Continuation Fest 2008
April 13, 2008

Delimited Continuation in the Grammar of Japanese

Rui Otake
otake@mail.tains.tohoku.ac.jp

Linguistic Knowledge

- Syntax:

  Is the given string of words are grammatical expression of that language?

- Interpretations:

  What does it mean? (Semantics)

  How to spell it? (Orthography)

  How to pronounce it? (Phonology)

Syntax

- 太郎が走った (Taro ran)
- 太郎が花子を見た (Taro saw Hanako)
- * 太郎が花子を走った
- * が太郎見たを花子

Type Logical Grammar

Type T ::= s|n|... (atomic)
| $T_1 \rightarrow T_2$ (function)

Signature:

Taroo	⊢	n
ga	⊢	$n \rightarrow n_{subj}$
waratta	⊢	$n_{subj} \rightarrow s$

Type Logical Grammar

> : backward application

X ⊢ T : "X is a grammatical expression of type T"

$$\dfrac{\dfrac{\text{Taroo} \vdash n \quad \text{ga} \vdash n \rightarrow n_{subj}}{(\text{Taroo} > \text{ga}) \vdash n_{subj}} \rightarrow E \qquad \text{waratta} \vdash n_{subj} \rightarrow s}{((\text{Taroo} > \text{ga}) > \text{waratta}) \vdash s} \rightarrow E$$

Type Logical Grammar

Word learning involves type inference

Unknown word: xxxxxxx ⊢ ???

((Taroo > ga) > xxxxxxx) ⊢ s

Taroo ⊢ n

ga ⊢ n→n_{subj}

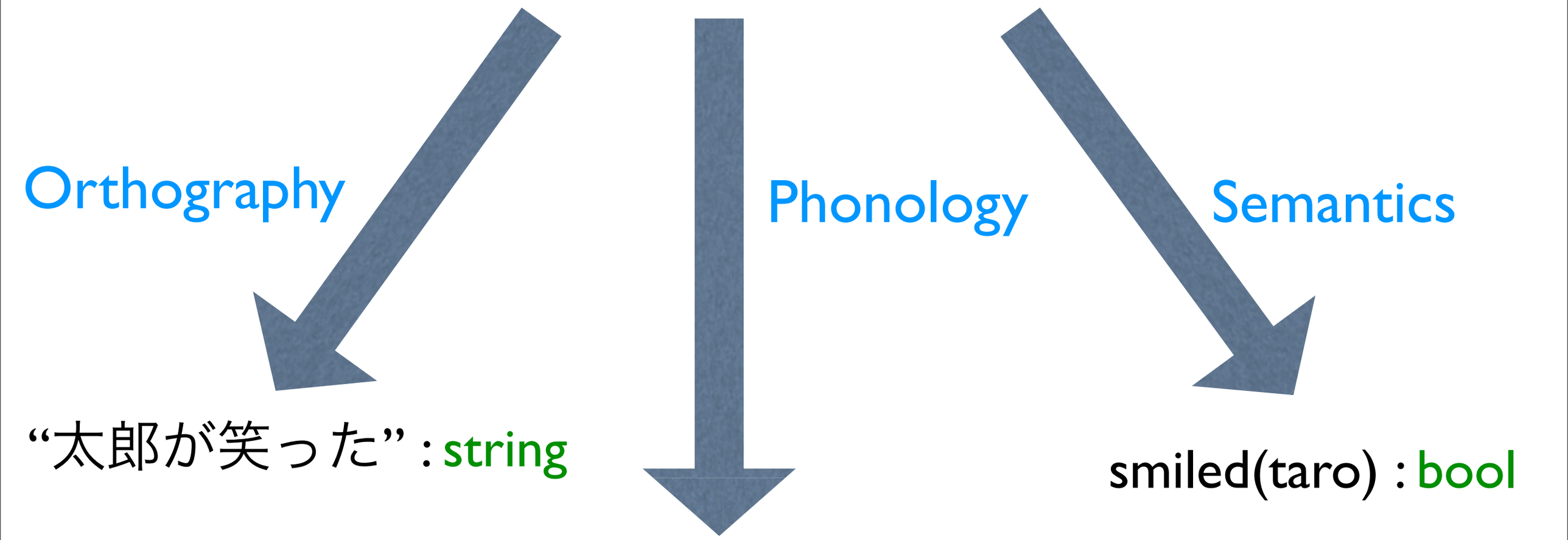
Inferred type:

xxxxxxx ⊢ n_{subj}→S

Intererpretations in Different Domains

Grammaticality : Typeability

((Taroo > ga) > waratta) ⊢ s



Orthography Domain

```
Signature:  Taroo     ⊢ n
            ga        ⊢ n → n_subj
            waratta   ⊢ n_subj → s
```

```
Interpretation:
  Type s    = string
  Type n    = string
  Taroo     = “太郎”                  : string
  ga        = λx.x ⌢ “が”             : string → string
  waratta   = λx.x ⌢ “笑った”         : string → string
```

Orthography Domain

`Taroo`	=	“太郎”	: string
`ga`	=	λx.x ⌒ “が”	: string → string
`waratta`	=	λx.x ⌒ “笑った”	: string → string

((`Taroo` > `ga`) > `waratta`) : string

⇝ (“太郎” > (λx.x ⌒ “ が”)) > (λx.x ⌒ “笑った”)

⇝ (“太郎 ⌒ “が”) > (λx.x ⌒ “笑った”)

⇝ (“太郎” ⌒ “が”) ⌒ “笑った”

⇝ “太郎が笑った”

Phonology Domain

Signature:

Taroo	⊢	n
ga	⊢	$n \rightarrow n_{subj}$
waratta	⊢	$n_{subj} \rightarrow s$

Interpretation:

Type s	=	phon	
Type n	=	phon	
Taroo	=	/ta'ro:/	: phon
ga	=	λx.x + /ga/	: phon → phon
waratta	=	λx.x + /waratta/	: phon → phon

Phonology Domain

Taroo	=	/ta'ro:/	: phon
ga	=	λx.x + /ga/	: phon → phon
waratta	=	λx.x + /waratta/	: phon → phon

NB: + denotes concatnation of two phonetic representations, e.g. /m/ + /a/ = /ma/

((Taroo > ga) > waratta) : phon

⇝ (/ta'ro:/ > (λx.x + /ga/)) > (λx.x + /waratta/)

⇝ (/ta'ro:/ + /ga/) > (λx.x + /waratta/)

⇝ (/ta'ro:/ + /ga/) + /waratta/

⇝ /ta'ro:gawaratta/

Semantics Domain

Signature:

Taroo	⊢	n
ga	⊢	$n \rightarrow n_{Nom}$
waratta	⊢	$n_{Nom} \rightarrow s$

Interpretation:

Type s	=	bool	
Type n	=	ind	
Taroo	=	taro	: ind
ga	=	λx.x	: ind → ind
waratta	=	λx.smiled(x)	: ind → bool

Semantics Domain

`Taroo`	=	taro	: ind
`ga`	=	λx.x	: ind → ind
`waratta`	=	λx.smiled(x)	: ind → bool

`((Taroo > ga) > waratta)` : bool

⇝ (taro > (λx.x)) > (λx.smiled(x))

⇝ taro > (λx.smiled(x))

⇝ smiled(taro)

Empty Categories

Null pronouns

```
pro ⊢ n
```

pro	: ind	
“”	: string	null
//	: phon	null

```
((Taroo > ga) > (pro > kaita)) ⊢ s
```

“太郎が書いた”	: string
/ta’ro:gakaita/	: phon
wrote(taro,pro)	: bool

Empty Categories

Relativizer (as type shifting operator)

REL	⊢ cn	→(n→s) → s

λnr.ιx.n(x)&r(x)	: (ind→ind)	→ (ind→bool) → ind
λnr. r “” ⌒ n	: string	→ (string→string) → string
λnr. r // + n	: phon	→ (phon→phon) → phon

((Taroo > ga) > kaita’) > (hon>REL) ⊢ n

“太郎が書いた本”	: string
/ta’ro:gakaitaho’n/	: phon
ιx. book(x) & wrote(taro,x)	: ind

Delimited Continuation in Japanese

- Quantifier
- Focus/Presupposition
- (Multiple) question
- Wh-island
- Split-quantifier

Quantifier

- Taroo-ga Hanako-o hometa.
  Taro-NOM Hanako-ACC praised
  'Taro praised Hanako.'

  praised(taro,hanako)

- Taroo-ga daremo-o hometa.
  Taro-NOM everyone-ACC praised
  'Taro praised everyone.'

  ∀x. praised(taro,x)

Quantifier

- [Taroo-ga ____-o hometa]
  Taro-NOM -ACC praised

  λx. praised(taro,x)

Quantifier

- [Taroo-ga ___-o hometa]
  Taro-NOM -ACC praised

[Taroo-ga Hanako-o hometa]
(λx. praised(taro,x)) hanako
⇝ praised(taro, hanako)

[Taroo-ga daremo-o hometa]
(λk.∀x.kx)(λx. praised(taro,x))

⇝ ∀x. praised(taro,x)

Quantifier

- [Taroo-ga ___-o hometa]
  Taro-NOM -ACC praised

  λx. praised(taro,x)

daremo = shift k. ∀x.kx

dareka = shift k. ∃x.kx

Whole sentence is now enclosed by *reset*.

Quantifier

daremo = shift k. ∀x.kx

reset ((Taroo > ga) > ((daremo > o) > hometa))

⇝ (reset ((taro > λx.x) > (((shift k. ∀x.kx) > λx.x) > λxy. praised(y,x))))

⇝(reset (taro > (((shift k. ∀x.kx) > λx.x) > λxy. praised(y,x))))

⇝ (reset (reset (∀x. (λy.taro > ((y > λx.x) > λxy.praised(y,x)))x)))

⇝ (reset (reset (∀x. taro > ((x > λx.x) > λxy.praised(y,x)))))

⇝ ∀x. praised(taro,x)

Quantifier

daremo = shift k. k “誰も”

(reset ((Taroo > ga) > ((daremo > o) > hometa)))

↝ (reset (“太郎が”) >

(((shift k.k “誰も”) > λx.x⁀ “を”) > λxy.y⁀x⁀ “褒めた”)))

↝ (reset (reset (λx.(“太郎が”) >

((x > λx.x⁀ “を”) > λxy.y⁀x⁀ “褒めた”))x))

↝ “太郎が誰もを褒めた”

Focus

- [Taroo-ga Hanako-mo hometa].
  Taro-NOM Hanako-also praised
  'Taro *also* praised *Hanako*.'

praised(taro,hanako) & ∃y≠hanako. praised(taro,y)

presupposition

x mo = shift k. kx & ∃y≠x.ky

Question

- [Taroo-ga dare-o home(masi)ta] ka?
  Taro-NOM who-ACC praised QUES
  'Whom did Taro praise?'

  Q λx. praised(taro,x)

dare = shift k. λx. kx
x ka = Q (reset x)

ka introduces *reset*.

Question

- Hanako-wa [Taroo-ga dare-o hometa] ka tazuneta.
  Hanako-TOP Taro-NOM who-ACC praised QUES asked
  'Hanako asked whom Taro praised.'

ask(hanako, Q λx. praised(taro,x))

dare = shift k. λx. kx
x ka = Q (reset x)

ka introduces *reset*.

Question

dare = shift k. λx. kx
x ka = Q(reset x)

reset (((Taroo > ga) > ((dare > o) > hometa)) ka)

↝ (reset (Q(reset (taro > (((shift k. λx. kx) > λx.x) > λxy.praised(y,x))))))

↝ (reset (Q(reset (reset (λx.(λx.(taro>((x>λx.x)>λxy.praised(y,x))))x)))))

↝ (reset (Q(reset (reset λx.(taro > ((x > λx.x) > λxy.praised(y,x)))))))

↝ (reset (Q(reset (reset λx.(taro > (x > λxy.praised(y,x)))))))

↝ Q λx. praised(taro,x)

Multiple Question

- Hanako-wa [Taroo-ga hon-o mita]-ka tazeneta.
  Hanako-TOP Taro-NOM a.book-ACC saw QUES asked
  'Hanako asked whether Taro saw a book.'

- Hanako-wa [Taroo-ga nani-o mita]-ka tazeneta.
  Hanako-TOP Taro-NOM what-ACC saw QUES asked
  'Hanako asked whom Taro saw.'

- Hanako-wa [dare-ga nani-o mita]-ka tazeneta.
  Hanako-TOP who-NOM who-ACC saw QUES asked
  'Hanako asked who saw whom.'

- Hanako-wa [dare-ga nani-o doosita]-ka tazeneta.
  Hanako-TOP who-NOM what-ACC did.what QUES asked
  'Hanako asked who did what to what.'

Multiple Question

- Hanako-wa [Taroo-ga hon-o mita]-ka tazeneta.
- Hanako-wa [Taroo-ga nani-o mita]-ka tazeneta.
- Hanako-wa [dare-ga nani-o mita]-ka tazeneta.
- Hanako-wa [dare-ga nani-o doosita]-ka tazeneta.

Type of question depends on the number of question words (cf. Printf problem)

Q saw(taro, a-book) : Q bool
Q λx.saw(taro, x) : Q (ind → bool)
Q λxy.saw(y, x) : Q (ind → ind → bool)
Q λxyR.R(y, x) : Q (ind → ind →
(ind → ind → bool) → bool)

Wh-island

✓ Hanako-wa [[dare-ga nani-o kaita]-ka tazeneta]-ka?

Hanako-TOP who-NOM what-ACC wrote QUES asked QUES

'Did taro ask who wrote what?'

X Hanako-wa [[dare-ga nani-o kaita]-ka tazeneta]-ka?

'*Who* did Hanako ask what *he* wrote?'

'*What* did Hanako ask who wrote *it*?'

Both dare and nani must be associated with

ka in the embedded clause (the nearest enclosing *reset*)

Split Quantifier

- [Dare-ga kaita hon]-mo omosiroi.
  who-NOM wrote book -every interesting.
  'Every book that a person wrote is interesting.'

$\forall x.\ \text{person}(x) \supset \text{interesting}(\iota y.\ \text{book}(y) \ \&\ \text{wrote}(x,y))$

$\text{dare} = \text{shift}\ k.\ \lambda x.\ kx$
$\text{mo} = \text{shift}\ k.\ \forall x.k(\text{reset}\ x)$

Summary

- Type Logical Grammar
  - Analogy with programming language
    - Non-typable code cannot run
    - ungrammatical expression cannot be interpreted.

Summary

- Use of delimited continuation
  - Quantifier
  - Focus/Presupposition
  - (Multiple) questions
  - Wh-island
  - Split-quantifier