Efficient NORMAL−FORM Parsing

for Combinatory Categorial Grammar

Jason M. Eisner

University of Pennsylvania

June 26, 1996 at ACL

CCG and the Spurious Ambiguity Problem

[John likes Mary]	**S**	**(sentence)**
John **[likes Mary]**	**S\NP**	**(sentence missing NP to its left – "\")**
[John likes] Mary	**S/NP**	**(sentence missing NP to its right – "/")**

... can conjoin this with other predicates

[John likes], and [Sue hates], that woman in the hat

... can ask who satisfies it

Who does [John like]?

... can state who satisfies it

It is MARY that [John likes]. / *[John likes] MARY.*

CCG allows linguistically useful extra constituents ...

CCG and the Spurious Ambiguity Problem

Two parses for an unambiguous sentence:

[[John likes] Mary]	(non−standard parse)
[John [likes Mary]]	(standard parse)

the [aide in the] Senate [that D'Amato says Clinton tried to] bribe

... but CCG forces hundreds of extra parses on us.

Today's Talk

– Sketch of CCG formalism
  + the B combinators

– A solution to spurious ambiguity

– Why the solution works (formal intuitions)

– Important extensions of the solution
  + the S combinator (straightforward)
  + the T combinator (work in progress)
  + restrictions on the rules

 Phrase Structure

forward rules

>B0:	A/B	B	→	A
>B1:	A/B	B/C	→	A/C
	A/B	B\C	→	A\C
>B2:	A/B	B/C/D	→	A/C/D
	A/B	B/C\D	→	A/C\D
	A/B	B\C/D	→	A\C/D
	A/B	B\C\D	→	A\C\D

etc.

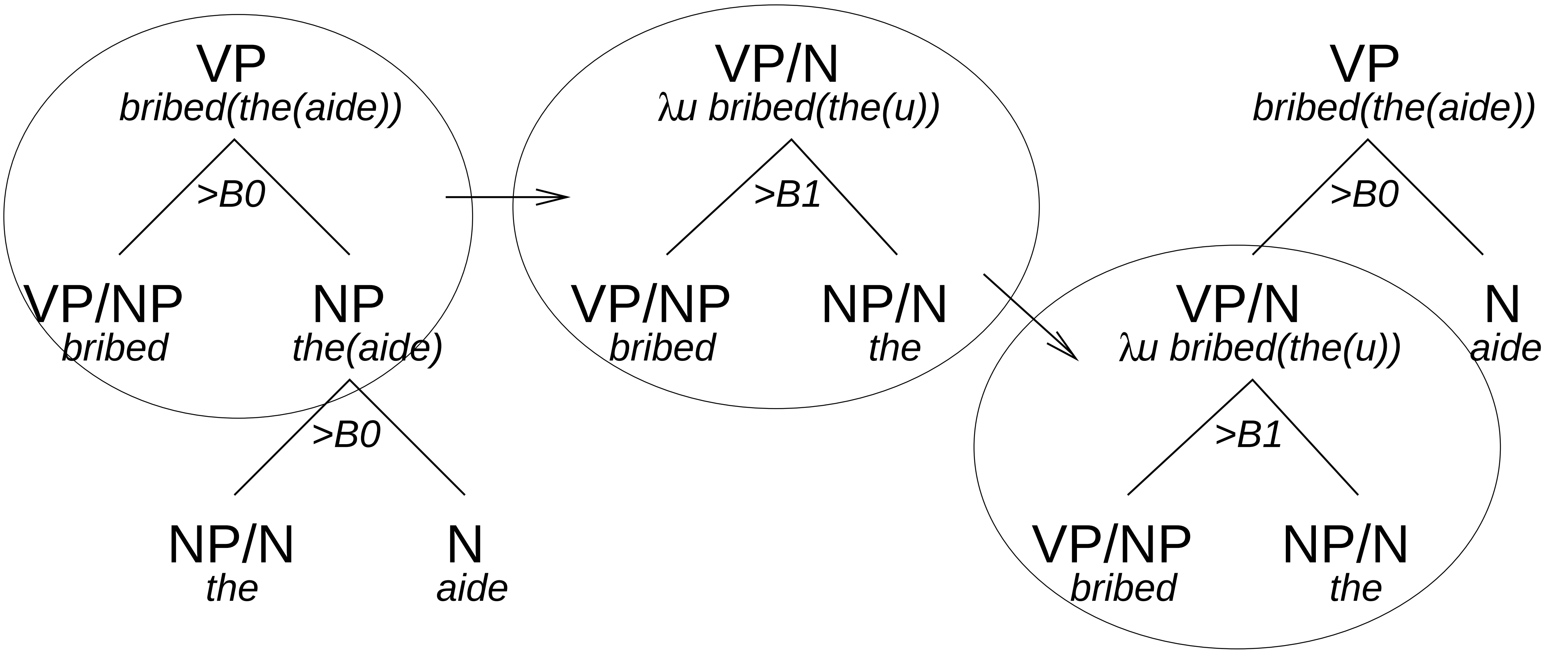
backward rules

<B0:	B	A\B	→	A
<B1:	B\C	A\B	→	A\C
	B/C	A\B	→	A/C
<B2:	B\C\D	A\B	→	A\C\D
	B\C/D	A\B	→	A\C/D
	B/C\D	A\B	→	A/C\D
	B/C/D	A\B	→	A/C/D

etc.

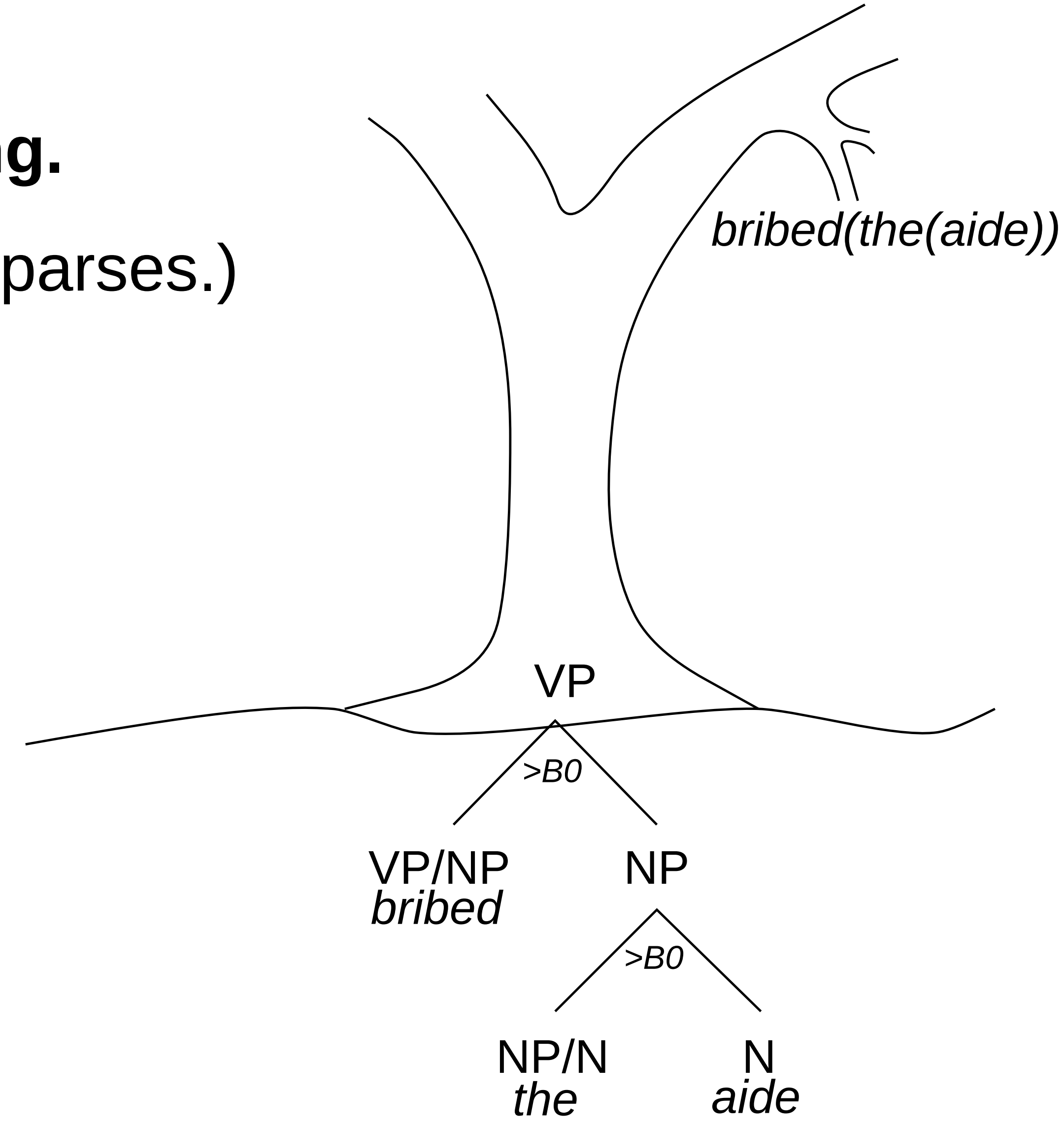
Sketch of CCG Formalism: **Example**

>B0:	A/B *f*	B *x*	$\rightarrow$	A *f(x)*
>B1:	A/B *f*	B/C *g*	$\rightarrow$	A/C *λu f(g(u))*



Exactly one parse per reading.

(Efficiently suppress all other parses.)



How can we rule out extra parses?

Yes, allow all of CCG's non−standard constituents,

both when useful *[D'Amato said Clinton tried],*
and [maybe he said she failed], to bribe that aide.

and when useless. *[D'Amato said Clinton tried] to bribe that aide.*

BUT:

1 parse not 5 **1 parse not 5**

[[D'Amato said Clinton tried] [to bribe that aide]]

assemble 1 parse not 25

and in this case, disallow even that 1 parse!

(but do allow: [[D'Amato] [said Clinton tried to bribe that aide]])

A Solution to Spurious Ambiguity: **The Tactics**

Standard kind of spurious ambiguity:

Forward (or backward) "chains"

VP/NP NP/N N

2 parses

A/A A/B B\C/D/E E/F F\G

14 parses

The OUTPUT of forward composition (>B1, >B2, >B3, ...)

may not be the primary (left) INPUT to any forward rule.

(>B0, >B1, >B2, >B3 ...)

The OUTPUT of backward composition (>B1, >B2, >B3, ...)

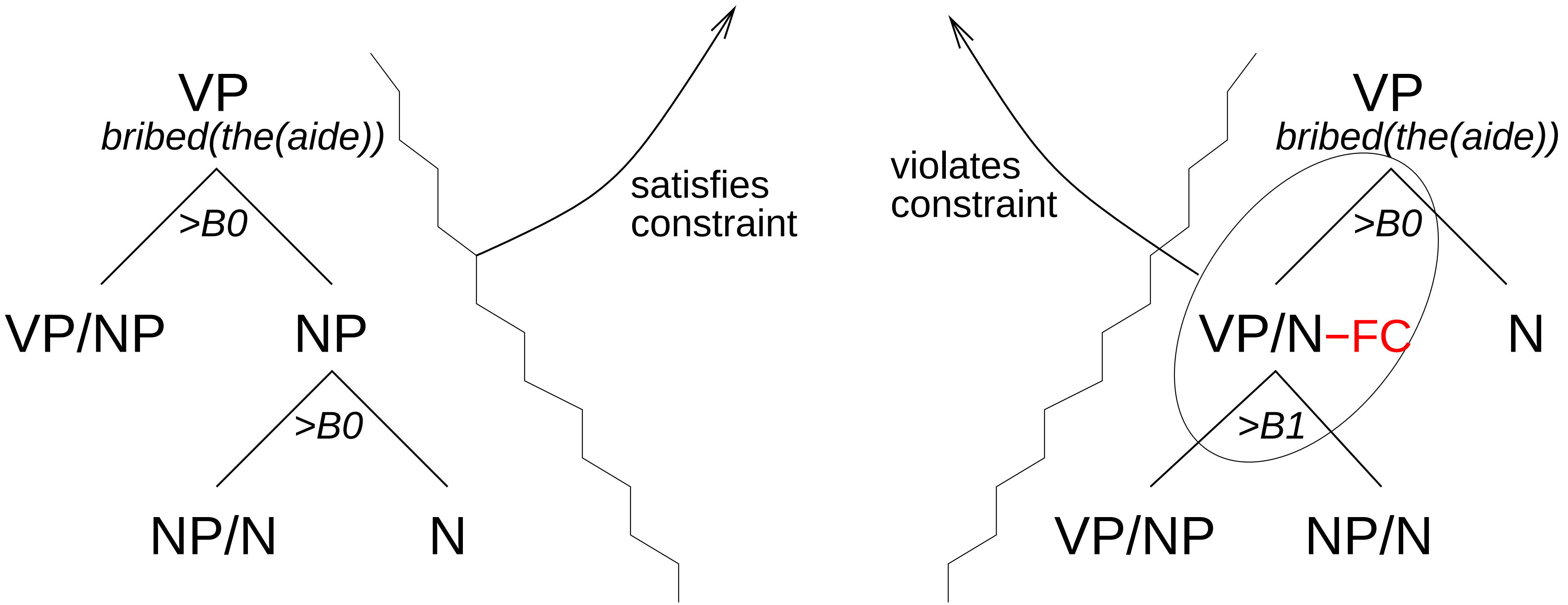
may not be the primary (right) INPUT to any backward rule.

(>B0, >B1, >B2, >B3 ...)

A Solution to Spurious Ambiguity: **The Tactics in Action**

The OUTPUT of forward composition (>B1, >B2, >B3, ...)

may not be the primary (left) INPUT to any forward rule.
(>B0, >B1, >B2, >B3 ...)



(a "normal−form" tree)

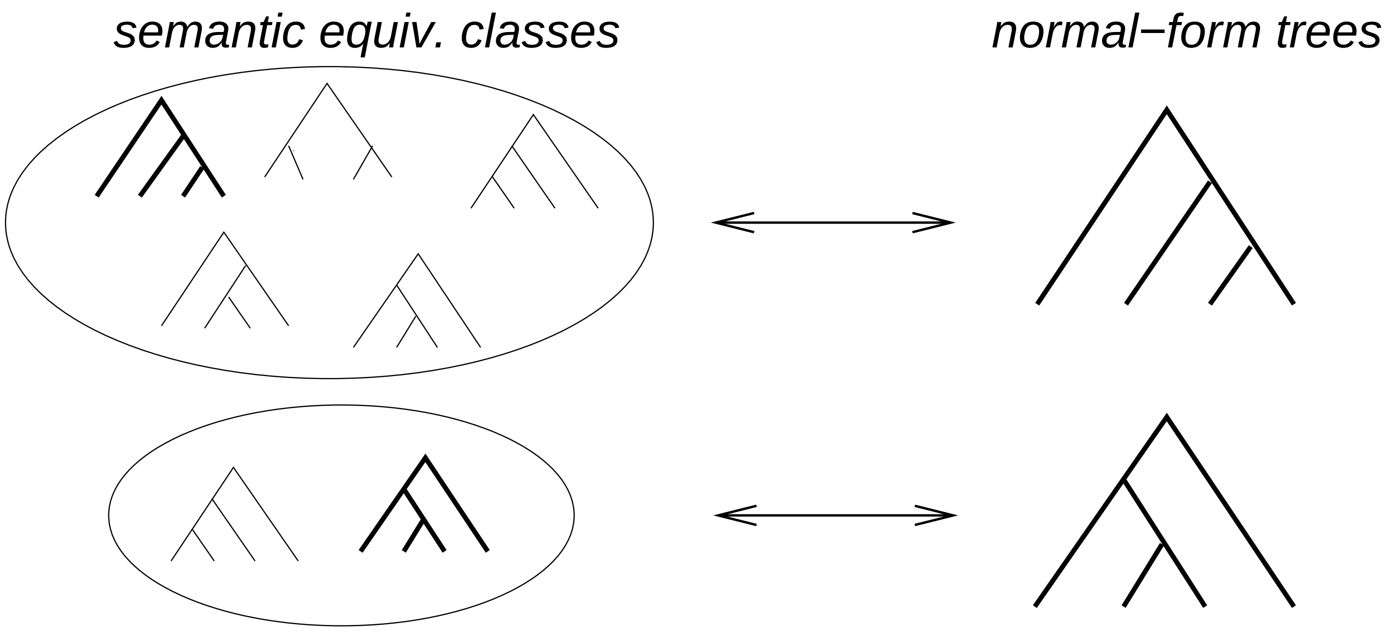
A Solution to Spurious Ambiguity: **The Result**

For CCG with the generalized composition rules (including mixed), these tactics

(1) eliminate ONLY spurious ambiguity (safety)

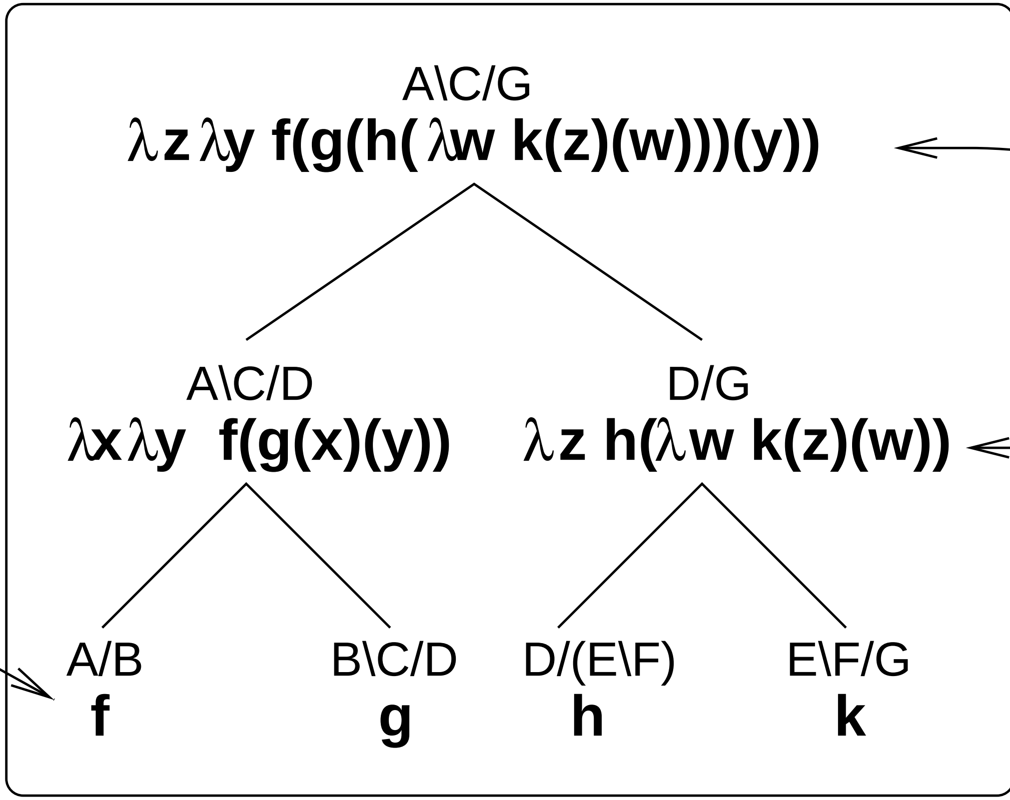
(2) eliminate ALL spurious ambiguity (completeness)

1−1 correspondence:



Formal Intuitions: **What is Spurious Ambiguity?**

A syntax tree takes interps of the words,



and combines them semantically

into an interp. of the phrase.

So a syntax tree on n words
computes an n−ary function: $\lambda f \lambda g \lambda h \lambda k\ (\lambda z \lambda y\ f(g(h(\lambda w\ k(z)(w)))(y)))$

Two trees on the same n words are semantically equivalent
iff they compute the same n−ary semantic function.

Formal Intuitions: **What is Spurious Ambiguity?**

Two trees on the same n words are semantically equivalent iff they compute the same n−ary semantic function.

What this definition is NOT:

(1) Does this mean "iff they compute the same lambda−term"?

(2) Do we eliminate one parse from each of these pairs?

[quietly [knock twice]]

[[quietly knock] twice]

denote
same action

[π equals [[2 plus 3] over 4]]

[π equals [2 plus [3 over 4]]]

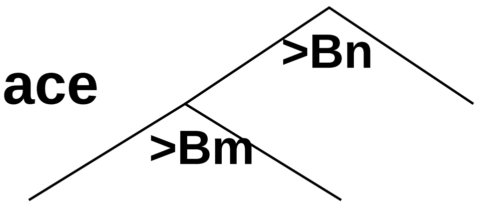
denote same
truth value ("false")

Formal Intuitions: **Existence Theorem**

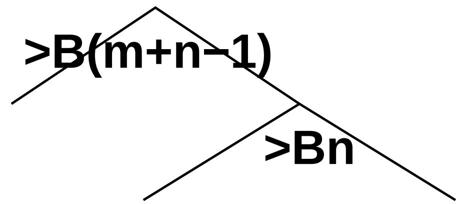
Theorem. **For every tree T we cut down with our constraints, we leave standing a semantically equivalent tree, NF(T).**

Proof. **To construct NF(T) from T, essentially**

replace



throughout with



Construction used is inductive.

Takes O(1) time, if NF(T’) is known for T’ smaller than T.

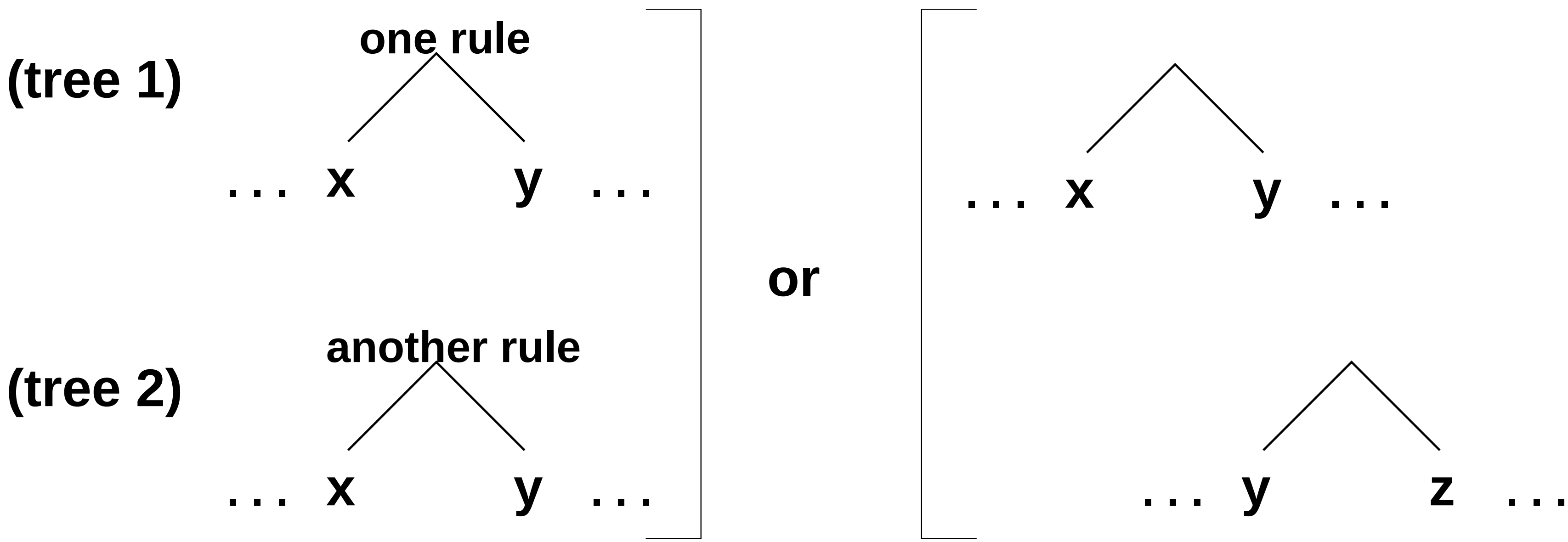
Formal Intuitions: **Uniqueness Theorem**

Theorem. **We never leave two equivalent trees standing.**

Proof. **Given two distinct trees that we keep.**

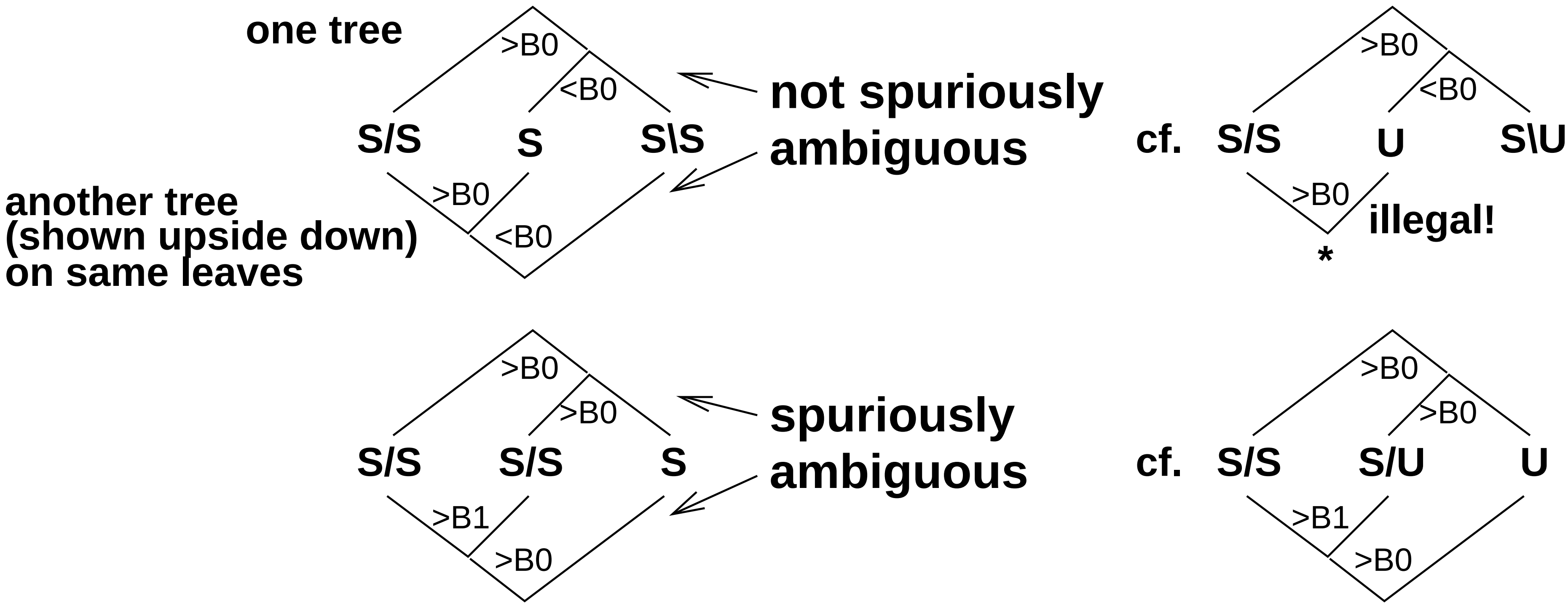
They must differ somewhere syntactically:

so contain either



Show that they differ semantically as a result.

Formal Intuitions: **The Spurious Ambiguity Lemma**



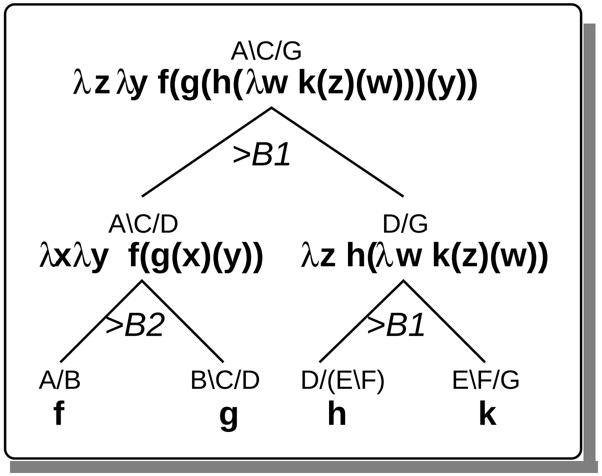
2 parses on the same sequence of words are spuriously ambiguous ...

Def. **... iff spuriosity is robust under changes to words' semantics.**

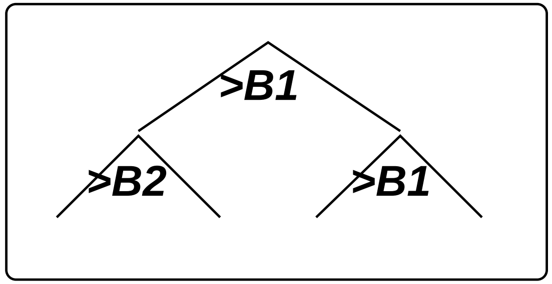
Equiv def. **... iff ambiguity is robust under changes to words' syntax.**

Easy syntactic characterization of a semantic property!

Formal Intuitions: **Proof of Spurious Ambig. Lemma**



no−category syntax tree



restricted combinator

λf λg λh λk (λz λy f(g(h(λw k(z)(w)))(y)))

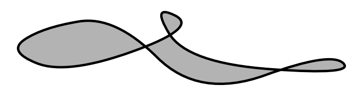
injective   *injective*

most general polymorphic type

(B → A) → (D → C → B) → (X → D) → (G → X) → (G → C → A)

can write as (A|C|G) | (X|G) | (D|X) | (B|C|D) | (A|B)

n−ary function in model



Extensions: **The S and T combinators**

If we add the S (substitution) combinator, we need a new restriction:

Just as

> **The OUTPUT of (>B1, >B2, >B3, ...)**
> **may not be the primary (left) INPUT to (>B0, >B1, >B2, >B3, ...)**

now

> **The OUTPUT of (>B2, >B3, ...)**
> **may not be the primary (left) INPUT to >S**

If we add the T (type−raising) combinator,

the ambiguities get much trickier! Work in progress.

Extensions: **Making TR visible to the grammar**

If type−raising is only lexical,

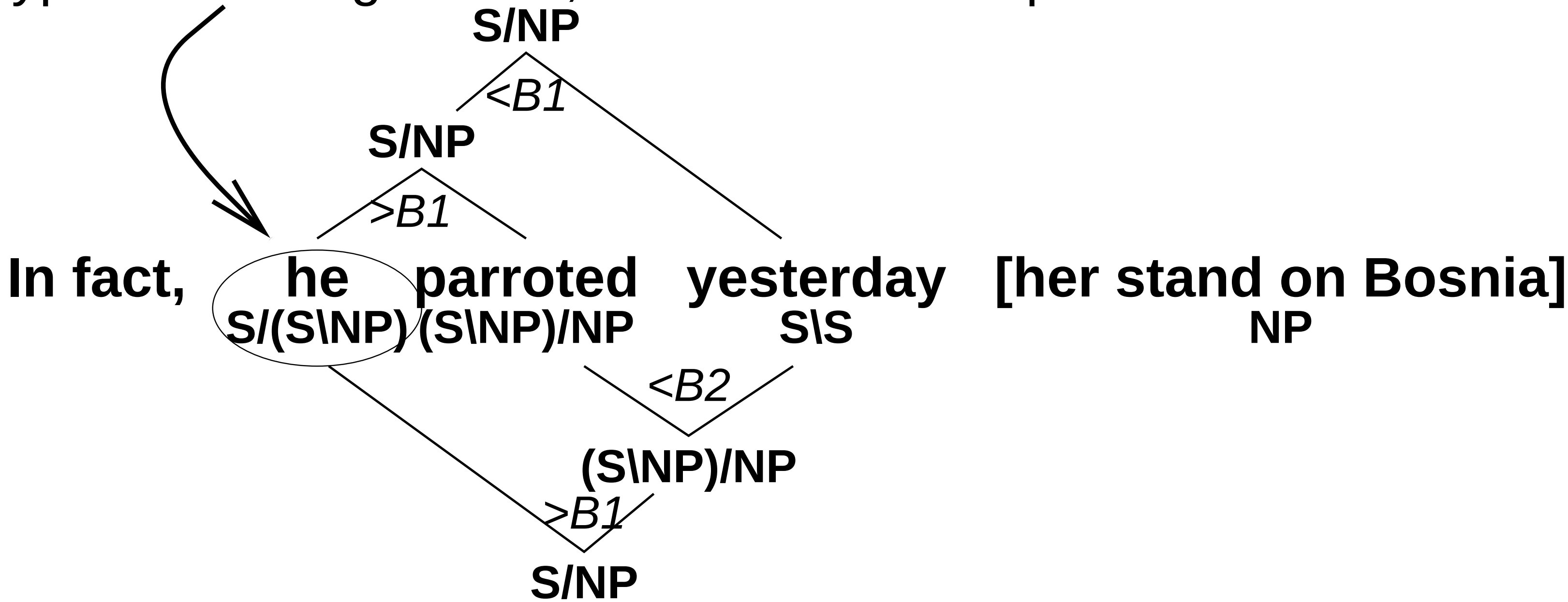
our definition can't see this ambiguity:

John	**likes**	**Mary**	**John**	**likes**	**Mary**
NP	**(S\NP)/NP**	**NP**	**S/(S\NP)**	**(S\NP)/NP**	**NP**

parses of different sentences!

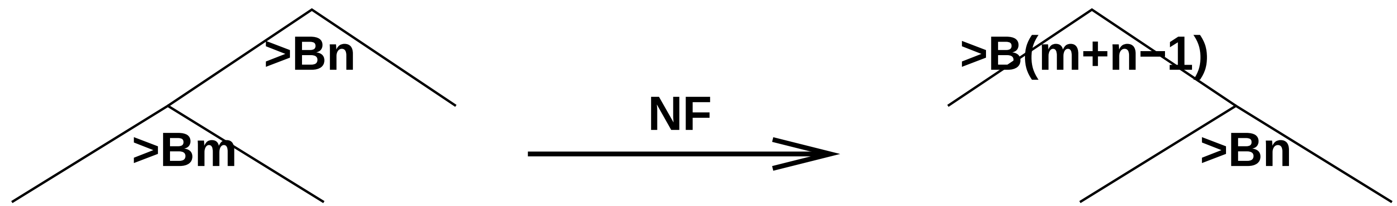
and the ambiguity below depends on funny "lexical" properties

of type−raised arguments, so doesn't look spurious:



Extensions: **Restrictions on CCG rules**

In practice, a CCG grammar may state WHICH rules can apply, & WHEN.



allowed by CCG
but skipped by parser

allowed by CCG?
if not, we're in trouble.

Solution:

Don't change the theorems, change the parser!

Karttunen 1986: **No constraints on parses. Whenever we find a new parse of a constituent, check that it's not redundant.**

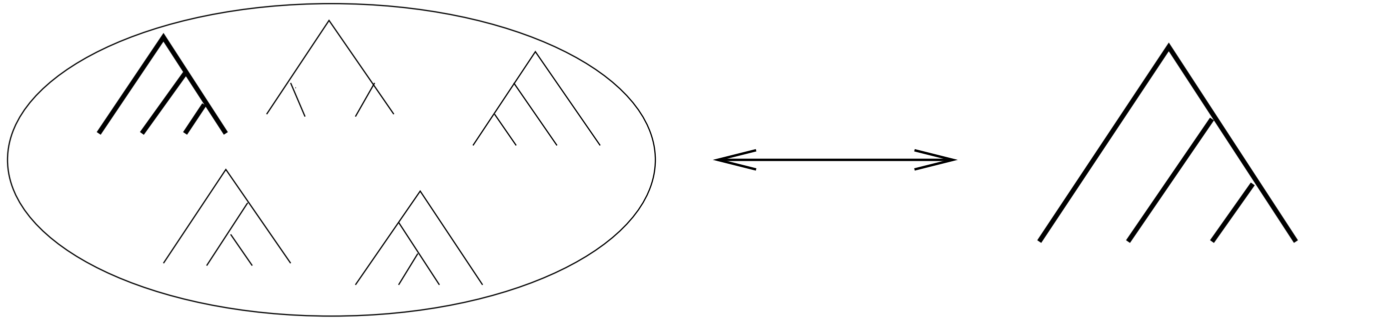
But checking new parse against old parses takes exponential time.

New idea: **See if its NF matches an old parse's. Can do in O(1) time.**

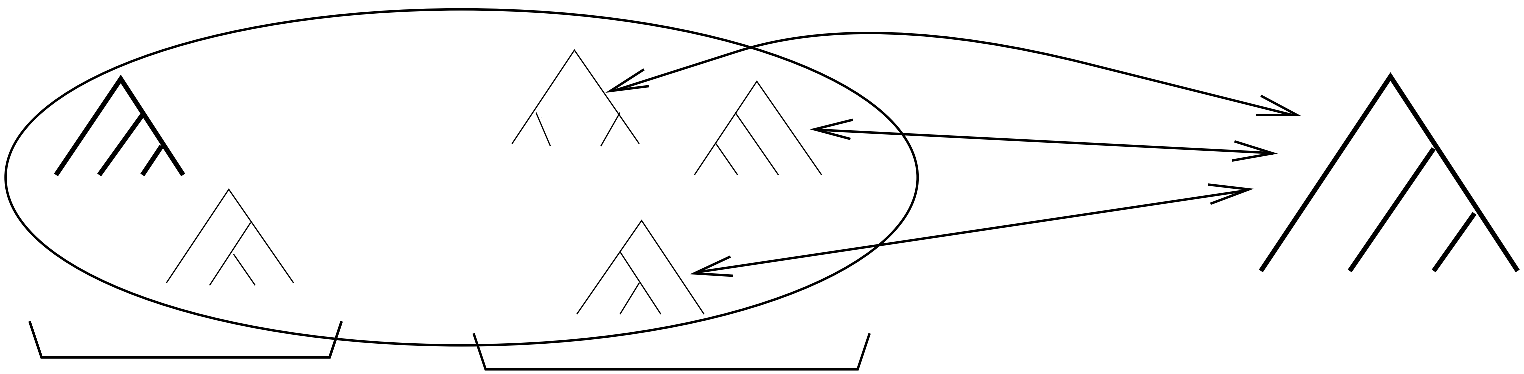
Extensions: **Finding Equiv Classes instead of NFs**

Have proved 1−1 correspondence:

semantic equiv. classes *normal−form trees*



So use each NF tree as a magnet for its equivalence class:



not found by parser

(disallowed by grammar, or conflict with prior "incremental" commitments)

keep just one of these legal parses − e.g. the first, or the best according to prosody or discourse module

Summary of Results

+ A useful model−theoretic definition of spurious ambiguity

. . . and a lemma giving a syntactic test for it.

+ Easy, fast parser for CCG with the B and S rules.

Simple constraints provably eliminate all spurious ambiguity.

+ Fast parser still possible if grammar rules have nasty restrictions:

Rapidly group legal (sub)trees by semantic equivalence class −

just have each NF tree point to the legal trees in its class.