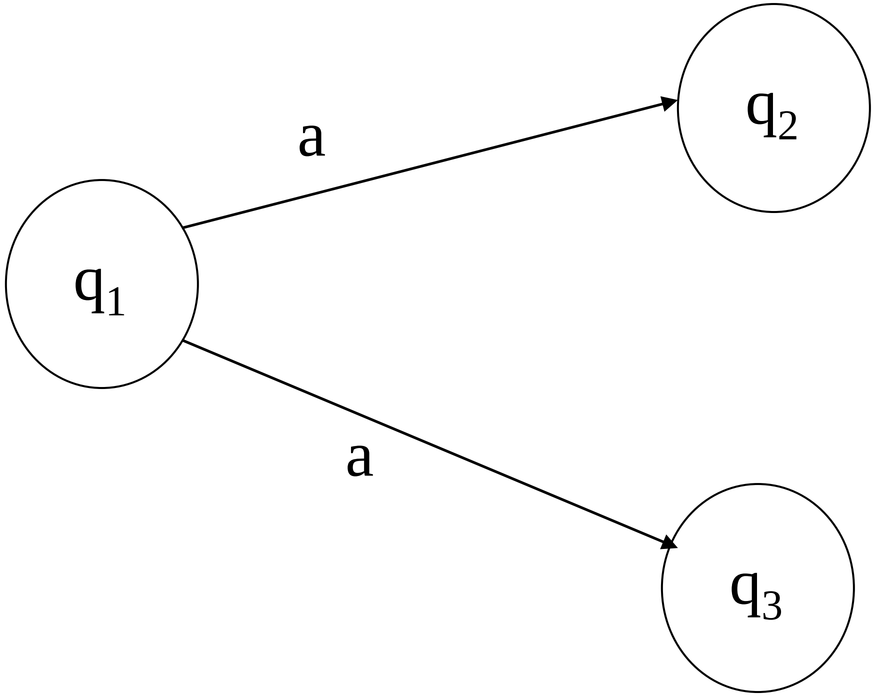
Theory of Computation

NFSA - recap

Lecture 11 - Manuel Mazzara

Adding nondeterminsm



$$\delta(q_1,a)= \{q_2, q_3\}$$

Nondeterministic FSA

- A nondeterministic FSA (NDFSA) is a tuple

$<Q, I, \delta, q_0, F>$, where
  - Q, I, q_0, F are defined as in (D)FSAs
  - $\delta: Q \times I \to \mathcal{P}(Q)$

A set of states

NDFSA into DFSA

- **NDFSA have the same power then DFSA**
- Given a NDFSA, an equivalent DFSA can be **automatically** computed as follows:

  If $\mathbf{A_{ND} = <Q, I, \delta, q_0, F>}$ then $\mathbf{A_D = <Q_D, I, \delta_D, q_{0D}, F_D>}$ with

  - $Q_D = \mathcal{P}(Q)$
  - $\delta_D(q_D, i) = \bigcup_{q \in q_D} \delta(q, i)$
  - $q_{0D} = \{q_0\}$
  - $F_D = \{q_D \mid q_D \in Q_D \wedge q_D \cap F \neq \emptyset\}$

Example (today in lab)

- The concept is simple, just take some time to fully go trough an example

- You will see an example in detail during the lab sessions

- There are quite good online examples too
  - https://www.youtube.com/watch?v=pnyXgIXpKnc

Why ND?

- **NDFSAs are not more powerful than FSAs**, but they are not useless
  - **It can be easier to design a NDFSA**
  - They can be exponentially smaller w.r.t. the number of states
  - See the example in the lab

- Example: a NDFSA with 5 states becomes in the worst case an FSA with 2^5 states

Is the class of languages recognized by NFAs **closed under complement**?

You should be able to build a <u>**formal proof**</u> - it is simple, do not search too far!

Theoretical Computer Science

Nondeterministic TM

Lecture 11 - Manuel Mazzara

Nondeterministic TM

- To define a nondeterministic TM (NDTM), we need to change the **transition function** and, if used as a transducer, the **translation mapping**
- All the other elements remain as in a (D)TM
- The transition function is

$$\delta: (Q\text{-}F) \times I \times \Gamma^k \rightarrow \mathcal{P}(Q \times \Gamma^k \times \{R,L,S\}^{k+1})$$

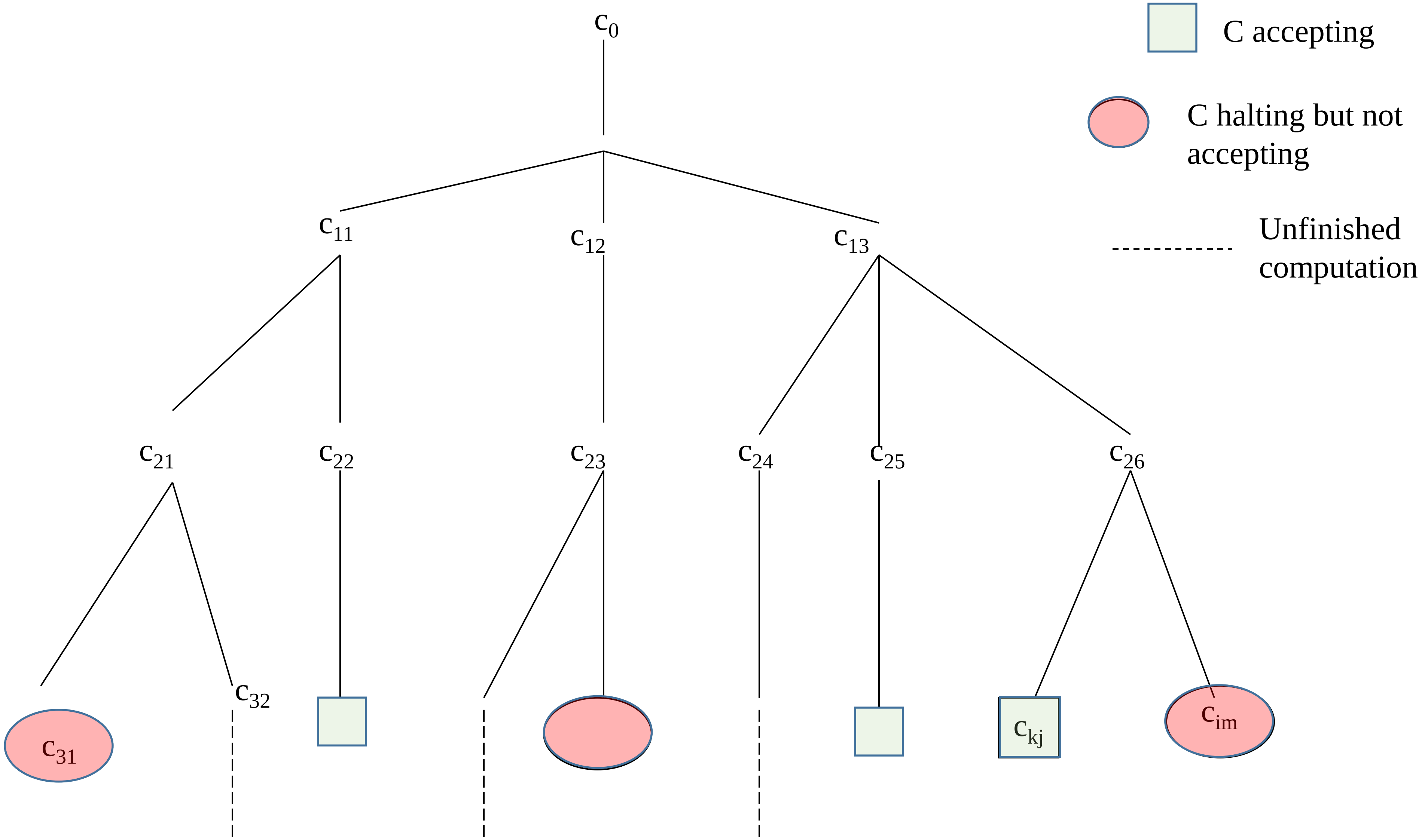
We have not seen this in detail, but it works like every other transducer

and the output mapping

$$\eta: (Q\text{-}F) \times I \times \Gamma^k \rightarrow \mathcal{P}(O \times \{R,S\})$$

Characteristic of nondeterminism is that the computation is “branching”

TM computation tree

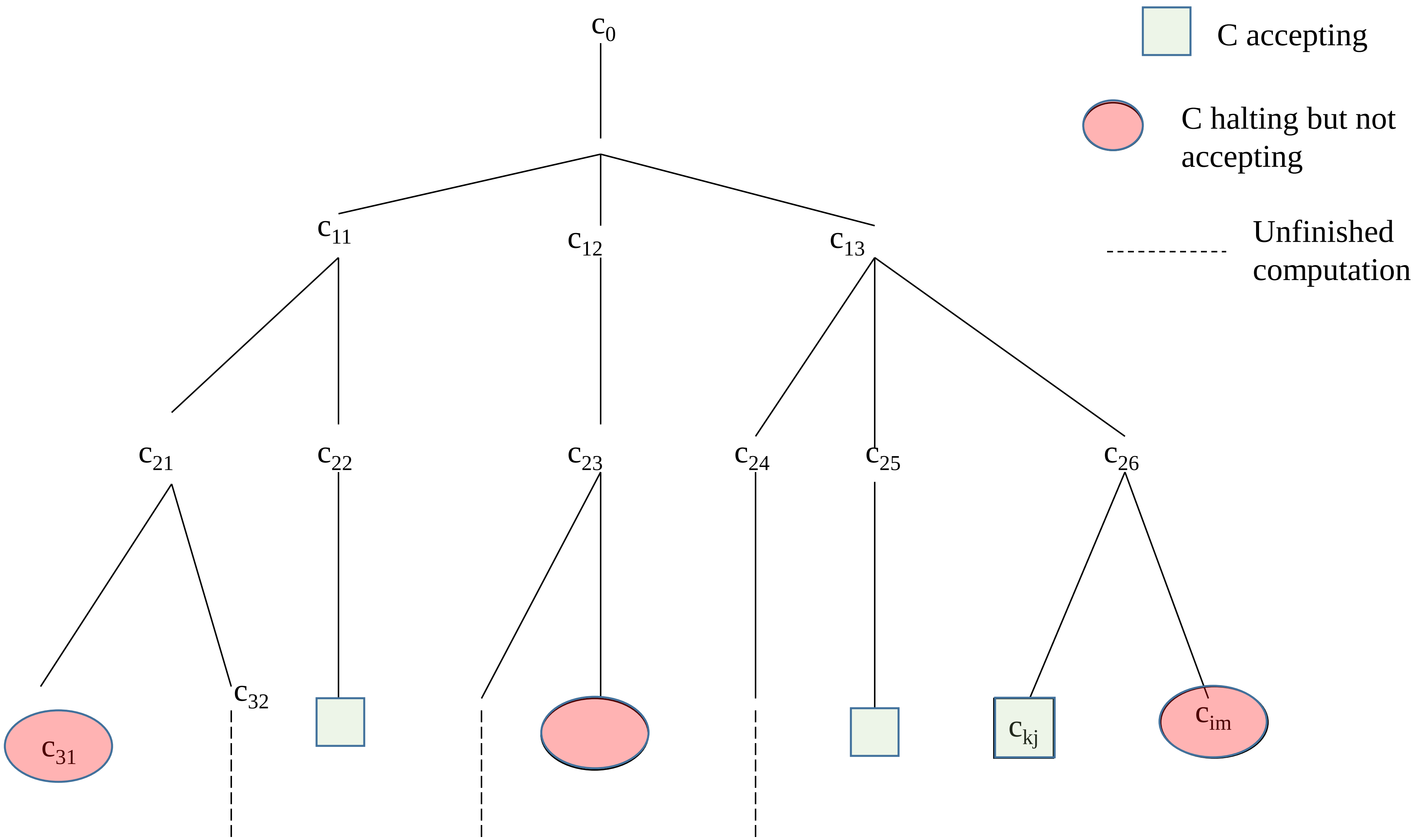


Acceptance condition

- A string $x \in I^*$ is accepted by a NDTM if and only if **there exists a computation that terminates in an accepting state**
  - **Existential** nondeterminism

- The problem of accepting a string can be reduced to **a visit of the TM computation tree**
  - *How should we perform the visit?*
  - What about the **relationship between DTMs and NDTMs**?

NDTM computation tree



Trees, unlike linked lists, one-dimensional arrays and other linear data structures, which are canonically traversed in linear order, can be traversed in multiple ways

What algorithms do you know for tree traversal?

Which one can be used to traverse infinite trees?

Visiting the computation tree

- We know different kinds of visits:
  - Depth-first visit
  - **Breadth-first visit**

- A depth-first visit cannot work
- **The computation tree may exhibit infinite paths**
- The algorithm would "get stuck"
- **Breadth-first visit algorithm**

DTM vs NDTM

- Can we build a **DTM that visits a tree level by level**?
  - It is a cumbersome exercise, but it is theoretically possible
- We can build a **DTM that establishes whether a NDTM recognizes a string *x***
- Given a NDTM, we can simulate via DTM
- **ND does not add power to TMs**

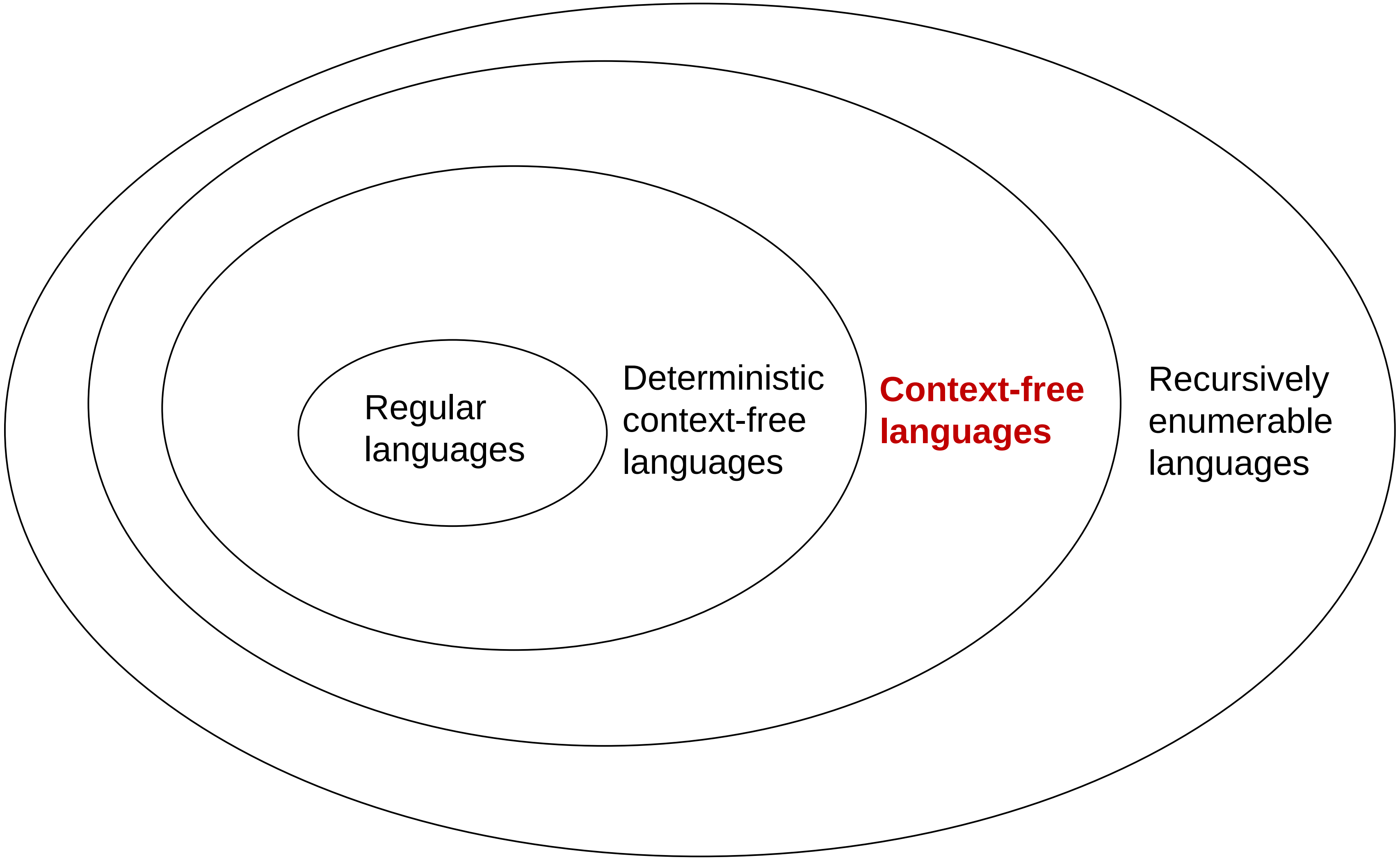
Summary

- **DFSA and NFSA have the same expressive power**

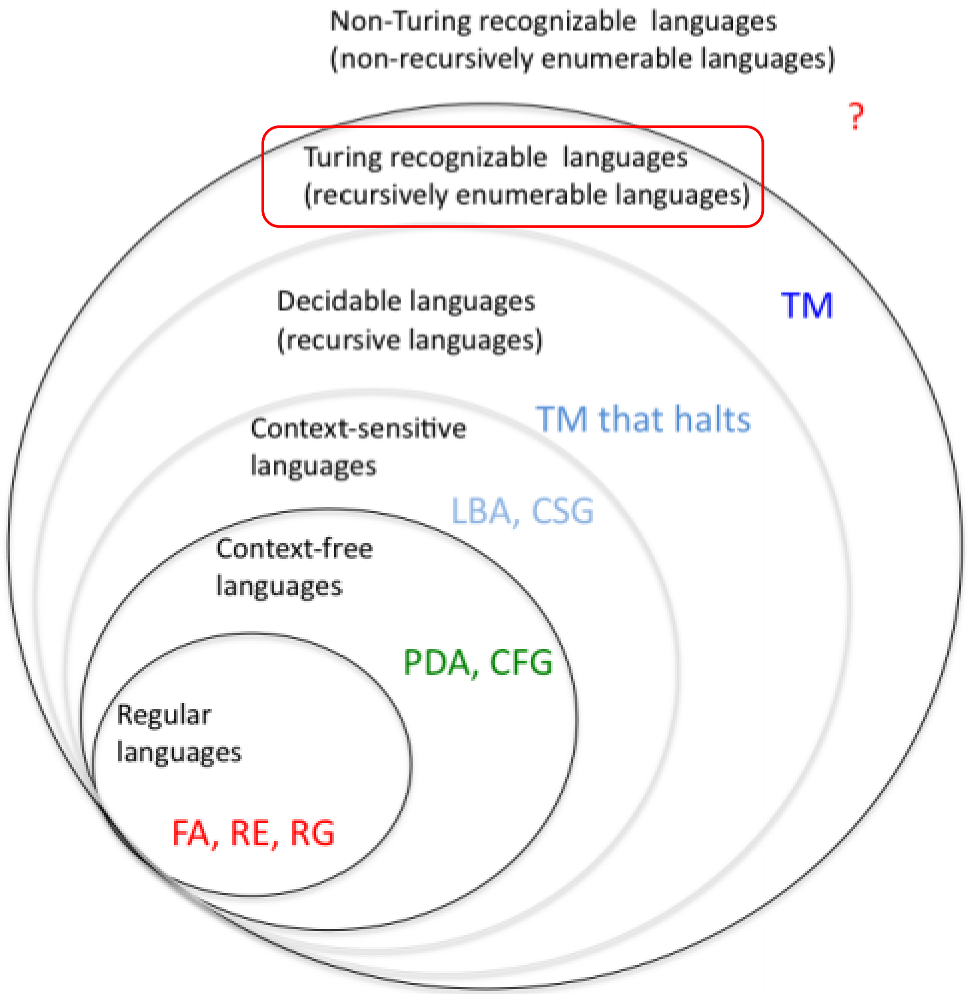
- **DTM and NTM have the same expressive power**

- What about PDA?
  - Deterministic vs nondeterministic context-free languages

The bigger picture



A jump ahead



FA: finite state automaton
RE: regular expression
RG: regular grammar

CFG: context-free grammar
PDA: pushdown automaton

LBA: linear-bounded automaton
CSG: context-sensitive grammar
TM: Turing machine

Theoretical Computer Science

Nondeterministic PDA

Lecture 11 - Manuel Mazzara

…one little step back before fully jumping into nondeterminism!

ε-moves

- In this course we have considered PDA with ε-moves, but we did not consider FSA with ε-moves
  - FSA with ε-moves if not properly constrained is nondeterministic, like PDA

- One of the possible generalization of NFSA is the **Nondeterministic Finite State Automata with ε-moves**, sometime called NFSA-ε or NFA-ε
  - NFSA-ε is equivalent to NFSA
  - NFSA is equivalent to FSA
  - NFSA-ε is also equivalent to FSA

PDA "sensitiveness" to model modifications

- FSAs are simple to manage, all the variants and generalizations we have seen have the **same expressive power** in terms of language recognition

- PDA is more complex, slight variations of the model influence significantly the expressiveness
  - Acceptance criteria (final state vs empty stack)
  - Number of stacks (differently from TMs)
  - **Deterministic vs nondeterministic**
  - With ε-moves vs without ε-moves

PDA with ε-moves vs PDA without ε-moves

- A notable example of PDA “sensitiveness”

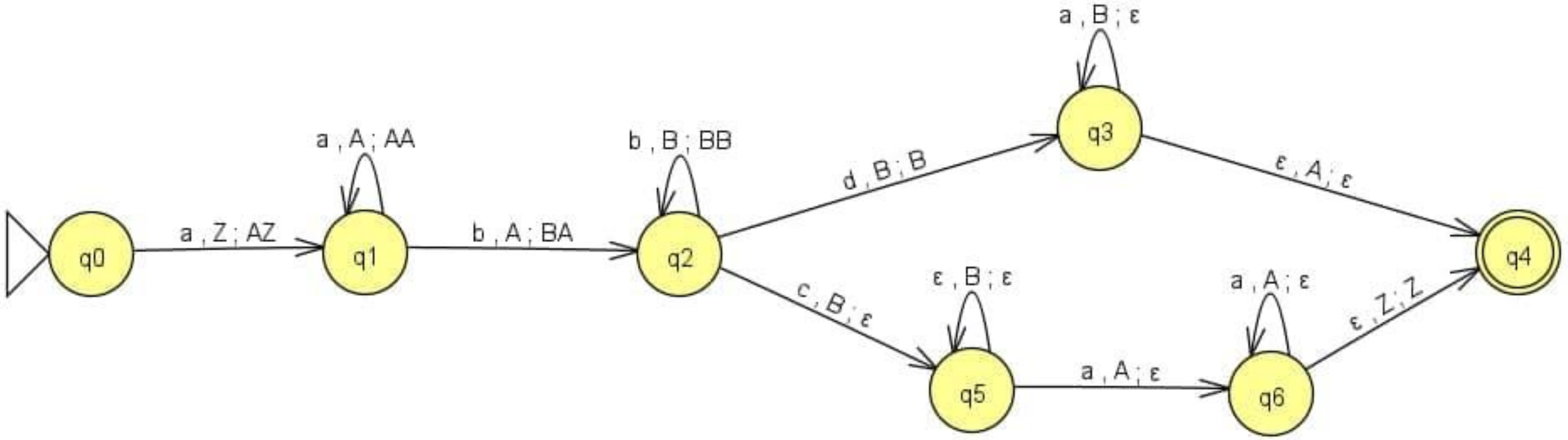
- PDA without ε -moves are also known as **realtime deterministic pushdown automata**

- They are less powerful than deterministic PDA

Example

- The following language L is deterministic and can be recognized by a deterministic PDA but cannot be recognized by any *realtime deterministic pushdown automata*

$$L = \{a^n b^p c a^n \mid p, n > 0\} \cup \{a^n b^p d b^p \mid p, n > 0\}$$



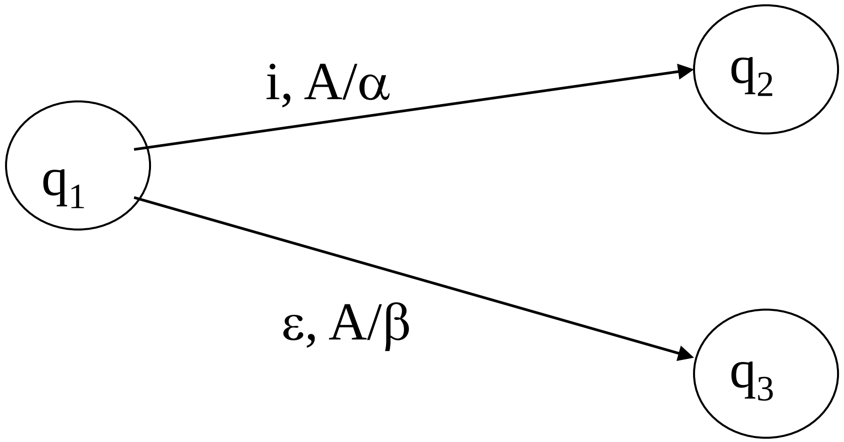
Thanks to Mansur K. for designing and drawing

ε-moves and PDAs

- **ε-moves** came with the following constraint:

$$\text{If } \delta(q,\varepsilon,A)\neq\perp\text{, then } \delta(q,i,A)=\perp \quad \forall i\in I$$

- Without this constraint the presence of ε-moves would make PDAs intrinsically **<u>nondeterministic</u>**



Adding nondeterminism to PDAs

- **Removing the constraint already makes the PDA nondeterministic**
- Additionally, we can have nondeterminism by **changing the transition function of a PDA** and consequently:
  - transitions among configurations
  - acceptance condition

Definition

A **nondeterministic PDA (NDPDA)** is a tuple

<Q, I, Γ, δ, q_0, Z_0, F>

- Q, I, Γ, q_0, Z_0, F as in (D)PDA
- δ is the **transition function** defined as

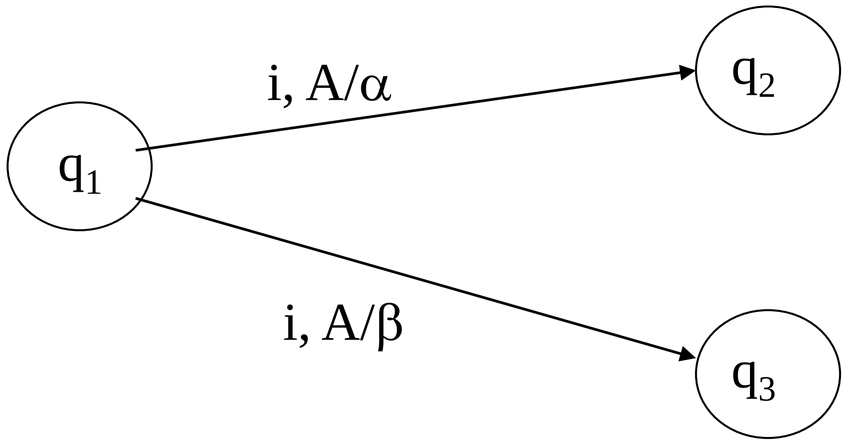
$$\boldsymbol{\delta: Q \times (I \cup \{\varepsilon\}) \times \Gamma \rightarrow \mathcal{P}_{\mathcal{F}}(Q \times \Gamma^*)}$$

- What is the $\mathcal{F}$ in $\mathcal{P}_{\mathcal{F}}$?
- Why $\mathcal{F}$?

Transition function

$$\delta\text{: } Q \times (I \cup \{\varepsilon\}) \times \Gamma \rightarrow \mathcal{P}_{\mathcal{F}}(Q \times \Gamma^*)$$

- $\mathcal{P}_{\mathcal{F}}$ indicates the *finite* subsets of $Q \times \Gamma^*$
  - Why did we not specify it for NDTM?
- Graphically:



Effects of nondeterminism

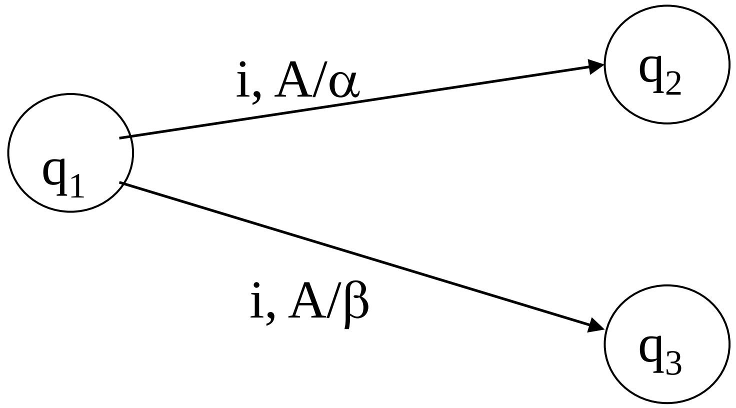
- ND **does not add expressive power** to
  - TMs
  - FSAs

- Does ND add expressive power to DPDAs?

NDPDAs vs DPDAs (1)

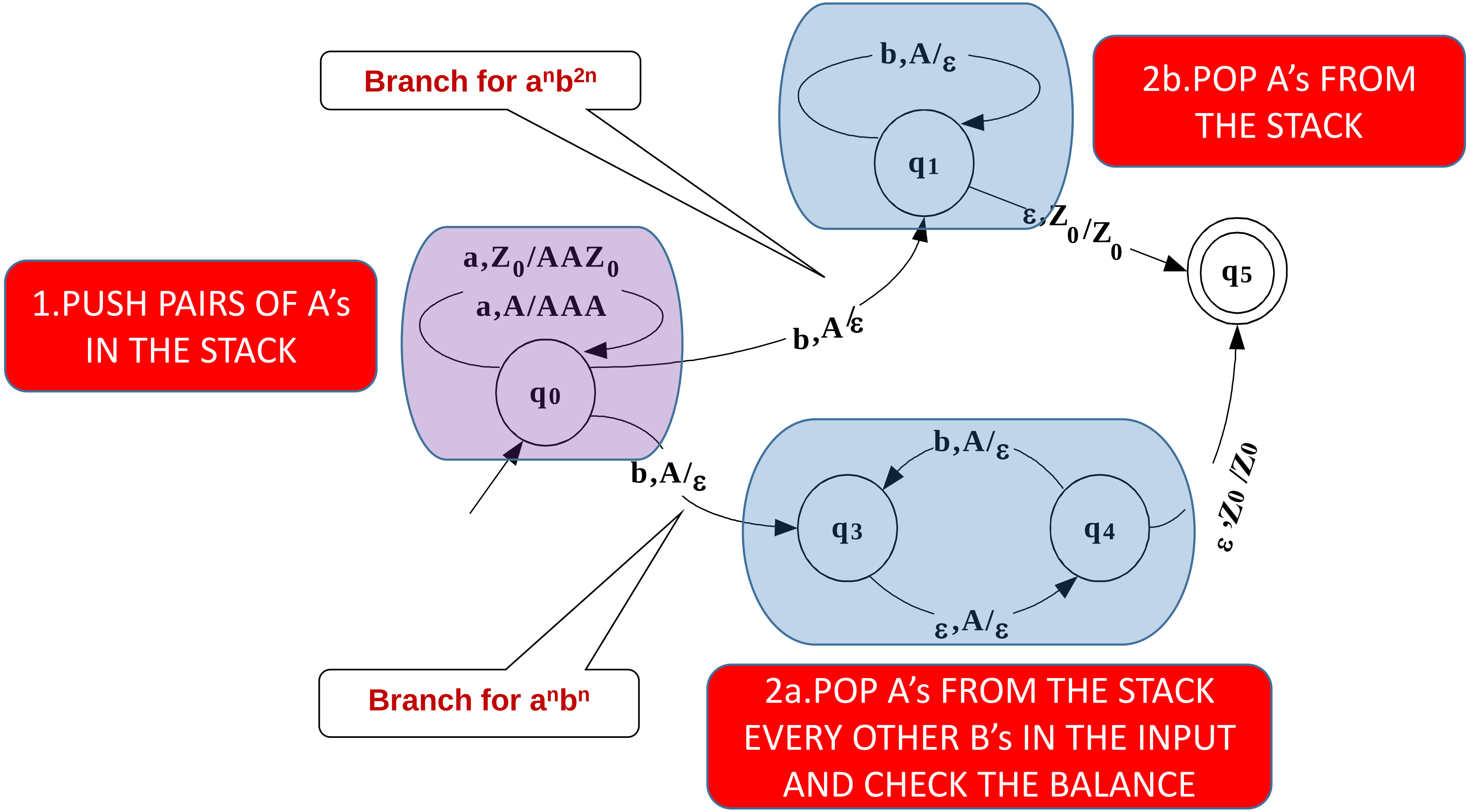
- Obviously **a NDPDA can recognize all the languages recognizable by DPDAs**

- ND allows transitions as follows:

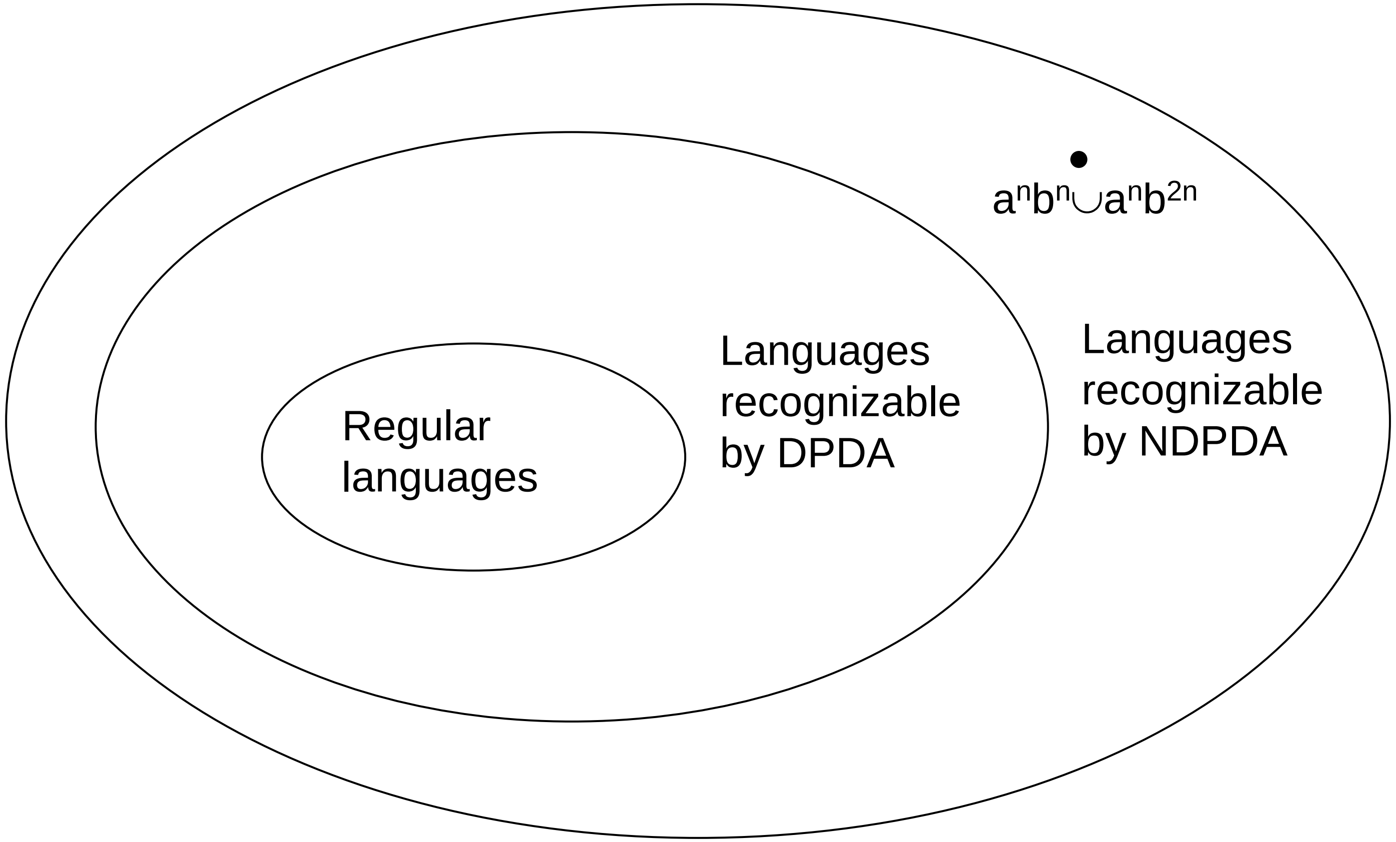


- **NDPDAs (as TMs) can recognize $\{a^n b^n \mid n \geq 1\} \cup \{a^n b^{2n} \mid n \geq 1\}$**

$\{a^n b^n \mid n \geq 1\} \cup \{a^n b^{2n} \mid n \geq 1\}$

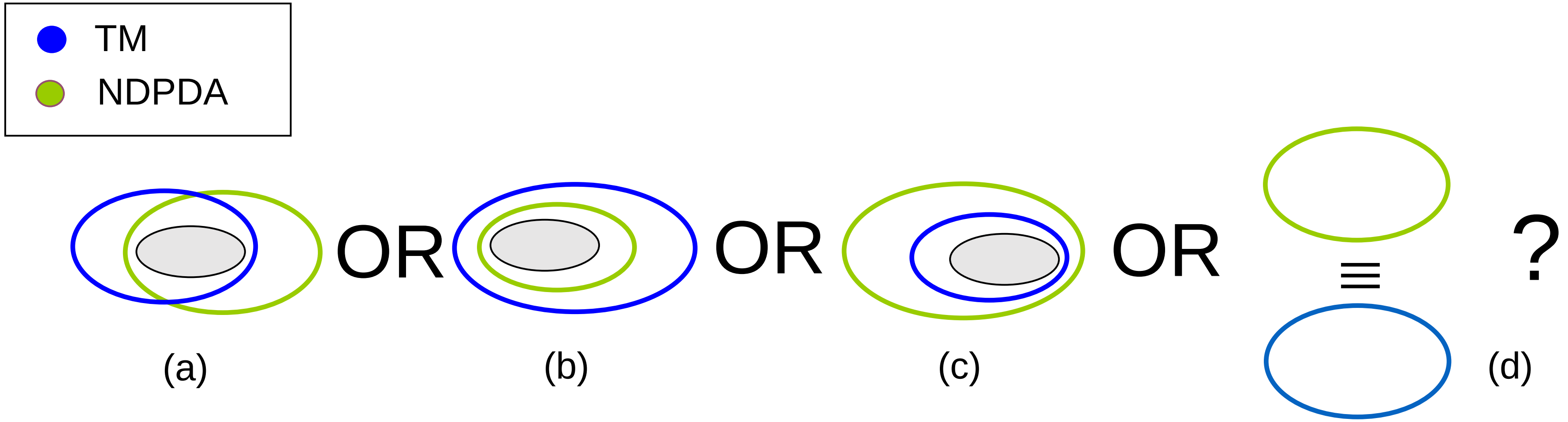


NDPDAs vs DPDAs (2)



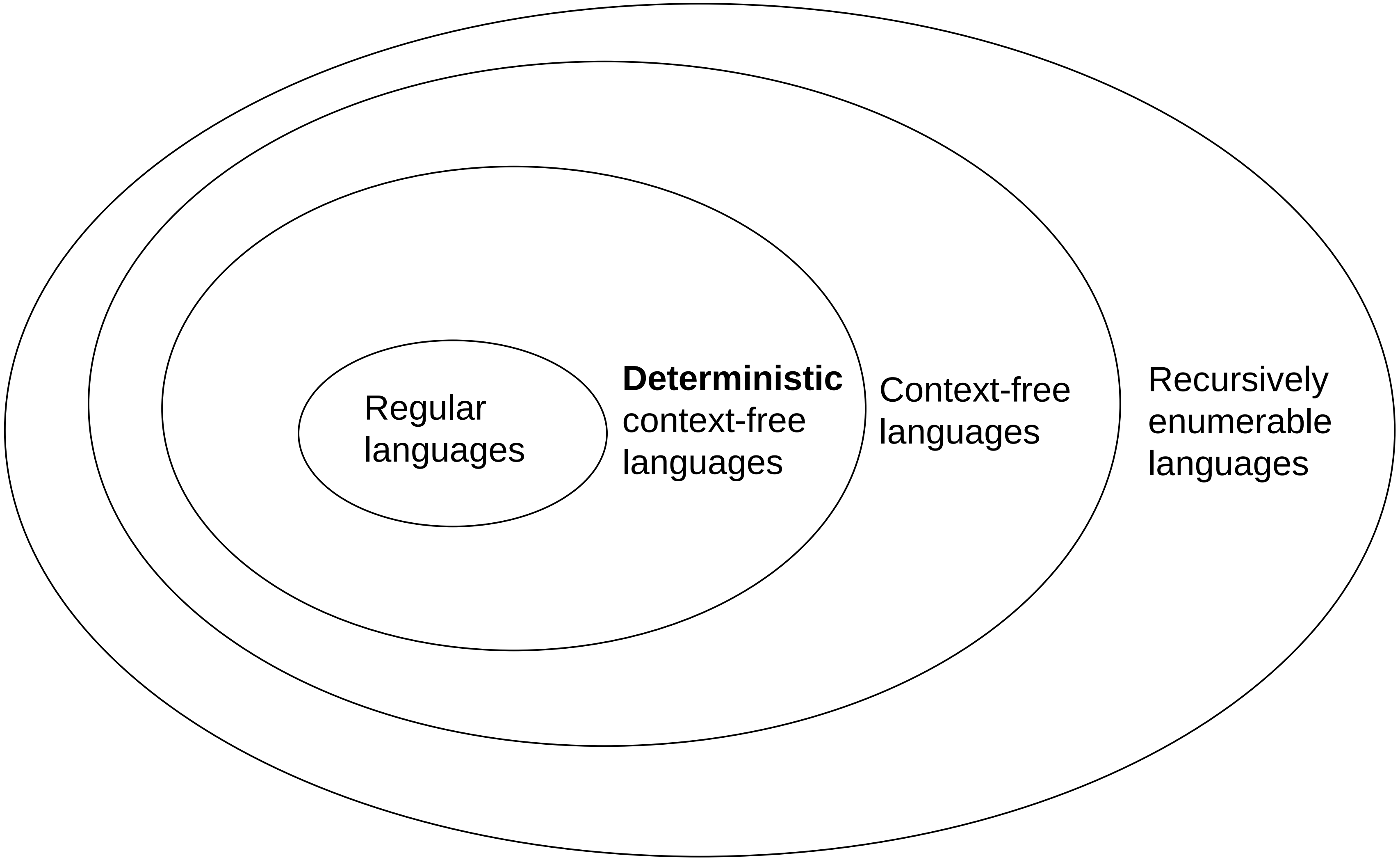
Languages recognizable by NDPDAs are called <u>context-free languages</u>

NDPDA vs TM



- (a) and (c): NO!
  - A (N)DTM can simulate a NDPDA by using the tape as a stack
- (d): NO!
  - **$\{a^n b^n \mid n \geq 1\} \cup \{a^n b^{2n} \mid n \geq 1\}$ is recognizable by both**

The bigger picture



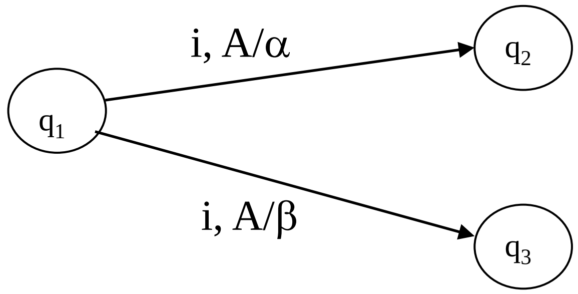
Closure properties in DPDAs

- In DPDAs we have
  - **Closure w.r.t. complement**
  - **Non-closure w.r.t. union, intersection, difference**

- Does changing the power of the automata change their behavior w.r.t. operations?

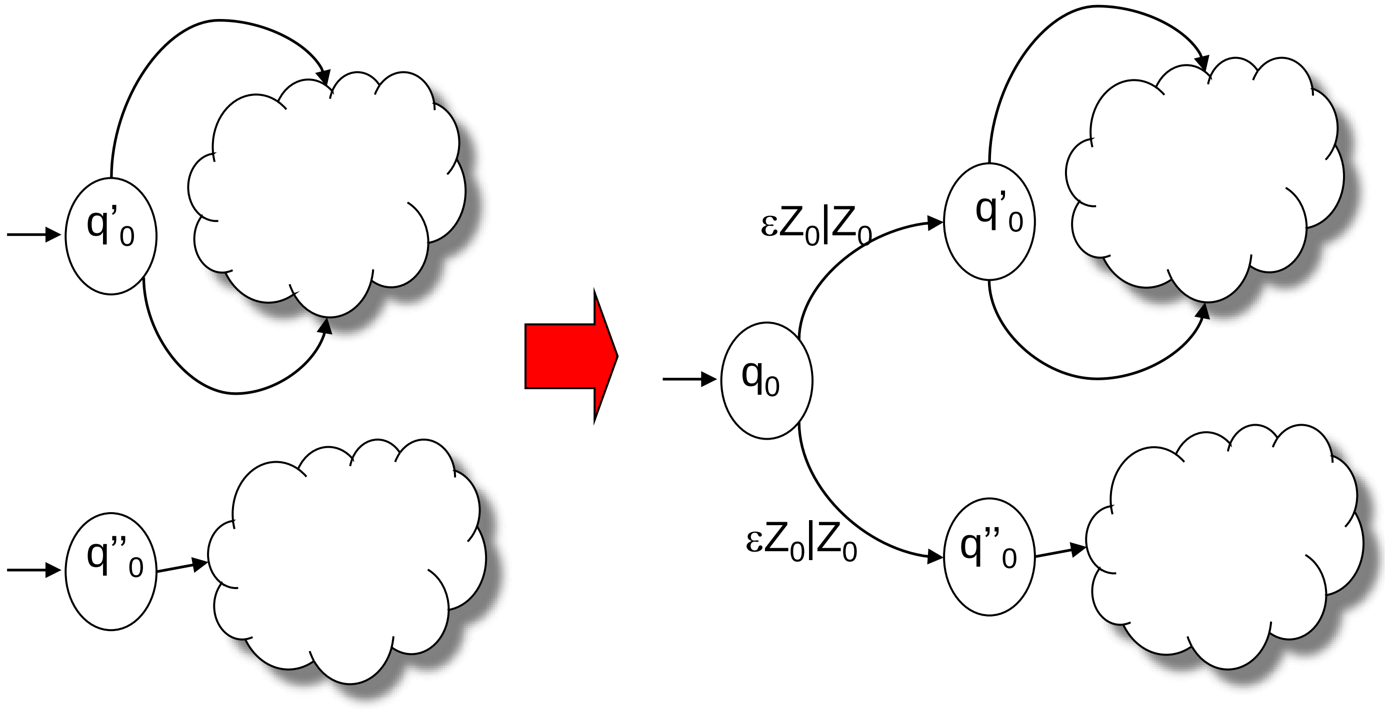
Union (1)

- **NDPDAs are <u>closed</u> under union**
  - Intuition:



- Given two NDPDAs, P_1 and P_2, we can always build a NDPDA that represents the union by creating a new initial state that is connected to both initial states of P_1 and P_2 with an ε-move

Union (2)



Intersection

- **The closure w.r.t. intersection still does <u>not</u> hold**
- Consider
  - $\{a^n b^n c^*\}$
  - $\{a^* b^n c^n\}$

  both are recognizable by (N)DPDAs, but

  $\{a^n b^n c^*\} \cap \{a^* b^n c^n\} = \{a^n b^n c^n\}$ is not recognizable by any NDPDA

Complement (1)

- If a class of languages is closed w.r.t. union, but not w.r.t. intersection it cannot be closed w.r.t. complement
  - **We can write intersection in terms of union and complement**
- **NDPDAs are <u>not closed</u> w.r.t complement**

Remarks

- If a machine is deterministic and its computation terminates, the complement can be obtained by:
  - Completing the machine
  - Swapping accepting and non accepting states
- **Nondeterminism or infinite computation does not allow the application of this approach**

Complement (2)

- For NDPDAs, computations can always be made terminating (as for DPDAs)
- However, ND can cause this problem:

  One can have two computations:

  - $c_0 = \langle q_0, x, Z_0 \rangle \mid\!-^{*}\!- c_1 = \langle q_1, \varepsilon, \gamma \rangle$
  - $c_0 = \langle q_0, x, Z_0 \rangle \mid\!-^{*}\!- c_2 = \langle q_2, \varepsilon, \gamma \rangle$

  with $q_1 \in F$ and $q_2 \notin F$

$\rightarrow$ even if we swap accepting and non accepting state, x is still accepted

Theoretical Computer Science

Generative Grammars

Lecture 11 - Manuel Mazzara

Models for languages

Models suitable to **recognize/accept, translate, compute** languages

- They “receive” an input string and process it

→ **Operational models (Automata)**

Models suitable to **describe how to generate** a language

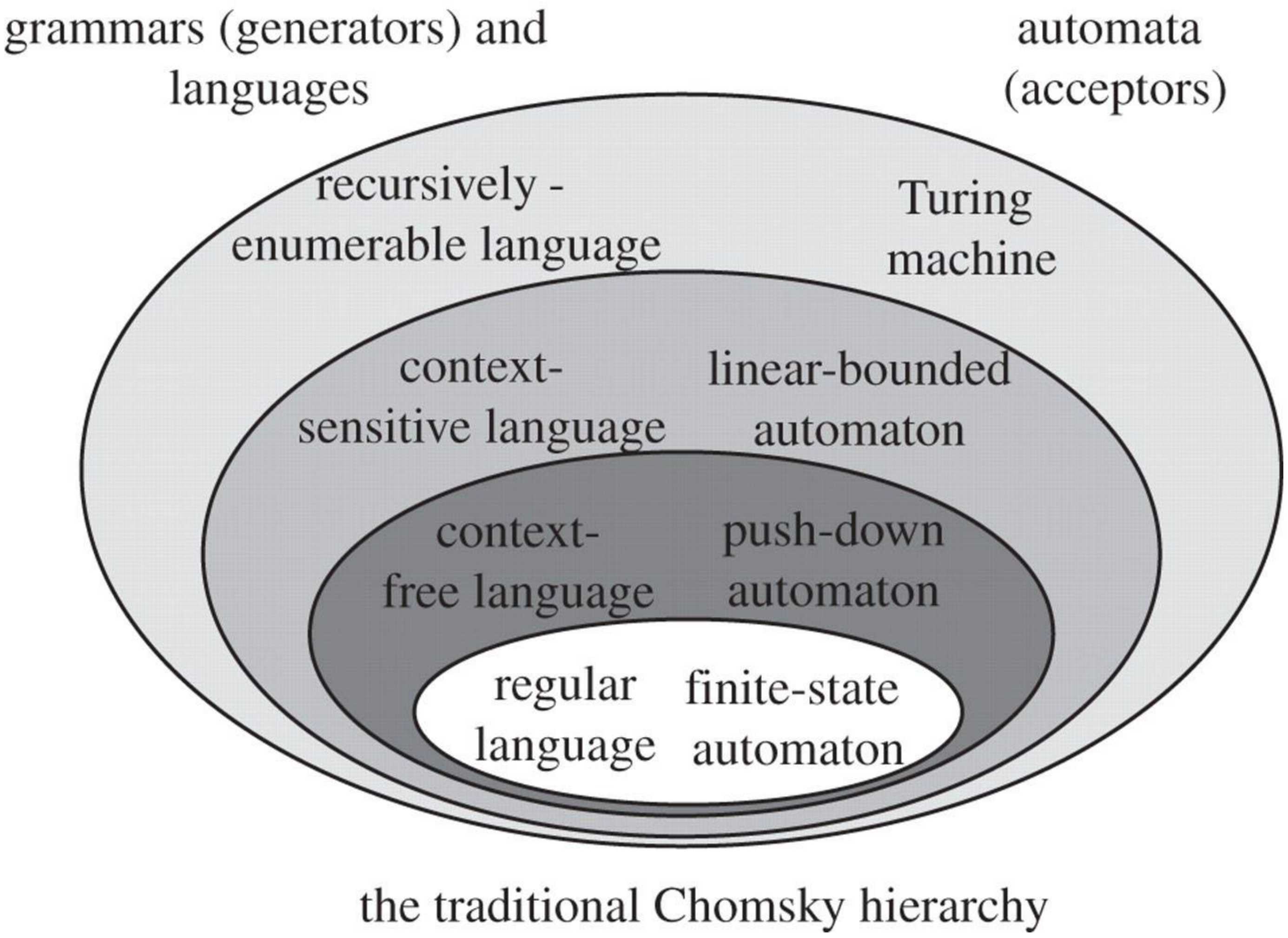
- Sets of rules to build phrases of a language

→ **Generative models (Grammars)**

Automata, languages, and grammars

Chomsky hierarchy	Grammars	Languages	Minimal automaton
Type-0	Unrestricted	Recursively enumerable	Turing machine
Type-1	Context-sensitive	Context-sensitive	**(Linear bounded automaton)**
Type-2	Context-free	Context-free	NDPDA
Type-3	Regular	Regular	FSA

Generators vs acceptors



Grammars (1)

- **Generative models** produce strings
  - grammar (or syntax)
- **A grammar is a set of rules** to build the phrases of a language
  - It applies to any notion of language (natural, artificial…)
- **A formal grammar** generates strings of a language through a rewriting process

Grammars (2)

- **A grammar is a set of linguistic rules**
- It is composed by
  - a main object: **initial symbol**
  - composing objects: **nonterminal symbols**
  - base elements: **terminal symbols**
  - refinement rules: **productions**

- We will see the **formalization**

Rewriting

- **Rewriting** relevant to many fields
  - Mathematics
  - Computer science
  - Logic
- It consists of a wide range of methods for **replacing subterms** of a “formula” with other terms
  - **Potentially nondeterministic**
  - **Remember NPDAs and importance for parsing!**

Linguistic rules (1)

- **Natural languages** are explained through rules such as:
  - A phrase is made of a **subject followed by a predicate**
  - A subject can be a **noun** or a **pronoun** or…
  - A predicate can be a **verb followed by a complement**

- **Programming languages** are expressed similarly:
  - A program consists of a **declarative part** and an **executable part**
  - The declarative part …
  - The executable part consists of a **statement sequence**
  - A statement can be …

Linguistic rules (2)

- In general, a **linguistic rule** describes a "**main object**"
  - Examples: a program, a message, …

  as a sequence of "**composing objects**"

- Each "composing object" is **refined** by **replacing/rewriting** it with more detailed objects until a sequence of **base elements** (that cannot be further refined) is obtained

Definition

- A grammar is a tuple **$<V_N, V_T, P, S>$** where
  - V_N is the **<u>nonterminal alphabet</u>** (or vocabulary)
  - V_T is the **<u>terminal alphabet</u>** (or vocabulary)
  - $V=V_N \cup V_T$
  - $S \in V_N$ is a particular element of V_N called <u>axiom</u> or **<u>initial symbol</u>**
  - $P \subseteq V^* \cdot V_N \cdot V^* \times V^*$ is the (finite) set of <u>rewriting rules</u> or **<u>productions</u>**

- **A grammar $G=<V_N, V_T, P, S>$ generates a language on the alphabet V_T**

Productions

- A **production** is an element of $\mathbf{V^* \cdot V_N \cdot V^* \times V^*}$
  - This is usually denoted as

    $\mathbf{<\alpha, \beta>}$ where

    $\alpha \in \mathbf{V^* \cdot V_N \cdot V^*}$ and $\beta \in \mathbf{V^*}$
- We generally indicate a production as $\alpha \rightarrow \beta$
  - **α is a sequence of symbols including <u>at least one nonterminal symbol</u>**
  - **β is a (potentially empty) sequence of (terminal or non terminal) symbols**

Immediate derivation relation

$\boldsymbol{\alpha \Rightarrow \beta}$ (β is obtained by immediate derivation from α)

– $\alpha \in V^* \cdot V_N \cdot V^*$ and $\beta \in V^*$

if and only if

$\alpha = \alpha_1\alpha_2\alpha_3$, $\beta = \alpha_1\beta_2\alpha_3$ and $\alpha_2 \rightarrow \beta_2 \in P$

$\rightarrow$ α_2 is rewritten as β_2 in the **context** $<\alpha_1, \alpha_3>$

And finally, the word "context" appears!

Example of derivations (1)

In the grammar G

- $V_N = \{S, A, B, C, D\}$
- $V_T = \{a,b,c\}$
- S is the initial symbol
- $P = \{S \rightarrow AB,\ BA \rightarrow cCD,\ CBS \rightarrow ab,\ A \rightarrow \varepsilon\}$

- $aa\underline{\mathbf{BA}}S \Rightarrow aa\underline{\mathbf{cCD}}S$
- $bc\underline{\mathbf{CBS}}Add \Rightarrow bc\underline{\mathbf{ab}}Add$

Example of derivations (2)

- $G=\langle\{S,A,B, C, D\}, \{a,b,c\}, P, S\rangle$
  - $P=\{S\rightarrow aACD,\ A\rightarrow aAC|\varepsilon,\ B\rightarrow b,\ CD\rightarrow BDc,\ CB\rightarrow BC,\ D\rightarrow\varepsilon\}$
- Some derivations
  - $\mathbf{S} \Rightarrow a\mathbf{A}CD \Rightarrow a\mathbf{CD} \Rightarrow a\mathbf{B}Dc \Rightarrow ab\mathbf{D}c \Rightarrow abc$
  - $S \Rightarrow aACD \Rightarrow aaACCD \Rightarrow aaCBDc \Rightarrow aaBCDc \Rightarrow aabCDc \Rightarrow aabBDcc \Rightarrow aabbDcc \Rightarrow aabbcc$
  - $S \Rightarrow aACD \Rightarrow aaACCD \Rightarrow aaCCD \Rightarrow aaCC$

Language generated by a grammar

- Given a grammar **G=$\langle V_N, V_T, P, S \rangle$**
- **$L(G)=\{x \mid x \in V_T^* \wedge S \Rightarrow^+ x\}$**
- Informally the language generated by a grammar G is **the set of all strings**
  - **Consisting only of terminal symbols**

  that can be **derived from S**
  - **In any number of steps**

Examples (1)

- V_N = {S} - usually *capital symbols are used*
- V_T = {a,b} - usually *non-capital symbols are used*
- S is the initial symbol
  - It is not mandatory to call it S

- $P = \{ S \rightarrow aSb,$
  $S \rightarrow ba,$
  $\}$

<u>What is the generated language on the alphabet {a,b}?</u>

Examples (2)

$$\{a^n bab^n | n \geq 0\} = \{ba, abab, aababb, aaababbb, \ldots\}$$

What if we change the production set as follows?

P = { S $\rightarrow$ aSb,
S $\rightarrow$ ab,
}

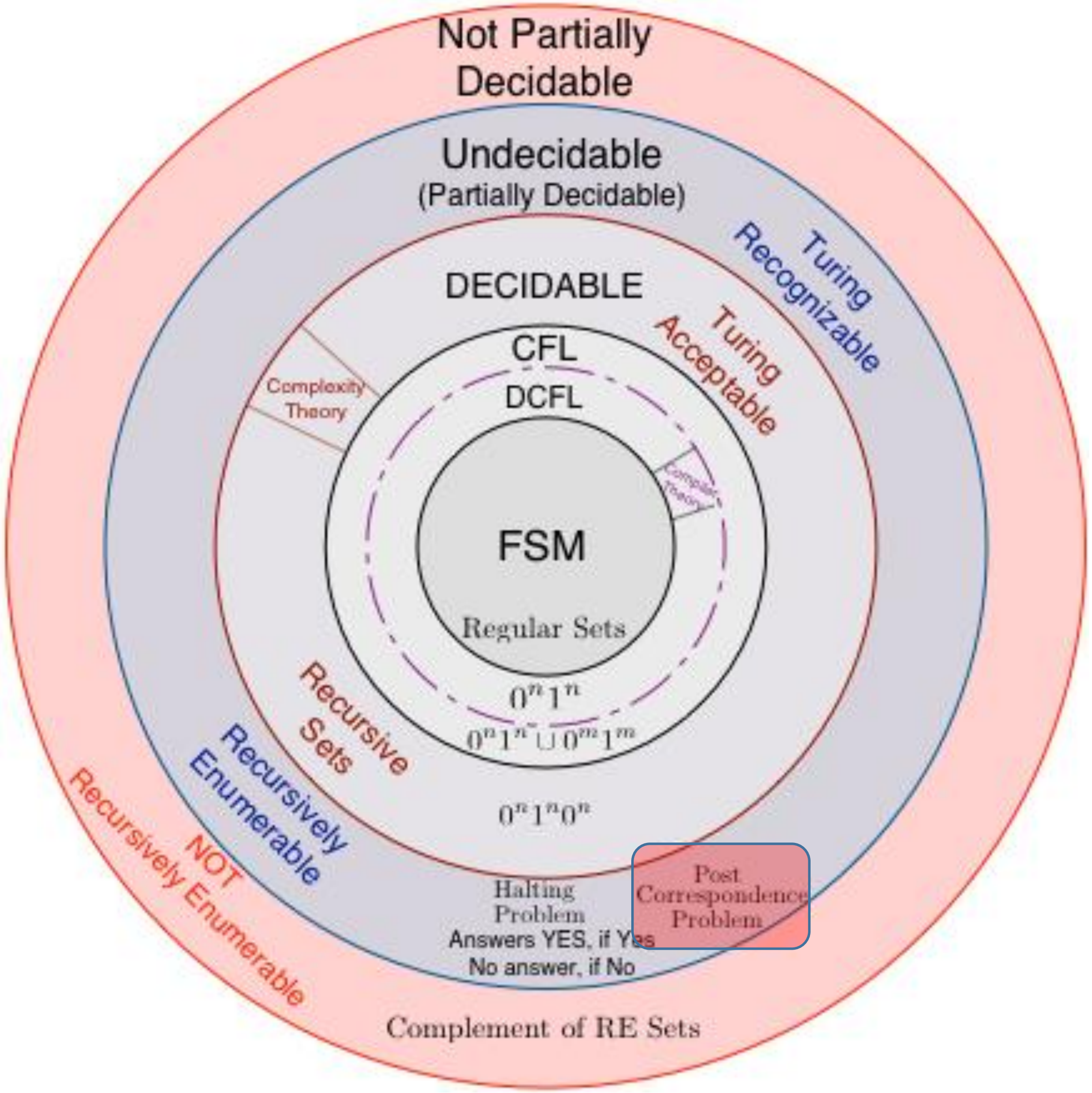
Do you recognize a friend?

Examples (3)

- $G=\langle\{S\}, \{a,b\}, \{S\rightarrow aSb|ab\}, S\rangle$
  - $\{S\rightarrow aSb|ab\}$ is an abbreviation for $\{S\rightarrow aSb, S\rightarrow ab\}$
- Some derivations
  - $S \Rightarrow ab$
  - $S \Rightarrow aSb \Rightarrow aabb$
  - $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaabbb$
- An easy generalization **$L(G)=\{a^nb^n|n>0\}$**
  - $L(G)=\{a^nb^n|n\geq 0\}$ if we substitute $S\rightarrow ab$ with $S\rightarrow \varepsilon$

Examples (4)

- G=<{S,A,B}, {a,b,0}, P, S>
  - P={S$\rightarrow$aA, A$\rightarrow$aS, S$\rightarrow$bB, B$\rightarrow$bS, S$\rightarrow$0}
- Some derivations
  - S $\Rightarrow$ 0
  - S $\Rightarrow$ aA $\Rightarrow$ aaS $\Rightarrow$ aa0
  - S $\Rightarrow$ bB $\Rightarrow$ bbS $\Rightarrow$ bb0
  - S $\Rightarrow$ aA $\Rightarrow$ aaS $\Rightarrow$ aabB $\Rightarrow$ aabbS $\Rightarrow$ aabb0
- An easy generalization **L(G)={aa, bb}*.0**


Not Partially Decidable
Undecidable
(Partially Decidable)
Turing Recognizable
DECIDABLE
Turing Acceptable
CFL
DCFL
Complexity Theory
FSM
Regular Sets
0^n1^n
$0^n1^n \cup 0^m1^m$
Recursive Sets
Recursively Enumerable
$0^n1^n0^n$
NOT Recursively Enumerable
Halting Problem
Answers YES, if Yes
No answer, if No
Post Correspondence Problem
Complement of RE Sets