Implementing Argument Alternations II

Miriam Butt (Konstanz)
and
Tracy Holloway King (PARC)

ESSLLI 2006, August, Malaga

parc

Lexical Rules

Passive via Lexical Rule:

```
PASS(SCHEMATA)= {SCHEMATA
                |SCHEMATA
                 (^ PARTICIPLE)=c PASS
                 (^ OBJ)–>(^ SUBJ)
                 { (^ SUBJ)–>(^ OBL-AG)
                 |(^ SUBJ)–>NULL}}.
```

parc

Lexical Rules

"Dative Shift" in English:

Kim gave a bone to the dog.	Kim gave the dog a bone.
SUBJ OBJ OBL	SUBJ OBJ OBJ-TH

This is also a clear candidate for a Lexical Rule:

```
DAT-SHIFT(SCHEMATA) = { SCHEMATA
                        (^ OBL PCASE) =c TO
                      |SCHEMATA
                        (^ OBJ) –> (^ OBJ-TH)
                        (^ OBL) –> (^ OBJ)}.
```

parc

Dative Shift

Conventional Linguistic Wisdom:

Not all ditransitive verbs allow for Dative Shift

I lowered the box to John.	Ann yelled the news to Beth.
? I lowered John the box.	? Ann yelled Beth the news.

But Bresnan and Nikitina (2003) show via corpus work that this is a fallacy of arm-chair linguistics.

parc

Dative Shift

Bresnan and Nikitina (2003):

> Therefore, when he got to purgatory, Buddha **lowered him the silver thread of a spider** as his last chance for salvation. www.inch.com/ fujimura/ImofGrmain.htm
>
> I think he was poking fun at the charges that Blackmore has been making that he chronically forgets words — he went over to Jon Lord during 'Smoke' and seemed to be getting Jon to **yell him the words**!! www.thehighwaystar.com/reviews/namerica/asbuandr.htm

parc

Dative Shift

So ditransitives generally can undergo an argument alternation.

Question: why does this alternation exist?

a) Processing Reasons (see Wasow 2002)
- Put heavy NPs at the very end (heavy NP-shift)
- Put pronouns near the verb (most of the examples deemed to be bad by linguists are good if you substitute a pronoun).

parc

Dative Shift

Question: why does this alternation exist?

b) Semantic Reasons
- OBJ supposed to be the more "affected" thing.

(1) John taught Pashto to the CIA agents.

(2) John taught the CIA agents Pashto.

Entailment: They learned Pashto in (2), but not in (1).

parc

Load/Spray

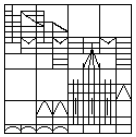
Same with the famous Load/Spray Alternation (see Levin and Rappaport Hovav 2005).

(1) John loaded hay onto the wagon.

(2) John loaded the wagon with hay.

Entailment: wagon is full in (2), but not in (1).

parc



Implementation

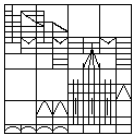
The Dative-Shift **Lexical Rule** is easy to implement.

For the **Processing Preferences**, we could use XLE's inbuilt OT constraints (inspired by Optimality Theory).

XLE

For the **Semantics**, we could think about adding a feature (if this is useful). We cannot do much more because LFG is a theory of syntax, not of semantics — but we would like to anticipate the semanticists' needs.

parc Palo Alto Research Center



Dative-Shift and Passive

Passives and Datives Interact

(1) The bone was given to the dog (by John).

(2) The dog was given the bone (by John).

How to implement this? XLE

```
DITRANS(P) = @(PASS @(DAT-SHIFT (^ PRED) =
                      'P<(^ SUBJ) (^ OBJ) (^ OBL)>'))
```

parc Palo Alto Research Center

Templates/Lexical Rules

We have now stacked the passive template on top of the dative shift one.

- Lexical Rules can interact with one another unproblematically (but feeding/bleeding!).
- We are beginning to create something of a Template Hierarchy. What is the status of this?

parc Palo Alto Research Center

The lexicon à la Flickinger

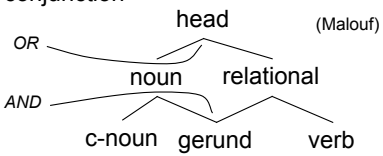
A hierarchical structure of classes

- Each class represents some piece of syntactic information
  bakes belongs to:
  - the third-person singular present-tense class (like *appears*)
  - the transitive/intransitive class (like *cooked*)
  - and others
- Classes may be subclasses of other classes
- Classes may partition other classes along several dimensions

parc Palo Alto Research Center

The HPSG lexicon: a type hierarchy

- More specific types *inherit* information from less specific
- Types and subtypes:
  - A mathematical relation between structures: AND/OR lattice
  - Different subtypes represent alternatives/disjunction
  - Multiple supertypes represent conjunction



- LFG does not use typed feature structures for lexical generalizations

  … but type inheritance is not the only (best?) way to express generalizations

parc
Palo Alto Research Center

LFG: Relations between descriptions

LFG can encode linguistic generalizations as relations between descriptions of structures

- LFG functional description is a collection of equations
- These can be named
- This name can stand for those equations in linguistic descriptions
- Named descriptions are referred to as *templates*
- Interpretation: Simple substitution
  Template-description is substituted for template-name that appears in (is invoked by) another description

parc
Palo Alto Research Center

3SG and PRESENT templates

3SG = (^ SUBJ PERSON) = 3
(^ SUBJ NUM) = SG.

"3SG names (^ SUBJ PERSON)=3 (^ SUBJ NUM)=SG"

PRESENT = (^ TENSE) = PRES.

@ marks invocation

Substitute (^ TENSE)=PRES for @PRESENT in other descriptions

parc
Palo Alto Research Center

Templates enable hierarchical generalizations

- Template definitions can refer to other templates by name
  - E.g. further divide 3SG into:

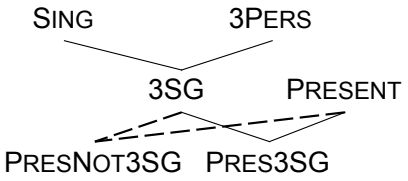
    3PERS = (^ SUBJ PERSON) = 3.
    SING = (^ SUBJ NUM) = SG.

    then 3SG = @3PERS @SING.
- Hierarchy of references represents inclusion hierarchy of named descriptions
- Frequently repeated subdescriptions
  - specified in one place
  - effective in many

parc
Palo Alto Research Center

Hierarchy of template invocations

Sharing in verb agreement



PRESNOT3SG = ~@3SG @PRESENT.

⇒ ~[@SING @3PERS] ⇒ ~[(^ SUBJ NUM)=SG
(^ SUBJ PERS=3]

- Boolean combinations of template references
  (just like ordinary descriptions)
- Sharing is distinct from mode of combination

parc
Palo Alto Research Center

Templates with parameters: Valency

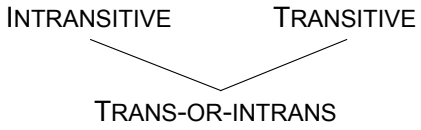
Pargram convention:
Parameters begin with _

- TRANS-OR-INTRANS(_P) =
  { (^ PRED) = '_P<SUBJ, OBJ>'
  | (^ PRED) = '_P<SUBJ>' }.
- PRED value as a parameter of the template
  @TRANS-OR-INTRANS(bake)
  ⇒ { (^ PRED) = '**bake**<SUBJ, OBJ>'
  | (^ PRED) = '**bake**<SUBJ>' }
- Arguments can substitute for any part of an f-description
  - Attributes
  - Values
  - Semantic relation-names
  - Descriptions

parc
Palo Alto Research Center

Valency hierarchy

- TRANS-OR-INTRANS(P) =
  { @INTRANSITIVE(P) | @TRANSITIVE(P) }.
- INTRANSITIVE(P) = (^ PRED)='P<SUBJ>



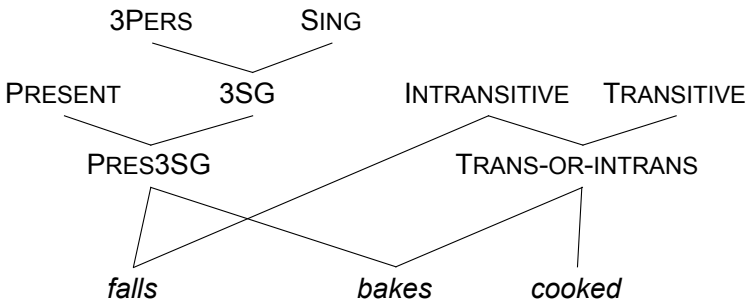
parc
Palo Alto Research Center

Templates and generalizations: *bakes*

- bakes: @TRANS-OR-INTRANS(bake)
  @PRES3SG
- TRANS-OR-INTRANS(p): shared by *eat, cooked,…*
- PRES3SG: shared by *appears*, *goes*, *cooks*,…
- PRESENT:
  – used by PRES3SG template
  – shared by *bake*, *laugh*, etc.

parc
Palo Alto Research Center

Lexical sharing



parc
Palo Alto Research Center

Type hierarchy vs. templates

- Templates can play the same role as hierarchical type systems in theories like HPSG
- A notational device for factoring descriptions
  - Interpreted as simple substitution
  - Not part of a formal ontology
  - Do not require an elaborate mathematical characterization

parc
Palo Alto Research Center

Templates also invoked by Rules

- Rule annotations can also call templates
  - Global changes, typo prevention
- Example: adjunct annotation

PP: ! $ (^ ADJUNCT) (! ADJ-TYPE)=VP
ADVP: ! $ (^ ADJUNCT) (! ADJ-TYPE)=VP

ADJ(_T) = ! $ (^ ADJUNCT) (! ADJ-TYPE)=_T.

PP: @(ADJ VP) PP: @(ADJ NP)
ADVP: @(ADJ VP) ADVP: @(ADJ S)

parc
Palo Alto Research Center

Templates: Rules

Example: null pronouns

Push it!

NULL-PRON(_P) = (_P PRED)='pro'
(_P PRON-TYPE)=null.

VPimp --> VP: @(NULL-PRON (^ SUBJ)).

VPimp --> VP: (^ SUBJ PRED)='pro'
(^ SUBJ PRON-TYPE)=null.

parc
Palo Alto Research Center

Templates and "Principles"

- Subject principle: every verb has a subject.
- Implementaton:

  VERB = (^ SUBJ).

  – Put @VERB in every verbal entry.

  or

  – Put @VERB in the templates called by the verbal entries.

parc Palo Alto Research Center

Lexical Rules

- Theoretical construct
- Templates can often achieve the same result
  - Disjunction of several templates
  - Parameterization of a complex template

Lexical Rules are not All-Powerful:
- great for Argument Deletion or Alternation
- not suitable for Argument Addition

parc Palo Alto Research Center

Benefactives

Benefactives can appear with many (?all) verbs in English.

Mary ran **for cancer research**.

Mary baked a cake **for Jane**.

Mary baked **Jane** a cake. Dative Shift!

parc Palo Alto Research Center

Benefactives

In German, the distribution of "free datives" is different, but not well described.

Maria buk **Jana** einen Kuchen**.** (German)
`Maria baked Jana a cake.'

Maria kaufte **ihm** einen Kuchen**.** (German)
`Maria bought him a cake.'

*Maria schwamm **ihm.**
`Maria swam (for) him.'

parc Palo Alto Research Center

Representation

- In many languages benefactives act like arguments (e.g., dative shift, German dative)
- How can we create appropriate lexical entries?
  - want to capture generalizations
  - control interactions with other argument alternation processes

Expanded Lexical Entries

- Could list the benefactive alternation in the lexical entries
  - go V { (^PRED)='_P<(^SUBJ)>'
    |(^PRED)='_P<(^SUBJ)(^BEN)>'}
  - miss generalization about distribution
  - prone to error in entries
- Using templates can help
  - go V @(V-SUBJ go).
  - V-SUBJ(_P) = { (^PRED)='_P<(^SUBJ)>'
    |(^PRED)='_P<(^SUBJ)(^BEN)>'}

Lexical Rules

- Expanding templates does not capture regularity
- Would like something like the passive lexical rule

```
PASS(_SUBCAT) =
   {  _SUBCAT   (^PASS)=-  "Mary baked the cake"
    |_SUBCAT
     (^OBJ)-->(^SUBJ)   (^PASS)=+
     { (^SUBJ)-->(^OBJ-AG) "The cake was baked by Mary"
       |(^SUBJ)-->NULL  "The cake was baked"
   }}
```

Benefactive Lexical Rule

- From a theoretical perspective, a benefactive lexical rule seems simple

```
bene(_SUBCAT) =
   _SUBCAT
   NULL --> (^ BEN).
```

  - For the passive, SUBJ became NULL
    (^ PRED)='bake<NULL,(^SUBJ)>'
  - For the benefactive, NULL becomes BEN

NULL --> (^ BEN)

- This simple looking statement has serious implementational effects
- Where in the argument slots should the new argument be realized?
  - (^PRED)='bake<(^SUBJ)(^OBJ)(^BEN)>'
  - (^PRED)='bake<(^SUBJ)(^BEN)(^OBJ)>'
  - (^PRED)='bake<(^BEN)(^SUBJ)(^OBJ)>'

Linguistically probably an OBL-GO and want first option since LFG assumes a hierarchy of Grammatical Functions.

A-Str and F-Str

A-str Hierarchy: agent < goal/exp < th/pt

GF Hierarchy: SUBJ < OBJ < OBJ-TH < OBL

Linking Theory tries to match up these hierarchies as best as possible, but allows for variation.

But Linking Theory, as seemingly simple as it is, has not been implemented.

Still, XLE encodes argument structure as separate from GFs (although it doesn't make this obvious).

Representation of PREDs

- (^PRED)='seem<(^OBL)(^COMP)>(^SUBJ)'
  - It seems to me that this is difficult.
- Internal representation:

PRED(var(0), seem)	citation form of verb
lex_id(var(0),1)	unique index of verb
SUBJ(var(0), var(1))	name of GF
nonarg(var(0),1,var(1))	(non)arg slot of GF
OBL(var(0),var(2))	
arg(var(0),1,var(2))	
COMP(var(0),var(3))	
arg(var(0),2,var(3))	

XLE

Renaming Arguments

- Renaming an argument affects the name part of the PRED facts
  - OBJ(var(0), var(1)) ==> SUBJ(var(0), var(1))
  - main issue is not to create two arguments of the same name (violation of uniqueness in LFG)

Deleting Arguments

- Deleting an argument removes the name and alters the (non)arg fact
  - SUBJ(var(0), var(1)) ==> 0
  - arg(var(0),1,var(1)) ==> arg(var(0),1,NULL)
- NULL holds a slot in the PRED, but is not subject to completeness and coherence
- Since the (non)arg fact remains, no other arguments of the PRED are affected

Adding an Argument

- Need to know both name and (non)arg number
  - Name is usually easy (e.g. BEN, OBL-GO)
- Number (position among the PRED's arguments) is much more difficult
  - Make it the first
  - Make it the last
  - Specify for each case

New Argument is First

- If the new argument is first, all the other arguments need to be (automatically) moved over one slot
  - NULL --> (^ BEN)
  - BEN(var(0),var(3))
    arg(var(0),1,var(3))
    SUBJ(var(0),var(1))
    arg(var(0),2,var(1))
    OBJ(var(0),var(2))
    arg(var(0),3,var(2))
  - (^ PRED)='bake<(^BEN)(^SUBJ)(^OBJ)>'

New Argument is Last

- If the new argument is last, the correct new slot number must be tallied, but the existing arguments are not affected
- NULL --> (^ BEN)
  - BEN(var(0),var(3))
    arg(var(0),3,var(3))
    SUBJ(var(0),var(1))
    arg(var(0),1,var(1))
    OBJ(var(0),var(2))
    arg(var(0),2,var(2))
  - (^ PRED)='bake<(^SUBJ)(^OBJ)(^BEN)>'

New Argument is Specified

- If the new argument is specified, the correct new slot number is known, but
  - any arguments following it must be incremented
  - need to be careful not to leave gaps in numbering
- NULL --> (^ BEN), arg(%%,2,%%)
  - BEN(var(0),var(3))
    arg(var(0),2,var(3))
    SUBJ(var(0),var(1))
    arg(var(0),1,var(1))
    OBJ(var(0),var(2))
    arg(var(0),3,var(2))
  - (^ PRED)='bake<(^SUBJ)(^BEN)(^OBJ)>'

Theory vs. Implementation

- Implementationally, all three possibilities feasible
  - first, last, specified
  - specifying provides the most control but also the most room for error (creation of invalid PREDs)
- Theoretically, these constructions are not well understood, but probably inserting the BEN last is desirable.
  - XLE implementation is waiting for theoretical guidance in making a choice
  - The xfr system can manipulate PREDs in this way, but this does not help parsing

Other Possibility: Benefactives as the Reverse of Passive

- Have a benefactive argument slot in all verb frames
  - (^PRED)='_P<(^SUBJ)(^BEN)>'
  - (^PRED)='_P<(^SUBJ)(^OBJ)(^BEN)>'
  - …
- Have a lexical rule to rewrite this slot to NULL

Benefactive Lexical Rule

- bene(_SUBCAT) -->
  { _SUBCAT "Mary baked a cake for Jane"
  |_SUBCAT
  (^BEN)-->NULL "Mary baked a cake"
  }
- Works technically, but non-benefactives have unintuitive null argument
  - (^PRED)='bake<(^SUBJ)(^OBJ) NULL>'

Interactions with Other Lexical Rules

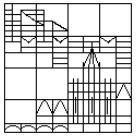
- Other lexical rules can alter the benefactive slot or pass it through
  - similar to any other argument
  - passive+benefactive
    - Mary baked a cake for Jane no pass, no bene-del
      (^PRED)='bake<(^SUBJ)(^OBJ)(^BEN)>'
    - A cake was baked for Jane pass, no bene-del
      (^PRED)='bake<NULL (^SUBJ) (^BEN)>'
    - A cake was baked pass, bene-del
      (^PRED)='bake<NULL (^SUBJ) NULL>'

Interactions continued

- Passive could act on benefactive if this was appropriate
  - I jumped for Mary
    (^PRED)='jump<(^SUBJ)(^BEN)>'
  - (For) Mary was jumped by me
    (^PRED)='jump<NULL (^SUBJ)>'
- Ordering of application is important since cannot rename an argument once it is made NULL
  - pass applied before bene-del above

Benefactives Summary

- Intuitively, benefactives add an argument to a predicate
- Implementationally, this is complicated because where this argument is in the PRED must be specified
- One possibility is to always posit the benefactive and then delete it via a lexical rule, as with passive
- However, this possibility is not supported by crosslinguistic facts, e.g., Bantu Applicatives.



Applicatives/Morphology

Chichewa (extra beneficiary argument licensed via morphology on the verb).

One would not posit a BEN argument for all verbs in these types of languages, since the morphology provides the clue!