A type theoretic approach to information state update in issue based dialogue management

Robin Cooper
Göteborg University

The Department of Linguistics
The Records and Dialogue Semantics Project

http://www.ling.gu.se/~cooper/records/ (These slides are there)

- Records, types and computational dialogue semantics
- Funded by
- 2003–2005
- Robin Cooper, Thierry Coquand, Staffan Larsson, Peter Ljunglöf, Bengt Nordström, Aarne Ranta

⇐ contents
Ingredients from (Martin-Löf) type theory

- records and record types
- dependent types
- “propositions” as types (of proofs)
- types as objects
- functions (\(\lambda\)-calculus)
- dependent function types
Four linguistic theories

- Montague semantics
  dynamic binding; improved treatment of intensionality (including perception); improved treatment of context dependence (resources)

- DRT
  $\lambda$-DRT; improved treatment of intensionality (including perception); improved treatment of context dependence (resources)

- situation semantics
  compositional treatment; dynamic binding; type discipline, for better or worse . . .

- HPSG
  binding and functions; both types and objects; no need to “code” semantics

Main advantage: you can get aspects of all four theories going at the same time.
Adding issue based dialogue management

- a declarative theoretical treatment of the information state update approach to dialogue
- integration with semantics (intensionality, anaphora, ...)

↩ contents
Combining “traditional” semantics and dialogue management

- no clear boundary between dialogue management and semantics
- cf. semantics and pragmatics
- Jonathan Ginzburg’s inspiration
- to Manfred Pinkal (Edilog): “Perhaps we are doing semantics after all.”
An example: Cross-speaker intensional identity

A: Sam says there’s a bug in the program.
B: Pat is looking for it
Contents

1  Records and compositional semantics  9
2  Issue based dialogue management  20
3  Update rules in type theory  31
A  Questions  53
B  Actions and Dialogue Moves  56

⇐ contents
1. Records and compositional semantics
Records and record types

If \( a_1 : T_1, a_2 : T_2(a_1), \ldots, a_n : T_n(a_1, a_2, \ldots, a_{n-1}) \),

the record:

\[
\begin{bmatrix}
    l_1 &= a_1 \\
    l_2 &= a_2 \\
    \vdots \\
    l_n &= a_n \\
    \vdots
\end{bmatrix}
\]

is of type:

\[
\begin{bmatrix}
    l_1 : T_1 \\
    l_2 : T_2(l_1) \\
    \vdots \\
    l_n : T_n(l_1, l_2, \ldots, l_{n-1})
\end{bmatrix}
\]
a man owns a donkey

Record type:
\[
\begin{bmatrix}
  x & : & \text{Ind} \\
  c_1 & : & \text{man}(x) \\
  y & : & \text{Ind} \\
  c_2 & : & \text{donkey}(y) \\
  c_3 & : & \text{own}(x, y)
\end{bmatrix}
\]

Record:
\[
\begin{bmatrix}
  x & = & a \\
  c_1 & = & p_1 \\
  y & = & b \\
  c_2 & = & p_2 \\
  c_3 & = & p_3
\end{bmatrix}
\]

where

\( a, b \) are of type \text{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)

\[\text{contents}\]
a man owns a donkey

Content (intension) is a record type:

\[
\begin{bmatrix}
  x & : & Ind \\
  c_1 & : & \text{man}(x) \\
  y & : & Ind \\
  c_2 & : & \text{donkey}(y) \\
  c_3 & : & \text{own}(x,y)
\end{bmatrix}
\]

- a record of this type may have additional fields
- the types man(x), donkey(y), own(x,y) are dependent types of proofs
Meaning

A function from contexts (modelled as records) to record types, i.e. of type $(T)RecType$, where $T$ is some record type.

A man owns a donkey

$\lambda r:Rec \left( \begin{array}{l} x : Ind \\ c_1 : \text{man}(x) \\ y : Ind \\ c_2 : \text{donkey}(y) \\ c_3 : \text{own}(x,y) \end{array} \right)$ of type $(Rec)RecType$
Meanings as dependent functions

Sam owns a donkey

\[ \lambda r: \left[ \begin{array}{c}
    x : \text{Ind} \\
    c_1 : \text{named}(x, \text{"Sam"}) \\
    c_3 : \text{own}(r.x,y)
  \end{array} \right] \left( \begin{array}{c}
    y : \text{Ind} \\
    c_2 : \text{donkey}(y)
  \end{array} \right) \]

cf. Montague, Kaplan
and within type theory using type theoretical contexts Ranta, Ahn, Piwek
among many others
Two techniques exploited in compositional treatment of anaphora

- manifest fields (Coquand, Pollack and Takeyama)
- metavariables (Göteborg work on proof editing)
Manifest fields

If \( a : T \), then \( T_a \) is a singleton type
\( b : T_a \) iff \( b = a \)

A manifest field in a record type is one whose type is a singleton type, e.g.
\[
\begin{array}{l}
[ x : T_a ]
\end{array}
\]
written for convenience as
\[
\begin{array}{l}
[ x=a : T ]
\end{array}
\]

Allows record types to be “progressively instantiated”.

We will allow dependent unique types, i.e. where \( a \) can be represented by a path in a record type.
Metavariables

We use metavariables (anonymous variables) ‘?’ in manifest fields in order to
treat anaphoric constructions.
Metavariables are resolved to paths.
Ultimately we plan to use a variant of David Beaver’s OT version of centering
theory for resolution.
Meaning of *Sam says there’s a bug in the program*

\[
\lambda r : \begin{cases}
  x & : \text{Ind} \\
  c_1 & : \text{named}(x, \text{“Sam”}) \\
  y & : \text{Ind} \\
  \text{res} & : \text{Rec} \\
  c_2 & : \text{program}(y) \mid \text{res} \\
  c_3 & : (r_1 : \begin{cases}
    z & : \text{Ind} \\
    c & : \text{program}(z) \mid \text{res} \\
  \end{cases})[c : \text{eq}(r_1.z, y, \text{Ind})]
\end{cases}
\]

\[
\left[
\begin{array}{c}
  p=\lambda r_2 : \text{Rec} \left( \begin{cases}
    x & : \text{Ind} \\
    c_4 & : \text{bug}(x, r.y) \\
  \end{cases} \right) : (\text{Rec})\text{RecType} \\
  c_5
\end{array}\right)
\]

\[\leftarrow \text{contents}\]
Meaning (underspecified) of *Pat is looking for it*

\[
\lambda r : \left[ \begin{array}{c}
x : Ind \\
c_1 : \text{named}(x, \text{“Pat”}) \\
p = \lambda r : ? \left( [y=?:\text{Ind}] \right) : (?)\text{RecType} \\
c_2 : \text{seek}(r.x, p)
\end{array} \right]
\]

We need to specify (by anaphora resolution):

- the type of context in which the *it* is interpreted (first and third question marks)
- which individual in a context of that type *it* is to denote (second question mark)
2. Issue based dialogue management
Information state update

• Complex structure information states
  – gameboard
  – records

• non-monotonic updates
  e.g. questions disappear from QUD (Questions under discussion)

cf. classical dynamic or update semantics
Issue based dialogue management

• Determination of the next dialogue contribution driven largely by QUD
  almost all contributions raise or address an issue (question under discussion)

• Issues can be raised by addressing them
  giving an answer to a question that hasn’t be explicitly stated (question accommodation)
Larsson’s formulation of update rules


- Update rules are defined on information states which are modelled as records
- A single utterance may unleash a whole chain of updates (i.e. generalisations beyond monolithic utterance updates)
- Exploits ordering of effects within update rules and ordering of update rules (prolog style)
- Prolog style use of variables (cf discussion by Johan Bos in connection with Dipper)
A simple information state, \( \mathcal{r} \)

Larsson, p. 56 (adapted)

\[
\begin{align*}
\text{private} & = \\
\text{agenda} & = [] \\
\text{plan} & = \\
& \quad [\text{raise}(?A.\text{how}(A)), \\
& \quad \text{findout}(?B.\text{dest\_city}(B)), \\
& \quad \text{findout}(?C.\text{dep\_city}(C)), \\
& \quad \text{findout}(?D.\text{month}(D)), \\
& \quad \text{findout}(?E.\text{dep\_date}(E)), \\
& \quad \text{findout}(?F.\text{class}(F)), \\
& \quad \text{consultDB}(?G.\text{price}(G))] \\
\text{bel} & = [] \\
\text{com} & = [] \\
\text{qud} & = [?H.\text{price}(H)] \\
\text{lu} & = \\
& \quad \text{speaker} = \text{usr} \\
& \quad \text{moves} = \{\text{ask}(?H.\text{price}(H))\}
\end{align*}
\]
\( r \) is of type:

Adapted from Larsson, p. 32

\[
\begin{cases}
\text{private} : \\
\text{plan} : [\text{Action}] \\
\text{bel} : \text{RecType} \\
\text{agenda} : [\text{Action}] \\
\text{com} : \text{RecType} \\
\text{qud} : [\text{Question}] \\
\text{shared} : \\
\text{lu} : \begin{cases}
\text{speaker} : \text{Participant} \\
\text{moves} : \{\text{Move}\}
\end{cases}
\end{cases}
\]

We call this type \( IS \), the type of information states.
\( r \) is also of type:

A singleton type using manifest fields

\[
\begin{align*}
\text{private :} & \quad \text{agenda}=&[
\quad \text{[Action]} \\
& \quad \text{[raise(\( A \).how(\( A \))),} \\
& \quad \text{findout(\( B \).dest\_city(\( B \))),} \\
& \quad \text{findout(\( C \).dep\_city(\( C \))),}
\]
\text{plan=}& \quad \text{findout(\( D \).month(\( D \))),} \quad \text{[Action]} \\
& \quad \text{findout(\( E \).dep\_date(\( E \))),} \\
& \quad \text{findout(\( F \).class(\( F \))),} \\
& \quad \text{consultDB(\( G \).price(\( G \)))}
\]
\text{bel=}=&\quad \text{[]} \\
\text{com=}=&\quad \text{[]} \\
\text{qud=}=&\quad \text{[\( H \).price(\( H \))] : [Question]}
\]
\text{shared :} & \quad \text{lu} \quad \text{[Move]}
\end{align*}
\]
Another type for $r$:

*Not* a singleton type, but some fields are manifest

$$\begin{align*}
\text{agenda} = &\quad : \quad [\text{Action}] \\
\quad &\quad [\text{raise(?A\text{.how}(A)),} \\
\quad &\quad \text{findout(?B\text{.dest\_city}(B)),} \\
\quad &\quad \text{findout(?C\text{.dep\_city}(C)),} \\
\quad &\quad \text{findout(?D\text{.month}(D),} \\
\quad &\quad \text{findout(?E\text{.dep\_date}(E),} \\
\quad &\quad \text{findout(?F\text{.class}(F')),} \\
\quad &\quad \text{consultDB(?G\text{.price}(G))}] \\
\text{bel} = &\quad : \quad \text{RecType} \\
\text{com} = &\quad : \quad \text{RecType} \\
\text{qud} = &\quad : \quad [\text{Question}] \\
\text{lu} = &\quad : \quad \text{Participant} \\
\quad &\quad \{\text{Move}\} \\
\end{align*}$$

$\iff$ contents
Typing as underspecification

- there is typically more than one object of each type
- singleton types correspond to total specification
- non-deterministic updates can be expressed in terms of types
Reasoning about updates

Updates
\[ \text{record}_1 \Rightarrow \text{record}_2 \Rightarrow \text{record}_3 \]

Reasoning about updates
\[ \text{record type}_1 \Rightarrow \text{record type}_2 \Rightarrow \text{record type}_3 \]

record \(\sim\rightarrow\) information state
record type \(\sim\rightarrow\) information about an information state
The type of the total information state, \( TIS \)

\[
\begin{array}{c|c}
\text{is} & IS \quad \% \text{info state} \\
\text{res} & RES \quad \% \text{resources} \\
\text{mod} & MOD \quad \% \text{module interfaces} \\
\text{flags} & FLAGS \quad \% \text{flags for language, domain, not necessarily part of tis}
\end{array}
\]

⇐ contents
3. Update rules in type theory
The basic intuition

If \( r_i : T_i \), then \( r_{i+1} : T_{i+1}(r_i) \)
Expressed as a function from records to record types
i.e., a function of type \((T_i)\text{RecType}\)

\[ \lambda r : T_i(T_{i+1}(r_i)) \]

Important: these are the same kinds of functions as our (static) meaning functions.
The discourse so far provides the context for the current utterance.
Adding conditions on the total information state

Exploits “propositions as types”

\[
\begin{align*}
\text{tis} & : TIS \\
\text{cond} & : T_{\text{cond}}
\end{align*}
\]
An update rule specification

integrateUsrAsk
class: integrate
Larsson, p. 40

\[
\lambda r : [\text{is}: \begin{cases} \text{private} : [\text{agenda} : [\text{Action}]] \\
\text{shared} : [\text{lu} : [\begin{cases} \text{speaker=usr} : \text{DP} \\
\text{moves} : \{\text{Move}\} \end{cases}]] \\
\text{qud} : [\text{Question}] \end{cases} \\
\text{cond} : [\begin{cases} q : \text{Question} \\
\text{c} : \text{member(ask(\text{cond.q}), tis.is.shared.lu.moves)} \end{cases}] \\
(\text{tis} : [\begin{cases} \text{private} : [\text{agenda=respond(\text{r.cond.q})}|\text{r.tis.is.private.agenda} : [\text{Action}]] \\
\text{shared} : [\text{qud}=\text{r.cond.q}|\text{r.tis.is.shared.qud} : [\text{Question}]] \end{cases}]])]
\]

\[\equiv\text{contents}\]
Suppose current info state is of type

\[
\text{tis: is :}\begin{aligned}
\text{private: } & \text{agenda=}[\text{Action}] \\
& \text{plan=}[\text{Action}] \\
& \text{bel=}[\text{RecType}] \\
\text{shared: } & \text{com=}[\text{RecType}] \\
& \text{qud=}[\text{Question}] \\
\text{lu: } & \begin{cases}
\text{speaker=usr} \\
\text{moves=\{ask(?A.time_next_train(Bclna, \text{ } A))\}} \\
\end{cases}
\end{aligned}
\]

\[\text{lu: moves=\{ask(?A.time_next_train(Bclna, \text{ } A))\}:\{\text{Move}\}}\]

\[\Leftrightarrow \text{contents}\]
then new info state must be of type

\[
\text{agenda = [respond(?A.time_next_train(Bclna, A))]: [Action]}
\]

\[
\text{plan = []: [Action]}
\]

\[
\text{bel = []: RecType}
\]

\[
\text{com = []: RecType}
\]

\[
\text{qud = [?A.time_next_train(Bclna, A)]: [Question]}
\]

\[
\text{lu: [speaker = usr: DP]}
\]

\[
\text{moves = \{ask(?A.time_next_train(Bclna, A))\}: \{Move\}}
\]

We know more about the type than the function expresses.

⇐ contents
How have we used the update rule to draw this conclusion?

1. instantiation of the update function with the type of the current info state
2. relabelling of shared paths
3. compute fixed point type of resulting function
4. garbage collection of unused “support” fields
Instantiation of update function

Suppose that the update rule has the function $\lambda r : T_1(T_2(r))$
and that what we know about the information state is that it is of type $T_{\text{curr}}$
The rule only fires if $T_{\text{curr}} \subseteq T_1$

We can instantiate the rule to $\lambda r : T_{\text{curr}}(T_2(r))$
Relabelling of shared paths

For the next step we will need to ensure that \( T_{\text{curr}} \) and \( T_2(r) \) do not have any paths in common which may have different values (see integrateUsrAsk)

We therefore prefix any shared path in \( T_{\text{curr}} \) with the label ‘prev(ious)’
Fixed point types of functions

A *fixed point type* for a function \( f : (T)RecType \) is a type \( T \) such that
\[ a : T \text{ iff } a : f(a) \]

If \( f = \lambda r : T_1(T_2(r)) \) and \( T_1, T_2(r) \) do not share any paths
then the fixed point type of \( f \) is \( T_1 \cup T_2(r)’ \)
(where \( T_2(r)’ \) is like \( T_2(r) \) except that dependence on \( r \) has been replaced by
dependence on corresponding fields in the new type).
An example

Stripped down to fit on slides

\[ \lambda r : \begin{cases} \text{prev} : & \begin{cases} \text{tis} : & \begin{cases} \text{is} : & \text{private} : & \text{agenda} = [] : \text{[Action]} \end{cases} \\ \text{shared} : & \text{qud} = [] : \text{[Question]} \end{cases} \\ \end{cases} \\\\
\text{tis} : & \begin{cases} \text{is} : & \text{shared} : & \text{lu} : & \text{speaker} = \text{usr} \\ \text{moves} = \{ \text{ask}(\text{?A.time	extunderscore next	extunderscore train(Bclna,A))} \} : \text{[Move]} \\
\end{cases} \\
\text{cond} : & \begin{cases} \text{q} : \text{Question} \\ \text{c} : \text{member(ask(cond.q), tis.is.shared.lu.moves)} \\
\end{cases} \\
\end{cases} \\
\text{tis} : & \begin{cases} \text{is} : & \text{private} : & \text{agenda} = \text{respond}(r.\text{cond.q}) | r.\text{prev}.\text{tis.is.private.agenda} : \text{[Action]} \\ \text{shared} : & \text{qud} = r.\text{cond.q} | r.\text{prev}.\text{tis.is.shared.qud} : \text{[Question]} \\
\end{cases} \\
\end{cases} \]

⇐ contents
The fixed point type

prev : [tis: [is: [private : [agenda=[] : [Action]]]]]
      [shared : [qud=[] : [Question]]]]

private: agenda=respond(cond.q)|prev.tis.is.private.agenda : [Action]]

shared: qud=cond.q|prev.tis.is.shared.qud:[Question]

lu: [speaker=usr:DP
    moves={ask(?A.time_next_train(Bclna,A)):{Move}] ]

cond: [q : Question
c : member(ask(cond.q), tis.is.shared.lu.moves) ]

⇐ contents
Reasoning about unique solutions

prev: [tis: [is: [private: [agenda=[] : [Action]]] [shared: [qud=[] : [Question]]]]]]

tis: [is: [private: [agenda=respond(cond.q)|prev.tis.is.private.agenda : [Action]]] [shared: [qud=cond.q|prev.tis.is.shared.qud:[Question]]]]

tis: [is: [private: [agenda=respond(cond.q)|prev.tis.is.private.agenda : [Action]]] [shared: [qud=cond.q|prev.tis.is.shared.qud:[Question]]]]

cond: [q=?A.time_next_train(Bclna,A) : Question] [c:member(ask(cond.q), tis.is.shared.lu.moves)]

⇐ contents
Substituting values of manifest fields

prev:
  tis:
    is:
      private:
        agenda=[[]: [Action]]
    shared:
      qud=[[]: [Question]]
private:
  agenda=[respond(?A.time_next_train(Bclna,A)) : [Action]]
  qud=[?A.time_next_train(Bclna,A) : [Question]]
  lu:
    speaker=usr:DP
    moves={ask(?A.time_next_train(Bclna,A)) : {Move}}
cond:
  q=?A.time_next_train(Bclna,A) : [Question]
c:member(ask(cond.q), tis.is.shared.lu.moves)
Garbage collection

Remove prev- and cond-paths not depended on in tis-paths

[tis: is: shared: 

private: 

agenda=[respond(?A.time_next_train(Bclna,A)) : [Action]]

qud=[?A.time_next_train(Bclna,A)]:[Question]

lu:

speaker=usr:DP

moves={ask(?A.time_next_train(Bclna,A))}:{Move}]

] ] ] ]

⇐ contents
Using meanings to update information states

- used fixed point types of meaning functions
- i.e. presupposition associated with speech acts, not with the store of knowledge in the information state (Staffan Larsson, pc)
- assertions (addressing issues under discussion) refine the field tis.is.shared.com

cf. use of type theoretical contexts to capture dynamic aspects of meaning: Ahn, Piwek, Ranta
After *Sam says there’s a bug in the program*

<table>
<thead>
<tr>
<th>tis:</th>
<th>is:</th>
<th>shared:</th>
<th>com=</th>
</tr>
</thead>
<tbody>
<tr>
<td>x:Ind</td>
<td>c₁:named(x, “Sam”)</td>
<td>y:Ind</td>
<td>res:Rec</td>
</tr>
</tbody>
</table>
| c₂:program(y)↾res | c₃:(r₁: [z:Ind
c₄:program(z)↾res] )ervice(r₁.z, y, Ind] |
| p=λr₂ : Rec ([x:Ind
c₅:say(x, p)]):((Rec)RecType |

\[\text{contents}\]
After *Pat is looking for it*

\[
\begin{align*}
\text{com} &= \begin{cases}
  x: \text{Ind} \\
  c_1: \text{named}(x, "Sam") \\
  y: \text{Ind} \\
  \text{res: Rec} \\
  c_2: \text{program}(y) \mid \text{res} \\
  c_3: (r_1: \begin{cases}
    z: \text{Ind} \\
    c: \text{program}(z) \mid \text{res}
  \end{cases}) [c: \text{eq}(r_1.z, y, \text{Ind})] \\
  p_1 = \lambda r_2: \text{Rec} (\begin{cases}
    x: \text{Ind} \\
    c_4: \text{bug}(x, y)
  \end{cases}): (\text{Rec})\text{RecType} \\
  c_5: \text{say}(x, p_1) \\
  z: \text{Ind} \\
  c_6: \text{named}(z, "Pat") \\
  p_2 = \lambda r: ? (\begin{cases}
    y = ?: \text{Ind}
  \end{cases}): (?\text{RecType}) \\
  c_2: \text{seek}(z, p_2)
\end{cases}
\end{align*}
\]
After anaphora resolution

\[
\frac{\text{com=} \quad \begin{array}{l}
\left[ \begin{array}{l}
x: \text{Ind} \\
c_1: \text{named}(x, \text{“Sam”}) \\
y: \text{Ind} \\
\text{res}: \text{Rec} \\
c_2: \text{program}(y) | \text{res} \\
c_3: (r_1: \left[ \begin{array}{l}
z: \text{Ind} \\
c: \text{program}(z) | \text{res} \end{array} \right] ) \left[ \begin{array}{l}
c: \text{eq}(r_1.z, y, \text{Ind}) \end{array} \right] \\
p_1 = \lambda r_2 : \text{Rec} ( \left[ \begin{array}{l}
x: \text{Ind} \\
c_4: \text{bug}(x, y) \end{array} \right] ) : (\text{Rec})\text{RecType} \\
c_5: \text{say}(x, p_1) \\
z: \text{Ind} \\
c_6: \text{named}(z, \text{“Pat”}) \\
p_2 = \lambda r : \mathcal{F}(p_1) ( \left[ \begin{array}{l}
y = r.x: \text{Ind} \end{array} \right] ) : (\mathcal{F}(p_1))\text{RecType} \\
c_2: \text{seek}(z, p_2)
\end{array} \right]} \right] : \text{RecType}
}{\text{contents} 49/58}
i.e.,

\[
\begin{align*}
\text{com} &= \begin{cases} 
  \text{x:Ind} \\
  c_1: \text{named(x, "Sam")} \\
  y: \text{Ind} \\
  \text{res: Rec} \\
  c_2: \text{program(y) | res} \\
  c_3: (r_1: \begin{cases} 
    \text{z: Ind} \\
    \text{c: program(z) | res} 
  \end{cases} ) [c: \text{eq}(r_1.z, y, \text{Ind})] \\
  p_1 = \lambda r_2: \text{Rec} (\begin{cases} 
    \text{x: Ind} \\
    \text{c_4: bug(x, y)} 
  \end{cases} ) (\begin{cases} 
    \text{RecType} 
  \end{cases}) \\
  c_5: \text{say(x, p_1)} \\
  z: \text{Ind} \\
  c_6: \text{named(z, "Pat")} \\
  p_2 = \lambda r: \begin{cases} 
    \text{x: Ind} \\
    \text{c_4: bug(x, y)} 
  \end{cases} (\begin{cases} 
    \text{y = r . x: Ind} 
  \end{cases}) (\begin{cases} 
    \text{RecType} 
  \end{cases}) \\
  c_2: \text{seek(z, p_2)} \\
\end{cases} \\
\end{align*}
\]

\[\Rightarrow \text{contents}\]
Implementation plans

• Not the next version of TrindiKit

• Towards an abstract machine for reasoning about information state updates
Conclusions

Type theory with records provides us with a powerful formalism for

- semantics

- feature-based analyses

- information state update

and, importantly, an integrated approach.
A. Questions
Contents of questions

Who left?

\(?(\lambda r : \left[ x : \text{Ind} \right] (\left[c: \text{leave}(r.x)\right]))\)

Did a representative leave?

\(? (\lambda r : [] (\left[\begin{array}{l} x : \text{Ind} \\ c_1 : \text{representative}(x) \\ c_2 : \text{leave}(x) \end{array}\right]))\)

where [] is a variant notation for \text{Rec}, the type of all records

Types of functions in these examples: (\left[ x : \text{Ind} \right]\)\text{RecType}, (\text{Rec})\text{RecType}

Both these functions belong to the polymorphic type \{\text{Rec}\}\text{RecType}

which can also be thought of as the type of partial functions from records to record types

\(\Leftarrow \text{contents}\)

54/58
‘?’, the question function

a polymorphic function of type \( (\{Rec\}RecType)Question \)
i.e. it maps any function from records (of a given type) to record types to a question.

*Question* is a basic type.
B. Actions and Dialogue Moves
Actions

raise : (Question)Action
findout : (Question)Action
respond : (Question)Action
consultDB : (Question)Action
Dialogue moves

ask : *(Question)*Move

cf. illocutionary force