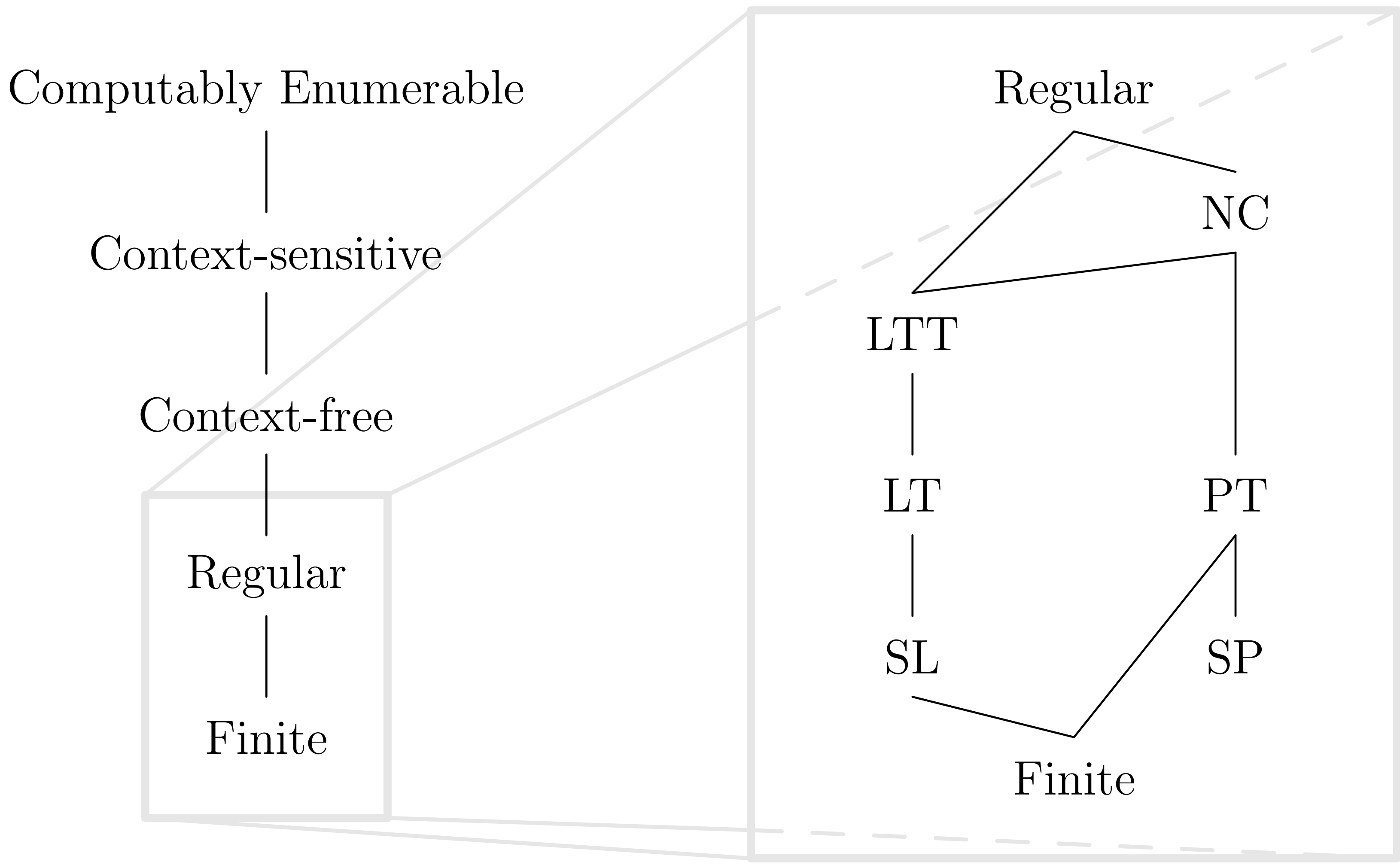
Subregular Complexity

Jeffrey Heinz

jeffrey.heinz@stonybrook.edu

Computational Complexity



Part I

Some Motivation

Some linguistics as motivation

ptak thole hlad plast sram mgla vlas flitch dnom rtut

Halle, M. 1978. In *Linguistic Theory and Pyschological Reality*. MIT Press.

Phonotactics

possible English words	impossible English words
thole	ptak
plast	hlad
flitch	sram
	mgla
	vlas
	dnom
	rtut

Phonotactics

possible English words	impossible English words
thole	ptak
plast	hlad
flitch	sram
	mgla
	vlas
	dnom
	rtut

This is knowledge English speakers have learned, but were not taught.

Phonotactics - Samala Version

ʃtojonowonowaʃ

stojonowonowaʃ

stojonowonowas

ʃtojonowonowas

pisotonosikiwat

pisotonoʃikiwat

sanisotonosikiwas

ʃanipisotonoʃikiwas

Phonotactics - Samala Version

possible Samala words	impossible Samala words
ʃtojonowonowaʃ	stojonowonowaʃ
stojonowonowas	ʃtojonowonowas
pistonoskiwat	pisotonoʃikiwat
sanisotonoskiwas	ʃanipisotonoʃikiwas

1. How do Samala speakers know which of these words belong to different columns? How did they acquire this knowledge?
2. By the way, *ʃtoyonowonowaʃ* means ‘it stood upright’ (Applegate 1972)

Phonotactics - Samala Version

possible Samala words	impossible Samala words
ʃtojonowonowaʃ	stojonowonowaʃ
stojonowonowas	ʃtojonowonowas
pistonoskiwat	pisotonoʃikiwat
sanisotonoskiwas	ʃanipisotonoʃikiwas

1. How do Samala speakers know which of these words belong to different columns? How did they acquire this knowledge?

2. By the way, *ʃtoyonowonowaʃ* means 'it stood upright' (Applegate 1972)

Exercise

1. Write a DFA or regular expression for the language of those strings which do not begin with `pt`. Assume an alphabet {`p`, `t`, `k`, `o`}.
2. Write a DFA or regular expression for the language of those strings which do not contain both `s` and `S`. Assume an alphabet {`s`, `S`, `t`, `o`}.

My own interests in subregular complexity...

...began with these observations:

1. Many phonotactic patterns are regular.
2. Many regular patterns are *not* possible phonotactic patterns.

So *which* regular languages constitute possible phonotactic patterns?

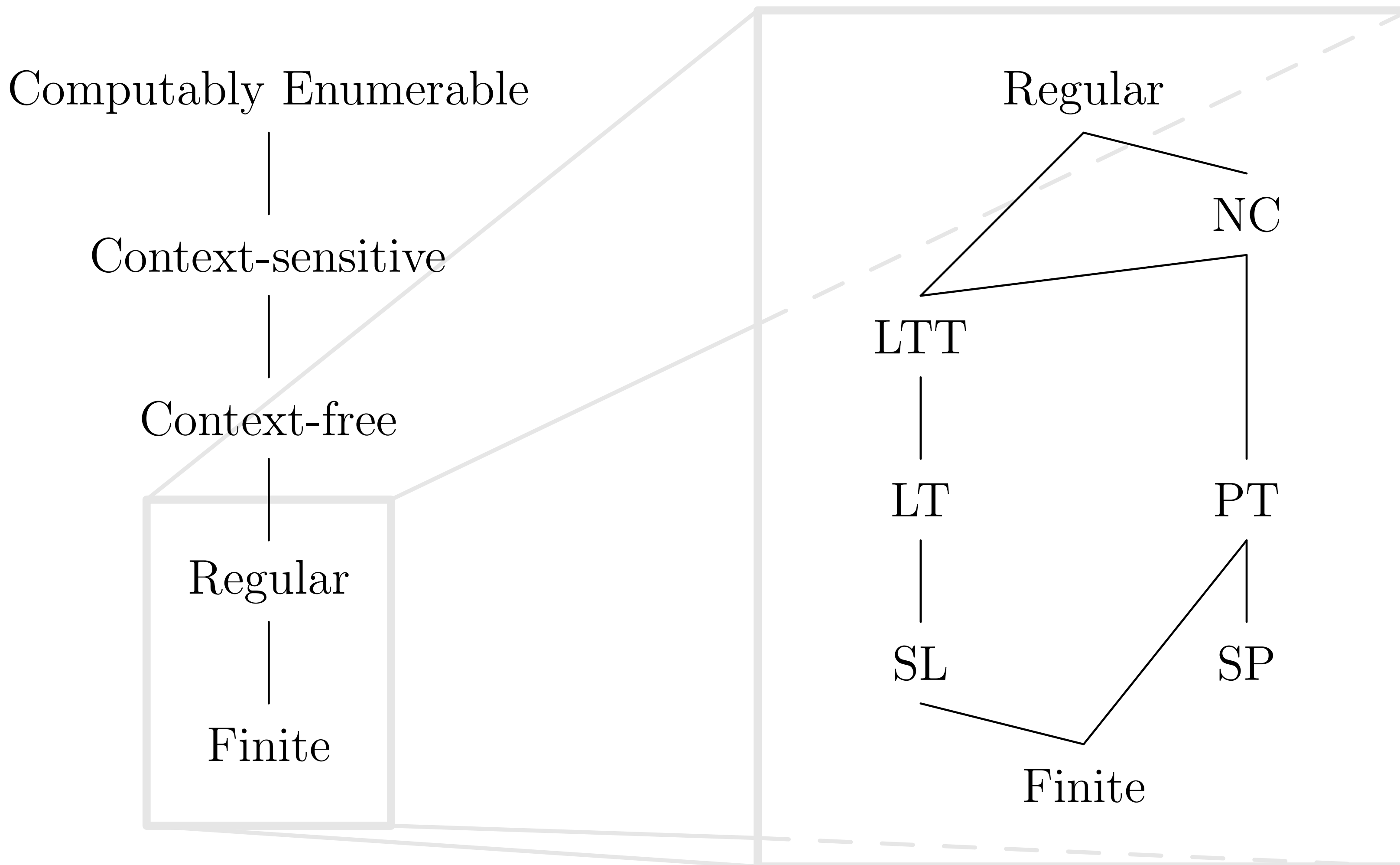
Along the way I learned:

1. There is a rich field of study in computer science on *subregular* complexity.
2. There is a rich field of study in computer science on *machine learning* of regular languages and transductions.
3. There are many applications in linguistics, artificial intelligence, robotic planning and control, model checking, ...

Part II

Measuring Complexity

Computational Complexity



Some hypotheses

Some ways to measure complexity

A regular language L_1 is *more complex* than L_2 if

1. The smallest DFA recognizing L_1 is larger than the smallest DFA recognizing L_2.
2. The smallest regular expression recognizing L_1 is larger than the smallest regular expression recognizing L_2.

In contrast to these *intensional* measures

1. The subregular classes we study today will provide complexity measures *independent* of the size of such representations.

Some simple languages (let $\Sigma = \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}\}$)

(1) Strings end with a `b`.

(2) The second to last symbol in all strings is `b`.

(3) The third to last symbol in all strings is `b`.

Some simple languages (let Σ={a,b,c,d})

(4) Strings contain at least one b.

(5) Strings contain at most one b.

(6) Strings contain exactly one b.

Some simple languages (let Σ={`a`,`b`,`c`,`d`})

(7) Strings contain at least one `bb` substring.

(8) Strings contain at least two `b`s.

(9) Strings contain at least two `bb` substrings.

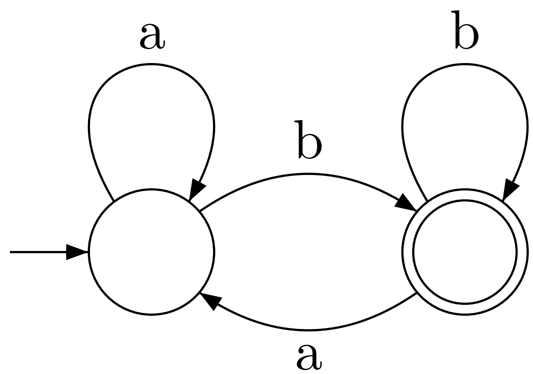
Some simple languages (let Σ={a,b,c,d})

(10) Strings contain an a between every pair of bs.

(11) Strings contain an odd number of bs.

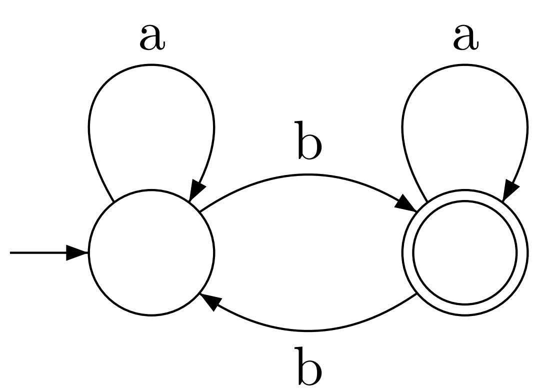
Comparing “final-b” with “odd-b”

Strings end with `b`.



`(a + b)* b`

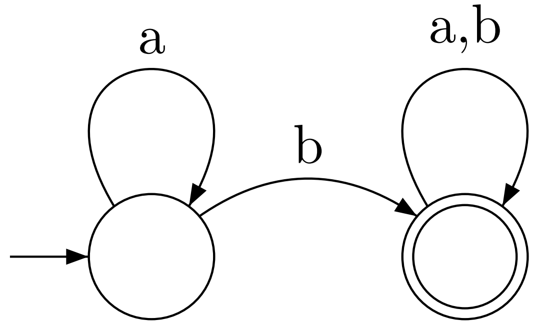
Strings have odd many `b`s.



`a* ba* (a* ba* ba* )*`

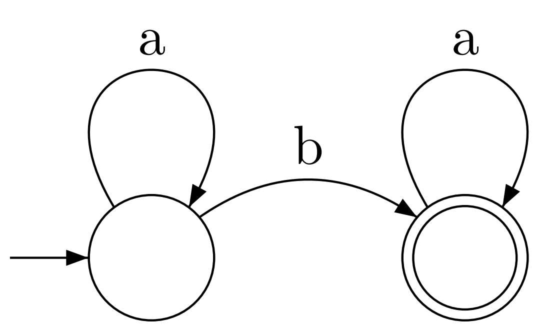
Comparing "at least one b" with "exactly one b"

Strings with at least one `b`.



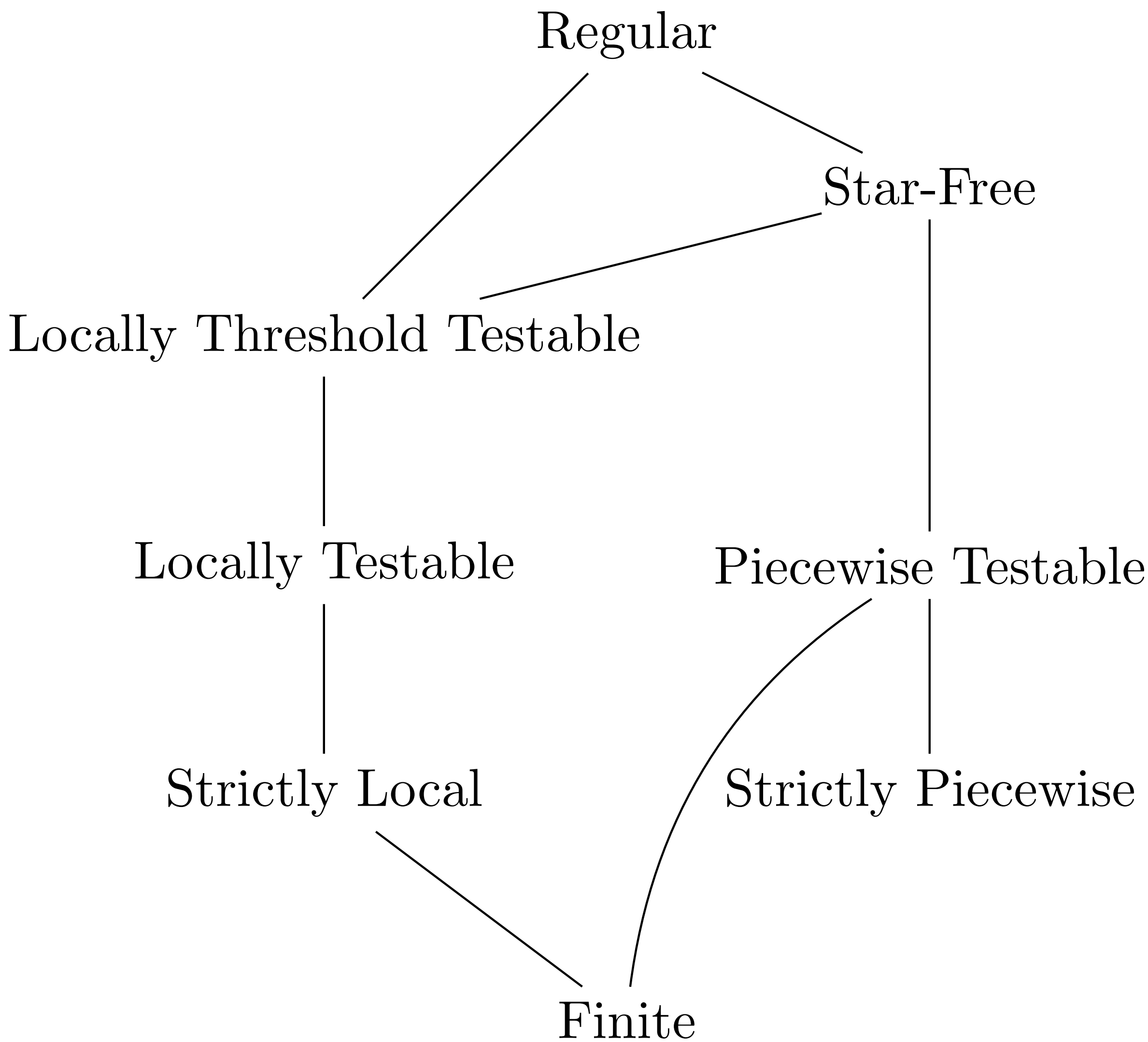
`a* b(a+b)*`

Strings with exactly one `b`.

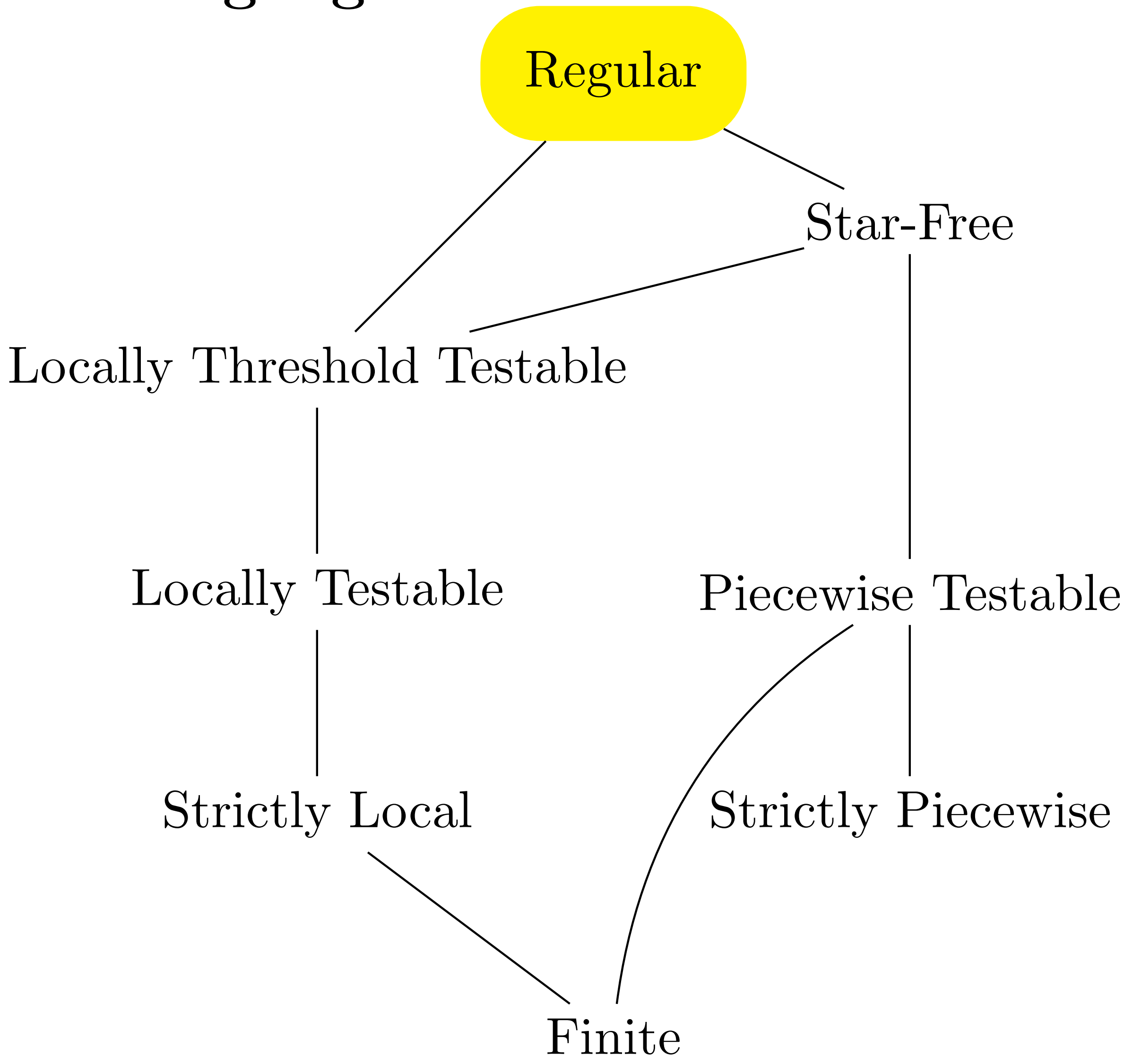


`a* ba*`

Subregular Complexity



Regular Languages



Regular Languages

Myhill/Nerode Theorem

- For all $L \subseteq \Sigma^*, u \in \Sigma^*$, let $T_L(u) = \{v \mid uv \in L\}$.
- L is regular iff $\big|\{T_L(u) \mid u \in \Sigma^*\}\big|$ is finite.

Theorem

- L is regular iff $L \in [\![\mathrm{DFA}]\!] = [\![\mathrm{NFA}]\!] = [\![\mathrm{RE}]\!] = [\![\mathrm{GRE}]\!]$
$= [\![\mathrm{MSO}(+1)]\!] = [\![\mathrm{MSO}(<)]\!]$

Characterizing Regular Languages

Myhill/Nerode Theorem

- For all $L \subseteq \Sigma^*, u \in \Sigma^*$, let $T_L(u) = \{v \mid uv \in L\}$.
- L is regular iff $\left|\{T_L(u) \mid u \in \Sigma^*\}\right|$ is finite.

Theorems

- L is regular iff $L \in \boxed{[\![\mathrm{DFA}]\!] = [\![\mathrm{NFA}]\!]} = [\![\mathrm{RE}]\!] = [\![\mathrm{GRE}]\!]$ $= [\![\mathrm{MSO}(+1)]\!] = [\![\mathrm{MSO}(<)]\!]$

Deterministic and Non-deterministic Finite-state Acceptors

Characterizing Regular Languages

Myhill/Nerode Theorem

- For all $L \subseteq \Sigma^*, u \in \Sigma^*$, let $T_L(u) = \{v \mid uv \in L\}$.
- L is regular iff $\left|\{T_L(u) \mid u \in \Sigma^*\}\right|$ is finite.

Theorems

- L is regular iff $L \in [\![\mathrm{DFA}]\!] = [\![\mathrm{NFA}]\!] = \boxed{[\![\mathrm{RE}]\!] = [\![\mathrm{GRE}]\!]}$ $= [\![\mathrm{MSO}(+1)]\!] = [\![\mathrm{MSO}(<)]\!]$

Regular Expressions and Generalized Regular Expressions

Characterizing Regular Languages

Myhill/Nerode Theorem

- For all $L \subseteq \Sigma^*, u \in \Sigma^*$, let $T_L(u) = \{v \mid uv \in L\}$.
- L is regular iff $\left|\{T_L(u) \mid u \in \Sigma^*\}\right|$ is finite.

Theorems

- L is regular iff $L \in [\![\mathrm{DFA}]\!] = [\![\mathrm{NFA}]\!] = [\![\mathrm{RE}]\!] = [\![\mathrm{GRE}]\!]$
  $= \boxed{[\![\mathrm{MSO}(+1)]\!] = [\![\mathrm{MSO}(<)]\!]}$

Monadic Second Order logic with successor (+1) and precedence (<)

Regular Expressions

Syntax

REs include

- each $\sigma \in \Sigma$
- ϵ
- $\varnothing$

If R and S are REs then so are

- (R·S) *(concatenation)*
- (R+S) *(union)*
- (R^*) *(Kleene star)*

Semantics

- $[\![\sigma]\!] = \{\sigma\}$
- $[\![\epsilon]\!] = \{\epsilon\}$
- $[\![\varnothing]\!] = \{\}$

- $[\![(R{\cdot}S)]\!] = [\![R]\!] \cdot [\![S]\!]$
- $[\![(R{+}S)]\!] = [\![R]\!] \cup [\![S]\!]$
- $[\![(R^*)]\!] = [\![R]\!]^*$

Generalized Regular Expressions

Syntax

GREs include

- each $\sigma \in \Sigma$
- ϵ
- $\varnothing$

If R and S are GREs then so are

- $(\mathrm{R}{\cdot}\mathrm{S})$ *(concatenation)*
- $(\mathrm{R}{+}\mathrm{S})$ *(union)*
- (R^{*}) *(Kleene star)*
- $(\mathrm{R}\&\mathrm{S})$ *(intersection)*
- $(\overline{\mathrm{R}})$ *(complement)*

Semantics

- $[\![\sigma]\!] = \{\sigma\}$
- $[\![\epsilon]\!] = \{\epsilon\}$
- $[\![\varnothing]\!] = \{\}$

- $[\![(\mathrm{R}{\cdot}\mathrm{S})]\!] = [\![\mathrm{R}]\!] \cdot [\![\mathrm{S}]\!]$
- $[\![(\mathrm{R}{+}\mathrm{S})]\!] = [\![\mathrm{R}]\!] \cup [\![\mathrm{S}]\!]$
- $[\![(\mathrm{R}^{*})]\!] = [\![\mathrm{R}]\!]^{*}$
- $[\![(\mathrm{R}\&\mathrm{S})]\!] = [\![\mathrm{R}]\!] \cap [\![\mathrm{S}]\!]$
- $[\![\overline{\mathrm{R}}]\!] = \Sigma^{*} - [\![\mathrm{R}]\!]$

Generalized Regular Expressions

Syntax

GREs include

- each $\sigma \in \Sigma$
- ϵ
- $\varnothing$

If R and S are GREs then so are

- $(\mathrm{R}\cdot\mathrm{S})$ *(concatenation)*
- $(\mathrm{R}+\mathrm{S})$ *(union)*
- (R^*) *(Kleene star)*
- $(\mathrm{R}\&\mathrm{S})$ *(intersection)*
- $(\overline{\mathrm{R}})$ *(complement)*

Semantics

- $[\![\sigma]\!] = \{\sigma\}$
- $[\![\epsilon]\!] = \{\epsilon\}$
- $[\![\varnothing]\!] = \{\}$

- $[\![(\mathrm{R}\cdot\mathrm{S})]\!] = [\![\mathrm{R}]\!]\cdot[\![\mathrm{S}]\!]$
- $[\![(\mathrm{R}+\mathrm{S})]\!] = [\![\mathrm{R}]\!]\cup[\![\mathrm{S}]\!]$
- $[\![(\mathrm{R}^*)]\!] = [\![\mathrm{R}]\!]^*$
- $[\![(\mathrm{R}\&\mathrm{S})]\!] = [\![\mathrm{R}]\!]\cap[\![\mathrm{S}]\!]$
- $[\![\overline{\mathrm{R}}]\!] = \Sigma^* - [\![\mathrm{R}]\!]$

Adding intersection and complement does not increase power of REs!

Cat-Union Expressions

Syntax

CUEs include

- each $\sigma \in \Sigma$
- ϵ
- $\varnothing$

If R and S are CUEs then so are

- (R·S) *(concatenation)*
- (R+S) *(union)*
- (R^*) *(Kleene star)*
- (R&S) *(intersection)*
- ($\overline{\mathrm{R}}$) *(complement)*

Semantics

- $[\![\sigma]\!] = \{\sigma\}$
- $[\![\epsilon]\!] = \{\epsilon\}$
- $[\![\varnothing]\!] = \{\}$

- $[\![(\mathrm{R}\cdot\mathrm{S})]\!] = [\![\mathrm{R}]\!]\cdot[\![\mathrm{S}]\!]$
- $[\![(\mathrm{R}+\mathrm{S})]\!] = [\![\mathrm{R}]\!]\cup[\![\mathrm{S}]\!]$
- $[\![(\mathrm{R}^*)]\!] = [\![\mathrm{R}]\!]^*$
- $[\![(\mathrm{R}\&\mathrm{S})]\!] = [\![\mathrm{R}]\!]\cap[\![\mathrm{S}]\!]$
- $[\![\overline{\mathrm{R}}]\!] = \Sigma^* - [\![\mathrm{R}]\!]$

Cat-Union Expressions

Syntax

CUEs include

- each $\sigma \in \Sigma$
- ϵ
- $\varnothing$

If R and S are CUEs then so are

- (R·S) *(concatenation)*
- (R+S) *(union)*
- (R^*) *(Kleene star)*
- (R&S) *(intersection)*
- $(\overline{R})$ *(complement)*

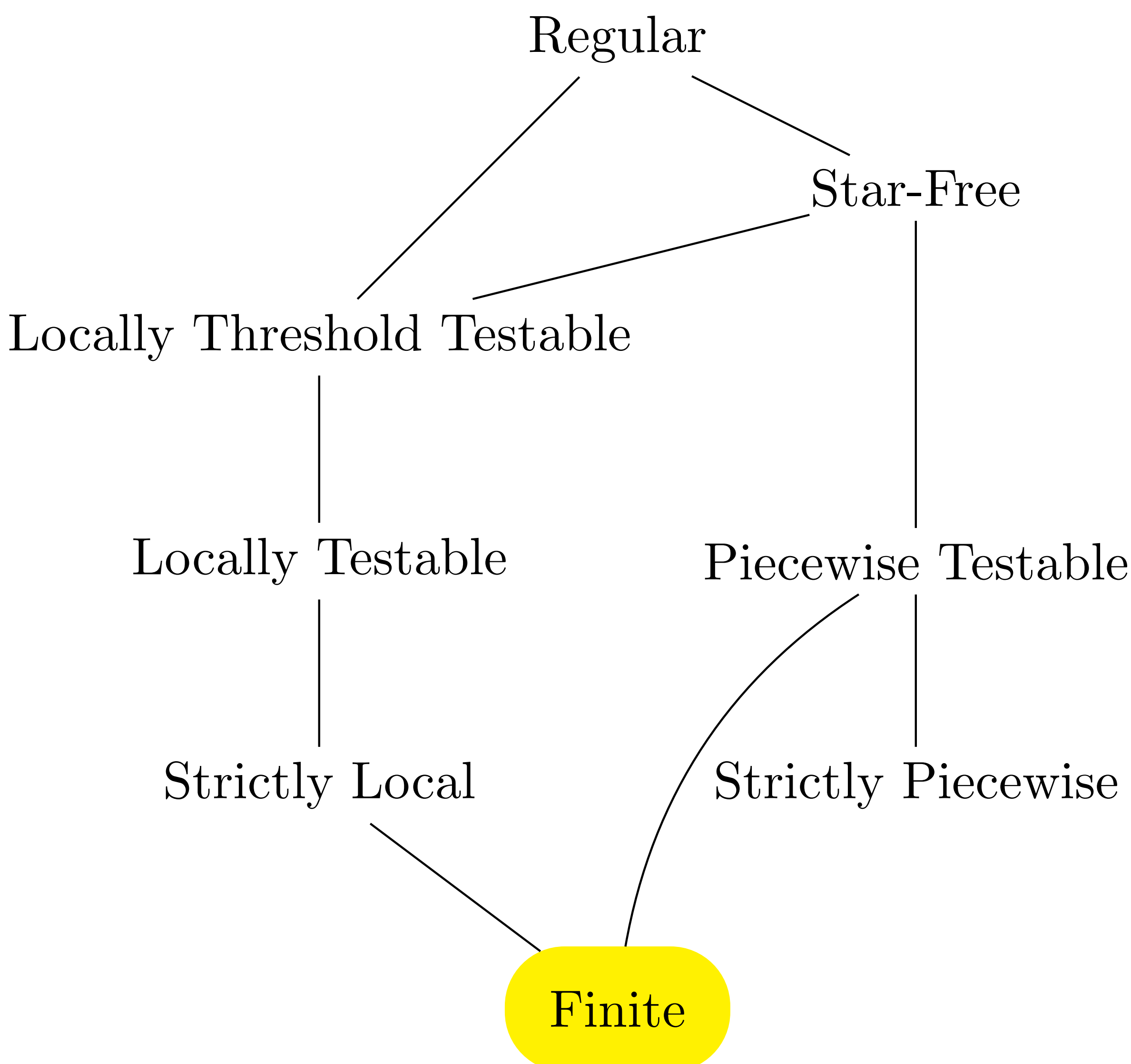
Semantics

- $[\![\sigma]\!] = \{\sigma\}$
- $[\![\epsilon]\!] = \{\epsilon\}$
- $[\![\varnothing]\!] = \{\}$

- $[\![(R \cdot S)]\!] = [\![R]\!] \cdot [\![S]\!]$
- $[\![(R+S)]\!] = [\![R]\!] \cup [\![S]\!]$
- $[\![(R^*)]\!] = [\![R]\!]^*$
- $[\![(R\&S)]\!] = [\![R]\!] \cap [\![S]\!]$
- $[\![\overline{R}]\!] = \Sigma^* - [\![R]\!]$

Theorem: $[\![\text{CUE}]\!] = \{L \subseteq \Sigma^* \mid |L| \text{ is finite}\} \subsetneq [\![\text{RE}]\!] = [\![\text{GRE}]\!]$

Finite Languages



Star-Free Regular Expressions

Syntax		**Semantics**
SFEs include		
• each $\sigma \in \Sigma$		• $[\![\sigma]\!] = \{\sigma\}$
• ϵ		• $[\![\epsilon]\!] = \{\epsilon\}$
• $\varnothing$		• $[\![\varnothing]\!] = \{\}$
If R and S are SFEs then so are		
• (R·S)	*(concatenation)*	• $[\![(R{\cdot}S)]\!] = [\![R]\!] \cdot [\![S]\!]$
• (R+S)	*(union)*	• $[\![(R{+}S)]\!] = [\![R]\!] \cup [\![S]\!]$
• (R^*)	*(Kleene star)*	• $[\![(R^*)]\!] = [\![R]\!]^*$
• (R&S)	*(intersection)*	• $[\![(R\&S)]\!] = [\![R]\!] \cap [\![S]\!]$
• ($\overline{R}$)	*(complement)*	• $[\![\overline{R}]\!] = \Sigma^* - [\![R]\!]$

Star-Free Regular Expressions

Syntax

SFEs include

- each $\sigma \in \Sigma$
- ϵ
- $\varnothing$

If R and S are SFEs then so are

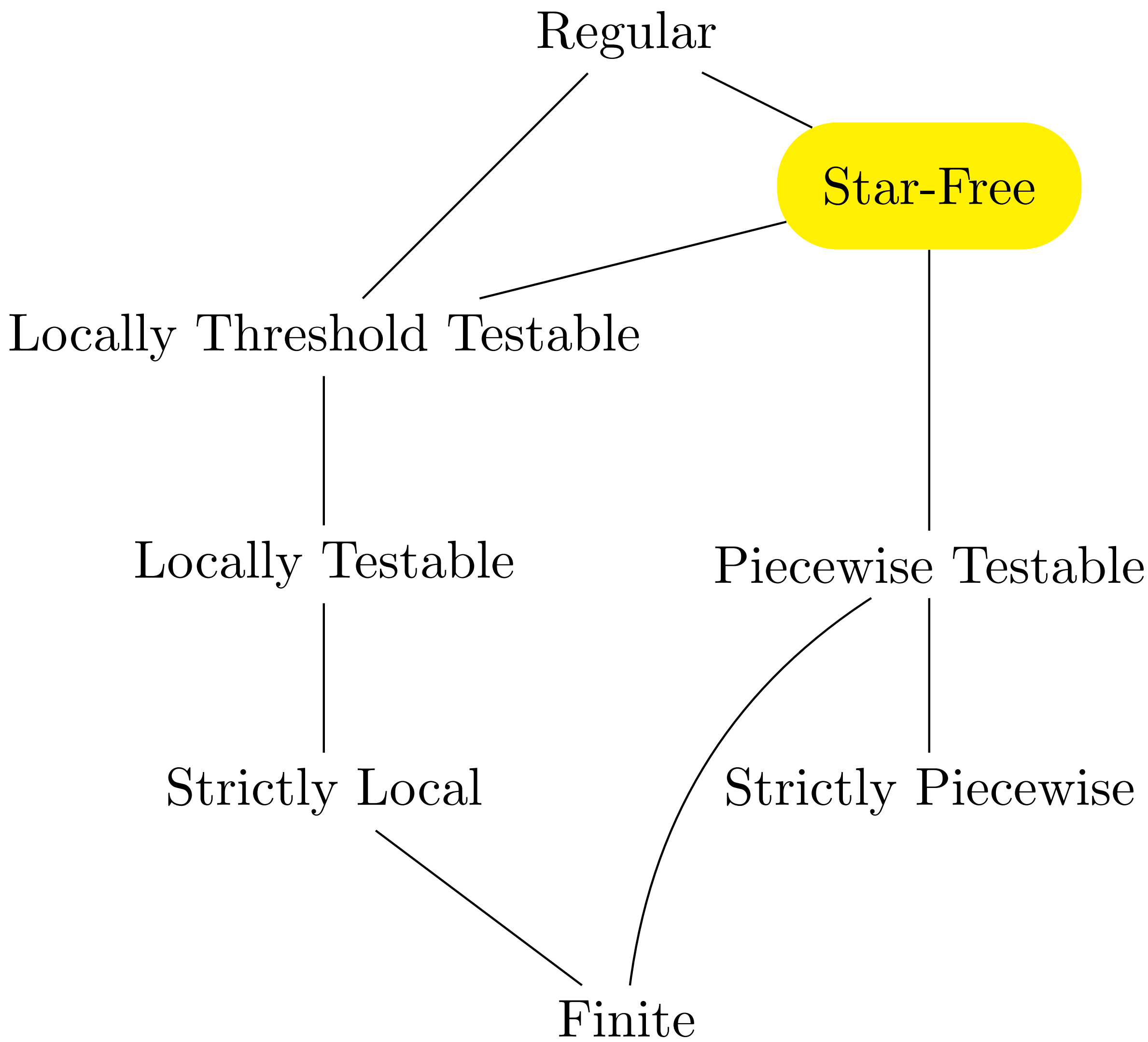
- $(\mathrm{R}\cdot\mathrm{S})$ *(concatenation)*
- $(\mathrm{R}+\mathrm{S})$ *(union)*
- (R^*) *(Kleene star)*
- $(\mathrm{R}\&\mathrm{S})$ *(intersection)*
- $(\overline{\mathrm{R}})$ *(complement)*

Semantics

- $[\![\sigma]\!] = \{\sigma\}$
- $[\![\epsilon]\!] = \{\epsilon\}$
- $[\![\varnothing]\!] = \{\}$
- $[\![(\mathrm{R}\cdot\mathrm{S})]\!] = [\![\mathrm{R}]\!]\cdot[\![\mathrm{S}]\!]$
- $[\![(\mathrm{R}+\mathrm{S})]\!] = [\![\mathrm{R}]\!]\cup[\![\mathrm{S}]\!]$
- $[\![(\mathrm{R}^*)]\!] = [\![\mathrm{R}]\!]^*$
- $[\![(\mathrm{R}\&\mathrm{S})]\!] = [\![\mathrm{R}]\!]\cap[\![\mathrm{S}]\!]$
- $[\![\overline{\mathrm{R}}]\!] = \Sigma^* - [\![\mathrm{R}]\!]$

Theorem: $[\![\mathrm{SFE}]\!] \subsetneq [\![\mathrm{RE}]\!] = [\![\mathrm{GRE}]\!]$

Star-free Languages



Expression Summary

Finite Languages	Star-Free Languages	Regular Languages
concatenation	concatenation	concatenation
union	union	union
	complement	**Kleene star**
	(intersection)	(complement)
		(intersection)

Expressivity $\longrightarrow$

Exercise

Write Star-Free Expressions for the following languages.
Let $\Sigma = \{\texttt{a, b}\}$.

1. Σ^*.
2. Strings which end with a `b`.
3. Strings which contain a `bb`.
4. Strings which do not contain a `bb`.
5. Strings which contain two `b`s.
6. Strings which do not contain two `b`s.

Characterizing Star-Free Languages

Grammar-independent characterization

- $L \in [\![\text{SFE}]\!]$ iff there exists n such that for all $x, y, z \in \Sigma^*$ and $m > n$ if $xy^nz \in L$ then $xy^mz \in L$.

Theorems

- $L \in [\![\text{SFE}]\!]$ iff L is definable with First Order logic with precedence ($L \in [\![\text{FO}(<)]\!]$).
- L is Star-Free iff the syntactic monoid of its DFA is *aperiodic*.

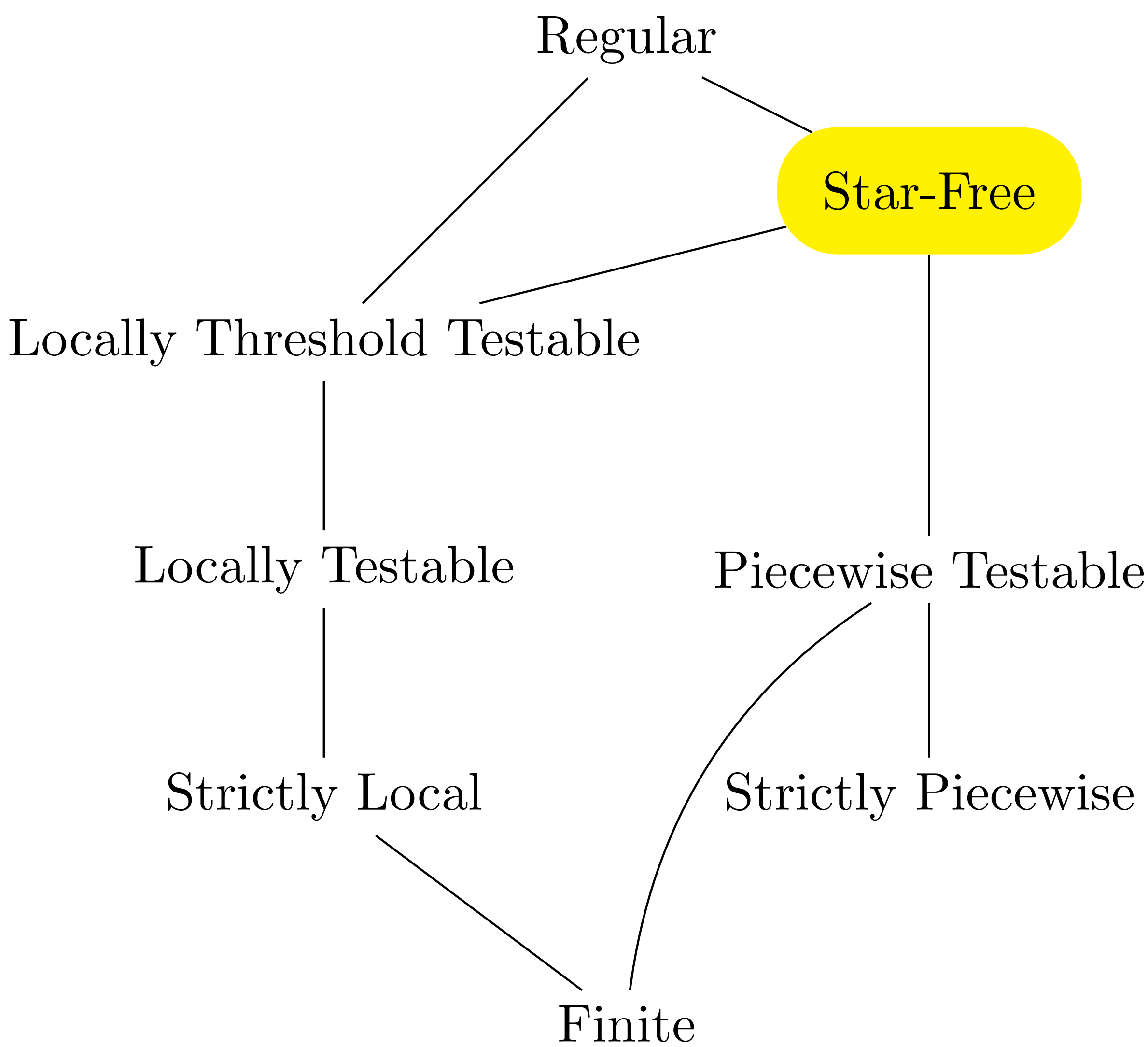
(McNaughton and Papert 1971)

Exercise

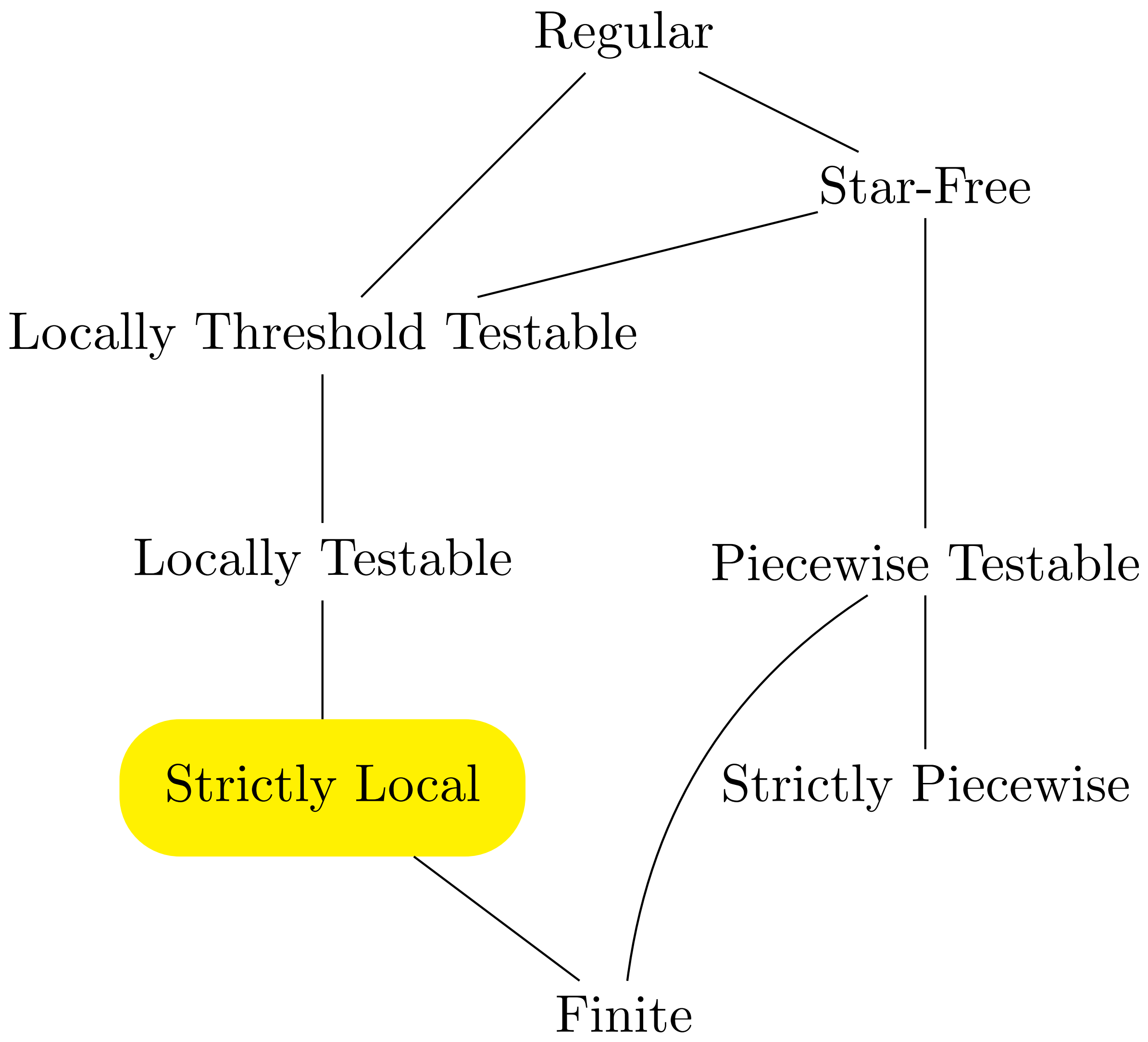
Prove the language below is not Star-Free.

(11) Strings contain an odd number of `b`s.

Star-free Languages



Strictly Local Languages



Strictly Local Languages

Intuitively, a language is SL if can be defined by forbidding finitely many strings from appearing at the beginnings, middles, and ends of words.

Key expressions

$w\overline{\varnothing}$	strings beginning with w
$\overline{\varnothing}w\overline{\varnothing}$	strings containing w
$\overline{\varnothing}w$	strings ending with w

Strictly Local Languages

Intuitively, a language is SL if can be defined by forbidding finitely many *substrings* from appearing at the beginnings, middles, and ends of words.

Key expressions

$\overline{w\overline{\varnothing}}$	strings *not* beginning with w
$\overline{\overline{\varnothing}w\overline{\varnothing}}$	strings *not* containing w
$\overline{\overline{\varnothing}w}$	strings *not* ending with w

Strictly Local Languages

A Formal Definition

A language L is Strictly Local if there are three, possibly empty, sets of strings

$$\begin{aligned} \mathrm{P} &= \{p_1, p_2, \ldots p_m\} \\ \mathrm{W} &= \{w_1, w_2, \ldots w_n\} \\ \mathrm{S} &= \{s_1, s_2, \ldots s_\ell\} \end{aligned}$$

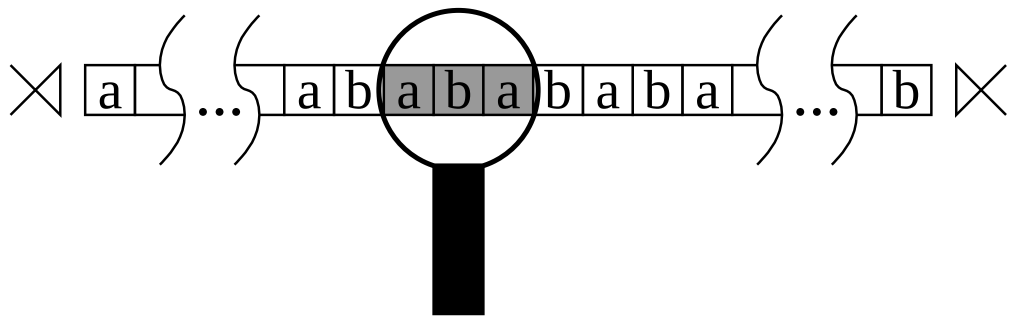
and a SFE expression E of the form

$$\mathop{\&}_{p_i \in P} \overline{p_i \overline{\varnothing}} \quad \mathop{\&}_{w_i \in W} \overline{\overline{\varnothing} w_i \overline{\varnothing}} \quad \mathop{\&}_{s_i \in S} \overline{\overline{\varnothing} s_i}$$

such that $L = [\![E]\!]$.

Strictly Local Languages: Scanners

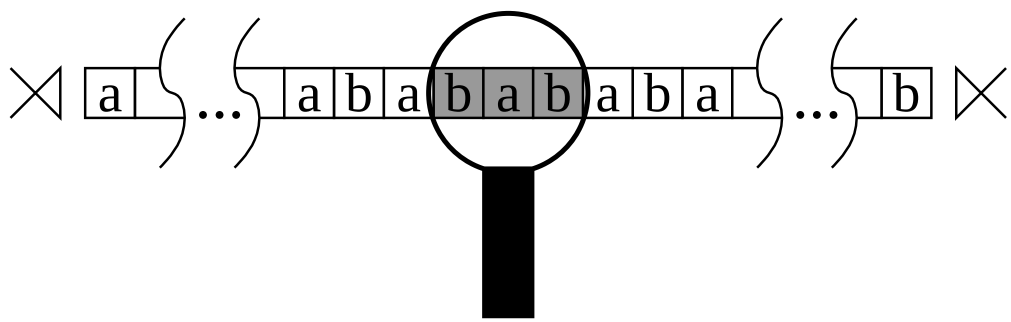
Intuitively, if L is Strictly k-Local, then deciding whether a string w belongs to L simply requires scanning w for forbidden prefixes, substrings, and suffixes. If any is found w is rejected. If every prefix, substring, and suffix in w is permissible then w is accepted.



(McNaughton and Papert 1971, Rogers and Pullum 2011)

Strictly Local Languages: Scanners

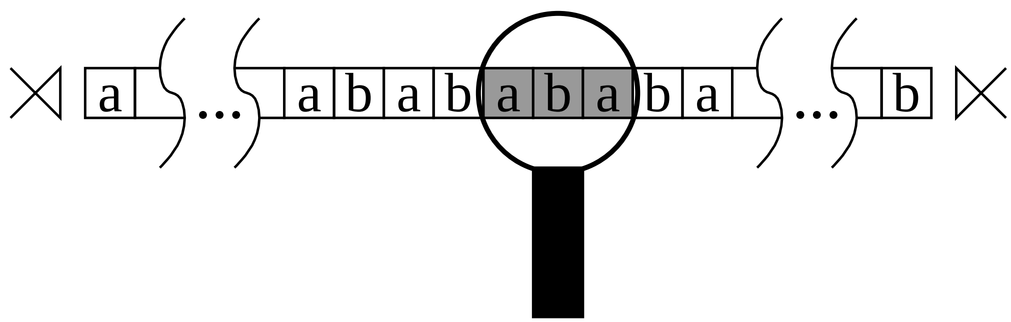
Intuitively, if L is Strictly k-Local, then deciding whether a string w belongs to L simply requires scanning w for forbidden prefixes, substrings, and suffixes. If any is found w is rejected. If every prefix, substring, and suffix in w is permissible then w is accepted.



(McNaughton and Papert 1971, Rogers and Pullum 2011)

Strictly Local Languages: Scanners

Intuitively, if L is Strictly k-Local, then deciding whether a string w belongs to L simply requires scanning w for forbidden prefixes, substrings, and suffixes. If any is found w is rejected. If every prefix, substring, and suffix in w is permissible then w is accepted.



(McNaughton and Papert 1971, Rogers and Pullum 2011)

More exercises

Recall some of the languages mentioned earlier.

(1) Strings end with a `b`.

(2) The second to last symbol in all strings is `b`.

(3) The third to last symbol in all strings is `b`.

For each, one explain why it is Strictly Local. Assume $\Sigma = \{\texttt{a,b}\}$.

Let L=$[\![(\mathtt{ba})^*]\!]$

Logical Characterization

Conjunctions of Negative Literals (with successor)

$$E = \overline{\mathtt{a}\overline{\varnothing}} \;\&\; \overline{\overline{\varnothing}\mathtt{aa}\overline{\varnothing}} \;\&\; \overline{\overline{\varnothing}\mathtt{bb}\overline{\varnothing}} \;\&\; \overline{\overline{\varnothing}\mathtt{a}}$$

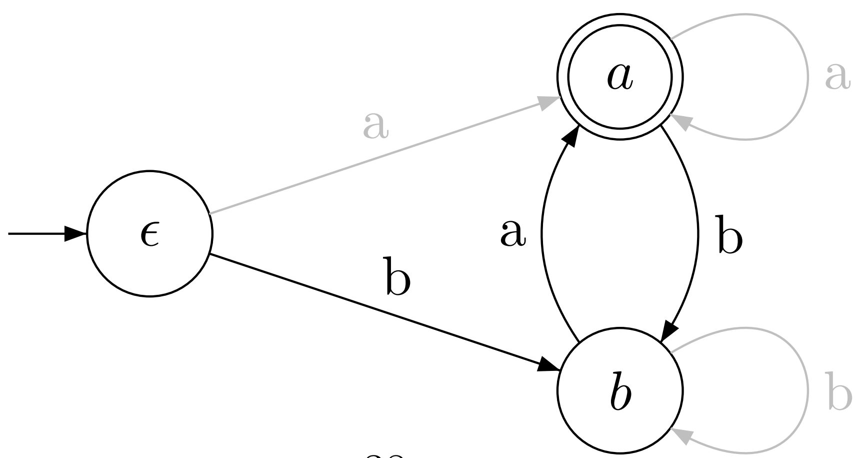
$$\phi = \neg\rtimes\mathtt{a} \wedge \neg\mathtt{aa} \wedge \neg\mathtt{bb} \wedge \neg\mathtt{a}\ltimes$$

where the above are interpreted as *substrings* (per the successor word model)

Let L=$[\![(\texttt{ba})^*]\!]$

DFA Characterization

$$
\begin{aligned}
Q &= \Sigma^{\leq k} \\
q_0 &= \epsilon \\
F &= q \in Q \text{ such that } q \text{ is not a forbidden suffix} \\
\delta(q,a) &= \texttt{Suffix}^{k-1}qa \text{ if } q, \texttt{Suffix}^{k-1}qa \text{ are not forbidden}
\end{aligned}
$$



Strictly k-Local languages

A language L is Strictly k-Local if there is a Strictly Local expression E such that $L = [\![E]\!]$ and

$$k = \max\left\{|w| \mid w \in W\right\} \cup \left\{|v| + 1 \mid P \cup S\right\} .$$

Exercises

For what k are the following Strictly Local?

(1) Strings end with a `b`.

(2) The second to last symbol in all strings is `b`.

(3) The third to last symbol in all strings is `b`.

Theorems

1. $\mathrm{SL}_1 \subsetneq \mathrm{SL}_2 \ldots \mathrm{SL}_k \subsetneq \mathrm{SL}_{k+1} \ldots \subseteq \mathrm{SL}$
2. $\mathrm{FIN} \subsetneq \mathrm{SL}$
3. For each k, $\mathrm{FIN} \not\subseteq \mathrm{SL}_k$
4. For each k, SL_k is closed under intersection, but neither complement nor union.

Grammar-independent Characterization of SL

Suffix Substitution Closure

A language L is Strictly Local iff there is a k such that for all strings $u_1, v_1, u_2, v_2 \in \Sigma^*$ and for all strings x of length $k-1$ whenever $u_1xv_1, u_2xv_2 \in L$ then $u_1xv_2 \in L$.

In a picture

$$
\begin{array}{cccc}
 & \vdash k-1 \dashv & & \\
u_1 & x & v_1 & \in L \\
u_2 & x & v_2 & \in L \\
\hline
u_1 & x & v_2 & \in L
\end{array}
$$

Proving some languages are not SL

The SSC helps us prove languages are not Strictly k-Local and even not Strictly Local for any k.

- To show L is not SL_k, find u_1, v_1, u_2, v_2 and x of length $k-1$ such that

In a picture

$$
\begin{array}{cccc}
 & \vdash k-1 \dashv & & \\
u_1 & x & v_1 & \in L \\
u_2 & x & v_2 & \in L \\
\hline
u_1 & x & v_2 & \notin L
\end{array}
$$

- To show L is not SL, find u_1, v_1, u_2, v_2 and x for each k!

Exercises

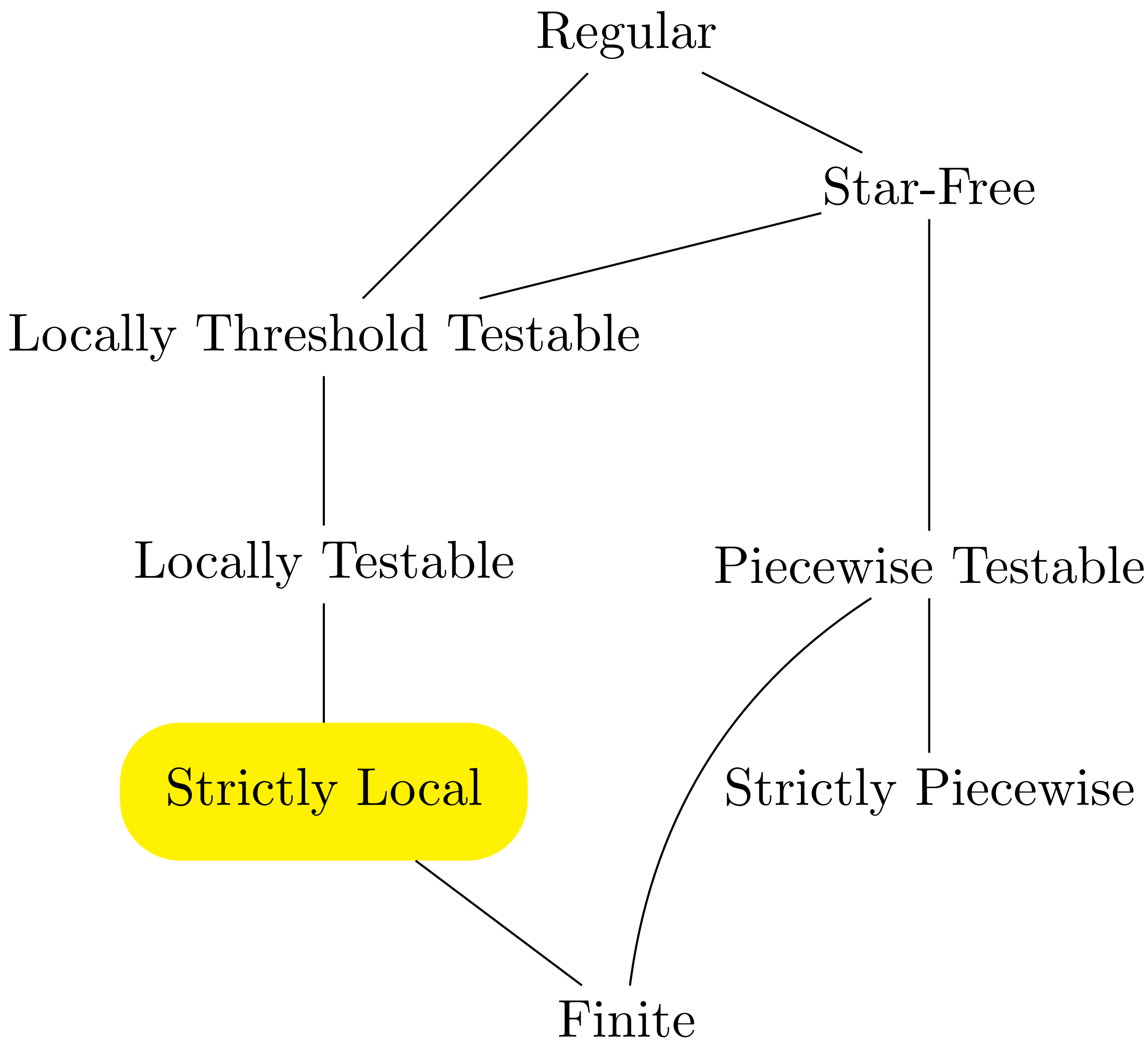
Prove the following languages are not SL for any k.

(4) Strings contain at least one `b`.

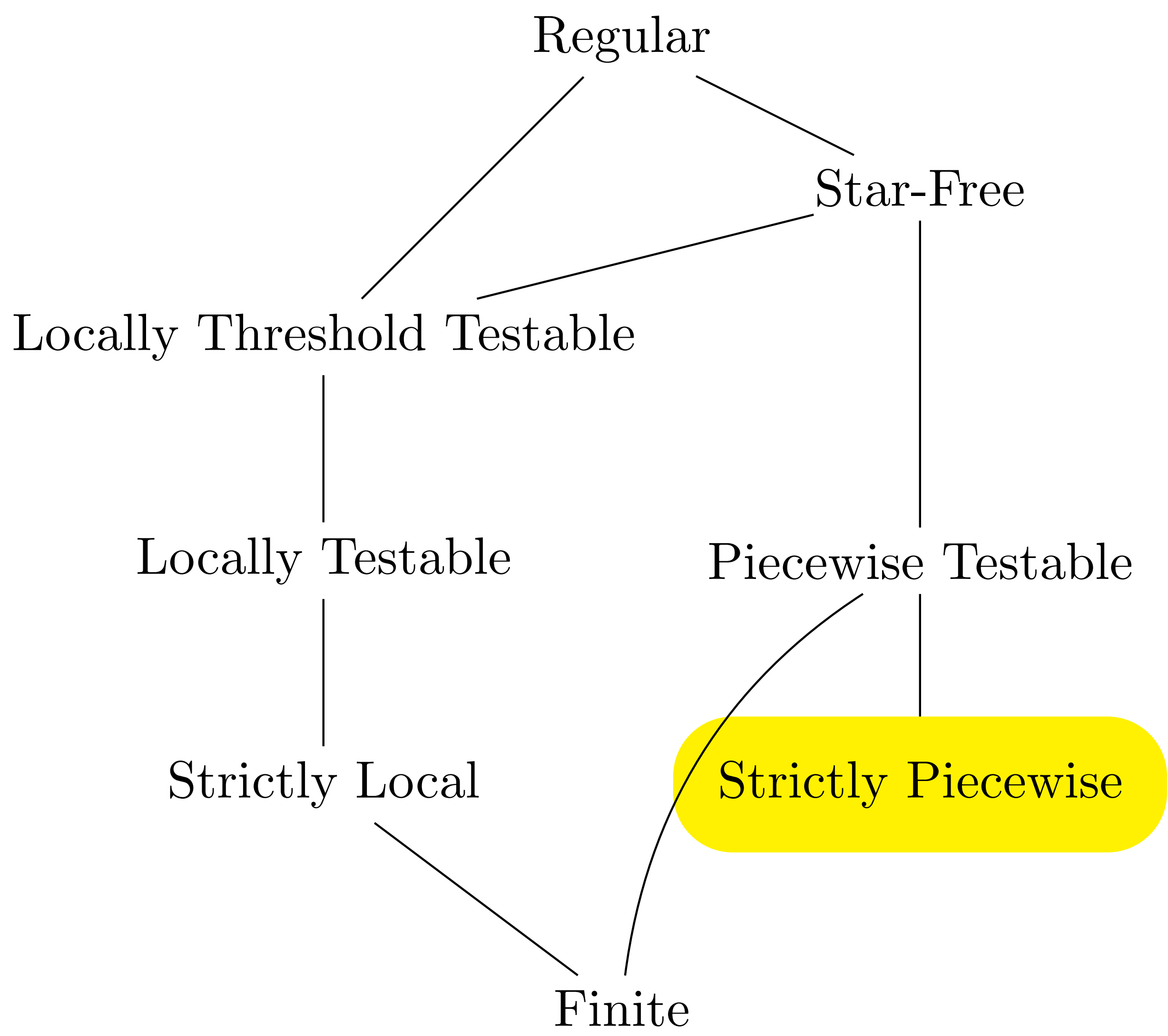
(5) Strings contain at most one `b`.

(10) Strings contain an `a` between every pair of `b`s.

Strictly Local Languages



Strictly Piecewise Languages



Strictly Piecewise Languages

Intuitively, a language is SP if can be defined by forbidding finitely many *subsequences*.

Key Expression

$\overline{\varnothing}\sigma_1\overline{\varnothing}\sigma_2\overline{\varnothing}\ldots\sigma_n\overline{\varnothing}$	strings containing $\sigma_1\sigma_2\ldots\sigma_n$ as a subsequence

For example, $\overline{\varnothing}\mathtt{a}\overline{\varnothing}\mathtt{b}\overline{\varnothing}$ designates those strings containing `ab` as a subsequence like: `ab`, `cccacccccccbcccc` and so on.

Strictly Piecewise Languages

Intuitively, a language is SP if can be defined by forbidding finitely many *subsequences*.

Key Expression

$\overline{\overline{\varnothing}\sigma_1\overline{\varnothing}\sigma_2\overline{\varnothing}\ldots\sigma_n\overline{\varnothing}}$	strings *not* containing $\sigma_1\sigma_2\ldots\sigma_n$ as a subsequence

For example, $\overline{\overline{\varnothing}\mathtt{a}\overline{\varnothing}\mathtt{b}\overline{\varnothing}}$ designates those strings *not* containing `ab` as a subsequence.

Strictly Piecewise Languages

A Formal Definition

A language L is Strictly Piecewise if there is finite set of strings $W =$

$$\begin{array}{rcl} w_1 & = & \sigma^1_1\sigma^1_2\ldots\sigma^1_{|w_1|} \\ w_2 & = & \sigma^2_1\sigma^2_2\ldots\sigma^2_{|w_2|} \\ \ldots & & \\ w_n & = & \sigma^n_1\sigma^n_2\ldots\sigma^n_{|w_n|} \end{array}$$

and a SFE expression E of the form

$$\mathop{\&}_{w_i \in W} \overline{\overline{\varnothing}\sigma^i_1\overline{\varnothing}\sigma^i_2\overline{\varnothing}\ldots\sigma^i_{|w_i|}\overline{\varnothing}}$$

such that $L = [\![E]\!]$.

Exercises

1. Recall this language mentioned earlier.

   (5) Strings contain at most one `b`.

   Explain why it is Strictly 2-Piecewise. Assume Σ ={`a`,`b`}.

2. Recall Samala.

possible Samala words	impossible Samala words
ʃtojonowonowaʃ	stojonowonowaʃ
stojonowonowas	ʃtojonowonowas
pistonoskiwat	pisotonoʃikiwat
sanisotonoskiwas	ʃanipisotonoʃikiwas

   Assuming Σ ={`s`,`S`,`t`,`o`}, what are the forbidden subsequences?

Strictly Piecewise Languages

Intuitively, if L is Strictly k-Piecewise, then deciding whether a string w belongs to L simply requires checking its k-subsequences for forbidden ones. If any is found w is rejected. If every subsequence in w is permissible then w is accepted.

(Rogers et al. 2010)

Let L=$\mathtt{a}^* + \mathtt{b}^*$

Logical Characterization

Conjunctions of Negative Literals (with precedence)

$$E = \overline{\overline{\varnothing}\mathtt{a}\overline{\varnothing}\mathtt{b}\overline{\varnothing}} \ \& \ \overline{\overline{\varnothing}\mathtt{b}\overline{\varnothing}\mathtt{a}\overline{\varnothing}}$$

$$\phi = \neg\mathtt{ab} \wedge \neg\mathtt{ba}$$

where the above are interpreted as *subsequences* (per the precedence word model)

Theorems

1. $\mathrm{SP}_1 \subsetneq \mathrm{SP}_2 \ldots \mathrm{SP}_k \subsetneq \mathrm{SP}_{k+1} \ldots \subseteq \mathrm{SP}$
2. $\mathrm{FIN} \not\subseteq \mathrm{SP}$
3. For each k, SP_k is closed under intersection, but neither complement nor union.

(Rogers et al. 2010)

Characterizing Strictly Piecewise Languages

Subsequence Closure

A language L is Strictly Piecewise iff whenever $w \in L$ every subsequence of w also belongs to L.

Subsequence Closure helps us prove languages are not Strictly Piecewise.

(Rogers et al. 2010)

Exercises

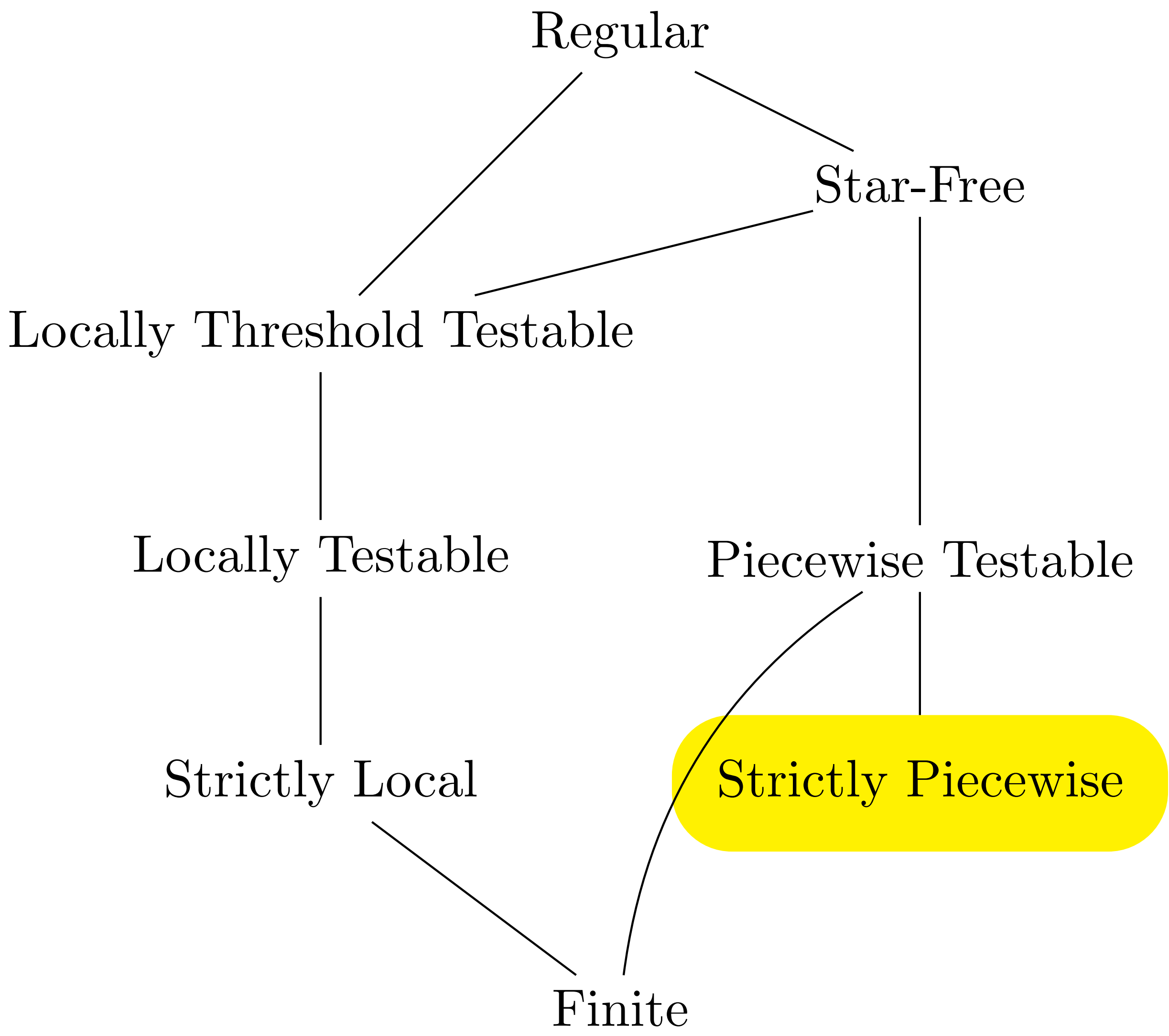
Prove the following languages are not SP.

(4) Strings contain at least one `b`.

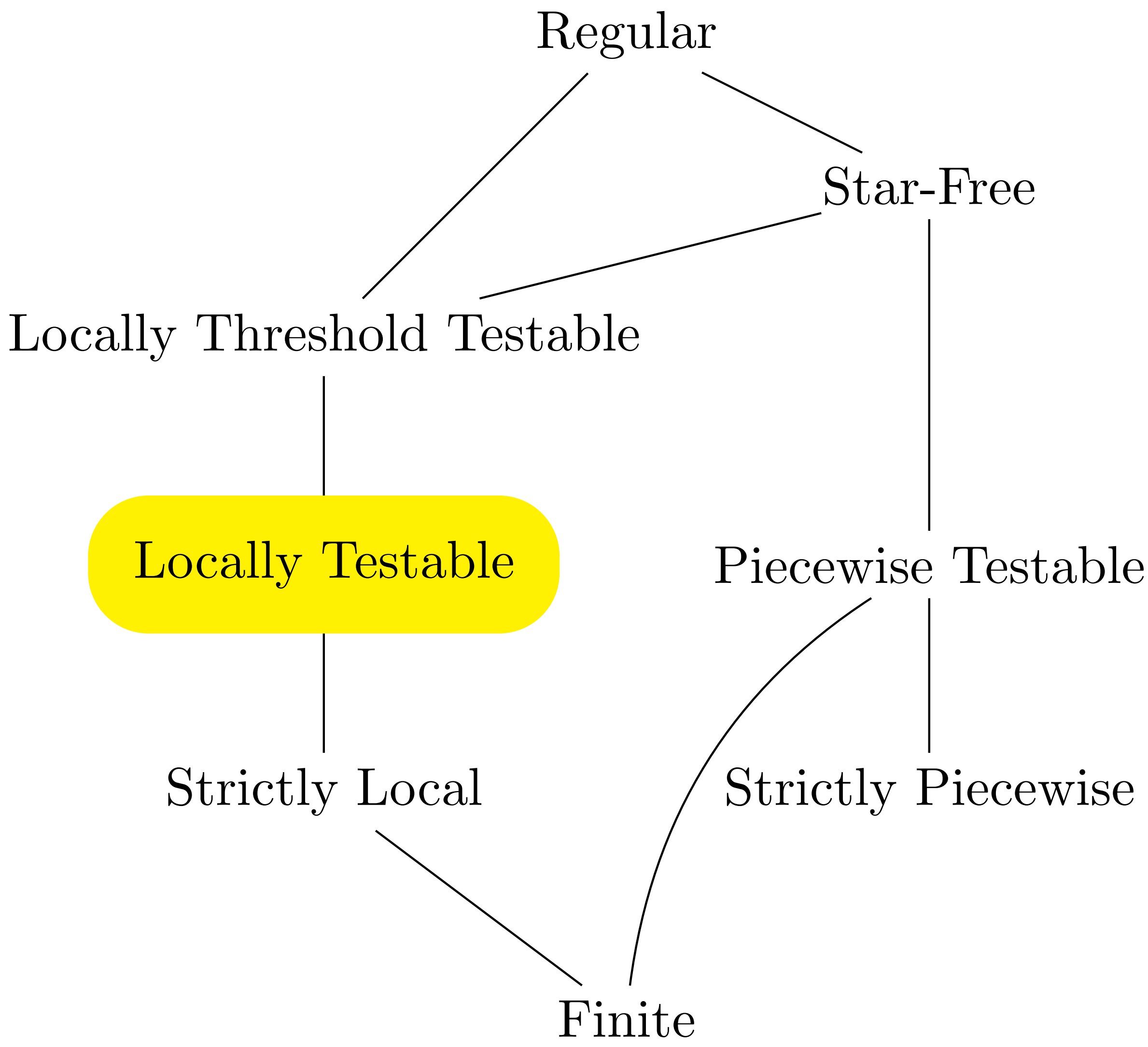
(10) Strings contain an `a` between every pair of `b`s.

(11) Strings contain an odd number of `b`s.

Strictly Piecewise Languages



Locally Testable Languages



Locally Testable Languages

A language L is Locally k-Testable if it a Boolean combination of finitely many Strictly k-Local languages.

Boolean operations (and regular expression equivalents)

- intersection (L_1 & L_2)
- union ($L_1 + L_2$)
- complement ($\overline{L}$)

A language is Locally Testable if it is Locally k-Testable for some k.

Exercises

Recall some of the languages mentioned earlier.

(4) Strings contain at least one `b`.

(7) Strings contain at least one `bb` substring.

For each, one explain why it is Locally Testable. What is the k value?

L = strings containing `a` or strings containing `bb`

Logical Characterization

Propositional logic (with successor)

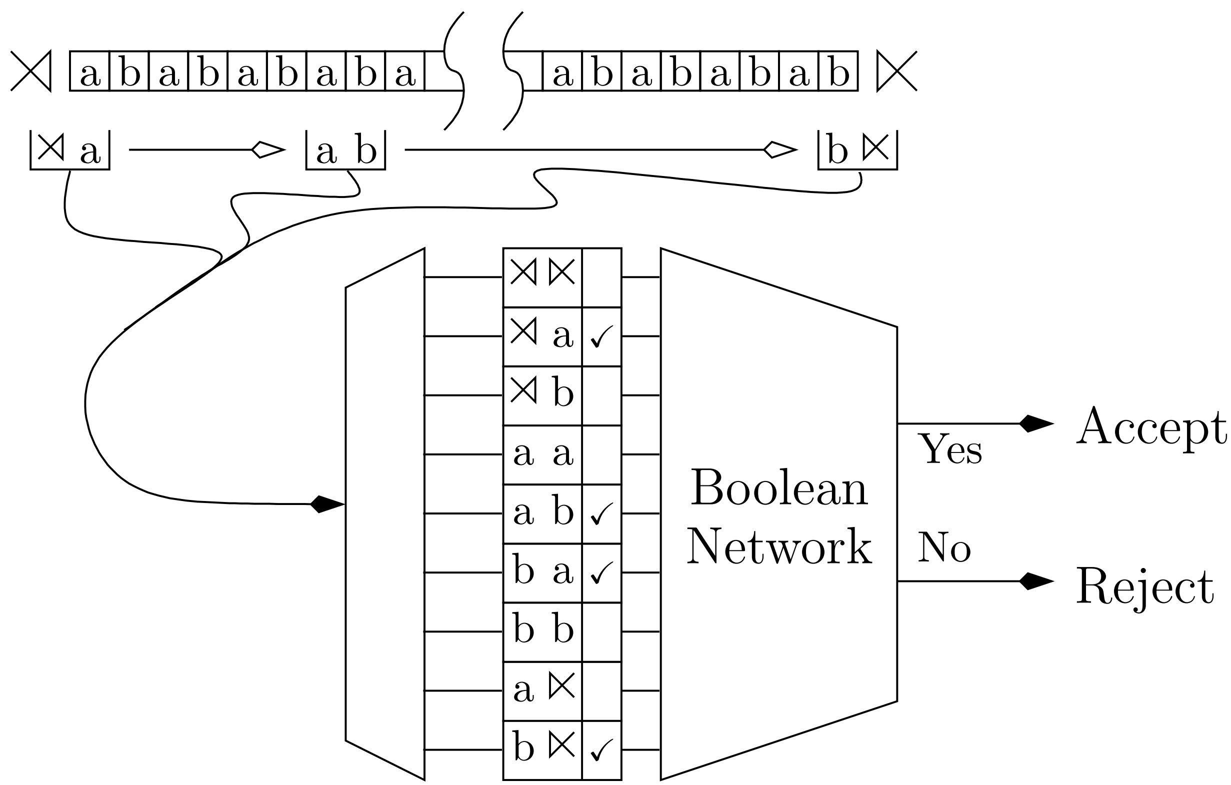
$$E = \overline{\varnothing}\mathtt{a}\overline{\varnothing} + \overline{\varnothing}\mathtt{bb}\overline{\varnothing}$$

$$\phi = \mathtt{a} \vee \mathtt{bb}$$

where the above are interpreted as *substrings* (per the successor word model)

Locally Testable Languages

Intuitively, membership in an LT_k language depends only on the sets of prefixes of length $k-1$, substrings of length k, and suffixes of length $k-1$. So these elements need to be identified and stored in memory to decide membership.



Grammar-independent characterization of Locally Testable Languages

Locally Testability

A language L is Locally Testable iff there exists k such that for all $u, v \in \Sigma^*$, if u and v have the same $k-1$ prefix, k-long substrings, and $k-1$ suffix then either $u, v \in L$ or $u, v \notin L$.

(McNaughton and Papert 1971, Rogers and Pullum 2011)

Exercise

Using Locally Testable Equivalence, prove the language below is not PT.

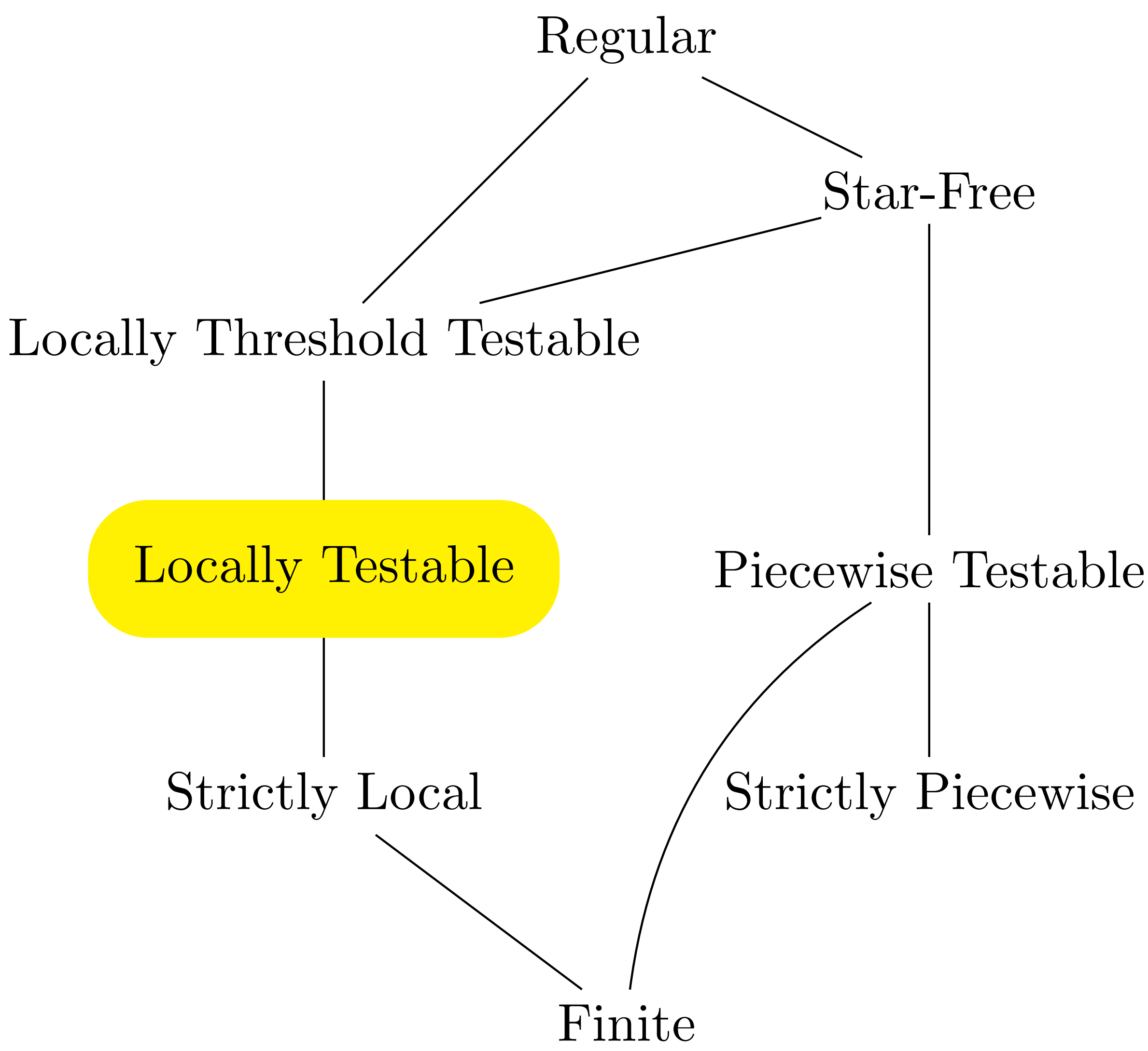
(5) String contain exactly one `b`.

Theorems

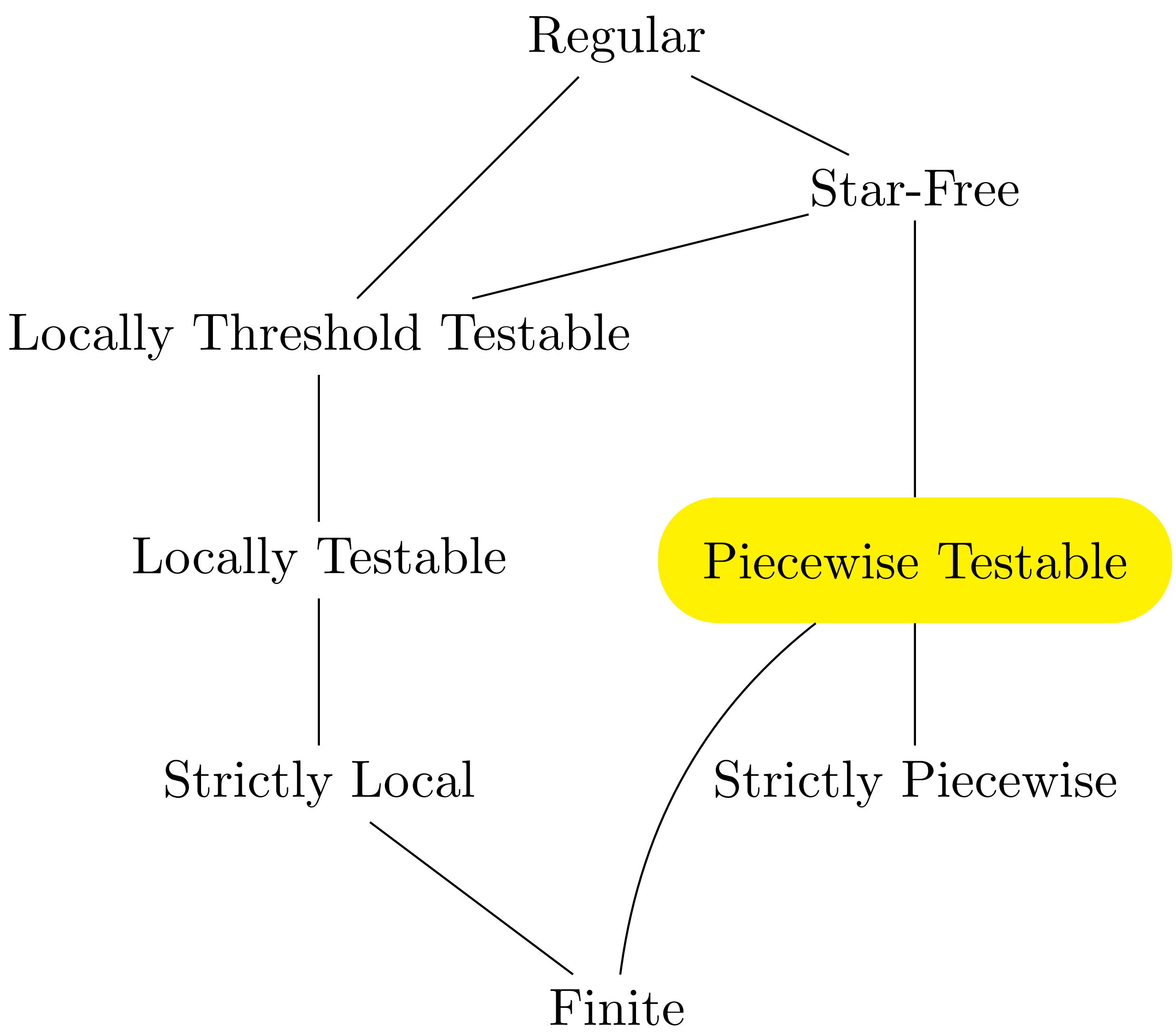
1. $\mathrm{LT}_1 \subsetneq \mathrm{LT}_2 \ldots \mathrm{LT}_k \subsetneq \mathrm{LT}_{k+1} \ldots \subseteq \mathrm{LT}$.
2. For each k, $\mathrm{SL}_k \subsetneq LT_k$.
3. LT and SP are incomparable.
4. Closing LT under concatenation yields the Star-Free languages.

(McNaughton and Papert 1971, Rogers and Pullum 2011)

Locally Testable Languages



Piecewise Testable Languages



Piecewise Testable Languages

A language L is Piecewise k-Testable if it a Boolean combination of finitely many Strictly k-Piecewise languages.

Boolean operations (and regular expression equivalents)

- intersection (L_1 & L_2)
- union ($L_1 + L_2$)
- complement ($\overline{L}$)

A language is Piecewise Testable if it is Piecewise k-Testable for some k.

Exercises

Recall this language mentioned earlier.

(8) Strings contain at least two `b`s.

Explain why it is Piecewise Testable. What is the k value?

Example L

Consider the set of strings w such that if w contains a `bb` subsequence then w also contains an `aa` subsequence.

Logical Characterization

Propositional logic (with precedence)

$$\phi = \texttt{bb} \rightarrow \texttt{aa}$$

where the above are interpreted as *subsequences* (per the precedence word model)

Piecewise Testable Languages

Intuitively, membership in an PT_k language depends only on the set of subsequences of length k. So these elements need to be identified and stored in memory to decide membership.

Grammar-independent characterization of Piecewise Testable Languages

Piecewise Testability

A language L is Piecewise Testable iff there exists k such that for all $u, v \in \Sigma^*$, if u and v have the same k-long subsequences then either $u, v \in L$ or $u, v \notin L$.

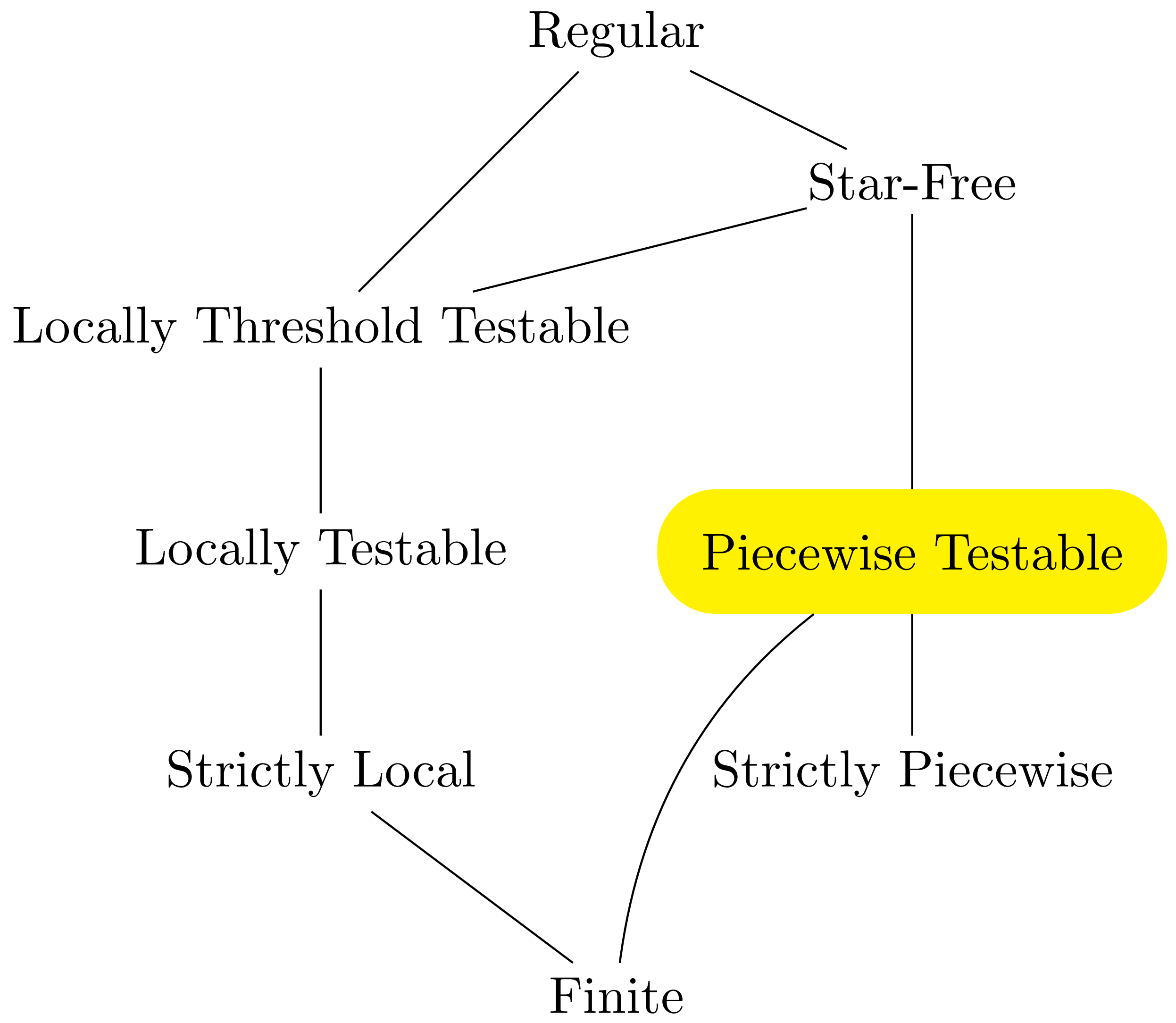
(Simon 1975)

Theorems

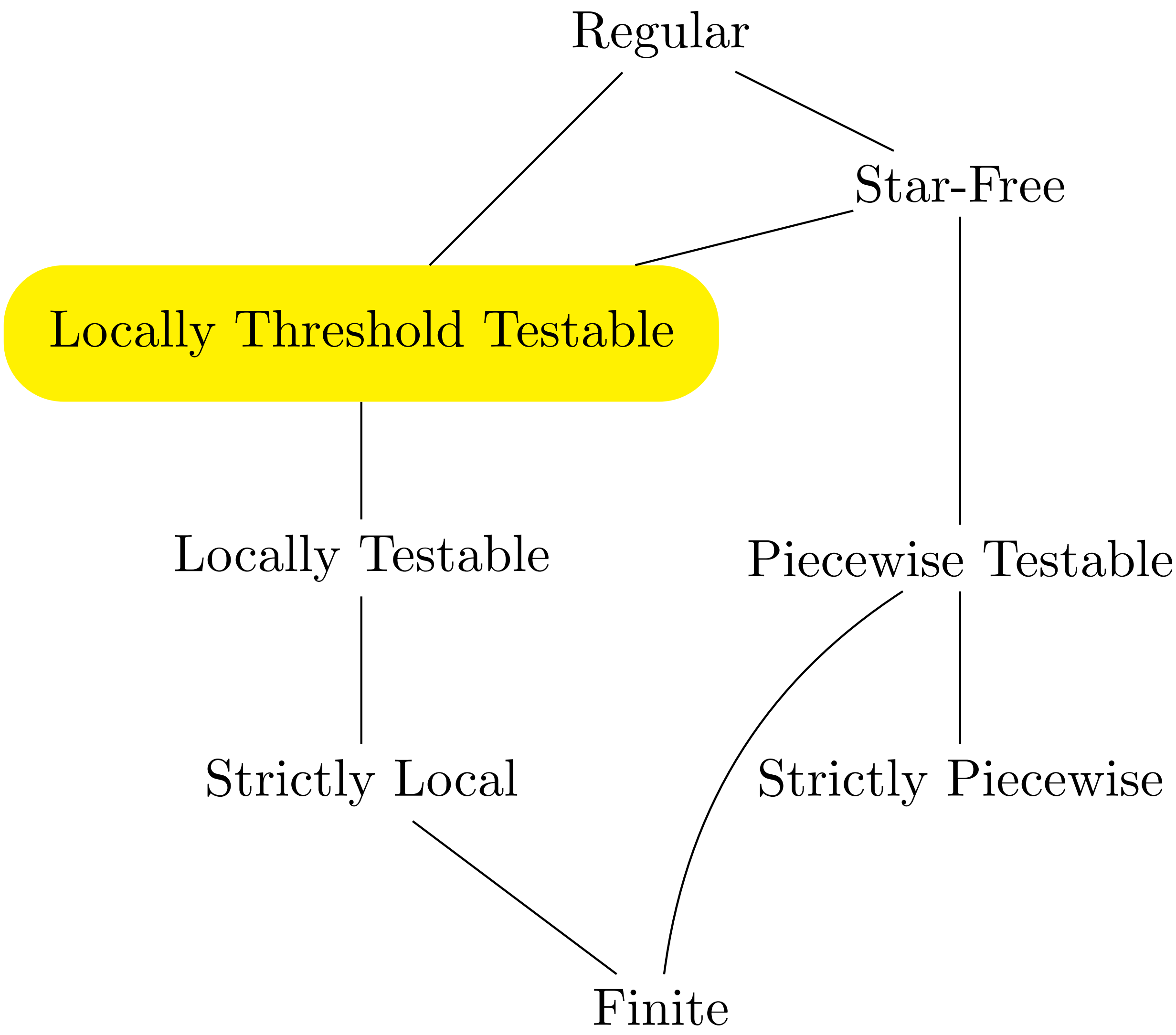
1. $\mathrm{PT}_1 \subsetneq \mathrm{PT}_2 \ldots \mathrm{PT}_k \subsetneq \mathrm{PT}_{k+1} \ldots \subseteq \mathrm{PT}$.
2. For each k, $\mathrm{SP}_k \subsetneq PT_k$.
3. PT and LT are incomparable.

(Rogers et al. 2010, 2013)

Piecewise Testable Languages

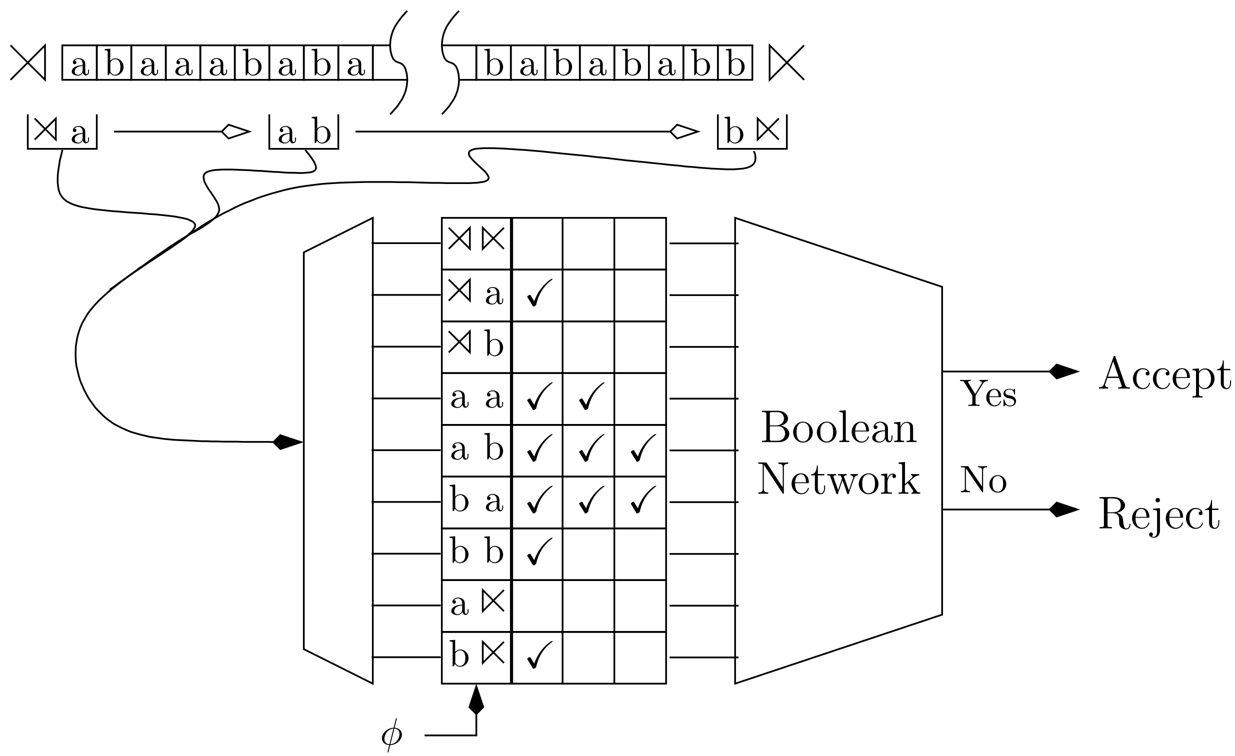


Locally Threshold Testable Languages



Locally Threshold Testable Languages

Intuitively, membership in an $\mathrm{LT}_{t,k}$ language depends only on the sets of prefixes of length $k-1$, substrings of length k, and suffixes of length $k-1$, *counting their occurences* up to some threshold t. So this information needs to be identified and stored in memory to decide membership.



Grammar-independent characterization of Locally Threshold Testable Languages

Locally Theshold Testability

A language L is Locally Testable iff there exists k such that for all $u, v \in \Sigma^*$, if u and v have the same $k-1$ prefix, $k-1$ suffix, and the same number of occurrences of the same k-long substrings, counting up to some threshold t, then either $u, v \in L$ or $u, v \notin L$.

(Thomas 1982, Rogers and Pullum 2011)

Exercises

- Explain why (9) is LTT. What are the t and k values?

  (9) Strings contain two `bb` substrings.

- Explain why the language below is not LTT for any t, k.

  (L) Strings do not contain `ab` as a subsequence. Assume Σ={`a`,`b`,`c`}.

Logical Characterization

A language L is Locally Theshold Testable iff L is definable with First Order logic with successor ($L \in [\![\mathrm{FO}(+1)]\!]$).

(Thomas 1982, 1999)

Exercise

Using Locally Testable Equivalence, prove the language below is not PT.

(5) String contain exactly one `b`.

Theorems

1. For each k, $\mathrm{LT}_{t,k} \subsetneq \mathrm{LT}_{t,k+1} \subseteq \mathrm{LT}$.

2. For each k, $\mathrm{LT}_{t,k} \subsetneq \mathrm{LT}_{t+1} \subseteq \mathrm{LT}$.

3. For each k, $t > 1$, $\mathrm{LT}_{k} \subsetneq \mathrm{LT}_{t,k}$.

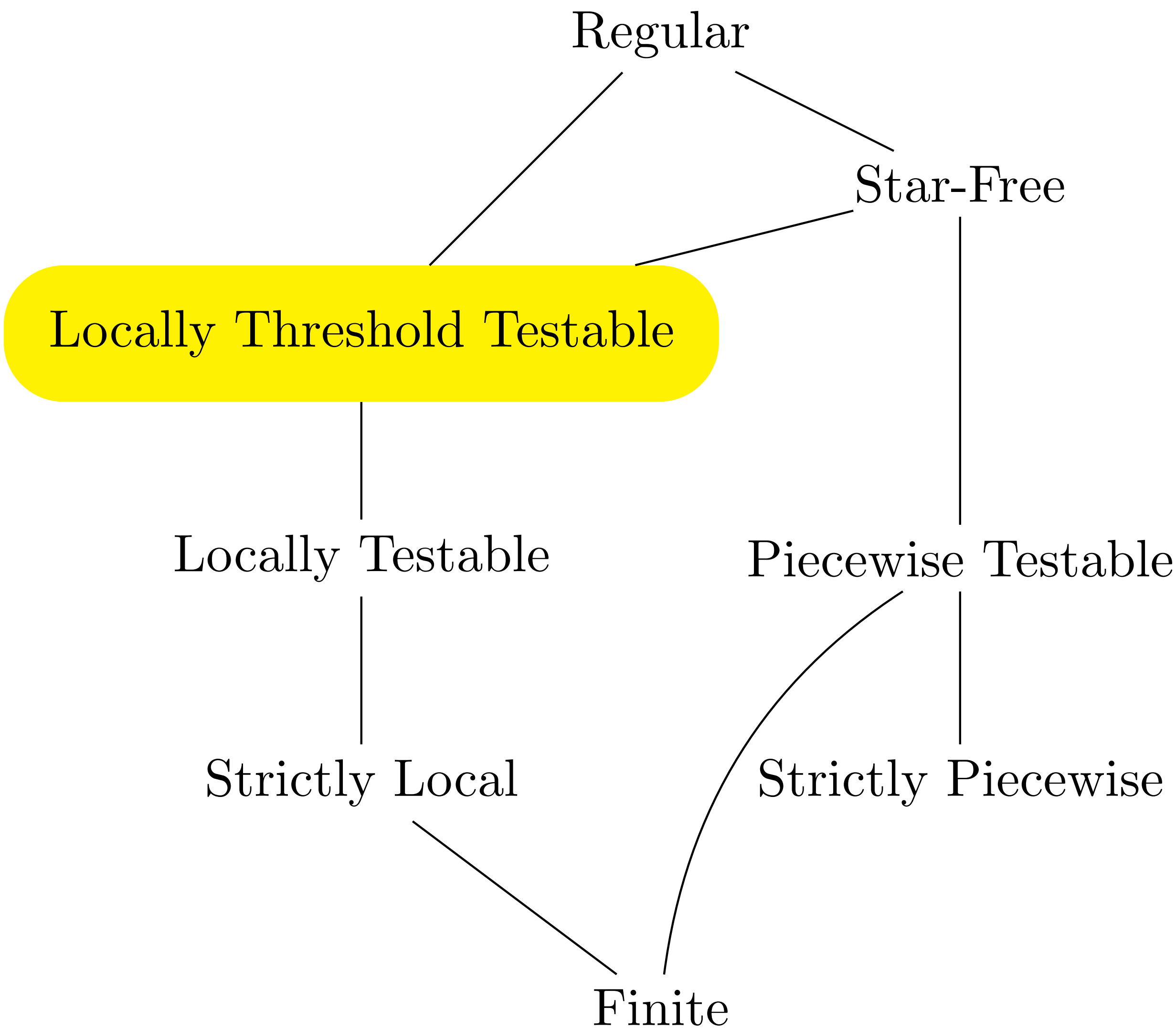
4. $\mathrm{LTT}_{1,k} = \mathrm{LT}_{k}$.

5. LTT and SP are incomparable.

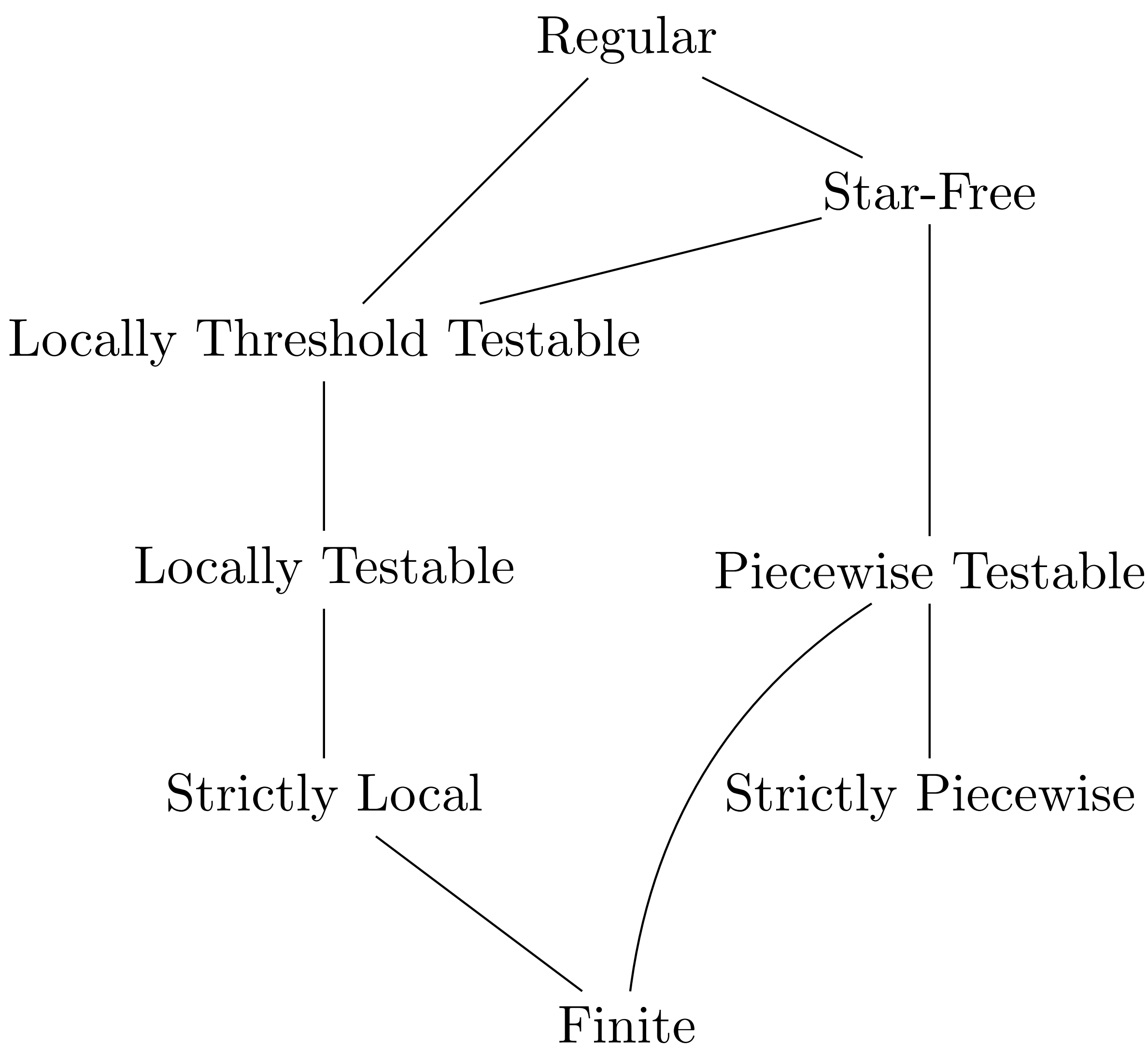
6. $\mathrm{LTT} \subsetneq \mathrm{SF}$.

(Thomas 1982, Rogers and Pullum 2011)

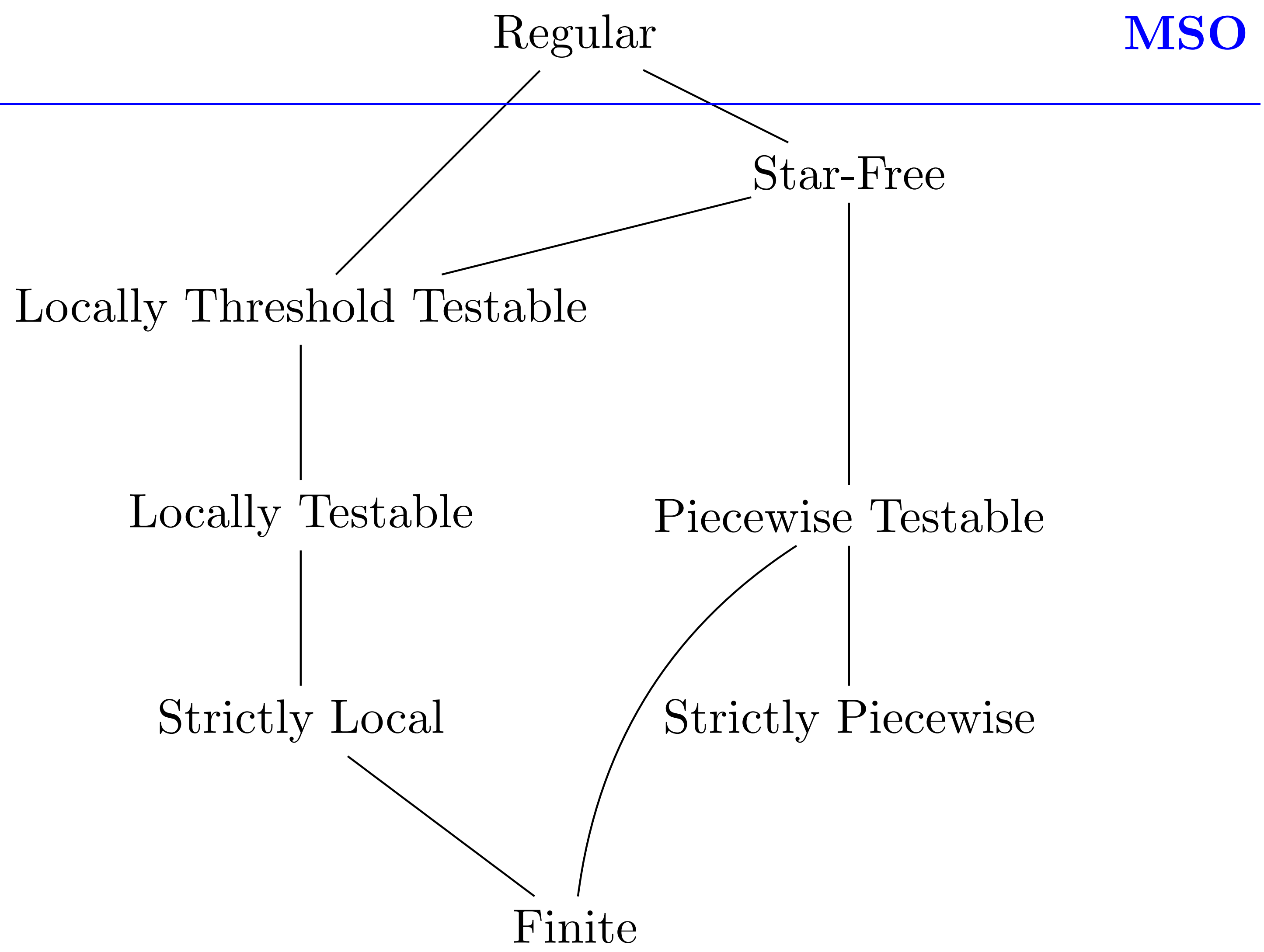
Locally Threshold Testable Languages



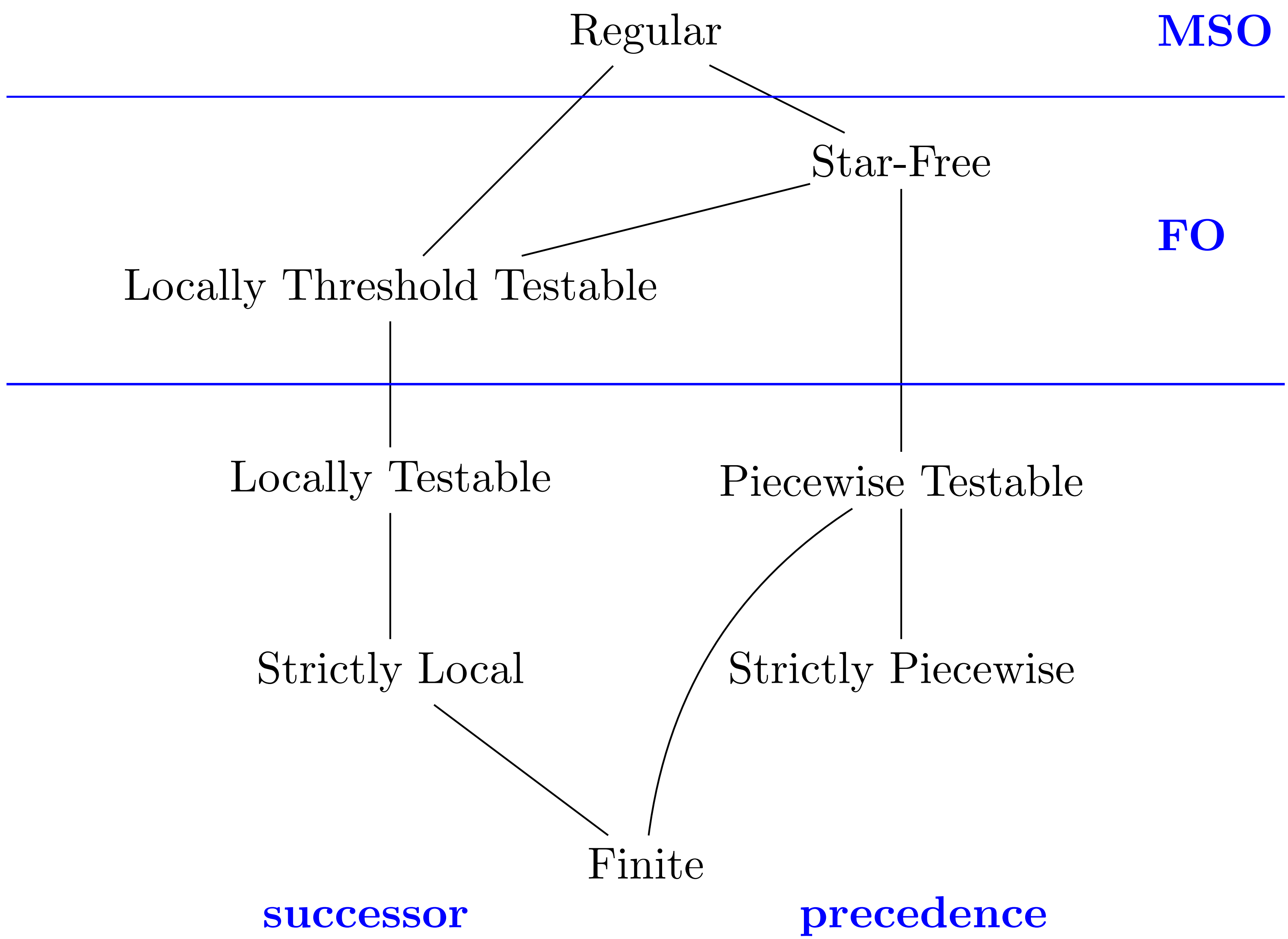
Subregular classes – A logical summary



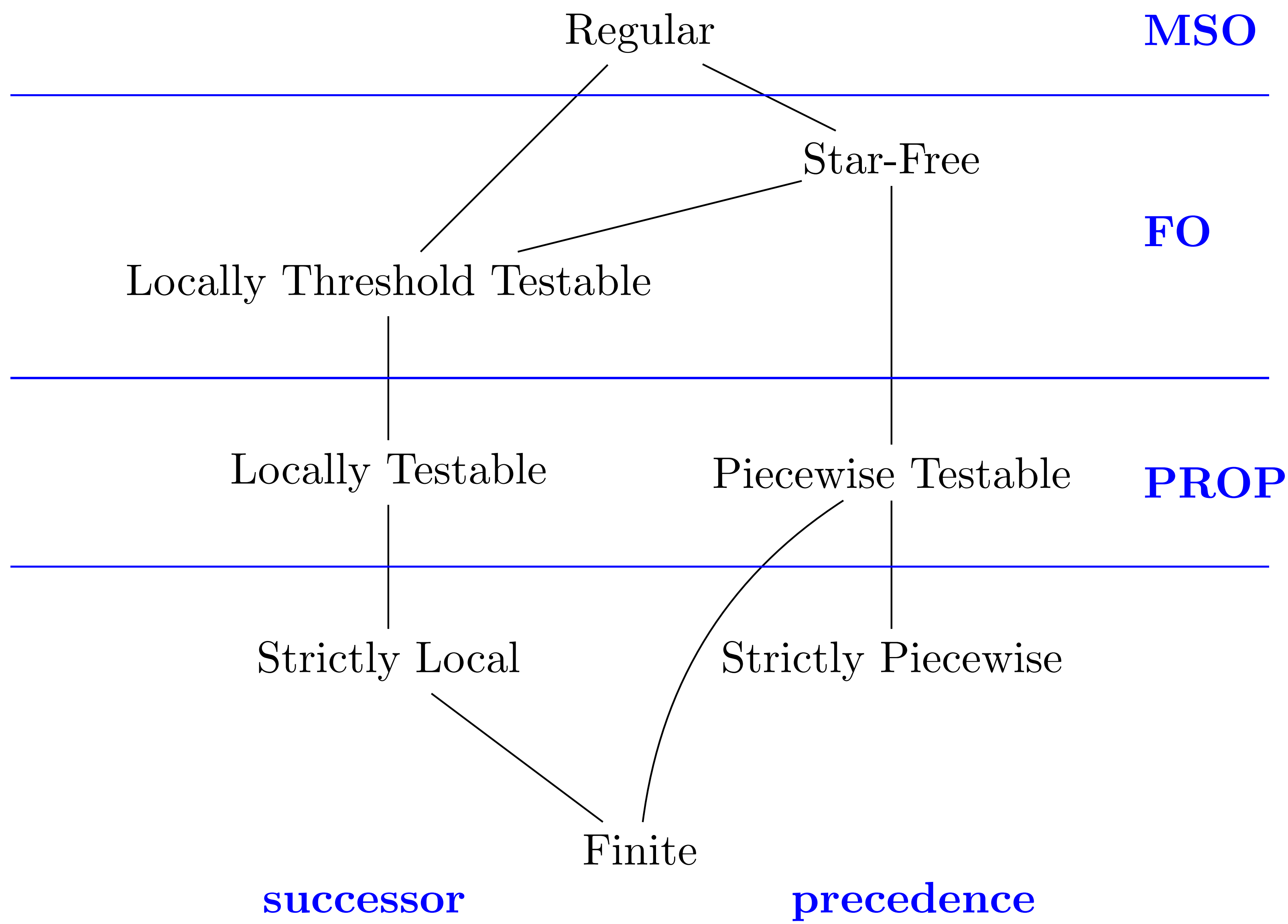
Subregular classes – A logical summary



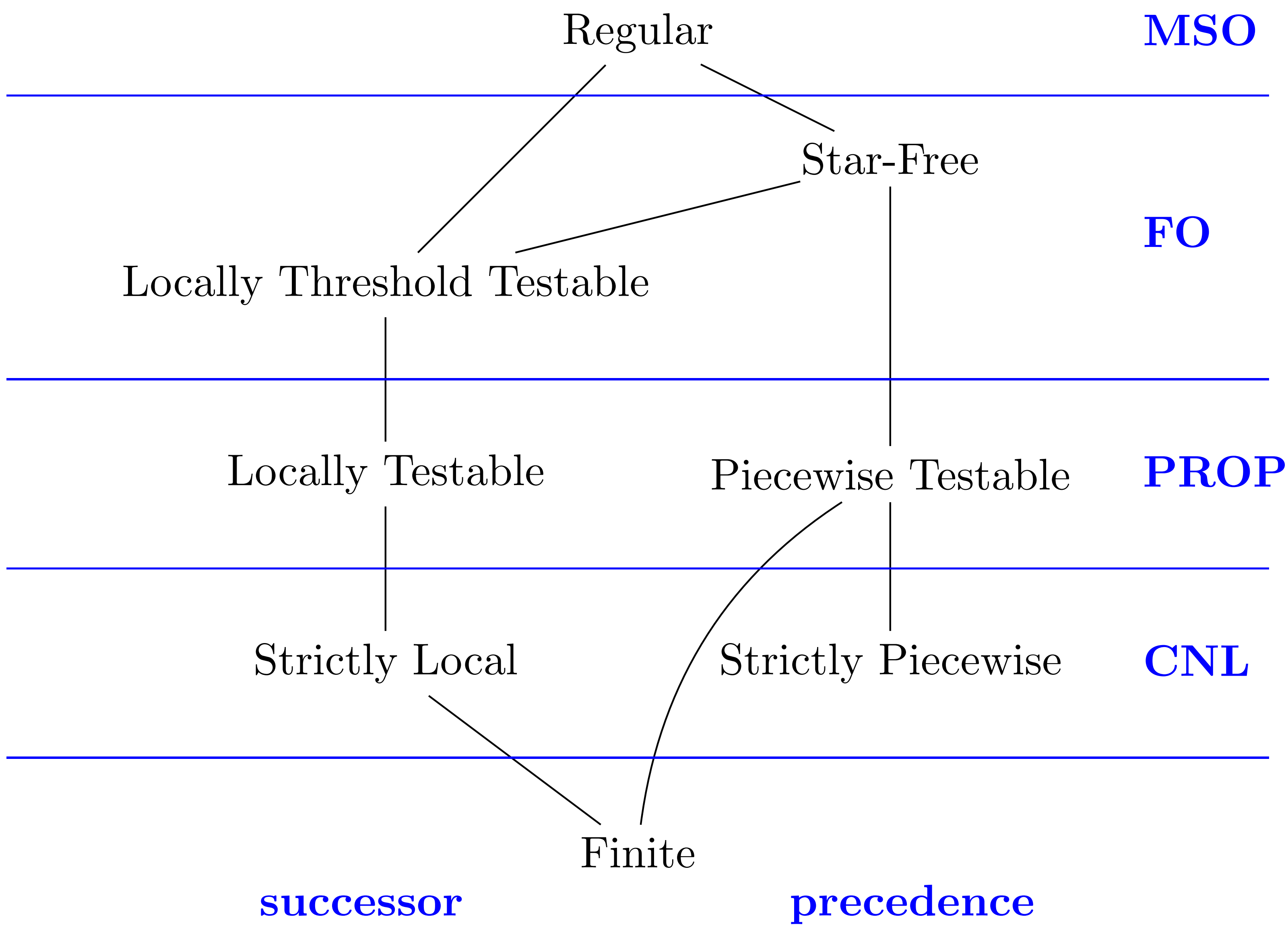
Subregular classes – A logical summary



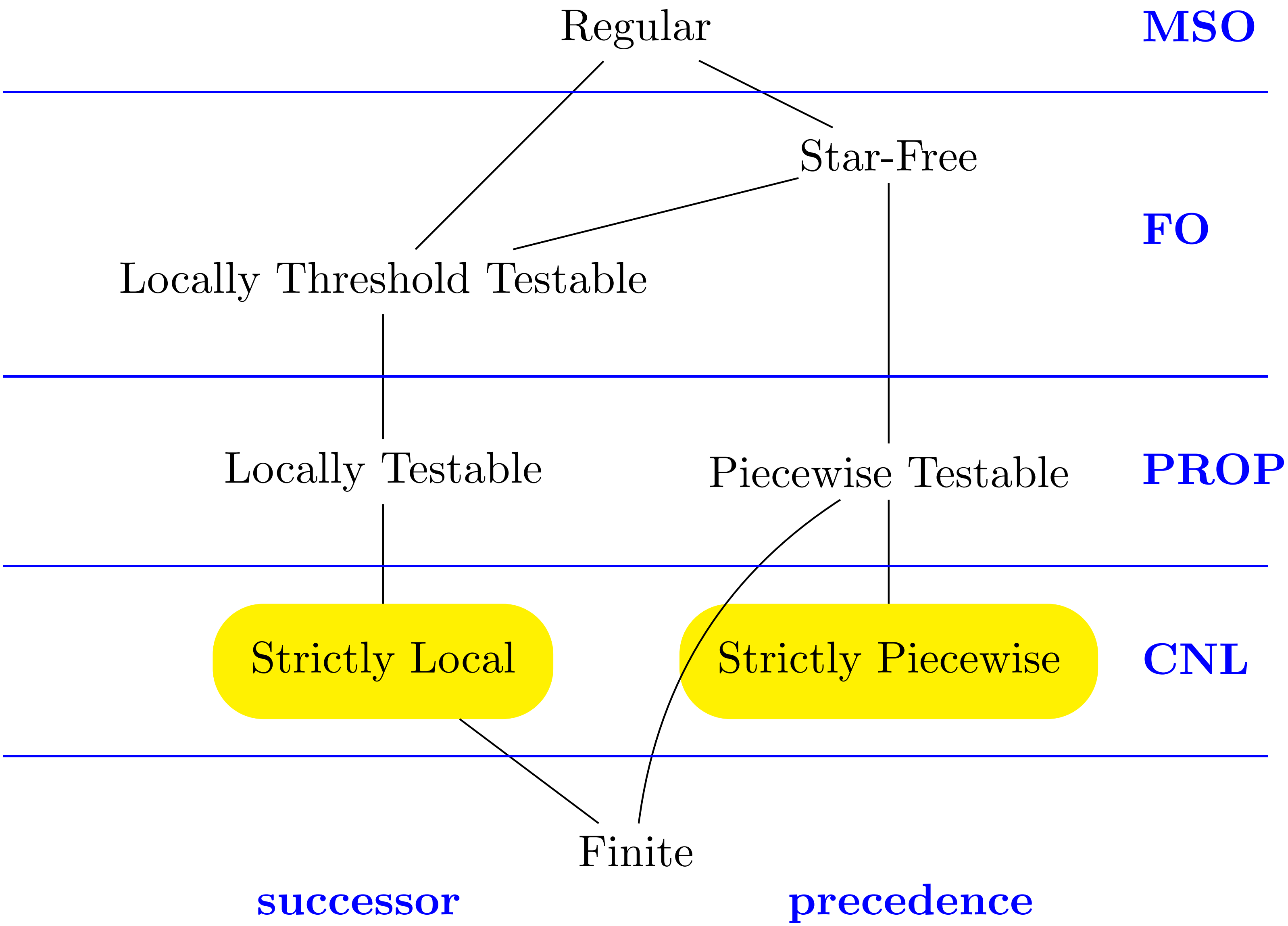
Subregular classes – A logical summary



Subregular classes – A logical summary



Natural language phonotactics are simple!



Remember This:

1. There are many important subregular classes of formal languages.
2. They are natural with converging definitions in terms of regular expressions, automata, logic, and abstract algebra.
3. They have grammar-independent characterizations which identify the kind of memory necessary for deciding membership.
4. They provide complexity measures for classifying regular patterns.
5. Probabilistic variants of these classes also exist.
6. SL_k, SP_k, LT_k, PT_k and $\mathrm{LTT}_{t,k}$ are machine-learnable from positive examples under various definitions of "learnable."

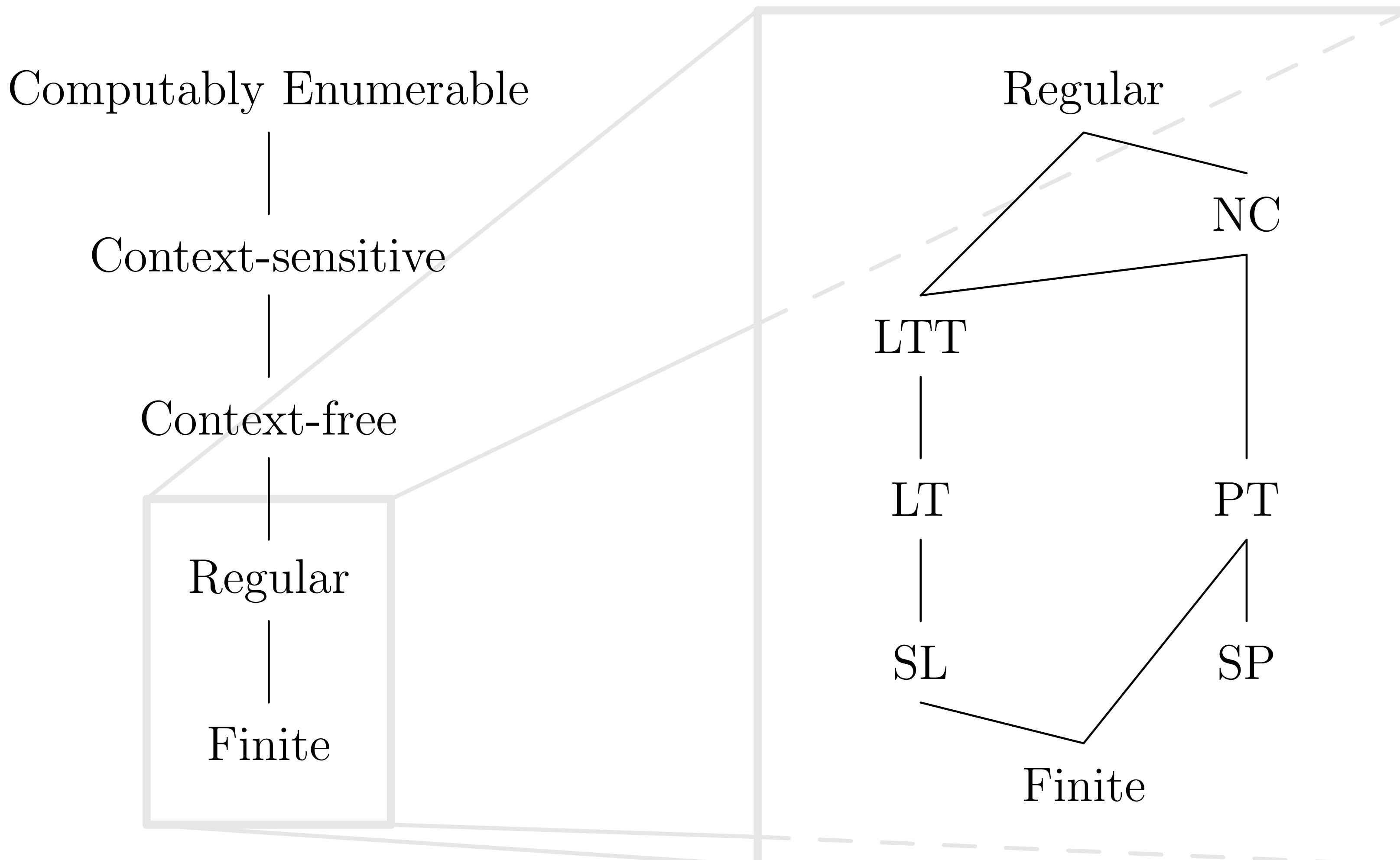
Current research

1. Regular string tranductions
2. Regular tree languages and tree transductions
3. Regular sets and transductions over other representations (e.g. graphs)

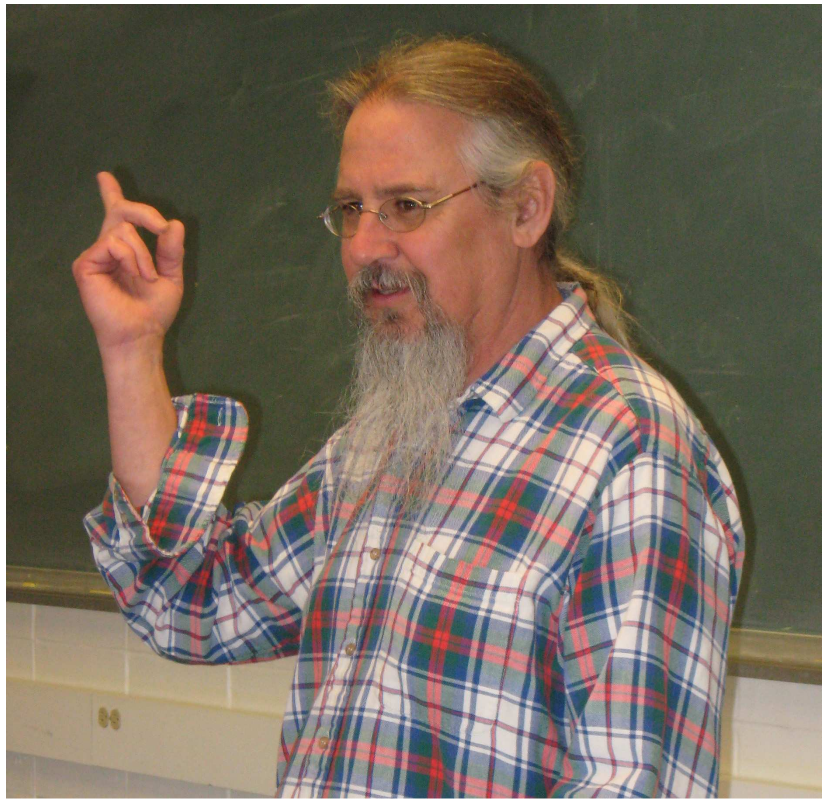
Applications

Much of this work has applications in linguistics and natural language processing, but also in other domains such as artificial intelligence, planning and control, model checking, ...

Computational Complexity

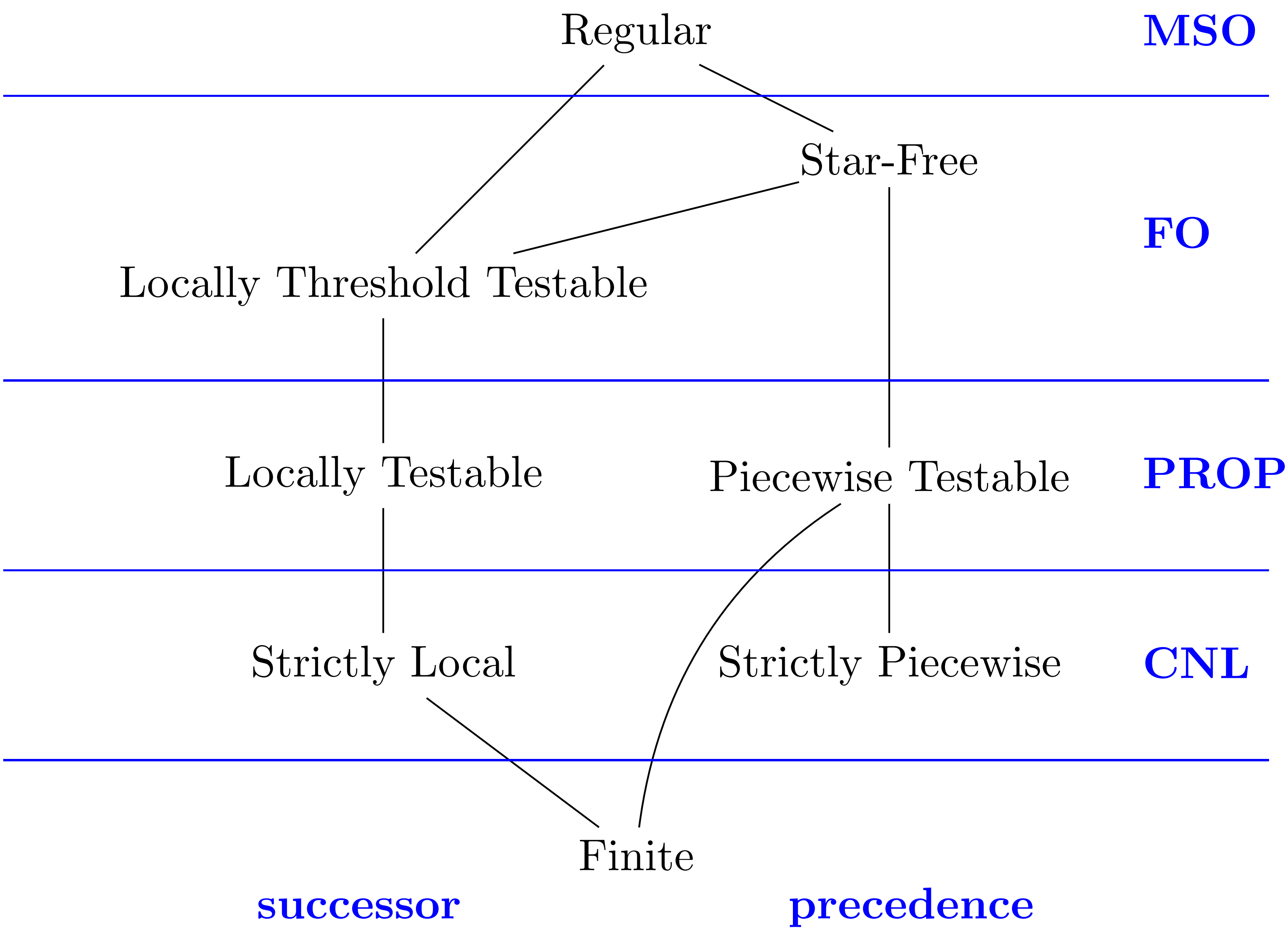


Thanks



Jim Rogers

Thank you!


Regular
MSO
Star-Free
FO
Locally Threshold Testable
Locally Testable
Piecewise Testable
PROP
Strictly Local
Strictly Piecewise
CNL
Finite
successor
precedence