“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 评 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.

a uniform semantics for Mandarin *wh*-NPs
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`
- `Ent := Z₃ | L₄ | W₅ | . . .`

- `[[ Zhāngsān ]] := λq. q (λp. p Z₃)`
- `[[ Lǐsì ]] := λq. q (λp. p L₄)`
- `[[ shéi ]] := λq. λx. q (λp. p x)`

- `[[ ài ]] := λo’. λs. o’ (λo. s (λx. o (λy. love x y)))`
- `love :: Ent → Ent → Bool`
Ài shéi
≡ λo'. λs. o' (λo. s (λx. o (λy. love x y))) (λq. λy'. q (λp. p y'))
≡ λs. (λq. λy'. q (λp. p y')) (λo. s (λx. o (λy. love x y)))
≡ λs. λy'. (λo. s (λx. o (λy. love x y))) (λp. p y')
≡ λs. λy'. s (λx. (λp. p y') (λy. love x y))
≡ λs. λy'. s (λx. (λy. love x y) y')
≡ λs. λy'. s (λx. love x y')

Zhāngsān ([Ài] [shéi])
≡ (λq. q (λp. p Z₃)) (λs. λy'. s (λx. love x y'))
≡ (λs. λy'. s (λx. love x y')) (λp. p Z₃)
≡ λy'. (λp. p Z₃) (λx. love x y')
≡ λy'. (λx. love x y') Z₃
≡ λy'. love Z₃ y'

shéi ([Ài] [shéi])
≡ (λq. λx'. q (λp. p x')) (λs. λy'. s (λx. love x y'))
≡ λx'. (λs. λy'. s (λx. love x y')) (λp. p x')
≡ λx'. λy'. love x' y'

a uniform semantics for Mandarin wh-NPs
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 \textit{wh}-variables wouldn’t be straightforwardly possible:

\[
\llbracket ‘love’\rrbracket \llbracket ‘who’\rrbracket \\
\equiv (\lambda o. \lambda x. o (\lambda y. \text{love} \, x \, y)) (\lambda p. \lambda y’. p \, y’)
\equiv \lambda x. (\lambda p. \lambda y’. p \, y’) (\lambda y. \text{love} \, x \, y)
\equiv \lambda x. \lambda y’. (\lambda y. \text{love} \, x \, y) \, y’
\equiv \lambda x. \lambda y’. \text{love} \, x \, y’
\]

\[
\llbracket ‘everybody’\rrbracket (\llbracket ‘love’\rrbracket \llbracket ‘who’\rrbracket)
\equiv (\lambda p. \forall x (\text{person} \, x) \rightarrow (p \, x)) (\lambda x. \lambda y’. \text{love} \, x \, y’)
\equiv \forall x (\text{person} \, x) \rightarrow ((\lambda x. \lambda y’. \text{love} \, x \, y’) \, x)
\equiv \forall x (\text{person} \, x) \rightarrow (\lambda y’. \text{love} \, x \, y’)
\]

a uniform semantics for Mandarin \textit{wh}-NPs
embedded constituent questions 1

(9) 張三 想知道 李四 愛 誰。
Zhāngsān xiǎng zhīdào Lísi ài shéi.
Zhangsan wonders Lisi loves who
‘Zhangsan wonders who Lisi loves.’

(10) 張三 認為 李四 愛 誰？
Zhāngsān rènwéi Lísi ài shéi?
Zhangsan thinks Lisi loves who
‘Who does Zhangsan think Lisi loves?’

(11) 張三 知道 李四 愛 誰。
Zhāngsān zhīdào Lísi ài shéi.
Zhangsan knows Lisi loves who
‘Zhangsan knows who Lisi loves.’

張三 知道 李四 愛 誰？
Zhāngsān zhīdào Lísi 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 Lísì rènwéi Wángwǔ ài shéi.
Z3 wonders Lisi thinks Wangwu loves who
‘Zhangsan wonders who Lisi thinks Wangwu loves.’
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 Lìsì 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 Lìsì 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 Lìsì bǎ Wángwǔ jièshào gěi shéi.
Zhangsan wonders Lisi Wangwu introduced to who
‘Zhangsan wonders who Lisi introduced Wangwu to.’

a uniform semantics for Mandarin wh-NPs
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.

\[
\text{export } l \ q \ p := \text{xprt } l \ (\lambda r. \ q \ (r \ p))
\]

\[
\begin{align*}
\text{xprt } [\ ] & := \lambda q. \ q \ (\lambda p. \ p) \\
\text{xprt } [\text{True} \ | \ l] & := \lambda q. \ \text{xprt } l \ (\lambda r. \ \lambda x. \ q \ (\lambda p. \ \lambda x. \ r \ (p \ x))) \\
\text{xprt } [\text{False} \ | \ l] & := \lambda q. \ \text{xprt } l \ (\lambda r. \ q \ (\lambda p. \ \lambda x. \ r \ (p \ x)))
\end{align*}
\]

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.

\[
\text{export } [\text{False, True}] \quad \text{func} \quad \text{arg} \\
\equiv \quad \lambda y. \ \text{func} \ (\lambda x. \ \text{arg} \ x \ y)
\]

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

\[
\text{export } [\ ] \quad \text{func} \quad \text{arg} \\
\equiv \quad \text{func} \quad \text{arg}
\]

a uniform semantics for Mandarin wh-NPs
sketch of the lexical analysis of wh scope 2

\[ \text{xprt} \[ \] \equiv \lambda q''. q'' \left( \lambda p''. p'' \right) \]

\[ \text{xprt} \[ \text{True} \] \equiv \lambda q'. \text{xprt} \[ \] \left( \lambda r'. \lambda y. q' \left( \lambda p'. r' \left( p' y \right) \right) \right) \]
\[ \equiv \lambda q'. \left( \lambda q''. q'' \left( \lambda p''. p'' \right) \right) \left( \lambda r'. \lambda y. q' \left( \lambda p'. r' \left( p' y \right) \right) \right) \]
\[ \equiv \lambda q'. \left( \lambda r'. \lambda y. q' \left( \lambda p'. r' \left( p' y \right) \right) \right) \left( \lambda p''. p'' \right) \]
\[ \equiv \lambda q'. \lambda y. q' \left( \lambda p'. \left( \lambda p''. p'' \right) \left( p' y \right) \right) \]
\[ \equiv \lambda q'. \lambda y. q' \left( \lambda p'. p' y \right) \]

\[ \text{xprt} \[ \text{False, True} \] \equiv \lambda q. \text{xprt} \[ \text{True} \] \left( \lambda r. q \left( \lambda p. \lambda x. r \left( p x \right) \right) \right) \]
\[ \equiv \lambda q. \left( \lambda q'. \lambda y. q' \left( \lambda p'. p' y \right) \right) \left( \lambda r. q \left( \lambda p. \lambda x. r \left( p x \right) \right) \right) \]
\[ \equiv \lambda q. \lambda y. \left( \lambda r. q \left( \lambda p. \lambda x. r \left( p x \right) \right) \right) \left( \lambda p'. p' y \right) \]
\[ \equiv \lambda q. \lambda y. q \left( \lambda p. \lambda x. \left( \lambda p'. p' y \right) \left( p x \right) \right) \]
\[ \equiv \lambda q. \lambda y. q \left( \lambda p. \lambda x. p x y \right) \]

\[ \text{export} \[ \text{False, True} \] \text{ func arg} \equiv \text{xprt} \[ \text{False, True} \] \left( \lambda r. \text{func} \left( r \text{ arg} \right) \right) \]
\[ \equiv \left( \lambda q. \lambda y. q \left( \lambda p. \lambda x. p x y \right) \right) \left( \lambda r. \text{func} \left( r \text{ arg} \right) \right) \]
\[ \equiv \lambda y. \left( \lambda r. \text{func} \left( r \text{ arg} \right) \right) \left( \lambda p. \lambda x. p x y \right) \]
\[ \equiv \lambda y. \text{func} \left( \left( \lambda p. \lambda x. p x y \right) \text{ arg} \right) \]
\[ \equiv \lambda y. \text{func} \left( \lambda x. \text{arg} x y \right) \]

a uniform semantics for Mandarin wh-NPs
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.

\[
[r\text{èn}\text{wé}i] := \lambda c. \lambda s. s (\lambda x. \text{export } l \text{ (think } x \text{ ) } c)
\]

think :: Ent → Bool → Bool

The translation of a sentence embedding verb such as $r\text{èn}\text{wé}i$ ‘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 $r\text{èn}\text{wé}i$ combines.

Question embedding verbs such as $xi\text{ăng zhīdào}$ 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:

\[ \forall (p :: a \rightarrow \text{Bool}) := p = \lambda x :: a. \top \]
\[ \forall (p :: b \rightarrow a \rightarrow \text{Bool}) := p = \lambda y :: b. \lambda x :: a. \top \]

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, ⟨a, b⟩ is a conjoinable type

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

∅ ⊩ Conj t
Conj b ⊩ Conj ∀a.(a → b)

or the more appealing notation of Haskell\(^1\) [PH\(^+\)97]:

\begin{verbatim}
   class Conj a
  instance Conj t
  instance (Conj b) ⇒ Conj (a → b)
\end{verbatim}

\(^1\)A functional language spoken in Nottingham, Yale, and other places.

a uniform semantics for Mandarin wh-NPs
generalized connectives and type overloading

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

\[
X \sqcap Y = X \land Y \text{ if } X \text{ and } Y \text{ are truth values}
\]
\[
= \{ \langle z, x \sqcap y \rangle \mid \langle z, x \rangle \in X \text{ and } \langle z, y \rangle \in Y \}\n\]
if \( X \) and \( Y \) are functions

(which are represented as sets of ordered pairs)

\[
X \sqcup Y = X \lor Y \text{ if } X \text{ and } Y \text{ are truth values}
\]
\[
= \{ \langle z, x \sqcup y \rangle \mid \langle z, x \rangle \in X \text{ and } \langle z, y \rangle \in Y \}\n\]
if \( X \) and \( Y \) are functions.

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

\[
\text{class Conj } a \text{ where}
\]
\[
\sqcap, \sqcup :: a \to a \to a
\]
\[
\text{instance Conj } t \text{ where}
\]
\[
x \sqcap y := x \land y
\]
\[
x \sqcup y := x \lor y
\]
\[
\text{instance (Conj } b) \Rightarrow \text{ Conj } (a \to b) \text{ where}
\]
\[
x \sqcap y := \lambda z. (x z) \sqcap (y z)
\]
\[
x \sqcup y := \lambda z. (x z) \sqcup (y z)
\]
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.

\textbf{class} SemiLattice \( a \) \textbf{where}

\( \sqcap :: a \to a \to a \)
\( \sqsubseteq :: a \to a \to \text{Bool} \)

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

\textbf{data} Bool := False | True

\textbf{data} Ent := ZL\(_7 \) | Z\(_3 \) | L\(_4 \)

\textbf{instance} SemiLattice Bool \textbf{where}

True \( \sqcap x := x \)
False \( \sqcap x := \text{False} \)

\textbf{instance} SemiLattice Ent \textbf{where}

\( x \sqcap x := x \)
\( Z_3 \sqcap L_4 := ZL_7 \)
\( L_4 \sqcap Z_3 := ZL_7 \)

\textbf{instance} (SemiLattice \( a \)) \Rightarrow SemiLattice (b \to a) \textbf{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) ⇒ BooleanLattice a where
  ⊤, ⊥ :: a
  ¬ :: a → a
  ⊔ :: a → a → a
  ∀, ∃ :: a → Bool

  ⊥ := ¬⊤
  x ⊔ y := ¬(¬x ⊓ ¬y)
  ∀x := x = ⊤
  ∃x := x ≠ ⊥
```

```
instance BooleanLattice Bool where
  ⊤ := True
  ¬True := False
  ¬False := True
```

```
instance (BooleanLattice a) ⇒ BooleanLattice (b → a) where
  ⊤ := λy. ⊤
  ¬x := λy. ¬(x y)
```

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 \[\text{[Gaz80, PR83]}\]
\( \sqcup \) generalized disjunction \[\text{[Gaz80, PR83]}\]
\( \sqsubseteq \) generalized entailment \[\text{[GS89]}\]
\( \bot \) zero \[\text{[GS82, p. 202]}\]
\( \exists \) existential closure \[\text{various}\]
\( \forall \) universal closure \[\text{various}\]

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

\[ =, \neq :: a \rightarrow a \rightarrow \text{Bool} \]

\[ \sqsubseteq :: (\text{SemiLattice } a) \Rightarrow a \rightarrow a \rightarrow \text{Bool} \]
\[ x \sqsubseteq y := (x \sqcap y) = x \]

\[ \forall :: (\text{BooleanLattice } a) \Rightarrow a \rightarrow \text{Bool} \]
\[ \forall x := x = \top \]

\[ \exists :: (\text{BooleanLattice } a) \Rightarrow a \rightarrow \text{Bool} \]
\[ \exists x := x \neq \bot \]

\[ \dot{i} :: (a \rightarrow b) \rightarrow a \rightarrow a \rightarrow \text{Bool} \]
\[ \dot{i} x := \lambda u. \lambda w. (x u) = (x w) \]

The last three predicates are closure operators in the sense that

\[ \forall(\forall x) \equiv \forall x \quad x \sqsubseteq y \text{ entails } (\forall x) \sqsubseteq (\forall y) \]
\[ \exists(\exists x) \equiv \exists x \quad x \sqsubseteq y \text{ entails } (\exists x) \sqsubseteq (\exists y) \]
\[ \dot{i}(\dot{i} x) \equiv \dot{i} x \]

Note that \( \dot{i} \) is often called a “question operator” despite the fact that it really only is a strong exhaustivity operator.
generalized predicates based on equality 2

\[ \forall (\forall x :: a) \]
\[ \equiv \forall (x :: a = \top :: a) \]
\[ \equiv ((x :: a = \top :: a) :: \text{Bool} = \top :: \text{Bool}) :: \text{Bool} \]
\[ \equiv (x :: a = \top :: a) :: \text{Bool} \]
\[ \equiv \forall (x :: a) \]

Similarly for \( \exists \).

\[ i(i\vec{x}) \]
\[ \equiv i(\lambda u'. \lambda w'. (x u') = (x w')) \]
\[ \equiv \lambda u. \lambda w. ((\lambda u'. \lambda w'. (x u') = (x w')) u) = ((\lambda u'. \lambda w'. (x u') = (x w')) w) \]
\[ \equiv \lambda u. \lambda w. (\lambda w'. (x u) = (x w')) = (\lambda w'. (x w) = (x w')) \]
\[ \equiv \lambda u. \lambda w. (x u) = (x w) \]
\[ \equiv i\vec{x} \]
putting it all together

\textbf{data} \; \text{Bool} \; := \; \text{False} \mid \text{True} \\
\textbf{data} \; \text{Ent} \; := \; Z_3 \mid L_4 \mid W_5 \mid \ldots \\
\textbf{data} \; \text{Ind} \; := \; \text{Idx}_0 \mid \text{Idx}_1 \mid \ldots \\

\llbracket \text{Zhāngsān} \rrbracket :: (\text{BooleanLattice } a) \Rightarrow \\
\quad \quad ((\text{Ind} \rightarrow (\text{Ind} \rightarrow \text{Ent} \rightarrow \text{Bool}) \rightarrow \text{Bool}) \rightarrow a) \rightarrow a \\
\llbracket \text{Zhāngsān} \rrbracket := \lambda q. q (\lambda w. \lambda p. p w Z_3) \\

\llbracket \text{shéi} \rrbracket :: (\text{BooleanLattice } a) \Rightarrow \\
\quad \quad ((\text{Ind} \rightarrow (\text{Ind} \rightarrow \text{Ent} \rightarrow \text{Bool}) \rightarrow \text{Bool}) \rightarrow a) \rightarrow \text{Ent} \rightarrow a \\
\llbracket \text{shéi} \rrbracket := \lambda q. \lambda x. q (\lambda w. \lambda p. (\text{person} \sqcap p) \; w \; x) \\
\text{person} :: \text{Ind} \rightarrow \text{Ent} \rightarrow \text{Bool} \\

All NP types are instances of \\

(\text{BooleanLattice } a, \text{BooleanLattice } b) \Rightarrow \\
\quad \quad ((\text{Ind} \rightarrow (\text{Ind} \rightarrow \text{Ent} \rightarrow \text{Bool}) \rightarrow \text{Bool}) \rightarrow a) \rightarrow b \\

a \; \text{uniform semantics for Mandarin wh-NPs}
Schema for intransitive verbs:

\[ [iv] :: (BooleanLattice \ a) \Rightarrow ((\text{Ind} \rightarrow (\text{Ind} \rightarrow \text{Ent} \rightarrow \text{Bool}) \rightarrow \text{Bool}) \rightarrow \text{Bool}) \rightarrow \text{Ind} \rightarrow \text{a} \]

\[ [iv] := \lambda s'. \lambda w. s' (\lambda s. s w (\lambda w. \lambda x. iv w x)) \]

\[ iv :: \text{Ind} \rightarrow \text{Ent} \rightarrow \text{Bool} \]

Schema for transitive verbs:

\[ [tv] :: (BooleanLattice \ a, BooleanLattice \ b) \Rightarrow ((\text{Ind} \rightarrow (\text{Ind} \rightarrow \text{Ent} \rightarrow \text{Bool}) \rightarrow \text{Bool}) \rightarrow \text{Bool}) \rightarrow \text{a} \]

\[ \rightarrow (((\text{Ind} \rightarrow (\text{Ind} \rightarrow \text{Ent} \rightarrow \text{Bool}) \rightarrow \text{Bool}) \rightarrow \text{a}) \rightarrow \text{b}) \rightarrow \text{Ind} \rightarrow \text{b} \]

\[ [tv] := \lambda o'. \lambda s'. \lambda w. s' (\lambda s. o' (\lambda o. s w (\lambda w. \lambda x. o w (\lambda w. \lambda y. tv w x y)))) \]

\[ tv :: \text{Ind} \rightarrow \text{Ent} \rightarrow \text{Ent} \rightarrow \text{Bool} \]

Schema for sentence embedding verbs:

\[ [sev] :: (BooleanLattice \ a, BooleanLattice \ b) \Rightarrow (\text{Ind} \rightarrow \text{b}) \rightarrow ((\text{Ind} \rightarrow (\text{Ind} \rightarrow \text{Ent} \rightarrow \text{Bool}) \rightarrow \text{Bool}) \rightarrow \text{Bool}) \rightarrow \text{a} \]

\[ \rightarrow (((\text{Ind} \rightarrow (\text{Ind} \rightarrow \text{Ent} \rightarrow \text{Bool}) \rightarrow \text{Bool}) \rightarrow \text{a}) \rightarrow \text{Ind} \rightarrow \text{a} \]

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

A uniform semantics for Mandarin wh-NPs
sentembver := λv. λc. λs′. λw. s′ (λs. s w (λw. λx. export [False|[l] (v w x) c))

Lexical entry for 哲 ‘think’:

[哲] := sentembver think
think :: Ind → Ent → (Ind → Bool) → Bool

Lexical entry for 想哲 ‘wonder’:

[想哲] := sentembver wonder
wonder :: (BooleanLattice b) ⇒ Ind → Ent → (Ind → a → b) → Bool

Lexical entry for 想哲 ‘wish’:

[想哲] := sentembver (λw. λx. λp. wish w x (λw′. ∃ (p w′)))
wish :: Ind → Ent → (Ind → Bool) → Bool
polar questions

(25) 張三 知道 李四 會 來。
Zhāngsān zhīdào Lìsì 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 Lìsì 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 Lìsì 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 Lìsì huì-bu-huì lái.
Zhangsan knows Lisi will-not-will come 'Zhangsan knows whether Lisi will come.' (ambiguous?)

(29) 張三 認為 李四 會不會 來？
Zhāngsān rènwéi Lìsì huì-bu-huì lái?
Zhangsan thinks Lisi will-not-will come 'Does Zhangsan think Lisi will come?'

a uniform semantics for Mandarin wh-NPs
## sketch of the analysis of polar questions

<table>
<thead>
<tr>
<th>kind of interrogative</th>
<th>extension</th>
<th>type</th>
<th>uncurried type</th>
</tr>
</thead>
<tbody>
<tr>
<td>one constituent</td>
<td>unary relation</td>
<td>$a \to \text{Bool}$</td>
<td>$(a) \to \text{Bool}$</td>
</tr>
<tr>
<td>two constituents</td>
<td>binary relation</td>
<td>$b \to a \to \text{Bool}$</td>
<td>$(b, a) \to \text{Bool}$</td>
</tr>
<tr>
<td>three constituents</td>
<td>ternary relation</td>
<td>$c \to b \to a \to \text{Bool}$</td>
<td>$(c, b, a) \to \text{Bool}$</td>
</tr>
<tr>
<td>polar</td>
<td>nullary relation</td>
<td>()</td>
<td>()</td>
</tr>
</tbody>
</table>

Let polar interrogatives have denotations of type $\text{Ind} \to 1 \to \text{Bool}$. 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


