Applied Automata Theory

Roland Meyer

TU Kaiserslautern

Table of Contents I

Table of Contents II

Part A Finite Words

1. Regular Languages and Finite Automata

Basic Notions

Definition (Words)

- Finite alphabet = finite set of letters $\Sigma = \{a, b, c, \ldots, n\}$
- Finite word over Σ = finite sequence of letters $w = a_0 \cdot \ldots \cdot a_{n-1}$ with $a_i \in \Sigma$ for all $i \in [0, n-1]$
- Length of word w is $|w| := n$
- Empty word ε with $|\varepsilon| := 0$
- i-th symbol in w denoted by $w(i) := a_i$
- Set of all finite words over Σ is Σ^*
- Set of all non-empty words over Σ is $\Sigma^+ := \Sigma^* \setminus \{\varepsilon\}$
- Concatenation of words $w, v \in \Sigma^*$ is $w \cdot v \in \Sigma^*$

Basic Notions

Definition (Languages and operations)

- Language is a (typically infinite) set $L \subseteq \Sigma^*$
- Set-theoretic operations apply to languages $L_1, L_2 \subseteq \Sigma^*$:

$$\underset{\text{union}}{L_1 \cup L_2} \qquad \underset{\text{intersection}}{L_1 \cap L_2} \qquad \underset{\text{difference}}{L_1 \setminus L_2} \qquad \underset{\text{complement}}{\overline{L_1} := \Sigma^* \setminus L_1}$$

- Concatenation $L_1 \cdot L_2 := \{w \cdot v \in \Sigma^* \mid w \in L_1 \text{ and } v \in L_2\}$
- Kleene star $L^* := \bigcup_{i \in \mathbb{N}} L^i$ with $L^0 := \{\varepsilon\}$ and $L^{i+1} := L \cdot L^i$ for all $i \in \mathbb{N} := \{0, 1, 2, \ldots\}$.

Regular Languages

Definition (Regular languages)

The class of regular languages over alphabet Σ, denoted by REG_Σ, is the smallest class of languages that satisfies

(1) $\emptyset \in \mathrm{REG}_\Sigma$ and $\{a\} \in \mathrm{REG}_\Sigma$ for all $a \in \Sigma$ and

(2) if $L_1, L_2 \in \mathrm{REG}_\Sigma$ then also $L_1 \cup L_2, L_1 \cdot L_2, L_1^* \in \mathrm{REG}_\Sigma$.

So every regular language is obtained by application of finitely many operations in (2) from the languages in (1).

Notation

- Avoid brackets: * binds stronger than $\cdot$ binds stronger than $\cup$
- Write $\{a\}$ as a

Example: $\varepsilon \cup (a \cup b)^* \cdot b$. We have ε since $\{\varepsilon\} = \emptyset^*$.

Closure Properties of Regular Languages

Observation

- Finite sets of words form regular languages
- Regular languages not closed under infinite unions
- By definition, regular languages closed under *, $\cdot$, $\cup$

Goal

- Show that REG_Σ also closed under remaining operations on sets: $\cap$, $\overline{}$, $\setminus$. Note that $L_1 \setminus L_2 = L_1 \cap \overline{L_2}$.
- Need alternative characterization of regular languages
- It is not only about proving closure: need a representation where operations can be computed efficiently
- Languages are infinite sets. Finite representations not always easy to find (one of the sports of TCS)

Finite Automata: Syntax

Definition (Finite automaton)

A non-deterministic finite automaton (NFA) is a tuple $A = (\Sigma, Q, q_0, \rightarrow, Q_F)$ with

- alphabet Σ,
- finite set of states Q, initial state $q_0 \in Q$, final states $Q_F \subseteq Q$, and
- transition relation $\rightarrow \subseteq Q \times \Sigma \times Q$. Write $q \xrightarrow{a} q'$ rather than $(q, a, q') \in \rightarrow$.

Size of A is $|A| := |\Sigma| + |Q| + 1 + |Q_F| + |\rightarrow|$. Note

$$|A| \leq |\Sigma| + |Q| + 1 + |Q| + |Q|^2|\Sigma| \in \mathbf{O}(|Q|^2|\Sigma|).$$

For Σ fixed, this is in $\mathbf{O}(|Q|^2)$. Number of states is important.

Finite Automata: Semantics

Definition (Run and language)

Run of $A = (\Sigma, Q, q_0, \rightarrow, Q_F)$ is a sequence

$$q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \ldots q_{n-1} \xrightarrow{a_{n-1}} q_n.$$

Also say this is a run of A on word $w := a_0 \ldots a_{n-1}$.
We write $q_0 \xrightarrow{w} q_n$ if there are intermediary states.
Run is accepting if $q_n \in Q_F$.
Language of A is

$$L(A) := \{w \in \Sigma^* \mid q_0 \xrightarrow{w} q \text{ with } q \in Q_F\}.$$

If $L = L(A)$ we say L is accepted or recognized by automaton A.

From Regular Languages to Finite Automata

Goal

Show that regular languages are recognizable by NFAs.

Idea

Apply operations from REG to NFAs.

Proposition (NFA languages are closed under $\cdot$ and $\cup$)

Consider two NFAs A_1 and A_2.

(1) There is an NFA $A_1 \cdot A_2$ so that $L(A_1 \cdot A_2) = L(A_1) \cdot L(A_2)$.

(2) There is an NFA $A_1 \cup A_2$ so that $L(A_1 \cup A_2) = L(A_1) \cup L(A_2)$.

From Regular Languages to Finite Automata

Proposition (NFA languages are closed under *)

Consider an NFA A. There is an NFA A^* with $L(A^*) = L(A)^*$.

Construction

Let $A = (\Sigma, Q, q_0, \rightarrow, Q_F)$. Define

$$A^* := (\Sigma, Q \cup \{q_0'\}), q_0', \rightarrow \cup \rightarrow', Q_F \cup \{q_0'\})$$

where $q_0' \xrightarrow{a}' q$ if $q_0 \xrightarrow{a} q$ and $q_f \xrightarrow{a}' q$ if $q_0 \xrightarrow{a} q$ for all $q_f \in Q_F$.

An illustration is given in the handwritten notes.

Theorem

If $L \in \mathrm{REG}_\Sigma$ then there is an NFA A with $L = L(A)$.

From Finite Automata to Regular Languages

Goal

Show the reverse: NFA languages are regular.

Idea

- Represent NFA with $n \in \mathbb{N}$ states by system of n equations
- Solve this system using Arden's lemma

Lemma (Arden 1960)

Let $U, V \subseteq \Sigma^*$ with $\varepsilon \notin U$. Consider $L \subseteq \Sigma^*$. Then

$$L = U \cdot L \cup V \quad \textit{iff} \quad L = U^* \cdot V.$$

Proof.

Please see the handwritten notes. □

From Finite Automata to Regular Languages

Observation

- Only-if direction ($\Rightarrow$) in Arden's lemma means such an equation has a unique solution.
- Use this as tool to construct regular language for a given NFA.

Theorem

If L is recognized by an NFA, then L is regular.

Proof sketch.

Please see the handwritten notes. □

Example

Please see the handwritten notes.

Deterministic Finite Automata

Definition

An NFA $A = (\Sigma, Q, q_0, \rightarrow, Q_F)$ is called deterministic or DFA if for all $a \in \Sigma$ and all $q \in Q$

$$\text{there is precisely one state } q' \in Q \text{ with } q \xrightarrow{a} q'.$$

Deterministic automata are convenient in applications.

Goal

Show that for every NFA A there is a deterministic finite automaton A' with $L(A) = L(A')$.

Powerset Construction

Theorem (Rabin & Scott 1959)

For every NFA A with $n \in \mathbb{N}$ states there is a DFA A' with at most 2^n states that satisfies $L(A) = L(A')$.

Construction: Powerset

Let $A = (\Sigma, Q, q_0, \rightarrow, Q_F)$. Set $A' := (\Sigma, \mathbb{P}(Q), \{q_0\}, \rightarrow', Q'_F)$ with

$$Q_1 \xrightarrow{a}' Q_2 \text{ where } Q_2 := \{q_2 \in Q \mid q_1 \xrightarrow{a} q_2 \text{ for some } q_1 \in Q_1\}$$

and moreover

$$Q'_F := \{Q' \subseteq Q \mid Q' \cap Q_F \neq \emptyset\}.$$

Note that A' is deterministic. For every $a \in \Sigma$ and $Q_1 \subseteq Q$ there is a goal state (which may be $\emptyset \in \mathbb{P}(Q)$). This goal state is unique.

Closure under Complementation

Consequence of Rabin & Scott: closure of regular languages under complementation

Note

Consider NFA A. It is not easy to find NFA for $\overline{L(A)}$. Why?

$$L(A) = w \in \Sigma^* \text{ so that there is an accepting run of } A \text{ on } w.$$
$$\overline{L(A)} = w \in \Sigma^* \text{ so that all runs of } A \text{ on } w \text{ do not accept.}$$

To give an automaton for $\overline{L(A)}$, we thus have to translate this $\forall$-quantifier into an $\exists$-quantifier. For DFAs A', this works:

$$L(A') = w \in \Sigma^* \text{ so that there is an accepting run of } A' \text{ on } w.$$
$$\overline{L(A')} = w \in \Sigma^* \text{ so that there is a run of } A' \text{ on } w \text{ that does not accept.}$$

Closure under Complementation

Proposition (Closure under $\overline{}$)

Consider a DFA A. Then there is a DFA $\overline{A}$ with $L(\overline{A}) = \overline{L(A)}$.

Construction: Swap final states

Let $A = (\Sigma, Q, q_0, \rightarrow, Q_F)$. Define $\overline{A} := (\Sigma, Q, q_0, \rightarrow, Q \setminus Q_F)$.

Summary

Summary

Let $L = L(A)$ for an NFA A with $n \in \mathbb{N}$ states

- There are DFAs for L and $\overline{L}$ with at most 2^n states
- The bound is optimal: there is a family $(L_n)_{n\in\mathbb{N}}$ of languages L_n that

  are recognized by an NFA with $n + 1$ states but
  that cannot be recognized by a DFA with $< 2^n$ states.

- Only considering states reachable from q_0 often yields much smaller automata

Decidability and Complexity

Problems

Consider an NFA A.

Emptiness:	$L(A) = \emptyset$?
Universality:	$L(A) = \Sigma^*$?
Membership:	Given also $w \in \Sigma^*$. Does $w \in L(A)$ hold?

Focus on emptiness and reduce remaining problems to it

More Decidable Problems

Intersection:	$L(A_1) \cap L(A_2) = \emptyset$?
Equivalence:	$L(A_1) = L(A_2)$?
Inclusion:	$L(A_1) \subseteq L(A_2)$?

Emptiness

Theorem

Emptiness for NFAs can be solved in time $\mathbf{O}(|\rightarrow|)$.

Idea

Compute reachable states $R_0 \subseteq R_1 \subseteq \ldots$ until fixed point $R_k = R_{k+1}$

Proof.

Let $A = (\Sigma, Q, q_0, \rightarrow, Q_F)$. Define $R_0 := \{q_0\}$ and

$$R_{i+1} := R_i \cup \{q' \in Q \mid q \in R_i \text{ and } q \xrightarrow{a} q' \text{ for some } a \in \Sigma\}$$

Consider $k \in \mathbb{N}$ with $R_k = R_{k+1}$. If $R_k \cap Q_F \neq \emptyset$ return $L(A)$ not empty. Otherwise return $L(A)$ empty.

- Reaches fixed point after at most $|Q|$ steps. Gives $\mathbf{O}(|Q||\rightarrow|)$.
- Sufficient to use each $q \xrightarrow{a} q'$ at most once. Linear in $|\rightarrow|$.

$\square$

2. Weak Monadic Second-Order Logic

Weak Monadic Second-Order Logic

Goal

NFAs (and also regular languages) operational models
Logics are declarative: specifications often more intuitive and more concise

- Solve decidability problems in logic: satisfiability and validity
- With automata: emptiness checks

WMSO: Syntax

Fix alphabet Σ (parameter of the logic)

Need signature $Sig = (Fun, Pred)$

- Here, purely relational signature with $Fun = \emptyset$
- Define $Pred := \{< /_2, suc/_2\} \cup \{P_a/_1 \mid a \in \Sigma\}$.

Consider two countably infinite sets

- $V_1 = \{x, y, z, \ldots\}$ of first-order variables
- $V_2 = \{X, Y, Z, \ldots\}$ of second-order variables

Definition (Syntax of WMSO)

Formulas in WMSO (over Sig, V_1, V_2) are defined by

$$\varphi ::= \underbrace{x < y \mid suc(x, y) \mid P_a(x)}_{\text{Predicates from signature}} \mid X(x) \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \exists x : \varphi \mid \exists X : \varphi$$

where $x, y \in V_1$ and $X \in V_2$.

WMSO: Syntax

Definition (Notation and abbreviations)

Notation to make signature explicit:

$\mathrm{WMSO} = \mathrm{WMSO}[<, suc]$: all WMSO formulas

$\mathrm{WMSO}[<], \mathrm{WMSO}[suc]$: formulas that only use predicates $<$ and *suc*

$\mathrm{FO}[<, suc], \mathrm{FO}[<], \mathrm{FO}[suc]$: first-order formulas (over V_1, only)

Abbreviations: Let $\varphi, \psi \in \mathrm{WMSO}$. We set

$$\begin{aligned} \varphi \wedge \psi &:= \neg(\neg\varphi \vee \neg\psi) & \varphi \rightarrow \psi &:= \neg\varphi \vee \psi \\ \forall x : \varphi &:= \neg\exists x : \neg\varphi & \forall X : \varphi &:= \neg\exists X : \neg\varphi \\ x \leq y &:= \neg(y < x) & x = y &:= x \leq y \wedge y \leq x \\ \mathit{first}(x) &:= \neg\exists y : y < x & \mathit{last}(x) &:= \neg\exists y : x < y \end{aligned}$$

WMSO: Syntax

Definition (Bound and free variables)

Consider formula $\varphi \in$ WMSO.

- Variable $x \in V_1$ is bound in φ if syntax tree contains occurrence of $\exists x$ above x. Similar for $X \in V_2$.
- Variable that occurs in φ and is not bound is free in φ
- Write $\varphi(x_1, \ldots, x_m, X_1, \ldots X_n)$ to indicate that free variables of φ among $x_1, \ldots, X_n$
- Formula without free variables called closed or sentence
- Assume bound and free variables disjoint. Can always be achieved by α-conversion of bound variables:

  (Bad) $x < z \wedge \forall x : x < y$ $\qquad$ $x < z \wedge \forall x' : x' < y$ (Good)

Example

$\neg \exists y : y < x$	y bound, x free, notation $\mathit{first}(x)$
$\exists x : \mathit{first}(x) \wedge X(x)$	x bound, X free

WMSO: Semantics

Intuitive meaning

First-order variables: natural numbers $\mathbb{N}$ (positions in a word)

$x < y$, $suc(x, y)$: usual $<$ and successor on $\mathbb{N}$

Second-order variables: finite sets of natural numbers

$X(x)$: x is in set X

What does WMSO stand for?

W = Weak: quantify over finite sets

M = monadic: quantify over elements of the domain. Polyadic = quantify over tuples.

SO = second-order: with quantification over sets of elements. Third-order with quantification over sets of sets of elements.

WMSO: Semantics

Example

$$\exists X : (\exists x : \mathit{first}(x) \land X(x)) \land (\forall x : X(x) \rightarrow \exists y : x < y \land X(y))$$

There is a finite set of natural numbers

- that contains 0 (and thus is not empty) and
- for every element contains a larger one.

Such a set has to be infinite

- Formula is unsatisfiable

WMSO: Semantics

To give semantics, need *Sig*-structures $S = (D^S, <^S, suc^S, ({P_a}^S)_{a\in\Sigma})$ with

$$D^S = \text{domain of elements (to talk about and quantify over)}$$

$$P_a^S \subseteq D^S, <^S, suc^S \subseteq D^S \times D^S = \text{interpretation of predicate symbols}$$

Restrict ourselves to particular *Sig*-structures that are associated to words

Definition (Word structures)

Let $w \in \Sigma^*$. Its word structure is $S(w) := (D^w, <^w, suc^w, (P_a^w)_{a\in\Sigma})$ with

$$D^w := \{0, \ldots, |w|-1\} \qquad <^w := <^{\mathbb{N}} \cap (D^w \times D^w)$$

$$suc^w := \{(0,1), \ldots, (|w|-2, |w|-1)\} \qquad P_a^w := \{k \in D^w \mid w(k) = a\}$$

WMSO: Semantics

Definition (Satisfaction relation $\models$ for WMSO)

Let $w \in \Sigma^*$ and $\varphi \in \text{WMSO}$. To define whether φ holds in $S(w)$, need an interpretation $I : V_1 \cup V_2 \nrightarrow D^w \cup \mathbb{P}(D^w)$ that assigns (sets of) positions to free variables in φ (maybe to others, not important). With this:

$$\begin{array}{lll}
S(w), I \models P_a(x) & \text{if} & P_a^w(I(x)) \\
S(w), I \models suc(x,y) & \text{if} & suc^w(I(x), I(y)) \\
S(w), I \models x < y & \text{if} & I(x) <^w I(y) \\
S(w), I \models X(x) & \text{if} & I(x) \in I(X) \\
S(w), I \models \neg\varphi & \text{if} & S(w), I \not\models \varphi \\
S(w), I \models \varphi_1 \vee \varphi_2 & \text{if} & S(w), I \models \varphi_1 \text{ or } S(w), I \models \varphi_2 \\
S(w), I \models \exists x : \varphi & \text{if} & \text{there is } k \in D^w \text{ so that } S(w), I[k/x] \models \varphi \\
S(w), I \models \exists X : \varphi & \text{if} & \text{there is } M \subseteq D^w \text{ (potentially empty)} \\
 & & \qquad\qquad \text{so that } S(w), I[M/X] \models \varphi.
\end{array}$$

Here, $I[k/x](x) := k$ and $I[k/x](y) := I(y)$ for $y \neq x$. Similar for X.

WMSO: Semantics

Definition (Equivalence)

Two formulas $\varphi, \psi \in$ WMSO are called equivalent, denoted by $\varphi \equiv \psi$, if for all $w \in \Sigma^*$ and all $I : V_1, V_2 \nrightarrow D^w \cup \mathbb{P}(D^w)$ we have

$$S(w), I \models \varphi \quad \text{iff} \quad S(w), I \models \psi.$$

Remark

- The empty word ε has the empty word structure with $D^\varepsilon = \emptyset$.
- The empty word does not satisfy first-order existential quantifiers. It does satisfy all first-order universal quantifiers:

$$S(\varepsilon) \not\models \exists x : x = x \qquad\qquad S(\varepsilon) \models \forall x : \neg(x = x)$$

- The empty word does satisfy second-order existential quantifiers

$$S(\varepsilon) \models \exists X : \forall x : X(x) \to P_a(x)$$

WMSO: Semantics

Interested in closed formulas

- For φ closed, $S(w), I \models \varphi$ does not depend on I
- Yet need I for satisfaction of subformulas

Definition (Satisfiability, validity, model)

Consider closed formula $\varphi \in \text{WMSO}$

- Say φ is satisfiable if there is $w \in \Sigma^*$ so that $S(w) \models \varphi$
- In this case, call $S(w)$ a model of φ
- Formula without model is unsatisfiable
- If $S(w) \models \varphi$ for all $w \in \Sigma^*$, then φ is valid

Observation

φ is valid iff $\neg\varphi$ is unsatisfiable.

WMSO: Semantics

Set of words that satisfy a formula form a language

Definition (Language defined by φ, definability)

Consider closed formula $\varphi \in \text{WMSO}$. The language defined by φ is

$$L(\varphi) := \{w \in \Sigma^* \mid S(w) \models \varphi\}.$$

Language $L \subseteq \Sigma^*$ is WMSO-definable if there is a formula $\varphi \in \text{WMSO}$ with $L = L(\varphi)$.

Notions $\text{WMSO}[suc], \text{WMSO}[<], \text{FO}[suc], \text{FO}[<]$-definable by restricting φ.

Example

Please see the handwritten notes.

First Hierarchy of Languages

Distinguish between
$\mathrm{FO}[suc], \mathrm{FO}[<], \mathrm{FO}[<, suc], \mathrm{WMSO}[suc], \mathrm{WMSO}[<], \mathrm{WMSO}[<, suc]$-definability

Lemma

$$\begin{array}{lcll}
L \text{ is } FO[<, suc]\text{-definable} & \text{iff} & L \text{ is } FO[<]\text{-definable} & (1)\\
L \text{ is } WMSO[<, suc]\text{-definable} & \text{iff} & L \text{ is } WMSO[<]\text{-definable} & (2)\\
L \text{ is } WMSO[<, suc]\text{-definable} & \text{iff} & L \text{ is } WMSO[suc]\text{-definable} & (3)\\
L \text{ is } WMSO[<, suc]\text{-definable} & \text{iff} & L \text{ is } WMSO_0\text{-definable.} & (4)
\end{array}$$

$WMSO_0$ = WMSO without first-order variables but with new predicates:

$$X \subseteq Y, Sing(X), Suc(X, Y), X \subseteq P_a \text{ with } a \in \Sigma$$

Meaning: X is subset of Y, X is a singleton set, X and Y are singletons $X = \{x\}$ and $Y = \{y\}$ with $suc(x, y)$, all positions in X have letter a.

WMSO vs. FO: later. $\mathrm{FO}[suc]$ vs. $\mathrm{FO}[<]$: not this lecture.

From Finite Automata to WMSO

Goal

Establish REG $=$ WMSO-definable.

First Subgoal: $\subseteq$

Show that regular languages are definable in WMSO

Theorem (Büchi I, 1960)

Let A be an NFA. We can effectively construct a WMSO-formula φ_A so that $L(\varphi_A) = L(A)$.

Proof.

Please see handwritten notes. □

Example

Please see handwritten notes.

From WMSO to Finite Automata

Second Subgoal: $\supseteq$

- Show that WMSO-definable languages are regular
- To this end, represent all models of a WMSO-formula by an NFA

Approach

Proceed by induction on structure of φ

Problem

$\exists X : \varphi(X)$ is closed but $\varphi(X)$ contains X free

Theorem (Büchi II, 1960)

Let $\varphi \in WMSO$. We can effectively construct an NFA A_φ that satisfies $L(A_\varphi) = L(\varphi)$.

Büchi's Theorem

Theorem (Büchi I+II, 1960)

A language $L \subseteq \Sigma^*$ is regular iff it is WMSO-definable.

Corollary

It is decidable whether a WMSO-formula is satisfiable/valid.

Worst-case complexity of automata construction

Consider NFAs A and B with at most $n \in \mathbb{N}$ states.

$$A \cup B \rightsquigarrow 2n+1 \text{ states} \qquad \overline{A} \rightsquigarrow 2^n \text{ states} \qquad \pi_x(A) \rightsquigarrow n \text{ states.}$$

Thus, formula with $k \in \mathbb{N}$ connectives may yield automaton of size

$$\underbrace{2^{2^{\cdot^{\cdot^{\cdot^{2^c}}}}}}_{k\text{-times}} \quad \text{with } c \in \mathbb{N}.$$

Consequences of Büchi's Theorem

Observation

Construction from NFAs to WMSO gave formulas of particular shape.
Existential WMSO, denoted by ∃WMSO, is restriction of WMSO to formulas

$$\exists X_0 : \ldots \exists X_n : \varphi,$$

where φ does not contain second-order quantification.

Corollary

Every closed formula $\varphi \in$ WMSO has an equivalent closed formula $\psi \in \exists$WMSO. Thus a language is WMSO-definable iff it is definable in ∃WMSO.

Proof.

Let $\varphi \in$ WMSO. Build A_φ with Büchi II. Build $\psi = \varphi_{A_\varphi}$ with Büchi I. □

3. Star-free Languages

Star-free Languages

Goals

(1) Show that FO[<] defines a strict subclass of regular languages

(2) Find alternative characterization:

FO[<]-definable iff represented by star-free regular expression

Recapitulation

First-order formulas are WMSO-formulas without second-order variables
Example over $\Sigma = \{a, b, c\}$:

$$\varphi := \forall x : P_a(x) \to \exists y : x < y \land P_b(y)$$

States that every letter a is followed by a letter b:

$$L(\varphi) = \{a, b, c\}^* \cdot b \cdot \{b, c\}^* \cup \{b, c\}^*$$

Note: $\mathit{first}(x)$, $\mathit{last}(x)$, $x = y$ still in FO[<]

Star-free Languages

Towards Goal (1)

Known: FO[$<$]-definable languages are regular

Show: Language $(aa)^*$ is not FO[$<$]-definable:

For all $\psi \in$ FO[$<$] we have $L(\psi) \neq (aa)^*$.

Hence: FO[$<$]-definable languages form strict subclass of regular languages

Ehrenfeucht-Fraïssé Games

Tool from finite model theory (logic) for proving inexpressibility results

The game — informally

Set-up:

- Two players: spoiler and duplicator
- Two words: v and w over Σ
- Number of rounds: $k \in \mathbb{N}$
- Potentially some existing edges between positions

Per round

- Spoiler selects position in v or w
- Duplicator selects fresh position in other word and connects them by a line
  - Positions must have same letter (preserve P_a)
  - New line not allowed to cross existing lines (preserve $<$)
- Next round

Winning

- Duplicator loses if cannot reply
- Duplicator wins if number of rounds passes without loss

Ehrenfeucht-Fraïssé Games

Definition (Partial isomorphism between word structures)

Consider $S(v)$ and $S(w)$. A partial isomorphism between $S(v)$ and $S(w)$ is a partial function $p : D^v \nrightarrow D^w$ so that

(1) Function p is injective.

(2) For all $x \in dom(p)$ and all $a \in \Sigma$ we have $P_a^v(x)$ iff $P_a^w(p(x))$.

(3) For all $x, y \in dom(p)$ we have $x <^v y$ iff $p(x) <^w p(y)$.

Let $\overline{s} = (s_1, \dots, s_n)$ and $\overline{t} = (t_1, \dots, t_n)$ two vectors of positions in D^v and D^w. Write $\overline{s} \mapsto \overline{t}$ for partial function $p := \{(s_1, t_1), \dots, (s_n, t_n)\}$.

Understanding requirements (1) to (3) wrt. informal game

(1) = fresh position   (2) = identical labels   (3) = no crossing edges

Interpretation of EF-games

- Let $S(v), S(w)$ two word structures with designated positions $\overline{s}, \overline{t}$
- Duplicator tries to establish partial isomorphism, starting from $\overline{s} \mapsto \overline{t}$
- Spoiler tries to avoid this

Ehrenfeucht-Fraïssé Games

Definition (EF-Game)

Consider $S(v), S(w)$ with $\overline{s}, \overline{t}$ vectors of positions in D^v and D^w. Let $k \in \mathbb{N}$.
An EF-game $G_k((S(v), \overline{s}), (S(w), \overline{t}))$ has the following elements and rules:

- k rounds
- Initial configuration $\overline{s} \mapsto \overline{t}$
- Given configuration r, a round consists of the following moves:
  - Spoiler chooses $s \in D^v$ or $t \in D^w$
  - Duplicator chooses $t \in D^w$ or $s \in D^v$
  - Game continues with $r \cup \{(s, t)\}$ as new configuration

Duplicator wins k rounds if last configuration is partial isomorphism.
Duplicator wins $G_k((S(v), \overline{s}), (S(w), \overline{t}))$ if has a winning strategy: whatever moves spoiler does, duplicator can win k rounds.

Ehrenfeucht-Fraïssé Theorem

Where is this going?

Now we know what an EF-game does: compares word structures $S(v)$ and $S(w)$. So what? Overall goal is EF-theorem:

duplicator wins $G_k((S(v), \overline{s}), (S(w), \overline{t}))$ iff v and w cannot be distinguished by FO[$<$]-formulas of quantifier-depth $\leq k$.

Ehrenfeucht-Fraïssé Theorem

Definition (Quantifier-depth)

The quantifier-depth $qd(\varphi)$ with $\varphi \in \mathrm{FO}[<]$ is the maximal nesting depth of quantifiers in φ:

$$\begin{aligned} qd(x < y) &:= 0 & qd(P_a(x)) &:= 0 \\ qd(\neg\varphi) &:= qd(\varphi) & qd(\varphi_1 \vee \varphi_2) &:= max\{qd(\varphi_1), qd(\varphi_2)\} \\ qd(\exists x : \varphi) &:= 1 + qd(\varphi) \end{aligned}$$

Definition (k-equivalence)

Consider $S(v), S(w)$ with $\overline{s}$, $\overline{t}$. Then $(S(v), \overline{s})$ and $(S(w), \overline{t})$ are k-equivalent, denoted $(S(v), \overline{s}) \equiv_k (S(w), \overline{t})$, if for all $\varphi(\overline{x})$ with $qd(\varphi) < k$ we have

$$S(v), I[\overline{s}/\overline{x}] \models \varphi \quad \text{iff} \quad S(w), I[\overline{t}/\overline{x}] \models \varphi.$$

In the case of empty sequences $\overline{s} = \varepsilon = \overline{t}$, equivalence $S(v) \equiv_k S(w)$ means the structures satisfy same sentences of quantifier-depth up to k.

Ehrenfeucht-Fraïssé Theorem

Theorem (Ehrenfeucht, Fraïssé, 1954, 1961)

Duplicator wins $G_k((S(v), \overline{s}), (S(w), \overline{t}))$ *iff* $(S(v), \overline{s}) \equiv_k (S(w), \overline{t})$.

Why is this cool?

Because it gives a pumping argument!

Proposition

Language $(aa)^*$ is not FO[<]-definable.

Lemma

Duplicator wins $G_k(a^{2^k}, a^{2^k+1})$.

Proof (of lemma and proposition).

Please see the handwritten notes. □

Proof of the Ehrenfeucht-Fraïssé Theorem

Lemma (How to win an EF-game?)

(1) *Duplicator wins* $G_0((S(v), \overline{s}), (S(w), \overline{t}))$ *iff* $\overline{s} \mapsto \overline{t}$ *is a partial isomorphism.*

(2) *Duplicator wins* $G_{k+1}((S(v), \overline{s}), (S(w), \overline{t}))$ *iff*

(2.a) $\forall s \in D^v : \exists t \in D^w$: *Duplicator wins* $G_k((S(v), s.\overline{s}), (S(w), t.\overline{t}))$ *and*

(2.b) $\forall t \in D^w : \exists s \in D^v$: *Duplicator wins* $G_k((S(v), s.\overline{s}), (S(w), t.\overline{t}))$.

Intuition

$G_k((S(v), s.\overline{s}), (S(w), t.\overline{t}))$ gives arbitrary first step in $G_{k+1}((S(v), \overline{s}), (S(w), \overline{t}))$.

Proof (of Ehrenfeucht-Fraïssé Theorem).

Please see the handwritten notes. □

Star-free Languages

Towards Goal (2)

- Find subclass of REG that characterizes FO[$<$]-definable languages
- Want algebraic characterization (as opposed to logical) that highlights closure properties

Star-free Languages

Definition (Star-free Languages)

The class of star-free languages over alphabet Σ, denoted by SF_Σ, is the smallest class of languages that satisfies

(1) $\emptyset, \{\varepsilon\} \in \mathsf{SF}_\Sigma$ and $\{a\} \in \mathsf{SF}_\Sigma$ for all $a \in \Sigma$ and

(2) if $L_1, L_2 \in \mathsf{SF}_\Sigma$ then also $L_1 \cup L_2, L_1 \cdot L_2, \overline{L_1} \in \mathsf{SF}_\Sigma$.

Remark

- Complement is not an operator on REG, but it can be derived.
- Complement may yield * in alternative representations of the language.

Example

Please see handwritten notes.

From Star-free Languages to FO[$<$]

Goal

Establish SF $=$ FO[$<$]-definable.

Theorem (McNaughton and Papert I, 1971)

Let $L \in \mathrm{SF}_\Sigma$. We can effectively construct a FO[$<$]-formula φ_L so that $L(\varphi_L) = L$.

Proof.

Homework. $\square$

From FO[$<$] to Star-free Languages

Goal $\supseteq$

Establish SF $\supseteq$ FO[$<$]-definable.

Insights

- Relation $\equiv_k$ has finite index, i.e., finitely many classes.
- Every class of $\equiv_k$ can be characterized by single formula.

With this, give inductive construction of SF-representation for FO[$<$]-defined language.

From FO[<] to Star-free Languages

Lemma

Consider structures $(S(s), \bar{s})$ with $|\bar{s}| = n \in \mathbb{N}$. For every $k \in \mathbb{N}$, equivalence $\equiv_k$ has finite index.

Proof.

Please see handwritten notes. $\square$

Lemma

For every equivalence class $[(S(v), \bar{s})]_{\equiv_k}$ there is a formula $\varphi_{[(S(v),\bar{s})]_{\equiv_k}}$ of $qd(\varphi_{[(S(v),\bar{s})]_{\equiv_k}}) \leq k$ so that

$$(S(w), \bar{t}) \in [(S(v), \bar{s})]_{\equiv_k} \quad \textit{iff} \quad S(w), I[\bar{t}/\bar{x}] \models \varphi_{[(S(v),\bar{s})]_{\equiv_k}}.$$

Proof.

Please see handwritten notes. $\square$

McNaughton and Papert's Theorem

Theorem (McNaughton and Papert II, 1971)

Let φ an FO$[<]$ sentence. We can effectively construct $L \in \mathrm{SF}_\Sigma$ so that $L(\varphi) = L$.

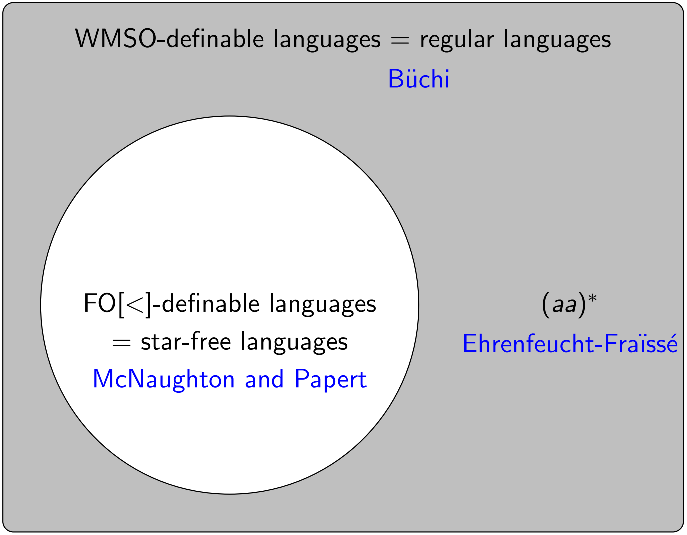
Proof.

Please see handwritten notes. □

Theorem (McNaughton and Papert I+II, 1971)

A language $L \subseteq \Sigma^*$ is star-free iff it is FO$[<]$-definable.

The World of Finite Words ... as we know it now



4. Presburger Arithmetic

Presburger Arithmetic

Goal

- State properties of sets of natural numbers
- Use restricted language of first-order arithmetic: addition, no multiplication, quantification
- Compute solution space (free variables)
- Compute truth value (closed formulas)

Two approaches

- Automata theoretic: Represent solution space via automaton
- Logical: Establish quantifier elimination result

Presburger Arithmetic: Syntax

- Signature $Sig = (Fun, Pred)$ with $Fun = \{0/_0, 1/_0, +/_2\}$ and $Pred = \{< /_2\}$
- Infinite set of first-order variables V

Definition (Syntax of Presburger arithmetic)

Terms built from variables and function symbols:

$$t ::= 0 \mid 1 \mid x \mid t_1 + t_2 \quad \text{with } x \in V.$$

Formulas in Presburger arithmetic defined by

$$\varphi ::= t_1 < t_2 \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \exists x : \varphi.$$

Set of all formulas denoted by PA.

Presburger Arithmetic: Syntax

Definition (Abbreviations)

Abbreviations: Consider terms t_1, t_2, $n \in \mathbb{N}$, and $x \in V$. We set

$$t_1 > t_2 := t_2 < t_1 \qquad t_1 \leq t_2 := \neg(t_1 > t_2)$$

$$t_1 \geq t_2 := t_2 \leq t_1 \qquad t_1 = t_2 := t_1 \leq t_2 \wedge t_1 \geq t_2$$

$$n := \underbrace{1 + \ldots + 1}_{n\text{-times}} \qquad nx := \underbrace{x + \ldots + x}_{n\text{-times}}$$

Abbreviations for formulas: as before.

Definition (Bound and free variables)

Like for WMSO. Sentences have no free variables.

Presburger Arithmetic: Semantics

Fixed structure $(\mathbb{N}, 0^{\mathbb{N}}, 1^{\mathbb{N}}, +^{\mathbb{N}}, <^{\mathbb{N}})$

Definition (Satisfaction relation $\models$ for PA)

Consider formula $\varphi \in \mathrm{PA}$. An interpretation $I : V \nrightarrow \mathbb{N}$ assigns a natural number to each free variable in φ (and maybe to others, not important). With this:

$$\begin{array}{lll} I \models t_1 < t_2 & \text{if} & I(t_1) <^{\mathbb{N}} I(t_2) \\ I \models \neg\varphi & \text{if} & I \not\models \varphi \\ I \models \varphi_1 \vee \varphi_2 & \text{if} & I \models \varphi_1 \text{ or } I \models \varphi_2 \\ I \models \exists x : \varphi & \text{if} & \text{there is } n \in \mathbb{N} \text{ so that } I[n/x] \models \varphi. \end{array}$$

Interpretation of terms (note that $I(x) \in \mathbb{N}$):

$$I(0) := 0^{\mathbb{N}} \qquad I(1) := 1^{\mathbb{N}} \qquad I(t_1 + t_1) := I(t_1) +^{\mathbb{N}} I(t_2).$$

Definition (Equivalence)

Formulas $\varphi, \psi \in \mathrm{PA}$ are equivalent, $\varphi \equiv \psi$, if for all $I : V \nrightarrow \mathbb{N}$ we have

$$I \models \varphi \quad \text{iff} \quad I \models \psi.$$

Presburger Arithmetic: Semantics

Definition (Truth, solutions, definability)

Consider closed formula $\varphi \in \mathrm{PA}$.

- Say φ is true if satisfied by all interpretations.
- Otherwise φ satisfied by no interpretation and call it false.

Consider formula $\psi \in \mathrm{PA}$ with $n \in \mathbb{N}$ free variables $\overline{x}$.

- Restrict ourselves to interpretations $I : V \nrightarrow \mathbb{N}$ with $dom(I) = \overline{x}$.
- Assume variables are ordered, write I as vector $\overline{v} \in \mathbb{N}^n$.
- Call $\overline{v} \in \mathbb{N}^n$ with $\overline{v} \models \psi$ a model or solution of ψ.
- Formula ψ is satisfiable if there is $\overline{v} \in \mathbb{N}^n$ with $\overline{v} \models \psi$.
- If all $\overline{v} \in \mathbb{N}^n$ satisfy ψ, call ψ valid.
- Solution space of ψ is

$$Sol(\psi) := \{\overline{v} \in \mathbb{N}^n \mid \overline{v} \models \psi\}.$$

A set $S \subseteq \mathbb{N}^k$ is Presburger-definable if there is $\psi \in \mathrm{PA}$ with $S = Sol(\psi)$.

Representing Solution Spaces

Goal

Represent $Sol(\psi)$ by a DFA A_ψ.

Problem

A_ψ accepts words whereas $Sol(\psi)$ contains numbers.

Definition (Least-significant bit first encoding, language of a formula)

Relation $lsbf \subseteq \mathbb{N} \times \{0,1\}^*$ encodes $k \in \mathbb{N}$ by the set $lsbf(k) := binary(k) \cdot 0^*$. Binary notation has least-significant bit first. Extend relation to vectors:

$$lsbf \subseteq \mathbb{N}^n \times (\{0,1\}^n)^* \quad \text{with } n \in \mathbb{N}.$$

The language of $\psi \in PA$ is

$$L(\psi) := \bigcup_{\overline{v} \in Sol(\psi)} lsbf(\overline{v}).$$

Representing Solution Spaces

Theorem (Büchi 1960, Wolper & Boigelot 2000, Esparza 2012)

Let $\psi \in PA$. We can effectively construct a DFA A_ψ with $L(A_\psi) = L(\psi)$.

Corollary

It is decidable, whether ψ is satisfiable/valid.

Approach

$$A_{\neg\psi} := \overline{A_\psi} \qquad A_{\varphi \vee \psi} := A_\varphi \cup A_\psi \qquad A_{\exists x : \psi} := \pi_x(A_\psi)$$

Representing Solution Spaces

Remains to construct automaton for solutions of atomic formulas.

Notation

Atomic formulas can be assumed to be in form

$$\psi = a_1x_1 + \ldots + a_nx_n \leq b$$

with $a_1, \ldots, a_n, b \in \mathbb{Z}$. With $\overline{a} \in \mathbb{Z}^n$ and $\overline{x} \in V^n$ vectors, write as

$$\overline{a} \cdot \overline{x} \leq b.$$

For the construction, please see handwritten notes.

Lemma (Termination)

Let $\psi = \overline{a} \cdot \overline{x} \leq b$ and $s = \sum_{i=1}^{n} |a_i|$. The states $j \in \mathbb{Z}$ added to the worklist satisfy

$$-|b| - s \leq j \leq |b| + s.$$

Quantifier Elimination

Goal

Decide truth of a sentence $\varphi \in \mathrm{PA}$.

Approach (Replace quantifiers by concrete values)

A logic admits quantifier elimination if for any formula of the form

$$\forall/\exists x_1 \ldots \forall/\exists x_n : \varphi(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

there is an equivalent formula $\psi(y_1, \ldots, y_m)$.

Definition (Modulo m)

Extend signature of Presburger arithmetic by $\equiv_m$ for all $m \geq 2$.

Remark

Note that $\mathrm{PA}[<]$ and $\mathrm{PA}[<, (\equiv_m)_{m\geq 2}]$ equally expressive:

$$x \equiv_m y \quad \text{iff} \quad \exists z : (x \leq y \wedge y - x = mz) \vee (x > y \wedge x - y = mz).$$

Quantifier Elimination

Theorem (Presburger 1929)

Consider $\exists x : \varphi(x, y_1, \ldots, y_m) \in PA[<, (\equiv_m)_{m\geq 2}]$. *We can effectively construct* $\psi(y_1, \ldots, y_m) \in PA[<, (\equiv_m)_{m\geq 2}]$ *with*

$$\exists x : \varphi(x, y_1, \ldots, y_m) \underset{\textit{logical equivalence}}{\equiv} \psi(y_1, \ldots, y_m).$$

Proof.

Please see handwritten notes. □

Corollary

Given a sentence $\varphi \in PA$*, we can decide whether it is true or false.*

Phrased differently, the theory of structure $(\mathbb{N}, 0^{\mathbb{N}}, 1^{\mathbb{N}}, <^{\mathbb{N}}, +^{\mathbb{N}})$ *is decidable.*

Part B Infinite Words

Where are we?

Learned so far...

- REG/Finite automata, WMSO/FO formulas, Presburger arithmetic/Semilinear sets/Parikh images.
- Now following model checking problem makes sense:

$$A \models \varphi \quad \text{defined by} \quad L(A) \subseteq L(\varphi).$$

  A usually called system, φ usually called specification, check whether A is model of φ (in the sense of $\models$).
- Systems features: regular or regular + counting.

Sometimes, finite words are not sufficient...

- Operating systems typically not meant to terminate: $\Box\Diamond\texttt{req}$
- New class of automata: Büchi automata — system.
- New logic: Linear-time Temporal Logic (LTL) — specification.
- New system features: Büchi pushdown automata – recursion.

5. ω-Regular Languages and Büchi Automata

Goals and Problems

Goal

Recognize infinite words with finite automata

- What is an accepting run? Final states fail!
- Büchi condition: visit final states infinitely often.

Solve algorithmic problems

- Emptiness: Does the automaton accept a word?
- Language equivalence: Do automata A and B accept the same language?

Key challenges

Determinisation/complementation.

Applications

- Model checking MSO — second-order variables range over infinite sets.
- Model checking LTL as syntactic fragment of MSO.

Basic Notions

Let Σ be a finite alphabet.

Definition

- ω-word over Σ = infinite sequence $w = a_0 \cdot a_1 \ldots$ with $a_i \in \Sigma$ for all $i \in \mathbb{N}$.
- Set of all infinite words over Σ is Σ^ω.
- ω-language $L \subseteq \Sigma^\omega$ = set of ω-words.
- Let $w \in \Sigma^\omega$ and $a \in \Sigma$. Then $|w|_a \in \mathbb{N} \cup \{\omega\}$ = number of a in w.

Concatenation

- Impossible to concatenate $v, w \in \Sigma^\omega$
- If $v \in \Sigma^*$ and $w \in \Sigma^\omega$, then $v \cdot w \in \Sigma^\omega$.
- Let $V \subseteq \Sigma^*$ and $W \subseteq \Sigma^\omega$, then $V \cdot W := \{v \cdot w \mid v \in V, w \in W\} \subseteq \Sigma^\omega$.
- Let $v \in \Sigma^+$. Then $v^\omega := v \cdot v \cdot v \cdot \ldots$
- Let $L \subseteq \Sigma^*$ with $L \cap \Sigma^+ \neq \emptyset$. Then

$$L^\omega := \{v_0 \cdot v_1 \cdot v_2 \cdot \ldots \mid v_i \in L \setminus \{\varepsilon\} \text{ for all } i \in \mathbb{N}\}.$$

Basic Notions

Example

Set of all words with

- infinitely many b
- so that two b are separated by even number of a:

$$a^* \cdot ((aa)^* \cdot b)^\omega .$$

Next step

Define ω-regular languages

- Choose ω-iteration of regular languages.
- "Correct definition" as follows: has natural corresponding automaton model.

ω-Regular Languages

Definition (ω-regular languages)

A language $L \subseteq \Sigma^\omega$ is ω-regular if there are regular languages $V_0, \ldots, V_{n-1} \subseteq \Sigma^*$, $W_0, \ldots, W_{n-1} \subseteq \Sigma^*$ with $W_i \cap \Sigma^+ \neq \emptyset$ for all $i \in [0, n-1]$ so that

$$L = \bigcup_{i=0}^{n-1} V_i \cdot W_i^\omega.$$

Example

Please see handwritten notes.

Lemma

ω-regular languages are closed under

- *union*
- *concatenation from left with regular languages.*

For remaining closure properties: automata helpful.

Büchi Automata

- Syntactically finite automata
- Acceptance condition changed

Definition (Büchi automaton (syntax and semantics))

- A non-deterministic Büchi automaton (NBA) is a tuple $A = (\Sigma, Q, q_0, \rightarrow, Q_F)$ with the usual states Q, initial state $q_0 \in Q$, final states $Q_F \subseteq Q$, transition relation $\rightarrow \subseteq Q \times \Sigma \times Q$.
- Run of A is an infinite sequence

$$r = q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \ldots$$

- If $w = a_0 \cdot a_1 \cdot a_2 \cdot \ldots$, we have a run of A on w.
- Write $q_0 \xrightarrow{w}$ to indicate there is a run of A on w. (States not important.)
- Let $Inf(r) :=$ states that occur infinitely often in r.
- Run r is accepting if $Inf(r) \cap Q_F \neq \emptyset$.
- ω-language of A is

$$L(A) := \{w \in \Sigma^\omega \mid \text{there is an accepting run of } A \text{ on } w\}.$$

Büchi Automata

Comment

Acceptance = one final state visited infinitely often

= set of final states visited infinitely often ($\Leftarrow$ as Q_F finite set).

Example

The automata can be found in the handwritten notes. Let $\Sigma = \{a, b\}$.

$L_1 := (a^* \cdot b)^\omega$	Infinitely many b.
$L_2 := (a \cup b)^* \cdot a^\omega$	Finitely many b.

Note that $L_2 = \overline{L_1} = \Sigma^\omega \setminus L_1$.
Automaton A_2 for L_2 is non-deterministic while A_1 for L_1 is deterministic.

Deterministic Büchi Automata

Definition (Deterministic Büchi automaton)

An NBA $A = (\Sigma, Q, q_0, \rightarrow, Q_F)$ is deterministic (DBA) if for all $a \in \Sigma$ and all $q \in Q$ there is precisely one state $q' \in Q$ with $q \xrightarrow{a} q'$.

Not by accident that A_2 is NBA while A_1 is DBA.

- L_2 can not be recognized by a DBA.
- In sharp contrast to NFA = DFA-recognizable languages.

Theorem

There are ω-languages that are NBA-recognizable but not DBA-recognizable.

Consequence

There are NBAs that cannot be determinized into DBAs.

Since $L_2 = (a \cup b)^* \cdot a^\omega$, one may assume that

$$\underbrace{\omega\text{-regular languages}}_{\text{expressions/closure}} = \underbrace{\text{NBA-recognizable languages}}_{\text{automata}}$$

This in fact holds.

6. Linear-time Temporal Logic

Linear-time Temporal Logic

- Specification language for model checking:

  in a model checking problem $A \models \varphi$, formula φ is typically in LTL
- Used in industry as PSL = property specification language (variant of LTL, like statemachines in UML are derived from finite automata)
- Proposed by Amir Pnueli in 1977, Turing award 1996

Linear-time Temporal Logic

Idea of LTL

Subset of MSO useful for specification

No quantifiers, more complex and intuitive operators

- Understand word as a sequence of (sets of) system actions over time
- Interpret formula at a single moment/point in the word

$$\underbrace{\overline{\qquad\qquad\alpha\qquad\qquad}}\ a\ \overline{\qquad\qquad\beta\qquad\qquad}$$

a is now, β is the future, operators only make claims about the future

Remark

- LTL is a linear-time temporal logic that talks about words
- CTL is a branching-time temporal logic that talks about computation trees

$$E\bigcirc(x \wedge A\bigcirc z).$$

- CTL* unifies and generalizes LTL and CTL

Linear-time Temporal Logic

Goal

- Translate LTL into NBA for model checking
- LTL can be understood as a subset of MSO

  Therefore, we know this translation can be done
- But it is strictly less expressive than MSO

  Therefore, we obtain a faster and easier algorithm

LTL: Syntax

Recall

For translation of WMSO formulas $\varphi(X_1, \dots, X_n)$

- Used NFAs over $\{0,1\}^n$, vectors of Booleans

In LTL

There is a finite set of propositions $\mathcal{P}$ (with typical elements $p, q, \dots \in \mathcal{P}$)

- Mimic second-order variables X_i
- Finite in every system

Define alphabet $\Sigma := \mathbb{P}(\mathcal{P})$

- Letters are again vectors:

$$a \in \Sigma \quad \text{means} \quad a \subseteq \mathcal{P} \quad \text{with} \quad a = \begin{array}{c} \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 1 \end{pmatrix} \end{array} \begin{array}{l} p_1 \in a \\ p_2 \notin a \\ \quad\vdots \\ p_n \in a \end{array}$$

- We use set notation: $p \in a$

Why this alphabet? Systems do multiple action at a time/components are in one state each

LTL: Syntax

Definition (Syntax of LTL)

Formulas in LTL over $\Sigma := \mathbb{P}(\mathcal{P})$ are defined by

$$\varphi ::= p \mid \varphi \vee \psi \mid \neg\varphi \mid \underbrace{\bigcirc\varphi}_{\text{next}} \mid \underbrace{\varphi\,\mathcal{U}\,\psi}_{\text{until}} \qquad \text{where } p \in \mathcal{P}$$

Definition (Abbreviations)

Use standard abbreviations for Boolean operators. Moreover:

$$\underbrace{\Diamond\varphi}_{\text{eventually}} := \mathit{true}\,\mathcal{U}\,\varphi \qquad \underbrace{\Box\varphi}_{\text{always}} := \neg\Diamond\neg\varphi \qquad \underbrace{\varphi\,\mathcal{R}\,\psi}_{\text{release}} := \neg(\neg\varphi\,\mathcal{U}\,\neg\psi)$$

Definition (Size)

The size of an LTL formula is defined inductively by

$$|p| := 1 \qquad |\neg\varphi| := 1 + |\varphi| \qquad |\bigcirc\varphi| := 1 + |\varphi|$$

$$|\varphi * \psi| := |\varphi| + 1 + |\psi| \quad \text{with } * \in \{\vee, \wedge, \mathcal{U}, \mathcal{R}\}$$

LTL: Semantics

Intuitive meaning

$p =$	proposition p holds at the current position
$\bigcirc\varphi =$	the next position satisfies φ
$\varphi \,\mathcal{U}\, \psi =$	φ holds in all positions until ψ holds ψ definitely holds some time later (or already now)
$\Diamond\varphi =$	there is some future moment in which φ holds
$\Box\varphi =$	from now on, φ holds in all moments in the future
$\varphi \,\mathcal{R}\, \psi =$	ψ holds as long as it is not released by φ (dual of until) ψ may hold forever or there is a moment with ψ and φ

LTL: Semantics

Definition (Satisfaction relation $\models$ for LTL)

Let $w = a_0 \cdot a_1 \cdot a_2 \ldots \in \Sigma^\omega = \mathbb{P}(\mathcal{P})^\omega$. The satisfaction relation $\models$ is defined inductively as follows (for all $i \in \mathbb{N}$):

$$\begin{array}{lll}
w, i \models p & \text{if} & p \in a_i \\
w, i \models \varphi \vee \psi & \text{if} & w, i \models \varphi \text{ or } w, i \models \psi \\
w, i \models \neg\varphi & \text{if} & w, i \not\models \varphi \\
w, i \models \bigcirc\varphi & \text{if} & w, i+1 \models \varphi \\
w, i \models \varphi\, \mathcal{U}\, \psi & \text{if} & \text{there is } k \geq i \text{ so that} \\
 & & \quad \text{for all } i \leq j < k \text{ we have } w, j \models \varphi \\
 & & \quad \text{and } w, k \models \psi.
\end{array}$$

An LTL formula φ defines a language $L(\varphi) \subseteq \Sigma^\omega$ by interpreting it in the first position of a word:

$$L(\varphi) := \{w \in \Sigma^\omega \mid w, 0 \models \varphi\}.$$

LTL: Semantics

Example

- Infinitely often φ: $\Box\Diamond\varphi$
- Finitely often φ: $\Diamond\Box\neg\varphi$
- Every request is followed by an acknowledge: $\Box(\mathsf{req} \rightarrow \Diamond\mathsf{ack})$
- If there are infinitely many positions with p, then there are infinitely many positions with q:

$$\Box\Diamond p \rightarrow \Box\Diamond q \quad \text{or equivalently} \quad \Box\Diamond q \vee \Diamond\Box\neg p.$$

Definition (Equivalence)

Two LTL formulas φ, ψ are called equivalent, denoted by $\varphi \equiv \psi$, if for all $w \in \Sigma^\omega$ and all $i \in \mathbb{N}$ we have

$$w, i \models \varphi \quad \text{iff} \quad w, i \models \psi.$$

Language-theoretic Considerations

Every letter $a \in \Sigma = \mathbb{P}(\mathcal{P})$ can be described by its characteristic formula

$$\chi_a := \bigwedge_{p \in a} p \wedge \bigwedge_{p \notin a} \neg p.$$

With this, capture languages over Σ by LTL formulas

Example

Language $(a \cdot b)^\omega$ defined by

$$\chi_a \wedge \Box((\chi_a \rightarrow \bigcirc \chi_b) \wedge (\chi_b \rightarrow \bigcirc \chi_a))$$

Language $\underbrace{(a \cdot (a \cup b))^\omega}_{\text{even positions have an } a}$ not LTL-definable

- LTL-definable languages are definable in FO on infinite words
- Words of even length are not definable in FO on finite words
- Similar argument applies here

Positive Normal Form and Properties of Until

Definition (Positive normal form)

An LTL formula over $\Sigma = \mathbb{P}(\mathcal{P})$ is in positive normal form if it is constructed from

$$p, \neg p \text{ with } p \in \mathcal{P} \quad \text{and} \quad \vee, \wedge, \bigcirc, \mathcal{U}, \mathcal{R}.$$

Lemma

For every formula φ there is ψ in positive normal form with $\varphi \equiv \psi$ and $|\psi| \leq 2|\varphi|$.

Proof.

Use the following equivalences:

$$\begin{aligned}
\neg \bigcirc \varphi &\equiv \bigcirc \neg \varphi \\
\neg(\varphi\, \mathcal{U} \psi) &\equiv \neg(\neg(\neg\varphi)\, \mathcal{U} \neg(\neg\psi)) \equiv \neg\varphi\, \mathcal{R}\, \neg\psi \\
\neg(\varphi\, \mathcal{R}\, \psi) &\equiv \neg\neg(\neg\varphi\, \mathcal{U}\, \neg\psi) \equiv \neg\varphi\, \mathcal{U}\, \neg\psi
\end{aligned}$$

□

Positive Normal Form and Properties of Until

For translation of LTL into Büchi automata, use unrolling of until

Lemma (Inductive property of until)

For all $\varphi, \psi \in LTL$ we have $\varphi \,\mathcal{U}\, \psi \equiv \psi \vee (\varphi \wedge \bigcirc(\varphi \,\mathcal{U}\, \psi))$.

Logical equivalence $\equiv$ in LTL in fact a congruence

Lemma

If $\varphi \equiv \psi$ and φ is part of a larger formula $\theta(\varphi)$, then $\theta(\varphi) \equiv \theta(\psi)$.

As a consequence

$$\begin{aligned}\varphi \,\mathcal{U}\, \psi &\equiv \psi \vee (\varphi \wedge \bigcirc(\varphi \,\mathcal{U}\, \psi))\\ &\equiv \psi \vee (\varphi \wedge \bigcirc(\psi \vee (\varphi \wedge \bigcirc(\varphi \,\mathcal{U}\, \psi))))\\ &\equiv \dots\end{aligned}$$

Gives a means to check $\varphi \,\mathcal{U}\, \psi$ at position i:

either ψ holds or φ holds and $\varphi \,\mathcal{U}\, \psi$ holds in the next position $i+1$

Have to ensure ψ eventually holds (unrolling happens finitely many times)

Final states forbid infinite unrollings

From LTL to NBA

Goal

Translate LTL into NBA

- without using intermediary FO representation
- and then Büchi's result

Why is LTL easier than MSO?

- Like the automaton, LTL only looks into the future
- Construction does not follow the inductive structure of formulas (safes complementation at each negation)
- Instead, keep track of satisfaction of all subformulas while reading input

Generalized Büchi Automata

Definition (Generalized NBA)

A generalized non-deterministic Büchi automaton (GNBA) is a tuple $A = (\Sigma, Q, Q_I, \rightarrow, (Q_F^i)_{1 \leq i \leq k})$ with

- set of initial states $Q_I \subseteq Q$ (instead of $q_0 \in Q$)
- family of final states $(Q_F^i)_{1 \leq i \leq k}$ with $Q_F^i \subseteq Q$ for all $1 \leq i \leq k$

A run is still

$$r = q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} \ldots \quad \text{with } q_0 \in Q_I$$

A run is accepting if $\mathit{Inf}(r) \cap Q_F^i \neq \emptyset$ for all $1 \leq i \leq k$

Every set of final states is visited infinitely often

Generalized Büchi Automata

"Generalization" does not increase expressiveness of the automaton model

Lemma

For every GNBA A there is an NBA A' with $L(A) = L(A')$ and $|Q'| \leq k|Q| + 1$.

Idea

Use counters from intersection construction:

$$L(A) = \bigcap_{1 \leq i \leq k} L(A_i) \quad \text{with} \quad A_i = (\Sigma, Q_I, \rightarrow, Q_F^i).$$

Direct construction

- Several initial states into one $\rightsquigarrow$ pic new state
- Several sets of final states to one:
  - Use counters in new states: $Q' := Q \times \{1, \ldots, k\}$
  - (q, i) means: next final state is expected from Q_F^i
  - New final states: $Q_F^i \times \{i\}$ for some $1 \leq i \leq k$ (any i will do)

Fisher-Ladner Closure and Hintikka Sets

Idea of the translation

- States in the automaton are subformulas of $\theta \in \mathrm{LTL}$
- Intuitively, we take the formulas that currently hold

Definition (Fisher-Ladner Closure)

Let $\theta \in \mathrm{LTL}$ be a formula in positive normal form. Its Fisher-Ladner closure $FL(\theta) \subseteq \mathrm{LTL}$ is the smallest set of LTL formulas in positive normal form so that

1 $\theta \in FL(\theta)$ and

2.a if $\varphi * \psi \in FL(\theta)$ then $\{\varphi, \psi\} \subseteq FL(\theta)$ for $* \in \{\wedge, \vee\}$

2.b if $\varphi \,\mathcal{U}\, \psi \in FL(\theta)$ then $\psi \vee (\varphi \wedge \bigcirc(\varphi \,\mathcal{U}\, \psi)) \in FL(\theta)$

2.c if $\varphi \,\mathcal{R}\, \psi \in FL(\theta)$ then $\psi \wedge (\varphi \vee \bigcirc(\varphi \,\mathcal{R}\, \psi)) \in FL(\theta)$

2.d if $\bigcirc\varphi \in FL(\theta)$ then $\varphi \in FL(\theta)$

Fisher-Ladner Closure and Hintikka Sets

- Fisher-Ladner closure defined purely syntactically
- Hintikka sets are sets of subformulas $M \subseteq FL(\theta)$ that are closed under *satisfaction of subformulas (what else has to hold)*

$$\text{if } \varphi \vee \psi \in M \text{ then } \varphi \in M \text{ or } \psi \in M$$

- Single out those sets that do not contain contradictions p and $\neg p$

Fisher-Ladner Closure and Hintikka Sets

Definition (Hintikka set)

Let $\theta \in \text{LTL}$ be a formula in positive normal form. A Hintikka set for θ is a subset $M \subseteq FL(\theta)$ that satisfies the following closure properties:

$$\begin{array}{rcl}
\varphi \vee \psi \in M & \text{implies} & \varphi \in M \text{ or } \psi \in M \\
\varphi \wedge \psi \in M & \text{implies} & \varphi \in M \text{ and } \psi \in M \\
\varphi\, \mathcal{U}\, \psi \in M & \text{implies} & \psi \in M \text{ or } (\varphi \in M \text{ and } \bigcirc(\varphi\, \mathcal{U}\, \psi) \in M) \\
\varphi\, \mathcal{R}\, \psi \in M & \text{implies} & \psi \in M \text{ and } (\varphi \in M \text{ or } \bigcirc(\varphi\, \mathcal{R}\, \psi) \in M)
\end{array}$$

A Hintikka set $M \subseteq FL(\theta)$ is consistent if there is no $p \in \mathcal{P}$ with $\{p, \neg p\} \subseteq M$. By $\mathcal{H}(\theta)$ we denote the set of all consistent Hintikka sets for θ.

The set of propositions that occur positively/negatively in $M \subseteq FL(\theta)$ is

$$\mathcal{P}^+(M) := M \cap \mathcal{P} \qquad\qquad \mathcal{P}^-(M) := \{p \in \mathcal{P} \mid \neg p \in M\}$$

Vardi-Wolper Construction

Construct an automaton $\mathcal{A}_\theta$ that accepts precisely the models of θ

States $=$ consistent Hintikka sets

- What are the subformulas that hold at this position in the model
- Guess them in every step
- Need consistency
  - Within Hintikka sets: automaton does not guess inconsistencies
  - With $\bigcirc$: if $\bigcirc\varphi$ is guessed then φ has to hold at the next position

Final states

- Construction relies on unrolling of $\mathcal{U}$ and $\mathcal{R}$
  - This is already part of $FL(\theta)$ and Hintikka sets
- Until $\mathcal{U}$ yields accepting states
  - Forbids infinite unrollings (have a set of final states for each $\varphi\,\mathcal{U}\,\psi \in FL(\theta)$)

Vardi-Wolper Construction

Definition (Vardi-Wolper automaton)

Consider an LTL formula θ in positive normal form. Let $\varphi_1 \mathcal{U} \psi_1, \ldots, \varphi_k \mathcal{U} \psi_k$ be all $\mathcal{U}$-formulas in $FL(\theta)$. The Vardi-Wolper automaton is

$$\mathcal{A}_\theta := (\mathcal{H}(\theta), Q_I, \rightarrow, (Q_F^i)_{1 \leq i \leq k})$$

with

$$Q_I := \{M \in \mathcal{H}(\theta) \mid \theta \in M\}$$

//Sets that contain θ

$$Q_F^i := \{M \in \mathcal{H}(\theta) \mid \varphi_i \mathcal{U} \psi_i \notin M \text{ or } \psi_i \in M\}$$

//If the ith until formula needs to be fulfilled then this happens in M

$$M \xrightarrow{a} M' \quad \text{if} \quad \{\psi \in FL(\theta) \mid \bigcirc\psi \in M\} \subseteq M' \quad \text{and} \quad \mathcal{P}^+(M) \subseteq a \quad \text{and} \quad \mathcal{P}^-(M) \cap a = \emptyset$$

If $FL(\theta)$ does not contain until formulas, select $Q_F = Q$ as final states.

Vardi-Wolper Construction

Guess Hintikka set M_0 that contains θ

- This selects subformulas that also hold at position 0

If the automaton arrives at M, then M contains (potentially negated) propositions p, $\neg p$, and formulas $\bigcirc\psi$

- These formulas do not have further decompositions
- Make claims about what has to hold at this position ($\bigcirc\psi$ makes claims about next position)

If the automaton takes a transition

- it only uses a letter that is consistent with the current propositions: all positive propositions occur, none of the negative propositions is used
- it reaches a state that is consistent with the guesses of $\bigcirc$ in the previous set (if $\bigcirc\psi \in M$ then $\psi \in M'$)

Vardi-Wolper Construction

Theorem (Vardi, Wolper 1986)

Consider $\theta \in LTL$. The automaton $\mathcal{A}_\theta$ satisfies $L(\theta) = L(\mathcal{A}_\theta)$ and $|\mathcal{A}_\theta| \leq 2^{8|\theta|}$.

7. Model Checking Pushdown Systems (Recursive Programs)

Model Checking Pushdown Systems

Goal

Decide $P \models \varphi$ for P a pushdown system

Technically: Reachability of accepting loops

Key element in the algorithm

Given a set of configurations C, compute the set of all predecessors:

$$pre^*(C) := \bigcup_{i \in \mathbb{N}} X_i \qquad \text{with} \qquad X_0 := C \qquad X_{i+1} := X_i \cup pre(X_i) \quad \text{for all } i \in \mathbb{N}.$$

Here, $pre(C)$ = immediate predecessors of C.

Problem

For finite state systems, sequence $(X_i)_{i \in \mathbb{N}}$ reaches a fixed point.

For infinite state systems like PDS, sequence $(X_i)_{i \in \mathbb{N}}$ usually does not converge.

Model Checking Pushdown Systems

Solution: Representation structures

Finite structures that represent infinite sets of configurations.

Should have good properties — a wish list:

- Closed under $\cup$, or even all Boolean operations
- Closed under *pre*
- Decidable membership problem ($c \in R$ for c configuration, R representation)

Note that $\cup$ and *pre* are needed for $X_{i+1} := X_i \cup pre(X_i)$.

Example

- Timed automata $\rightarrow$ sets of configurations represented by regions.
- Well-structured transition systems $\rightarrow$ sets of configurations represented by minimal elements.
- Lossy channel systems $\rightarrow$ sets of configurations represented by simple regular expressions.

Model Checking Pushdown Systems

Here: configurations are pairs (q, w) of state q and stack content w.

Representation structure: P-NFA

P-NFA A accepts configuration (q, w) of pushdown system P if A accepts w from the initial state s_q.

Warning

A represents the set of configurations of P.
A does not represent the behaviour/transitions of P.

Contribution

- NFAs are closed under Boolean operations
- Membership is decidable

$\Rightarrow$ Algorithm to compute $pre^*(C)$

$\Rightarrow$ Exploit it for model checking PDS against LTL

Pushdown Systems: Syntax

Idea

- Pushdown systems are pushdown automata (Kellerautomaten).
- But do not consider them as language acceptors.
- Interested in their configurations and configuration changes.

Definition (Syntax of Pushdown Systems)

A pushdown system (PDS) is a triple $P = (Q, \Gamma, \rightarrow)$ with

- set of states Q
- stack alphabet Γ
- set of transitions $\rightarrow \subseteq (Q \times \Gamma) \times (Q \times \Gamma^*)$

Usually write $q \xrightarrow{\gamma / w} q'$ instead of $((q, \gamma), (q', w)) \in \rightarrow$.

Pushdown Systems: Semantics

Definition (Semantics of Pushdown Systems)

Let $P = (Q, \Gamma, \to)$ be a PDS. Its behaviour is defined in terms of

- configurations (q, w) with state $q \in Q$ and stack content $w \in \Gamma^*$.
- Denote the set of all configurations by $CF := Q \times \Gamma^*$.

The PDS induces the following transitions relation $\to \subseteq CF \times CF$ between configurations:

$$(q_1, \gamma \cdot w') \to (q_2, w \cdot w') \quad \text{if} \quad q_1 \xrightarrow{\gamma/w} q_2 \quad \text{in } P.$$

Pushdown Systems: Semantics

The predecessor function abstracts from transitions and talks about sets of configurations.

Definition (Predecessors)

Let $C \subseteq CF$ be a set of configurations in a PDS. The set of immediate predecessors of C is

$$pre(C) := \{c' \in CF \mid c' \rightarrow c \text{ with } c \in C\}.$$

The set of all predecessors of C (all configurations from which C is reachable) is

$$pre^*(C) := \{c' \in CF \mid c' \rightarrow^* c \text{ with } c \in C\}.$$

Here, $\rightarrow^*$ is the reflexive and transitive closure of $\rightarrow$. We also use

$$pre^+(C) := pre(pre^*(C)).$$

Representation Structure: P-NFA

How to represent a set of configurations?

Definition (P-NFA)

Let $P = (Q, \Gamma, \rightarrow)$ be a PDS. A P-NFA is an NFA $A = (\Gamma, S, S_I, \rightarrow, S_F)$ where

$$S_I := \{s_q \mid q \in Q\}.$$

A accepts configuration (q, w) if $s_q \xrightarrow{w} s_F$ with $s_F \in S_F$.

The set of all configurations accepted by A is $CF(A)$.

A set of configurations $C \subseteq CF$ is regular if $C = CF(A)$ for some P-NFA A.

Computing $pre^*(C)$

Goal:

Given: Pushdown system $P = (Q, \Gamma, \rightarrow)$ and a regular set of configurations $C = CF(A)$ for some P-NFA A.

Compute: Another P-NFA A_{pre^*} that represents $pre^*(CF(A))$.

Approach

Compute $pre^*(C) = \bigcup_{i \in \mathbb{N}} X_i$ with

$$X_0 := C \quad \text{and} \quad X_{i+1} := X_i \cup pre(X_i) \quad \text{for all } i \in \mathbb{N}.$$

So we intend to construct the sequence

$$X_0 \subseteq X_1 \subseteq X_2 \subseteq \dots \quad \text{until} \quad X_{i+1} = X_i \quad \text{for some } i \in \mathbb{N}.$$

Then $pre^*(C) = X_i$.

Computing $pre^*(C)$

Problem

Existence of such a fixed point is not guaranteed. As an example, consider $P = (\{q\}, \{\gamma\}, \{q \xrightarrow{\gamma/\varepsilon} q\})$. Then for $C = \{(q, \varepsilon)\}$ we have

$$X_i = \{(q,\varepsilon), \ldots, (q, \gamma^i)\} \quad \text{for all } i \in \mathbb{N}.$$

Hence, $X_{i+1} \neq X_i$ for all $i \in \mathbb{N}$.

Solution

Compute $pre^*(C)$ as the limit of a different sequence of sets of configurations:

$$Y_0 \ \subseteq \ Y_1 \ \subseteq \ Y_2 \ \subseteq \ \ldots$$

This sequence will satisfy three conditions:

(Term) There is $i \in \mathbb{N}$ so that $Y_i = Y_{i+1}$.

(Compl) $X_i \subseteq Y_i$ for all $i \in \mathbb{N}$.

(Sound) $Y_i \subseteq \bigcup_{j \in \mathbb{N}} X_j$ for all $i \in \mathbb{N}$.

Computing $pre^*(C)$

Construction

Y_i = set of configurations accepted by a P-NFA A_i:

$$Y_0 = CF(A_0) \quad \subseteq \quad Y_1 = CF(A_1) \quad \subseteq \quad Y_2 = CF(A_2) \quad \subseteq \quad \ldots$$

From A_i to A_{i+1}: only add transitions, never change the states.
This already shows (Term), at most $|S|^2|\Gamma|$ transitions can be added.

Definition (Sequence $(A_i)_{i\in\mathbb{N}}$ and A_{pre^*})

Let $P = (Q, \Gamma, \rightarrow)$ be a PDS and $A = (\Gamma, S, S_I, \rightarrow_A, S_F)$ be a P-NFA.
We define $A_0 := A$. Morever, let $A_i = (\Gamma, S, S_I, \rightarrow_i, S_F)$. Then we set

$$A_{i+1} := (\Gamma, S, S_I, \rightarrow_i \cup \rightarrow_{new}, S_F) \quad \text{where}$$

$$s_{q_1} \xrightarrow{\gamma}_{new} s \quad \text{if} \quad s_{q_2} \xrightarrow{w}_i s \text{ and } q_1 \xrightarrow{\gamma/w} q_2 \text{ in } P.$$

Define $A_{pre^*} := A_i$ with $A_i = A_{i+1}$.

Computing $pre^*(C)$

Intuition and Remark

1. Configuration $(q_1, \gamma \cdot w')$ is an immediate predecessor of $(q_2, w \cdot w')$ wrt. transition $q_1 \xrightarrow{\gamma/w} q_2$. So if $w \cdot w'$ is accepted from s_{q_2},

$$s_{q_2} \xrightarrow{w}_i s \xrightarrow{w'}_i s_F \in S_F,$$

then the new transition accepts $\gamma \cdot w'$ from s_{q_1}:

$$s_{q_1} \xrightarrow{\gamma}_{new} s \xrightarrow{w'}_i s_F \in S_F.$$

2. There are two strategies for adding transitions:

lazy	Only add a transition if it leads to a final state.
eager	Always add a transition, as defined.

Computing $pre^*(C)$

Theorem (Bouajjani, Esparza, Maler '97)

Consider a PDS P and a set of configurations accepted by a P-NFA A. We can construct (in polynomial time) a P-NFA A_{pre^*} so that

$$CF(A_{pre^*}) = pre^*(CF(A)).$$

Warning

For the predecessor computation to be correct, we have to assume that A has no edges leading to an initial state. This can always be achieved by preprocessing A.

Proof.

Assume we already proved (Compl) and (Sound) for sequence $(Y_i)_{i\in\mathbb{N}}$. Then

$$\supseteq \qquad pre^*(CF(A)) = \bigcup_{i\in\mathbb{N}} X_i \overset{\text{(Compl)}}{\subseteq} \bigcup_{i\in\mathbb{N}} Y_i = Y_k \quad \text{for } A_k = A_{k+1} = A_{pre^*}$$

$$\subseteq \qquad CF(A_{pre^*}) = Y_k \underset{\text{(Sound)}}{\subseteq} \bigcup_{i\in\mathbb{N}} X_i = pre^*(CF(A)) \quad \text{for some } k \in \mathbb{N}.$$

Computing $pre^*(C)$

Lemma (Completeness)

$X_i \subseteq Y_i$ for all $i \in \mathbb{N}$.

The soundness proof needs a technical lemma, which relies on the warning from the previous slide.

Lemma

If $s_q \xrightarrow{w}_i s$ then $(q, w) \rightarrow^* (q', v)$ for some $q' \in Q$, $v \in \Gamma^*$ so that $s_{q'} \xrightarrow{v}_0 s$.

Intuition

If (q, w) is accepted in the ith iteration, then it leads to a configuration (q', v) that is accepted initially.

Lemma (Soundness)

$Y_i \subseteq pre^*(C)$ for all $i \in \mathbb{N}$.

Model Checking LTL

To define $P \models \varphi$ with $P = (Q, \Gamma, \rightarrow)$ and $\varphi \in \text{LTL}$, assign propositions to states:

$$\lambda : Q \rightarrow \mathbb{P}(\mathcal{P}) \quad \text{with } \mathcal{P} \text{ a finite set of propositions.}$$

Goal: (Global) model checking

- Global model checking: Compute the set $C \subseteq CF$ of all configurations $c \in C$ so that every run starting from c satisfies φ.
- Classical model checking: Does every run starting from c_{init} satisfy φ?
- Model checking can be solved with global model checking: is $c_{init} \in C$?

From global model checking to accepting runs

To solve global model checking, construct the Büchi pushdown system $P \times A_{\neg\varphi}$.

Look for an accepting run in $P \times A_{\neg\varphi}$.

Model Checking LTL

Definition (Büchi Pushdown System)

A Büchi pushdown system (BPDS) is a tuple $BP = (Q, \Gamma, \rightarrow, Q_F)$ with

- $(Q, \Gamma, \rightarrow)$ a PDS and
- $Q_F \subseteq Q$ a set of final states.

The semantics is defined in terms of infinite runs

$$r = (q_0, w_0) \rightarrow (q_1, w_1) \rightarrow \dots$$

A run is accepting if $q_i \in Q_F$ for infinitely many configurations (q_i, w_i).

Accepting run problem

Given a BPDS BP, compute the set $C \subseteq CF$ of all configurations $c \in C$ so that BP has an accepting run from c.

Model Checking LTL

Following proposition relates the accepting run problem to reachability in PDS.

Proposition

BP has an accepting run from $c \in CF$ if and only if
there are configurations $(q, \gamma), (q_F, u), (q, \gamma \cdot v) \in CF$ with $q_F \in Q_F$ so that

(1) $c \rightarrow^* (q, \gamma \cdot w)$ for some $w \in \Gamma^*$ and

(2) $(q, \gamma) \rightarrow^+ (q_F, u) \rightarrow^* (q, \gamma \cdot v)$.

To check existence of an accepting run, reformulate conditions:

(1') $c \in pre^*(\{q\} \times \gamma \cdot \Gamma^*)$

(2') $(q, \gamma) \in pre^+((Q_F \times \Gamma^*) \cap pre^*(\{q\} \times \gamma \cdot \Gamma^*))$.

Note the beauty!

Statement about emptiness (set-theoretic) turned into an algorithmic problem via combinatorial reasoning.

Model Checking LTL

Theorem (Bouajjani, Esparza, Maler '97)

The accepting run problem of BPDS can be solved in polynomial time.

Algorithm

- Find all configurations (q, γ) for which (2') holds (at most $|Q||\Gamma|$ many):
  - Construct BP-NFA for $pre^*(\{q\} \times \gamma \cdot \Gamma^*)$.
  - Intersect with $Q_F \times \Gamma^*$: keep stack contents from s_{q_F} with $q_F \in Q_F$.
  - Compute $pre^*((Q_F \times \Gamma^*) \cap pre^*(\{q\} \times \gamma \cdot \Gamma^*))$
  - Compute another single *pre*:

    $$pre(pre^*((Q_F \times \Gamma^*) \cap pre^*(\{q\} \times \gamma \cdot \Gamma^*))) = pre^+((Q_F \times \Gamma^*) \cap pre^*(\{q\} \times \gamma \cdot \Gamma^*)).$$

  - Check $(q, \gamma) \in pre^+((Q_F \times \Gamma^*) \cap pre^*(\{q\} \times \gamma \cdot \Gamma^*))$.
- For all (q, γ) that satisfy (2'), compute $pre^*(\{q\} \times \gamma \cdot \Gamma^*)$.
- Take the union of all these sets $pre^*(\{q\} \times \gamma \cdot \Gamma^*)$.

8. More on Infinite Words

More on Infinite Words: MSO

- Syntax of WSMO.
- But interpreted over infinite words. In particular, second-order quantifiers may range over infinite sets:

$$\exists X : (\exists x : \mathit{first}(x) \wedge X(x)) \wedge (\forall x : X(x) \rightarrow \exists y : x < y \wedge X(y))$$

  is satisfiable in MSO.

Main Result

Satisfiability is decidable in MSO.

Proof.

- Construct NBA A_φ so that

$$L(A_\varphi) = \{w \in \Sigma^\omega \mid S(w) \models \varphi\} = L(\varphi).$$

  Reuse techniques for WMSO.
- Check emptiness for $L(A_\varphi)$.

□

More on Infinite Words: Rabin, Streett, and Muller

Büchi automata

Infinite run $r = q_0 \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \dots$ satisfies $\mathit{Inf}(r) \cap Q_F \neq \emptyset$.

Rabin automata

Final states are pairs $\mathcal{F} = \{(G_1, F_1), \dots, (G_n, F_n)\}$ with $G_1, \dots, F_n \subseteq Q$.

Run is accepting if

$$\underbrace{\mathit{Inf}(r) \cap G_i \neq \emptyset}_{\text{States that should occur in the infinite}} \quad \text{and} \quad \underbrace{\mathit{Inf}(r) \cap F_i = \emptyset}_{\text{States forbidden in the infinite}} \quad \text{for some } 1 \leq i \leq n.$$

Streett automata

Dual of Rabin acceptance. Final states again $\mathcal{F} = \{(G_1, F_1), \dots, (G_n, F_n)\}$.

Run is accepting if

$$\mathit{Inf}(r) \cap G_i \neq \emptyset \quad \text{implies} \quad \mathit{Inf}(r) \cap F_i \neq \emptyset \quad \text{for all } 1 \leq i \leq n.$$

More on Infinite Words: Rabin, Streett, and Muller

Muller automata

Final states are sets $\mathcal{F} = \{Q_1, \dots, Q_n\}$ with $Q_1, \dots, Q_n \subseteq Q$.

Run is accepting if $Inf(r) \in \mathcal{F}$.

Büchi acceptance is a special case of Rabin, Streett, and Muller acceptance.

Main Result

All models define the same ω-languages:

$$\begin{aligned}\omega - \text{regular languages} &= \text{NBA-acceptable languages}\\ &= \text{Rabin-acceptable languages}\\ &= \text{Streett-acceptable languages}\\ &= \text{Muller-acceptable languages.}\end{aligned}$$

More on Infinite Words: Safra

Goal

Direct determinisation of NBA.

But not every NBA can be determinised.

Solution

Determinise NBA into Rabin/Muller automaton.

Idea: Apply a refined powerset construction to NBA.

States are trees with complex labelling.

In the lecture, we used Safraless, algebraic approach.

More on Infinite Words: Algorithms

- Check emptiness of NBA when they are given as composition

$$A_1 \times \ldots \times A_n.$$

- Check emptiness of Rabin, Streett, and Muller automata.

Part C Finite Trees

Goals and Problems

Words = structures with one successor predicate $suc(x, y)$
Trees = structures with several successors, say $suc_L(x, y)$ and $suc_R(x, y)$.

Trees in Computer Science

- Parse trees of programs
- Abstract data types
- XML document processing

Here: Automata on Trees

finite word-languages = sets of finite words
ω-languages = sets of infinite words
Now: tree-languages = sets of finite trees

Application: Validity of XML documents.

Underlying problem

What are the sets of trees recognized by a finite tree automaton.

9. Bottom-Up and Top-Down Tree Automata

Finite Trees

Definition (Finite tree)

A finite tree is a finite subset $T \subseteq \mathbb{N}^*$ satisfying the following closure properties:

(1) If $w.n \in T$ then $w \in T$.

(2) For $n > 0$ and $w.n \in T$ we have $w.(n-1) \in T$.

Condition (1): If a node is part of a tree, so is its father.

Condition (2): Children are labeled consecutively.

Definition (Ranked alphabet)

A ranked alphabet is a pair (Σ, rk) consisting of a finite set Σ and a rank function

$$rk : \Sigma \to \mathbb{N}.$$

Call $rk(a)$ the rank of letter $a \in \Sigma$.
Denote the letters of rank $n \in \mathbb{N}$ by $\Sigma_n := \{a \in \Sigma \mid rk(a) = n\}$.

Intuitively:

- a node with letter a expects $rk(a)$ children,
- similar to arities of function and predicate symbols.

Finite Trees

Definition (Σ-trees)

Let (Σ, rk) be a ranked alphabet. A Σ-tree is a function

$$t : T \to \Sigma$$

where T is a finite tree as defined above and additionally the following holds:
For all $w \in T$ with $t(w) = a \in \Sigma$, we have

$$w.i \in T \quad \text{iff} \quad i < rk(a) \quad \text{for all } i \in \mathbb{N}.$$

Use $\mathcal{T}_\Sigma$ to denote the set of all Σ-trees.

Condition states that if w is labeled by $a \in \Sigma$ then it has precisely $rk(a)$ children.

Note on Σ-trees

- There are no two nodes with same label but different number of children.
- The alphabet gives an upper bound on the number of children in a tree.

Excursion: Yield of a Tree

Idea: Read the word that consists of the leaf letters left first.

Definition (Yield)

Let $t : T \to \Sigma$ be a tree. Its yield is defined inductively:

(1) If $T = \{\varepsilon\}$ then $yield(t) := t(\varepsilon)$.

(2) Let $T = \{\varepsilon\} \cup 0.T_0 \cup \ldots \cup n.T_n$. Define subtrees

$$t_i : T_i \to \Sigma \quad \text{by} \quad t_i(w) := t(i.w) \quad \text{for all } 0 \leq i \leq n.$$

With this:

$$yield(t) := yield(t_0) \cdot \ldots \cdot yield(t_n).$$

Bottom-Up Tree Automata: Syntax

Two automaton models for trees

Finite automata read words from left to right.

But theory would not change if we read words from right to left.

Trees look different when read from top to bottom vs. bottom-up:

- From top to bottom, we distribute information from one node to many.
- Bottom-up we aggregate information from children.

Gives different theories.

Definition (Bottom-up tree automaton: syntax)

A bottom-up tree automaton (BUTA) is a tuple $A = ((\Sigma, rk), Q, \to, Q_F)$ with

- finite set of states Q, final states $Q_F \subseteq Q$, and
- transition relation $\to = (\to_a)_{a \in \Sigma}$ with

$$\to_a \subseteq Q^n \times Q \quad \text{where} \quad n = rk(a).$$

Bottom-Up Tree Automata: Semantics

A run of a BUTA labels nodes of a tree by states:

- starting from the leafs, stopping at the root (bottom-up)
- transitions read states at the roots of the subtrees.

No initial state:

- Encoded into the transition relation for $a \in \Sigma$ with $rk(a) = 0$.
- Take $\rightarrow_a \subseteq Q^0 \times Q$ as $\rightarrow_a \subseteq Q$.
- This means the initial state is chosen according to the leaf letter.
- Slight difference when compared to finite automata (but can always extend finite automaton by one state to achieve this effect).

Bottom-Up Tree Automata: Semantics

Definition ((Accepting) run, tree language)

A run of a BUTA $A = ((\Sigma, rk), Q, \rightarrow, Q_F)$ on a Σ-tree $t : T \rightarrow \Sigma$ is a function

$$r : T \rightarrow Q$$

so that for all $w \in T$ we have

$$(q_0, \ldots, q_{n-1}) \rightarrow_a q$$

where $a = t(w)$, $n = rk(a)$, $q = r(w)$, and $q_i = r(w.i)$ for all $i \in [0, n-1]$.
A run is accepting if $r(\varepsilon) \in Q_F$.
The (tree) language of A is

$$L(A) := \{t \in \mathcal{T}_\Sigma \mid A \text{ has an accepting run on } t\}.$$

A tree language $L \subseteq \mathcal{T}_\Sigma$ is called regular if there is a BUTA A with $L = L(A)$.

Determinism and Complementation

Definition (Deterministic BUTA)

A BUTA $A = (\Sigma, Q, \to, Q_F)$ is called deterministic (DBUTA) if for all $a \in \Sigma$ and all $(q_0, \ldots, q_{n-1}) \in Q^n$ with $n = rk(a)$ there is precisely one $q \in Q$ so that

$$(q_0, \ldots, q_{n-1}) \ \to_a \ q.$$

Are deterministic BUTA as expressive as non-deterministic BUTA?

Yes, apply the powerset construction.

Determinism and Complementation

Theorem (Rabin & Scott on tree automata)

A tree language is accepted by a BUTA iff it is accepted by a DBUTA.

Proof.

Consider $L(A)$ with $A = (\Sigma, Q^A, \rightarrow^A, Q_F^A)$ a BUTA.
Define the DBUTA $A' := (\Sigma, \mathbb{P}(Q^A), \rightarrow, Q_F)$ where

$$Q_F := \{Q \subseteq Q^A \mid Q \cap Q_F^A \neq \emptyset\}$$

and for every $a \in \Sigma$ with $rk(a) = n$ we have

$$(Q_0, \ldots, Q_{n-1}) \rightarrow_a Q$$

where $Q := \{q \in Q^A \mid \exists q_0 \in Q_0, \ldots, q_{n-1} \in Q_{n-1} : (q_0, \ldots, q_{n-1}) \rightarrow_a^A q\}$. □

Determinism and Complementation

As a consequence, regular tree languages are closed under complementation.

Lemma (Closure under complementation)

Let A be a DBUTA accepting L. Then there is a DBUTA $\overline{A}$ accepting $\overline{L}$.

Proof.

Swap final and non-final states.

If $A = (\Sigma, Q, \rightarrow, Q_F)$, set $\overline{A} := (\Sigma, Q, \rightarrow, Q \setminus Q_F)$. $\square$

Regular tree languages are also closed under union.

10. XML Schema Languages

XML Schema Languages

An XML document

```
<lecture>
  <title>Applied Automata Theory</title>
  <block>
    <title>Finite Words</title>
    <topic>
      <title>WMSO</title>
      <goal>Satisfiability</goal>
      <approach>Buechi</approach>
    </topic>
  </block>
</lecture>
```

yields a tree that

- reflects the structure of the document
- without the data.

XML Schema Languages

Goal

Pose requirements on the structure of XML documents:

- Every lecture is split into blocks.
- Blocks are divided into topics.

Observation

- Requirements describe a tree language over the alphabet of tags.
- Such a description is called a schema.
- Document is valid wrt. a schema if it belongs to the tree language defined by the schema.

XML Schema Languages

Several XML schema languages exist

Document Type Definitions (DTD), XML Schema, Relax NG

Out interest: connection to automata theory

- Expressiveness (not here)
- Algorithmic problems
  - Is a document valid wrt. a schema? (membership in the language)
  - Is there a document that is valid for this schema? (sanity check, emptiness in language theory, subproblem for inclusion)
  - Are all documents valid wrt. one schema valid for another schema? (needed when merging archives/companies, inclusion in language theory)

Document Type Definitions

A document type definition (DTD) is an extended context-free grammar.

Has regular expressions on the right hand side, the content model.

Tree language of this grammar = all derivation trees.

```
<!DOCTYPE LECTURE [
    <!ELEMENT lecture   (title, (block+ | (topic, exercise?)+))>
    <!ELEMENT block     (title, (topic, exercise?)+)>
    <!ELEMENT topic     (title, goal, problem?, approach)>
    <!ELEMENT title     (#PCDATA)>
    ...
]>
```

Operators in the content model:

| :=	choice	+ :=	one or more occurrences
, :=	sequence	? :=	zero or one occurrence

#PCDATA := parsed character data, arbitrary character sequence for data

Document Type Definitions

The DTD

```
<!DOCTYPE LECTURE [
    <!ELEMENT lecture   (title, (block+ | (topic, exercise?)+))>
    <!ELEMENT block     (title, (topic, exercise?)+)>
    <!ELEMENT topic     (title, goal, problem?, approach)>
    <!ELEMENT title     (#PCDATA)>
    ...
]>
```

as an extended context-free grammar:

$$\begin{aligned} \textit{lecture} &\to \textit{title} \cdot (\textit{block}^+ + (\textit{topic}.(\textit{exercise} + \varepsilon))^+) \\ \textit{block} &\to \textit{title} \cdot (\textit{topic} \cdot (\textit{exercise} + \varepsilon))^+ \\ \textit{topic} &\to \textit{title} \cdot \textit{goal} \cdot (\textit{problem} + \varepsilon) \cdot \textit{approach} \\ \textit{title} &\to \varepsilon \\ &\dots \end{aligned}$$

To define the tree language described by a DTD, need hedge automata.

Unranked Trees

Reminder

In a ranked alphabet (Σ, rk), letters $a \in \Sigma$ have a rank $rk(a)$.

Σ-trees $t : T \to \Sigma$ obey the ranks.

Unranked trees

Consider again unranked alphabet Σ and corresponding unranked trees:

- Each node has arbitrarily but finitely many children.
- Tree $t : T \to \Sigma$ without further constraints is called an unranked tree.
- Call children $t_0, \ldots, t_{n-1}$ in $(a, (t_0, \ldots, t_{n-1}))$ a hedge.

Hedge Automata

Hedge automata process unranked trees bottom-up.

Goal: Solve membership.

Problem

Number of successors of a node is not bounded (unbounded, but finite branching in the language).

- Transitions cannot be listed.
- Represent symbolically this infinite number of transitions.

Definition (Hedge automata: syntax)

A (non-deterministic) hedge automaton (NHA) is a tuple $A = (\Sigma, Q, \rightarrow, Q_F)$ with

- Q a finite set of states, final states $Q_F \subseteq Q$, and
- $\rightarrow \subseteq \mathbb{P}(Q^*) \times \Sigma \times Q$.

We require $R \subseteq Q^*$ on the lhs of transitions to be regular.

These R are called horizontal languages.

Hedge Automata

Definition (Hedge automata: semantics)

Let $A = (\Sigma, Q, \to, Q_F)$ be an NHA.

A run of A on $t : T \to \Sigma$ is a function

$$r : T \to Q$$

so that for all $w \in T$ with $r(w) = q$, $t(w) = a$, and $n =$ number of successors of w, we have a transition

$$R \to_a q \quad \text{with} \quad r(w.0) \dots r(w.(n-1)) \in R.$$

To apply a transition $R \to_a q$ at a leaf, we need $\varepsilon \in R$.

A run is accepting if $r(\varepsilon) \in Q_F$.

Language of A is

$$L(A) := \{t : T \to \Sigma \mid \text{there is an accepting run of } A \text{ on } t\}.$$

Document Type Definitions

Definition (Document type definition)

A document type definition (DTD) is a tuple $D = (\Sigma, s, \delta)$ with

- start symbol $s \in \Sigma$
- function $\delta : \Sigma \to \mathrm{REG}_\Sigma$ that assigns each $a \in \Sigma$ a regular expression over Σ.

From DTDs to Hedge Automata

To define the language of a DTD $D = (\Sigma, s, \delta)$, understand it as hedge automaton

$$A_D := (\Sigma, \{q_a \mid a \in \Sigma\}, \to, \{q_s\}).$$

For the transitions, understand $L(\delta(a)) \subseteq \Sigma^*$ as subset of Q^* by taking $a_1 \dots a_n$ as $q_{a_1} \dots q_{a_n}$. With this:

$$L(\delta(a)) \to_a q_a \quad \text{for all } a \in \Sigma.$$

The language of a DTD is $L(D) := L(A_D)$.