Lecture 9:
LTL and Büchi Automata
LTL Property Patterns

Quite often the requirements of a system follow some simple patterns. Sometimes we want to specify that a property should only hold in a certain context, called the scope of a property. Typical scopes are:

**Global**: The property should hold on the whole path (i.e. on all suffixes of a run).

**Before** $R$: The property should hold before the first appearance of $R$ (i.e. on all suffixes before the first suffix satisfying $R$).

**After** $Q$: The property should hold after the first appearance of $Q$. 
Between \( Q \) and \( R \): The property should hold in all sequences in which the first suffix satisfies \( Q \) and the last one satisfies \( R \).

After \( Q \) until \( R \): As before, but also includes the sequences in which \( Q \) appears, but is never followed by an \( R \).

Note: In the following patterns, scopes are interpreted in a way that always includes the suffix at which the event triggering the scope happens, but excludes the suffix at which the event ending the scope happens.
Scopes

Global

Before R

After Q

Between Q and R

After Q until R
LTL Property Patterns: Absence

Absence patterns specify that “$P$ is false” within the scope:

- **Global**: $G \neg P$
- **Before $R$**: $(F R) \rightarrow (\neg P \lor R)$
- **After $Q$**: $G(Q \rightarrow G \neg P)$
- **Between $Q$ and $R$**: $G((Q \land \neg R \land F R) \rightarrow (\neg P \lor R))$
- **After $Q$ until $R$**: $G((Q \land \neg R) \rightarrow (\neg P \lor R))$
Existence patterns specify that “\( P \) becomes true” within the scope:

- **Global**
  \[ \mathbf{F} P \]

- **Before** \( R \)
  \[ \neg R \mathbf{W} (P \land \neg R) \]

- **After** \( Q \)
  \[ (\mathbf{G} \neg Q) \lor (\mathbf{F}(Q \land \mathbf{F} P)) \]

- **Between** \( Q \) and \( R \)
  \[ \mathbf{G}((Q \land \neg R) \rightarrow (\neg R \mathbf{W} (P \land \neg R))) \]

- **After** \( Q \) until \( R \)
  \[ \mathbf{G}((Q \land \neg R) \rightarrow (\neg R \mathbf{U} (P \land \neg R))) \]
LTL Property Patterns

The property patterns are useful to make specifying easier: If a desired property falls into one of those patterns, the pattern can be ‘instantiated’ for appropriate $P$, $Q$, and $R$.

The patterns (and the definition of scopes) from the previous slides were taken from: http://patterns.projects.cis.ksu.edu/, where you can find many more of them (also for other temporal logics).

There are also other patterns available expressing:

- Universality: “$P$ is true” (dual of absence)
- Precedence: “$S$ precedes $P$”
- Response: “$S$ responds to $P$”
- Etc., etc.
Relations between Temporal and Logical Operators

\[ \begin{align*}
X(\phi_1 \lor \phi_2) & \equiv X\phi_1 \lor X\phi_2 \\
X(\phi_1 \land \phi_2) & \equiv X\phi_1 \land X\phi_2 \\
X \neg \phi & \equiv \neg X\phi \\
F(\phi_1 \lor \phi_2) & \equiv F\phi_1 \lor F\phi_2 \\
\neg F\phi & \equiv G\neg \phi \\
G(\phi_1 \land \phi_2) & \equiv G\phi_1 \land G\phi_2 \\
\neg G\phi & \equiv F\neg \phi \\
(\phi_1 \land \phi_2) U \psi & \equiv (\phi_1 U \psi) \land (\phi_2 U \psi) \\
\phi U (\psi_1 \lor \psi_2) & \equiv (\phi U \psi_1) \lor (\phi U \psi_2)
\end{align*} \]
Idempotence and Recursion Laws

\[ F \phi \equiv F F \phi \]
\[ G \phi \equiv G G \phi \]
\[ \phi U \psi \equiv \phi U (\phi U \psi) \]

\[ F \phi \equiv \phi \lor X F \phi \]
\[ G \phi \equiv \phi \land X G \phi \]
\[ \phi U \psi \equiv \psi \lor (\phi \land X (\phi U \psi)) \]
\[ \phi W \psi \equiv \psi \lor (\phi \land X (\phi W \psi)) \]
\[ \phi R \psi \equiv (\phi \land \psi) \lor (\psi \land X (\phi R \psi)) \]
LTL Model Checking

We now turn to the question how to check whether a given Kripke structure satisfies a given formula. In the context of temporal logics, this is called model checking (i.e. checking whether all runs are models of the given formula).

Like in the case of safety properties, we follow an automata-theoretic approach:

1. We introduce a new class of automata that can express LTL properties.

2. We show how to translate properties into these automata.

3. We check whether the intersection of the system and the automaton for the negation of the property is empty.

Literature: Clarke, Grumberg, Peled: *Model Checking*, MIT Press, 1999
Büchi Automata: Definition

A Büchi automaton is a tuple

\[ B = \langle \Sigma, S, S^0, \Delta, F \rangle \]

such that

\( \Sigma \) is a finite alphabet,
\( S \) is a finite set of states,
\( S^0 \subseteq Q \) are the initial states,
\( \Delta \subseteq S \times \Sigma \times S \) is the transition relation, and
\( F \subseteq S \) are the accepting states.

So far, Büchi automata look exactly like finite automata. However, they operate on infinite words, and they have a different acceptance condition (see next slide).
Büchi Automata: Acceptance and Language

Let $B = \langle \Sigma, S, S^0, \Delta, F \rangle$ be a Büchi automaton.

A run of $B$ on an infinite word $\sigma \in \Sigma^\omega$ is an infinite sequence of states $\rho \in S^\omega$ such that $\rho(0) \in S^0$, and $(\rho(i), \sigma(i), \rho(i + 1)) \in \Delta$ for all $i \geq 0$.

We call a run $\rho$ accepting iff for infinitely many indices $i$ it holds that $\rho(i) \in F$ (i.e. $\rho$ infinitely often visits accepting states).

A word $\sigma \in \Sigma^\omega$ is accepted by $B$ iff there is an accepting run on $\sigma$ in $B$.

The language of $B$, denoted $\mathcal{L}(B) \subseteq \Sigma^\omega$ is defined as the set of infinite words over $\Sigma$ accepted by $B$. 
Büchi Automata: Examples

“infinitely often b”

“infinitely often ab”
Operations on Büchi Automata

Like finite automata, the languages accepted by Büchi automata are closed under boolean operations. We will examine the following operations:

**Intersection** of Büchi automata $B_1$ and $B_2$:
construct $B$ with $L(B) = L(B_1) \cap L(B_2)$

**Union** of Büchi automata $B_1$ and $B_2$:
construct $B$ with $L(B) = L(B_1) \cup L(B_2)$

**Complementation** of Büchi automaton $B_1$:
construct $B$ with $L(B) = \Sigma^* \setminus L(B_1)$

**Emptiness check:**
given $B$, check if $L(B) = \emptyset$
Intersection of Büchi automata

The construction of the intersection automaton works a little differently from the finite-state case. We need to check whether both sets of accepting states occur infinitely often.

Idea: We create two copies of the intersected state space.
In the first copy, we check for occurrence of the first acceptance set.
In the second copy, we check for occurrence of the second acceptance set.
We jump back and forth between the copies whenever we find an accepting state.

Let $B_1 = \langle \Sigma, S_1, S_0^1, \Delta_1, F_1 \rangle$, $B_2 = \langle \Sigma, S_2, S_0^2, \Delta_2, F_2 \rangle$. We define the intersection automaton (or: product automaton) to be $B = \langle \Sigma, S, S_0^0, \Delta, F \rangle$, where

$$S = S_1 \times S_2 \times \{1, 2\}, \quad S_0^0 = S_1^0 \times S_2^0 \times \{1\}, \quad F = F_1 \times S_2 \times \{1\}$$

$\Delta$ as defined on the next slide.
\((\langle s, t, 1 \rangle, a, \langle s', t', 1 \rangle) \in \Delta \iff (s, a, s') \in \Delta_1, (t, a, t') \in \Delta_2, s \notin F_1\)

\((\langle s, t, 1 \rangle, a, \langle s', t', 2 \rangle) \in \Delta \iff (s, a, s') \in \Delta_1, (t, a, t') \in \Delta_2, s \in F_1\)

\((\langle s, t, 2 \rangle, a, \langle s', t', 2 \rangle) \in \Delta \iff (s, a, s') \in \Delta_1, (t, a, t') \in \Delta_2, t \notin F_2\)

\((\langle s, t, 2 \rangle, a, \langle s', t', 1 \rangle) \in \Delta \iff (s, a, s') \in \Delta_1, (t, a, t') \in \Delta_2, t \in F_2\)
Intersection: Example

B1

s0

s1

B2

t0

t1

B1 x B2

s0,t0,1

s1,t1,2

s0,t0,2

s1,t1,1
Union and Complement

**Union**: Juxtapose both automata (like in the finite case)

**Complement**: Complicated!
The complement construction, when applied to a Büchi automaton with $n$ states, results in an automaton with $O(n!)$ states. We will skip it in this course.

Details on Complementation: see e.g.

W. Thomas, *Automata on Infinite Objects*,
Chapter 4 in *Handbook of Theoretical Computer Science*,

or I. Walukiewicz, Lecture notes on *Automata and Logic*, Chapter 3,
[www.labri.fr/Perso/~igw/Papers/igw-eefss01.ps](http://www.labri.fr/Perso/~igw/Papers/igw-eefss01.ps)
Emptiness check

Observation: $L(\mathcal{B}) \neq \emptyset$ iff there is an accepting state $s \in F$ that is reachable from an initial state \textit{and} reachable from itself (the latter with a non-empty path).

This condition can be checked with an algorithm that takes linear time in the size of the Büchi automaton. (We will see how later.)
Note on determinism

In the case of finite automata, we could convert each non-deterministic automaton into a language-equivalent deterministic automaton. This is not the case with Büchi automata.

In other words, non-deterministic Büchi automata are strictly more expressive than deterministic Büchi automata.

“Eventually, only b will occur.”

Not expressible by a deterministic Büchi automaton!
Generalised Büchi Automata

A variant of the model are so-called generalised Büchi automata. They differ from (normal) Büchi automata only in the acceptance condition, which is a ‘set of acceptance sets’, i.e. $\mathcal{F} \subseteq 2^S$

In a generalised Büchi automaton, a run $\rho$ is accepting iff, for $\mathcal{F} = \{F_1, \ldots, F_n\}$ and each index $1 \leq i \leq n$, we have that $\rho$ visits infinitely many states from $F_i$.

Generalised Büchi automata can be translated back into Büchi automata. (Take the $n$-fold intersection of the automaton with itself, with acceptance sets $F_1$ through $F_n$.)