

Applied Automata Theory

Roland Meyer

TU Kaiserslautern

Table of Contents I

1 Regular Languages and Finite Automata

- Regular Languages
- Finite Automata
- Equivalence
- Determinism and Complementation
- Decidability and Complexity

2 Weak Monadic Second-Order Logic

- Syntax and Semantics of WMSO
- Büchi's Theorem

3 Star-free Languages

- Ehrenfeucht-Fraïssé Games
- Star-free Languages
- McNaughton and Papert's Theorem

4 Presburger Arithmetic

- Syntax and Semantics of Presburger Arithmetic
- Representing Solution Spaces
- Quantifier Elimination
- Existential Presburger Arithmetic

5 Semi-linear Sets

- Definition of Semi-linear Sets

Table of Contents II

- Closure Properties of Semi-linear Sets
- Theorem of Ginsburg and Spanier

6 Parikh's Theorem

7 ω -Regular Languages and Büchi Automata

- ω -Regular Languages
- Büchi Automata
- Determinism

8 Linear-time Temporal Logic

- Syntax and Semantics of LTL
- From LTL to NBA

9 Model Checking Pushdown Systems

- Syntax and Semantics of Pushdown Systems
- Representation Structure: P -NFA
- Computing Predecessors
- Model Checking LTL

10 More on Infinite Words

11 Bottom-Up and Top-Down Tree Automata

- Syntax and Semantics of Bottom-Up Tree Automata
- Determinism and Complementation

12 XML Schema Languages

Table of Contents III

- Document Type Definitions
- Unranked Trees and Hedge Automata

Part A Finite Words

1. Regular Languages and Finite Automata

Basic Notions

Definition (Words)

- **Finite alphabet** = finite set of **letters** $\Sigma = \{a, b, c, \dots, n\}$
- **Finite word over Σ** = finite sequence of letters $w = a_0 \cdot \dots \cdot a_{n-1}$ 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 $L_1, L_2 \subseteq \Sigma^*$:

$$\begin{array}{c} L_1 \cup L_2 \\ \text{union} \end{array}$$

$$\begin{array}{c} L_1 \cap L_2 \\ \text{intersection} \end{array}$$

$$\begin{array}{c} L_1 \setminus L_2 \\ \text{difference} \end{array}$$

$$\begin{array}{c} \overline{L_1} := \Sigma^* \setminus L_1 \\ \text{complement} \end{array}$$

- **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 $i \in \mathbb{N} := \{0, 1, 2, \dots\}$.

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) $\emptyset \in \text{REG}_\Sigma$ and $\{a\} \in \text{REG}_\Sigma$ for all $a \in \Sigma$ and
- (2) if $L_1, L_2 \in \text{REG}_\Sigma$ then also $L_1 \cup L_2, L_1 \cdot L_2, L_1^* \in \text{REG}_\Sigma$.

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 , $\bar{}$, \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

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

Also say this is a **run of A on word $w := a_0 \dots a_{n-1}$** .

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 $L \in \text{REG}_\Sigma$ 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

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

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 $q \in Q$

there is precisely one state $q' \in Q$ 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?

$L(A) = w \in \Sigma^*$ so that **there is** an accepting run of A on w .

$\overline{L(A)} = w \in \Sigma^*$ so that **all** runs of A on w 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:

$L(A') = w \in \Sigma^*$ so that **there is** an accepting run of A' on w .

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

Closure under Complementation

Proposition (Closure under $\bar{}$)

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

Construction: Swap final states

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

Summary

Summary

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

- There are DFAs for L and \bar{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 $R_0 \subseteq R_1 \subseteq \dots$ 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 $|\rightarrow|$.



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

- $V_1 = \{x, y, z, \dots\}$ of **first-order variables**
- $V_2 = \{X, Y, Z, \dots\}$ 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:

$WMSO = WMSO[<, suc]$: all WMSO formulas

$WMSO[<]$, $WMSO[suc]$: formulas that only use predicates $<$ and suc

$FO[<, suc]$, $FO[<]$, $FO[suc]$: **first-order** formulas (over V_1 , only)

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

$$\varphi \wedge \psi := \neg(\neg\varphi \vee \neg\psi)$$

$$\forall x : \varphi := \neg\exists x : \neg\varphi$$

$$x \leq y := \neg(y < x)$$

$$first(x) := \neg\exists y : y < x$$

$$\varphi \rightarrow \psi := \neg\varphi \vee \psi$$

$$\forall X : \varphi := \neg\exists X : \neg\varphi$$

$$x = y := x \leq y \wedge y \leq x$$

$$last(x) := \neg\exists y : x < y$$

WMSO: Syntax

Definition (Bound and free variables)

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

- 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 $\varphi(x_1, \dots, x_m, X_1, \dots, X_n)$ to indicate that free variables of φ among x_1, \dots, 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**:

$$\text{(Bad)} \quad x < z \wedge \forall x : x < y \qquad x < z \wedge \forall x' : x' < y \quad \text{(Good)}$$

Example

$$\neg \exists y : y < x$$

y bound, x free, notation *first*(x)

$$\exists x : \textit{first}(x) \wedge X(x)$$

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.

Example

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

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

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

$P_a^S \subseteq D^S, <^S, suc^S \subseteq D^S \times D^S =$ 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

$$D^w := \{0, \dots, |w| - 1\}$$

$$<^w := <^{\mathbb{N}} \cap (D^w \times D^w)$$

$$suc^w := \{(0, 1), \dots, (|w| - 2, |w| - 1)\}$$

$$P_a^w := \{k \in D^w \mid w(k) = a\}$$

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** $I : V_1 \cup V_2 \rightarrow D^w \cup \mathbb{P}(D^w)$ that assigns (sets of) positions to free variables in φ (maybe to others, not important). With this:

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

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

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

- The empty word **does satisfy** second-order existential quantifiers

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

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 **WMSO[suc], WMSO[<], FO[suc], FO[<]-definable** by restricting φ .

Example

Please see the handwritten notes.

First Hierarchy of Languages

Distinguish between

$FO[suc]$, $FO[<]$, $FO[<, suc]$, $WMSO[suc]$, $WMSO[<]$, $WMSO[<, suc]$ -definability

Lemma

L is $FO[<, suc]$ -definable iff L is $FO[<]$ -definable (1)

L is $WMSO[<, suc]$ -definable iff L is $WMSO[<]$ -definable (2)

L is $WMSO[<, suc]$ -definable iff L is $WMSO[suc]$ -definable (3)

L is $WMSO[<, suc]$ -definable iff L is $WMSO_0$ -definable. (4)

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

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

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

$WMSO$ vs. FO : later. $FO[suc]$ vs. $FO[<]$: not this lecture.

From Finite Automata to WMSO

Goal

Establish $\text{REG} = \text{WMSO-definable}$.

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 $\varphi \in \text{WMSO}$. 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.

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

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

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

Consequences of Büchi's Theorem

Observation

Construction from NFAs to WMSO gave formulas of particular shape.

Existential WMSO, denoted by $\exists\text{WMSO}$, is restriction of WMSO to formulas

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

where φ does not contain second-order quantification.

Corollary

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

Proof.

Let $\varphi \in \text{WMSO}$. 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 $\text{FO}[\prec]$ defines a **strict** subclass of regular languages

(2) Find alternative characterization:

$\text{FO}[\prec]$ -definable iff represented by **star-free** regular expression

Recapitulation

First-order formulas are WMSO-formulas without second-order variables

Example over $\Sigma = \{a, b, c\}$:

$$\varphi := \forall x : P_a(x) \rightarrow \exists y : x < y \wedge P_b(y)$$

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: $\text{first}(x)$, $\text{last}(x)$, $x = y$ still in $\text{FO}[\prec]$

Star-free Languages

Towards Goal (1)

Known: FO[<]-definable languages are regular

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

For all $\psi \in \text{FO}[<]$ 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 $p : D^v \rightarrow D^w$ so that

- (1) Function p is injective.
- (2) For all $x \in \text{dom}(p)$ and all $a \in \Sigma$ we have $P_a^v(x)$ iff $P_a^w(p(x))$.
- (3) For all $x, y \in \text{dom}(p)$ we have $x <^v y$ iff $p(x) <^w p(y)$.

Let $\bar{s} = (s_1, \dots, s_n)$ and $\bar{t} = (t_1, \dots, t_n)$ two vectors of positions in D^v and D^w . Write $\bar{s} \mapsto \bar{t}$ 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 \bar{s}, \bar{t}
- Duplicator tries to establish partial isomorphism, starting from $\bar{s} \mapsto \bar{t}$
- Spoiler tries to avoid this

Ehrenfeucht-Fraïssé Games

Definition (EF-Game)

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

- k rounds
- Initial configuration $\bar{s} \mapsto \bar{t}$
- 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 $G_k((S(v), \bar{s}), (S(w), \bar{t}))$** 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 $G_k((S(v), \bar{s}), (S(w), \bar{t}))$ 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 $\varphi \in \text{FO}[<]$ is the maximal nesting depth of quantifiers in φ :

$$qd(x < y) := 0$$

$$qd(P_a(x)) := 0$$

$$qd(\neg\varphi) := qd(\varphi)$$

$$qd(\varphi_1 \vee \varphi_2) := \max\{qd(\varphi_1), qd(\varphi_2)\}$$

$$qd(\exists x : \varphi) := 1 + qd(\varphi)$$

Definition (k -equivalence)

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

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

In the case of empty sequences $\bar{s} = \varepsilon = \bar{t}$, 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 $G_k((S(v), \bar{s}), (S(w), \bar{t}))$ iff $(S(v), \bar{s}) \equiv_k (S(w), \bar{t})$.

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 $G_0((S(v), \bar{s}), (S(w), \bar{t}))$ iff $\bar{s} \mapsto \bar{t}$ is a partial isomorphism.*
- (2) *Duplicator wins $G_{k+1}((S(v), \bar{s}), (S(w), \bar{t}))$ iff*
 - (2.a) $\forall s \in D^v : \exists t \in D^w : \text{Duplicator wins } G_k((S(v), s.\bar{s}), (S(w), t.\bar{t}))$ *and*
 - (2.b) $\forall t \in D^w : \exists s \in D^v : \text{Duplicator wins } G_k((S(v), s.\bar{s}), (S(w), t.\bar{t}))$.

Intuition

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

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) $\emptyset, \{\varepsilon\} \in SF_{\Sigma}$ and $\{a\} \in SF_{\Sigma}$ for all $a \in \Sigma$ and
- (2) if $L_1, L_2 \in SF_{\Sigma}$ then also $L_1 \cup L_2, L_1 \cdot L_2, \overline{L_1} \in SF_{\Sigma}$.

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

Proof.

Homework. □

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. □

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

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

Proof.

Please see handwritten notes. □

McNaughton and Papert's Theorem

Theorem (McNaughton and Papert II, 1971)

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

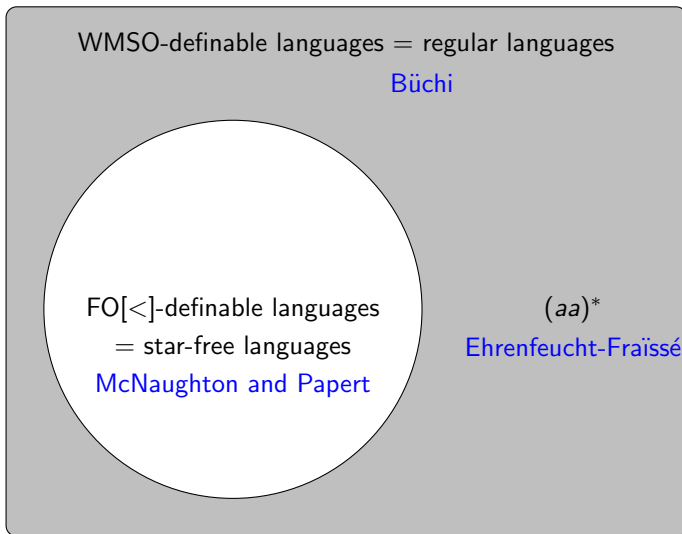
Proof.

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



Part B Natural Numbers

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

$$t_1 > t_2 := t_2 < t_1$$

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

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

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

$$n := \underbrace{1 + \dots + 1}_{n\text{-times}}$$

$$nx := \underbrace{x + \dots + x}_{n\text{-times}}$$

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

$$I \models t_1 < t_2 \quad \text{if} \quad I(t_1) <^{\mathbb{N}} I(t_2)$$

$$I \models \neg\varphi \quad \text{if} \quad I \not\models \varphi$$

$$I \models \varphi_1 \wedge \varphi_2 \quad \text{if} \quad I \models \varphi_1 \text{ and } I \models \varphi_2$$

$$I \models \exists x : \varphi \quad \text{if} \quad \text{there is } n \in \mathbb{N} \text{ so that } I[n/x] \models \varphi.$$

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

$$I(0) := 0^{\mathbb{N}}$$

$$I(1) := 1^{\mathbb{N}}$$

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

Definition (Equivalence)

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

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

Presburger Arithmetic: Semantics

Definition (Truth, solutions, definability)

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

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

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

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

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

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

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

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

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

$$A_{\neg\psi} := \overline{A_\psi}$$

$$A_{\varphi \vee \psi} := A_\varphi \cup A_\psi$$

$$A_{\exists x:\psi} := \pi_x(A_\psi)$$

Representing Solution Spaces

Remains to construct automaton for solutions of atomic formulas.

Notation

Atomic formulas can be assumed to be in form

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

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

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

For the construction, please see handwritten notes.

Lemma (Termination)

Let $\psi = \bar{a} \cdot \bar{x} \leq b$ 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 $\varphi \in \text{PA}$.

Approach (Replace quantifiers by concrete values)

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

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

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

To obtain quantifier elimination for Presburger arithmetic, we extend the signature by \equiv_m for all $m \geq 2$. The semantics is as expected.

Remark

Note that $\text{PA}[\lt]$ and $\text{PA}[\lt, (\equiv_m)_{m \geq 2}]$ are 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 $\exists x : \varphi(x, y_1, \dots, y_m) \in PA[<, (\equiv_m)_{m \geq 2}]$. We can effectively construct $\psi(y_1, \dots, y_m) \in PA[<, (\equiv_m)_{m \geq 2}]$ with

$$\exists x : \varphi(x, y_1, \dots, y_m) \quad \underset{\text{logical equivalence}}{\equiv} \quad \psi(y_1, \dots, y_m).$$

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.

Existential Presburger Arithmetic

Existential Presburger arithmetic **avoids negation**, but introduces equality and disjunction.

Definition (Existential Presburger arithmetic)

Consider the signature $Sig = (\{0/0, 1/0, +/2\}, \{\leq /2, = /2\})$. Formulas in **existential Presburger arithmetic** are defined by

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

We use $\exists PA$ to denote the set of all formulas in existential Presburger arithmetic.

Existential Presburger Arithmetic

With quantifier elimination, every Presburger formula is equivalent to an existential formula.

Theorem

For every formula $\varphi \in PA$ there is a formula $\psi \in \exists PA$ with $\varphi \equiv \psi$.

Proof.

With the quantifier elimination result, $\varphi \in PA$ has an equivalent formula $\varphi \equiv \rho$ with $\rho \in PA[<, (\equiv_m)_{m \geq 2}]$ **quantifier-free** and **negation-free**.

We remove the congruences in ρ by

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

The resulting formula is $\psi \in \exists PA$. □

Existential Presburger Arithmetic

Motivation

The interest in \exists PA is that **satisfiability** has a **low complexity**.

The proof encodes satisfiability into **integer linear programming** (ILP), which is the following problem:

Given: A matrix $A \in \mathbb{Z}^{m \times n}$ and $\bar{b} \in \mathbb{Z}^m$.

Problem: Does $A \cdot \bar{x} \geq \bar{b}$ have an integer solution $\bar{x} \in \mathbb{Z}^n$?

Theorem (von zur Gathen and Sieveking 1978)

ILP is NP-complete.

Existential Presburger Arithmetic

Lemma

Satisfiability in $\exists PA$ is NP-complete.

To check satisfiability of $\varphi \in \exists PA$, first move the existential quantifiers to the top. This takes linear time and yields a formula $\varphi' \equiv \varphi$.

In the next step, replace every disjunct $\psi_1 \vee \psi_2$ by one of its components, either ψ_1 or ψ_2 . This turns φ' into a formula φ'' , again in linear time. Intuitively, φ'' guesses the disjuncts that will be satisfied.

The resulting formula φ'' actually is an ILP problem. Extending it by lower bound constraints ensures we find a solution in \mathbb{N}^n . The formula is now φ''' .

Use Theorem 59 to solve the ILP problem φ''' in NP.

Altogether, this yields a non-deterministic algorithm that runs in polynomial time and reports positively iff $\exists PA$ is satisfiable.

NP-hardness is by NP-hardness of 0/1-ILP.

5. Semi-linear Sets

Semi-linear Sets: Definition

Goal

Show that semi-linear sets are precisely the sets of numbers that are Presburger-definable.

Consequences

Closure of semi-linear sets under **complement** (cool).

Closure of Presburger-definable sets under iteration.

Definition (Semi-linear sets)

Let $\bar{c} \in \mathbb{N}^n$ be a vector and $P \subseteq \mathbb{N}^n$ a **finite** set of vectors. We define

$$L(\bar{c}, P) := \{\bar{v} \in \mathbb{N}^n \mid \text{for each } p \in P \text{ there is } k_p \in \mathbb{N} \text{ so that } \bar{v} = \bar{c} + \sum_{p \in P} k_p \bar{p}\}.$$

Here, \bar{c} is called **constant** and P is the set of **periods**.

A set $M \subseteq \mathbb{N}^n$ is **linear** if $M = L(\bar{c}, P)$ for some $\bar{c} \in \mathbb{N}^n$ and $P \subseteq \mathbb{N}^n$ finite.

A set $S \subseteq \mathbb{N}^n$ is **semi-linear** if it is a finite union of linear sets.

Semi-linear Sets: Definition

Remark

- (1) Given a linear set $L(\bar{c}, P) \subseteq \mathbb{N}^n$ and a vector $\bar{v} \in \mathbb{N}^n$, it is **decidable** whether $\bar{v} \in L(\bar{c}, P)$ holds. The same decidability holds for semi-linear sets.
- (2) Linear sets are not closed under **any** of the Boolean operations: if $M_1, M_2 \subseteq \mathbb{N}^n$ are linear, then $\overline{M_1}$, $M_1 \cup M_2$, and $M_1 \cap M_2$ need not be linear.
- (3) The class of semi-linear sets properly includes the linear sets, i.e., every linear set is semi-linear.

Semi-linear Sets: Closure Properties

Definition (Linear functions)

A function $f : \mathbb{N}^n \rightarrow \mathbb{N}^m$ is called **linear** if

$$f(\bar{x} + \bar{y}) = f(\bar{x}) + f(\bar{y}) \quad \text{and} \quad f(k\bar{x}) = kf(\bar{x}) \quad \text{with } k \in \mathbb{N}.$$

Lemma (Closure under linear functions)

Let $S \subseteq \mathbb{N}^n$ be semi-linear and $f : \mathbb{N}^n \rightarrow \mathbb{N}^m$ be linear. Then $f(S) \subseteq \mathbb{N}^m$ is semi-linear.

Semi-linear Sets: Closure Properties

Definition (Iteration)

Let $A \subseteq \mathbb{N}^n$. We define

$$A^* := \{\bar{v}_1 + \dots + \bar{v}_k \in \mathbb{N}^n \mid \bar{v}_1, \dots, \bar{v}_k \in A\}.$$

Lemma (Closure under iteration)

If $S \subseteq \mathbb{N}^n$ is semi-linear, so is S^* .

Proof.

Let $S = L(\bar{c}_1, P_1) \cup \dots \cup L(\bar{c}_l, P_l)$. One can show that

$$S^* = \bigcup_{J \subseteq \{1, \dots, l\}} L\left(\sum_{i \in J} \bar{c}_i, \bigcup_{i \in J} P_i \cup \{\bar{c}_i\}\right).$$

□

Semi-linear Sets: Closure Properties

Lemma

If $S \subseteq \mathbb{N}^n$ is semi-linear and $\bar{c} \in \mathbb{N}^n$, then

$$\bar{c} + S := \{\bar{c} + \bar{x} \mid \bar{x} \in S\} \text{ is semi-linear.}$$

Theorem (Closure under \cup and \cap)

Let $S_1, S_2 \subseteq \mathbb{N}^n$ be semi-linear. Then $S_1 \cup S_2$ and $S_1 \cap S_2$ are semi-linear.

Proof.

For \cup there is nothing to do.

For \cap , it is sufficient to show that the intersection of linear sets forms a semi-linear set. For a semi-linear set $M_1 \cup M_2$, we then use

$$M \cap (M_1 \cup M_2) = (M \cap M_1) \cup (M \cap M_2).$$

The proof is on the board. □

Semi-linear Sets: Closure Properties

An **application** of the above closure properties is the following result.

Lemma (Closure under taking the inverse of linear functions)

Let $S \subseteq \mathbb{N}^m$ be semi-linear and $f : \mathbb{N}^n \rightarrow \mathbb{N}^m$ be linear. Then $f^{-1}(S) \subseteq \mathbb{N}^n$ is semi-linear.

Theorem of Ginsburg and Spanier

Theorem (Ginsburg and Spanier)

A set $S \subseteq \mathbb{N}^n$ is Presburger-definable if and only if it is semi-linear.

The proof is on the board.

Corollary (Closure properties)

If $S \subseteq \mathbb{N}^n$ is semi-linear, then \overline{S} is semi-linear.

If $S \subseteq \mathbb{N}^n$ is Presburger-definable, then so is S^ .*

6. Parikh's Theorem

Parikh Images

Goal

The Parikh image of a word $\psi(w)$ counts the occurrences of letters.

The goal is to show that $\psi(L(G))$ is semi-linear for every context-free grammar G .

The classical proof of Parikh directly shows semi-linearity of Parikh-images.

We present a different approach due to Verma, Seidl, and Schwentick from 2006: the Parikh image can be captured directly by a **small** Presburger formula.

What we show

Given a context-free grammar G , we construct in **linear time** an **existential** Presburger formula φ_G so that $Sol(\varphi_G) = \psi(L(G))$.

This is interesting as satisfiability for existential Presburger arithmetic is only NP-complete.

Part C 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: $\Box\Diamond\text{req}$
- New class of automata: **Büchi automata** — **system**.
- New logic: **Linear-time Temporal Logic (LTL)** — **specification**.
- New system **features**: Büchi **pushdown** automata — **recursion**.

7. 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 $w = a_0 \cdot a_1 \dots$ 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 $v^\omega := v \cdot v \cdot v \cdot \dots$
- Let $L \subseteq \Sigma^*$ with $L \cap \Sigma^+ \neq \emptyset$. Then

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

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 $V_0, \dots, V_{n-1} \subseteq \Sigma^*$, $W_0, \dots, W_{n-1} \subseteq \Sigma^*$ 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

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

- If $w = a_0 \cdot a_1 \cdot a_2 \cdot \dots$, 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.

8. 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

_____ α _____ a _____ β _____

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:

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

- 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

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

Definition (Abbreviations)

Use standard abbreviations for Boolean operators. Moreover:

$$\underbrace{\diamond \varphi}_{\text{eventually}} := \text{true } \mathcal{U} \varphi \quad \underbrace{\square \varphi}_{\text{always}} := \neg \diamond \neg \varphi \quad \underbrace{\varphi \mathcal{R} \psi}_{\text{release}} := \neg(\neg \varphi \mathcal{U} \neg \psi)$$

Definition (Size)

The **size** of an LTL formula is defined inductively by

$$\begin{aligned} |p| &:= 1 & |\neg \varphi| &:= 1 + |\varphi| & |\bigcirc \varphi| &:= 1 + |\varphi| \\ |\varphi * \psi| &:= |\varphi| + 1 + |\psi| & \text{with } * \in \{\vee, \wedge, \mathcal{U}, \mathcal{R}\} & & & \end{aligned}$$

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

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

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

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: $\Box(\text{req} \rightarrow \Diamond\text{ack})$
- 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

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

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:

$$\neg \bigcirc \varphi \equiv \bigcirc \neg \varphi$$

$$\neg(\varphi \mathcal{U} \psi) \equiv \neg(\neg(\neg\varphi) \mathcal{U} \neg(\neg\psi)) \equiv \neg\varphi \mathcal{R} \neg\psi$$

$$\neg(\varphi \mathcal{R} \psi) \equiv \neg\neg(\neg\varphi \mathcal{U} \neg\psi) \equiv \neg\varphi \mathcal{U} \neg\psi$$



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

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

A run is **accepting** if $\text{Inf}(r) \cap Q_F^i \neq \emptyset$ 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: $Q' := Q \times \{1, \dots, k\}$
 - ▶ (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 $\theta \in \text{LTL}$
- Intuitively, we take the formulas that currently hold

Definition (Fisher-Ladner Closure)

Let $\theta \in \text{LTL}$ be a formula in positive normal form. Its **Fisher-Ladner closure** $FL(\theta) \subseteq \text{LTL}$ 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)*
 - if $\varphi \vee \psi \in M$ then $\varphi \in M$ 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 $M \subseteq \text{FL}(\theta)$ that satisfies the following closure properties:

$$\varphi \vee \psi \in M \quad \text{implies} \quad \varphi \in M \text{ or } \psi \in M$$

$$\varphi \wedge \psi \in M \quad \text{implies} \quad \varphi \in M \text{ and } \psi \in M$$

$$\varphi \mathcal{U} \psi \in M \quad \text{implies} \quad \psi \in M \text{ or } (\varphi \in M \text{ and } \bigcirc(\varphi \mathcal{U} \psi) \in M)$$

$$\varphi \mathcal{R} \psi \in M \quad \text{implies} \quad \psi \in M \text{ and } (\varphi \in M \text{ or } \bigcirc(\varphi \mathcal{R} \psi) \in M)$$

A Hintikka set $M \subseteq \text{FL}(\theta)$ 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 $M \subseteq \text{FL}(\theta)$** is

$$\mathcal{P}^+(M) := M \cap \mathcal{P}$$

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

Vardi-Wolper Construction

Construct an automaton \mathcal{A}_θ 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 $\varphi_1 \mathcal{U} \psi_1, \dots, \varphi_k \mathcal{U} \psi_k$ 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 i th until formula needs to be fulfilled then this happens in M

$$M \xrightarrow{a} M' \quad \text{if} \quad \{\psi \in FL(\theta) \mid \bigcirc \psi \in M\} \subseteq M'$$

$$\text{and} \quad \mathcal{P}^+(M) \subseteq a \quad \text{and} \quad \mathcal{P}^-(M) \cap a = \emptyset$$

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}_θ satisfies $L(\theta) = L(\mathcal{A}_\theta)$ and $|\mathcal{A}_\theta| \leq 2^{8|\theta|}$.

9. 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**:

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

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 $X_{i+1} := X_i \cup \text{pre}(X_i)$.

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
- ⇒ Algorithm to compute $pre^*(C)$
- ⇒ 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 $P = (Q, \Gamma, \rightarrow)$ 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 $\rightarrow \subseteq CF \times CF$ between configurations:

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

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

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

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:

$$Y_0 \subseteq Y_1 \subseteq Y_2 \subseteq \dots$$

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 :

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

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 $A_0 := A$. Moreover, let $A_i = (\Gamma, S, S_I, \rightarrow_i, S_F)$. Then we set

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

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

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

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

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

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 i th 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, Γ, \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 $pre^*({q} \times \gamma \cdot \Gamma^*)$.

10. 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**:

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

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 $\text{Inf}(r) \cap Q_F \neq \emptyset$.

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

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

Streett automata

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

Run is **accepting** if

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

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 $\text{Inf}(r) \in \mathcal{F}$.

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

$$A_1 \times \dots \times A_n.$$

- 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.

11. 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**

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

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

$$t : T \rightarrow \Sigma$$

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}_Σ 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 $t : T \rightarrow \Sigma$ be a tree. Its **yield** is defined inductively:

- (1) If $T = \{\varepsilon\}$ then $yield(t) := t(\varepsilon)$.
- (2) Let $T = \{\varepsilon\} \cup 0.T_0 \cup \dots \cup n.T_n$. Define subtrees

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

With this:

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

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 $A = ((\Sigma, rk), Q, \rightarrow, Q_F)$ with

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

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

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

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

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

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

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

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

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

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 \bar{A} accepting \bar{L} .

Proof.

Swap final and non-final states.

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

Regular tree languages are also closed under union.

12. 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

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:

$$\textit{lecture} \rightarrow \textit{title} \cdot (\textit{block}^+ + (\textit{topic} \cdot (\textit{exercise} + \varepsilon))^+)$$
$$\textit{block} \rightarrow \textit{title} \cdot (\textit{topic} \cdot (\textit{exercise} + \varepsilon))^+$$
$$\textit{topic} \rightarrow \textit{title} \cdot \textit{goal} \cdot (\textit{problem} + \varepsilon) \cdot \textit{approach}$$
$$\textit{title} \rightarrow \varepsilon$$

...

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 $t : T \rightarrow \Sigma$ obey the ranks.

Unranked trees

Consider again **unranked alphabet** Σ and corresponding **unranked trees**:

- Each node has arbitrarily but finitely many children.
- Tree $t : T \rightarrow \Sigma$ without further constraints is called an **unranked tree**.
- Call children t_0, \dots, t_{n-1} in $(a, (t_0, \dots, t_{n-1}))$ 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 $A = (\Sigma, Q, \rightarrow, Q_F)$ be an NHA.

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

$$r : T \rightarrow Q$$

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

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

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

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

Language of A is

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

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 $\delta : \Sigma \rightarrow \text{REG}_\Sigma$ 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

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

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:

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

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