

Minimalist Parsing: simplicity and feature unification

Sandiway Fong

Departments of Linguistics and Computer Science

University of Arizona

Tucson AZ USA

Conference on Language and Recursion. University of Mons, Belgium. March 14-16th 2011

Linguistic Theory

Computation and Linguistic Theory

- Principles and Parameters Approach (Chomsky 1980)
 - Linguistic principle = declarative statement:
 - Noun phrases must receive (abstract) Case (*Case Filter*)
 - An anaphor must be A-bound in its Governing Category (*Binding Principle A*)
- Minimalist Program (e.g. Chomsky 2001)
 - Precisely specified sequence of (feature-driven) operations
 - Merge: external and internal (= Move)
 - Agree: Probe and Goal
(interaction between uninterpretable and interpretable features)

Principles and Parameters Approach

Principles-and-Parameters Parser

Examples ...

[154] minswu-nun younghee-ka mwues-ul mekESTA-ko sayngkakha-ni

Run Language Theory Parsers History Options

younghee
mekestta
sayngkakha
minswu
sayngkakha
mwues
younghee

parser operations corresponding to linguistic principles

PAPPI: Fong (1990)

Filters	
<input type="checkbox"/>	Theta Criterion
<input type="checkbox"/>	D-structure Theta Condition
<input type="checkbox"/>	Subjacency
<input type="checkbox"/>	Wh-movement in Syntax
<input type="checkbox"/>	S-bar Deletion
<input type="checkbox"/>	Case Filter
<input type="checkbox"/>	Case Condition on ECs
<input type="checkbox"/>	Coindex Subject
<input type="checkbox"/>	Condition A
<input type="checkbox"/>	Condition B
<input type="checkbox"/>	Condition C
<input type="checkbox"/>	ECP
<input type="checkbox"/>	Control
<input type="checkbox"/>	License Clitics
<input type="checkbox"/>	License Object pro
<input type="checkbox"/>	ECP at LF
<input type="checkbox"/>	FI: License operator/variables
<input type="checkbox"/>	FI: Quantifier Scoping
<input type="checkbox"/>	FI: Reanalyze Bound Proforms
<input type="checkbox"/>	License Clausal Arguments
<input type="checkbox"/>	License Syntactic Adjuncts
<input type="checkbox"/>	Wh Comp Requirement
Generators	
<input type="checkbox"/>	Parse PF
<input type="checkbox"/>	Parse S-Structure
<input type="checkbox"/>	Assign Theta-Roles
<input type="checkbox"/>	Inherent Case Assignment
<input type="checkbox"/>	Assign Structural Case
<input type="checkbox"/>	Trace Theory
<input type="checkbox"/>	Functional Determination
<input type="checkbox"/>	Free Indexation
<input type="checkbox"/>	Expletive Linking
<input type="checkbox"/>	LF Movement

Theory is (relatively) unconstrained

- declarative principles can be “coded up” and combined in many different ways and yet remain “faithful” to the theory

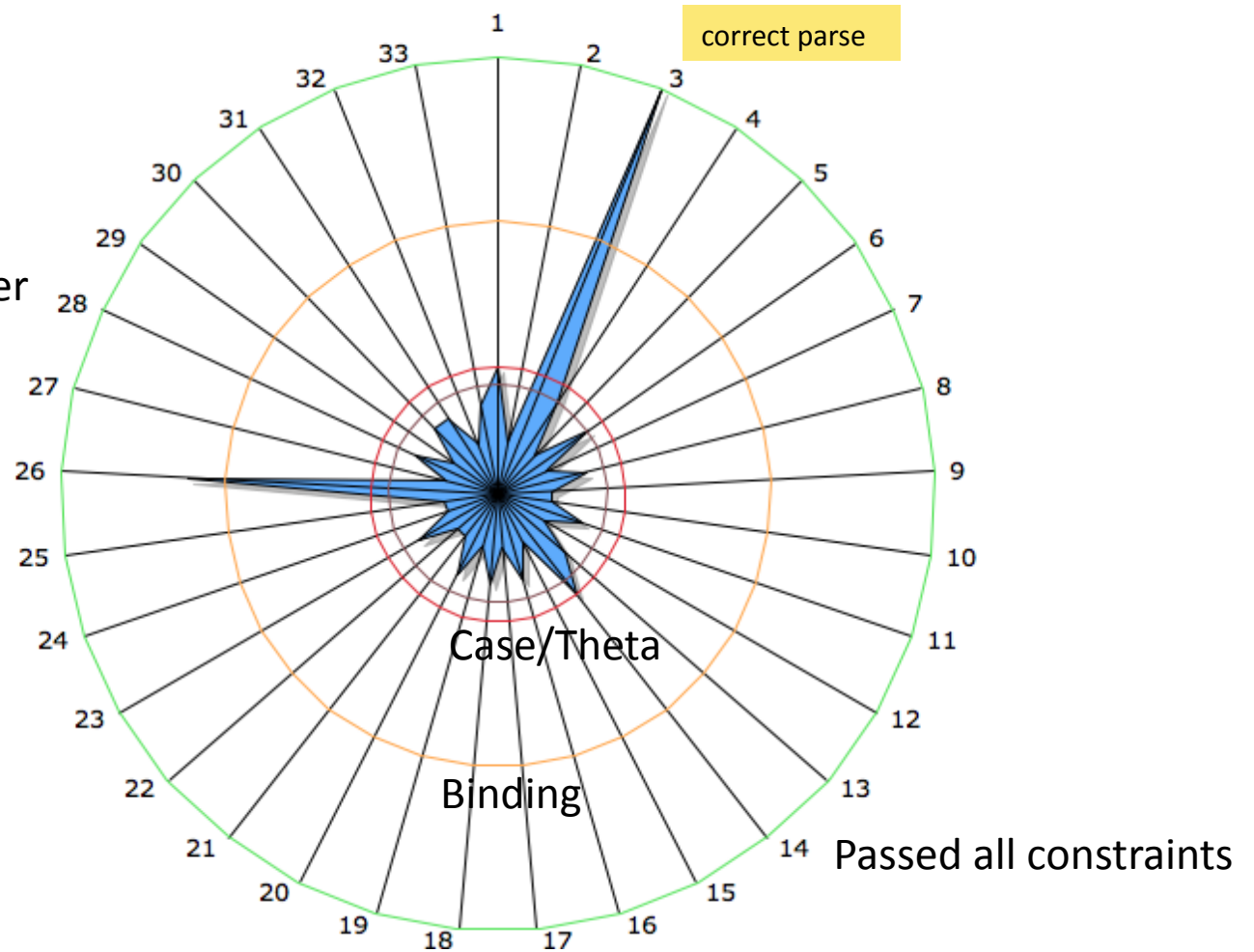
Principles and Parameters Approach

Parsing: John is too stubborn to talk to Bill

Generate and
test:

33 candidate parses
examined by the parser

*most are eliminated
early by Case/Theta
theory sub-system*



Minimalist Program

The screenshot shows a window titled 'treeviewer.tcl' with a 'TREE VIEWER' header and a date '(January 2010)'. A zoom control is set to '(x1)'. On the left, a list of operations is shown, with '2. merge v & V' highlighted. The main area displays a parse tree for the sentence 'John likes Mary'. The root node 'c' branches into 'c' (containing 'john') and 't'. Node 't' branches into 't' (containing 'john') and 'v'. Node 'v' branches into 'v*' and 'V'. Node 'V' branches into 'V' (containing 'like') and 'n' (containing 'mary'). A blue box highlights the text 'parser operations Correspond directly to linguistic operations'. Below the main tree, a smaller tree shows the 'v*' node branching into 'V' (containing 'like') and 'n' (containing 'mary'). Below this smaller tree, the text 'Probe [v*] agrees with goal [n mary]' is displayed.

1. theta merge V & N
2. merge v & V
3. theta merge N & v
4. merge T & v
5. move to spec-T
6. merge C & T

parser operations
Correspond directly
to linguistic operations

1. theta merge V & N
2. merge v & V
3. theta merge N & v
4. merge T & v
5. move to spec-T
6. merge C & T

Probe [v*] agrees with goal [n mary]

Theory is highly constrained

- at each Merge step, operations are precisely specified, not much “wriggle room” for implementation

Prior work on grammar formalisms for MP theories e.g.
(Stabler, 1998)
(Lecomte & Retoré, 2001)

A Simple Implementation

- Take a theory in the Minimalist Program
 - e.g. Derivation by Phase (Chomsky 2001)
- What is the simplest possible computational implementation that we can get away with?

Simple does not necessarily equal efficient or “minimal”

Example

- Task: sorting a list of numbers (n : size of list)
 - 5
 - 3
 - 8
 - 1
 - 7
 - 9
 - 4 ...
- Simplest implementation
 - only operation is to front lowest number
 - $O(n^2)$ comparisons
- Quicksort
 - recursively sort around a ***pivot*** number
 - same worst case but typically fast with $O(n \lg n)$ comparisons

Scaling is important in sorting but for linguistic computation?

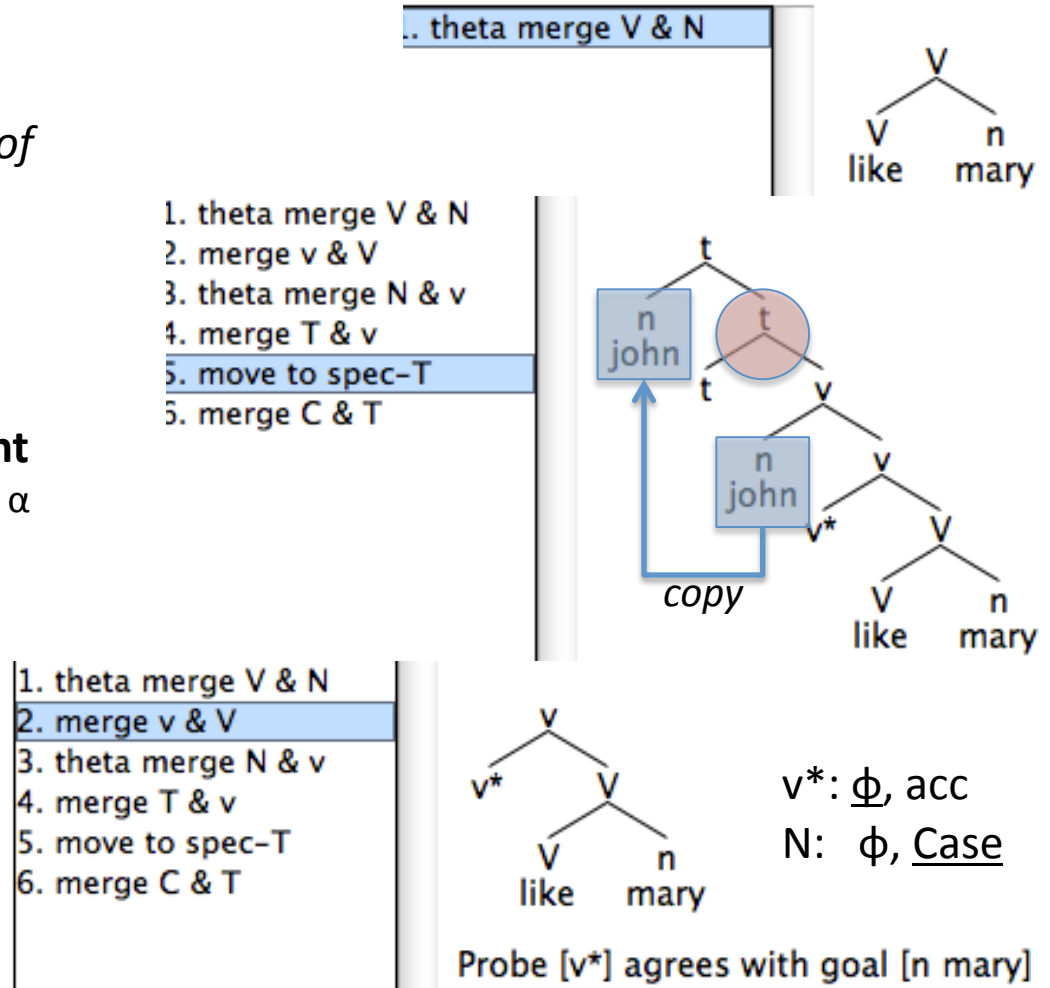
Example

- Phrase Structure Grammar
 - $S \Rightarrow NP VP$
 - $VP \Rightarrow V NP$
 - $VP \Rightarrow V$
 - $VP \Rightarrow VP PP$
 - $NP \Rightarrow D N$
 - $NP \Rightarrow N$
 - $PP \Rightarrow P NP$
- Use the grammar directly
 - recursive descent
 - bottom-up
- Transform the grammar into a finite state machine with a stack
 - Earley's algorithm (*as you go*)
 - LR(k) Parsing (*offline is for free*)

Search space: Minimize the length of the derivation

Basic Computation

- start with lexical array of syntactic objects: $\{\alpha, \dots, \omega\}$
- **Merge** “an indispensable operation of a recursive system”
- (external)
 - two syntactic objects (SOs): α, β
 - create merged SO: $\{\alpha, \beta\}$
 - $\text{label}(\{\alpha, \beta\}) = \text{label}(\alpha)$ or $\text{label}(\beta)$
- (internal), implements **Displacement**
 - SOs: α and γ, γ properly contained in α
 - create SO: $\{\alpha, \gamma\}$
 - $\text{label}(\{\alpha, \gamma\}) = \text{label}(\alpha)$
- **Agree**
 - active probe SO: α (active = uninterpretable features), goal SO: β
 - match and delete uninterpretable features of probe and goal
- **Convergent derivation:** uninterpretable features must be eliminated



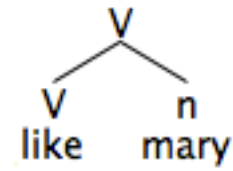
Basic Implementation (1)

Definite clause grammar (DCG) (*simplified*)

$V([\nu V N]) \rightarrow V(V), n(N).$

$V([\nu Verb]) \rightarrow [Verb].$

- phonetic matrix: $f(\text{pmatrix}, \textit{like})$
- (takes an) argument: $f(\text{arg}, +)$
- uninterpretable Case:



- features (N)
- $f(\text{pmatrix}, \textit{john})$
- $f(\text{arg}, +)$
- $f(\text{case}, _)$

variable

% (big V) verb classes

$bV(n('V', [], [V, N])) \rightarrow bV\emptyset(V), n\emptyset(N), \{\text{theta}(V), \text{theta}(N)\}$ report '*theta merge V & N*'.

$bV\emptyset(n('V', [f(\text{pmatrix}, \textit{Verb}), f(\text{arg}, +)], [])) \rightarrow [\textit{Verb}], \{\text{transitive}(\textit{Verb})\}.$

$bV\emptyset(n('V', [f(\text{pmatrix}, \textit{Verb}), f(\text{arg}, +)], [])) \rightarrow [\textit{Verb}], \{\text{unaccusative}(\textit{Verb})\}.$

$bV\emptyset(n('V', [f(\text{pmatrix}, \textit{Verb})], [])) \rightarrow [\textit{Verb}], \{\text{unergative}(\textit{Verb})\}.$

$\text{transitive}(\textit{like}).$ $\text{transitive}(\textit{expect}).$

$\text{unergative}(\textit{run}).$ $\text{unaccusative}(\textit{arrive}).$

$n\emptyset(n(n, [f(\text{pmatrix}, \textit{BNoun}) | Fs], [])) \rightarrow [\textit{BNoun}], \{\text{bareNoun}(\textit{BNoun}, Fs)\}.$

$\text{bareNoun}(\textit{john}, [f(\text{phi}, 3\text{-sg-m}), f(\text{case}, _), f(\text{arg}, +)]).$

Basic Implementation (2)

Definite clause grammar (DCG) (*simplified*)

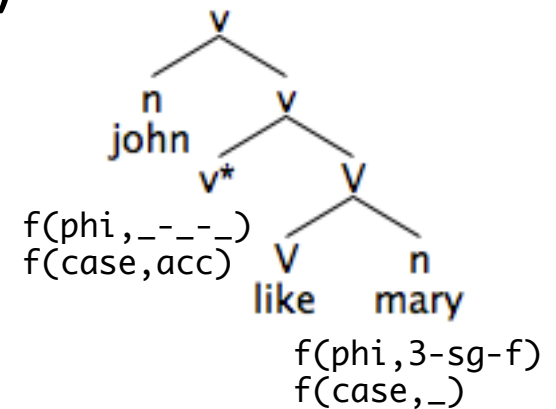
$v([_v N v]) \rightarrow n(N), v(v).$

$v([_v v V]) \rightarrow v(v), V(V).$

$v([_v^*]) \rightarrow [].$

– features (v^*)

- uninterpretable ϕ -features: $f(\phi, _ _ _ _)$
- can value accusative Case: $f(\text{case}, \text{acc})$

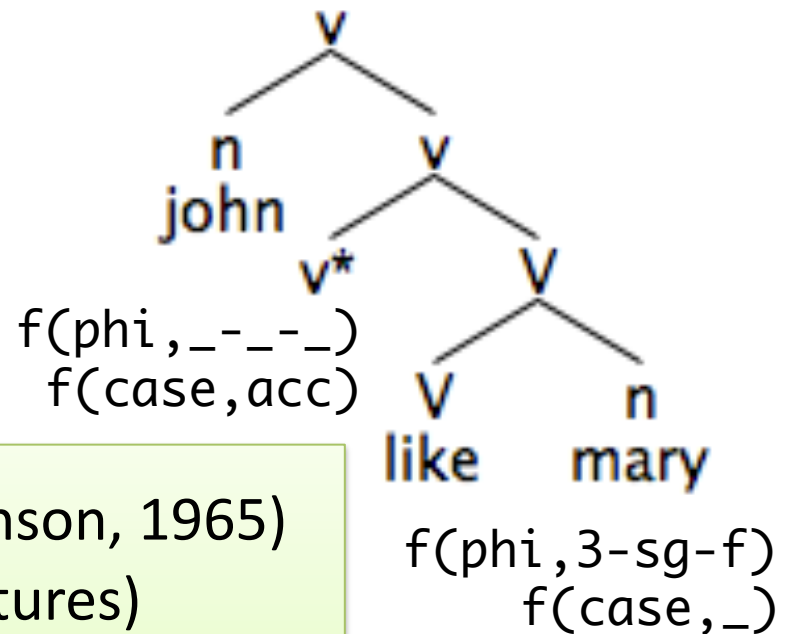


triggers Agree

```
% v*/vP
v(n(v, [], [N, V])) --> n0(N), {theta(N)}, v1(V) report 'theta merge N & v'.
v1(n(v, [], [V, BV])) --> v0(V), bV(BV), {goals(BV, Goals), agree(V, Goals)}
report 'merge v & V'.
v0(n('v*', [f(phi, _ _ _ _), f(case, acc)], [])) --> [].
```

Basic Implementation (2)

- Agree(v^* ,N)



Operation: unification (Robinson, 1965)
 (match and instantiate unvalued features)

<u>Probe</u>	<u>Goal</u>
$f(\text{phi}, _ - _ - _)$	$f(\text{phi}, 3\text{-sg-f})$
$f(\text{case}, \text{acc})$	$f(\text{case}, _)$

Uninterpretable/unvalued features (*represented by variables*) are eliminated

Basic Implementation (3)

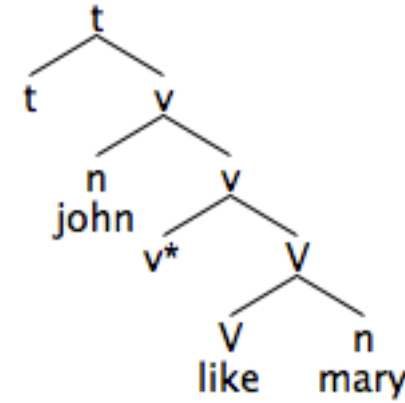
Definite clause grammar (DCG)

$T([_T N v]) \rightarrow n(N), v(v).$

$T([_T N v]) \rightarrow v(v), \{N \text{ a goal}\}$

$T([_T]) \rightarrow [].$

– features (ϕ -complete T)



defective T
 ϕ -incomplete
 e.g. infinitivals

- uninterpretable ϕ -features: $f(\phi, _ _ _ _)$
- ✗ can value nominative Case: $f(\text{case}, \text{nom})$
- EPP

triggers Agree

$t(n(t, [], [N, T])) \rightarrow n\theta(N), \{\text{nonarg}(N)\}, t1(T, _)$ report 'merge expl & T'.

% EPP: (1) merge

$t(n(t, [], [Goal, T])) \rightarrow t1(T, Goal)$ report 'move to spec-T'.

% EPP: (2) move with maximize matching

$t1(n(t, [], [T, V]), G) \rightarrow t\theta(T), v(V), \{\text{goals}(V, Goals), \text{agree}(T, Goals)\},$
 $Goals = [G | _]$ report 'merge T & v'.

$t\theta(n(t, [f(\phi, _ _ _ _), f(\text{case}, \text{nom})], [])) \rightarrow [].$

Putting it all together (1)

The screenshot shows a window titled 'treeviewer.tcl' with a 'TREE VIEWER' header. On the left, a list of operations is displayed:

1. theta merge V & N
2. merge v & V
3. theta merge N & v
4. merge T & v
5. move to spec-T
6. merge C & T

The main area of the window displays a partial parse tree for the sentence 'like mary'. The root node is 'V', which branches into two children: 'V' and 'n'. The word 'like' is positioned below the left 'V' node, and the word 'mary' is positioned below the 'n' node.

```
graph TD; V1[V] --- V2[V]; V1 --- n[n]; V2 --- like[like]; n --- mary[mary]
```

Additional interface elements include a zoom control set to '(x1)' and a date '(January 2010)' in the top right corner.

Putting it all together (2)

treeviewer.tcl
TREE VIEWER
January 2010
Zoom: (x1) (x4)

1. theta merge V & N
2. merge v & V
3. theta merge N & v
4. merge T & v
5. move to spec-T
6. merge C & T

```
graph TD; V1[V] --- v_star[v*]; V1 --- V2[V]; V2 --- V3[V]; V2 --- n[n]; V3 --- like[like]; V3 --- mary[mary]
```

Probe [v*] agrees with goal [n mary]

Putting it all together (3)

treeviewer.tcl
TREE VIEWER
(January 2010)
Zoom: (x1) (x4)

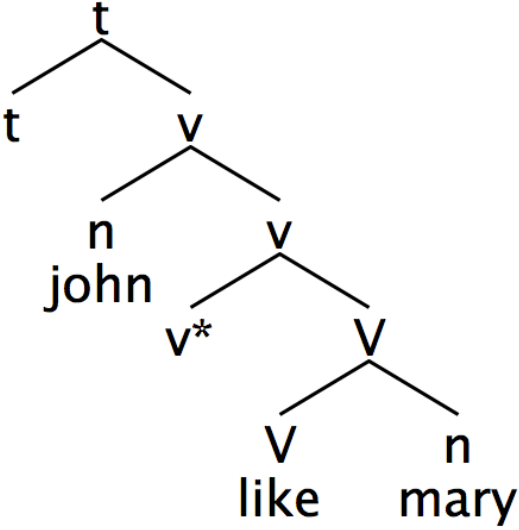
1. theta merge V & N
2. merge v & V
3. theta merge N & v
4. merge T & v
5. move to spec-T
6. merge C & T

```
graph TD; v1[v] --- n1[n]; v1 --- v2[v]; n1 --- john[john]; v2 --- v3[v*]; v2 --- V[V]; V --- v4[v]; V --- n2[n]; v4 --- like[like]; n2 --- mary[mary];
```


Putting it all together (4)

treeviewer.tcl
TREE VIEWER
(January 2010)
Zoom: (x1) (x4)

1. theta merge V & N
2. merge v & V
3. theta merge N & v
4. merge T & v
5. move to spec-T
6. merge C & T



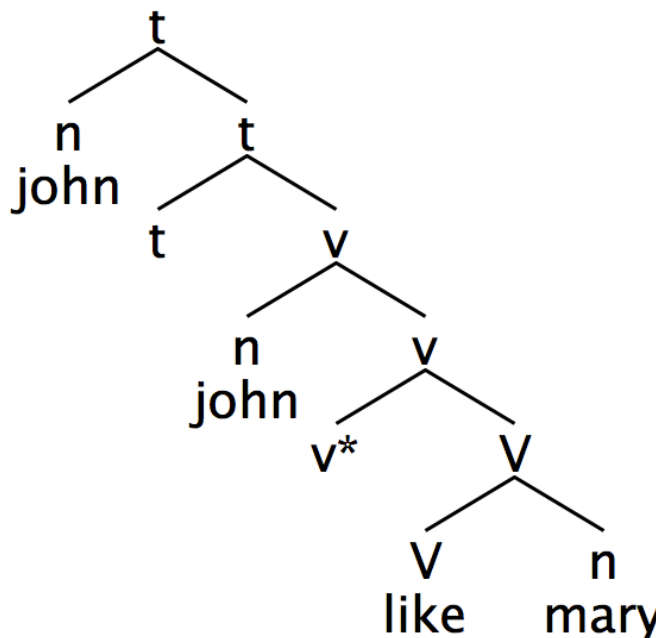
```
graph TD; t1[t] --- t2[t]; t1 --- v1[v]; v1 --- n1[n]; n1 --- john[john]; v1 --- v2[v]; v2 --- v3[v*]; v2 --- V1[V]; V1 --- V2[V]; V2 --- like[like]; V1 --- n2[n]; n2 --- mary[mary]
```

Probe [t] agrees with goal [n john]

Putting it all together (5)

treeviewer.tcl
TREE VIEWER
January 2010
Zoom: (x1) (x4)

1. theta merge V & N
2. merge v & V
3. theta merge N & v
4. merge T & v
5. move to spec-T
6. merge C & T

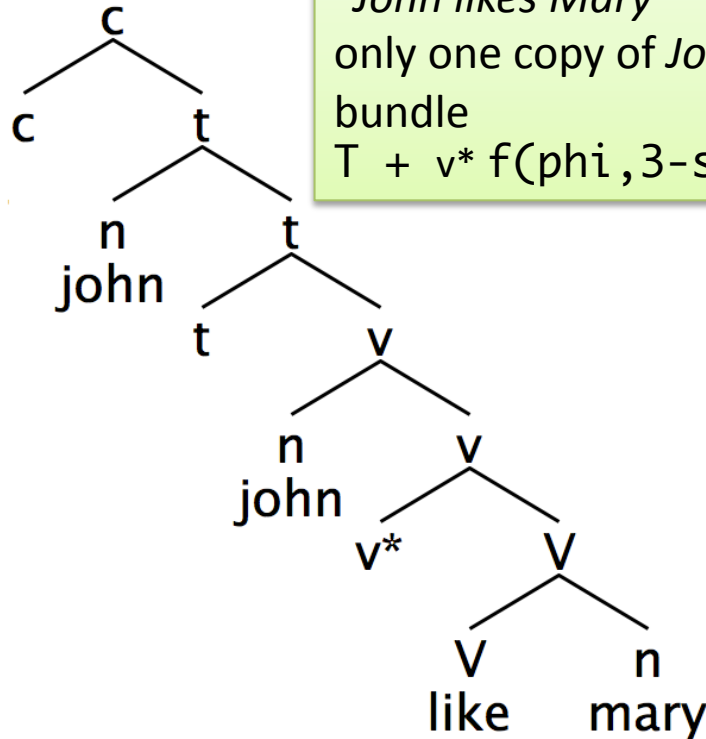


```
graph TD; t1[t] --- n1[n]; t1 --- t2[t]; n1 --- john1[john]; t2 --- t3[t]; t2 --- v1[v]; t3 --- john2[john]; t3 --- v2[v]; v1 --- v3[v*]; v1 --- v4[V]; v3 --- v5[V]; v3 --- n2[n]; v5 --- like[like]; n2 --- mary[mary]
```

(EPP requirement)

Putting it all together (6)

1. theta merge V & N
2. merge v & V
3. theta merge N & v
4. merge T & v
5. move to spec-T
6. merge C & T



Operation: Spell-Out

(not currently implemented)

"John likes Mary"

only one copy of *John* is pronounced

bundle

$T + v^* f(\phi, 3\text{-sg-f}) + v(\text{like}) = \text{likes}$

Examples

assume, are raising constructions and their exceptional-Case-marking (ECM) counterparts, as shown schematically in (4a), where β is the matrix clause, α is an infinitival with YP a verbal phrase (the case most relevant here), and P is the probe: T with a raising verb (case (4b)), v with an ECM transitive verb (case (4c)).¹⁰

- (4) a. [β P [α [Subj [H YP]]]]
- b. i. there are likely to be awarded several prizes
 - ii. several prizes are likely to be awarded
 - c. i. we expect there to be awarded several prizes
 - ii. we expect several prizes to be awarded

The Case/agreement properties of Subj in (4a), and its overt location, are determined by properties of the matrix probe P, not internally to α . α is a TP with defective head T_{def} , which is unable to determine Case/agreement but has an EPP-feature, overtly manifested in (4c). Raising-ECM parallels give good reason to believe that the EPP-feature is manifested in (4b) as well, by trace of the matrix subject; preference for Merge over (more complex) Move gives a plausible reason for the surface distinction between [Spec, T_{def}] in (4b) and in (4c) (see MI). In (4bi) and (4ci), the EPP-feature of T_{def} is satisfied by Merge of expletive; in (4bii) and (4cii), by raising of the direct object.

(Chomsky 2001)

A Worked Example

Consider the derivation of

- several prizes are likely to be awarded (= 4(b)(ii))

awarded = *award* + *-ed* (adjectival participle)

-ed ϕ -incomplete: uninterpretable Number and Gender only
uninterpretable Case
morphologically unrealized in English (cf. Icelandic)

?- parse([be,likely,be,ed,award,several,prizes]).

Probe [a!case ed] agrees with goal [n!case several prizes]

Probe [tdef] agrees with goal [n!case several prizes]

Probe [t] agrees with goal [n several prizes]

[c[c][t[n several prizes][t[t][v[v be][a[a likely][t[n several prizes][t[tdef][v[v be][a[a ed][V[V award][n several prizes]]]]]]]]]]]]

Agree(a,N)

-ed: ϕ , Case

N: ϕ , Case

A Worked Example

The screenshot shows a window titled 'treeviewer.tcl' with a 'TREE VIEWER' header. On the left, a list of 10 steps is shown, with the first step highlighted. The main area displays a parse tree for the sentence 'award several prizes'. The root node is 'V', which branches into 'V' and 'n'. The word 'award' is under the left 'V', 'several' is under the 'n', and 'prizes' is under the right 'n'. A zoom control is visible in the top right corner, set to (x4).

treeviewer.tcl

TREE VIEWER

(January 2010)

Zoom: (x1) (x4)

1. theta merge V & N
2. merge PRT & V
3. merge v & A
4. merge T & v
5. move to spec-T
6. merge A & Tdef
7. merge v & A
8. merge T & v
9. move to spec-T
10. merge C & T

V

V n

award several prizes

A Worked Example

treeviewer.tcl (January 2010)

1. theta merge V & N
2. merge PRT & V
3. merge v & A
4. merge T & v
5. move to spec-T
6. merge A & Tdef
7. merge v & A
8. merge T & v
9. move to spec-T
10. merge C & T

Tree structure:
a
├── a
│ └── ed
└── V
 ├── V
 │ └── award
 └── n
 └── several prizes

Probe [a!case ed] agrees with goal [n!case several prizes]

Notation: !case means feature is unvalued

Agree(a,N)
-ed: ϕ , Case
N: ϕ , Case

A Worked Example

treeviewer.tcl

TREE VIEWER (January 2010)

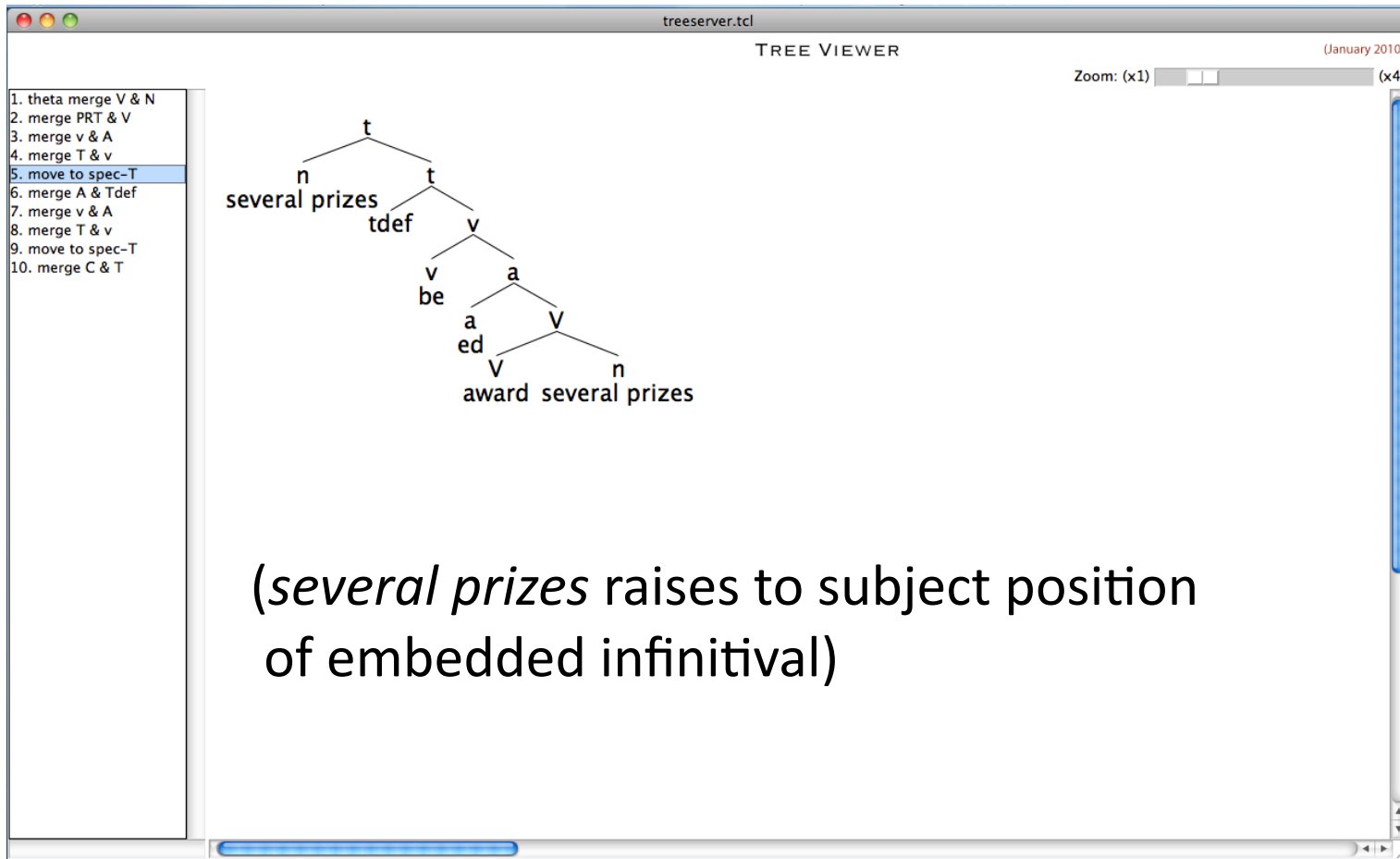
Zoom: (x1) (x4)

1. theta merge V & N
2. merge PRT & V
3. merge v & A
4. merge T & v
5. move to spec-T
6. merge A & Tdef
7. merge v & A
8. merge T & v
9. move to spec-T
10. merge C & T

Probe [tdef] agrees with goal [n!case several prizes]

Agree(Tdef,N)
Tdef: ϕ
N: ϕ , Case

A Worked Example



treeviewer.tcl
TREE VIEWER
(January 2010)
Zoom: (x1) (x4)

1. theta merge V & N
2. merge PRT & V
3. merge v & A
4. merge T & v
5. move to spec-T
6. merge A & Tdef
7. merge v & A
8. merge T & v
9. move to spec-T
10. merge C & T

```
graph TD
    t1[t] --- n1[n]
    t1 --- t2[t]
    n1 --- sp[several prizes]
    t2 --- tdef[tdef]
    t2 --- v1[v]
    v1 --- v2[v]
    v1 --- a1[a]
    v2 --- be[be]
    a1 --- a2[a]
    a1 --- V[V]
    a2 --- ed[ed]
    V --- V2[V]
    V --- n2[n]
    V2 --- award[award]
    n2 --- sp2[several prizes]
```

(*several prizes* raises to subject position of embedded infinitival)

A Worked Example

Matrix T

1. theta merge V & N
 2. merge PRT & V
 3. merge v & A
 4. merge T & v
 5. move to spec-T
 6. merge A & Tdef
 7. merge v & A
 8. merge T & v
 9. move to spec-T
 10. merge C & T

Agree(T,N)
 T: ϕ , Nominative
 N: ϕ , Case

Case for *-ed* also valued
 because of earlier unification step
 Agree(a,N)
-ed: ϕ , Case
 N: ϕ , Case

Probe [t] agrees with goal [n several prizes]

Unification presents an possible advantage: computation is more local

Probing with multiple goals

What about P_{rt}? Its ϕ -features are deleted at stage α and should therefore be invisible to Match by the probe. Case of P_{rt} cannot be valued and the derivation crashes, contrary to fact.

The problem is overcome if Spell-Out takes place at the strong-phase level. Then the ϕ -features of P_{rt} are still visible at stage β of the cycle, though deleted; they disappear at the strong-phase level CP or ν P, as the phase is transmitted to the phonological component. At stage α of the cycle, the ϕ -features of P_{rt} are valued by P_{rt}-DO matching, as just discussed. At the next stage, the probe T/ ν matches the (still visible) goal P_{rt}, valuing its Case feature; and the probe matches the goal DO, valuing the Case feature of DO as well as its own features (since DO is ϕ -complete).

compare with...

(Chomsky 2001)

A Worked Example

treeviewer.tcl
TREE VIEWER
(January 2010)
Zoom: (x1) (x4)

1. theta merge V & N
2. merge PRT & V
3. merge v & A
4. merge T & v
5. move to spec-T
6. merge A & Tdef
7. merge v & A
8. merge T & v
9. move to spec-T
10. merge C & T

Spell-Out
Several prizes are likely to be awarded

```
graph TD
    c1[c] --- c2[c]
    c1 --- t1[t]
    c2 --- np1[several prizes]
    t1 --- t2[t]
    t1 --- v1[v]
    t2 --- a1[a]
    t2 --- t3[t]
    v1 --- v2[v]
    v1 --- a2[a]
    v2 --- be[be]
    a2 --- likely[likely]
    t3 --- np2[several prizes]
    t3 --- t4[tdef]
    t4 --- v3[v]
    t4 --- a3[a]
    v3 --- v4[v]
    v3 --- a4[a]
    v4 --- ed[ed]
    a4 --- V[V]
    V --- award[award]
    a3 --- n[n]
    n --- np3[several prizes]
```

Another Example

Consider also the derivation of

- There are likely to be awarded several prizes (= 4(b)(i))

pleonastic *there*: ϕ -incomplete
(Person only)

Agree(a,N)
-ed: ϕ , Case
N: ϕ , Case

?- parse([be,likely,**there**,be,ed,award,several,prizes]).
 Probe [a!case ed] agrees with goal [n!case several prizes]
 Probe [tdef] agrees with goal [n!case several prizes]
 Probe [t!phi] agrees with goal [n!phi there]
 Probe [t] agrees with goal [n several prizes]
 [c[c][t[n there][t[t][v[v be][a[a likely][t[n there][t[tdef][v[v
 be][a[a ed][V[V award][n several prizes]]]]]]]]]]

Agree(T,N)
T: ϕ
N: ϕ

Another Example

The screenshot shows a window titled 'treeviewer.tcl' with a 'TREE VIEWER' header and a '(January 2010)' date. A zoom slider is set to '(x1)'. On the left, a list of 10 merge operations is shown, with '2. merge PRT & V' selected. The main area displays a syntax tree for the sentence 'award several prizes'. The root node 'a' branches into 'a' and 'V'. The 'a' node branches into 'ed' and 'V'. The 'V' node branches into 'award' and 'n'. The 'n' node branches into 'several' and 'prizes'. Below the tree, the text reads: 'Probe [a!case ed] agrees with goal [n!case several prizes]'. A green box at the bottom contains the following text:

Agree(a,N)
-ed: ϕ , Case
N: ϕ , Case

Another Example

treeviewer.tcl

TREE VIEWER

Zoom: (x1)

1. theta merge V & N
2. merge PRT & V
3. merge v & A
4. merge T & v
5. merge expl & T
6. merge A & Tdef
7. merge v & A
8. merge T & v
9. move to spec-T
10. merge C & T

Probe [t!phi] agrees with goal [n!phi there]
 Probe [t] agrees with goal [n several prizes]

1st Agree(T,N)
 T: ϕ , Nominative
 N: ϕ

2nd Agree(T,N)
 T: ϕ , Nominative
 N: ϕ , Case

there : ϕ -feature indirectly valued by 2nd Agree(T,N)
-ed: Case also indirectly valued by 2nd Agree(T,N)
 However, T must probe past *there* all the way on down

Another Example

treeviewer.tcl

TREE VIEWER (January 2010)

Zoom: (x1) (x4)

1. theta merge V & N
2. merge PRT & V
3. merge v & A
4. merge T & v
5. merge expl & T
6. merge A & Tdef
7. merge v & A
8. merge T & v
9. move to spec-T
10. merge C & T

Spell-Out
There are likely to be awarded several prizes

(there raises to matrix subject position)

Examples

assume, are raising constructions and their exceptional-Case-marking (ECM) counterparts, as shown schematically in (4a), where β is the matrix clause, α is an infinitival with YP a verbal phrase (the case most relevant here), and P is the probe: T with a raising verb (case (4b)), v with an ECM transitive verb (case (4c)).¹⁰

- (4) a. [β P [α [Subj [H YP]]]]
b. i. there are likely to be awarded several prizes
ii. several prizes are likely to be awarded
c. i. we expect there to be awarded several prizes
ii. we expect several prizes to be awarded

- Examples (ECM):

- (i) we expect there to be awarded several prizes
- (ii) we expect several prizes to be awarded

[Spec, T_{def}] in (4b) and in (4c) (see MI). In (4bi) and (4ci), the EPP-feature of T_{def} is satisfied by Merge of expletive; in (4bii) and (4cii), by raising of the direct object.

(Chomsky 2001)

Example: *we expect several prizes to be awarded*

treeviewer.tcl

TREE VIEWER

Zoom: (x1) (x4)

1. theta merge V & N
2. merge PRT & V
3. merge v & A
4. merge T & v
5. move to spec-T
6. merge V (ecm) & Tdef
7. merge v & V
8. theta merge N & v
9. merge T & v
10. move to spec-T
11. merge C & T

**Prior Unification:
locality advantage**

Case
award several prizes

Probe [v*] agrees with goal [n several prizes]

Example: *we expect several prizes to be awarded*

treeviewer.tcl

TREE VIEWER (January 2010)

Zoom: (x1) [] (x4)

1. theta merge V & N
2. merge PRT & V
3. merge v & A
4. merge T & v
5. move to spec-T
6. merge V (ecm) & Tdef
7. merge v & V
8. theta merge N & v
9. merge T & v
10. move to spec-T
11. merge C & T

Spell-Out
We expect several prizes to be awarded

Conclusions

- described a (non-trivial) implementation of a probe-goal system for Case and verbal inflection
- we want to (1) minimize # operations, and (2) localize goal search as far as possible
- unification for uninterpretable feature valuation can help both (1) and (2)