Presupposition projection as a scope phenomenon

Julian Grove

CLASP, University of Gothenburg

LINGUAE seminar, February 25, 2021

The empirical observation

We have linguistic devices that grammatically encode what we take for granted in making an utterance.

The empirical observation

We have linguistic devices that grammatically encode what we take for granted in making an utterance.

1. Karlos brought his car.

The empirical observation

We have linguistic devices that grammatically encode what we take for granted in making an utterance.

1. Karlos brought his car.
   - Karlos has a car. (presupposition)

The empirical observation

How do we identity presuppositions?

The empirical observation

How do we identity presuppositions?

- family-of-sentence tests (Chierchia and McConnell-Ginet, 2000)

The empirical observation

How do we identity presuppositions?

- family-of-sentence tests (Chierchia and McConnell-Ginet, 2000)
- "hey, wait a minute!" test (von Fintel, 2004)

The empirical observation

How do we identity presuppositions?

- family-of-sentence tests (Chierchia and McConnell-Ginet, 2000)
- “hey, wait a minute!” test (von Fintel, 2004)

The empirical observation

How do we identity presuppositions?

- family-of-sentence tests (Chierchia and McConnell-Ginet, 2000)
- “hey, wait a minute!” test (von Fintel, 2004)

Major research question: what grammatical properties of an expression give rise to its presuppositions?

How do we identity presuppositions?

- family-of-sentence tests (Chierchia and McConnell-Ginet, 2000)
- “hey, wait a minute!” test (von Fintel, 2004)

Major research question: what grammatical properties of an expression give rise to its presuppositions?

A compositional account answers two questions:

How do we identity presuppositions?

- family-of-sentence tests (Chierchia and McConnell-Ginet, 2000)
- “hey, wait a minute!” test (von Fintel, 2004)

Major research question: what grammatical properties of an expression give rise to its presuppositions?

A compositional account answers two questions:

- How do we grammatically encode presuppositions in simple expressions (presupposition triggers)?

The empirical observation

How do we identity presuppositions?

- family-of-sentence tests (Chierchia and McConnell-Ginet, 2000)
- "hey, wait a minute!" test (von Fintel, 2004)

Major research question: what grammatical properties of an expression give rise to its presuppositions?

A compositional account answers two questions:

- How do we grammatically encode presuppositions in simple expressions (presupposition triggers)?
- How do presuppositions project in complex expressions?

The empirical observation

How do we identity presuppositions?

- family-of-sentence tests (Chierchia and McConnell-Ginet, 2000)
- “hey, wait a minute!” test (von Fintel, 2004)

Major research question: what grammatical properties of an expression give rise to its presuppositions?

A compositional account answers two questions:

- How do we grammatically encode presuppositions in simple expressions (presupposition triggers)?
- How do presuppositions project in complex expressions?
  - The “projection problem” (Langendoen and Savin, 1971)

Today's talk

Outline:

Today's talk

Outline:

- Investigate an influential compositional framework for studying presupposition projection: "satisfaction theory" (Geurts, 1996)

Today's talk

Outline:

- Investigate an influential compositional framework for studying presupposition projection: "satisfaction theory" (Geurts, 1996)
  - Heim 1983

Today's talk

Outline:

- Investigate an influential compositional framework for studying presupposition projection: "satisfaction theory" (Geurts, 1996)
  - Heim 1983
  - compositionally derived conditions on dynamic update $\rightsquigarrow$ presuppositions

Today's talk

Outline:

- Investigate an influential compositional framework for studying presupposition projection: "satisfaction theory" (Geurts, 1996)
  - Heim 1983
  - compositionally derived conditions on dynamic update
    $\rightsquigarrow$ presuppositions
  - the "proviso" problem

Outline:

- Investigate an influential compositional framework for studying presupposition projection: "satisfaction theory" (Geurts, 1996)
  - Heim 1983
  - compositionally derived conditions on dynamic update $\rightsquigarrow$ presuppositions
  - the "proviso" problem
- Reconsider the satisfaction account in light of a scope-taking mechanism

Today's talk

Outline:

- Investigate an influential compositional framework for studying presupposition projection: "satisfaction theory" (Geurts, 1996)
  - Heim 1983
  - compositionally derived conditions on dynamic update $\rightsquigarrow$ presuppositions
  - the "proviso" problem
- Reconsider the satisfaction account in light of a scope-taking mechanism
  - Explore predictions

Outline:

- Investigate an influential compositional framework for studying presupposition projection: "satisfaction theory" (Geurts, 1996)
  - Heim 1983
  - compositionally derived conditions on dynamic update $\rightsquigarrow$ presuppositions
  - the "proviso" problem
- Reconsider the satisfaction account in light of a scope-taking mechanism
  - Explore predictions
- Presupposition triggers in the scopes of propositional attitude verbs

We are here

1. The satisfaction theory
2. A scopal account
3. Presupposition and propositional attitude verbs

Rough sketch of out it works

- Basic ideas come from Heim 1983

Rough sketch of out it works

- Basic ideas come from Heim 1983
- Sentences denote *context change potentials*

Rough sketch of out it works

- Basic ideas come from Heim 1983
- Sentences denote *context change potentials*

$$[\![\Delta \quad ; \quad \textit{it's raining}]\!]$$

Rough sketch of out it works

- Basic ideas come from Heim 1983
- Sentences denote *context change potentials*

$$\begin{aligned} & [\![\Delta \quad ; \quad \textit{it's raining}]\!] \\ = \; & [\![\Delta]\!] + [\![\textit{it's raining}]\!] \end{aligned}$$

Rough sketch of out it works

- Basic ideas come from Heim 1983
- Sentences denote *context change potentials*

$$
\begin{aligned}
& [\![\Delta \;\; ; \;\; \textit{it's raining}]\!] \\
& = [\![\Delta]\!] + [\![\textit{it's raining}]\!] \\
\text{e.g.,} \quad & = \{w \in \mathcal{W} \mid [\![\Delta]\!]^w = \top\} \cap \{w \in \mathcal{W} \mid \textbf{rain}\, w\}
\end{aligned}
$$

Rough sketch of out it works

- Basic ideas come from Heim 1983
- Sentences denote *context change potentials*

$$
\begin{aligned}
& [\![\Delta \;\; ; \;\; \textit{it's raining}]\!] \\
= \;& [\![\Delta]\!] + [\![\textit{it's raining}]\!] \\
\text{e.g.,} \quad = \;& \{w \in \mathcal{W} \mid [\![\Delta]\!]^{w} = \top\} \cap \{w \in \mathcal{W} \mid \mathbf{rain}\, w\}
\end{aligned}
$$

- What is $+$? (Depends on your more specific theory.)

Rough sketch of out it works

- Basic ideas come from Heim 1983
- Sentences denote *context change potentials*

$$
\begin{aligned}
& [\![\Delta \;\; ; \;\; \textit{it's raining}]\!] \\
= \;& [\![\Delta]\!] + [\![\textit{it's raining}]\!] \\
\text{e.g.,} \quad = \;& \{w \in \mathcal{W} \mid [\![\Delta]\!]^{w} = \top\} \cap \{w \in \mathcal{W} \mid \mathbf{rain}\, w\}
\end{aligned}
$$

- What is $+$? (Depends on your more specific theory.)
  - Might amount to set intersection (of sets of worlds, assignments, …)

Rough sketch of how it works

- What if the sentence updating Δ has presuppositions?

Rough sketch of how it works

- What if the sentence updating Δ has presuppositions?

$$[\![\Delta]\!] + [\![\textit{Karlos brought his car}]\!]$$

Rough sketch of how it works

- What if the sentence updating Δ has presuppositions?

$$[\![\Delta]\!] + [\![\textit{Karlos brought his car}]\!]$$

- Δ "admits" *Karlos brought his car* only if Δ entails *Karlos has a car.*

- What if the sentence updating Δ has presuppositions?

$$[\![\Delta]\!] + [\![\textit{Karlos brought his car}]\!]$$

- Δ "admits" *Karlos brought his car* only if Δ entails *Karlos has a car.*
  - "Stalnaker's bridge" (von Fintel, 2008)

Rough sketch of how it works

- What if the sentence updating Δ has presuppositions?

$$[\![\Delta]\!] + [\![\textit{Karlos brought his car}]\!]$$

- Δ “admits” *Karlos brought his car* only if Δ entails *Karlos has a car.*
    - “Stalnaker’s bridge” (von Fintel, 2008)
- Foregoing assumptions are meant to provide a way of determining what a sentence’s presuppositions *are*:

- What if the sentence updating Δ has presuppositions?

$$[\![\Delta]\!] + [\![\textit{Karlos brought his car}]\!]$$

- Δ "admits" *Karlos brought his car* only if Δ entails *Karlos has a car.*
  - "Stalnaker's bridge" (von Fintel, 2008)
- Foregoing assumptions are meant to provide a way of determining what a sentence's presuppositions *are*:
  - S_1 presupposes S_1 iff every context Δ, such that $[\![\Delta]\!] + [\![S_1]\!]$ is successful, entails S_2.

- What if the sentence updating Δ has presuppositions?

$$[\![\Delta]\!] + [\![\textit{Karlos brought his car}]\!]$$

- Δ "admits" *Karlos brought his car* only if Δ entails *Karlos has a car.*
  - "Stalnaker's bridge" (von Fintel, 2008)
- Foregoing assumptions are meant to provide a way of determining what a sentence's presuppositions *are*:
  - S_1 presupposes S_1 iff every context Δ, such that $[\![\Delta]\!] + [\![S_1]\!]$ is successful, entails S_2.
  - *Karlos brought his car* $\rightsquigarrow$ *Karlos has a car*

Rough sketch of how it works

Explaining projection behavior: just a matter of using $+$ in the right way.

1. Karlos has a car, and he brought his car.

Rough sketch of how it works

Explaining projection behavior: just a matter of using $+$ in the right way.

1. Karlos has a car, and he brought his car.
   - $\Delta + [\![(1)]\!] = (\Delta + [\![\textit{K has a car}]\!]) + [\![\textit{K brought his car}]\!]$

Rough sketch of how it works

Explaining projection behavior: just a matter of using $+$ in the right way.

1. Karlos has a car, and he brought his car.
   - $\Delta + [\![(1)]\!] = (\Delta + [\![\textit{K has a car}]\!]) + [\![\textit{K brought his car}]\!]$
   - Update is successful if each of the individual updates is successful.

Explaining projection behavior: just a matter of using $+$ in the right way.

1. Karlos has a car, and he brought his car.
   - $\Delta + [\![(1)]\!] = (\Delta + [\![\textit{K has a car}]\!]) + [\![\textit{K brought his car}]\!]$
   - Update is successful if each of the individual updates is successful.
   - … if Δ entails *if Karlos has a car, then Karlos has a car*

Rough sketch of how it works

Explaining projection behavior: just a matter of using $+$ in the right way.

1. Karlos has a car, and he brought his car.
   - $\Delta + [\![(1)]\!] = (\Delta + [\![\textit{K has a car}]\!]) + [\![\textit{K brought his car}]\!]$
   - Update is successful if each of the individual updates is successful.
   - … if Δ entails *if Karlos has a car, then Karlos has a car*
     - $\rightsquigarrow$ No presuppositions for (1).

Explaining projection behavior: just a matter of using $+$ in the right way.

1. Karlos has a car, and he brought his car.
   - $\Delta + [\![(1)]\!] = (\Delta + [\![\textit{K has a car}]\!]) + [\![\textit{K brought his car}]\!]$
   - Update is successful if each of the individual updates is successful.
   - … if Δ entails *if Karlos has a car, then Karlos has a car*
     - $\rightsquigarrow$ No presuppositions for (1).
     - I'll say that the presupposition in (1) is "filtered".

Explaining projection behavior: just a matter of using $+$ in the right way.

1. Karlos has a car, and he brought his car.
   - $\Delta + [\![(1)]\!] = (\Delta + [\![\textit{K has a car}]\!]) + [\![\textit{K brought his car}]\!]$
   - Update is successful if each of the individual updates is successful.
   - … if Δ entails *if Karlos has a car, then Karlos has a car*
     - ⋆ $\rightsquigarrow$ No presuppositions for (1).
     - ⋆ I'll say that the presupposition in (1) is "filtered".
2. If Karlos has a car, he brought his car.

Explaining projection behavior: just a matter of using $+$ in the right way.

1. Karlos has a car, and he brought his car.
   - $\Delta + [\![(1)]\!] = (\Delta + [\![\textit{K has a car}]\!]) + [\![\textit{K brought his car}]\!]$
   - Update is successful if each of the individual updates is successful.
   - … if Δ entails *if Karlos has a car, then Karlos has a car*
     - ★ $\rightsquigarrow$ No presuppositions for (1).
     - ★ I'll say that the presupposition in (1) is "filtered".
2. If Karlos has a car, he brought his car.
   - $\Delta + [\![(2)]\!]$:

Rough sketch of how it works

Explaining projection behavior: just a matter of using $+$ in the right way.

1. Karlos has a car, and he brought his car.
   - $\Delta + [\![(1)]\!] = (\Delta + [\![\textit{K has a car}]\!]) + [\![\textit{K brought his car}]\!]$
   - Update is successful if each of the individual updates is successful.
   - … if Δ entails *if Karlos has a car, then Karlos has a car*
     - ★ $\rightsquigarrow$ No presuppositions for (1).
     - ★ I'll say that the presupposition in (1) is "filtered".
2. If Karlos has a car, he brought his car.
   - $\Delta + [\![(2)]\!]$:
     - ★ $\Delta_1 = \Delta + [\![\textit{K has c}]\!]$

Rough sketch of how it works

Explaining projection behavior: just a matter of using $+$ in the right way.

1. Karlos has a car, and he brought his car.
   - $\Delta + [\![(1)]\!] = (\Delta + [\![\textit{K has a car}]\!]) + [\![\textit{K brought his car}]\!]$
   - Update is successful if each of the individual updates is successful.
   - … if Δ entails *if Karlos has a car, then Karlos has a car*
     - $\rightsquigarrow$ No presuppositions for (1).
     - I'll say that the presupposition in (1) is "filtered".
2. If Karlos has a car, he brought his car.
   - $\Delta + [\![(2)]\!]$:
     - $\Delta_1 = \Delta + [\![\textit{K has c}]\!]$
     - $\Delta_2 = \Delta + [\![\textit{K has c}]\!] + [\![\textit{K brought c}]\!]$

Rough sketch of how it works

Explaining projection behavior: just a matter of using $+$ in the right way.

1. Karlos has a car, and he brought his car.
   - $\Delta + [\![(1)]\!] = (\Delta + [\![K\ has\ a\ car]\!]) + [\![K\ brought\ his\ car]\!]$
   - Update is successful if each of the individual updates is successful.
   - … if Δ entails *if Karlos has a car, then Karlos has a car*
     - $\rightsquigarrow$ No presuppositions for (1).
     - I'll say that the presupposition in (1) is "filtered".
2. If Karlos has a car, he brought his car.
   - $\Delta + [\![(2)]\!]$:
     - $\Delta_1 = \Delta + [\![K\ has\ c]\!]$
     - $\Delta_2 = \Delta + [\![K\ has\ c]\!] + [\![K\ brought\ c]\!]$
     - $= \Delta - (\Delta_1 - \Delta_2)$

Rough sketch of how it works

Explaining projection behavior: just a matter of using $+$ in the right way.

1. Karlos has a car, and he brought his car.
   - $\Delta + [\![(1)]\!] = (\Delta + [\![K\ \textit{has a car}]\!]) + [\![K\ \textit{brought his car}]\!]$
   - Update is successful if each of the individual updates is successful.
   - … if Δ entails *if Karlos has a car, then Karlos has a car*
     - $\rightsquigarrow$ No presuppositions for (1).
     - I'll say that the presupposition in (1) is "filtered".
2. If Karlos has a car, he brought his car.
   - $\Delta + [\![(2)]\!]$:
     - $\Delta_1 = \Delta + [\![K\ \textit{has}\ c]\!]$
     - $\Delta_2 = \Delta + [\![K\ \textit{has}\ c]\!] + [\![K\ \textit{brought}\ c]\!]$
     - $= \Delta - (\Delta_1 - \Delta_2)$
   - Update is successful if each of the individual updates is.

Rough sketch of how it works

Explaining projection behavior: just a matter of using $+$ in the right way.

1. Karlos has a car, and he brought his car.
   - $\Delta + [\![(1)]\!] = (\Delta + [\![K\ has\ a\ car]\!]) + [\![K\ brought\ his\ car]\!]$
   - Update is successful if each of the individual updates is successful.
   - … if Δ entails *if Karlos has a car, then Karlos has a car*
     - $\rightsquigarrow$ No presuppositions for (1).
     - I'll say that the presupposition in (1) is "filtered".
2. If Karlos has a car, he brought his car.
   - $\Delta + [\![(2)]\!]$:
     - $\Delta_1 = \Delta + [\![K\ has\ c]\!]$
     - $\Delta_2 = \Delta + [\![K\ has\ c]\!] + [\![K\ brought\ c]\!]$
     - $= \Delta - (\Delta_1 - \Delta_2)$
   - Update is successful if each of the individual updates is.
     - $\rightsquigarrow$ No presuppositions for (2).

The proviso problem

Geurts (1996): big problem!

Geurts (1996): big problem!

1. It's raining, and my car is too far away.

Geurts (1996): big problem!

1. It's raining, and my car is too far away.
   - Update successful if each individual update is.

Geurts (1996): big problem!

1. It's raining, and my car is too far away.
   - Update successful if each individual update is.
     - ★ $\Delta + [\![(1)]\!] = (\Delta + [\![\textit{it's raining}]\!]) + [\![\textit{my car is too far away}]\!]$

Geurts (1996): big problem!

1. It's raining, and my car is too far away.
   - Update successful if each individual update is.
     - ★ $\Delta + [\![(1)]\!] = (\Delta + [\![\textit{it's raining}]\!]) + [\![\textit{my car is too far away}]\!]$
     - ★ …iff the context entails *if it's raining, I have a car*

The proviso problem

Geurts (1996): big problem!

1. It's raining, and my car is too far away.
   - Update successful if each individual update is.
     - $\Delta + [\![(1)]\!] = (\Delta + [\![\textit{it's raining}]\!]) + [\![\textit{my car is too far away}]\!]$
     - …iff the context entails *if it's raining, I have a car*
     - (1) $\rightsquigarrow$ *if it's raining, I have a car* ☹

Geurts (1996): big problem!

1. It's raining, and my car is too far away.
    - Update successful if each individual update is.
        - $\Delta + [\![(1)]\!] = (\Delta + [\![\textit{it's raining}]\!]) + [\![\textit{my car is too far away}]\!]$
        - …iff the context entails *if it's raining, I have a car*
        - (1) $\rightsquigarrow$ *if it's raining, I have a car* ☹
2. If the airport is nearby, I can pick my sister up when she lands.

Geurts (1996): big problem!

1. It's raining, and my car is too far away.
   - Update successful if each individual update is.
     - $\Delta + [\![(1)]\!] = (\Delta + [\![\textit{it's raining}]\!]) + [\![\textit{my car is too far away}]\!]$
     - …iff the context entails *if it's raining, I have a car*
     - (1) $\rightsquigarrow$ *if it's raining, I have a car* ☹
2. If the airport is nearby, I can pick my sister up when she lands.
   - Individual updates to Δ:

Geurts (1996): big problem!

1. It's raining, and my car is too far away.
   - Update successful if each individual update is.
     - $\Delta + [\![(1)]\!] = (\Delta + [\![\textit{it's raining}]\!]) + [\![\textit{my car is too far away}]\!]$
     - ...iff the context entails *if it's raining, I have a car*
     - (1) $\rightsquigarrow$ *if it's raining, I have a car* ☹
2. If the airport is nearby, I can pick my sister up when she lands.
   - Individual updates to Δ:
     - $\Delta_1 = \Delta + [\![\textit{airport nearby}]\!]$

Geurts (1996): big problem!

1. It's raining, and my car is too far away.
   - Update successful if each individual update is.
     - $\Delta + [\![(1)]\!] = (\Delta + [\![\textit{it's raining}]\!]) + [\![\textit{my car is too far away}]\!]$
     - …iff the context entails *if it's raining, I have a car*
     - (1) $\rightsquigarrow$ *if it's raining, I have a car* ☹
2. If the airport is nearby, I can pick my sister up when she lands.
   - Individual updates to Δ:
     - $\Delta_1 = \Delta + [\![\textit{airport nearby}]\!]$
     - $\Delta_2 = \Delta + [\![\textit{airport nearby}]\!] + [\![\textit{pick sister up}]\!]$

The proviso problem

Geurts (1996): big problem!

1. It's raining, and my car is too far away.
   - Update successful if each individual update is.
     - ★ $\Delta + [\![(1)]\!] = (\Delta + [\![\textit{it's raining}]\!]) + [\![\textit{my car is too far away}]\!]$
     - ★ …iff the context entails *if it's raining, I have a car*
     - ★ (1) $\rightsquigarrow$ *if it's raining, I have a car* ☹
2. If the airport is nearby, I can pick my sister up when she lands.
   - Individual updates to Δ:
     - ★ $\Delta_1 = \Delta + [\![\textit{airport nearby}]\!]$
     - ★ $\Delta_2 = \Delta + [\![\textit{airport nearby}]\!] + [\![\textit{pick sister up}]\!]$
     - ★ $= \Delta - (\Delta_1 - \Delta_2)$

Geurts (1996): big problem!

1. It's raining, and my car is too far away.
   - Update successful if each individual update is.
     - $\Delta + [\![(1)]\!] = (\Delta + [\![\textit{it's raining}]\!]) + [\![\textit{my car is too far away}]\!]$
     - ...iff the context entails *if it's raining, I have a car*
     - (1) $\rightsquigarrow$ *if it's raining, I have a car* ☹
2. If the airport is nearby, I can pick my sister up when she lands.
   - Individual updates to Δ:
     - $\Delta_1 = \Delta + [\![\textit{airport nearby}]\!]$
     - $\Delta_2 = \Delta + [\![\textit{airport nearby}]\!] + [\![\textit{pick sister up}]\!]$
     - $= \Delta - (\Delta_1 - \Delta_2)$
   - (2) $\rightsquigarrow$ *if the airport is nearby, I have a sister* ☹

- According to the satisfaction theory, filtration is automatic.

The proviso problem

- According to the satisfaction theory, filtration is automatic.
- But sometimes it shouldn't happen.

We are here

1. The satisfaction theory
2. A scopal account
3. Presupposition and propositional attitude verbs

Following Heim and Kratzer (1998), we can encode an expression's presuppositions by allowing meanings to be partial.

Compositional semantics

Following Heim and Kratzer (1998), we can encode an expression's presuppositions by allowing meanings to be partial.

- $[\![\textit{his}_i\ \textit{car}]\!]^{w,g}$

Following Heim and Kratzer (1998), we can encode an expression's presuppositions by allowing meanings to be partial.

- $[\![his_i\ car]\!]^{w,g}$
  - the car of $g(i)$ in w, if one exists

Following Heim and Kratzer (1998), we can encode an expression's presuppositions by allowing meanings to be partial.

- $[\![\textit{his}_i \ \textit{car}]\!]^{w,g}$
  - the car of $g(i)$ in w, if one exists
  - #, otherwise

Compositional semantics

1. Karlos brought his car.

1. Karlos brought his car.

   - *Functional Application* (Heim and Kratzer, 1998, p. 105, ex. 13′)
     If α is a branching node and $\{\beta, \gamma\}$ the set of its daughter, then for any assignment a, α is in the domain of $[\![\cdot]\!]^a$ if both β and γ are, and $[\![\beta]\!]^a$ is a function whose domain contains $[\![\gamma]\!]^a$. In that case, $[\![\alpha]\!]^a = [\![\beta]\!]^a([\![\gamma]\!]^a)$.

1. Karlos brought his car.

   - *Functional Application* (Heim and Kratzer, 1998, p. 105, ex. 13′)
     If α is a branching node and $\{\beta, \gamma\}$ the set of its daughter, then for any assignment a, α is in the domain of $[\![\cdot]\!]^a$ if both β and γ are, and $[\![\beta]\!]^a$ is a function whose domain contains $[\![\gamma]\!]^a$. In that case, $[\![\alpha]\!]^a = [\![\beta]\!]^a([\![\gamma]\!]^a)$.

   - $[\![\textit{Karlos brought his}_i \textit{ car}]\!]^{w,g}$

(1) Karlos brought his car.

- *Functional Application* (Heim and Kratzer, 1998, p. 105, ex. 13′)
  If α is a branching node and $\{\beta, \gamma\}$ the set of its daughter, then for any assignment a, α is in the domain of $[\![\cdot]\!]^a$ if both β and γ are, and $[\![\beta]\!]^a$ is a function whose domain contains $[\![\gamma]\!]^a$. In that case, $[\![\alpha]\!]^a = [\![\beta]\!]^a([\![\gamma]\!]^a)$.

- $[\![\textit{Karlos brought his}_i\ \textit{car}]\!]^{w,g}$
  - ★ $[\![\textit{brought}]\!]^{w,g}([\![\textit{his}_i\ \textit{car}]\!]^{w,g})([\![\textit{Karlos}]\!]^{w,g})$

1. Karlos brought his car.

   - *Functional Application* (Heim and Kratzer, 1998, p. 105, ex. 13′)
     If α is a branching node and $\{\beta, \gamma\}$ the set of its daughter, then for any assignment a, α is in the domain of $[\![\cdot]\!]^a$ if both β and γ are, and $[\![\beta]\!]^a$ is a function whose domain contains $[\![\gamma]\!]^a$. In that case, $[\![\alpha]\!]^a = [\![\beta]\!]^a([\![\gamma]\!]^a)$.

   - $[\![\textit{Karlos brought his}_i\ \textit{car}]\!]^{w,g}$
     - $[\![\textit{brought}]\!]^{w,g}([\![\textit{his}_i\ \textit{car}]\!]^{w,g})([\![\textit{Karlos}]\!]^{w,g})$
     - 1, if $g(i)$ has a car and Karlos brought it (in w)

1. Karlos brought his car.

- *Functional Application* (Heim and Kratzer, 1998, p. 105, ex. 13′)
  If α is a branching node and $\{\beta, \gamma\}$ the set of its daughter, then for any assignment a, α is in the domain of $[\![\cdot]\!]^a$ if both β and γ are, and $[\![\beta]\!]^a$ is a function whose domain contains $[\![\gamma]\!]^a$. In that case, $[\![\alpha]\!]^a = [\![\beta]\!]^a([\![\gamma]\!]^a)$.

- $[\![\textit{Karlos brought his}_i \textit{ car}]\!]^{w,g}$
  - $[\![\textit{brought}]\!]^{w,g}([\![\textit{his}_i \textit{ car}]\!]^{w,g})([\![\textit{Karlos}]\!]^{w,g})$
  - 1, if $g(i)$ has a car and Karlos brought it (in w)
  - 0, if $g(i)$ has a car and Karlos didn't bring it

1. Karlos brought his car.

- *Functional Application* (Heim and Kratzer, 1998, p. 105, ex. 13′)
  If α is a branching node and $\{\beta, \gamma\}$ the set of its daughter, then for any assignment a, α is in the domain of $[\![\cdot]\!]^a$ if both β and γ are, and $[\![\beta]\!]^a$ is a function whose domain contains $[\![\gamma]\!]^a$. In that case, $[\![\alpha]\!]^a = [\![\beta]\!]^a([\![\gamma]\!]^a)$.

- $[\![\textit{Karlos brought his}_i\textit{ car}]\!]^{w,g}$
  - $[\![\textit{brought}]\!]^{w,g}([\![\textit{his}_i\textit{ car}]\!]^{w,g})([\![\textit{Karlos}]\!]^{w,g})$
  - 1, if $g(i)$ has a car and Karlos brought it (in w)
  - 0, if $g(i)$ has a car and Karlos didn't bring it
  - #, if $g(i)$ doesn't have a car

1. If Karlos has a car, Karlos brought his car.

1. If Karlos has a car, Karlos brought his car.

   - *Material Conditional Rule*
     Given a material conditional, $[[\textit{if}\ \phi]\ \psi]$, and an assignment, a, if $[\![\phi]\!]^a = 0$, then $[\![[[\textit{if}\ \phi]\ \psi]]\!]^a = 1$. If $[\![\phi]\!]^a = 1$ and $[\![\psi]\!]^a$ is defined, then $[\![[[\textit{if}\ \phi]\ \psi]]\!]^a = [\![\psi]\!]^a$. $[\![[[\textit{if}\ \phi]\ \psi]]\!]^a$ is undefined if $[\![\phi]\!]^a$ is.

1. If Karlos has a car, Karlos brought his car.

- *Material Conditional Rule*
  Given a material conditional, $[[\textit{if}\ \phi]\ \psi]$, and an assignment, a, if $[\![\phi]\!]^a = 0$, then $[\![[[\textit{if}\ \phi]\ \psi]]\!]^a = 1$. If $[\![\phi]\!]^a = 1$ and $[\![\psi]\!]^a$ is defined, then $[\![[[\textit{if}\ \phi]\ \psi]]\!]^a = [\![\psi]\!]^a$. $[\![[[\textit{if}\ \phi]\ \psi]]\!]^a$ is undefined if $[\![\phi]\!]^a$ is.

- $[\![\textit{if Karlos has a car, Karlos brought his}_i\ \textit{car}]\!]^{w,g}$

1. If Karlos has a car, Karlos brought his car.

   - *Material Conditional Rule*
     Given a material conditional, $[[\mathit{if}\ \phi]\ \psi]$, and an assignment, a, if $[\![\phi]\!]^a = 0$, then $[\![[[\mathit{if}\ \phi]\ \psi]]\!]^a = 1$. If $[\![\phi]\!]^a = 1$ and $[\![\psi]\!]^a$ is defined, then $[\![[[\mathit{if}\ \phi]\ \psi]]\!]^a = [\![\psi]\!]^a$. $[\![[[\mathit{if}\ \phi]\ \psi]]\!]^a$ is undefined if $[\![\phi]\!]^a$ is.
   - $[\![\textit{if Karlos has a car, Karlos brought his}_i\textit{ car}]\!]^{w,g}$
     - 1, if $[\![\textit{K has c}]\!]^{w,g} = 0$, or $[\![\textit{K has a car}]\!]^{w,g} = 1$ and $[\![\textit{K brought his car}]\!]^{w,g} = 1$

1. If Karlos has a car, Karlos brought his car.

- *Material Conditional Rule*
  Given a material conditional, $[[if\ \phi]\ \psi]$, and an assignment, a, if $[\![\phi]\!]^a = 0$, then $[\![[[if\ \phi]\ \psi]]\!]^a = 1$. If $[\![\phi]\!]^a = 1$ and $[\![\psi]\!]^a$ is defined, then $[\![[[if\ \phi]\ \psi]]\!]^a = [\![\psi]\!]^a$. $[\![[[if\ \phi]\ \psi]]\!]^a$ is undefined if $[\![\phi]\!]^a$ is.

- $[\![\textit{if Karlos has a car, Karlos brought his}_i\textit{ car}]\!]^{w,g}$
  - ★ 1, if $[\![\textit{K has c}]\!]^{w,g} = 0$, or $[\![\textit{K has a car}]\!]^{w,g} = 1$ and $[\![\textit{K brought his car}]\!]^{w,g} = 1$
  - ★ 0, if $[\![\textit{K has a car}]\!]^{w,g} = 1$ and $[\![\textit{K brought his car}]\!]^{w,g} = 0$

1. If Karlos has a car, Karlos brought his car.

   - *Material Conditional Rule*
     Given a material conditional, $[[if\ \phi]\ \psi]$, and an assignment, a, if $[\![\phi]\!]^a = 0$, then $[\![[[if\ \phi]\ \psi]]\!]^a = 1$. If $[\![\phi]\!]^a = 1$ and $[\![\psi]\!]^a$ is defined, then $[\![[[if\ \phi]\ \psi]]\!]^a = [\![\psi]\!]^a$. $[\![[[if\ \phi]\ \psi]]\!]^a$ is undefined if $[\![\phi]\!]^a$ is.

   - $[\![\textit{if Karlos has a car, Karlos brought his}_i\textit{ car}]\!]^{w,g}$
     - 1, if $[\![\textit{K has c}]\!]^{w,g} = 0$, or $[\![\textit{K has a car}]\!]^{w,g} = 1$ and $[\![\textit{K brought his car}]\!]^{w,g} = 1$
     - 0, if $[\![\textit{K has a car}]\!]^{w,g} = 1$ and $[\![\textit{K brought his car}]\!]^{w,g} = 0$
     - #, if $[\![\textit{K has a car}]\!]^{w,g} = 1$ and $[\![\textit{K brought his car}]\!]^{w,g} = \#$

1. If Karlos has a car, Karlos brought his car.

   - *Material Conditional Rule*
     Given a material conditional, $[[\mathit{if}\ \phi]\ \psi]$, and an assignment, a, if $[\![\phi]\!]^a = 0$, then $[\![[[\mathit{if}\ \phi]\ \psi]]\!]^a = 1$. If $[\![\phi]\!]^a = 1$ and $[\![\psi]\!]^a$ is defined, then $[\![[[\mathit{if}\ \phi]\ \psi]]\!]^a = [\![\psi]\!]^a$. $[\![[[\mathit{if}\ \phi]\ \psi]]\!]^a$ is undefined if $[\![\phi]\!]^a$ is.
   - $[\![\textit{if Karlos has a car, Karlos brought his}_i\textit{ car}]\!]^{w,g}$
     - 1, if $[\![\textit{K has c}]\!]^{w,g} = 0$, or $[\![\textit{K has a car}]\!]^{w,g} = 1$ and $[\![\textit{K brought his car}]\!]^{w,g} = 1$
     - 0, if $[\![\textit{K has a car}]\!]^{w,g} = 1$ and $[\![\textit{K brought his car}]\!]^{w,g} = 0$
     - #, if $[\![\textit{K has a car}]\!]^{w,g} = 1$ and $[\![\textit{K brought his car}]\!]^{w,g} = \#$
     - $\rightsquigarrow$ no presupposition ☺

A proviso problem crops up in this setting, as well.

Compositional semantics: a rudimentary analysis of conditionals

A proviso problem crops up in this setting, as well.

1. If the airport is nearby, I'll pick my sister up when she lands.

A proviso problem crops up in this setting, as well.

1. If the airport is nearby, I'll pick my sister up when she lands.
   - $[\![\textit{airport nearby, pick up sister}]\!]^{w,g}$

A proviso problem crops up in this setting, as well.

1. If the airport is nearby, I'll pick my sister up when she lands.
   - $[\![\textit{airport nearby, pick up sister}]\!]^{w,g}$
     - 1, if $[\![\textit{airport nearby}]\!]^{w,g} = 0$, or $[\![\textit{airport nearby}]\!]^{w,g} = 1$ and $[\![\textit{pick up sister}]\!]^{w,g} = 1$

A proviso problem crops up in this setting, as well.

1. If the airport is nearby, I'll pick my sister up when she lands.
   - $[\![\textit{airport nearby, pick up sister}]\!]^{w,g}$
     - 1, if $[\![\textit{airport nearby}]\!]^{w,g} = 0$, or $[\![\textit{airport nearby}]\!]^{w,g} = 1$ and $[\![\textit{pick up sister}]\!]^{w,g} = 1$
     - 0, if $[\![\textit{airport nearby}]\!]^{w,g} = 1$ and $[\![\textit{pick up sister}]\!]^{w,g} = 0$

A proviso problem crops up in this setting, as well.

1. If the airport is nearby, I'll pick my sister up when she lands.
   - $[\![\textit{airport nearby, pick up sister}]\!]^{w,g}$
     - ★ 1, if $[\![\textit{airport nearby}]\!]^{w,g} = 0$, or $[\![\textit{airport nearby}]\!]^{w,g} = 1$ and $[\![\textit{pick up sister}]\!]^{w,g} = 1$
     - ★ 0, if $[\![\textit{airport nearby}]\!]^{w,g} = 1$ and $[\![\textit{pick up sister}]\!]^{w,g} = 0$
     - ★ #, if $[\![\textit{airport nearby}]\!]^{w,g} = 1$ and $[\![\textit{pick up sister}]\!]^{w,g} = \#$

A proviso problem crops up in this setting, as well.

1. If the airport is nearby, I'll pick my sister up when she lands.

   - $[\![$*airport nearby, pick up sister*$]\!]^{w,g}$
     - 1, if $[\![$*airport nearby*$]\!]^{w,g} = 0$, or $[\![$*airport nearby*$]\!]^{w,g} = 1$ and $[\![$*pick up sister*$]\!]^{w,g} = 1$
     - 0, if $[\![$*airport nearby*$]\!]^{w,g} = 1$ and $[\![$*pick up sister*$]\!]^{w,g} = 0$
     - #, if $[\![$*airport nearby*$]\!]^{w,g} = 1$ and $[\![$*pick up sister*$]\!]^{w,g} = \#$
     - $\rightsquigarrow$ *if the airport is nearby, I have a sister* ☹

Scope-taking

Used by Heim and Kratzer (1998) in the analysis of quantifiers

Used by Heim and Kratzer (1998) in the analysis of quantifiers

1. Every dog slept.

Scope-taking

Used by Heim and Kratzer (1998) in the analysis of quantifiers

1. Every dog slept.

   - Quantifier Raising: [[*every dog*]$_i$ [t_i *slept*]]

Scope-taking

Used by Heim and Kratzer (1998) in the analysis of quantifiers

1. Every dog slept.
   - Quantifier Raising: $[[\textit{every dog}]_i \ [t_i \ \textit{slept}]]$
     - Generally assumed to be clause-bounded

Used by Heim and Kratzer (1998) in the analysis of quantifiers

1. Every dog slept.
   - Quantifier Raising: $[[\textit{every dog}]_i\ [t_i\ \textit{slept}]]$
     - Generally assumed to be clause-bounded
   - *Predicate Abstraction* (Heim and Kratzer, 1998, p. 186, ex. 4)
     Let α be a branching node with daughters β and γ, where β dominates only a numerical index i. Then, for any variable assignment a,
     $[\![\alpha]\!]^a = \lambda x \in D.[\![\gamma]\!]^{a[x/i]}$.

Used by Heim and Kratzer (1998) in the analysis of quantifiers

1. Every dog slept.
   - Quantifier Raising: $[[\textit{every dog}]_i \ \ [t_i \ \ \textit{slept}]]$
     - Generally assumed to be clause-bounded
   - *Predicate Abstraction* (Heim and Kratzer, 1998, p. 186, ex. 4)
     Let α be a branching node with daughters β and γ, where β dominates only a numerical index i. Then, for any variable assignment a, $[\![\alpha]\!]^a = \lambda x \in D.[\![\gamma]\!]^{a[x/i]}$.
   - Predicate Abstraction and Functional Application:
     $[\![[[\textit{every dog}]_i \ \ [t_i \ \ \textit{slept}]]]\!]^g = [\![\textit{every dog}]\!]^g(\lambda x.[\![t_i \textit{slept}]\!]^{g[x/i]})$

These rules are all we need to salvage examples in which the presuppositions are too weak, provided:

These rules are all we need to salvage examples in which the presuppositions are too weak, provided:

- we allow the presupposition trigger to take scope over the conditional filter

These rules are all we need to salvage examples in which the presuppositions are too weak, provided:

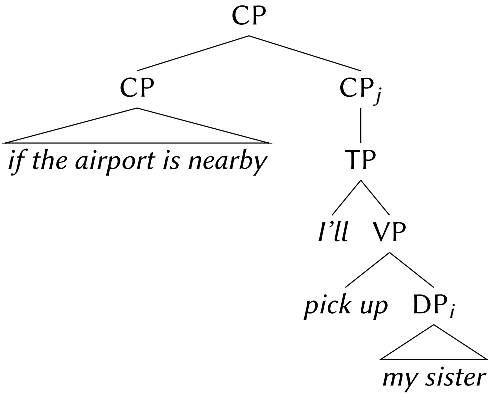
- we allow the presupposition trigger to take scope over the conditional filter
- This can be accomplished if we allow phrases of *any* category to take scope…

These rules are all we need to salvage examples in which the presuppositions are too weak, provided:

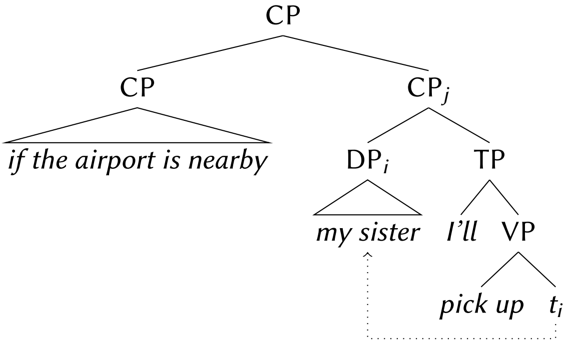
- we allow the presupposition trigger to take scope over the conditional filter
- This can be accomplished if we allow phrases of *any* category to take scope…
  - via *roll-up pied piping*, a.k.a., *cyclic scope-taking*.

These rules are all we need to salvage examples in which the presuppositions are too weak, provided:

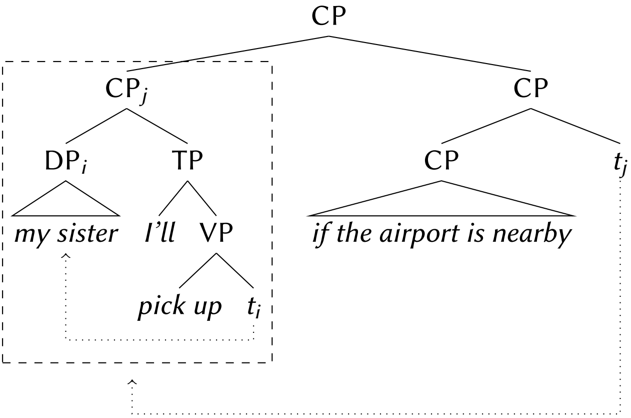
- we allow the presupposition trigger to take scope over the conditional filter
- This can be accomplished if we allow phrases of *any* category to take scope…
  - via *roll-up pied piping*, a.k.a., *cyclic scope-taking*.
  - Used by Charlow (2020) to account for the exceptional scoping properties of indefinites


CP
CP
CP_j
if the airport is nearby
TP
I'll
VP
pick up
DP_i
my sister

Back to conditionals: cyclic scope-taking



Back to conditionals: cyclic scope-taking



Back to conditionals: interpreting cyclic scope

- $[\![[[[\textit{my sister}]_i \textit{I'll pick up } t_i]_j[\textit{if the airport is nearby } t_j]]]\!]^{w,g}$

Back to conditionals: interpreting cyclic scope

- $[\![[[[\textit{my sister}]_i \textit{I'll pick up } t_i]_j[\textit{if the airport is nearby } t_j]]]\!]^{w,g}$
  - Predicate Abstraction and Functional Application: $(\lambda x.[\![\textit{if the airport is nearby } t_j]\!]^{w,g[x/j]})([\![\textit{I'll pick my sister up}]\!]^{w,g})$

Back to conditionals: interpreting cyclic scope

- $[\![[[[\textit{my sister}]_i \textit{I'll pick up } t_i]_j[\textit{if the airport is nearby } t_j]]]\!]^{w,g}$
  - Predicate Abstraction and Functional Application: $(\lambda x.[\![\textit{if the airport is nearby } t_j]\!]^{w,g[x/j]})([\![\textit{I'll pick my sister up}]\!]^{w,g})$
  - $\neq \#$ iff I have a sister

Back to conditionals: interpreting cyclic scope

- $[\![[[[\textit{my sister}]_i \textit{I'll pick up } t_i]_j [\textit{if the airport is nearby } t_j]]]\!]^{w,g}$
  - Predicate Abstraction and Functional Application: $(\lambda x.[\![\textit{if the airport is nearby } t_j]\!]^{w,g[x/j]})([\![\textit{I'll pick my sister up}]\!]^{w,g})$
  - $\neq \#$ iff I have a sister

Back to conditionals: interpreting cyclic scope

- $[\![[[[\textit{my sister}]_i \textit{I'll pick up } t_i]_j[\textit{if the airport is nearby } t_j]]]\!]^{w,g}$
  - Predicate Abstraction and Functional Application: $(\lambda x.[\![\textit{if the airport is nearby } t_j]\!]^{w,g[x/j]})([\![\textit{I'll pick my sister up}]\!]^{w,g})$
  - $\neq \#$ iff I have a sister

Conclusion: a freer definition of the "Quantifier Raising" rule allows us to circumvent the proviso problem, provided…

Back to conditionals: interpreting cyclic scope

- $[\![[[[\textit{my sister}]_i \textit{I'll pick up } t_i]_j[\textit{if the airport is nearby } t_j]]]\!]^{w,g}$
  - Predicate Abstraction and Functional Application:
    $(\lambda x.[\![\textit{if the airport is nearby } t_j]\!]^{w,g[x/j]})([\![\textit{I'll pick my sister up}]\!]^{w,g})$
  - $\neq \#$ iff I have a sister

Conclusion: a freer definition of the "Quantifier Raising" rule allows us to circumvent the proviso problem, provided…

- we have a non-compositional, static analysis of conditionals (and other filters)

A better analysis of filters?

The current analysis of conditionals involves a new, syncategorematic compositional rule (The Material Conditional Rule).

A better analysis of filters?

The current analysis of conditionals involves a new, syncategorematic compositional rule (The Material Conditional Rule).

We can do better by sophisticating the type system a little bit.

A better analysis of filters?

The current analysis of conditionals involves a new, syncategorematic compositional rule (The Material Conditional Rule).

We can do better by sophisticating the type system a little bit.

- Simple types (what we're using now):

$$\mathcal{T} ::= e \mid t \mid \mathcal{T} \rightarrow \mathcal{T}$$

A better analysis of filters?

The current analysis of conditionals involves a new, syncategorematic compositional rule (The Material Conditional Rule).

We can do better by sophisticating the type system a little bit.

- Simple types (what we're using now):

$$\mathcal{T} ::= e \mid t \mid \mathcal{T} \to \mathcal{T}$$

- We'll add "Maybe" types:

$$\mathcal{T} ::= e \mid t \mid \mathcal{T} \to \mathcal{T} \mid \mathcal{T}_{\#}$$

A better analysis of filters?

The current analysis of conditionals involves a new, syncategorematic compositional rule (The Material Conditional Rule).

We can do better by sophisticating the type system a little bit.

- Simple types (what we're using now):

$$\mathcal{T} ::= e \mid t \mid \mathcal{T} \rightarrow \mathcal{T}$$

- We'll add "Maybe" types:

$$\mathcal{T} ::= e \mid t \mid \mathcal{T} \rightarrow \mathcal{T} \mid \mathcal{T}_{\#}$$

  - E.g., the type $e_{\#}$ is that of something which is either an individual (e.g., Karlos) or undefined (#).

A better analysis of filters?

The current analysis of conditionals involves a new, syncategorematic compositional rule (The Material Conditional Rule).

We can do better by sophisticating the type system a little bit.

- Simple types (what we're using now):

$$\mathcal{T} ::= e \mid t \mid \mathcal{T} \to \mathcal{T}$$

- We'll add "Maybe" types:

$$\mathcal{T} ::= e \mid t \mid \mathcal{T} \to \mathcal{T} \mid \mathcal{T}_{\#}$$

  - E.g., the type $e_{\#}$ is that of something which is either an individual (e.g., Karlos) or undefined (#).
  - This move allows us to treat partial functions as total; e.g., a partial function of type $e \to t$ is now a total function of type $e \to t_{\#}$ that maps the part of its domain on which it is not defined to #.

A better analysis of filters?

A meaning for *if*:

- $$[\![if]\!]^{w,g} = \lambda p^{t_\#}, q^{t_\#}. \begin{cases} 1 & p = 0 \\ 1 & p = 1 \text{ and } q = 1 \\ 0 & p = 1 \text{ and } q = 0 \\ \# & p = \# \\ \# & p = 1 \text{ and } q = \# \end{cases}$$

A better analysis of filters?

A meaning for *if*:

- $[\![if]\!]^{w,g} = \lambda p^{t\#}, q^{t\#}. \begin{cases} 1 & p = 0 \\ 1 & p = 1 \text{ and } q = 1 \\ 0 & p = 1 \text{ and } q = 0 \\ \# & p = \# \\ \# & p = 1 \text{ and } q = \# \end{cases}$

Other meanings can remain unmodified from the simply typed setting, except for those for presupposition triggers.

A better analysis of filters?

A meaning for *if*:

- $$[\![if]\!]^{w,g} = \lambda p^{t_\#}, q^{t_\#}. \begin{cases} 1 & p = 0 \\ 1 & p = 1 \text{ and } q = 1 \\ 0 & p = 1 \text{ and } q = 0 \\ \# & p = \# \\ \# & p = 1 \text{ and } q = \# \end{cases}$$

Other meanings can remain unmodified from the simply typed setting, except for those for presupposition triggers.

- $[\![his_i\ car]\!]^{w,g}$ (now of type $e_\#$)

A meaning for *if*:

- $$[\![\mathit{if}]\!]^{w,g} = \lambda p^{t_\#}, q^{t_\#}. \begin{cases} 1 & p = 0 \\ 1 & p = 1 \text{ and } q = 1 \\ 0 & p = 1 \text{ and } q = 0 \\ \# & p = \# \\ \# & p = 1 \text{ and } q = \# \end{cases}$$

Other meanings can remain unmodified from the simply typed setting, except for those for presupposition triggers.

- $[\![\mathit{his}_i\ \mathit{car}]\!]^{w,g}$ (now of type $e_\#$)
  - the unique car of $g(i)$ in w if one exists; otherwise, #

Two type shifts

In addition, we will introduce two type shifts, which allow for the smooth integration of simple types and *Maybe* types.

Two type shifts

In addition, we will introduce two type shifts, which allow for the smooth integration of simple types and Maybe types.

- $(\cdot)^{\eta} : \alpha \to \alpha_{\#}$ ('return')

Two type shifts

In addition, we will introduce two type shifts, which allow for the smooth integration of simple types and Maybe types.

- $(\cdot)^{\eta} : \alpha \rightarrow \alpha_{\#}$ ('return')
- $a^{\eta} = a$

Two type shifts

In addition, we will introduce two type shifts, which allow for the smooth integration of simple types and Maybe types.

- $(\cdot)^\eta : \alpha \to \alpha_\#$ ('return')
- $a^\eta = a$

Two type shifts

In addition, we will introduce two type shifts, which allow for the smooth integration of simple types and Maybe types.

- $(\cdot)^{\eta} : \alpha \to \alpha_{\#}$ ('return')
- $a^{\eta} = a$

- $(\cdot)^{\gg=} : \alpha_{\#} \to (\alpha \to \beta_{\#}) \to \beta_{\#}$ ('bind')

Two type shifts

In addition, we will introduce two type shifts, which allow for the smooth integration of simple types and Maybe types.

- $(\cdot)^{\eta} : \alpha \rightarrow \alpha_{\#}$ ('return')
- $a^{\eta} = a$

- $(\cdot)^{\gg=} : \alpha_{\#} \rightarrow (\alpha \rightarrow \beta_{\#}) \rightarrow \beta_{\#}$ ('bind')
- $\#^{\gg=} = \lambda f^{\alpha \rightarrow \beta_{\#}}.\#$

Two type shifts

In addition, we will introduce two type shifts, which allow for the smooth integration of simple types and Maybe types.

- $(\cdot)^{\eta} : \alpha \to \alpha_{\#}$ ('return')
- $a^{\eta} = a$

- $(\cdot)^{\gg=} : \alpha_{\#} \to (\alpha \to \beta_{\#}) \to \beta_{\#}$ ('bind')
- $\#^{\gg=} = \lambda f^{\alpha \to \beta_{\#}}.\#$
- $a^{\gg=} = \lambda f^{\alpha \to \beta_{\#}}.f(a)$

Two type shifts

In addition, we will introduce two type shifts, which allow for the smooth integration of simple types and Maybe types.

- $(\cdot)^{\eta} : \alpha \to \alpha_{\#}$ (‘return’)
- $a^{\eta} = a$

- $(\cdot)^{\gg=} : \alpha_{\#} \to (\alpha \to \beta_{\#}) \to \beta_{\#}$ (‘bind’)
- $\#^{\gg=} = \lambda f^{\alpha \to \beta_{\#}}.\#$
- $a^{\gg=} = \lambda f^{\alpha \to \beta_{\#}}.f(a)$

Two type shifts

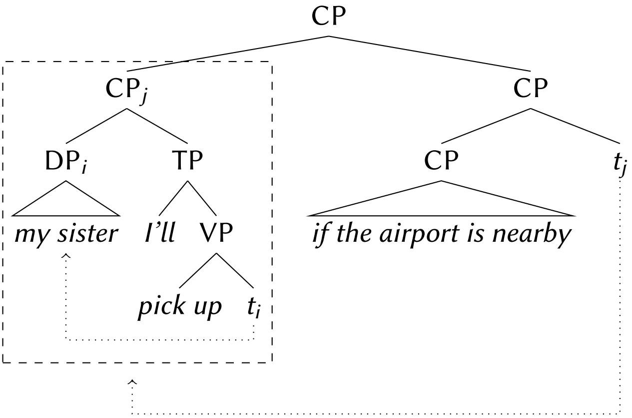
In addition, we will introduce two type shifts, which allow for the smooth integration of simple types and Maybe types.

- $(\cdot)^{\eta} : \alpha \to \alpha_{\#}$ ('return')
- $a^{\eta} = a$

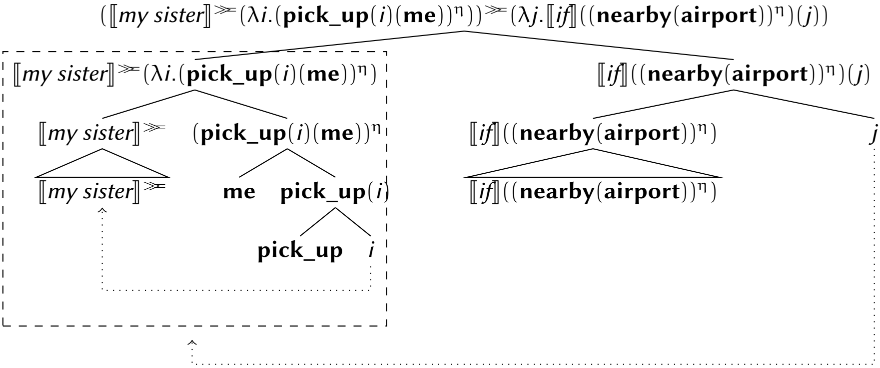
- $(\cdot)^{\gg=} : \alpha_{\#} \to (\alpha \to \beta_{\#}) \to \beta_{\#}$ ('bind')
- $\#^{\gg=} = \lambda f^{\alpha \to \beta_{\#}}.\#$
- $a^{\gg=} = \lambda f^{\alpha \to \beta_{\#}}.f(a)$

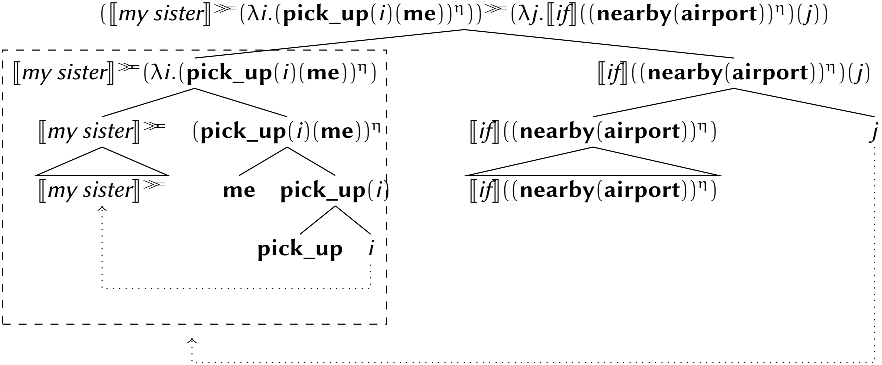
(Together, $(\cdot)^{\eta}$ and $(\cdot)^{\gg=}$ constitute a monad.)

Scoping out the consequent



Scoping out the consequent





1 if I have a sister, and I pick her up if the airport is nearby
0 if I have a sister, the airport is nearby, and I don't pick her up
\# if I don't have a sister

Filtering…

For examples like

1. If Karlos has a car, he brought his car.

we simply don't scope the consequent clause above the filter, allowing presupposition satisfaction to go through.

Summary

- Allowing a presupposition trigger to take scope past a filter causes its presupposition to project, even with an interpretation strategy as simple as that of Heim and Kratzer (1998).

Summary

- Allowing a presupposition trigger to take scope past a filter causes its presupposition to project, even with an interpretation strategy as simple as that of Heim and Kratzer (1998).
- Introducing Maybe types allows the system to be fully compositional.

Summary

- Allowing a presupposition trigger to take scope past a filter causes its presupposition to project, even with an interpretation strategy as simple as that of Heim and Kratzer (1998).
- Introducing Maybe types allows the system to be fully compositional.
- The foregoing analysis of filters is static, but making it dynamic (more straightforwardly in line with Heim (1983)) is a matter of further enriching the types (as done by, e.g., Rothschild (2011)).

We are here

Propositional attitude provisos

It has also been noted that something a proviso problem arises with propositional attitude verbs, as well.

It has also been noted that something a proviso problem arises with propositional attitude verbs, as well.

1. Ashley believes her car is in the parking lot.

Propositional attitude provisos

It has also been noted that something a proviso problem arises with propositional attitude verbs, as well.

1. Ashley believes her car is in the parking lot.
   - $\rightsquigarrow$ *Ashley has a car*

Propositional attitude provisos

It has also been noted that something a proviso problem arises with propositional attitude verbs, as well.

1. Ashley believes her car is in the parking lot.
   - $\rightsquigarrow$ *Ashley has a car*
   - (the *de re* reading)

It has also been noted that something a proviso problem arises with propositional attitude verbs, as well.

1. Ashley believes her car is in the parking lot.
   - $\rightsquigarrow$ *Ashley has a car*
   - (the *de re* reading)

Propositional attitude provisos

It has also been noted that something a proviso problem arises with propositional attitude verbs, as well.

1. Ashley believes her car is in the parking lot.
   - $\rightsquigarrow$ *Ashley has a car*
   - (the *de re* reading)

- Satisfaction accounts of such examples generally predict that they have propositional attitude presuppositions (e.g., Heim 1992).

It has also been noted that something a proviso problem arises with propositional attitude verbs, as well.

1. Ashley believes her car is in the parking lot.
   - $\rightsquigarrow$ *Ashley has a car*
   - (the *de re* reading)

- Satisfaction accounts of such examples generally predict that they have propositional attitude presuppositions (e.g., Heim 1992).
  - For (1), *Ashley believes she has a car.*

Propositional attitude provisos

It has also been noted that something a proviso problem arises with propositional attitude verbs, as well.

1. Ashley believes her car is in the parking lot.
    - $\rightsquigarrow$ *Ashley has a car*
    - (the *de re* reading)

- Satisfaction accounts of such examples generally predict that they have propositional attitude presuppositions (e.g., Heim 1992).
    - For (1), *Ashley believes she has a car.*
    - (the *de dicto* reading)

To analyze such examples, we can add a new atomic s type to make our system intensional.

Incorporating intensionality

To analyze such examples, we can add a new atomic s type to make our system intensional.

- New types: $\mathcal{T} ::= e \mid s \mid t \mid \mathcal{T} \rightarrow \mathcal{T} \mid \mathcal{T}_{\#}$

Incorporating intensionality

To analyze such examples, we can add a new atomic s type to make our system intensional.

- New types: $\mathcal{T} ::= e \mid s \mid t \mid \mathcal{T} \to \mathcal{T} \mid \mathcal{T}_{\#}$

Incorporating intensionality

To analyze such examples, we can add a new atomic s type to make our system intensional.

- New types: $\mathcal{T} ::= e \mid s \mid t \mid \mathcal{T} \rightarrow \mathcal{T} \mid \mathcal{T}_{\#}$

Propositions, in this setting, are functions of type $s \rightarrow t_{\#}$.

To analyze such examples, we can add a new atomic s type to make our system intensional.

- New types: $\mathcal{T} ::= e \mid s \mid t \mid \mathcal{T} \rightarrow \mathcal{T} \mid \mathcal{T}_{\#}$

Propositions, in this setting, are functions of type $s \rightarrow t_{\#}$.

- $[\![\textit{Karlos brought his car}]\!] =$

$$\lambda w^s. \begin{cases} 1 & \text{Karlos has a car and brought it} \\ 0 & \text{Karlos has a car and didn't bring it} \\ \# & \text{Karlos doesn't have a car} \end{cases}$$

Propositional attitude verbs

- $[\![\mathit{believes}]\!] = \lambda p^{s \to t_\#}, x^e, w^s.\forall w'^s : \mathbf{acc}_{w,x}(w') \Rightarrow p(w')$

$\Rightarrow$	$\top$	$\bot$	$\#$
$\top$	$\top$	$\bot$	$\#$
$\bot$	$\top$	$\top$	$\top$
$\#$	$\#$	$\#$	$\#$

$\{[\![\phi]\!]_{\mathcal{M},g'} \mid g[x]g'\}$	$[\![\ulcorner \forall x : \phi \urcorner]\!]_{\mathcal{M},g}$
$\{\top\}$	$\top$
$\{\bot\}$	$\bot$
$\{\#\}$	$\#$
$\{\top, \bot\}$	$\bot$
$\{\top, \#\}$	$\#$
$\{\bot, \#\}$	$\#$
$\{\top, \bot, \#\}$	$\#$

- $[\![believes]\!] = \lambda p^{s \to t_{\#}}, x^e, w^s . \forall w'^s : \mathbf{acc}_{w,x}(w') \Rightarrow p(w')$

$\Rightarrow$	$\top$	$\bot$	$\#$
$\top$	$\top$	$\bot$	$\#$
$\bot$	$\top$	$\top$	$\top$
#	#	#	#

$\{[\![\phi]\!]_{\mathcal{M},g'} \mid g[x]g'\}$	$[\![\ulcorner \forall x : \phi \urcorner]\!]_{\mathcal{M},g}$
$\{\top\}$	$\top$
$\{\bot\}$	$\bot$
$\{\#\}$	$\#$
$\{\top, \bot\}$	$\bot$
$\{\top, \#\}$	$\#$
$\{\bot, \#\}$	$\#$
$\{\top, \bot, \#\}$	$\#$

- Presupposition failure results if the presupposition fails to hold at some accessible world. (Inaccessible worlds don't matter.)

Propositional attitude filtering

1. Ashley believes her car is in the parking lot.

Propositional attitude filtering

1. Ashley believes her car is in the parking lot.
   - Taking the meaning of the embedded clause for granted:

Propositional attitude filtering

1. Ashley believes her car is in the parking lot.

   - Taking the meaning of the embedded clause for granted:
   - $\lambda w^s.\forall w'^s : \mathbf{acc}_{w,\text{Ashley}}(w') \Rightarrow$
     $$\begin{cases} 1 & \text{Ashley has a car, and it's in the lot in} w' \\ 0 & \text{Ashley has a car, and it's not in the lot in} \mathit{in} w' \\ \# & \text{Ashley doesn't have a car in} w' \end{cases}$$

1. Ashley believes her car is in the parking lot.

   - Taking the meaning of the embedded clause for granted:
   - $\lambda w^s.\forall w'^s : \mathbf{acc}_{w,\text{Ashley}}(w') \Rightarrow$
     $$\begin{cases} 1 & \text{Ashley has a car, and it's in the lot in} w' \\ 0 & \text{Ashley has a car, and it's not in the lot in} \mathit{in} w' \\ \# & \text{Ashley doesn't have a car in} w' \end{cases}$$
   - Defined at any world w such that
     $\forall w'^s : \mathbf{acc}_{w,\text{Ashley}} \Rightarrow$ Ashley has a car inw'

Out-scoping propositional attitude filters

What happens if we allow the embedded clause (and thus its presupposition trigger) to take scope?

What happens if we allow the embedded clause (and thus its presupposition trigger) to take scope?

- To do so, we need to upgrade our type-shifts to the intensional setting.

What happens if we allow the embedded clause (and thus its presupposition trigger) to take scope?

- To do so, we need to upgrade our type-shifts to the intensional setting.

What happens if we allow the embedded clause (and thus its presupposition trigger) to take scope?

- To do so, we need to upgrade our type-shifts to the intensional setting.
- $(\cdot)^{\eta} : \alpha \to s \to \alpha_{\#}$ (‘return’)

What happens if we allow the embedded clause (and thus its presupposition trigger) to take scope?

- To do so, we need to upgrade our type-shifts to the intensional setting.

- $(\cdot)^\eta : \alpha \rightarrow s \rightarrow \alpha_\#$ (‘return’)
- $a^\eta = \lambda w^s.a$

What happens if we allow the embedded clause (and thus its presupposition trigger) to take scope?

- To do so, we need to upgrade our type-shifts to the intensional setting.

- $(\cdot)^{\eta} : \alpha \to s \to \alpha_{\#}$ ('return')
- $a^{\eta} = \lambda w^{s}.a$

What happens if we allow the embedded clause (and thus its presupposition trigger) to take scope?

- To do so, we need to upgrade our type-shifts to the intensional setting.

- $(\cdot)^{\eta} : \alpha \to s \to \alpha_{\#}$ ('return')
- $a^{\eta} = \lambda w^{s}.a$

- $(\cdot)^{\ggcurly} : (s \to \alpha_{\#}) \to (\alpha \to s \to \beta_{\#}) \to s \to \beta_{\#}$ ('bind')

Out-scoping propositional attitude filters

What happens if we allow the embedded clause (and thus its presupposition trigger) to take scope?

- To do so, we need to upgrade our type-shifts to the intensional setting.

- $(\cdot)^{\eta} : \alpha \rightarrow s \rightarrow \alpha_{\#}$ ('return')
- $a^{\eta} = \lambda w^{s}.a$

- $(\cdot)^{\gg=} : (s \rightarrow \alpha_{\#}) \rightarrow (\alpha \rightarrow s \rightarrow \beta_{\#}) \rightarrow s \rightarrow \beta_{\#}$ ('bind')
- $m^{\gg=} = \lambda f^{\alpha \rightarrow s \rightarrow \beta_{\#}}, w^{s}. \begin{cases} \# & m(w) = \# \\ f(a)(w) & m(w) = a \end{cases}$

Out-scoping propositional attitude filters

$$(\llbracket \textit{her car} \rrbracket^{\ggcurly} (\lambda i^e.(\lambda w^s.\mathbf{in}(\mathbf{lot})(i)(w))^\eta))^{\ggcurly}(\lambda j^{s\to t_\#}, w^s.\forall w'^s : \mathbf{acc}_{w,\text{Ashley}}(w') \Rightarrow j(w'))$$

$$\llbracket \textit{her car} \rrbracket^{\ggcurly} (\lambda i^e.(\lambda w^s.\mathbf{in}(\mathbf{lot})(i)(w))^\eta) \qquad \lambda j^{s\to t_\#}, w^s.\forall w'^s : \mathbf{acc}_{w,\text{Ashley}}(w') \Rightarrow j(w')$$

$$\llbracket \textit{her car} \rrbracket^{\ggcurly} \quad (\lambda w^s.\mathbf{in}(\mathbf{lot})(i)(w))^\eta \qquad \lambda w^s.\forall w'^s : \mathbf{acc}_{w,\text{Ashley}}(w') \Rightarrow j(w')$$

$$\llbracket \textit{her car} \rrbracket^{\ggcurly} \qquad i \quad \mathbf{in(lot)}$$

$$\mathbf{in} \quad \mathbf{lot}$$

Out-scoping propositional attitude filters

$$(\llbracket \textit{her car} \rrbracket^{\gg=}(\lambda i^e.(\lambda w^s.\mathbf{in}(\mathbf{lot})(i)(w))^\eta))^{\gg=}(\lambda j^{s\to t_\#}, w^s.\forall w'^s : \mathbf{acc}_{w,\text{Ashley}}(w') \Rightarrow j(w'))$$

$$\llbracket \textit{her car} \rrbracket^{\gg=}(\lambda i^e.(\lambda w^s.\mathbf{in}(\mathbf{lot})(i)(w))^\eta)$$

$$\lambda j^{s\to t_\#}, w^s.\forall w'^s : \mathbf{acc}_{w,\text{Ashley}}(w') \Rightarrow j(w')$$

$$\llbracket \textit{her car} \rrbracket^{\gg=} \qquad (\lambda w^s.\mathbf{in}(\mathbf{lot})(i)(w))^\eta$$

$$\lambda w^s.\forall w'^s : \mathbf{acc}_{w,\text{Ashley}}(w') \Rightarrow j(w')$$

$$\llbracket \textit{her car} \rrbracket^{\gg=} \qquad i \qquad \mathbf{in}(\mathbf{lot})$$

$$\mathbf{in} \qquad \mathbf{lot}$$

- The result evaluates to:

Out-scoping propositional attitude filters

$$(\llbracket her\ car \rrbracket^{\ggcurly}(\lambda i^e.(\lambda w^s.\mathbf{in}(\mathbf{lot})(i)(w))^{\eta}))^{\ggcurly}(\lambda j^{s \to t_{\#}}, w^s.\forall w'^s : \mathbf{acc}_{w,\text{Ashley}}(w') \Rightarrow j(w'))$$

$$\llbracket her\ car \rrbracket^{\ggcurly}(\lambda i^e.(\lambda w^s.\mathbf{in}(\mathbf{lot})(i)(w))^{\eta})$$

$$\llbracket her\ car \rrbracket^{\ggcurly} \qquad (\lambda w^s.\mathbf{in}(\mathbf{lot})(i)(w))^{\eta}$$

$$\llbracket her\ car \rrbracket^{\ggcurly} \qquad i \qquad \mathbf{in}(\mathbf{lot})$$

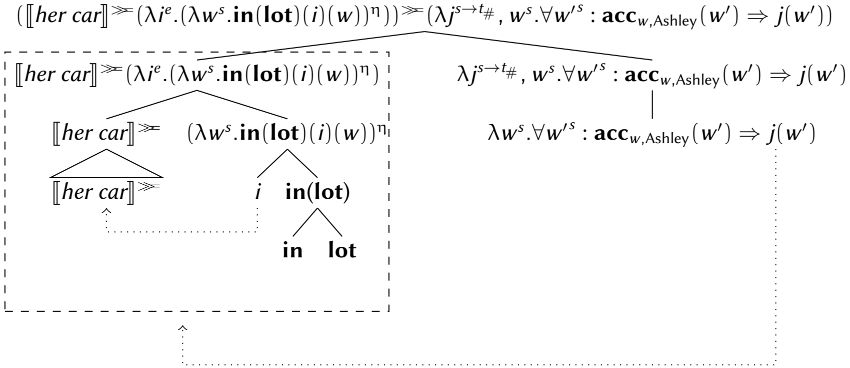
$$\mathbf{in} \qquad \mathbf{lot}$$

$$\lambda j^{s \to t_{\#}}, w^s.\forall w'^s : \mathbf{acc}_{w,\text{Ashley}}(w') \Rightarrow j(w')$$

$$\lambda w^s.\forall w'^s : \mathbf{acc}_{w,\text{Ashley}}(w') \Rightarrow j(w')$$

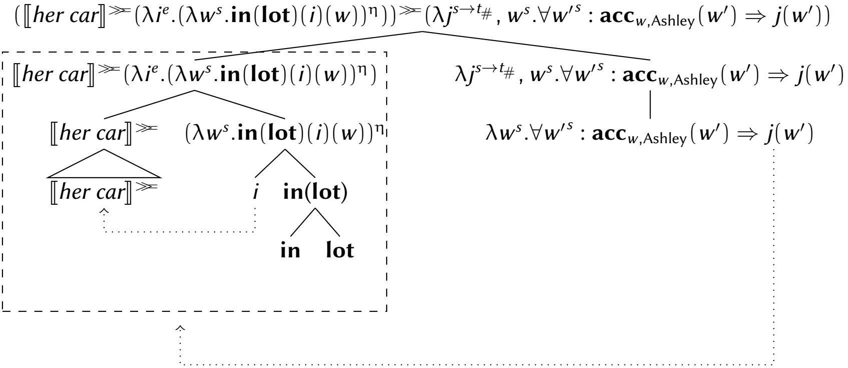
- The result evaluates to:
  - $\llbracket her\ car \rrbracket^{\ggcurly}(\lambda i^e, w^s.\forall w'^s : \mathbf{acc}_{w,\text{Ashley}}(w') \Rightarrow \mathbf{in}(\mathbf{lot})(i)(w'))$

Out-scoping propositional attitude filters



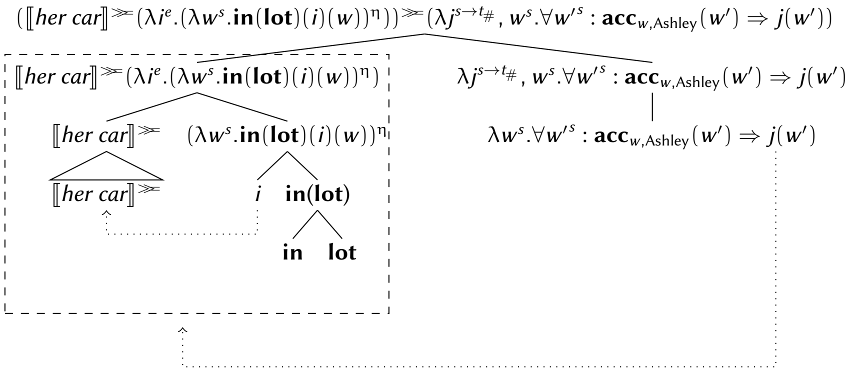
- The result evaluates to:
  - $\llbracket her\ car \rrbracket^{\gg=}(\lambda i^e, w^s.\forall w'^s : \mathbf{acc}_{w,\text{Ashley}}(w') \Rightarrow \mathbf{in}(\mathbf{lot})(i)(w'))$
  - For any world w:

Out-scoping propositional attitude filters



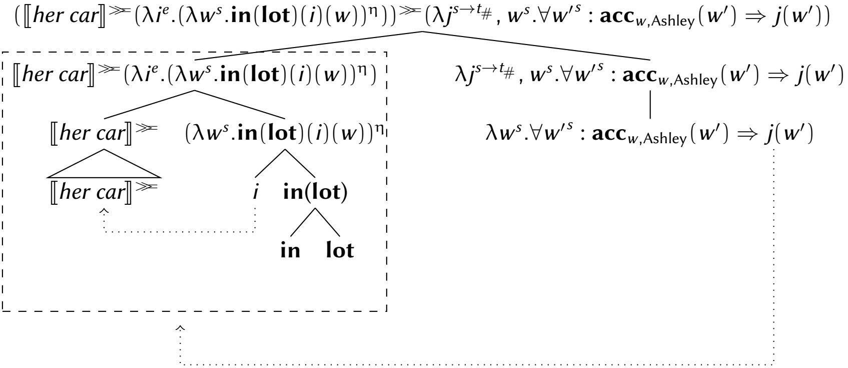
- The result evaluates to:
  - $\llbracket her\ car \rrbracket^{\gg=}(\lambda i^e, w^s.\forall w'^s : \mathbf{acc}_{w,\text{Ashley}}(w') \Rightarrow \mathbf{in}(\mathbf{lot})(i)(w'))$
  - For any world w:
    - 1, if Ashley has a car in w and believes in w that it's in the parking lot

Out-scoping propositional attitude filters



- The result evaluates to:
  - $\llbracket her\ car \rrbracket^{\gg=}(\lambda i^e, w^s.\forall w'^s : \mathbf{acc}_{w,\text{Ashley}}(w') \Rightarrow \mathbf{in}(\mathbf{lot})(i)(w'))$
  - For any world w:
    - 1, if Ashley has a car in w and believes in w that it's in the parking lot
    - 0, if Ashley has a car in w and believes in w that it's not in the parking lot

Out-scoping propositional attitude filters



- The result evaluates to:
  - $[\![\mathit{her\ car}]\!]^{\gg\!=}(\lambda i^e, w^s.\forall w'^s : \mathbf{acc}_{w,\text{Ashley}}(w') \Rightarrow \mathbf{in}(\mathbf{lot})(i)(w'))$
  - For any world w:
    - 1, if Ashley has a car in w and believes in w that it's in the parking lot
    - 0, if Ashley has a car in w and believes in w that it's not in the parking lot
    - #, if Ashley doesn't have a car in w

Conclusion

- The satisfaction account of presupposition projection allows us to describe projection using nothing more than the tools semanticists are used to (and need anyway).

Conclusion

- The satisfaction account of presupposition projection allows us to describe projection using nothing more than the tools semanticists are used to (and need anyway).
  - truth conditions (sets of worlds)

Conclusion

- The satisfaction account of presupposition projection allows us to describe projection using nothing more than the tools semanticists are used to (and need anyway).
  - truth conditions (sets of worlds)
  - compositional view of the syntax-semantics interface

Conclusion

- The satisfaction account of presupposition projection allows us to describe projection using nothing more than the tools semanticists are used to (and need anyway).
  - truth conditions (sets of worlds)
  - compositional view of the syntax-semantics interface
- These tools, with minor extensions, allow us to describe a rich array of projection behaviors: problems of automatic filtration are overcome by allowing presupposition triggers to take scope.

Conclusion

- The satisfaction account of presupposition projection allows us to describe projection using nothing more than the tools semanticists are used to (and need anyway).
  - truth conditions (sets of worlds)
  - compositional view of the syntax-semantics interface
- These tools, with minor extensions, allow us to describe a rich array of projection behaviors: problems of automatic filtration are overcome by allowing presupposition triggers to take scope.
- Could a scopal-mechanism be incorporated into pragmatic alternatives to the satisfaction account (Schlenker, 2008, 2009, 2010)?

References

Charlow, Simon. 2020. The scope of alternatives: indefiniteness and islands. *Linguistics and Philosophy* 43:427–472. https://doi.org/10.1007/s10988-019-09278-3.

Chierchia, Gennaro, and Sally McConnell-Ginet. 2000. *Meaning and Grammar*. Cambridge: MIT Press, second edition.

von Fintel, Kai. 2004. Would you Believe It? The King of France is Back! (Presuppositions and Truth-Value Intuitions). In *Descriptions and Beyond*, ed. Marga Reimer and Anne Bezuidenhout. New York: Oxford University Press.

von Fintel, Kai. 2008. What is presupposition accommodation, again? *Philosophical Perspectives* 22:137–170. https://doi.org/10.1111/j.1520-8583.2008.00144.x.

Geurts, Bart. 1996. Local Satisfaction Guaranteed: A Presupposition Theory and Its Problems. *Linguistics and Philosophy* 19:259–294. https://doi.org/10.1007/BF00628201.

References

Heim, Irene. 1983. On the Projection Problem for Presuppositions. In *Proceedings of the 2nd West Coast Conference on Formal Linguistics*, ed. Michael D. Barlow, Daniel P. Flickinger, and Michael Westcoat, 114–125. Stanford: Stanford University Press.

Heim, Irene. 1992. Presupposition Projection and the Semantics of Attitude Verbs. *Journal of Semantics* 9:183–221. `https://doi.org/10.1093/jos/9.3.183`.

Heim, Irene, and Angelika Kratzer. 1998. *Semantics in Generative Grammar*. Malden: Blackwell.

Langendoen, Donald T., and Harris B. Savin. 1971. The projection problem for presuppositions. In *Studies in Linguistic Semantics*, ed. Charles J. Fillmore and Donald T. Langendoen, 54–60. Holt, Rinehart and Winston.

Rothschild, Daniel. 2011. Explaining presupposition projection with dynamic semantics. *Semantics and Pragmatics* 4:1–43. `https://doi.org/10.3765/sp.4.3`.

References

Schlenker, Philippe. 2008. Be Articulate: A pragmatic theory of presupposition projection. *Theoretical Linguistics* 34:157–212. http://www.degruyter.com/view/journals/thli/34/3/article-p157.xml, publisher: De Gruyter Mouton Section: Theoretical Linguistics.

Schlenker, Philippe. 2009. Local Contexts. *Semantics and Pragmatics* 2:3–1–78. https://semprag.org/index.php/sp/article/view/sp.2.3, number: 0.

Schlenker, Philippe. 2010. Presuppositions and Local Contexts. *Mind* 119:377–391. https://doi.org/10.1093/mind/fzq032.