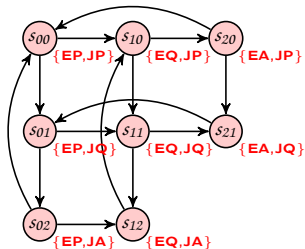
- **Step 6:** treat $\neg(\neg JQ \vee \neg E G \neg JA)$
 - $\neg JQ \vee \neg E G \neg JA$ holds in $\{s_{00}, s_{10}, s_{20}, s_{02}, s_{12}\}$,
so $\neg(\neg JQ \vee \neg E G \neg JA)$ holds in $\{s_{01}, s_{11}, s_{21}\}$
- **Step 7:** compute $E[\text{true} \cup \neg(\neg JQ \vee \neg E G \neg JA)]$
 - Start in $\{s_{01}, s_{11}, s_{21}\}$
 - Perform a backward reachability analysis over states for which true holds

Example: demanding children



- *Step 6: treat $\neg(\neg JQ \vee \neg E G \neg JA)$*
 - $\neg JQ \vee \neg E G \neg JA$ holds in $\{s_{00}, s_{10}, s_{20}, s_{02}, s_{12}\}$,
so $\neg(\neg JQ \vee \neg E G \neg JA)$ holds in $\{s_{01}, s_{11}, s_{21}\}$
- *Step 7: compute $E[\text{true} \cup \neg(\neg JQ \vee \neg E G \neg JA)]$*
 - Start in $\{s_{01}, s_{11}, s_{21}\}$
 - Perform a backward reachability analysis over states for which true holds

Example: demanding children



- **Step 6:** treat $\neg(\neg JQ \vee \neg E G \neg JA)$
 - $\neg JQ \vee \neg E G \neg JA$ holds in $\{s_{00}, s_{10}, s_{20}, s_{02}, s_{12}\}$,
so $\neg(\neg JQ \vee \neg E G \neg JA)$ holds in $\{s_{01}, s_{11}, s_{21}\}$
- **Step 7:** compute $E[\text{true} \cup \neg(\neg JQ \vee \neg E G \neg JA)]$
 - Start in $\{s_{01}, s_{11}, s_{21}\}$
 - Perform a backward reachability analysis over states for which true holds

Example: demanding children

Conclusion:

- So, $E [\text{true } U \neg(\neg \mathcal{I}Q \vee \neg E G \neg \mathcal{I}A)]$ holds in all states
- Hence, its negation $A G (\mathcal{I}Q \rightarrow A F \mathcal{I}A)$ holds in no state
- The requirement does not hold for the full Kripke Structure
- Why? Because in this case, there is a path in which only Ella progresses while John is not being served.
- Next, we look at the Kripke Structure with Fairness Constraints

Outline

- 1 Fairness for CTL
- 2 Strongly Connected Components
- 3 CTL Model Checking Algorithm
- 4 Example: demanding children
- 5 CTL Model Checking with Fairness**
- 6 Summary
- 7 Exercise

CTL Model Checking with Fairness

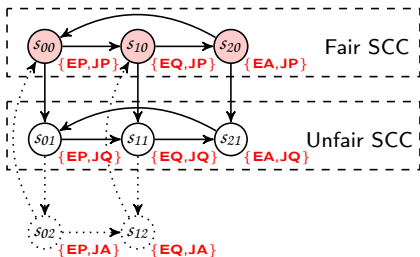
Recall: Kripke Structure $\mathcal{M} = \langle S, \mathcal{R}, \mathcal{L}, \mathcal{F} \rangle$ with fairness constraints $\mathcal{F} \subseteq 2^S$.

- A **path is fair** if it “hits” each fairness constraint infinitely often
- A **fair** SCC is an SCC that contains an element from each constraint $C \in \mathcal{F}$

Main idea of fair model checking for CTL:

- Special treatment for $s \models_{\mathcal{F}} E G f$: **check_fair_eg**
 - Restrict attention to $S' \subseteq S$ where f holds
 - Find a path to a **fair** non-trivial SCC in S'
- Label states where $E G$ true **fairly** holds with a new proposition symbol **fair**
- Treat the other operators using the original “unfair” procedures:
 - $s \models_{\mathcal{F}} p$ $s \models p \wedge \mathit{fair}$
 - $s \models_{\mathcal{F}} E X f$ $s \models E X (f \wedge \mathit{fair})$
 - $s \models_{\mathcal{F}} E [f U g]$ $s \models E [f U (g \wedge \mathit{fair})]$

CTL Model Checking with Fairness



- Assume fairness constraints $\neg EQ$ and $\neg JQ$.
- Remark: full graph is one big fair SCC, so $E G$ true holds everywhere

- $E G \neg JA$:
 - Restrict to subgraph with $\neg JA$
 - Find **fair** non-trivial SCCs
 - Do backward reachability
- Hence: $JQ \wedge E G \neg JA$ holds fairly in **NO** state
- Hence $E F (JQ \wedge E G \neg JA)$ holds nowhere fairly
- Hence, its negation, the requirement $A G (JQ \rightarrow A F JA)$ fairly holds everywhere!

Outline

- 1 Fairness for CTL
- 2 Strongly Connected Components
- 3 CTL Model Checking Algorithm
- 4 Example: demanding children
- 5 CTL Model Checking with Fairness
- 6 Summary**
- 7 Exercise

Summary

CTL model checking:

- SCC algorithm is used
- Tarjan's SCC algorithm runs one depth-first search, computing SCCs **on-the-fly**. Time complexity is linear
- CTL model checking can be done in time linear in the size of the formula as well as in the Kripke Structure
- Extension with Fairness Constraints is straightforward and is useful in practice
- Why not treat fairness in formulae?

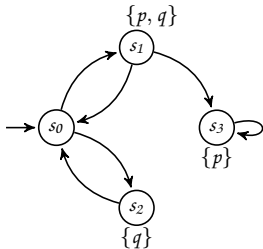
$$A [(G F C_1 \wedge G F C_2) \rightarrow \textit{Requirement}]$$

- fairness cannot be expressed in CTL
- for LTL all known algorithms are exponential in the size of the formula

Outline

- 1 Fairness for CTL
- 2 Strongly Connected Components
- 3 CTL Model Checking Algorithm
- 4 Example: demanding children
- 5 CTL Model Checking with Fairness
- 6 Summary
- 7 Exercise

Exercise



CTL formulae: p , $E [q R p]$, $A G E F p$,
 $A G p \vee A F q$

- Determine for each formula in which states of the above Kripke Structure it holds; use both the semantics and use the appropriate algorithms
- Extend the Kripke structure with the Fairness constraints $\mathcal{F} = \{ \{s_1\}, \{s_2\} \}$. In which states do the above formulae *fairly* hold?
- Similarly for the Fairness constraint $\mathcal{F} = \{ \{s_3\} \}$