Restricted LTL

- Consider the LTL subset

$$\phi ::= p \mid \phi \wedge \phi \mid \neg\phi \mid \mathsf{X}\,\phi \mid \phi\,\mathsf{U}\,\phi$$

- The subformulas of a formula are:

$$\mathrm{SF}(\phi) = \{\phi\} \cup \begin{cases} \mathrm{SF}(\phi_1) & ,\phi = \neg\phi_1 \\ \mathrm{SF}(\phi_1) \cup \mathrm{SF}(\phi_2) & ,\phi = \phi_1 \wedge \phi_2 \\ \mathrm{SF}(\phi_1) & ,\phi = \mathsf{X}\,\phi_1 \\ \mathrm{SF}(\phi_1) \cup \mathrm{SF}(\phi_2) & ,\phi = \phi_1\,\mathsf{U}\,\phi_2 \end{cases}$$

- The subformulas closed under negation and modulo double negation are

$$SF^{\neg}(\phi) = \{\phi, \neg\phi \mid \phi \in SF(\phi), \phi \neq \neg\phi'\}$$

Construction

Given the LTL formula ϕ, define the Alternating Büchi Automaton

$$\mathcal{A}_\phi = (S, s^0, \rho, F)$$

over the alphabet 2^{Prop} with

- $S = \mathrm{SF}^\neg(\phi)$
- $s^0 = \phi$
- $F = \{\psi \in S \mid \psi = \neg(\psi_1 \mathsf{U} \psi_2)\}$
- $$\begin{aligned}
\rho(p, a) &= \text{true, if } p \in a \\
\rho(p, a) &= \text{false, if } p \notin a \\
\rho(\psi_1 \wedge \psi_2, a) &= \rho(\psi_1, a) \wedge \rho(\psi_2, a) \\
\rho(\neg\psi, a) &= \overline{\rho(\psi, a)} \quad \text{where} \\
\rho(\mathsf{X}\,\psi, a) &= \psi \\
\rho(\psi_1 \mathsf{U} \psi_2, a) &= (\rho(\psi_1, a) \wedge \psi_1 \mathsf{U} \psi_2) \vee \rho(\psi_2, a)
\end{aligned}$$

$$\left[\begin{aligned}
\overline{\psi} &= \neg\psi, \text{ if } \psi \in S \\
\overline{\text{true}} &= \text{false} \\
\overline{\text{false}} &= \text{true} \\
\overline{\alpha \vee \beta} &= \overline{\alpha} \wedge \overline{\beta} \\
\overline{\alpha \wedge \beta} &= \overline{\alpha} \vee \overline{\beta}
\end{aligned}\right.$$

Nested Depth First Search (1/4)

Input: rooted graph with accepting states
Output: existence of accepting cycle

var V_1, V_2 : set;
main(){
    $V_1 := \emptyset$;
    $V_2 := \emptyset$;
    DFS(root);
    exit absent;
}

Nested Depth First Search (2/4)

The first DFS looks for accepting states:

DFS(s){
  if $s \in V_1$ return;
  $V_1 := V_1 \cup \{s\}$;
  for s' in succ(s){
    DFS(s')
  }
  if accepting(s) NDFS(s,s);
}

Nested Depth First Search (3/4)

The second DFS looks for accepting cycles:

NDFS(s,a){
  if $s \in V_2$ return;
  $V_2 := V_2 \cup \{s\}$;
  for s' in succ(s){
    if $s' = a$ then exit present;
    NDFS(s',a)
  }
}

Nested Depth First Search (4/4)

- Note that the second DFS does not clear V_2.
- Time and memory complexity are linear in the number of edges of the graph.
- Works 'on-the-fly':
  if a cycle is present, it may be found without searching the whole graph.
- Note that a cycle is present on the stack when exiting 'present'.

Stuttering

- Intuitively means moving between states with the same atomic propositions.
- A run is a sequence of subsets of Prop:

$$S_1' \, S_2' \, S_3' \cdots$$

- Defining

$$S^1 = S, \; S^{n+1} = S \, S^n,$$

we can uniquely write any run π as

$$S_1^{n_1} \, S_2^{n_2} \, S_3^{n_3} \cdots$$

where $S_i \neq S_{i+1}$.

- The stutter free variant of π, denoted $E(\pi)$, is

$$S_1 \, S_2 \, S_3 \cdots$$

Stuttering Invariance

- An LTL formula ϕ is stutter invariant if

$$\pi, 0 \models \phi \Leftrightarrow E(\pi), 0 \models \phi$$

- The fragment of LTL, without the next time operator X is stutter invariant. This fragment is denoted LTL-X.

Safety properties

- Intuitively it is a property that says that bad things never happen.
- A safety property is a property that allows failure detection in finitely many steps.
- Given a property.
  - A bad prefix is a finite prefix of a computation for which the property fails, such that any computation starting with the same prefix also fails the property.
  - The property is a safety property is every failing computation has a bad prefix.
- The following class of LTL formula's are safety formula's

$$\phi_s ::= p \mid \neg p \mid \phi_s \wedge \phi_s \mid \phi_s \vee \phi_s \mid X\, \phi_s \mid \phi_s\, R\, \phi_s$$

Liveness properties

- Intuitively it is a property that says that good things happen infinitely often.
- For LTL we can just say that a property is a liveness property if it isn't a safety property.

Fairness

- Intuitively it means that everybody gets their turn.
- Let there be N processes $1 \leq i \leq N$.
- *Weak fairness* If from a certain point in the computation a step is continuously enabled (p_i) then it is executed infinitely often (q_i).

$$\bigwedge_{1 \leq i \leq N} (\Diamond\Box p_i \rightarrow \Box\Diamond q_i) \sim \bigwedge_{1 \leq i \leq N} \Box\Diamond(\neg p_i \vee q_i)$$

- *Strong fairness* If a step is infinitely often enabled then it is infinitely often executed.

$$\bigwedge_{1 \leq i \leq N} (\Box\Diamond p_i \rightarrow \Box\Diamond q_i)$$

Model Checking and Fairness

- To check whether a property holds for fair computations, we can obviously check the formula

$$\text{fairness} \longrightarrow \text{property}$$

- Unfortunately, many tools will experience an exponential blow-up in the number of processes.
- If you are lucky, the tool will have support for fairness
- Otherwise, you will have to do the work yourself.

Fairness for continuously enabled processes.

Let process be a variable containing the process that made the last step.
Given a Büchi Automaton and K processes, make $K+2$ copies $(0, \cdots, K+1)$ of the Büchi automaton.

- Remove the acceptance conditions from all copies, except copy 0.
- In copy 0, redirect edges starting in accepting states to copy 1.
- In copy i $(1 \leq i \leq K)$ duplicate the edges to the next copy with the extra condition process $= i$ and add the condition process $\neq i$ to every old edge.
- In copy $k+1$, redirect all edges to copy 0.

Fairness for processes, which once enabled, remain enabled until taken.

Given a Büchi Automaton and K processes, make $K+2$ copies $(0, \cdots, K+1)$. of the Büchi automaton.

- Remove the acceptance conditions from all copies, except copy 0.
- In copy 0, redirect edges starting in accepting states to copy 1.
- In copy i $(1 \leq i \leq K)$ duplicate the edges to the next copy with the extra condition [process $= i$ or process i not enabled] and add the condition [process $\neq i$ and process i enabled] to every old edge.
- In copy $k+1$, redirect all edges to copy 0.