1 Discourse referents

Karttunen (1973)

Consider a device designed to read a text in some natural language, interpret it, and store the content in some manner, say, for the purpose of being able to answer questions about it. To accomplish this task, the machine will have to fulfill at least the following basic requirement. It has to be able to build a file that consists of records of all the individuals, that is, events, objects, etc., mentioned in the text and, for each individual, record whatever is said about it. Of course, for the time being at least, it seems that such a text interpreter is not a practical idea, but this should not discourage us from studying in abstract what kind of capabilities the machine would have to possess, provided that our study provides us with some insight into natural language in general.

- We don't always know which objects in the world we are talking about.
- Discourse referents can change and disappear. Their lives are governed by the conventions of language and how we use that language.
- Dynamic logics like that of Groenendijk (1999) concern themselves almost exclusively with information about the world. Other examples of such systems include Heim 1992 and Gunlogson 2001, which are inspired by Stalnaker's (1978) view of context, common ground, and update.
- Other dynamic logics focus on information about the discourse. In such models, we typically have an extensional model off to the side, but it is there only to support the discourse modeling techniques. Important systems of this sort include Discourse Representation Theory (DRT; Kamp 1981; Kamp and Reyle 1993), File-Change Semantics (Heim 1982, 1983), and Dynamic Predicate Logic (DPL; Groenendijk and Stokhof 1991). All of these theories trace back to Karttunen 1973.
- Attempts to model both world information and discourse information together: Chierchia 1995; Groenendijk et al. 1996; Asher and Lascarides 2003.
- This handout focuses on Dekker's (1994) Predicate Logic with Anaphora (PLA). It offers a kind of minimal dynamification of first-order logic. I think it's a good first step towards the more complex system of Heim (1983).
- Online implementation:
  http://www.christopherpotts.net/ling/implementations/pla/
2 Terms

The single square brackets interpret terms. The parameters:

- \( M = \langle F, D \rangle \) is an extensional model, with \( F \) the interpretation function and \( D \) the domain of entities.
- \( s \) is the current info state.
- \( e \) is a specific case in \( s \).
- \( g \) is the assignment function.

2.1 Constants

Values given by the interpretation function \( F \). So the only parameter they require is \( M \).

2.2 Variables

Values given by the assignment function \( g \).

2.3 Pronouns

Their values are defined in terms of the information state, so they require \( s \) and \( e \). If \( e \) is not specified, we get a column.

\[ [p_i]_{M,s,e,g} = e_{N_s-i} \text{ for all pronouns } p_i \text{ and } e \text{ and } s \text{ such that } e \in s \text{ and } N_s > i \]

Indices  Where \( N_s \), the length of all the cases in \( s \), is 5

- \([p_0]_{M,s,e,g} = e_{5-0} \), which is the final element in \( e \)
- \([p_1]_{M,s,e,g} = e_{5-1} \), which is the next to last element in \( e \)
- \([p_4]_{M,s,e,g} = e_{5-4} \), which is the first element in \( e \)
- \([p_5]_{M,s,e,g} = e_{5-5} \), which is undefined for \( e \)
3 Atomic formulae

\[ s[R_{t_1 \ldots t_n}]_{M, g} = \{ e \in s \mid ([t_1]_{M, s, e, g} \ldots [t_n]_{M, s, e, g}) \in F(R) \text{ (if } N_s > I_{t_1 \ldots t_n}) \} \]

(The parenthetical on the end specifies that the meaning is undefined if any pronoun has an undefined index for the state \( s \).)

3.1 Without pronouns

With no pronouns, we just evaluate the formula classically. If it is classically true, we keep all the cases. If it is classically false, we eliminate all the cases.

<table>
<thead>
<tr>
<th>Example</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00</td>
<td>01 01</td>
</tr>
<tr>
<td>01 [ [ (odd 1) ] ] _g 10 10</td>
<td></td>
</tr>
<tr>
<td>10 11</td>
<td></td>
</tr>
<tr>
<td>g(x) = 1</td>
<td></td>
</tr>
<tr>
<td>00 00</td>
<td>01 01</td>
</tr>
<tr>
<td>01 [ [ (odd x) ] ] _g 10 10</td>
<td></td>
</tr>
<tr>
<td>10 11</td>
<td></td>
</tr>
<tr>
<td>g(x) = 1</td>
<td></td>
</tr>
</tbody>
</table>

3.2 With pronouns

Now the updates can selectively eliminate cases.

<table>
<thead>
<tr>
<th>Example</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>00 00</td>
<td>00 00</td>
</tr>
<tr>
<td>01 [ [ (even p0) ] ] _g 10 10</td>
<td></td>
</tr>
<tr>
<td>10 11</td>
<td></td>
</tr>
<tr>
<td>g(x) = 1</td>
<td></td>
</tr>
<tr>
<td>01 01</td>
<td>01 01</td>
</tr>
<tr>
<td>01 [ [ (even p1) ] ] _g 10 10</td>
<td></td>
</tr>
<tr>
<td>10 11</td>
<td></td>
</tr>
<tr>
<td>g(x) = 1</td>
<td></td>
</tr>
</tbody>
</table>
4 Negation

With negation, we keep only those cases that cannot be extended into new cases that make the scope formula true:

\[ s[[-\varphi]]_{M,g} = \{ e \in s \mid \neg \exists e' : e \leq e' \land e' \in s[\varphi]_{M,g} \} \]

With \( s[[-\varphi]]_{M,g} \), we always get the complement of \( s[\varphi]_{M,g} \) back. So this is just like a classical negation.

<table>
<thead>
<tr>
<th>Example</th>
<th></th>
<th>Example</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>--</td>
<td>00</td>
<td>00</td>
</tr>
<tr>
<td>01 [[~(odd 1)]_g</td>
<td>--</td>
<td>01 [[~(even 1)]_g</td>
<td>01</td>
</tr>
<tr>
<td>10</td>
<td>--</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>11</td>
<td>--</td>
<td>11</td>
<td>11</td>
</tr>
</tbody>
</table>

Example \( \sim(p0 = 1) \)

<table>
<thead>
<tr>
<th>( S_{input} )</th>
<th></th>
<th>( S_{output} )</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>-</td>
<td>00</td>
<td>-</td>
</tr>
<tr>
<td>01</td>
<td>-</td>
<td>01</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td></td>
<td>11</td>
<td></td>
</tr>
</tbody>
</table>

interpretation of input formula without its neg

output is mirror image of \( S_{positive} \)
5 Conjunction

A conjunction is just an ordered sequence of updates:

\[ s[\phi \land \psi]_{M,g} = s[\phi]_{M,g} s[\psi]_{M,g} \]

So, unlike classical conjunction, this operation is not commutative. However, order matters only when one of the conjuncts is an existential. Otherwise, the behavior is classical.

6 Existentials

\[ s[\exists x \phi]_{M,g} = \{ e' \cdot d \mid d \in D \land e' \in s[\phi]_{M,g} \} \]

6.1 State extension

6.2 Elimination of cases
7 Equivalences and useful inputs to try

# DPL-style scope extension
((Ex:(even x)) & (p0 = 2))
iff
(Ex:((even x) & (x = 2)))

# noncommutative conjunction
((p0 = 1) & (Ex:(even x)))
is different from
((Ex:(even x)) & (p0 = 1))

# donkey sentence
# ‘if every number equals a number, then it equals it’
~((Ex:(Ey:(x = y))) & ~(p0 = p1))

# nested existentials
(Ex:(Ey:(Ez:((prime x) & ((odd y) & (null z))))))
(Ex:(Ey:(Ez:((x = 0) & ((y = 1) & (z = 2)))))
References


