Optimality Theory (Prince & Smolensky 1993)

Input: /tag/

Ranking1: NoCoda >> DEP >> PARSE

/tag/	NoCod	DEP	PARSE
F.ta.			*
.ta.g[e].		*!	
.tag.	*!		
.a.			**!
.ta.g.[e][e].	*!	*!	

Ranking2: NoCoda >> PARSE >> DEP

/tag/	NoCoda	PARSE	DEP
.ta.		*!	
F.ta.g[e].			*
.tag.	*!		
.a.		*!*	
.ta.g.[e][e].	*!		**

Ranking3:PARSE >> DEP >> NoCoda

/tag/	PARSE	DEP	NoCoda
.ta.	*!		
.ta.g[e].		*	*
F.tag.			*
.a.	*!*		
.ta.g.[e][e].		*!*	**

Operations over finite state machines

Intersection of two finite-state-automata A1, A2 ($A_1 \cap A_2$)

The intersection of two regular sets R_1, R_2 denotes the set of strings S such that $S \in R_1$ and $S \in R_2$ and is again a regular set.

Intersection(A_1, A_2)

make(I_{A1},I_{A2}) initial in $A_1 \cap A_2$
make(F_{A1},F_{A2}) final in $A_1 \cap A_2$
for each arc from u to v in A_1 labeled M
    for each arc from x to z in A_2 labeled M
        add an arc labeled M from (u,x) to (v,z) to $A_1 \cap A_2$

Composition of two finite-state transducers T_1, T_2 ($T_1 \oplus T_2$)

The composition of two regular relations R_1, R_2 denotes the set of string pairs (S_1,S_3) such that (S_1,S_2) $\in R_1$ and (S_2,S_3) $\in R_2$ and is again a regular relation.

Composition(T_1, T_2)

make(I_{T1},I_{T2}) initial in $T_1 \oplus T_2$
make(F_{A1},F_{A2}) final in $T_1 \oplus T_2$
for each arc from u to v in T_1 labeled M_1/M_2
    for each arc from x to z in T_2 labeled M_2/M_3
        add an arc labeled M_1/M_3 from (u,x) to (v,z) to $T_1 \oplus T_2$

Left_Restriction of a finite-state transducer T to a finite-state automaton A ($T \cap_{LEFT} A$)

Left_Restriction of a regular relation RR and a regular set RS denotes the set of string pairs (S_1, S_2) such that $S_1 \in$ RS and (S_1,S_2) $\in$ RR and is itself a regular relation. Right_Restriction ($T \cap_{RIGHT} A$) is defined analogously

Left_Restriction(T,A)

```
make(IA,IT) initial in T ∩LEFT A
make(FA,FT) final in T ∩LEFT A
for each arc from u to v in A labeled M
        for each arc from x to z in T labeled M/P
                add an arc labeled M/P from (u,x) to (v,z) to T ∩LEFT A
```

Left_Language of a finite-state transducer A (Left_Lang(A))

The Left_Language of a regular relation RR denotes the (regular) set of strings S_1 such that a string pair (S_1, S_2) $\in$ RR, for some S_2. Right_Language(RR) is defined analogously.

Left_Language(T,A)

```
makeIA initial in Left_Lang(A)
make FT final in Left_Lang(A)
for each arc from u to v in A labeled M/P
        add an arc labeled M from u to v to Left_Lang(A)
```

Ellison's Algorithm

takes a finite-state transducer T with Right_Lang(T) $\subseteq \{0,1\}^*$ and produces a finite-state transducer (regular relation) T' $\subseteq$ T containing all string pairs SP_1 such that there's no string pair $SP_2 \in T$ for which value(SP_2) < value(SP_1), where the value of a string pair value(S, N_1,...N_n) = $\Sigma N_{1\text{-}n}$.

LabelNodes(transducer)

```
for each state n in transducer
        harmony(n) undefined
harmony(I) ⇐ 00...0, I is the initial state
list ⇐ [I]
while list is not empty
        expand m begins
        m ⇐ most harmonic state in list
        delete m from list
        for each arc a:m → n from m
                if harmony(n) < harmony(m) + harmony(a)
                        delete n from list
                        harmony(n) ⇐ harmony(m) + harmony(a)
                        insert n in list
                else if harmony(n) undefined
                        harmony(n) ⇐ harmony(m) + harmony(a)
                        insert n in list
```

Prune(Transducer)

```
for each arc a:n→ m of transducer
        if harmony(a) + harmony(n) < harmony(m)
                then delete a
```

Optimality Theory using Finite-State-Transducers(Ellison 1994)

Candidates: ((b)a*)+
Constraints: !Onset, *Segment

Ranking1: !Onset >> *Segment

	!Onset	*Segment
Fba		**
a	*!	*

Ranking2: *Segment >> !Onset

	*Segment	!Onset
Fa	*	*
ba	**!	

As Regular Relations:

***Segment:**

{1,	1	}*
{a,	b	}

!Onset (1.Vs.):

{1,	0	}	{0,	0	}*
{a,	b	}	{a,	b	}

!Onset (2.Vs.):

(1)*	(	(0)+	(0	(1)*	)?	)*
(a)	(	(b)	(a	(a)	)	)

Evaluation

Given a constraint ranking R = $C_1, ..., C_n$, a regular candidate set S and a constraint transducer C:

Simple_Optimize(S, C) = Left_Lang(Prune(Left_Restriction(S,C))

Ranked_Optimize(S, R) = S, if |R| = 0

else

Ranked_Optimize(S, R) = Simple_Optimize(Ranked_Optimize(S, R')),
where R' = $C_1,...C_{n-1}$

Optimality Theory using Finite-State-Automata(Karttunen 199?)

!Onset : (b a)*

***Segment(0):** ε
***Segment(1):** {a, b}?
***Segment(2):** ({a, b}? {a, b})?

•
•
•

Evaluation

Given a constraint ranking $R = C_1,...C_n$, a regular candidate set S and a constraint automaton C:

Simple_Optimize(S, C) = $S \cap C$, if $S \cap C \neq \emptyset$

else

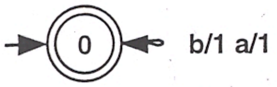
Simple_Optimize(S, C) = S

Ranked_Optimize(S, R) = S, if |R| = 0

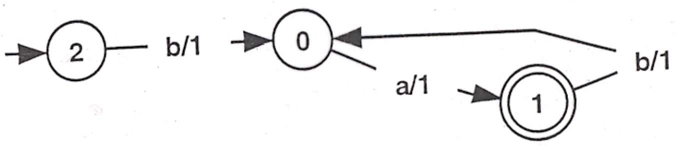
else

Ranked_Optimize(S, R) = Simple_Optimize(Ranked_Optimize(S, R')),
where $R' = C_1,...C_{n-1}$

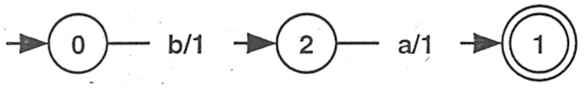
6 NoSegment



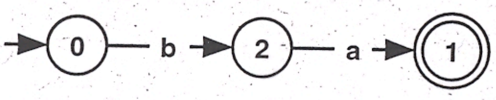
7 Left_Restriction(5,6)



8 Prune(7)

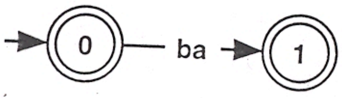


9 Left_Language(8)

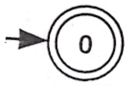


1 Ellison(94)

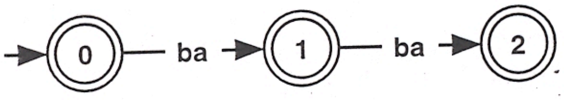
6 NoSegment(1)



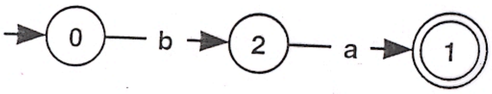
7 Intersection(3,6)



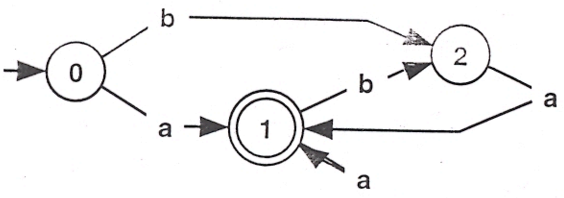
8 NoSegment(2)



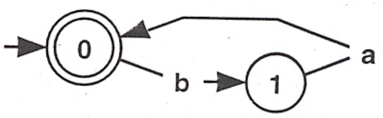
9 Intersection(3,8)



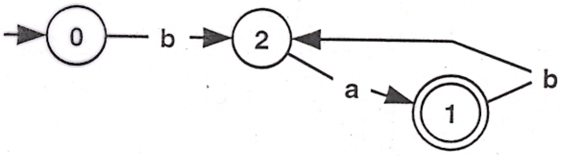
1 Candidate Set



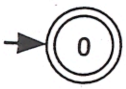
2 Onset



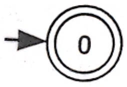
3 Intersection(1,2)



4 NoSegment(0)

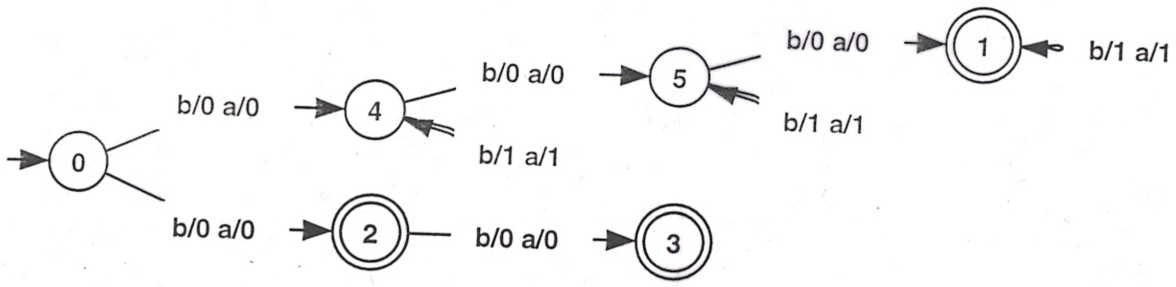


5 Intersection(3,4)

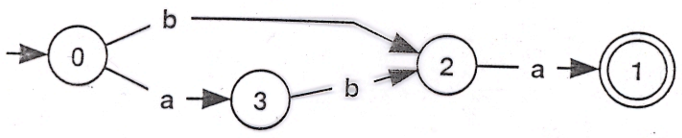


7 NoSegment

8 Composition(6,7)



9 7(1)



3 DOT:DEP

Direct Optimality Theory (Golston 1996)

Some Lexical Representations

/ba/	!Onset	NoCoda	NoSyl
			*

/a/	!Onset	NoCoda	NoSyl
	*		(*)

/ab/	!Onset	NoCoda	NoSyl
	*	*	(*)

Parse-Constraints

a	!Onset	Parse(Onset)	NoCoda
a	*!		
ab	*!		*
Fba		*	

a	Parse(Onset)	Onset	NoCoda
Fa		*	
ab		*	*!
ba	*!		

DEP-Constraints

/baba/	NoSyl	FootBin
	**	

/ba/	NoSyl	FootBin
	*	*

/ba/	FootBin	Dep(NoSyl)	NoSyl
ba	*!		*
Fbaba		*	**

/ba/	Dep(NoSyl)	FootBin	NoSyl
Fba			*
baba	*!	*	**

Generalized Alignment

	ALIGN COR	**ALIGN DOR**
cat	*	
tack		*
act	*	*

ALIGN(X)

X	XX	XXX	XXXX	XXXXX
0	1	3	6	10

Implementing Parse-Constraints

X	!Onset
	***

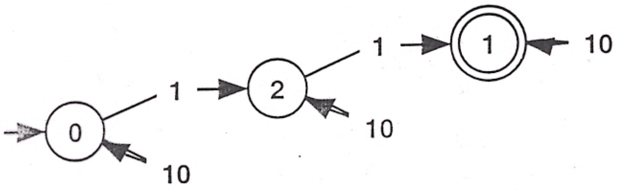
Parse(3): [01]* 1 0* 1 0* 1 [01]* ([01]* (1 0*){2} 1 [01]*)
Parse(2): [01]* 1 0* 1 [01]* ([01]* (1 0*){1} 1 [01]*)
Parse(1): [01]* 1 [01]* ([01]* (1 0*){0} 1 [01]*)

When C is a Constraint in Ellison Format and the actual lexical representation LR contains n violations of C then the set of optimal candidates w.r.t. Parse(C) from the actual candidate set CS_{akt} (a regular set) CS_{opt} (LR, CS_{act} , Parse(C)) is achieved as follows:

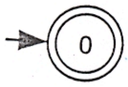
$ECS_{act} \Leftarrow C \cap_{Left} CS_{act}$

for(i from n to 0)
    if($CS_{opt} \Leftarrow$ (([01]* (1 0*){i} 1 [01]*) $\cap_{Right}$ ECS_{act}) $\neq \varepsilon$
        break

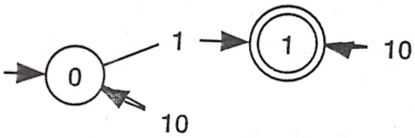
6 Parse(2)



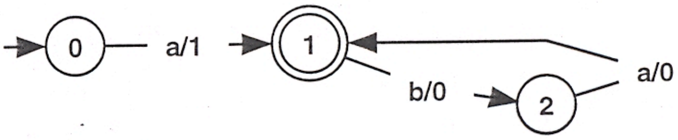
7 Right_Restriction(6,3)



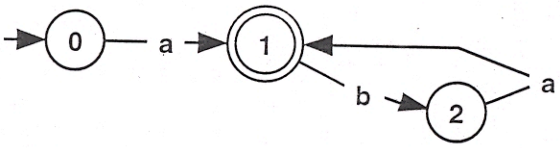
8 Parse(1)



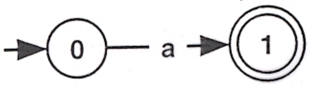
9 Right_Restriction(8,3)



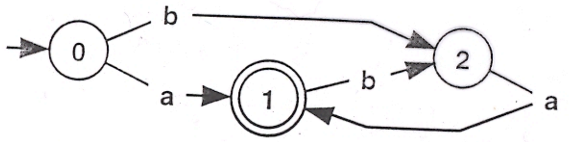
10 Left_Lang(9)



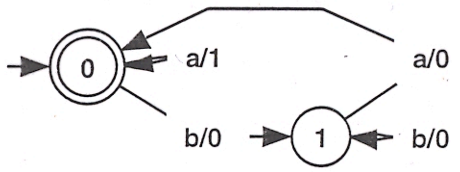
11 NoSegment(10)



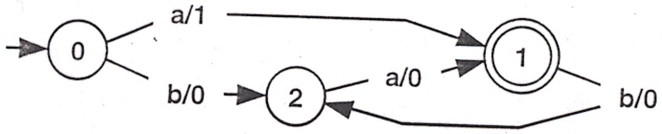
1 Candidate Set



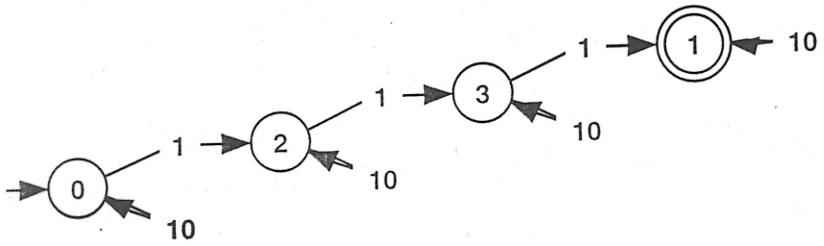
2 Onset



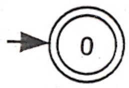
3 Right_Restriction(1,2)



4 Parse(3)



5 Right_Restriction(4,3)



4 DOT:PARSE

Implementing DEP-Constraints

Less than 3 violations

0* 1 0*	0* (1 0*){1}	
0* 1 0* 1 0*	0* (1 0*){2}	
		0* (0 0*){1-2}
or:	0* (1 0*){1-2}	0 (1 0)

At least 3 violations

[01]*(1 0*){3}[01]*

[01]* 0 [01]* 0 [01]* 0 [01]*	[01]* (0 [01]*){3}
[01]* 1 [01]*1 [01]* 1 [01]*	[01]* (1 [01]*)

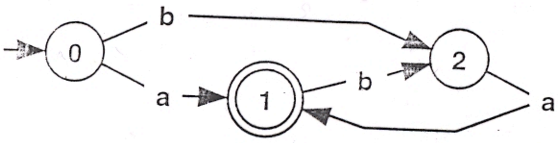
When C is a Constraint in Ellison Format and the actual lexical representation LR contains n violations of C then the set of optimal candidates w.r.t. DEP(C) from the actual candidate set CS_{akt} (a regular set) CS_{opt} (LR, CS_{akt} , DEP(C)) is achieved as follows:

Filter ⇐	0* (0 0*) {0-(n-1)}		[01]* (0 [01]*){n}
	0* (1 0*)	∪	[01]* (1 [01]*)

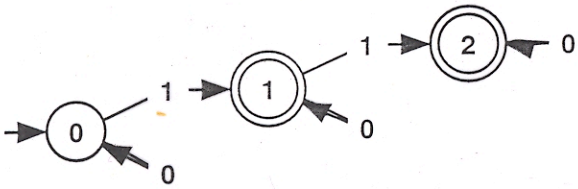
Dep ⇐ C ⊕ Filter

CS_{opt} ⇐ Ellisons Algorithm(Dep, CS_{akt})

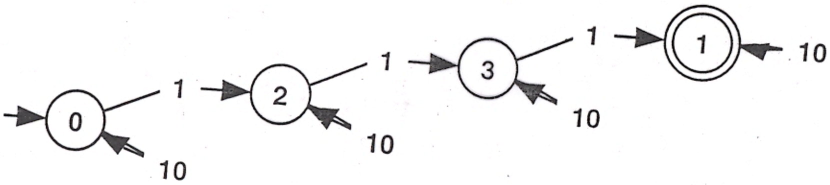
1 Candidate Set



2 Less_than_three(Automaton)

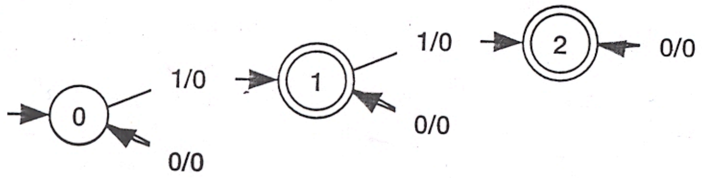


3 At_least_three(Automaton)

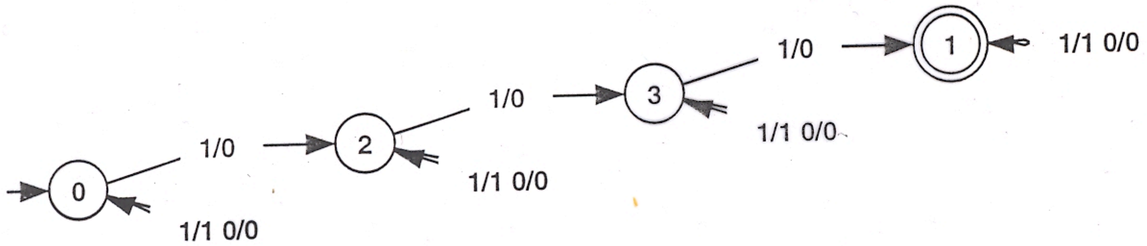


3 DOT:DEP

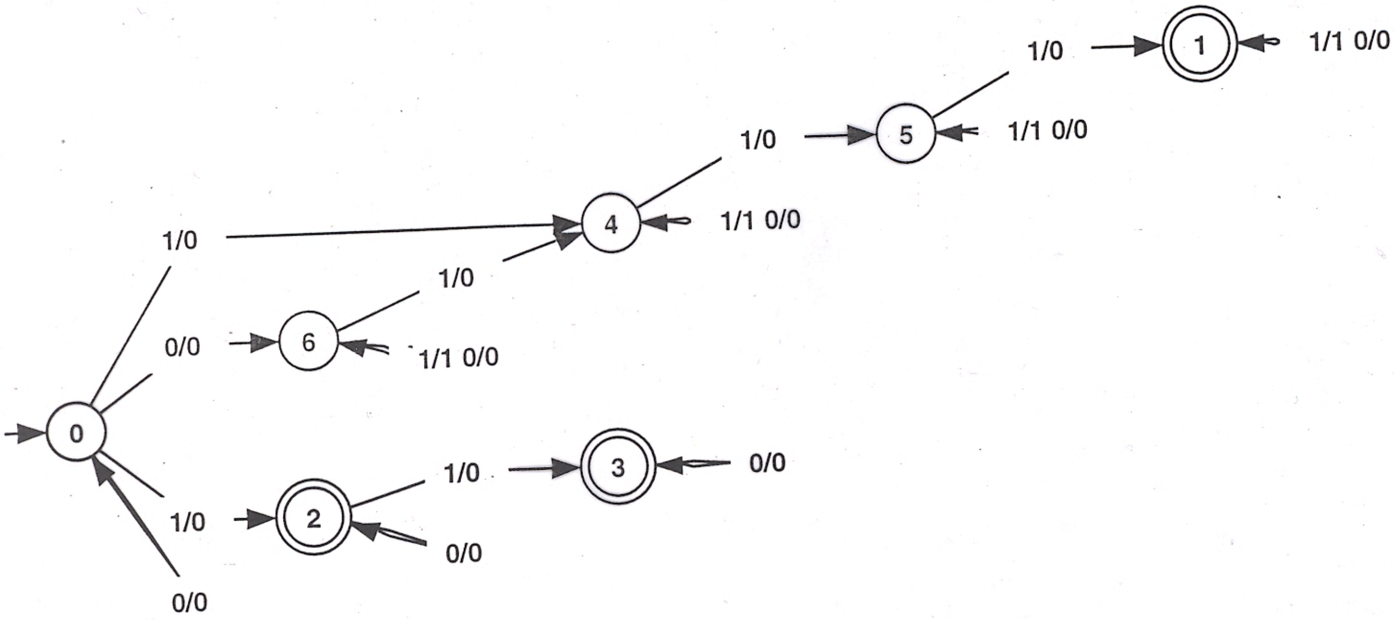
4 Less_than_three(Transducer)



5 At_least_three(Transducer)



6 Union(4,5)



Implementing GA-Constraints

Problem: GA-Constraints are not finite-state
(i.e. cannot be represented as Ellison-style transducers)

					Sum
ALIGN(X):	1	1	1	1	4
	X	X	X	X	
		1	2	3	5

Strategy:

- A method is given for finding at least one optimal candidate string S from the candidate set.
- The set of all strings inducing at most as many violations as S is constructed as an automaton A.
- A is intersected with the candidate automaton.

Preliminaries

Notation

A GA-Constraint can be characterized by a 3-tuple (Targets, Measures, Fillers)

where

> Measures is the set of symbols that count as distance measures
> Targets is the set of symbols whose distance to the left edge is measured
> Fillers is the set of all other symbols

Substrings

S_2 is a reduced string of S_1 iff S_3^S_4^S_5 = S_1 $\wedge$ S_3^S_5 = S_2, for any S_3, S_4, S_5
S_3 is a substring of S_1 iff S_3 is a reduced string of S_1 orS_3 is a substring of S_2 and S_2 a substring of S_1

Optimality

The optimality of a string is Y iff Opt(S) = (Y,Z)

Opt(ε) = (0,0)

Opt(String^Symbol) = (ActualViolations, ActualMeasures)
iff Opt(String) = (ActualViolations, ActualMeasures)
and Symbol is neither a Measure nor a target.

Opt(String^Symbol) = (ActualViolations, ActualMeasures+1)
iff Opt(String) = (ActualViolations, ActualMeasures)
and Symbol is a Measure.

Opt(String^Symbol) = (ActualViolations+ActualMeasures, ActualMeasures)
iff Opt(String) = (ActualViolations, ActualMeasures)
and Symbol is neither a target.

A string S_1 is less optimal than S_2 iff optimality(S_1) > optimality(S_2)

Finding an optimal candidate

- By removing stars a finite subset of the candidate set is created, which is shown to contain at least one optimal candidate..
- The (set of) optimal candidate(s) is computed by a variant of Ellison's algorithm (or brute force).

Removing Stars

If RE is an symbol from the alphabet or ε **then** Remove(RE) is RE

If RE is N* **then** Remove(RE) is ε.

If RE is $\{N_1, ..., N_n\}$ **then** Remove(RE) is $\{(\text{Remove}(N_1)), ..., \text{Remove}(N_n))\}$

If RE is N_1^...^N_n **then** Remove(RE) is $(\text{Remove}(N_1))$^...^$(\text{Remove}(N_n))$

Guaranteeing an optimal candidate

If RE_2 = Remove(RE_1) the following holds:

Stringset(RE_2) $\subseteq$ Stringset(RE_1)

$\forall X \in$ Stringset(RE_1) $\exists$ $Y \in$ Stringset(RE_2) Substring(Y, X)

Since: No substring of S is less optimal than S.

$\Rightarrow$ Stringset(Remove(RE_1)) contains at least one optimal string pair from Stringset(RE_1).

Finding all optimal candidates

If the optimality of a candidate set Cs (a regular set) w.r.t a GA-Constraint GAC is N the set of optimal candidates OC $\subseteq$ GAC is computed as follows

O $\Leftarrow$ Construct(N, Targets, Measures, Fillers)

OC $\Leftarrow$ O $\cap$ GAC

Construct(N, Targets, Measures,Fillers)

Generate an automaton A with start state S_0

Optimal_Automaton(S_0, N, 0, 0)

Add_Loops(A)

Add_Loops(Automaton)

Add a transition I—T→I for the start state I and each target symbol T.

Add a transition S—M→S for each state S without outgoing arc and each measure symbol M.

Add a transition S—F→S for each state S and each filler symbol F.

Optimal_Automaton (State, AllViolations, ActualViolations, ActualMeasures)

if(ActualViolations + ActualMeasures $\leq$ AllViolations)

if(ActualViolations < AllViolations) and (ActualViolations $\neq$ 0)
generate a new final state N_1
generate a transition State—M→N_1 for each violation measure
Construct(N_1, AllViolations, ActualViolations, ActualMeasures+1)

if(ActualMeasures < AllViolations)
generate a new final state N_2
generate a transition State—M→N_2 for each violation measure
Construct(N_2, AllViolations, ActualViolations+ActualMeasures,ActualMeasures)

An example derivation

Candidates: ((baba+)(y*a)*) = ((baba)(baba)*(y*a)*)

Constraint: Vowels shouldn‘t be seperated from the loeft edge by consonants.

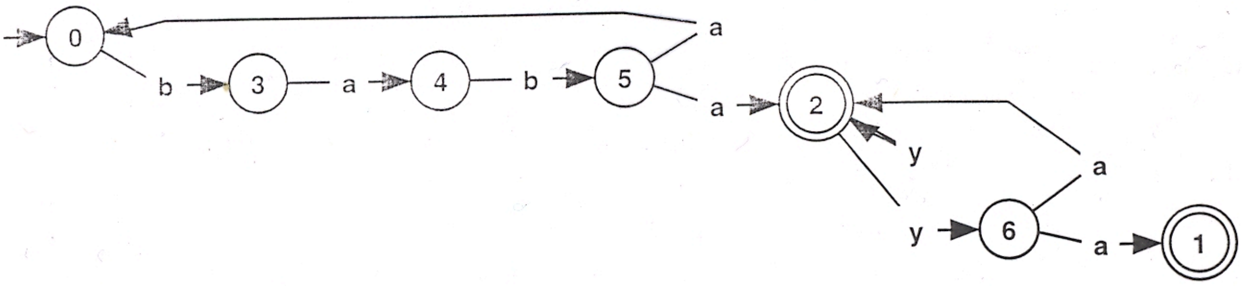
Targets = {a}
Measures = {b}
Fillers = {y}

Remove(Candidates) = baba
Violations(ba) = 3

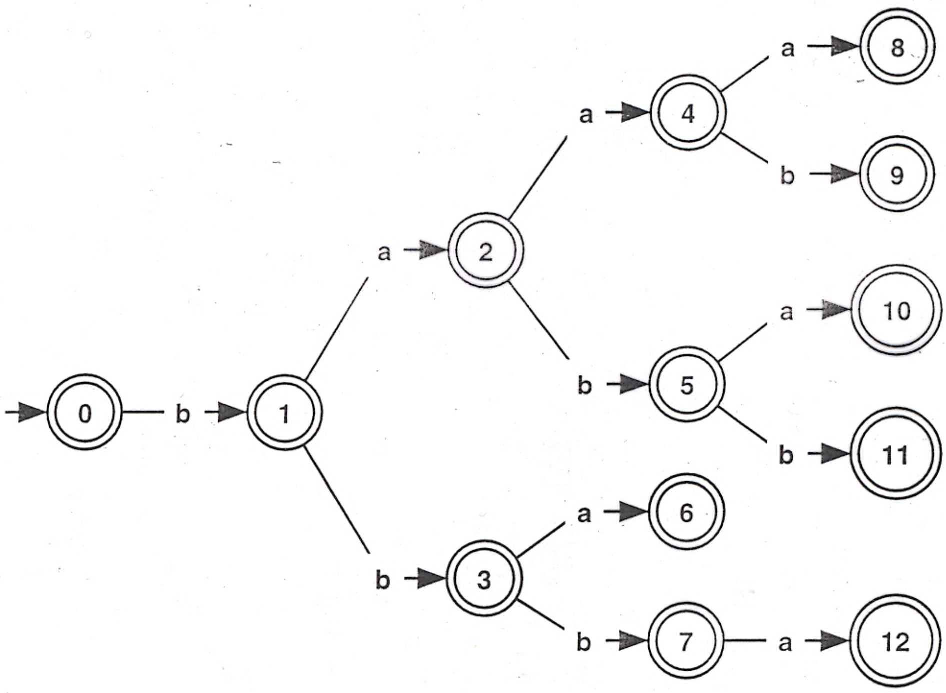
Optimal_Automaton

State	Measures	Violations	Sum
0	0 Ü 1	0M	0
1	1Ü 3	0 Ü 2	1
2	1Ü 5	1Ü 4	2
3	2Ü7	0 Ü 6	2
4	1Ü 9	2 Ü 8	3
5	2 Ü 11	1Ü10	3
6	2	2	4M
7	3M	0Ü 12	3
8	1	3M	4M
9	2	2	4M
10	2	3M	5M
11	3M	1	4M
12	3M	3M	6M

1 Candidate Set

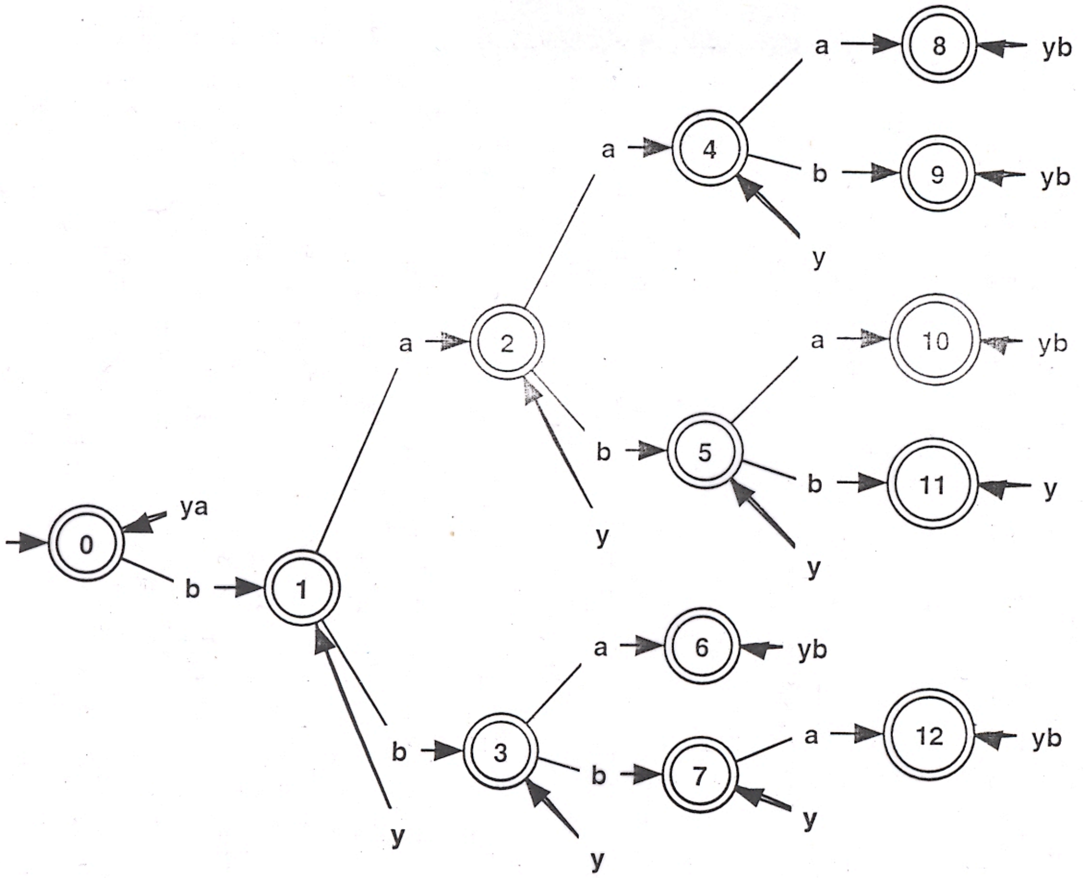


2 Optimal_Automaton



5 DOT:GA

3 Add_Loops



4 Result