Tree Automata, Logic and Verification of (Higher-Order-) Functional Programs

Olivier Serre

LIAFA (Université Paris Diderot – Paris 7 & CNRS)

16+18 June 2014
A toy example:

Main = MakeReport Nil
MakeReport x = if * (Commit x)
    else (AddData x MakeReport)
AddData y ϕ = if * (ϕ (Error End)) else (ϕ Cons(_, y))
A toy example:

\[
\text{Main} = \text{MakeReport } \text{Nil} \\
\text{MakeReport } x = \text{if } * (\text{Commit } x) \\
\phantom{\text{MakeReport } x = } \text{else } (\text{AddData } x \text{ MakeReport}) \\
\text{AddData } y \; \phi = \text{if } * (\phi (\text{Error End})) \text{ else } (\phi \; \text{Cons}(\_, y))
\]

Associated higher-order recursion scheme:

\[
\begin{align*}
\text{main} & \mapsto M \; \text{nil} \\
M \; x & \mapsto \text{or } (\text{commit } x) (A \; x \; M) \\
A \; y \; \phi & \mapsto \text{or } (\phi (\text{error end})) (\phi (\text{cons } y))
\end{align*}
\]
A toy example:

Main = MakeReport Nil
MakeReport x = if * (Commit x) else (AddData x MakeReport)
AddData y ϕ = if * (ϕ (Error End)) else (ϕ Cons(_, y))

Associated higher-order recursion scheme:
A toy example:

\[
\text{Main} = \text{MakeReport } \text{Nil} \\
\text{MakeReport } x = \text{if } * (\text{Commit } x) \text{ else (AddData } x \text{ MakeReport)} \\
\text{AddData } y \varphi = \text{if } * (\varphi (\text{Error End})) \text{ else (} \varphi \text{ Cons(}_, y))
\]

Associated higher-order recursion scheme:

\[
\begin{align*}
\text{main} & \leftrightarrow M \text{ nil} \\
M x & \leftrightarrow \text{or (commit } x) (A x M) \\
A y \varphi & \leftrightarrow \text{or (} \varphi \text{ (error end)) (} \varphi \text{ (cons } y))
\end{align*}
\]

\[
\mathcal{L}_{\text{err}} = \text{or}^* \text{ commit or}^* \text{ error end} \\
\varphi_{\text{err}} = \exists x \exists y (\text{commit}(x) \land \text{error}(y) \land x < y)
\]
The Recipe

Ingredients

- A program
  Here modelled as an infinite (say binary) node-labeled tree
  Branching? ⇒ Nondeterminism, environment, nature...
Ingredients

→ A program
Here modelled as an infinite (say binary) node-labeled tree
Branching? ⇒ Nondeterminism, environment, nature...

→ A specification
Here described by an MSO formula or a parity tree automaton

- Every branch that contains infinitely many $r$ also contains infinitely many $g$.
- There exists a subtree in which every branch contain infinitely many $a$.
- Any subtree rooted at a node labeled by $a$ contains a branch with infinitely many $b$. 
The Recipe

Ingredients

→ A program
  Here modelled as an infinite (say binary) node-labeled tree
  Branching? ⇒ Nondeterminism, environment, nature…

→ A specification
  Here described by an MSO formula or a parity tree automaton

Instructions

→ Decide whether the specification is satisfied

- Every branch that contains infinitely many \( r \) also contains infinitely many \( g \).
- There exists a subtree in which every branch contain infinitely many \( a \).
- Any subtree rooted at a node labeled by \( a \) contains a branch with infinitely many \( b \).
The Recipe

Ingredients

- **A program** [Wednesday]
  Here modelled as an infinite (say binary) node-labeled tree
  Branching? ⇒ *Nondeterminism, environment, nature...*

- **A specification** [Today]
  Here described by an MSO formula or a parity tree automaton

![](image)

- Every branch that contains infinitely many *r* also contains infinitely many *g*.
- There exists a subtree in which every branch contain infinitely many *a*.
- Any subtree rooted at a node labeled by *a* contains a branch with infinitely many *b*.

Instructions [Wednesday]

- **Decide** whether the specification is satisfied
Monadic Second Order Logic
Formulas are built up from atomic formulas of the form

- $x = y$
- $S(x, y)$ for words; $S_0(x, y)$ and $S_1(x, y)$ for binary trees
- $a(x)$ for any letter $a$
- $x \in X$

using the Boolean connectives $\lor$, $\land$, $\neg$, $\Rightarrow$ and the quantifiers $\exists$ and $\forall$
Formulas are built up from atomic formulas of the form

- $x = y$
- $S(x, y)$ for words; $S_0(x, y)$ and $S_1(x, y)$ for binary trees
- $a(x)$ for any letter $a$
- $x \in X$

using the Boolean connectives $\lor$, $\land$, $\neg$, $\Rightarrow$ and the quantifiers $\exists$ and $\forall$

Example: How to express that a word $u \in A^* a A^* b A^*$?

\[
\bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet
\]
Syntax & Semantics of MSO Logic

Formulas are built up from atomic formulas of the form

- $x = y$
- $S(x, y)$ for words; $S_0(x, y)$ and $S_1(x, y)$ for binary trees
- $a(x)$ for any letter $a$
- $x \in X$

using the Boolean connectives $\lor$, $\land$, $\neg$, $\Rightarrow$ and the quantifiers $\exists$ and $\forall$

Example: How to express that a word $u \in A^*aA^*bA^*$?

```
       a
     ⬤ ⬤ ⬤ ⬤ ⬤ ⬤ ⬤ ⬤ ⬤ ⬤ ⬤ ⬤ b
```
Syntax & Semantics of MSO Logic

**Formulas** are built up from atomic formulas of the form

- \( x = y \)
- \( S(x, y) \) for words; \( S_0(x, y) \) and \( S_1(x, y) \) for binary trees
- \( a(x) \) for any letter \( a \)
- \( x \in X \)

using the Boolean connectives \( \lor, \land, \neg, \Rightarrow \) and the **quantifiers** \( \exists \) and \( \forall \)

**Example:** How to express that a word \( u \in A^*aA^*bA^* \)?

\[
\begin{array}{c}
\bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \quad \bullet \\
\end{array}
\]

\[
\exists x \quad \exists y \quad [a(x) \land b(x) \land x < y]
\]

where \( x < y \) is a shorthand for

\[
\exists Z [x \in Z \land y \in Z \land (\forall z \in Z(\exists z', z' \in Z \land S(z, z')) \lor z = y) \land (\neg \exists y' \in Z, S(y, y'))]
\]
Formulas are built up from atomic formulas of the form
- $x = y$
- $S(x, y)$ for words; $S_0(x, y)$ and $S_1(x, y)$ for binary trees
- $a(x)$ for any letter $a$
- $x \in X$

using the Boolean connectives $\lor$, $\land$, $\neg$, $\Rightarrow$ and the quantifiers $\exists$ and $\forall$

Example: How to say that a tree contains a branch with infinitely many $a$’s?
**Formulas** are built up from atomic formulas of the form

- $x = y$
- $S(x, y)$ for words; $S_0(x, y)$ and $S_1(x, y)$ for binary trees
- $a(x)$ for any letter $a$
- $x \in X$

using the Boolean connectives $\lor$, $\land$, $\neg$, $\Rightarrow$ and the **quantifiers** $\exists$ and $\forall$

**Example:** How to say that a tree contains a branch with infinitely many $a$’s?

$$\exists X \left[ \exists x \ x \in X \right] \land \left[ \forall x \ \forall y \ (x \in X \land y \in X) \Rightarrow (x \leq y \lor y \leq x) \right] \land \left[ \forall x \ x \in X \Rightarrow (\exists y \ y \in X \land x < y) \right] \land \left[ \forall x \ x \in X \Rightarrow a(x) \right]$$
Once a finite alphabet $A$ is fixed any MSO formula defines a language in $A^*$. 

**Theorem (Büchi’60,Elgot’61)***

*A language of finite words is regular iff it is MSO definable.*
Finite Words: MSO Logic vs Regular Languages

Once a finite alphabet $A$ is fixed any MSO formula defines a language in $A^*$.  

**Theorem (Büchi’60,Elgot’61)**

A language of finite words is regular iff it is MSO definable.

**Proof (sketch).** ($\Rightarrow$) Fix an automaton $A = (\{1, \ldots, k\}, A, 1, \Delta, F)$.

Let $u = u_0 \cdots u_n \in A^*$. 

\[
\bullet --- \bullet --- \bullet --- \bullet --- \bullet --- \bullet --- \bullet --- \bullet --- \bullet
\]

\[u_0 \quad u_1 \quad u_2 \quad u_3 \quad u_4 \quad u_5 \quad u_6 \quad u_7 \quad u_8 \quad u_9\]
Once a finite alphabet $A$ is fixed any MSO formula defines a language in $A^*$. 

**Theorem (Büchi’60,Elgot’61)**  

A language of finite words is regular iff it is MSO definable.

**Proof (sketch).** ($\Rightarrow$) Fix an automaton $A = (\{1, \ldots, k\}, A, 1, \Delta, F)$. Let $u = u_0 \cdots u_n \in A^*$.

\[ q_0 = 1 \quad u_0 \quad q_1 \quad u_1 \quad q_2 \quad u_2 \quad q_3 \quad u_3 \quad q_4 \quad u_4 \quad q_5 \quad u_5 \quad q_6 \quad u_6 \quad q_7 \quad u_7 \quad q_8 \quad u_8 \quad q_9 \quad q_{10} \in F \]

$u \in L(A)$ iff $\exists q_0, \ldots, q_{n+1}$ with $q_0 = 1$, $q_{n+1} \in F$ and $q_{i+1} \in \Delta(q_i, a_i)$ for all $0 \leq i \leq n$. 

\[ u_0, u_1, u_2, u_3, u_4, u_5, u_6, u_7, u_8, u_9, \ldots, u_n, q_{n+1} \]
Finite Words: MSO Logic vs Regular Languages

Once a finite alphabet $A$ is fixed any MSO formula defines a language in $A^*$.

**Theorem (Büchi’60,Elgot’61)**

A language of finite words is regular iff it is MSO definable.

**Proof (sketch).** ($\Rightarrow$) Fix an automaton $A = (\{1, \ldots, k\}, A, 1, \Delta, F)$.

Let $u = u_0 \cdots u_n \in A^*$.

$$
\begin{array}{c}
q_0 = 1 \\
u_0 \\
\bullet \\
q_1 \\
u_1 \\
\bullet \\
q_2 \\
u_2 \\
\bullet \\
q_3 \\
u_3 \\
\bullet \\
q_4 \\
u_4 \\
\bullet \\
q_5 \\
u_5 \\
\bullet \\
q_6 \\
u_6 \\
\bullet \\
q_7 \\
u_7 \\
\bullet \\
q_8 \\
u_8 \\
\bullet \\
q_9 \\
u_9 \\
\bullet \\
q_{10} \in F
\end{array}
$$

$u \in L(A)$ iff $\exists q_0, \ldots, q_{n+1}$ with $q_0 = 1$, $q_{n+1} \in F$ and $q_{i+1} \in \Delta(q_i, a_i)$ for all $0 \leq i \leq n$.

Equivalently, $\exists$ partition $X_1 \uplus \cdots \uplus X_k$ such that

$\forall x [\text{first}(x) \Rightarrow x \in X_1]$

$\forall x \forall y [S(x, y) \Rightarrow \bigvee_{(i,a,j)\in\Delta} (x \in X_i \land a(x) \land y \in X_j)]$

$\forall x [\text{last}(x) \Rightarrow \bigvee_{(i,a,j)\in\Delta \cap F} (x \in X_i \land a(x))]$
Once a finite alphabet $A$ is fixed any MSO formula defines a language in $A^*$.

**Theorem (Büchi’60,Elgot’61)**

A language of finite words is regular iff it is MSO definable.

**Proof (sketch).**

$(\Leftarrow)$ First remark that the following syntax is equivalent to the previous one (and permits to avoid dealing with first-order quantification):

- $X \subseteq Y$: $\forall x \ x \in X \Rightarrow x \in Y$
- $\text{Sing}(X)$: $\forall x, y \in X, \ x = y$
- $a(X)$: $\forall x \ x \in X \Rightarrow a(x)$
- $\text{Suc}(X, Y)$: $\text{Sing}(X) \wedge \text{Sing}(Y) \wedge \forall x \in X \forall y \in Y, \ S(x, y)$
Finite Words: MSO Logic vs Regular Languages

Once a finite alphabet $A$ is fixed any MSO formula defines a language in $A^*$.

**Theorem (Büchi’60,Elgot’61)**

A language of finite words is regular iff it is MSO definable.

**Proof (sketch).**

$(\Leftarrow)$ First remark that the following syntax is equivalent to the previous one (and permits to avoid dealing with first-order quantification):

- $X \subseteq Y: \forall x \in X \Rightarrow x \in Y$
- $Sing(X): \forall x, y \in X, x = y$
- $a(X): \forall x \in X \Rightarrow a(x)$
- $Suc(X, Y): Sing(X) \land Sing(Y) \land \forall x \in X \forall y \in Y, S(x, y)$

and build for any formula $\varphi(X_1, \ldots, X_k)$ an automaton accepting those words on $A \times \{0, 1\}^k$ which satisfies $\varphi(X_1, \ldots, X_k)$.  

Once a finite alphabet $A$ is fixed any MSO formula defines a language in $A^*$.  

**Theorem (Büchi’60, Elgot’61)**

A language of finite words is regular iff it is MSO definable.

**Proof (sketch).**

$(\Leftarrow)$ First remark that the following syntax is equivalent to the previous one (and permits to avoid dealing with first-order quantification):

- $X \subseteq Y: \forall x \ x \in X \Rightarrow x \in Y$
- $\text{Sing}(X): \forall x, y \in X, \ x = y$
- $a(X): \forall x \ x \in X \Rightarrow a(x)$
- $\text{Suc}(X,Y): \text{Sing}(X) \land \text{Sing}(Y) \land \forall x \in X \ \forall y \in Y, \ S(x,y)$

and build for any formula $\varphi(X_1, \ldots, X_k)$ an automaton accepting those words on $A \times \{0, 1\}^k$ which satisfies $\varphi(X_1, \ldots, X_k)$.

For this it suffices to consider only connectives $\neg$ and $\lor$ and existential set quantification. An automaton for $\exists X_k \psi(X_1, \ldots, X_k)$ is built from the automaton for $\psi(X_1, \ldots, X_k)$ by now guessing the $k$–th additional component.
Infinite words: One needs to have an automata model that permits to capture “infinite events”: Büchi automata are well suited for that!
**Infinite words**: One needs to have an automata model that permits to capture “infinite events”: Büchi automata are well suited for that!

**Theorem (Büchi’62)**

*A language of infinite words is regular iff it is MSO definable.*

**Proof.** Once one knows that languages recognised by Büchi automata form a Boolean algebra the proof is identical to the previous one for finite words.
**Infinite words:** One needs to have an automata model that permits to capture “infinite events”: Büchi automata are well suited for that!

**Theorem (Büchi’62)**

>A language of infinite words is regular iff it is MSO definable.

**Proof.** Once one knows that languages recognised by Büchi automata form a Boolean algebra the proof is identical to the previous one for finite words.

**Infinite trees:** One needs to have an automata model that permits to capture “infinite events” as well as to deal with branching: $\omega$-tree automata are well suited for that!

**Theorem (Rabin’69)**

>A language of infinite trees is regular iff it is MSO definable.

**Proof.** Once one knows that languages recognised by $\omega$-tree automata form a Boolean algebra the proof is identical to the previous one for finite words.
Model-checking:

- Input: A formula and a model (e.g. a word or a tree)
- Output: Does the model satisfy the formula?

Equivalently:

- Input: An automaton and a model (e.g. a word or a tree)
- Output: Does the automaton accept the formula?
From Logic to Automata

Model-checking:

→ Input: A formula and a model (e.g. a word or a tree)
→ Output: Does the model satisfy the formula?

Equivalently:

→ Input: An automaton and a model (e.g. a word or a tree)
→ Output: Does the automaton accept the formula?

Satisfiability:

→ Input: A formula
→ Output: Is there a model that satisfies the formula?

Equivalently:

→ Input: An automaton
→ Output: Is the language accepted by the automaton empty?
Tree Automata

Definitions
Non-deterministic Parity Tree Automata

Non-deterministic parity tree automata: \( A = \langle Q, \Sigma, \Delta, q_{\text{in}}, \text{col} \rangle \)

- \( Q \): control states
- \( \Sigma \): labels alphabet
- \( \Delta \subseteq Q \times \Sigma \times Q \times Q \): transition relation
- \( q_{\text{in}} \): initial state
- \( \text{col} : Q \to \mathbb{N} \): colouring function

Run on a \( \Sigma \)-labeled (infinite binary) tree \( t \): \( Q \)-labelling of \( t \) consistent with \( \Delta \)

A branch is **accepting** iff the smallest colour infinitely often visited is even

A run is **accepting** iff all its branches are accepting

A tree is **accepted** iff there is an accepting run over it.
Tree Automata: an Example

$\Sigma = \{a, b\}$

$Q = \{q_1, q_2, q_3\}$

$A$

$\begin{align*}
q_1 & \xrightarrow{a} (q_1, q_3) \\
q_1 & \xrightarrow{a} (q_2, q_3) \\
q_1 & \xrightarrow{b} (q_1, q_3) \\
q_2 & \xrightarrow{a} (q_2, q_3) \\
q_2 & \xrightarrow{b} (q_1, q_3) \\
q_3 & \xrightarrow{a} (q_3, q_3) \\
q_3 & \xrightarrow{b} (q_3, q_3)
\end{align*}$

initial state: $q_1$

$F = \{q_2, q_3\}$
Tree Automata: an Example

\[ \Sigma = \{a, b\} \]

\[ \mathcal{A} \]

\[ Q = \{q_1, q_2, q_3\} \]

\[ q_1 \xrightarrow{a} (q_1, q_3) \quad q_2 \xrightarrow{a} (q_2, q_3) \]

\[ q_1 \xrightarrow{a} (q_2, q_3) \quad q_2 \xrightarrow{b} (q_1, q_3) \]

\[ q_1 \xrightarrow{b} (q_1, q_3) \quad q_3 \xrightarrow{b} (q_3, q_3) \]

\[ q_3 \xrightarrow{a} (q_3, q_3) \]

initial state: \( q_1 \)

\[ F = \{q_2, q_3\} \]
Tree Automata: an Example

\[ \Sigma = \{a, b\} \]

\[ Q = \{q_1, q_2, q_3\} \]

\[ F = \{q_2, q_3\} \]

\[ A \]

\[ \begin{align*}
q_1 & \xrightarrow{a} (q_1, q_3) & q_2 & \xrightarrow{a} (q_2, q_3) \\
q_1 & \xrightarrow{a} (q_2, q_3) & q_2 & \xrightarrow{b} (q_1, q_3) \\
q_1 & \xrightarrow{b} (q_1, q_3) & q_3 & \xrightarrow{b} (q_3, q_3) \\
q_3 & \xrightarrow{a} (q_3, q_3) & \\
\end{align*} \]
Tree Automata: an Example

$\Sigma = \{a, b\}$

$A$:

$Q = \{q_1, q_2, q_3\}$

$t$:

$q_1 \xrightarrow{a} (q_1, q_3)$
$q_1 \xrightarrow{a} (q_2, q_3)$
$q_1 \xrightarrow{a} (q_2, q_3)$
$q_1 \xrightarrow{b} (q_1, q_3)$
$q_2 \xrightarrow{a} (q_2, q_3)$
$q_2 \xrightarrow{b} (q_1, q_3)$
$q_3 \xrightarrow{a} (q_3, q_3)$
$q_3 \xrightarrow{b} (q_3, q_3)$

initial state: $q_1$

$F = \{q_2, q_3\}$
Tree Automata: an Example

\[ \Sigma = \{a, b\} \]

\[ Q = \{q_1, q_2, q_3\} \]

\[ F = \{q_2, q_3\} \]
Tree Automata: an Example

$\Sigma = \{a, b\}$

$Q = \{q_1, q_2, q_3\}$

$F = \{q_2, q_3\}$
Tree Automata: an Example

$\Sigma = \{a, b\}$

$Q = \{q_1, q_2, q_3\}$

$F = \{q_2, q_3\}$
Tree Automata: an Example

$\Sigma = \{a, b\}$

$t$: 

\[
\begin{align*}
Q &= \{q_1, q_2, q_3\} \\
\mathcal{A} &= \\
q_1 \xrightarrow{a} (q_1, q_3) & q_2 \xrightarrow{a} (q_2, q_3) \\
q_1 \xrightarrow{a} (q_2, q_3) & q_2 \xrightarrow{b} (q_1, q_3) \\
q_1 \xrightarrow{b} (q_1, q_3) & q_3 \xrightarrow{b} (q_3, q_3) \\
q_3 \xrightarrow{a} (q_3, q_3)
\end{align*}
\]

initial state: $q_1$ \quad $F = \{q_2, q_3\}$
Tree Automata: an Example

$\Sigma = \{a, b\}$

$Q = \{q_1, q_2, q_3\}$

$F = \{q_2, q_3\}$
Tree Automata: an Example

\[ \Sigma = \{a, b\} \]

\[ t: \]

\[ Q = \{q_1, q_2, q_3\} \]

\[ A: \]

\[ q_1 \xrightarrow{a} (q_1, q_3) \]
\[ q_2 \xrightarrow{a} (q_2, q_3) \]
\[ q_1 \xrightarrow{a} (q_2, q_3) \]
\[ q_2 \xrightarrow{b} (q_1, q_3) \]
\[ q_1 \xrightarrow{b} (q_1, q_3) \]
\[ q_3 \xrightarrow{b} (q_3, q_3) \]
\[ q_3 \xrightarrow{a} (q_3, q_3) \]

initial state: \( q_1 \)

\[ F = \{q_2, q_3\} \]
Tree Automata: an Example

$\Sigma = \{a, b\}$

$t$

\[ Q = \{q_1, q_2, q_3\} \]

\[ F = \{q_2, q_3\} \]

\[ A \]

\[ q_1 \xrightarrow{a} (q_1, q_3) \]

\[ q_2 \xrightarrow{a} (q_2, q_3) \]

\[ q_1 \xrightarrow{a} (q_2, q_3) \]

\[ q_2 \xrightarrow{b} (q_1, q_3) \]

\[ q_1 \xrightarrow{b} (q_1, q_3) \]

\[ q_3 \xrightarrow{b} (q_3, q_3) \]

\[ q_3 \xrightarrow{a} (q_3, q_3) \]

initial state: $q_1$
Tree Automata: an Example

\[ \Sigma = \{a, b\} \]

\[ t: \]

\[ Q = \{q_1, q_2, q_3\} \]

\[ Q = \{q_1, q_2, q_3\} \]

\[ \mathcal{A} \]

\[ q_1 \xrightarrow{a} (q_1, q_3) \quad q_2 \xrightarrow{a} (q_2, q_3) \]

\[ q_1 \xrightarrow{a} (q_2, q_3) \quad q_2 \xrightarrow{b} (q_1, q_3) \]

\[ q_1 \xrightarrow{b} (q_1, q_3) \quad q_3 \xrightarrow{b} (q_3, q_3) \]

\[ q_3 \xrightarrow{a} (q_3, q_3) \]

initial state: \( q_1 \)

\[ F = \{q_2, q_3\} \]
Tree Automata: an Example

Σ = \{a, b\}

\( \mathcal{A} \)

\( Q = \{q_1, q_2, q_3\} \)

\[ q_1 \xrightarrow{a} (q_1, q_3) \quad q_2 \xrightarrow{a} (q_2, q_3) \]

\[ q_1 \xrightarrow{a} (q_2, q_3) \quad q_2 \xrightarrow{b} (q_1, q_3) \]

\[ q_1 \xrightarrow{b} (q_1, q_3) \quad q_3 \xrightarrow{b} (q_3, q_3) \]

\[ q_3 \xrightarrow{a} (q_3, q_3) \]

\text{initial state: } q_1 \quad \mathcal{F} = \{q_2, q_3\}
Tree Automata: an Example

\[ \Sigma = \{ a, b \} \]

\[ t: \]

\[ Q = \{ q_1, q_2, q_3 \} \]

\[ A \]

\[ q_1 \xrightarrow{a} (q_1, q_3) \]
\[ q_2 \xrightarrow{a} (q_2, q_3) \]

\[ q_1 \xrightarrow{a} (q_2, q_3) \]
\[ q_2 \xrightarrow{b} (q_1, q_3) \]

\[ q_1 \xrightarrow{b} (q_1, q_3) \]
\[ q_3 \xrightarrow{b} (q_3, q_3) \]

\[ q_3 \xrightarrow{a} (q_3, q_3) \]

initial state: \[ q_1 \]

\[ F = \{ q_2, q_3 \} \]
Tree Automata: an Example

$\Sigma = \{a, b\}$

$A$

$Q = \{q_1, q_2, q_3\}$

1. $a \xrightarrow{q_1} (q_1, q_3)$
2. $a \xrightarrow{q_2} (q_2, q_3)$
3. $b \xrightarrow{q_1} (q_1, q_3)$
4. $b \xrightarrow{q_2} (q_1, q_3)$
5. $a \xrightarrow{q_3} (q_3, q_3)$
6. $a \xrightarrow{q_3} (q_3, q_3)$

$\text{initial state: } q_1 \quad F = \{q_2, q_3\}$

A branch is accepting if it has infinitely many occurrences of a state from $F$ (Büchi).

A run is accepting if all its branches are accepting ($\forall$).

A tree is accepted if there exists an accepting run ($\exists$).
Tree Automata: an Example

Σ = \{a, b\}

Q = \{q_1, q_2, q_3\}

\begin{align*}
q_1 &\xrightarrow{a} (q_1, q_3) \quad q_2 \xrightarrow{a} (q_2, q_3) \\
q_1 &\xrightarrow{a} (q_2, q_3) \quad q_2 \xrightarrow{b} (q_1, q_3) \\
q_1 &\xrightarrow{b} (q_1, q_3) \quad q_3 \xrightarrow{b} (q_3, q_3) \\
q_3 &\xrightarrow{a} (q_3, q_3)
\end{align*}

initial state : \(q_1\) \quad F = \{q_2, q_3\}

A branch is accepting if it has infinitely many occurrences of a state from \(F\) (Büchi).

A run is accepting if all its branches are accepting (\(\forall\)).

A tree is accepted if there exists an accepting run (\(\exists\)).
Tree Automata: an Example

$\Sigma = \{a, b\}$

$Q = \{q_1, q_2, q_3\}$

$A$

$t:\$

- $q_1 \xrightarrow{a} (q_1, q_1)$
- $q_1 \xrightarrow{a} (q_2, q_3)$
- $q_1 \xrightarrow{b} (q_1, q_3)$
- $q_2 \xrightarrow{a} (q_2, q_3)$
- $q_2 \xrightarrow{b} (q_1, q_3)$
- $q_3 \xrightarrow{a} (q_3, q_3)$
- $q_3 \xrightarrow{b} (q_3, q_3)$

$F = \{q_2, q_3\}$

initial state: $q_1$

A branch is accepting if it has infinitely many occurrences of a state from $F$ (Büchi).

A run is accepting if all its branches are accepting ($\forall$).

A tree is accepted if there exists an accepting run ($\exists$).
We usually consider $\omega$-regular accepting conditions over branches of a run.

- **Büchi**($F$): The branch contains *infinitely many* states of $F$.
- **Parity**: The most general, can simulate any other $\omega$-regular condition.
We usually consider $\omega$-regular accepting conditions over branches of a run.

- Büchi($F$): The branch contains *infinitely many* states of $F$.
- Parity: The most general, can simulate any other $\omega$-regular condition.

Languages recognized by non-deterministic parity automata are called *regular languages*:

- Closed under *Boolean operations* (Rabin 1969)
- Emptiness problem ($L(A) \neq \emptyset$) is decidable, it is in $\text{NP} \cap \text{co-NP}$. But no known polynomial algorithm.
Remember our example specifications from the very first slides:

→ Every branch that contains infinitely many $r$ also contains infinitely many $g$. 

Remember our example specifications from the very first slides:

- Every branch that contains infinitely many \( r \) also contains infinitely many \( g \).
- Run deterministically along all branches and go to a state with priority 1 when reading an \( r \), to a state with priority 0 when reading a \( g \) and to a state with priority 2 when reading something else.
Remember our example specifications from the very first slides:

- Every branch that contains infinitely many $r$ also contains infinitely many $g$.
- Run deterministically along all branches and go to a state with priority 1 when reading an $r$, to a state with priority 0 when reading a $g$ and to a state with priority 2 when reading something else.
- There exists a subtree in which every branch contain infinitely many $a$. 
Remember our example specifications from the very first slides:

- Every branch that contains infinitely many $r$ also contains infinitely many $g$.

  Run deterministically along all branches and go to a state with priority 1 when reading an $r$, to a state with priority 0 when reading a $g$ and to a state with priority 2 when reading something else.

- There exists a subtree in which every branch contain infinitely many $a$.

  Non-deterministically go down (and trivially accepts everywhere else) the tree and at some point check deterministically that all branches from that point contains infinitely many $a$. 
Remember our example specifications from the very first slides:

→ Every branch that contains infinitely many $r$ also contains infinitely many $g$.
→ *Run deterministically along all branches and go to a state with priority 1 when reading an $r$, to a state with priority 0 when reading a $g$ and to a state with priority 2 when reading something else.*

→ There exists a subtree in which every branch contain infinitely many $a$.
→ *Non-deterministically go down (and trivially accepts everywhere else) the tree and at some point check deterministically that all branches from that point contains infinitely many $a$.*

→ Any subtree rooted at a node labeled by $a$ contains a branch with infinitely many $b$. 
Remember our example specifications from the very first slides:

- Every branch that contains infinitely many $r$ also contains infinitely many $g$.
  - Run deterministically along all branches and go to a state with priority 1 when reading an $r$, to a state with priority 0 when reading a $g$ and to a state with priority 2 when reading something else.

- There exists a subtree in which every branch contain infinitely many $a$.
  - Non-deterministically go down (and trivially accepts everywhere else) the tree and at some point check deterministically that all branches from that point contains infinitely many $a$.

- Any subtree rooted at a node labeled by $a$ contains a branch with infinitely many $b$.
  - Use complementation!
Theorem (Rabin’69)

Let $L$ be a language of infinite trees. Then the following is equivalent.

$\rightarrow$ There exists a parity tree automaton that accepts $L$.

$\rightarrow$ There exists an MSO formula that defines $L$. 
Games

Mise en bouche: Chomp
The game of Chomp is like Russian Roulette for chocolate lovers :-) 
A move consists of chomping a square out of the chocolate bar along with any squares to the right and below. Players alternate moves. The upper left square is poisoned though and the player forced to chomp it loses (and actually dies...).
The game of Chomp is like Russian Roulette for chocolate lovers :-) A move consists of chomping a square out of the chocolate bar along with any squares to the right and below. Players alternate moves. The upper left square is poisoned though and the player forced to chomp it loses (and actually dies...).

Characteristics of this game:

- Zero sum (one player wins, the other loses)
- Finite duration
- Turn based
- Perfect information
- Deterministic

For us, this is the simplest kind of games (however very few is known about this ”simple” game...).
Possible configurations:

1

2

3

4

5

6

7

8

9

Modelling of the Game $3 \times 2$
Modelling of the Game $3 \times 2$

Associated arena:
Modelling of the Game $3 \times 2$

Associated arena:
Modelling of the Game $3 \times 2$

Associated arena:
Modelling of the Game $3 \times 2$

Associated arena:
Modelling of the Game $3 \times 2$

Associated arena:
Games

Computational aspects
Two-player turn based games on graphs

- Two players: Eve & Adam
- Perfect information: everyone knows about everything
- The players move in a turn based fashion.
- No randomisation
What is a Game? The Recipe

Ingredient:

→ A graph $G$. 
What is a Game? The Recipe

Ingredients:

- A graph $G$.
- Two players: Eve & Adam.
What is a Game? The Recipe

Ingredients:
- A graph $G$.
- Two players: Eve (○) & Adam (□). Shake!
What is a Game? The Recipe

Ingredients:
- A graph $G$.
- Two players: Eve & Adam, leading an arena $\mathcal{G} = (G, V_E, V_A)$
- Play: moving a token.
What is a Game? The Recipe

Ingredients:
- A graph $G$.
- Two players: Eve & Adam, leading an arena $\mathcal{G} = (G, V_E, V_A)$
- Play: moving a token.
What is a Game? The Recipe

Ingredients:

- A graph $G$.
- Two players: Eve & Adam, leading an arena $\mathcal{G} = (G, V_E, V_A)$.
- Play: moving a token.
What is a Game? The Recipe

Ingredients:
- A graph $G$.
- Two players: Eve & Adam, leading an arena $\mathcal{G} = (G, V_E, V_A)$
- Play: moving a token.
What is a Game? The Recipe

Ingredients:

→ A graph $G$.
→ Two players: Eve & Adam, leading an arena $\mathcal{G} = (G, V_E, V_A)$
→ Play: moving a token.
What is a Game? The Recipe

Ingredients:

- A graph $G$.
- Two players: Eve & Adam, leading an arena $G = (G, V_E, V_A)$
- Play: moving a token.
- Winning condition. example: reachability.

Eve wins if the play eventually visits a final state.
What is a Game? The Recipe

Ingredients:
- A graph $G$.
- Two players: Eve & Adam, leading an arena $G = (G, V_E, V_A)$
- Play: moving a token.
- Winning condition. example: Büchi.

Eve wins iff the plays visits final states infinitely often.
What is a Game? The Recipe

Ingredients:

- A graph $G$.
- Two players: Eve & Adam, leading an arena $\mathcal{G} = (G, V_E, V_A)$
- Play: moving a token.
- Winning condition. Example: parity.

Eve wins iff the smallest colour infinitely visited is even.
Example of a Play

Winning condition: reachability.
Example of a Play

Winning condition: reachability.
Example of a Play

Winning condition: reachability.

Eve wins!
Winning condition: reachability.
Example of a Play

Winning condition: \textit{reachability}.
Example of a Play

Winning condition: \textit{reachability}.
Example of a Play

Winning condition: reachability.
Example of a Play

Winning condition: reachability.

Infinite play won by Adam!

Infinite play won by Adam!
Strategies, Winning Positions

➤ Strategie for Eve: \( \varphi : V^* V_E \rightarrow V. \)

➤ Strategie for Adam: \( \psi : V^* V_A \rightarrow V. \)
→ Strategie for Eve: \( \varphi : V^*V_E \rightarrow V \).

→ Strategie for Adam: \( \psi : V^*V_A \rightarrow V \).

→ Eve respects \( \varphi \) during \( \lambda = v_0v_1v_2 \cdots \) iff

\[
\forall i \geq 0, \ v_i \in V_E \Rightarrow v_{i+1} = \varphi(v_0 \cdots v_i)
\]
Strategies, Winning Positions

→ Strategie for Eve: $\varphi : V^*V_E \rightarrow V$.

→ Strategie for Adam: $\psi : V^*V_A \rightarrow V$.

→ Eve respects $\varphi$ during $\lambda = v_0v_1v_2 \cdots$ iff

\[
\forall i \geq 0, \ v_i \in V_E \Rightarrow v_{i+1} = \varphi(v_0 \cdots v_i)
\]

→ $\varphi$ is a winning strategy for Eve from $v_0$ iff Eve wins any play starting from $v_0$ where she respects $\varphi$. 
Strategies, Winning Positions

→ Strategie for Eve : \( \varphi : V^*V_E \rightarrow V \).

→ Strategie for Adam : \( \psi : V^*V_A \rightarrow V \).

→ Eve respects \( \varphi \) during \( \lambda = v_0v_1v_2 \cdots \) iff

\[
\forall i \geq 0, \ v_i \in V_E \Rightarrow v_{i+1} = \varphi(v_0 \cdots v_i)
\]

→ \( \varphi \) is a winning strategy for Eve from \( v_0 \) iff Eve wins any play starting from \( v_0 \) where she respects \( \varphi \).

→ Winning positions: \( v_0 \) is winning for Eve iff she has a winning strategy from \( v_0 \).
Winning condition: Büchi.
Winning condition: Büchi.
Central question:

For a given game $G$, what are the winning positions for Eve ($W_E$)? for Adam ($W_A$)?
Central question:

For a given game $\mathcal{G}$, what are the winning positions for Eve ($W_E$)? for Adam ($W_A$)?

The game $\mathcal{G}$ is determined iff $W_E \cup W_A = V$. 
Central question:

For a given game $G$, what are the winning positions for Eve ($W_E$)? for Adam ($W_A$)?

The game $G$ is determined iff $W_E \cup W_A = V$.

*Where topology matters...*

**Theorem (Martin, 1975)**

*Borel games are determined*
Central question:
For a given game $\mathcal{G}$, what are the winning positions for Eve ($W_E$)? for Adam ($W_A$)?

The game $\mathcal{G}$ is determined iff $W_E \cup W_A = V$.

Where topology matters...

Theorem (Martin, 1975)
Borel games are determined

Corollary
Reachability / Büchi / Parity games are determined
Reachability Games: Resolution

Framework

- Game graph $\mathcal{G} = (V_E, V_A, E)$.
- $F \subseteq V$: final vertices.
- $\mathcal{G}$: reachability game played on $\mathcal{G}$. 
Reachability Games: Resolution

Framework

- Game graph $\mathcal{G} = (V_E, V_A, E)$.
- $F \subseteq V$: final vertices.
- $\mathcal{G}$: reachability game played on $\mathcal{G}$.

Questions:

- For a given $v \in V$, decide whether $v \in W_E$.
- More generally, compute $W_E$.
- Finally, compute a winning strategy.
Remembering the chocolate...

Possible configurations

1
2
3
4
5
6
7
8
9
Remembering the chocolate...

Associated game graph
Remembering the chocolate... 

Associated game graph
Remembering the chocolate...  

Associated game graph
Remembering the chocolate...

Associated game graph
Remembering the chocolate...
Algorithm

**Initialisation:** final vertices are winning for Eve.
**Algorithm**

**Initialisation:** final vertices are winning for Eve.

**Induction:**

1. A vertex of Eve such that there is an edge to a winning vertex is winning. A strategy for Eve from this vertex consists in going to the winning vertex and then follow a winning strategy from this vertex.

2. A vertex of Adam such that all outgoing edges goes to winning vertices for Eve is winning for Eve.
Algorithm

Initialisation: final vertices are winning for Eve.

Induction:

(1) A vertex of Eve such that there is an edge to a winning vertex is winning. A strategy for Eve from this vertex consists in going to the winning vertex and then follow a winning strategy from this vertex.

(2) A vertex of Adam such that all outgoing edges go to winning vertices for Eve is winning for Eve.

Theorem

One can compute in polynomial time the winning positions in a reachability game and construct memoryless strategies for both players.
Büchi games

To win in a Büchi game with final states $F$ Eve must:

- Reach a final vertex $f_1 \in F$, 
Büchi games

To win in a Büchi game with final states $F$ Eve must:

- Reach a final vertex $f_1 \in F$, 
- from which she can reach another final vertex $f_2 \in F$
To win in a Büchi game with final states $F$ Eve must:

- Reach a final vertex $f_1 \in F$,
- from which she can reach another final vertex $f_2 \in F$
- from which she can reach another final vertex $f_3 \in F$
- ...
Büchi games

To win in a Büchi game with final states $F$ Eve must:

- Reach a final vertex $f_1 \in F$,
- from which she can reach another final vertex $f_2 \in F$
- from which she can reach another final vertex $f_3 \in F$
- ...

Hence, one needs to

- Determine the largest subset $F^\infty$ of $F$ such that Eve has a strategy to always come back in
- and then consider the vertices from which she can force to reach $F^\infty$. 
To win in a Büchi game with final states $F$ Eve must:

- Reach a final vertex $f_1 \in F$,
- from which she can reach another final vertex $f_2 \in F$
- from which she can reach another final vertex $f_3 \in F$
- ...

Hence, one needs to

- Determine the largest subset $F^\infty$ of $F$ such that Eve has a strategy to always come back in
- and then consider the vertices from which she can force to reach $F^\infty$.

**Theorem**

*One can compute in polynomial time the winning positions in a Büchi game and construct memoryless strategies for both players.*
Tree Automata & Games

Acceptance & Emptiness
Parity Tree Automata: Emptiness

Non-deterministic parity tree automata: \( \mathcal{A} = \langle Q, \Sigma, \Delta, q_{\text{in}}, \text{col} \rangle \)

- \( Q \): control states
- \( \Sigma \): labels alphabet
- \( \Delta \subseteq Q \times \Sigma \times Q \times Q \): transition relation
- \( q_{\text{in}} \): initial state
- \( \text{col} : Q \to \mathbb{N} \): colouring function

Run on a \( \Sigma \)-labeled (infinite binary) tree \( t \): \( Q \)-labelling of \( t \) consistent with \( \Delta \)

\( \Delta = \{ \cdots (q_{\text{in}}, a, p, p) \quad (p, b, q, p)(p, b, p, p) \cdots \} \)

A branch is **accepting** iff the smallest colour infinitely often visited is even

A run is **accepting** iff all its branches are accepting

A tree is **accepted** iff there is an accepting run over it.
Non-deterministic parity tree automata: $\mathcal{A} = \langle Q, \Sigma, \Delta, q_{\text{in}}, \text{col} \rangle$

Acceptance game:
- Eve and Adam move a token along a branch of the tree
- Eve chooses the transition, \textit{i.e.} she plays elements in $\Delta$
- Adam chooses the branch, \textit{i.e.} he plays elements in $\{0, 1\}$
- Eve wins iff the branch is accepting

- Strategies of Eve $\simeq$ Runs
- Given a strategy of Eve, strategies of Adam $\simeq$ branches
- Eve wins the game iff exists a run s.t. all branch are accepting iff the tree is accepted
Non-deterministic parity tree automata: $\mathcal{A} = \langle Q, \Sigma, \Delta, q_{\text{in}}, \text{col} \rangle$

Emptiness game: one needs to construct the tree + accepting run

- Eve and Adam build a branch
- Eve chooses the node labels and the transition, \textit{i.e.} she plays elements in $\Delta$
- Adam chooses the branch, \textit{i.e.} he plays elements in $\{0, 1\}$
- Eve wins iff the branch is accepting
Parity Tree Automata: Emptiness

Non-deterministic parity tree automata: \( \mathcal{A} = (Q, \Sigma, \Delta, q_{\text{in}}, \text{col}) \)

Emptiness game: one needs to construct the tree + accepting run

- Eve and Adam build a branch
- Eve chooses the node labels and the transition, i.e. she plays elements in \( \Delta \)
- Adam chooses the branch, i.e. he plays elements in \( \{0, 1\} \)
- Eve wins iff the branch is accepting

\[
\forall (q, a, q_0, q_1) \in \Delta
\]

![Diagram](attachment://parity_tree_diagram.png)
Non-deterministic parity tree automata: $A = \langle Q, \Sigma, \Delta, q_{\text{in}}, \text{col} \rangle$

Emptiness game: one needs to construct the tree + accepting run

→ Eve and Adam build a branch
→ Eve chooses the node labels and the transition, i.e. she plays elements in $\Delta$
→ Adam chooses the branch, i.e. he plays elements in $\{0, 1\}$
→ Eve wins iff the branch is accepting

- Strategies of Eve $\simeq$ Trees with run
- Given a strategy of Eve, strategies of Adam $\simeq$ branches
- Eve wins the game iff exists a tree + a run s.t. all branch are accepting iff exists a tree accepted

\[ \forall (q, a, q_0, q_1) \in \Delta \]

Diagram:
- Node $q$ transitions to $q_0$ and $q_1$.
- Node $q_0$ transitions to $q$.
- Node $q_1$ transitions to $q$.
- Strategies of Eve $\simeq$ Trees with run.
- Given a strategy of Eve, strategies of Adam $\simeq$ branches.
- Eve wins the game iff exists a tree + a run s.t. all branch are accepting iff exists a tree accepted.
What is Next?

Aka why should you come back on Wednesday?
What We Did Today

Two equivalent formalism to specify properties of trees

- Monadic second order logic
- Parity tree automata

A tool to work with model-checking problems: parity games
What We Will Discuss on Wednesday

Two equivalent formalism to describes complex trees

- Higher-order Recursion Schemes
- Collapsible Pushdown Automata

Tool to work with model-checking problems: collapsible pushdown parity games