

Theory of Computer Science

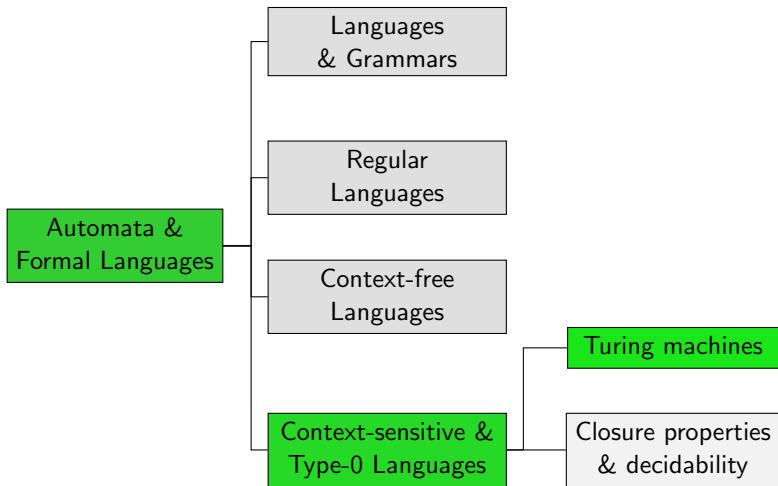
C8. Type-1 and Type-0 Languages: Closure & Decidability

Gabriele Röger

University of Basel

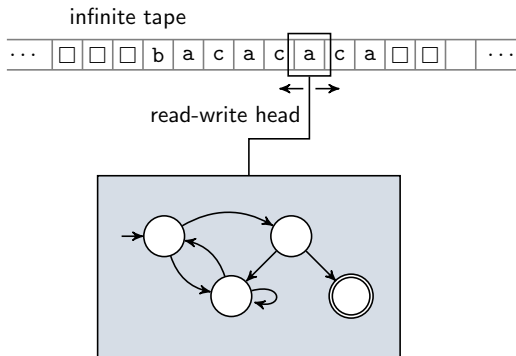
April 8, 2019

Overview



Turing Machines vs. Grammars

Reminder: Turing Machines – Conceptually



Reminder: Nondeterministic Turing Machine

Definition (Nondeterministic Turing Machine)

A nondeterministic **Turing machine (NTM)** is given by a 7-tuple $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \square, E \rangle$ with:

- Q finite non-empty set of **states**
- $\Sigma \neq \emptyset$ finite **input alphabet**
- $\Gamma \supset \Sigma$ finite **tape alphabet**
- $\delta : (Q \setminus E) \times \Gamma \rightarrow \mathcal{P}(Q \times \Gamma \times \{L, R, N\})$ **transition function**
- $q_0 \in Q$ **start state**
- $\square \in \Gamma \setminus \Sigma$ **blank symbol**
- $E \subseteq Q$ **end states**

One Automata Model for Two Grammar Types?

Don't we need
different automata models for
context-sensitive and type-0
languages?



Linear Bounded Automata: Idea

- **Linear bounded automata** are NTMs that may only use the **part of the tape occupied by the input word**.
- one way of formalizing this: NTMs where blank symbol may never be replaced by a different symbol

Linear Bounded Turing Machines: Definition

Definition (Linear Bounded Automata)

An NTM $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \square, E \rangle$

is called a **linear bounded automaton (LBA)**

if for all $q \in Q \setminus E$ and all transition rules $\langle q', c, y \rangle \in \delta(q, \square)$
we have $c = \square$.

German: linear beschränkte Turingmaschine

LBA's Accept Type-1 Languages

Theorem

The languages that can be accepted by linear bounded automata are exactly the context-sensitive (type-1) languages.

Without proof.

LBAs Accept Type-1 Languages

Theorem

The languages that can be accepted by linear bounded automata are exactly the context-sensitive (type-1) languages.

Without proof.

proof sketch for grammar \Rightarrow NTM direction:

- computation of the NTM follows the production of the word in the grammar **in opposite order**
- accept when only start symbol (and blanks) are left on the tape
- because language is context-sensitive, we never need additional space on the tape (empty word needs special treatment)

NTMs Accept Type-0 Languages

Theorem

The languages that can be accepted by nondeterministic Turing machines are exactly the type-0 languages.

Without proof.

NTMs Accept Type-0 Languages

Theorem

The languages that can be accepted by nondeterministic Turing machines are exactly the type-0 languages.

Without proof.

proof sketch for grammar \Rightarrow NTM direction:

- analogous to previous proof
- for grammar rules $w_1 \rightarrow w_2$ with $|w_1| > |w_2|$, we must “insert” symbols into the existing tape content; this is a bit tedious, but not very difficult

Deterministic Turing Machines

Definition (Deterministic Turing Machine)

A **deterministic Turing machine (DTM)** is a Turing machine $M = \langle Q, \Sigma, \Gamma, \delta, q_0, \square, E \rangle$ with $\delta : (Q \setminus E) \times \Gamma \rightarrow Q \times \Gamma \times \{L, R, N\}$.

German: deterministische Turingmaschine

Deterministic Turing Machines vs. Type-0 Languages

Theorem

For every type-0 language L there is a deterministic Turing machine M with $\mathcal{L}(M) = L$.

Without proof.

Deterministic Turing Machines vs. Type-0 Languages

Theorem

For every type-0 language L there is a deterministic Turing machine M with $\mathcal{L}(M) = L$.

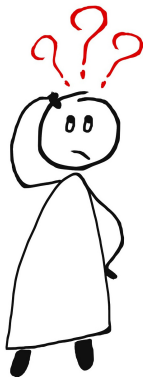
Without proof.

proof sketch:

- Let M' be an NTM with $\mathcal{L}(M') = L$.
- It is possible to construct a DTM that systematically searches for an accepting configuration in the computation tree of M' .

Note: It is an open problem whether an analogous theorem holds for type-1 languages and deterministic LBAs.

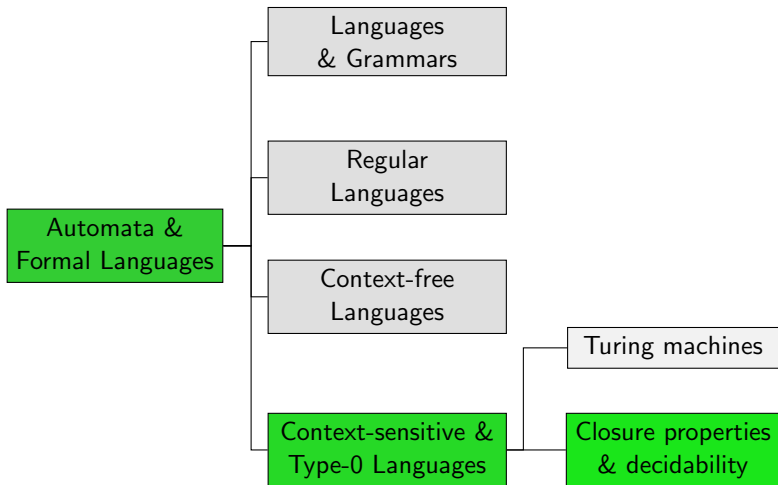
Questions



Questions?

Closure Properties and Decidability

Overview



Closure Properties

	Intersection	Union	Complement	Product	Star
Type 3	Yes	Yes	Yes	Yes	Yes
Type 2	No	Yes	No	Yes	Yes
Type 1	Yes ⁽²⁾	Yes ⁽¹⁾	Yes ⁽²⁾	Yes ⁽¹⁾	Yes ⁽¹⁾
Type 0	Yes ⁽²⁾	Yes ⁽¹⁾	No ⁽³⁾	Yes ⁽¹⁾	Yes ⁽¹⁾

Proofs?

- (1) proof via grammars, similar to context-free cases
- (2) without proof
- (3) proof in later chapters (part D)

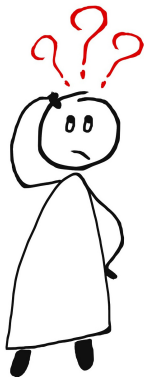
Decidability

	Word problem	Emptiness problem	Equivalence problem	Intersection problem
Type 3	Yes	Yes	Yes	Yes
Type 2	Yes	Yes	No	No
Type 1	Yes ⁽¹⁾	No ⁽³⁾	No ⁽²⁾	No ⁽²⁾
Type 0	No ⁽⁴⁾	No ⁽⁴⁾	No ⁽⁴⁾	No ⁽⁴⁾

Proofs?

- (1) same argument we used for context-free languages
- (2) because already undecidable for context-free languages
- (3) without proof
- (4) proofs in later chapters (part D)

Questions



Questions?

Summary

Summary

- **Turing machines** accept exactly the **type-0** languages. This is also true for **deterministic Turing machines**.
- **Linear bounded automata** accept exactly the **context-sensitive** languages.
- The context-sensitive and type-0 languages are **closed** under **almost all** usual operations.
 - exception: **type-0 not closed** under **complement**
- For context-sensitive and type-0 languages **almost no problem is decidable**.
 - exception: **word problem** for **context-sensitive** lang. decidable

What's Next?

contents of this course:

- A. **background** ✓
 - ▷ mathematical foundations and proof techniques
- B. **logic** ✓
 - ▷ How can knowledge be represented?
How can reasoning be automated?
- C. **automata theory and formal languages**
 - ▷ What is a computation?
- D. **Turing computability**
 - ▷ What can be computed at all?
- E. **complexity theory**
 - ▷ What can be computed efficiently?
- F. **more computability theory**
 - ▷ Other models of computability

What's Next?

contents of this course:

- A. **background** ✓
 - ▷ mathematical foundations and proof techniques
- B. **logic** ✓
 - ▷ How can knowledge be represented?
How can reasoning be automated?
- C. **automata theory and formal languages** ✓
 - ▷ What is a computation?
- D. **Turing computability**
 - ▷ What can be computed at all?
- E. **complexity theory**
 - ▷ What can be computed efficiently?
- F. **more computability theory**
 - ▷ Other models of computability