Syntactic generation

Problems with syntactic generation

Generation algorithms – techniques and efficiency

THE ROLE OF NATURAL LANGUAGE GENERATION
(here: syntactic generation, compared to parsing)

History

Considered trivial for a long time (in comparison to parsing)

Became an issue in connection with unification-based grammars

First simple attempts to reuse parsing tools turned out very badly

Problems

Underspecification is a typical problem (for building input representations)

Expressibility (also for building input representations)

Efficiency

Exploiting reuse potential (bi-directional grammars)

A FIRST APPROACH “SHAKE 'N BAKE” (Whitelock 1988)

Motivation

Very flexible – all combinations considered prior to testing feasibility

Originally used within symbolic machine translation

Functionality

Lexical entries are retrieved from lexicon by semantic relations of the input

All combinations of all words and phrases are tried by a shift-reduce parser

All phrases are returned which use up all of the input semantics

Assessment

Virtually no information about semantic relations in the input specification

Can be very expensive

SEMANTIC HEAD-DRIVEN GENERATION (Shieber et al. 1990)

Motivation

Same problems with top-down generation as with top-down parsing

Feasible bottom-up generation requires semantic monotonicity - strong

Functionality

Combined top-down and bottom-up traversal oriented on semantic head node

Looks for “pivot“ – “lowest“ node which shares semantics with root

Tries to connect “pivot” to root node

Recursively expands sister nodes in the course of the connection to root

Assessment

Rather efficient

Requirements on grammars - semantic headedness

THE ALGORITHM (1)
THE TOP-LEVEL PROCEDURE

It consists of three subprocedure calls:

```
generate(Root) :-
        % choose non-chain rule
    applicable_non_chain_rule(Root,Pivot,RHS),

        % generate all subconstituents
    generate_rhs(RHS),

        % generate material on path to root
    connect(Pivot,Root).
```

THE ALGORITHM (2)
THE RECURSIVE CALL VIA RIGHT HAND SIDES

It consists of a base case and a simple recursive call:

```
generate_rhs([]).

generate_rhs([First | Rest]) :-
        generate(First),
        generate_rhs(Rest).
```

THE ALGORITHM (3)
CONNECTING THE PIVOT TO THE ROOT

It consists of a base case and the general one, with three subprocedure calls:

```
connect(Pivot,Root) :-
        % choose chain rule
    applicable_chain_rule(Pivot,LHS,Root,RHS),
        % generate remaining siblings
    generate_rhs(RHS),
        % connect the new parent to the root
    connect(LHS,Root).
connect(Pivot,Root) :-
        % trivially connect pivot to root
    unify(Pivot,Root).
```

THE ALGORITHM (4)
FIND APPLICABLE NON-CHAIN RULES

It checks the semantics and picks a suitable rule:

```
applicable_non_chain_rule(Root,Pivot,RHS) :-
        % semantics of root and pivot are the same
    node_semantics(Root,Sem),
    node_semantics(Pivot,Sem),
        % choose a nonchain rule
    non_chain_rule(LHS,RHS),
        % … whose lhs matches the pivot
    unify(Pivot,LHS),
        % make sure the categories can connect
    chained_nodes(Pivot,Root).
```

THE ALGORITHM (5)
FIND APPLICABLE CHAIN RULES

It picks a suitable rule and tests it:

```
applicable_chain_rule(Pivot,Parent,Root,RHS) :-
        % choose a chain rule
    chain_rule(Parent,RHS,SemHead),
        % … whose semantic head matches the pivot
    unify(Pivot,SemHead),
        % make sure the categories can connect
    chained_nodes(Parent,Root).
```

AN EXAMPLE (1)
FRAGMENT OF A TOY GRAMMAR

Conventions

"/" separates syntax and semantics

subcategorization for complements performed lexically

Sentence/decl(S) → s(finite)/S. **(1)**

Sentence/imp(S) → vp(nonfinite[np(_)/you])/S.

s(form)S → s(finite)/S. **(2)**

vp(Form,Subcat)/S → vp(Form,[Compl | Subcat])/S,Compl. **(3)**

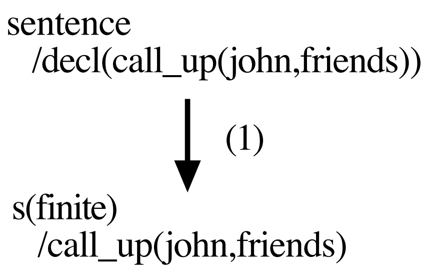
vp(finite,[np(_)/O,p/up,np(3-sing)/S])/call_up(S,O) → calls. **(4)**

np(3-sing)john → [john]. **(5)**

np(3-pl)friends → [friends]. **(6)**

p/up → [up]. **(7)**

AN EXAMPLE (2)



Generation starting with the category

sentence

and the semantics

decl(call_up(john,friends))

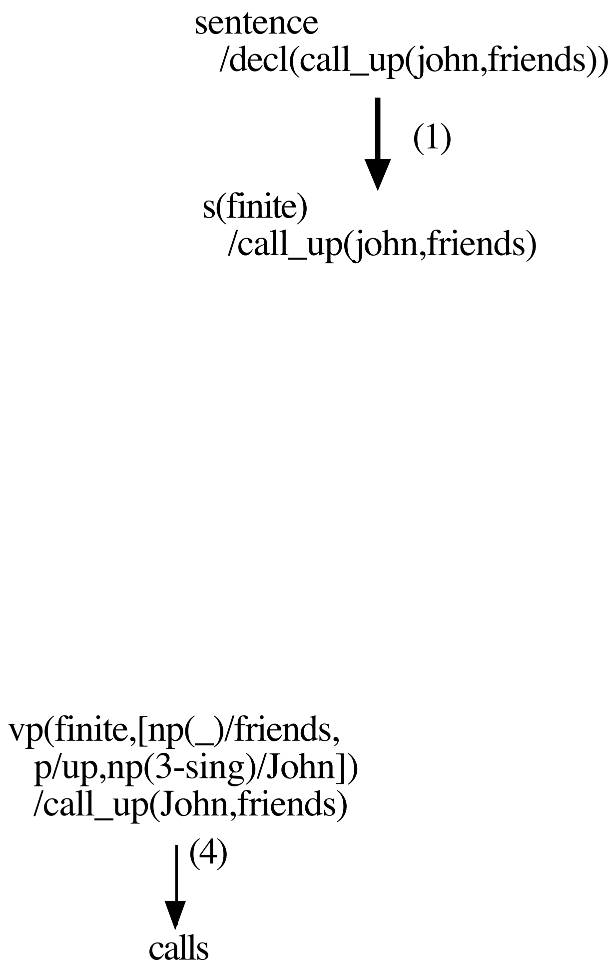
which ultimately yields

“John calls friends up”

The first step is finding a nonchain rule that will define the pivot (rule (1)) resulting in

s(finite)/call_up(john,friends)

AN EXAMPLE (3)



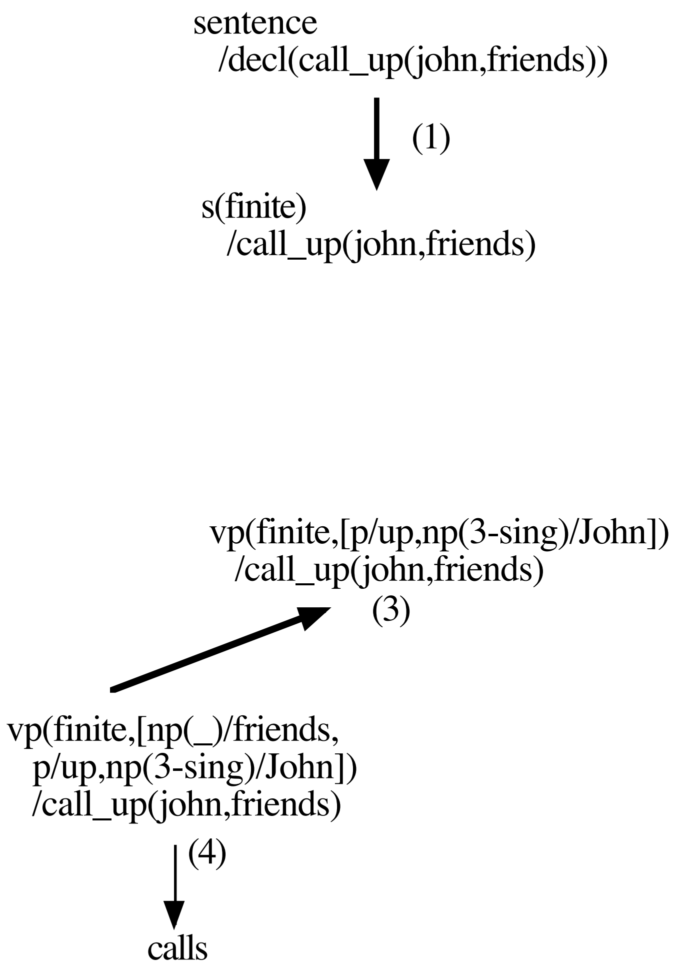
Generation continues recursively from the child node

s(finite)/call_up(john,friends)

The next step is finding a nonchain rule that will define the pivot (rule (4)) resulting in a temporarily dangling node:

vp(finite,[np(_)/O.p/up,np(3-sing)/S])
/call_up(S,O)

AN EXAMPLE (4)



Next, the pivot

vp(finite,[np(_)/friends.
p/up,np(3-sing)/John])
/call_up(John,friends)

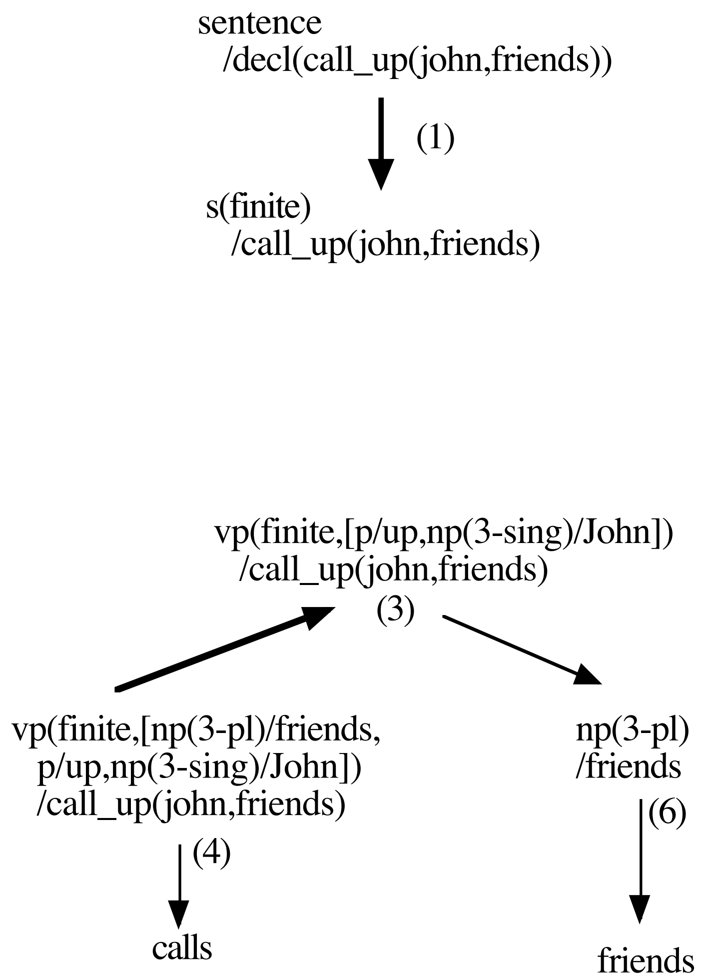
must be connected to the root

s(finite)/call_up(john,friends)

The only suitable chain rule
with matching semantic head is (3)
resulting in another node one level up:

vp(finite,[p/up,np(3-sing)/John])
/call_up(John,friends)

AN EXAMPLE (5)

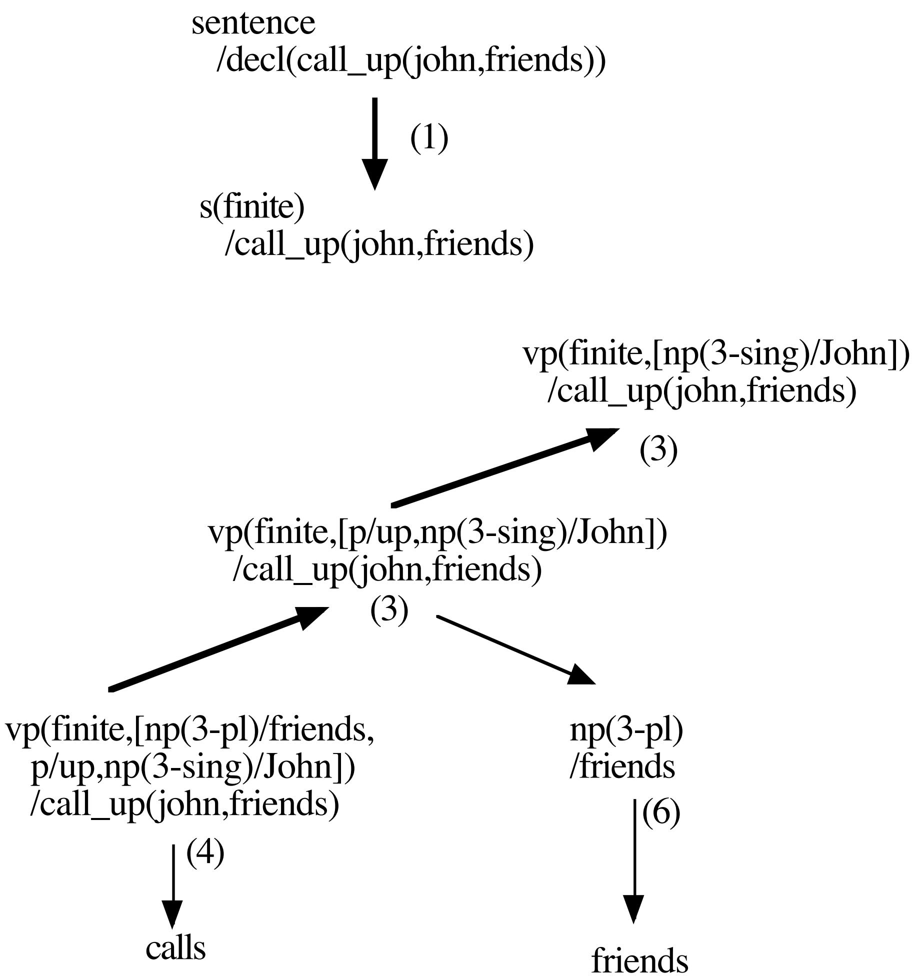


Unifying the pivot, recursive generation of the remaining RHS element

np(_)/friends

must be carried out, by rule (6)
Application of this rule yields the number of this constituent which is percolated in the tree through unification

AN EXAMPLE (6)



Again, the pivot

vp(finite,[p/up,np(3-sing)/John])

/call_up(John,friends)

must be connected to the root

s(finite)/call_up(john,friends)

The only suitable chain rule

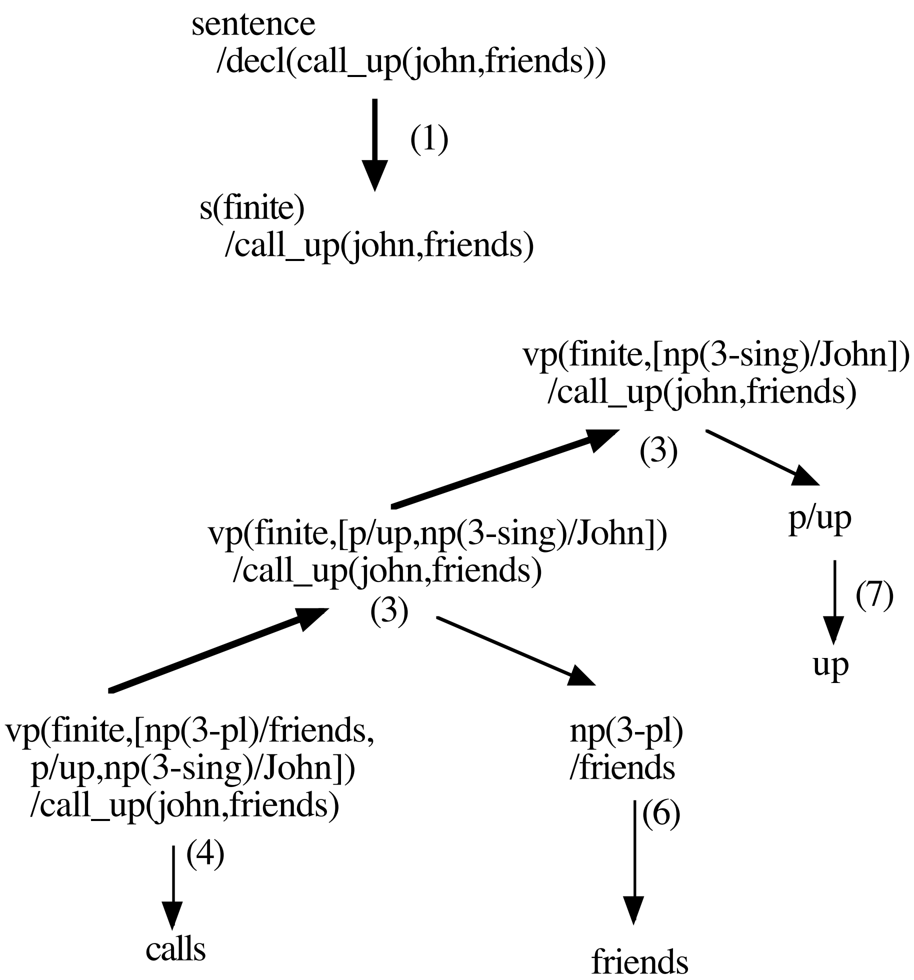
with matching semantic head still is (3)

resulting in another node one level up:

vp(finite,[np(3-sing)/John])

/call_up(John,friends)

AN EXAMPLE (7)



Again, unifying the pivot,

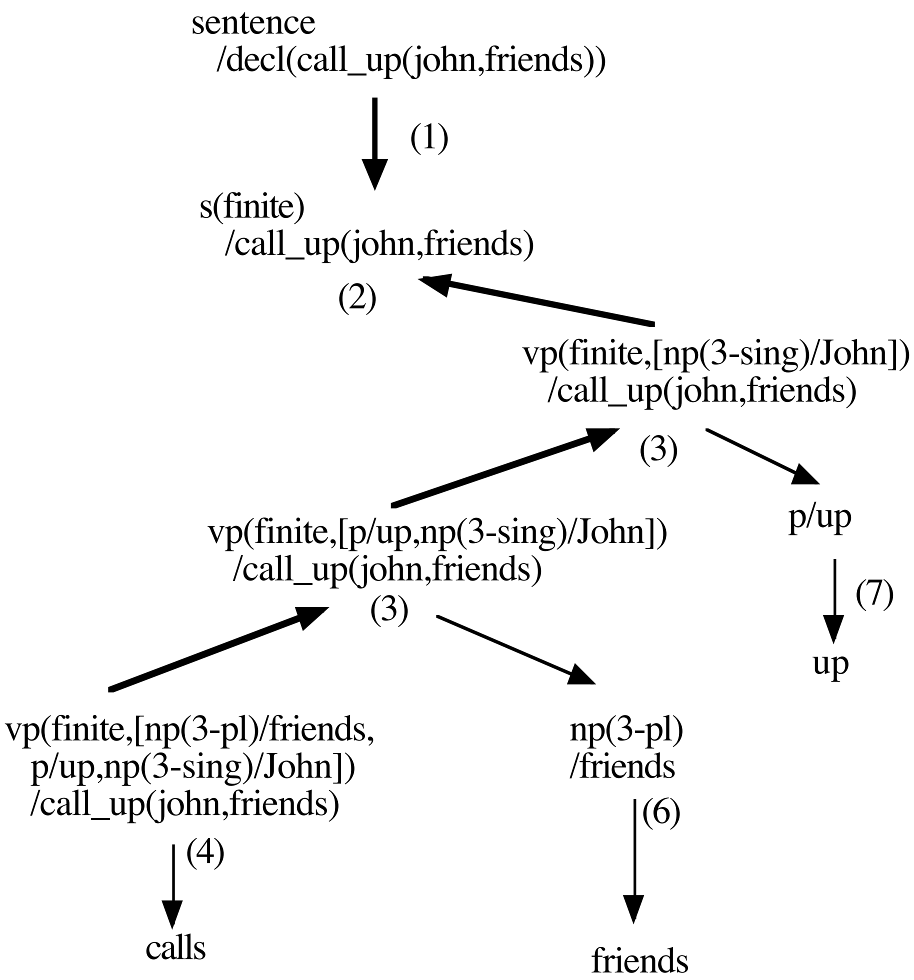
recursive generation of

the remaining RHS element

p/up

must be carried out, by rule (7)

AN EXAMPLE (8)



Ultimately, the pivot

vp(finite,[np(3-sing)/John])

/call_up(John,friends)

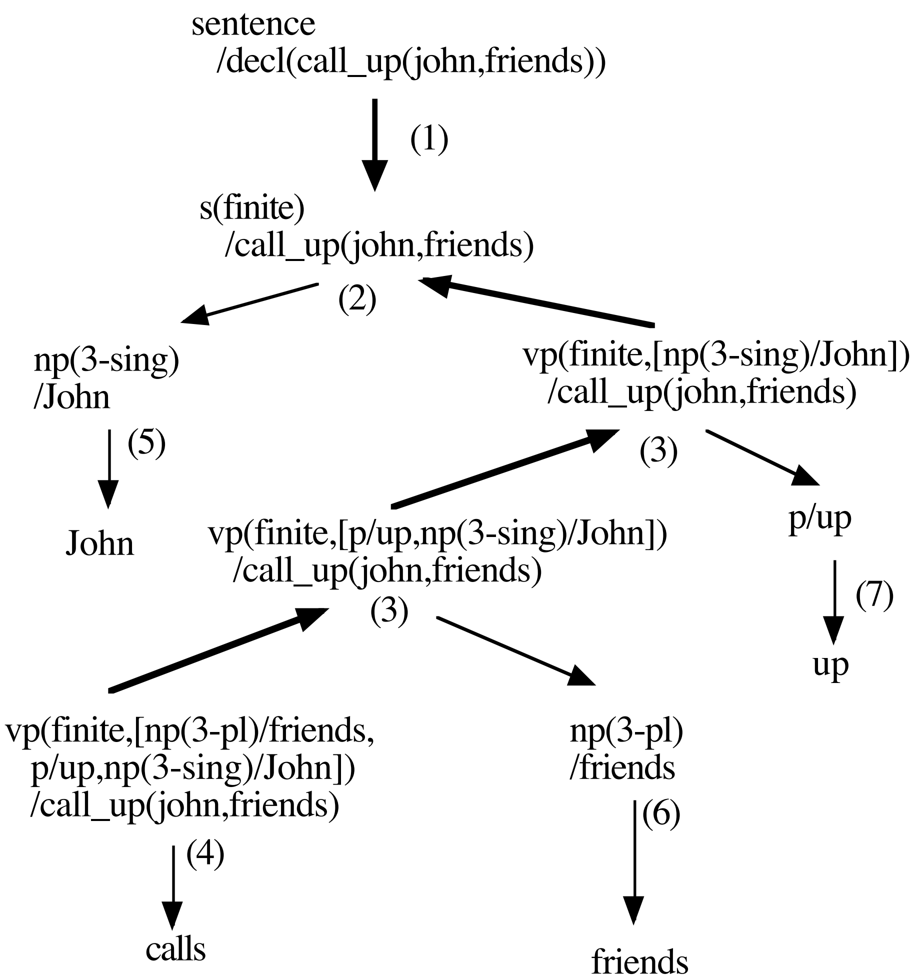
can be connected to the root

s(finite)/call_up(john,friends)

The only suitable chain rule

with matching semantic head here is (2)

AN EXAMPLE (9)



Finally, recursive generation of the LHS element

np(_)/John

must be carried out, by rule (5)

Moreover, the pivot

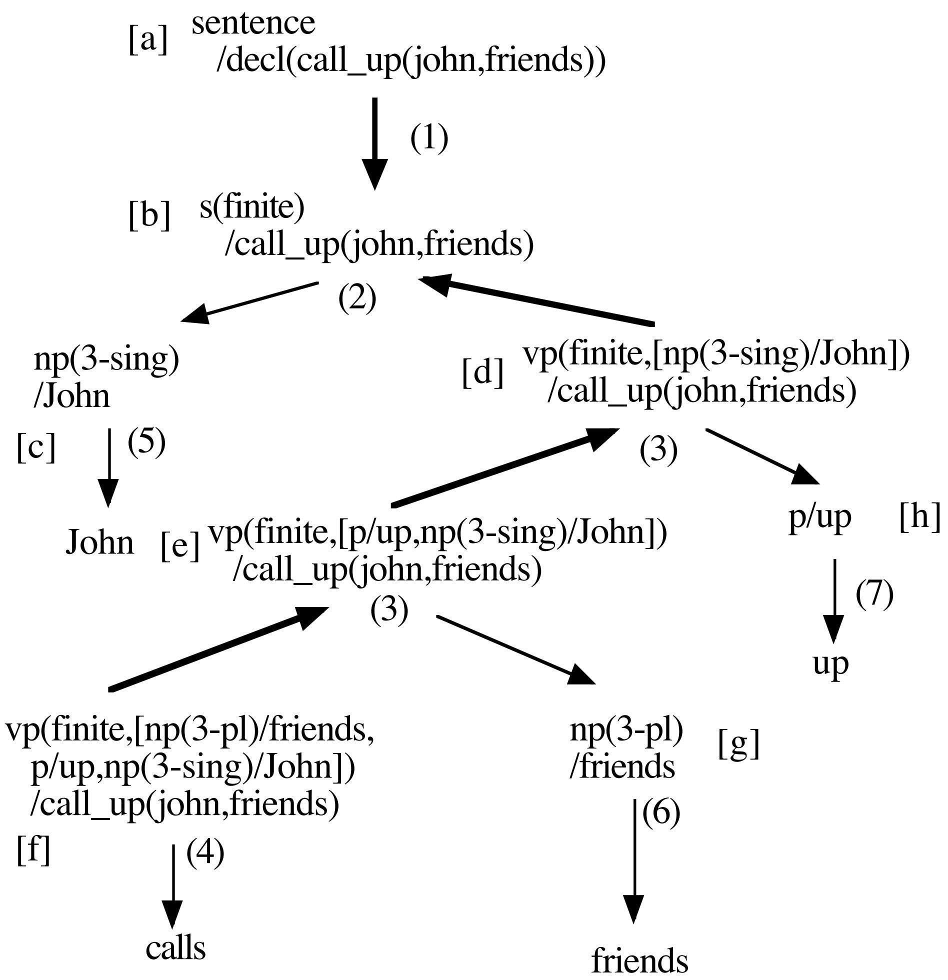
s(finite)/call_up(john,friends)

can be connected to the root

s(finite)/call_up(john,friends)

via identity

THE EXAMPLE – SUMMARY



Semantics

decl(call_up(john,friends))

Sentence

“John calls friends up”

Order of processing

1. Expand pivot [a] to [b]

2. Pivot for [b] is [f]

3. Connecting to [b] goes over [e]

4. Recursive expansion to [g]

5. Further connecting to [b] goes over [d]

6. Recursive expansion to [h]

7. Recursive expansion to [c]

THE CHART AS A DATA STRUCTURE

Components - edges

A two-dimensional matrix of *edges*

Edges are possibly partial rule instantion over a substring

Edges are indexed by start and end string positions

Properties of edges

Dot in a rule right-hand side indicates degree of completion

Active **edges (incomplete items) partial right-hand side**

Passive **edges (complete items) full right-hand side**

Fundamental rule

$[n_1,n_2,A \rightarrow B_1 \ldots B_{i-1} \bullet B_i \ldots B_n]$ and $[n_2,n_3,B_i \rightarrow C^+ \bullet]$

yields $[n_1,n_3,A \rightarrow B_1 \ldots B_i \bullet B_{i+1} \ldots B_n]$

CHART PARSING – 3 OPERATORS

Predictor

Applied to state with a non-terminal at right of the dot – rule expansion

S -> • VP, [0,0] yields states VP -> • Verb, [0,0] and VP -> • Verb NP, [0,0]

Scanner

Applied to state with a terminal at right of the dot – top-down input

Supports disambiguation of input

VP -> • Verb NP, [0,0] yields state VP -> Verb • NP, [0,1]

Completer

Applied to state with the dot in the rightmost position – rule completion

Completer looks at states with adjacent position expecting the category parsed

NP -> Det Nominal •, [1,3] & VP -> Verb • NP, [0,1] gives VP -> Verb NP •, [0,3]

MODIFYING A CHART FOR GENERATION PURPOSES (Kay 1996)

Motivation

Exploiting the chart for avoiding recomputation

Processing strategy in dependency of the state of the chart

Functionality

Chart is organized by semantic index values rather than by string positions

Each active edge is looking for a passive edge with the right index

A successful result is a passive edge that "uses up" all of the input semantics

Assessment

Much better than naive searching, but still some specific problems

AN EXAMPLE (1)

Example input expression

r:run(r),past(r),fast(r),arg1(r,j),name(j,John)

Example grammar

s(x) -> np(y) vp(x,y)

vp(x) -> vp(x) adv(x)

Lexicon entries – instantiating relevant ones yields the initial state of the chart

John	**np(x)**	**x:name(x,John)**
ran	**vp(x,y)**	**x:run(x),arg1(x,y),past(x)**
fast	**adv(x)**	**x:fast(x)**
quickly	**adv(x)**	**x:fast(x)**

AN EXAMPLE (2)

Processing steps

Interaction between "John" and "ran" yields (5)
Not finished, since not all input specifications have been consumed

	Word	*Category*	*Semantics*
(1)	**John**	**np(j)**	**j:name(j,John)**
(2)	**ran**	**vp(r,j)**	**r:run(r),arg1(r,j),past(r)**
(3)	**fast**	**adv(r)**	**r:fast(r)**
(4)	**quickly**	**adv(r)**	**r:fast(r)**
(5)	**John ran**	**s(r)**	**r:run(r),arg1(r,j),past(r) j:name(j,John)**

AN EXAMPLE (3)

Processing steps

Interaction between “ran” and “fast” yields (6)

Not finished, since no sentence found yet

	Word	*Category*	*Semantics*
(1)	**John**	**np(j)**	**j:name(j,John)**
(2)	**ran**	**vp(r,j)**	**r:run(r),arg1(r,j),past(r)**
(3)	**fast**	**adv(r)**	**r:fast(r)**
(4)	**quickly**	**adv(r)**	**r:fast(r)**
(5)	**John ran**	**s(r)**	**r:run(r),arg1(r,j),past(r) j:name(j,John)**
(6)	**ran fast**	**vp(r,j)**	**r:run(r),arg1(r,j),past(r),fast(r)**

AN EXAMPLE (4)

Processing steps

Interaction between “ran” and “quickly” yields (7)

Not finished, since not all input specifications have been consumed

	Word	*Category*	*Semantics*
(1)	**John**	**np(j)**	**j:name(j,John)**
(2)	**ran**	**vp(r,j)**	**r:run(r),arg1(r,j),past(r)**
(3)	**fast**	**adv(r)**	**r:fast(r)**
(4)	**quickly**	**adv(r)**	**r:fast(r)**
(5)	**John ran**	**s(r)**	**r:run(r),arg1(r,j),past(r) j:name(j,John)**
(6)	**ran fast**	**vp(r,j)**	**r:run(r),arg1(r,j),past(r),fast(r)**
(7)	**ran quickly**	**vp(r,j)**	**r:run(r),arg1(r,j),past(r),fast(r)**

AN EXAMPLE (5)

Processing steps

Interaction between “John” and “ran fast” yields (8) (similarly (9))

Finished, since all input specifications have been consumed

	Word	*Category*	*Semantics*
(1)	**John**	**np(j)**	**j:name(j,John)**
(2)	**ran**	**vp(r,j)**	**r:run(r),arg1(r,j),past(r)**
(3)	**fast**	**adv(r)**	**r:fast(r)**
(4)	**quickly**	**adv(r)**	**r:fast(r)**
(5)	**John ran**	**s(r)**	**r:run(r),arg1(r,j),past(r) j:name(j,John)**
(6)	**ran fast**	**vp(r,j)**	**r:run(r),arg1(r,j),past(r),fast(r)**
(7)	**ran quickly**	**vp(r,j)**	**r:run(r),arg1(r,j),past(r),fast(r)**
(8)	**John ran fast**	**s(r)**	**r:run(r),arg1(r,j),past(r),j:name(j,John),fast(r)**
(9)	**John ran quickly**	**s(r)**	**r:run(r),arg1(r,j),past(r),j:name(j,John),fast(r)**

PROCESSING THE EXAMPLE – SUMMARY

1. Lexical entries which subsume input specifications (variables instantiated)

2. Moving *ran* to the chart after moving *John* there (5) is built, due to the S rule

3. Since not all of the input is subsumed, it is put on the agenda

4. Moving *fast* to the chart yields interaction with *ran* (6) due to the VP rule

5. Moving *quickly* to the chart yields interaction with *ran* (7) due to the VP rule

6. No interaction between the VPs (6) or (7) and the adverbs (3) or (4),
since this would use parts of the semantic twice

7. Interaction between *John* and either VP (6) or (7) yields a sentence so that

- the entire expression is used

- no specification is used twice

A PROBLEM WITH CHART GENERATION

The observation

Intersective modification cause efficiency problems

Many unwanted combinations with any order of modifiers built

Reason

Both the syntactic category and the semantic index

compatible in structures before and after rule application

otherwise scope or syntactic ordering constraints prevent combinations

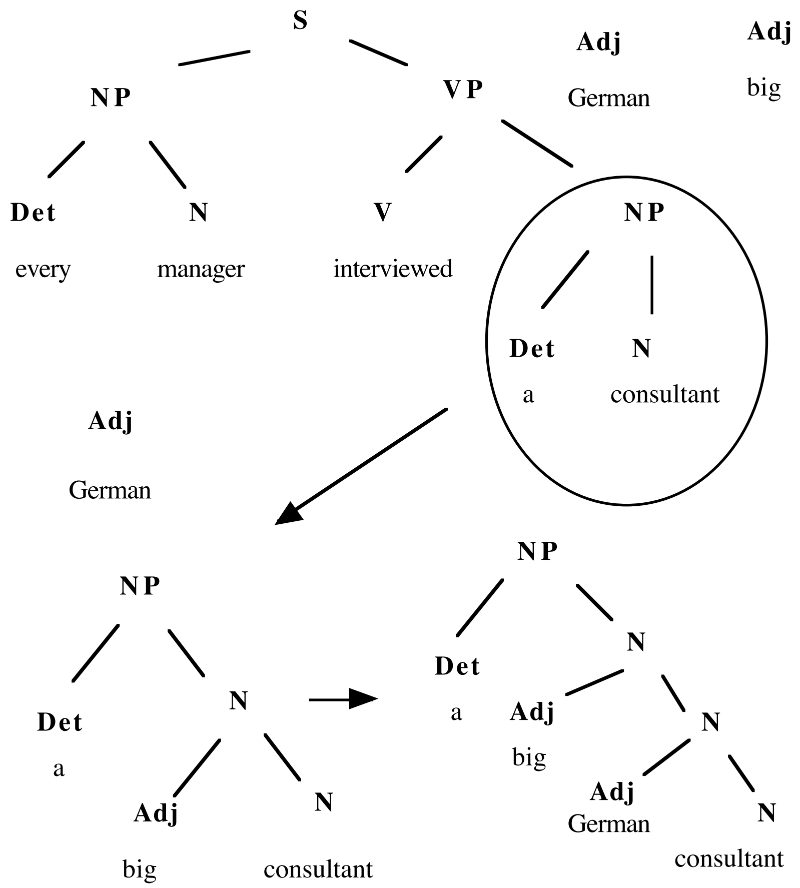
Measure

Separate the generation process in 2 phases

(Semi-lexicalist approach [Carroll et al. 1999])

Intersective modifiers adjoined in a postprocess (they do not change categories)

AN EXAMPLE (THE SEMI-LEXICALIST APPROACH)



Phase 1

Processing without modifiers

Phase 2

Adjoining intersective modifiers

1. big

2. German

ASSESSING THE SEMI-LEXICALIST APPROACH

Efficiency measures

Corpus	Standard chart	Two phase generation
44 Short dialog examples	856 edges / 5.4 msec	501 edges / 3.3 msec
First sentence below	923 edges / 5.6 msec	314 edges / 1.8 msec
Second sentence below	4710 edges / 54.8 msec	776 edges / 4.3 msec

“a manager in that office interviewed a new consultant from Germany”

“our manager organized an unusual additional weekly department conference”

(modifier order not constrained by the Grammar, 4! x 2 strings generated)

Coverage

Large grammar of English (including conjunction, extraposition, ellipsis)

Linguistic Grammars online: http://hpsg.stanford.edu/hpsg/lingo.html

FEATURING COORDINATE STRUCTURES (White 2004)

The general approach

Similar motivation as the semi-lexicalist approach

Different format of semantics and integrated process organization

Some measures

***Chunking* and *flattening* – identify subproblems (e.g., separate relative clause)**

Efficient data structures in the implementation

Lexical loop up supported by *indexing* scheme

Edge *pruning* and anytime search to address relatively free word orders

Efficiency

All measures contributing, best realizations found way under a second

OpenCCG realizer successfully used in two dialog systems

HANDLING DISJUNCTIVE INPUTS (White 2006)

Motivation

Language planning components produce sets of reasonable expressions

- **Paraphrases with no preferences among them**
- **Alternatives within context widely interchangable**
- **Surface realizer may decide**

Representation alternatives

Underspecified expressions

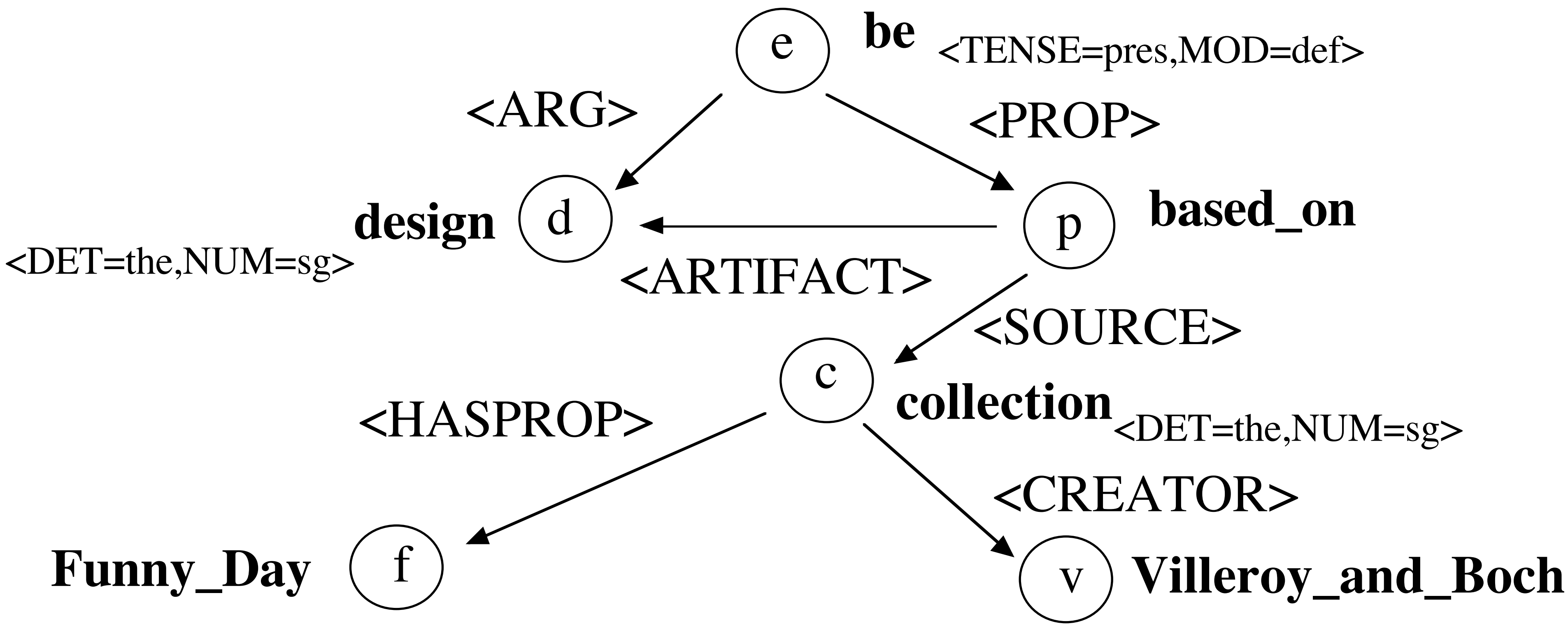
Explicit disjunctions (the alternative used here)

Functionality

Generate most alternatives in parallel (overlapping substructures)

Decide on the basis of corpus frequencies of surface expressions

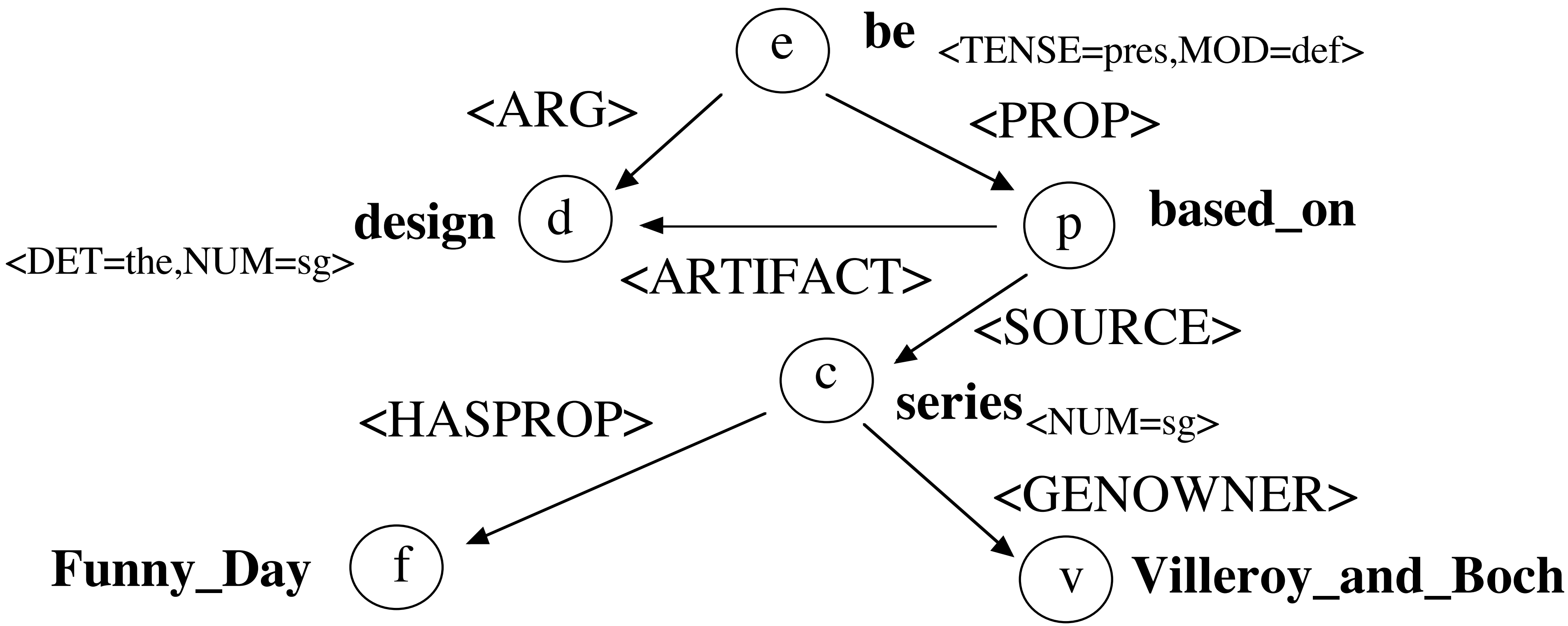
EXAMPLE REPRESENTATION (1)



Semantic dependency graph for

"The design is based on the Funny Day collection by Villeroy and Boch"

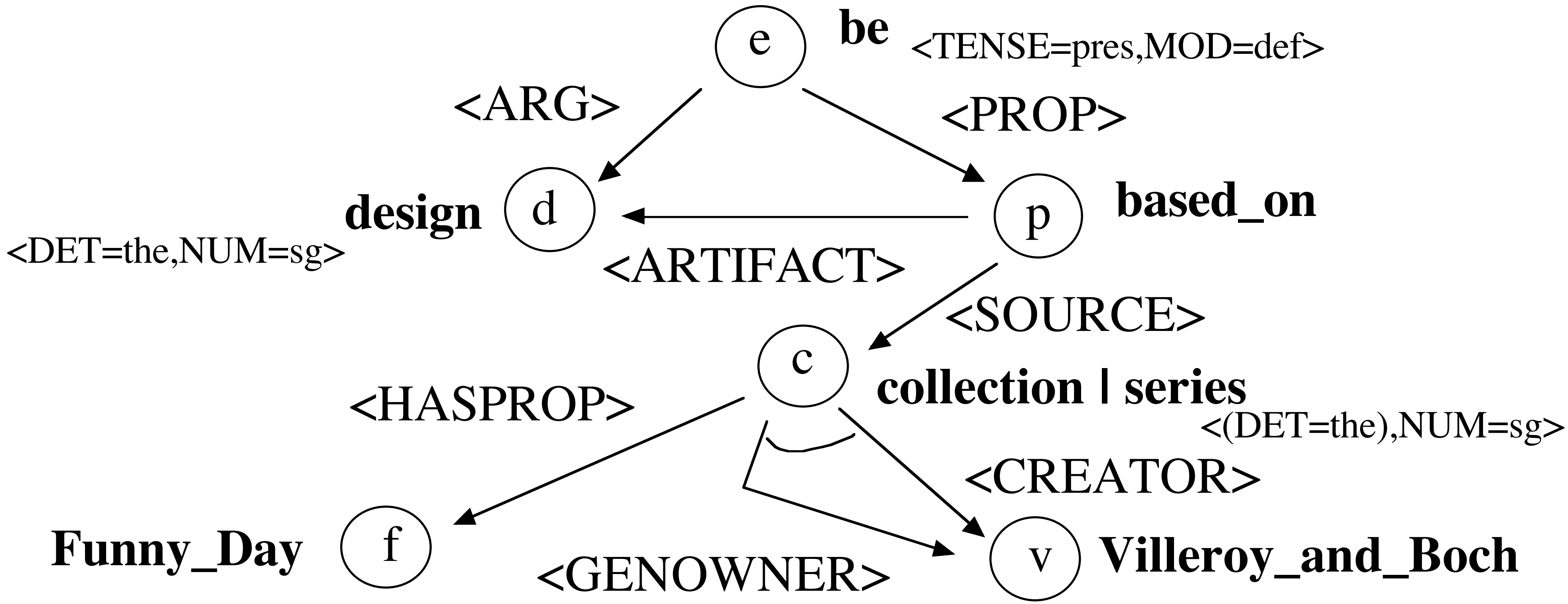
EXAMPLE REPRESENTATION (2)



Semantic dependency graph for

“The design is based on Villeroy and Boch's Funny Day series”

EXAMPLE REPRESENTATION (3)



Disjunctive Semantic dependency graph covering

“The design is based on (the Funny Day (collection | series)

by Villeroy and Boch | Villeroy and Boch's Funny Day (collection | series))”

THE PROCEDURE (SKETCH)

Flattening

Preprocessing step - array of elementary predications, alternations and options

Through tree traversal with incrementally building alternative groups

Edges

Edges associated with bit vectors to record coverage of alternatives

Lexical instantiation

Returns non-overlapping matches with coverage indicating bit vectors

Derivation

Edges may be introduced as alternatives

Edge combination involves a coverage check

Unpacking

Realizations recursively unpacked, filtering duplications

EVALUATION

Setting

Trigram language model used for scoring alternatives

Single best output and 10-best realizations

Efficiency gain measured against sequential processing

Results

	10-best two-stage		1-best anytime	
	time	**edges**	**time**	**edges**
disjunctive	**1.1**	**602**	**0.5**	**281**
sequential	**5.6**	**3550**	**4.1**	**2854**