"To say that polarity any *must be in the scope of a downward-entailing operator does not really explain the nature of* any, *just as the [Non-Entailment of Existence Condition] does not fully explain the nature of [Existential Polarity* Wh-*Phrases in Chinese]. [...] There must be something which forces* any *and Chinese [Existential Polarity* Wh-*Phrases] to be subject to the constraint which generates their distributions. This something should be some lexical property of NPIs though I do not know exactly what it is."*

—— 林若望 [Lin98, pp. 250f.]

a uniform semantics for Mandarin wh-NPs

martin jansche

the ohio state university

jansche@ling.ohio-state.edu

a first glance at the facts

(1)

你	喜歡	誰？
Nǐ	xǐhuān	shéi?
you	like	who

‘Who do you like?’

(2)

我	希望	誰	能	來	陪陪	我。
Wǒ	xīwàng	shéi	néng	lái	péipéi	wǒ.
I	wish	who	can	come	accompany	me

‘I wish somebody could come to accompany me.’

Example (33b) from [Lin96, ch. 4], illustrating the so-called Existential Polarity Wh (EPW) use of a *wh*-NP. See also [Lin98].

(3)

誰	都	可以	來。
Shéi	dōu	kěyǐ	lái.
who	even	may	come

‘Everybody can come.’

Example (2a) from [Lin96, ch. 3]. The *wh*-NP receives a universal (generalized distributive) interpretation.

(4)

你	喜歡	誰，	我	就	批評	誰。
Nǐ	xǐhuān	shéi,	wǒ	jiù	pīpíng	shéi.
you	like	who,	I	then	criticize	who

‘Whoever you like, I’ll criticize her/him.’

This is example (7b) from [Lin96, ch. 5], where it is classified as a bare conditional ‘donkey’ sentence.

where to go and how to get there

wanted

1. uniform denotations for all elements of each syntactic category
2. a compositional semantics compatible with a strongly lexicalist syntax/semantics interface
3. in the absence of syntactic displacement, a lexical semantic account of the scope of in-situ *wh*-elements
4. an explicit mechanism accounting for non-interrogative uses of *wh*-elements

basic assumptions

1. Interrogatives denote n-place relations(-in-intension), what Groenendijk and Stokhof [GS82] call 'question abstracts', the ingredients of what they later [GS97] call 'categorial theories' of questions (e.g. [Hau84, §5.3]).
2. These question denotations can be subject to various operations: exhaustification (question formation in the sense of [GS82]), existential closure, etc.
3. The lexical semantic properties of sentence embedders account for the different uses of *wh*-elements.
4. I focus on the simplest possible case: *wh*-NPs in unembedded argument positions.
5. I will assume type-theoretic notions and notation familiar from functional programming languages, specifically ML or, later on, Haskell.

stand-alone constituent questions

(5)

張三	愛	李四。
Zhāngsān	ài	Lǐsì.
Zhangsan	loves	Lisi

‘Zhangsan loves Lisi.’

(6)

誰	愛	李四？
Shéi	ài	Lǐsì?
who	loves	Lisi

‘Who loves Lisi?’

(7)

張三	愛	誰？
Zhāngsān	ài	shéi?
Zhangsan	loves	who

‘Who does Zhangsan love?’

(8)

誰	愛	誰？
Shéi	ài	shéi?
who	loves	who

‘Who loves who?’

sketch of the compositional analysis of in-situ wh

- generalizes to an even worse case than Montague's [Mon73]
- an application of Dekker's [Dek99] 'periscoping' technique

data Bool := False | True
data Ent := Z_3 | L_4 | W_5 | ...

$$
\begin{aligned}
[\![\textit{Zhāngsān}]\!] &:= \lambda q. \quad\;\;\; q\ (\lambda p.\, p\ \mathsf{Z}_3) \\
[\![\textit{Lǐsì}]\!] &:= \lambda q. \quad\;\;\; q\ (\lambda p.\, p\ \mathsf{L}_4) \\
[\![\textit{shéi}]\!] &:= \lambda q.\, \lambda x.\; q\ (\lambda p.\, p\ x)
\end{aligned}
$$

$[\![\textit{ài}]\!] := \lambda o'.\, \lambda s.\, o'\ (\lambda o.\, s\ (\lambda x.\, o\ (\lambda y.\, \mathsf{love}\ x\ y)))$
love :: Ent → Ent → Bool

$$
\begin{aligned}
&\llbracket \textit{ài} \rrbracket\ \llbracket \textit{shéi} \rrbracket \\
\equiv\ & \lambda o'.\, \lambda s.\, o'\ (\lambda o.\, s\ (\lambda x.\, o\ (\lambda y.\, \textsf{love}\ x\ y)))\ (\lambda q.\, \lambda y'.\, q\ (\lambda p.\, p\ y')) \\
\equiv\ & \lambda s.\, (\lambda q.\, \lambda y'.\, q\ (\lambda p.\, p\ y'))\ (\lambda o.\, s\ (\lambda x.\, o\ (\lambda y.\, \textsf{love}\ x\ y))) \\
\equiv\ & \lambda s.\, \lambda y'.\, (\lambda o.\, s\ (\lambda x.\, o\ (\lambda y.\, \textsf{love}\ x\ y)))\ (\lambda p.\, p\ y') \\
\equiv\ & \lambda s.\, \lambda y'.\, s\ (\lambda x.\, (\lambda p.\, p\ y')\ (\lambda y.\, \textsf{love}\ x\ y)) \\
\equiv\ & \lambda s.\, \lambda y'.\, s\ (\lambda x.\, (\lambda y.\, \textsf{love}\ x\ y)\ y') \\
\equiv\ & \lambda s.\, \lambda y'.\, s\ (\lambda x.\, \textsf{love}\ x\ y')
\end{aligned}
$$

$$
\begin{aligned}
&\llbracket \textit{Zhāngsān} \rrbracket\ (\llbracket \textit{ài} \rrbracket\ \llbracket \textit{shéi} \rrbracket) \\
\equiv\ & (\lambda q.\, q\ (\lambda p.\, p\ \textsf{Z}_3))\ (\lambda s.\, \lambda y'.\, s\ (\lambda x.\, \textsf{love}\ x\ y')) \\
\equiv\ & (\lambda s.\, \lambda y'.\, s\ (\lambda x.\, \textsf{love}\ x\ y'))\ (\lambda p.\, p\ \textsf{Z}_3) \\
\equiv\ & \lambda y'.\, (\lambda p.\, p\ \textsf{Z}_3)\ (\lambda x.\, \textsf{love}\ x\ y') \\
\equiv\ & \lambda y'.\, (\lambda x.\, \textsf{love}\ x\ y')\ \textsf{Z}_3 \\
\equiv\ & \lambda y'.\, \textsf{love}\ \textsf{Z}_3\ y'
\end{aligned}
$$

$$
\begin{aligned}
&\llbracket \textit{shéi} \rrbracket\ (\llbracket \textit{ài} \rrbracket\ \llbracket \textit{shéi} \rrbracket) \\
\equiv\ & (\lambda q.\, \lambda x'.\, q\ (\lambda p.\, p\ x'))\ (\lambda s.\, \lambda y'.\, s\ (\lambda x.\, \textsf{love}\ x\ y')) \\
\equiv\ & \lambda x'.\, (\lambda s.\, \lambda y'.\, s\ (\lambda x.\, \textsf{love}\ x\ y'))\ (\lambda p.\, p\ x') \\
\equiv\ & \lambda x'.\, \lambda y'.\, \textsf{love}\ x'\ y'
\end{aligned}
$$

sketch of a doomed analysis of in-situ wh

Suppose we had tried to use a variant of Montague's type for NPs. Then the periscoping of the *wh*-variables wouldn't be straightforwardly possible:

$$
\begin{aligned}
& [\![\textit{'love'}]\!]\ [\![\textit{'who'}]\!] \\
\equiv\ & (\lambda o.\, \lambda x.\, o\ (\lambda y.\, \text{love}\ x\ y))\ (\lambda p.\, \lambda y'.\, p\ y') \\
\equiv\ & \lambda x.\, (\lambda p.\, \lambda y'.\, p\ y')\ (\lambda y.\, \text{love}\ x\ y) \\
\equiv\ & \lambda x.\, \lambda y'.\, (\lambda y.\, \text{love}\ x\ y)\ y' \\
\equiv\ & \lambda x.\, \lambda y'.\, \text{love}\ x\ y'
\end{aligned}
$$

$$
\begin{aligned}
& [\![\textit{'everybody'}]\!]\ ([\![\textit{'love'}]\!]\ [\![\textit{'who'}]\!]) \\
\equiv\ & (\lambda p.\, \forall x (\text{person}\ x) \rightarrow (p\ x))\ (\lambda x.\, \lambda y'.\, \text{love}\ x\ y') \\
\equiv\ & \forall x (\text{person}\ x) \rightarrow ((\lambda x.\, \lambda y'.\, \text{love}\ x\ y')\ x) \\
\equiv\ & \forall x (\text{person}\ x) \rightarrow (\lambda y'.\, \text{love}\ x\ y')
\end{aligned}
$$

embedded constituent questions 1

(9)

張三	想知道	李四	愛	誰。
Zhāngsān	xiǎng zhīdào	Lǐsì	ài	shéi.
Zhangsan	wonders	Lisi	loves	who

'Zhangsan wonders who Lisi loves.'

(10)

張三	認為	李四	愛	誰？
Zhāngsān	rènwéi	Lǐsì	ài	shéi?
Zhangsan	thinks	Lisi	loves	who

'Who does Zhangsan think Lisi loves?'

(11)

張三	知道	李四	愛	誰。
Zhāngsān	zhīdào	Lǐsì	ài	shéi.
Zhangsan	knows	Lisi	loves	who

'Zhangsan knows who Lisi loves.'

張三	知道	李四	愛	誰？
Zhāngsān	zhīdào	Lǐsì	ài	shéi?
Zhangsan	knows	Lisi	loves	who

'Who does Zhangsan know Lisi loves?'

(12)

張三	想知道	李四	認為	王五	愛	誰。
Z3	xiǎng zhīdào	Lǐsì	rènwéi	Wángwǔ	ài	shéi.
Z3	wonders	Lisi	thinks	Wangwu	loves	who

'Zhangsan wonders who Lisi thinks Wangwu loves.'

embedded constituent questions 3

(13)	張三	想知道	誰	把誰	介紹	給誰？
	Zhāngsān	xiǎng zhīdào	shéi	bǎ shéi	jièshào	gěi shéi?
	Zhangsan	wonders	who	whom	introduced	to who

If construed as an appropriate kind of question, this sentence can be answered by any one of the following six sentences.

(14)	張三	想知道	李四	把誰	介紹	給誰。
	Zhāngsān	xiǎng zhīdào	Lǐsì	bǎ shéi	jièshào	gěi shéi.
	Zhangsan	wonders	Lisi	whom	introduced	to who

‘Zhangsan wonders who Lisi introduced to who.’

(15)	張三	想知道	誰	把李四	介紹	給誰。
	Zhāngsān	xiǎng zhīdào	shéi	bǎ Lǐsì	jièshào	gěi shéi.
	Zhangsan	wonders	who	Lisi	introduced	to who

‘Zhangsan wonders who introduced Lisi to who.’

(16)

張三	想知道	誰	把誰	介紹	給李四。
Zhāngsān	xiǎng zhīdào	shéi	bǎ shéi	jièshào	gěi shéi.
Zhangsan	wonders	who	whom	introduced	to Lisi

'Zhangsan wonders who introduced who to Lisi.'

(17)

張三	想知道	誰	把李四	介紹	給王五。
Zhāngsān	xiǎng zhīdào	shéi	bǎ Lǐsì	jièshào	gěi Wángwǔ.
Zhangsan	wonders	who	Lisi	introduced	to Wangwu

'Zhangsan wonders who introduced Lisi to Wangwu.'

(18)

張三	想知道	李四	把誰	介紹	給王五。
Zhāngsān	xiǎng zhīdào	Lǐsì	bǎ shéi	jièshào	gěi Wángwǔ.
Zhangsan	wonders	Lisi	whom	introduced	to Wangwu

'Zhangsan wonders who Lisi introduced to Wangwu.'

(19)

張三	想知道	李四	把王五	介紹	給誰。
Zhāngsān	xiǎng zhīdào	Lǐsì	bǎ Wángwǔ	jièshào	gěi shéi.
Zhangsan	wonders	Lisi	Wangwu	introduced	to who

'Zhangsan wonders who Lisi introduced Wangwu to.'

sketch of the lexical analysis of wh scope 1

An exportation relation, construed as a parameterized function, ensures the non-deterministic percolation of *wh*-information past an embedder.

export $l\ q\ p :=$ xprt $l\ (\lambda r.\, q\ (r\ p))$

$$
\begin{array}{lll}
\text{xprt } [\,] & := \lambda q. & q\ (\lambda p. \quad\quad p) \\
\text{xprt } [\text{True}\,|l] & := \lambda q.\, \text{xprt } l\ (\lambda r.\, \lambda x. & q\ (\lambda p. \quad\quad r\ (p\ x))) \\
\text{xprt } [\text{False}|l] & := \lambda q.\, \text{xprt } l\ (\lambda r. & q\ (\lambda p.\, \lambda x.\ r\ (p\ x)))
\end{array}
$$

The export function combines with an n-element list of Booleans and two more arguments, the last of which it supplies with n variables. Those variables corresponding to False on the list of Booleans are abstracted over immediately, and the result is fed to the second argument of the export function. After that the variables corresponding to True on the list of Booleans are abstracted over.

$$
\begin{array}{ll}
 & \text{export [False, True]} \quad \mathit{func} \quad \mathit{arg} \\
\equiv & \qquad \lambda y.\, \mathit{func}\ (\lambda x.\, \mathit{arg}\ x\ y)
\end{array}
$$

In the simplest case, exporting nothing just amounts to function application:

$$
\begin{array}{ll}
 & \text{export } [\,]\ \mathit{func}\ \mathit{arg} \\
\equiv & \qquad \mathit{func}\ \mathit{arg}
\end{array}
$$

sketch of the lexical analysis of wh scope 2

$$
\begin{aligned}
&\text{xprt } [\,] \\
\equiv\ & \lambda q''.\, q''\ (\lambda p''.\, p'')
\end{aligned}
$$

$$
\begin{aligned}
&\text{xprt } [\text{True}] \\
\equiv\ & \lambda q'.\, \text{xprt } [\,]\ (\lambda r'.\, \lambda y.\, q'\ (\lambda p'.\, r'\ (p'\ y))) \\
\equiv\ & \lambda q'.\, (\lambda q''.\, q''\ (\lambda p''.\, p''))\ (\lambda r'.\, \lambda y.\, q'\ (\lambda p'.\, r'\ (p'\ y))) \\
\equiv\ & \lambda q'.\, (\lambda r'.\, \lambda y.\, q'\ (\lambda p'.\, r'\ (p'\ y)))\ (\lambda p''.\, p'') \\
\equiv\ & \lambda q'.\, \lambda y.\, q'\ (\lambda p'.\, (\lambda p''.\, p'')\ (p'\ y)) \\
\equiv\ & \lambda q'.\, \lambda y.\, q'\ (\lambda p'.\, p'\ y)
\end{aligned}
$$

$$
\begin{aligned}
&\text{xprt } [\text{False, True}] \\
\equiv\ & \lambda q.\, \text{xprt } [\text{True}]\ (\lambda r.\, q\ (\lambda p.\, \lambda x.\, r\ (p\ x))) \\
\equiv\ & \lambda q.\, (\lambda q'.\, \lambda y.\, q'\ (\lambda p'.\, p'\ y))\ (\lambda r.\, q\ (\lambda p.\, \lambda x.\, r\ (p\ x))) \\
\equiv\ & \lambda q.\, \lambda y.\, (\lambda r.\, q\ (\lambda p.\, \lambda x.\, r\ (p\ x)))\ (\lambda p'.\, p'\ y) \\
\equiv\ & \lambda q.\, \lambda y.\, q\ (\lambda p.\, \lambda x.\, (\lambda p'.\, p'\ y)\ (p\ x)) \\
\equiv\ & \lambda q.\, \lambda y.\, q\ (\lambda p.\, \lambda x.\, p\ x\ y)
\end{aligned}
$$

$$
\begin{aligned}
&\text{export } [\text{False, True}]\ \mathit{func}\ \mathit{arg} \\
\equiv\ & \text{xprt } [\text{False, True}]\ (\lambda r.\, \mathit{func}\ (r\ \mathit{arg})) \\
\equiv\ & (\lambda q.\, \lambda y.\, q\ (\lambda p.\, \lambda x.\, p\ x\ y))\ (\lambda r.\, \mathit{func}\ (r\ \mathit{arg})) \\
\equiv\ & \lambda y.\, (\lambda r.\, \mathit{func}\ (r\ \mathit{arg}))\ (\lambda p.\, \lambda x.\, p\ x\ y) \\
\equiv\ & \lambda y.\, \mathit{func}\ ((\lambda p.\, \lambda x.\, p\ x\ y)\ \mathit{arg}) \\
\equiv\ & \lambda y.\, \mathit{func}\ (\lambda x.\, \mathit{arg}\ x\ y)
\end{aligned}
$$

sketch of the lexical analysis of wh scope 3

Sentence embedding verbs are predicates combining with a clause (denoting a curried n-place relation) and a subject generalized quantifier.

$[\![$*rènwéi*$]\!] := \lambda c.\, \lambda s.\, s\ (\lambda x.\, \text{export}\ l\ (\text{think}\ x)\ c)$

think :: Ent $\rightarrow$ Bool $\rightarrow$ Bool

The translation of a sentence embedding verb such as *rènwéi* 'to think' contains an open variable l, the rationale for which is that any instantiation of l with a list of Booleans that leads to a typeable expression corresponds to a possible scope assignment. The conceivable instantiations are constrained by the type of the core meaning of the predicate, in this case the type of the constant 'think', whose second argument must be a Boolean (approximating a proposition, since I'm ignoring intensionality for the moment). Thus all *wh*-information must be exported from the clause with which *rènwéi* combines.

Question embedding verbs such as *xiǎng zhīdào* have a core meaning that must combine with a question. In order to formalize that it's necessary to first think about the denotation of (potentially interrogative) clauses.

non-interrogative uses of wh-NPs

(20)

我	希望	誰	能	來	陪陪	我。
Wǒ	xīwàng	shéi	néng	lái	péipéi	wǒ.
I	wish	who	can	come	accompany	me

‘I wish somebody could come to accompany me.’

(21)

我	沒	買	甚麼。
Wǒ	méi	mǎi	shénme.
I	didn’t	buy	what

‘I didn’t buy anything.’

(22)

誰	都	可以	來。
Shéi	dōu	kěyǐ	lái.
who	even	may	come

‘Everybody can come.’

(23)

你	喜歡	甚麼，	就	買	甚麼。
Nǐ	xǐhuān	shénme,	jiù	mǎi	shénme.
you	like	what	then	buy	what

‘If you like something, just buy it.’

(24) ?

誰	喜歡	甚麼，	誰	就	買	甚麼。
Shéi	xǐhuān	shénme,	shéi	jiù	mǎi	shénme.
who	like	what	who	then	buy	what

‘Whoever likes something should buy it.’

sketch of the analysis of non-interrogative wh

- donkey binding corresponds to generalized entailment in the sense of [GS89]
- existential (universal) use modeled by existential (universal) closure
- exhaustive question reading(s) modeled by exhaustivity operator(s)

Recall that the quantifier symbols are definable in type theory and can be introduced as syntactic sugar:

$$\forall x \phi := \lambda x.\, \phi = \lambda x.\, \top$$
$$\exists x \phi := \lambda x.\, \phi \neq \lambda x.\, \bot$$

These operators need to be generalized so that they can take arbitrary (curried) relations as their arguments:

$$\forall\ (p :: \quad a \rightarrow \text{Bool}) := p = \quad \lambda x :: a.\, \top$$
$$\forall\ (p :: b \rightarrow a \rightarrow \text{Bool}) := p = \lambda y :: b.\, \lambda x :: a.\, \top$$

etc.

The set of admissible types of the input to these closure operators is precisely the set of conjoinable types in the sense of Partee and Rooth [PR83]. The conjoinable types also constitute the types of possible sentential meanings.

generalized connectives and type overloading 1

Partee and Rooth's definition of conjoinable types [PR83, p. 363]

(i) t is a conjoinable type
(ii) if b is a conjoinable type, then
for all a, $\langle a, b\rangle$ is a conjoinable type

is formalized type-theoretically by using predication over types or *qualified types* in the sense of [Jon92b]

$$\emptyset \Vdash \mathsf{Conj}\ \mathsf{t}$$
$$\mathsf{Conj}\ b \Vdash \mathsf{Conj}\ \forall a.(a \longrightarrow b)$$

or the more appealing notation of Haskell[1] [PH$^+$97]:

class Conj a
instance Conj t
instance $(\mathsf{Conj}\ b) \Rightarrow \mathsf{Conj}\ (a \longrightarrow b)$

[1] A functional language spoken in Nottingham, Yale, and other places.

generalized connectives and type overloading 2

Partee and Rooth's definition of generalized conjunction and disjunction [PR83, p. 364] closely resembles that of [Gaz80]:

$$
\begin{aligned}
X \sqcap Y &= X \wedge Y \text{ if } X \text{ and } Y \text{ are truth values} \\
&= \{\langle z, x \sqcap y\rangle \mid \langle z, x\rangle \in X \text{ and } \langle z, y\rangle \in Y\} \\
&\quad \text{if } X \text{ and } Y \text{ are functions} \\
&\quad \text{(which are represented as sets of ordered pairs)} \\
X \sqcup Y &= X \vee Y \text{ if } X \text{ and } Y \text{ are truth values} \\
&= \{\langle z, x \sqcup y\rangle \mid \langle z, x\rangle \in X \text{ and } \langle z, y\rangle \in Y\} \\
&\quad \text{if } X \text{ and } Y \text{ are functions.}
\end{aligned}
$$

My formalization transfers these notions into the language of Hindley/Milner type theory with type classes [Jon95]:

$$
\begin{aligned}
&\textbf{class } \text{Conj } a \textbf{ where} \\
&\quad \sqcap, \sqcup :: a \to a \to a \\
&\textbf{instance } \text{Conj t} \textbf{ where} \\
&\quad x \sqcap y := x \wedge y \\
&\quad x \sqcup y := x \vee y \\
&\textbf{instance } (\text{Conj } b) \Rightarrow \text{Conj } (a \to b) \textbf{ where} \\
&\quad x \sqcap y := \lambda z.\, (x\ z) \sqcap (y\ z) \\
&\quad x \sqcup y := \lambda z.\, (x\ z) \sqcup (y\ z)
\end{aligned}
$$

Now $\sqcap$ has a unique most general polymorphic type:

$$\sqcap :: (\text{Conj } a) \Rightarrow a \to a \to a$$

lattice types for natural language 1

The instances of the type class SemiLattice define the e/t–conjoinable types in the sense of [PR83] with the class method $\sqcap$ corresponding to generalized conjunction.

class SemiLattice a **where**
$\sqcap :: a \rightarrow a \rightarrow a$
$\sqsubseteq :: a \rightarrow a \rightarrow \mathsf{Bool}$

$x \sqsubseteq y := (x \sqcap y) = x$

data Bool := False | True

data Ent := ZL_7 | Z_3 | L_4

instance SemiLattice Bool **where**
$\mathsf{True} \sqcap x := x$
$\mathsf{False} \sqcap x := \mathsf{False}$

instance SemiLattice Ent **where**
$x \sqcap x := x$
$\mathsf{Z}_3 \sqcap \mathsf{L}_4 := \mathsf{ZL}_7$
$\mathsf{L}_4 \sqcap \mathsf{Z}_3 := \mathsf{ZL}_7$

instance (SemiLattice a) $\Rightarrow$ SemiLattice $(b \rightarrow a)$ **where**
$x \sqcap y := \lambda z.\, (x\ z) \sqcap (y\ z)$

lattice types for natural language 2

Generalizations of the usual Boolean connectives are obtained as methods in a more restrictive type class:

class (SemiLattice a) $\Rightarrow$ BooleanLattice a **where**

$$\begin{array}{ll} \top, \bot & :: a \\ \neg & :: a \to a \\ \sqcup & :: a \to a \to a \\ \forall, \exists & :: a \to \mathsf{Bool} \end{array}$$

$$\begin{array}{ll} \bot & := \neg\top \\ x \sqcup y & := \neg(\neg x \sqcap \neg y) \\ \forall x & := x = \top \\ \exists x & := x \neq \bot \end{array}$$

instance BooleanLattice Bool **where**

$\top :=$ True

$\neg$True $:=$ False

$\neg$False $:=$ True

instance (BooleanLattice a) $\Rightarrow$ BooleanLattice $(b \to a)$ **where**

$$\begin{array}{ll} \top & := \lambda y.\, \top \\ \neg x & := \lambda y.\, \neg(x\ y) \end{array}$$

Computational applications of classes of lattice types are discussed in [Jon92a, Jon95].

lattice types for natural language 3

Most of the generalized connectives defined so far have appeared in the literature in various forms:

$\sqcap$	generalized conjunction	[Gaz80, PR83]
$\sqcup$	generalized disjunction	[Gaz80, PR83]
$\sqsubseteq$	generalized entailment	[GS89]
$\perp$	**zero**	[GS82, p. 202]
$\exists$	existential closure	various
$\forall$	universal closure	various

A donkey sentence:

$$(\lambda x.\, \lambda y.\, (\text{farmer } x) \sqcap (\text{own } x\ y) \sqcap (\text{donkey } y)) \sqsubseteq \text{beat}$$

generalized predicates based on equality 1

$=, \neq :: a \to a \to \mathsf{Bool}$

$\sqsubseteq :: (\mathsf{SemiLattice}\ a) \Rightarrow a \to a \to \mathsf{Bool}$
$x \sqsubseteq y := (x \sqcap y) = x$

$\forall :: (\mathsf{BooleanLattice}\ a) \Rightarrow a \to \mathsf{Bool}$
$\forall x := x = \top$

$\exists :: (\mathsf{BooleanLattice}\ a) \Rightarrow a \to \mathsf{Bool}$
$\exists x := x \neq \bot$

$¿ :: (a \to b) \to a \to a \to \mathsf{Bool}$
$¿x := \lambda u.\, \lambda w.\, (x\ u) = (x\ w)$

The last three predicates are closure operators in the sense that

$$\begin{array}{rl} \forall(\forall x) \equiv \forall x & \quad x \sqsubseteq y \text{ entails } (\forall x) \sqsubseteq (\forall y) \\ \exists(\exists x) \equiv \exists x & \quad x \sqsubseteq y \text{ entails } (\exists x) \sqsubseteq (\exists y) \\ ¿(¿x) \equiv ¿x & \end{array}$$

Note that ¿ is often called a “question operator” despite the fact that it really only is a strong exhaustivity operator.

generalized predicates based on equality 2

$$
\begin{aligned}
&\forall(\forall x :: a)\\
\equiv\ &\forall(x :: a = \top :: a)\\
\equiv\ &((x :: a = \top :: a) :: \mathsf{Bool} = \top :: \mathsf{Bool}) :: \mathsf{Bool}\\
\equiv\ &(x :: a = \top :: a) :: \mathsf{Bool}\\
\equiv\ &\forall(x :: a)
\end{aligned}
$$

Similarly for $\exists$.

$$
\begin{aligned}
&¿(¿x)\\
\equiv\ &¿(\lambda u'.\, \lambda w'.\, (x\ u') = (x\ w'))\\
\equiv\ &\lambda u.\, \lambda w.\, ((\lambda u'.\, \lambda w'.\, (x\ u') = (x\ w'))\ u) = ((\lambda u'.\, \lambda w'.\, (x\ u') = (x\ w'))\ w)\\
\equiv\ &\lambda u.\, \lambda w.\, (\lambda w'.\, (x\ u) = (x\ w')) = (\lambda w'.\, (x\ w) = (x\ w'))\\
\equiv\ &\lambda u.\, \lambda w.\, (x\ u) = (x\ w)\\
\equiv\ &¿x
\end{aligned}
$$

putting it all together

$$
\begin{aligned}
&\textbf{data}\ \mathsf{Bool} &&:= \mathsf{False} \mid \mathsf{True} \\
&\textbf{data}\ \mathsf{Ent} &&:= \mathsf{Z}_3 \mid \mathsf{L}_4 \mid \mathsf{W}_5 \mid \ldots \\
&\textbf{data}\ \mathsf{Ind} &&:= \mathsf{Idx}_0 \mid \mathsf{Idx}_1 \mid \ldots
\end{aligned}
$$

$$
\begin{aligned}
[\![\textit{Zhāngsān}]\!] &:: \ (\mathsf{BooleanLattice}\ a) \Rightarrow \\
&\qquad ((\mathsf{Ind} \to (\mathsf{Ind} \to \mathsf{Ent} \to \mathsf{Bool}) \to \mathsf{Bool}) \to a) \to a \\
[\![\textit{Zhāngsān}]\!] &:= \lambda q.\, q\ (\lambda w.\, \lambda p.\, p\ w\ \mathsf{Z}_3)
\end{aligned}
$$

$$
\begin{aligned}
[\![\textit{shéi}]\!] &:: \ (\mathsf{BooleanLattice}\ a) \Rightarrow \\
&\qquad ((\mathsf{Ind} \to (\mathsf{Ind} \to \mathsf{Ent} \to \mathsf{Bool}) \to \mathsf{Bool}) \to a) \to \mathsf{Ent} \to a \\
[\![\textit{shéi}]\!] &:= \lambda q.\, \lambda x.\, q\ (\lambda w.\, \lambda p.\, (\mathsf{person} \sqcap p)\ w\ x) \\
\mathsf{person} &:: \mathsf{Ind} \to \mathsf{Ent} \to \mathsf{Bool}
\end{aligned}
$$

All NP types are instances of

$$
\begin{aligned}
&(\mathsf{BooleanLattice}\ a, \mathsf{BooleanLattice}\ b) \Rightarrow \\
&\quad ((\mathsf{Ind} \to (\mathsf{Ind} \to \mathsf{Ent} \to \mathsf{Bool}) \to \mathsf{Bool}) \to a) \to b
\end{aligned}
$$

Schema for intransitive verbs:

$$
\begin{aligned}
[\![\mathsf{iv}]\!] ::\ & (\mathsf{BooleanLattice}\ a) \Rightarrow \\
& (((\mathsf{Ind} \to (\mathsf{Ind} \to \mathsf{Ent} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Bool}) \to a) \to \mathsf{Ind} \to a
\end{aligned}
$$

$$[\![\mathsf{iv}]\!] := \lambda s'.\, \lambda w.\, s'\ (\lambda s.\, s\ w\ (\lambda w.\, \lambda x.\, iv\ w\ x))$$

$$iv :: \mathsf{Ind} \to \mathsf{Ent} \to \mathsf{Bool}$$

Schema for transitive verbs:

$$
\begin{aligned}
[\![\mathsf{tv}]\!] ::\ & (\mathsf{BooleanLattice}\ a, \mathsf{BooleanLattice}\ b) \Rightarrow \\
& (((\mathsf{Ind} \to (\mathsf{Ind} \to \mathsf{Ent} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Bool}) \to a) \\
& \to (((\mathsf{Ind} \to (\mathsf{Ind} \to \mathsf{Ent} \to \mathsf{Bool}) \to \mathsf{Bool}) \to a) \to b) \to \mathsf{Ind} \to b
\end{aligned}
$$

$$[\![\mathsf{tv}]\!] := \lambda o'.\, \lambda s'.\, \lambda w.\, s'\ (\lambda s.\, o'(\lambda o.\, s\ w\ (\lambda w.\, \lambda x.\, o\ w\ (\lambda w.\, \lambda y.\, tv\ w\ x\ y))))$$

$$tv :: \mathsf{Ind} \to \mathsf{Ent} \to \mathsf{Ent} \to \mathsf{Bool}$$

Schema for sentence embedding verbs:

$$
\begin{aligned}
[\![\mathsf{sev}]\!] ::\ & (\mathsf{BooleanLattice}\ a, \mathsf{BooleanLattice}\ b) \Rightarrow (\mathsf{Ind} \to b) \to \\
& (((\mathsf{Ind} \to (\mathsf{Ind} \to \mathsf{Ent} \to \mathsf{Bool}) \to \mathsf{Bool}) \to \mathsf{Bool}) \to a) \to \mathsf{Ind} \to a
\end{aligned}
$$

$$[\![\mathsf{sev}]\!] := \lambda c.\, \lambda s'.\, \lambda w.\, s'\ (\lambda s.\, s\ w\ (\lambda w.\, \lambda x.\, \mathsf{export}\ [\mathsf{False}|l]\ (sev\ w\ x)\ c))$$

sentembver $:= \lambda v.\, \lambda c.\, \lambda s'.\, \lambda w.\, s'\ (\lambda s.\, s\ w\ (\lambda w.\, \lambda x.\, \text{export}\ [\text{False}|l]\ (v\ w\ x)\ c))$

Lexical entry for *rènwéi* 'think':

$[\![$*rènwéi*$]\!]$:= sentembver think
think :: Ind $\rightarrow$ Ent $\rightarrow$ (Ind $\rightarrow$ Bool) $\rightarrow$ Bool

Lexical entry for *xiǎng zhīdào* 'wonder':

$[\![$*xiǎng zhīdào*$]\!]$:= sentembver wonder
wonder :: (BooleanLattice b) $\Rightarrow$ Ind $\rightarrow$ Ent $\rightarrow$ (Ind $\rightarrow a \rightarrow b$) $\rightarrow$ Bool

Lexical entry for *xīwàng* 'wish':

$[\![$*xīwàng*$]\!]$:= sentembver $(\lambda w.\, \lambda x.\, \lambda p.\, \text{wish}\ w\ x\ (\lambda w'.\, \exists\ (p\ w')))$
wish :: Ind $\rightarrow$ Ent $\rightarrow$ (Ind $\rightarrow$ Bool) $\rightarrow$ Bool

polar questions

(25)

張三	知道	李四	會	來。
Zhāngsān	zhīdào	Lǐsì	huì	lái.
Zhangsan	knows	Lisi	will	come

‘Zhangsan knows that Lisi will come.’

(26)

張三	知道	李四	會	來	嗎？
Zhāngsān	zhīdào	Lǐsì	huì	lái	ma?
Zhangsan	knows	Lisi	will	come	Q

‘Does Zhangsan know that Lisi will come?’

(27)

張三	知不知道	李四	會	來？
Zhāngsān	zhī-bu-zhīdào	Lǐsì	huì	lái?
Zhangsan	knows-not-knows	Lisi	will	come

‘Does Zhangsan know that Lisi will come?’

(28)

張三	知道	李四	會不會	來。
Zhāngsān	zhīdào	Lǐsì	huì-bu-huì	lái.
Zhangsan	knows	Lisi	will-not-will	come

‘Zhangsan knows whether Lisi will come.’
(ambiguous?)

(29)

張三	認爲	李四	會不會	來？
Zhāngsān	rènwéi	Lǐsì	huì-bu-huì	lái?
Zhangsan	thinks	Lisi	will-not-will	come

‘Does Zhangsan think Lisi will come?’

sketch of the analysis of polar questions

kind of interrogative	extension	type	uncurried type
one constituent	unary relation	$a \longrightarrow \mathsf{Bool}$	$(a) \longrightarrow \mathsf{Bool}$
two constituents	binary relation	$b \longrightarrow a \longrightarrow \mathsf{Bool}$	$(b, a) \longrightarrow \mathsf{Bool}$
three constituents	ternary relation	$c \longrightarrow b \longrightarrow a \longrightarrow \mathsf{Bool}$	$(c, b, a) \longrightarrow \mathsf{Bool}$
polar	nullary relation		$() \longrightarrow \mathsf{Bool}$

Let polar interrogatives have denotations of type $\mathsf{Ind} \longrightarrow 1 \longrightarrow \mathsf{Bool}$. Using vacuous abstraction over a variable of the unit type 1 enables a uniform analysis of the meaning of constituent interrogatives and polar interrogatives.

concluding thoughts

1. A lexical account of existential uses of *wh*-elements (EPWs) is compatible with Lin's [Lin96, Lin98] descriptive generalizations about the environments in which EPWs are licensed. Provided these generalizations were correct, they would translate directly into a description of the distribution of existential closure operators. This has the advantage that the problems of Lin's account concerning locality conditions will disappear.
2. Certain phenomena might require a more fully specified marking of the scope of an individual *wh*-element in the lexicon. This should be worked into the mechanisms for scope exportation.
3. Using qualified types and constructor classes might prove useful for a very abstract approach to dynamic semantics.

acknowledgments

I've been thinking about the topic of this presentation for quite some time now, and have benefited at various stages from discussions with and comments from Sigrid Beck, David Dowty, Erhard Hinrichs, Pauline Jacobson, Bob Kasper, Barbara Partee, Carl Pollard, Craige Roberts, Mariko Sugahara, Nathan Vaillette, Shravan Vasishth, and Neal Whitman. I'd furthermore like to thank Janice Fon 馮怡蓁, Huang Tsan 黃璨, Liu Zhiyong 劉智勇, and Wang Jinyi 王錦益 for sharing with me their native speaker intuitions. The usual disclaimers apply.

references

[Dek99] Paul Dekker. Scopes in discourse. *Journal of Language and Computation*. To appear 1999.

[Gaz80] Gerald Gazdar. A cross-categorial semantics for coordination. *Linguistics and Philosophy* 3: 407–409, 1980.

[GS82] Jeroen Groenendijk and Martin Stokhof. Semantic analysis of *Wh*-complements. *Linguistics and Philosophy* 5(2): 175–233, 1982.

[GS89] Jeroen Groenendijk and Martin Stokhof. Type shifting rules and the semantics of interrogatives. In *Properties, Types, and Meanings*, edited by Gennaro Chierchia, Barbara H. Partee, and Ray Turner, vol. 2 of *Studies in Linguistics and Philosophy*, pp. 21–68. Kluwer, Dordrecht, 1989.

[GS97] Jeroen Groenendijk and Martin Stokhof. Questions. In *Handbook of Logic and Language*, edited by Johan van Benthem and Alice ter Meulen, chap. 19. Elsevier, Amsterdam, 1997.

[Hau84] Roland R. Hausser. *Surface Compositional Grammar*, vol. 4 of *Studies in Theoretical Linguistics*. Fink, Munich, 1984.

[Jon92a] Mark P. Jones. Computing with lattices: An application of type classes. *Journal of Functional Programming* 2(4): 475–503, 1992.

[Jon92b] Mark P. Jones. A theory of qualified types. In *ESOP '92: European Symposium on Programming, Rennes, France*, no. 582 in Lecture Notes in Computer Science. Springer, Berlin and New York, 1992.

[Jon95] Mark P. Jones. Functional programming with overloading and higher-order polymorphism. In *Advanced Functional Programming, First International Spring School on Advanced Functional Programming Techniques, Bastad, Sweden*, edited by J. Jeuring and E. Meijer, no. 925 in Lecture Notes in Computer Science, pp. 97–136. Springer, Berlin and New York, 1995.

[Lin96] Jo-wang Lin. *Polarity Licensing and* Wh-*Phrase Quantification in Chinese*. Ph.D. thesis, University of Massachusetts, Amherst, 1996.

[Lin98] Jo-wang Lin. On existential polarity *Wh*-phrases in Chinese. *Journal of East Asian Linguistics* 7(3): 219–255, 1998.

[Mon73] Richard Montague. The proper treatment of quantification in ordinary English. In *Approaches to Natural Language*, edited by Jaako Hintikka, Julius Moravcsik, and Patrick Suppes. Reidel, Dordrecht, 1973.

[PH+97] John Peterson, Kevin Hammond (eds.) et al. *Report on the Programming Language Haskell: A Non-strict, Purely Functional Language, Version 1.4*. Technical report, Yale University and others, April 7, 1997.

[PR83] Barbara Partee and Mats Rooth. Generalized conjunction and type ambiguity. In *Meaning, Use, and Interpretation of Language*, edited by Rainer Bäuerle, Christian Schwarze, and Arnim von Stechow, pp. 361–383. de Gruyter, Berlin, 1983.