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

Fairness for CTL

Strongly Connected Components

CTL Model Checking Algorithm

Example: demanding children

CTL Model Checking with Fairness

Summary

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 ¬EQ as well as ¬JQ hold infinitely often

fairness constraints ensuring this:
\[ F = \{\{s_{00}, s_{01}, s_{02}, s_{20}, s_{21}\}, \{s_{00}, s_{10}, s_{20}, s_{02}, s_{12}\}\} \]
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 \( M = \langle S, R, L, F \rangle \), where:

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

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

\[
\text{fair}(\pi) \text{ iff } \forall C \in F. \{i \mid \pi(i) \in C\} \text{ is an infinite set}
\]
In $\text{CTL}^*$ with fairness semantics ($\models_{F}$), only fair paths will be considered.

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

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

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

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

- First, consider as Fairness constraint: $F = \{ \{s_3\}\}$
  - then all fair paths contain $s_3$ infinitely often
  - we have $s_0 \models_F A F G p$

- Next, consider as Fairness constraint: $F = \{ \{s_2\}\}$
  - then all fair paths contain $s_2$ infinitely often
  - in particular, fair paths cannot contain $s_3$
  - so $s_0 \not\models_F E F G p$
Outline

Fairness for CTL

Strongly Connected Components

CTL Model Checking Algorithm

Example: demanding children

CTL Model Checking with Fairness

Summary

Exercise
Strongly Connected Components

Given a directed graph $G = \langle V, E \rangle$
- let $s \rightarrow^*_G t$ mean that there is a path from node $s$ to $t$ in $G$
- a strongly connected component (SCC) is a maximal subgraph $S$ of $G$, such that for all $s, t \in S$, $s \rightarrow^*_G t$ and $t \rightarrow^*_G s$
- 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 $O(|V| + |E|)$ 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 most useful in model checking contexts
Strongly Connected Components

Idea behind Tarjan’s SCC algorithm
Given is a directed graph $G = \langle V, E \rangle$

- 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 $G = \langle V, E \rangle$
The depth-first search is delegated to the procedure DFS_SCC.

```
procedure FIND_SCC
    i := 0;
    empty the stack;
    leave all nodes unnumbered;
    for vertice $w \in V$ do
        if $w$ is not yet numbered then
            DFS_SCC($w$);
        end if
    end for
end procedure
```
procedure DFS_SCC(v)

\[ \text{root}[v] := \text{number}[v] := i := i + 1; \]
push \( v \) on the stack;
for successor \( w \) of \( v \) do
  if \( w \) is not yet numbered then
    DFS_SCC(w);
    \[ \text{root}[v] := \min(\text{root}[v], \text{root}[w]); \]
  else if \( \text{number}[w] < \text{number}[v] \) and \( w \) on the stack then
    \[ \text{root}[v] := \min(\text{root}[v], \text{number}[w]); \]
  end if
end for
if \( \text{root}[v] = \text{number}[v] \) then
  while top \( w \) of stack satisfies \( \text{number}(w) \geq \text{number}(v) \) 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.
We analyse the space and time requirements for running \texttt{FIND\_SCC} on a graph \(G = \langle V, E \rangle\):

- for every node:
  - \texttt{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 \(O(|V| + |E|)\)
Outline

- Fairness for CTL
- Strongly Connected Components
- CTL Model Checking Algorithm
- Example: demanding children
- CTL Model Checking with Fairness
- Summary
- Exercise
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.
Main loop of model checking CTL: check formula $f$ on a Kripke Structure $\langle S, R, L \rangle$.

By recursion on $f$, algorithm $MC_{\text{CTL}}(f)$ 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 $MC_{\text{CTL}}(f)$ employs a case distinction on the structure of $f$:

<table>
<thead>
<tr>
<th>$f$</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f = p$</td>
<td>add $p$ to $\text{label}(s)$ for those states $s$ with $p \in L(s)$</td>
</tr>
<tr>
<td>$f = g_0 \lor g_1$</td>
<td>$MC_{\text{CTL}}(g_0)$; $MC_{\text{CTL}}(g_1)$; add $f$ to all states labelled with $g_0$ or $g_1$</td>
</tr>
<tr>
<td>$f = \neg g$</td>
<td>$MC_{\text{CTL}}(g)$; add $f$ to all states not labelled with $g$</td>
</tr>
<tr>
<td>$f = E \times g$</td>
<td>$MC_{\text{CTL}}(g)$; add $f$ to all states with an $R$-successor labelled by $g$</td>
</tr>
<tr>
<td>$f = E [g_0 \lor g_1]$</td>
<td>$MC_{\text{CTL}}(g_0)$; $MC_{\text{CTL}}(g_1)$; $\text{CHECK_EU}(g_0, g_1)$</td>
</tr>
<tr>
<td>$f = E \bigcirc g$</td>
<td>$MC_{\text{CTL}}(g)$; $\text{CHECK_EG}(g)$</td>
</tr>
</tbody>
</table>

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

Observations:
- label all states where \( g \) holds
- search backwards over states where \( f \) holds
procedure CHECK_EG(f)
S' := \{s \mid f \in \text{label}(s)\};
SCC := \{C \mid C \text{ is a nontrivial SCC of } S'\};
T := \bigcup_{C \in SCC} \{s \mid s \in C\};
for all \(s \in T\) do
    \text{label}(s) := \text{label}(s) \cup \{E G f\};
end for
while \(T \neq \emptyset\) do
    choose \(s \in T\);
    T := T \setminus \{s\};
    for all \(t\) satisfying \(t \in S'\) and \(t \mathbin{R} s\) do
        if \(E G f \notin \text{label}(t)\) then
            \text{label}(t) := \text{label}(t) \cup \{E G f\};
            T := T \cup \{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
We analyse the time complexity for the standard CTL model checking algorithm of formula $f$ (with $|f|$ the number of subformulae) on Kripke Structure $M = \langle S, R, L \rangle$.

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

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

Fairness for CTL

Strongly Connected Components

CTL Model Checking Algorithm

**Example: demanding children**

CTL Model Checking with Fairness

Summary

Exercise
Example: demanding children

- Intended meaning: John or Ella is either Playing, posing Questions, getting Answers

Requirement: Whenever John asks a question, he eventually gets an answer
Formula: \( A \square (JQ \rightarrow A F JA) \)
Example: demanding children

- Intended meaning: John or Ella is either Playing, posing Questions, getting Answers

Step 1: express using basic operators

\[ A \bigwedge (JQ \rightarrow A F JA) \]
\[ \equiv \]
\[ \neg E [\text{true} U \neg(\neg JQ \lor \neg E G \neg JA)] \]
Example: demanding children

- Step 2: treat $E G \neg J A$
  - Restrict to the subgraph where $\neg J A$ holds
  - Find non-trivial SCCs
  - Backward reachability
Example: demanding children

- **Step 2: treat** $E \ G \ \neg J A$
  - **Restrict to the subgraph where** $\neg J A$ **holds**
  - **Find non-trivial SCCs**
  - **Backward reachability**
Example: demanding children

- ▶ Step 2: treat $EG \neg J A$
  - Restrict to the subgraph where $\neg J A$ holds
  - Find non-trivial SCCs
  - Backward reachability
Example: demanding children

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

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

\[ \{EP ,JP\} \quad \{EQ ,JP\} \quad \{EA ,JP\} \]

\[ \{EP ,JQ\} \quad \{EQ, JQ\} \quad \{EA, JQ\} \]

\[ \{EP, JA\} \quad \{EQ, JA\} \]

\[ \{EP, JG\} \quad \{EQ, JG\} \quad \{EA, JG\} \]

\[ \text{Step 3: treat } \neg E \ G \ \neg J A \]

\[ \quad \text{• } E \ G \ \neg J A \text{ holds in } \{s_{00}, s_{10}, s_{20}, s_{01}, s_{11}, s_{21}\}, \text{ so } \neg E \ G \ \neg J A \text{ holds in } \{s_{02}, s_{12}\} \]

\[ \text{Step 4: treat } \neg J Q \]

\[ \quad \text{• } \neg J Q \text{ holds in } \{s_{00}, s_{10}, s_{20}, s_{02}, s_{12}\} \]

\[ \text{Step 5: treat } \neg J Q \lor \neg E \ G \ \neg J A \]

\[ \quad \text{• } \neg J Q \lor \neg E \ G \ \neg J A \text{ holds in } \{s_{00}, s_{10}, s_{20}, s_{02}, s_{12}\} \cup \{s_{02}, s_{12}\} = \{s_{00}, s_{10}, s_{20}, s_{02}, s_{12}\} \]
Example: demanding children

Step 6: treat \( \neg((\neg JQ \lor \neg E \ G \neg J\ A) \) 

- \( \neg JQ \lor \neg E \ G \neg J\ A \) holds in \( \{s_{00}, s_{10}, s_{20}, s_{02}, s_{12}\} \), so \( \neg(\neg JQ \lor \neg E \ G \neg J\ A) \) holds in \( \{s_{01}, s_{11}, s_{12}\} \)

Step 7: compute \( E [true \ U \neg((\neg JQ \lor \neg E \ G \neg J\ A)] \) 

- Start in \( \{s_{01}, s_{11}, s_{12}\} \) 
- Perform a backward reachability analysis over states for which true holds
Example: demanding children

▶ Step 6: treat $\neg (\neg JQ \lor \neg \mathbf{E} G \neg JA)$
  - $\neg JQ \lor \neg \mathbf{E} G \neg JA$ holds in $\{s_{00}, s_{10}, s_{20}, s_{02}, s_{12}\}$, so $\neg (\neg JQ \lor \neg \mathbf{E} G \neg JA)$ holds in $\{s_{01}, s_{11}, s_{12}\}$

▶ Step 7: compute $\mathbf{E} \left[ \mathbf{true} \ U \neg (\neg JQ \lor \neg \mathbf{E} G \neg JA) \right]$
  - Start in $\{s_{01}, s_{11}, s_{12}\}$
  - Perform a backward reachability analysis over states for which true holds
Example: demanding children

Step 6: treat \( \neg (\neg JQ \lor \neg E \ G \neg JA) \)
- \( \neg JQ \lor \neg E \ G \neg JA \) holds in \( \{s_{00}, s_{10}, s_{20}, s_{02}, s_{12}\} \)
  so \( \neg (\neg JQ \lor \neg E \ G \neg JA) \) holds in \( \{s_{01}, s_{11}, s_{12}\} \)

Step 7: compute \( E [true \ U \ (\neg JQ \lor \neg E \ G \neg JA)] \)
- Start in \( \{s_{01}, s_{11}, s_{12}\} \)
- Perform a backward reachability analysis over states for which \( true \) holds
Example: demanding children

▶ Step 6: treat $\neg(\neg JQ \lor \neg E G \neg JA)$
  • $\neg JQ \lor \neg E G \neg JA$ holds in $\{s_{00}, s_{10}, s_{20}, s_{02}, s_{12}\}$, so $\neg(\neg JQ \lor \neg E G \neg JA)$ holds in $\{s_{01}, s_{11}, s_{12}\}$

▶ Step 7: compute $E [true U \neg(\neg JQ \lor \neg E G \neg JA)]$
  • Start in $\{s_{01}, s_{11}, s_{12}\}$
  • Perform a backward reachability analysis over states for which true holds
Example: demanding children

- **Step 6:** treat \( \neg (\neg JQ \lor \neg E \ G \neg JA) \)
  - \( \neg JQ \lor \neg E \ G \neg JA \) holds in \( \{s_{00}, s_{10}, s_{20}, s_{02}, s_{12}\} \),
  - so \( \neg (\neg JQ \lor \neg E \ G \neg JA) \) holds in \( \{s_{01}, s_{11}, s_{12}\} \)

- **Step 7:** compute \( E [\text{true} \ U \ (\neg JQ \lor \neg E \ G \neg JA)] \)
  - Start in \( \{s_{01}, s_{11}, s_{12}\} \)
  - Perform a backward reachability analysis over states for which true holds
Example: demanding children

- **Step 6:** treat $\neg (\neg JQ \lor \neg E \ G \neg JA)$
  - $\neg JQ \lor \neg E \ G \neg JA$ holds in $\{s_{00}, s_{10}, s_{20}, s_{02}, s_{12}\}$, so $\neg (\neg JQ \lor \neg E \ G \neg JA)$ holds in $\{s_{01}, s_{11}, s_{12}\}$

- **Step 7:** compute $E[true \ U \ (\neg JQ \lor \neg E \ G \neg JA)]$
  - Start in $\{s_{01}, s_{11}, s_{12}\}$
  - Perform a backward reachability analysis over states for which true holds
Example: demanding children

Conclusion:

- So, $\text{E} \left[ \text{true} \ U \left( \neg JQ \lor \neg \text{E} \ G \ (\neg J A) \right) \right]$ holds in all states
- Hence, its negation $\text{A} \ G \ (JQ \rightarrow \text{A} \ F \ J A)$ 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

Fairness for CTL

Strongly Connected Components

CTL Model Checking Algorithm

Example: demanding children

CTL Model Checking with Fairness

Summary

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

- 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 $C \in F$

Main idea of fair model checking for CTL:

- Special treatment for $s \models_F E G f$: 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:
  - $s \models_F p$ .......................................................... $s \models p \land \text{fair}$
  - $s \models_F E X f$ ...................................................... $s \models E X (f \land \text{fair})$
  - $s \models_F E [f U g]$ .............................................. $s \models E [f U (g \land \text{fair})]$
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: $JQ \land E G \neg JA$ holds fairly in NO state
Hence $E F (JQ \land E G \neg JA)$ holds nowhere fairly
Hence, its negation, the requirement $A G (JQ \rightarrow A F JA)$ fairly holds everywhere!
Outline

Fairness for CTL

Strongly Connected Components

CTL Model Checking Algorithm

Example: demanding children

CTL Model Checking with Fairness

Summary

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?

\[ A [(G F C_1 \land G F C_2) \rightarrow Requirement] \]

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

- Fairness for CTL
- Strongly Connected Components
- CTL Model Checking Algorithm
- Example: demanding children
- CTL Model Checking with Fairness
- Summary
- Exercise
Exercise

CTL formulae:

- $p$
- $E [q R p]$
- $A G E F p$
- $A ((G p) ∨ (F q))$

- 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 $F = \{ \{s_1\}, \{s_2\} \}$. In which states do the above formulae fairly hold?
- Similarly for the Fairness constraint $F = \{ \{s_3\} \}$