

Records as Resources

Using Martin-Löf Type Theory to Combine “Technologies” for Natural Language Interpretation

**Robin Cooper
Göteborg University**

The Technologies

- Montague Semantics
- Discourse Representation Theory
- Situation Semantics
- HPSG

Theories that have been used widely in computational semantic implementations.

Yet they don't provide the same coverage.

The type theoretical technology

Incorporating records into Martin-Löf type theory.

Work in Göteborg: Betarte, Tasistro, Coquand

Currently involved in Göteborgs Records and Dialogue Semantics project:

Robin Cooper

Thierry Coquand

Staffan Larsson

Peter Ljunglöf

Bengt Nordström

Aarne Ranta

Dependent record types and Martin-Löf type theory

- provide us with the tools we need for semantic notions (e.g. functions, binding and a way of dealing with intensionality) – Montague semantics
- provide an inherently dynamic analysis of anaphora essentially similar to the classical DRT approach (using dependent types) – Discourse Representation Theory
- provide a notion of truth as holding of some part of the world (a “proof object”) in virtue of the *propositions as types principle* – situation semantics
- provide us with feature structure like objects that can be used to combine phonological, syntactic and semantic information – HPSG

a man owns a donkey

Record type:

$$\left[\begin{array}{l} x : \text{Ind} \\ c_1 : \text{man}(x) \\ y : \text{Ind} \\ c_2 : \text{donkey}(y) \\ c_3 : \text{own}(x,y) \end{array} \right]$$

Record:

$$\left[\begin{array}{l} x = a \\ c_1 = p_1 \\ y = b \\ c_2 = p_2 \\ c_3 = p_3 \end{array} \right]$$

where

a, b are of type Ind, individuals

p_1 is a proof of $\text{man}(a)$

p_2 is a proof of $\text{donkey}(b)$

p_3 is a proof of $\text{own}(a, b)$

Compositionality

Sample derivation: *every man owns a donkey*

a donkey

$$\lambda R_1:([x:\text{Ind}])\text{RecType } \lambda R_2:([x:\text{Ind}])\text{RecType} \left[\begin{array}{l} \text{par} \quad : \quad [x : \text{Ind}] \\ \text{restr} \quad : \quad R_1 @ \text{par} \\ \text{scope} \quad : \quad R_2 @ \text{par} \end{array} \right]$$

@

$$\lambda r:[x:\text{Ind}]([c:\text{donkey}(r.x)])$$

=

$$\lambda R_2:([x:\text{Ind}])\text{RecType} \left[\begin{array}{l} \text{par} \quad : \quad [x : \text{Ind}] \\ \text{restr} \quad : \quad [c : \text{donkey}(\text{par}.x)] \\ \text{scope} \quad : \quad R_2 @ \text{par} \end{array} \right]$$

own a donkey

$\lambda \mathcal{N} : (([x:\text{Ind}])\text{RecType})\text{RecType} \lambda r_1 : [x:\text{Ind}] (\mathcal{N} @ \lambda r_2 : [x:\text{Ind}] ([c:\text{own}(r_1.x, r_2.x)]))$

@

$\lambda R_2 : ([x:\text{Ind}])\text{RecType} \left[\begin{array}{l} \text{par} \quad : \quad [x : \text{Ind}] \\ \text{restr} \quad : \quad [c : \text{donkey}(\text{par}.x)] \\ \text{scope} \quad : \quad R_2 @ \text{par} \end{array} \right]$

=

$\lambda r_1 : [x:\text{Ind}] \left(\left[\begin{array}{l} \text{par} \quad : \quad [x : \text{Ind}] \\ \text{restr} \quad : \quad [c : \text{donkey}(\text{par}.x)] \\ \text{scope} \quad : \quad [c : \text{own}(r_1.x, \text{par}.x)] \end{array} \right] \right)$

every man

$\lambda R_1:([x:\text{Ind}])\text{RecType } \lambda R_2:([x:\text{Ind}])\text{RecType}$

$\left[f : (r : \left[\begin{array}{l} \text{par} : [x : \text{Ind}] \\ \text{restr} : R_1 @ \text{par} \end{array} \right]) R_2 @ r.\text{par} \right]$

@

$\lambda r:[x:\text{Ind}]([c:\text{man}(r.x)])$

=

$\lambda R_2:([x:\text{Ind}])\text{RecType}$

$\left[f : (r : \left[\begin{array}{l} \text{par} : [x : \text{Ind}] \\ \text{restr} : [c : \text{man}(\text{par}.x)] \end{array} \right]) R_2 @ r.\text{par} \right]$

every man owns a donkey

$\lambda R_2:([\mathbf{x}:\text{Ind}])\text{RecType}$

$$\left[\mathbf{f} : (r : \left[\begin{array}{l} \text{par} : [\mathbf{x} : \text{Ind}] \\ \text{restr} : [\mathbf{c} : \text{man}(\text{par.x})] \end{array} \right]) R_2 @ r.\text{par} \right]$$

@

$$\lambda r_1:[\mathbf{x}:\text{Ind}] \left(\left[\begin{array}{l} \text{par} : [\mathbf{x} : \text{Ind}] \\ \text{restr} : [\mathbf{c} : \text{donkey}(\text{par.x})] \\ \text{scope} : [\mathbf{c} : \text{own}(r_1.\mathbf{x},\text{par.x})] \end{array} \right] \right)$$

=

$$\left[\mathbf{f} : (r : \left[\begin{array}{l} \text{par} : [\mathbf{x} : \text{Ind}] \\ \text{restr} : [\mathbf{c} : \text{man}(\text{par.x})] \end{array} \right]) \left[\begin{array}{l} \text{par} : [\mathbf{x} : \text{Ind}] \\ \text{restr} : [\mathbf{c} : \text{donkey}(\text{par.x})] \\ \text{scope} : [\mathbf{c} : \text{own}(r.\text{par.x},\text{par.x})] \end{array} \right] \right]$$

Flattening

$$\left[f : (r : \left[\begin{array}{l} \text{par.x} : \text{Ind} \\ \text{restr.c} : \text{man}(\text{par.x}) \end{array} \right]) \left[\begin{array}{l} \text{par.x} : \text{Ind} \\ \text{restr.c} : \text{donkey}(\text{par.x}) \\ \text{scope.c} : \text{own}(r.\text{par.x},\text{par.x}) \end{array} \right] \right]$$

Relabelling

$$\left[f : (r : \left[\begin{array}{l} x : \text{Ind} \\ c_1 : \text{man}(x) \end{array} \right]) \left[\begin{array}{l} y : \text{Ind} \\ c_2 : \text{donkey}(y) \\ c_3 : \text{own}(r.x,y) \end{array} \right] \right]$$

r-suppression (syntactic sugar)

$$\left[f : \left(\left[\begin{array}{l} x : \text{Ind} \\ c_1 : \text{man}(x) \end{array} \right] \right) \left[\begin{array}{l} y : \text{Ind} \\ c_2 : \text{donkey}(y) \\ c_3 : \text{own}(x,y) \end{array} \right] \right]$$

Records as resources

Domain restrictions

every/the man owns a donkey

Resource situations in situation semantics

“everything which is a man in situation s ”

“everything which is a man in record r ”

We use

$$r \models T$$

to represent the type of objects in record r which are of type T

$a : r \models T$ just in case for some $\ell, r : [\ell:T]$ and $r.\ell = a$

Anchored resources (specific)

$\lambda r_1 : \text{Rec}$

$\lambda r_2 : \text{Rec}$

$$\left[\text{f} : \left(r : \left[\begin{array}{l} \text{x} : r_1 \models \text{Ind} \\ \text{c}_1 : r_1 \models \text{man}(\text{x}) \end{array} \right] \right) \left[\begin{array}{l} \text{y} : r_2 \models \text{Ind} \\ \text{c}_2 : r_2 \models \text{donkey}(\text{y}) \\ \text{c}_3 : \text{own}(r.\text{x},\text{y}) \end{array} \right] \right] \\ @ \text{res}_1 @ \text{res}_2$$

“Every man in res_1 owns a donkey in res_2 .”

Non-anchored resources (non-specific, existentially quantified)

$$\left[\begin{array}{l} \text{res}_1 : \text{Rec} \\ \text{res}_2 : \text{Rec} \\ \text{f} : \left(r : \left[\begin{array}{l} \text{x} : \text{res}_1 \models \text{Ind} \\ \text{c}_1 : \text{res}_1 \models \text{man}(\text{x}) \end{array} \right] \right) \left[\begin{array}{l} \text{y} : \text{res}_2 \models \text{Ind} \\ \text{c}_2 : \text{res}_2 \models \text{donkey}(\text{y}) \\ \text{c}_3 : \text{own}(r.\text{x},\text{y}) \end{array} \right] \end{array} \right]$$

“There are resources $\text{res}_1, \text{res}_2$ such that every man in res_1 owns a donkey in res_2 .”

Quantification over resources (a kind of generic)

$$\left[f : (r : \left[\begin{array}{l} \text{res}_1 : \text{Rec} \\ x : \text{res}_1 \models \text{Ind} \\ c_1 : \text{res}_1 \models \text{man}(x) \end{array} \right]) \left[\begin{array}{l} \text{res}_2 : \text{Rec} \\ y : \text{res}_2 \models \text{Ind} \\ c_2 : \text{res}_2 \models \text{donkey}(y) \\ c_3 : \text{own}(r.x,y) \end{array} \right] \right]$$

“For every resource res_1 and every man x in res_1 there is a resource res_2 such that x owns a donkey in res_2 .”

Records (situations) as modules

Types restricted to resources import information from the resource (cf importing from modules in programming languages).

If $r : [a : r_1 \models T]$
then $r : [a : T]$

r can be used as a resource for another record.

This kind of “information spreading” not present in situation theory.

Implementation

Advantages of Oz:

- functional programming
- (constraint) logic programming
- support for records

A potential problem:

- How do you display a function?

Record Types as Oz procedures

Code:

```
Mod = proc {$ R}
    {Rec R}
    {Ind R^a}
    R^ac = man(R.a)
    {Ind R^b}
    R^bc = donkey(R.b)
    R^c = own(R.a R.b)
end
```

Value of Mod:

<P/1 Mod>

Value of

```
{SearchAll proc {$ Y} {Mod Y} end}
```

where j, m are individuals:

```
[rec(a:j ac:man(j) b:j bc:donkey(j)
    c:own(j j) ...)
  rec(a:j ac:man(j) b:m bc:donkey(m)
    c:own(j m) ...)
  rec(a:m ac:man(m) b:j bc:donkey(j)
    c:own(m j) ...)
  rec(a:m ac:man(m) b:m bc:donkey(m)
    c:own(m m) ...)]
```

Value of

```
{SearchAll
  proc {$ Y} Y = rec(a:j ac:man(j) b:m
                    bc:donkey(m) c:own(j m)
                    c2:beat(j m) c3:kick(m j))
  {Mod Y} end}
```

```
[rec(
  a:j
  ac:man(j)
  b:m
  bc:donkey(m)
  c:own(j m)
  c2:beat(j m)
  c3:kick(m j))]
```

```
Res=rec(a:j ac:man(j) b:m bc:donkey(m) c:own(j m)
        c2:beat(j m) c3:kick(m j))
```

Value of {SearchAll {{A {Donkey Res}} Walk}}

```
[rec(
  par:rec(x:m ...)
  restr:rec(c:donkey(m) ...)
  scope:rec(c:walk(m) ...) ...)]
```

Value of

```
{Map {SearchAll {{A {Donkey Res}} Walk}}
  FlattenRecType}
```

```
[rec(par_x:m restr_c:donkey(m)
      scope_c:walk(m) ...)]
```

Conclusions

Records in Martin-Löf type theory

- allow us to combine various natural language “technologies”
 - Montague lambda technology
 - DRT, dynamic semantics
 - typed feature structures (HPSG)
- exploit key features of MLTT
 - dependent types
 - propositions as types
- give us a promising approach to handling resource restrictions using module and situation like features of records