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

Robin Cooper
Göteborg University

GU Dialogue Systems Lab

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 VETENSKAPSRÅDET THE SWEDISH RESEARCH COUNCIL
- 2003–2005
- CHALMERS GÖTEBORG UNIVERSITY Computing Science — Computer Science and Engineering – Chalmers University of Technology and Göteborg University
- Robin Cooper, Thierry Coquand, Staffan Larsson, Peter Ljunglöf, Bengt Nordström, Aarne Ranta

Ingredients from (Martin-Löf) type theory

- records and record types
- dependent types
- “propositions” as types (of proofs)
- types as objects
- functions (λ-calculus)
- dependent function types

Four linguistic theories

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

- DRT
  λ-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.

GU Dialogue Systems Lab

Adding issue based dialogue management

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

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

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:

$$\begin{bmatrix} l_1 & = & a_1 \\ l_2 & = & a_2 \\ \ldots & & \\ l_n & = & a_n \\ \ldots & & \end{bmatrix}$$

is of type:

$$\begin{bmatrix} l_1 & : & T_1 \\ l_2 & : & T_2(l_1) \\ \ldots & & \\ l_n & : & T_n(l_1, l_2, \ldots, l_{n-1}) \end{bmatrix}$$

a man owns a donkey

Record type:

$$\begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{c}_1 & : & \text{man(x)} \\ \text{y} & : & \textit{Ind} \\ \text{c}_2 & : & \text{donkey(y)} \\ \text{c}_3 & : & \text{own(x,y)} \end{bmatrix}$$

Record:

$$\begin{bmatrix} \text{x} & = & a \\ \text{c}_1 & = & p_1 \\ \text{y} & = & b \\ \text{c}_2 & = & p_2 \\ \text{c}_3 & = & p_3 \end{bmatrix}$$

where

a, b are of type *Ind*, individuals

p_1 is a proof of man(a)

p_2 is a proof of donkey(b)

p_3 is a proof of own(a, b)

a man owns a donkey

Content (*intension*) is a record type:

$$\begin{bmatrix} \text{x} & : & \mathit{Ind} \\ \text{c}_1 & : & \text{man(x)} \\ \text{y} & : & \mathit{Ind} \\ \text{c}_2 & : & \text{donkey(y)} \\ \text{c}_3 & : & \text{own(x,y)} \end{bmatrix}$$

- 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

$$\lambda r{:}\mathit{Rec}\ (\begin{bmatrix} \mathrm{x} & : & \mathit{Ind} \\ \mathrm{c}_1 & : & \mathrm{man(x)} \\ \mathrm{y} & : & \mathit{Ind} \\ \mathrm{c}_2 & : & \mathrm{donkey(y)} \\ \mathrm{c}_3 & : & \mathrm{own(x,y)} \end{bmatrix}) \qquad \text{of type } (\mathit{Rec})\mathit{RecType}$$

Meanings as *dependent* functions

Sam owns a donkey

$$\lambda r{:}\begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{c}_1 & : & \text{named(x, “Sam”)} \end{bmatrix} \left(\begin{bmatrix} \text{y} & : & \textit{Ind} \\ \text{c}_2 & : & \text{donkey(y)} \\ \text{c}_3 & : & \text{own}(r\text{.x,y}) \end{bmatrix} \right)$$

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.
$\left[\ x\ :\ T_a\ \right]$
written for convenience as
$\left[\ x{=}a\ :\ T\ \right]$
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*

$$\lambda r : \begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{c}_1 & : & \text{named}(\text{x}, \text{“Sam”}) \\ \text{y} & : & \textit{Ind} \\ \text{res} & : & \textit{Rec} \\ \text{c}_2 & : & \text{program}(\text{y})\upharpoonright\text{res} \\ \text{c}_3 & : & (r_1{:}\begin{bmatrix} \text{z} & : & \textit{Ind} \\ \text{c} & : & \text{program}(\text{z})\upharpoonright\text{res} \end{bmatrix})\begin{bmatrix} \text{c} & : & \text{eq}(r_1.\text{z}, \text{y}, \textit{Ind}) \end{bmatrix} \end{bmatrix}$$

$$\left(\begin{bmatrix} \text{p}{=}\lambda r_2 : \textit{Rec}\,(\begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{c}_4 & : & \text{bug}(\text{x}, r.\text{y}) \end{bmatrix}) & : & (\textit{Rec})\textit{RecType} \\ \text{c}_5 & : & \text{say}(r.\text{x}, \text{p}) \end{bmatrix}\right)$$

Meaning (underspecified) of *Pat is looking for it*

$$\lambda r : \begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{c}_1 & : & \text{named(x, “Pat”)} \end{bmatrix}$$
$$(\begin{bmatrix} \text{p}=\lambda r : ?\ (\begin{bmatrix}\text{y}=?{:}\textit{Ind}\end{bmatrix}) & : & \text{(?)}\textit{RecType} \\ \text{c}_2 & : & \text{seek}(r.\text{x, p}) \end{bmatrix})$$

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

Staffan Larsson (2002) *Issue-based Dialogue Management*, Ph.D. diss., Department of Linguistics, Göteborg.

- 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)

GU Dialogue Systems Lab

A simple information state, r

Larsson, p. 56 (adapted)

$$
\begin{bmatrix}
\text{private} & = & \begin{bmatrix}
\text{agenda} & = & [] \\
\text{plan} & = & \begin{array}{l}
[\text{raise}(?A.\text{how}(A)), \\
\ \text{findout}(?B.\text{dest\_city}(B)), \\
\ \text{findout}(?C.\text{dep\_city}(C)), \\
\ \text{findout}(?D.\text{month}(D), \\
\ \text{findout}(?E.\text{dep\_date}(E), \\
\ \text{findout}(?F.\text{class}(F)), \\
\ \text{consultDB}(?G.\text{price}(G))]
\end{array} \\
\text{bel} & = & []
\end{bmatrix} \\
\text{shared} & = & \begin{bmatrix}
\text{com} & = & [] \\
\text{qud} & = & [?H.\text{price}(H)] \\
\text{lu} & = & \begin{bmatrix}
\text{speaker} & = & \text{usr} \\
\text{moves} & = & \{\text{ask}(?H.\text{price}(H))\}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix}
$$

⇐ contents

r is of type:

Adapted from Larsson, p. 32

$$\left[\begin{array}{lll} \text{private} & : & \left[\begin{array}{lll} \text{agenda} & : & [\mathit{Action}] \\ \text{plan} & : & [\mathit{Action}] \\ \text{bel} & : & \mathit{RecType} \end{array}\right] \\ \text{shared} & : & \left[\begin{array}{lll} \text{com} & : & \mathit{RecType} \\ \text{qud} & : & [\mathit{Question}] \\ \text{lu} & : & \left[\begin{array}{lll} \text{speaker} & : & \mathit{Participant} \\ \text{moves} & : & \{\mathit{Move}\} \end{array}\right] \end{array}\right] \end{array}\right]$$

We call this type *IS*, the type of information states.

r is *also* of type:

A *singleton type* using *manifest fields*

$$\left[\begin{array}{lcl} \text{private} & : & \left[\begin{array}{lcl} \text{agenda=[]} & : & [\textit{Action}] \\ \text{plan=} \begin{array}{l} \text{[raise(?}A\text{.how(}A\text{)),} \\ \text{findout(?}B\text{.dest\_city(}B\text{)),} \\ \text{findout(?}C\text{.dep\_city(}C\text{)),} \\ \text{findout(?}D\text{.month(}D\text{),} \\ \text{findout(?}E\text{.dep\_date(}E\text{),} \\ \text{findout(?}F\text{.class(}F\text{)),} \\ \text{consultDB(?}G\text{.price(}G\text{))]} \end{array} & : & [\textit{Action}] \\ \text{bel=[]} & : & \textit{RecType} \end{array}\right] \\ \text{shared} & : & \left[\begin{array}{lcl} \text{com=[]} & : & \textit{RecType} \\ \text{qud=[?}H\text{.price(}H\text{)]} & : & [\textit{Question}] \\ \text{lu} & : & \left[\begin{array}{lcl} \text{speaker=usr} & : & \textit{Participant} \\ \text{moves=}\{\text{ask(?}H\text{.price(}H\text{))}\} & : & \{\textit{Move}\} \end{array}\right] \end{array}\right] \end{array}\right]$$

Another type for r:

Not a singleton type, but some fields are manifest

$$
\left[\begin{array}{lll}
\text{private} & : & \left[\begin{array}{lll}
\text{agenda=[]} & : & [\textit{Action}] \\
\text{plan=} \begin{array}{l} [\text{raise}(?A.\text{how}(A)), \\ \text{findout}(?B.\text{dest\_city}(B)), \\ \text{findout}(?C.\text{dep\_city}(C)), \\ \text{findout}(?D.\text{month}(D), \\ \text{findout}(?E.\text{dep\_date}(E), \\ \text{findout}(?F.\text{class}(F)), \\ \text{consultDB}(?G.\text{price}(G))] \end{array} & : & [\textit{Action}] \\
\text{bel} & : & \textit{RecType}
\end{array}\right] \\
\text{shared} & : & \left[\begin{array}{lll}
\text{com} & : & \textit{RecType} \\
\text{qud=[}?H.\text{price}(H)] & : & [\textit{Question}] \\
\text{lu} & : & \left[\begin{array}{lll}
\text{speaker} & : & \textit{Participant} \\
\text{moves=}\{\text{ask}(?H.\text{price}(H))\} & : & \{\textit{Move}\}
\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

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

record$_1$ $\Rightarrow$ record$_2$ $\Rightarrow$ record$_3$

Reasoning about updates

record type$_1$ $\Rightarrow$ record type$_2$ $\Rightarrow$ record type$_3$

record $\rightsquigarrow$ information state
record type $\rightsquigarrow$ information about an information state

The type of the total information state, *TIS*

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

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)*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"

$$\begin{bmatrix} \text{tis} & : & \mathit{TIS} \\ \text{cond} & : & T_{cond} \end{bmatrix}$$

An update rule specification

integrateUsrAsk
class: integrate
Larsson, p. 40

$$
\lambda r : \left[\begin{array}{lll}
\text{tis} & : & \left[\text{is:} \left[\begin{array}{lll}
\text{private} & : & \left[\begin{array}{lll} \text{agenda} & : & [\mathit{Action}] \end{array}\right] \\
\text{shared} & : & \left[\begin{array}{lll}
\text{lu} & : & \left[\begin{array}{lll} \text{speaker=usr} & : & \mathit{DP} \\ \text{moves} & : & \{\mathit{Move}\} \end{array}\right] \\
\text{qud} & : & [\mathit{Question}]
\end{array}\right]
\end{array}\right]\right] \\
\text{cond:} & & \left[\begin{array}{lll}
\text{q} & : & \mathit{Question} \\
\text{c} & : & \text{member(ask(cond.q), tis.is.shared.lu.moves)}
\end{array}\right]
\end{array}\right]
$$

$$
(\left[\text{tis:} \left[\text{is:} \left[\begin{array}{ll}
\text{private:} & \left[\begin{array}{lll} \text{agenda=respond(}r\text{.cond.q)} | r\text{.tis.is.private.agenda} & : & [\mathit{Action}] \end{array}\right] \\
\text{shared:} & \left[\begin{array}{lll} \text{qud=}r\text{.cond.q} | r\text{.tis.is.shared.qud} & : & [\mathit{Question}] \end{array}\right]
\end{array}\right]\right]\right])
$$

Suppose current info state is of type

$$\left[\text{tis}: \left[\begin{array}{ll}\text{is} : \left[\begin{array}{ll}\text{private}: & \left[\begin{array}{ll}\text{agenda=[]}: & [\mathit{Action}] \\ \text{plan=[]} & :[\mathit{Action}] \\ \text{bel=[]} & :\mathit{RecType}\end{array}\right] \\ \text{shared}: & \left[\begin{array}{l}\text{com=[]}:\mathit{RecType} \\ \text{qud=[]}:[\mathit{Question}] \\ \text{lu}: \left[\begin{array}{ll}\text{speaker=usr} & :\mathit{DP} \\ \text{moves=}\{\text{ask}(?A.\text{time\_next\_train}(\text{Bclna}, A))\} & :\{\mathit{Move}\}\end{array}\right]\end{array}\right]\end{array}\right] \\ \ldots\end{array}\right]\right]$$

then new info state must be of type

$$
\left[\begin{array}{ll}
\text{tis:} & \left[\begin{array}{ll}
\text{is} : & \left[\begin{array}{ll}
\text{private:} & \left[\begin{array}{ll}
\text{agenda=[respond(?}A\text{.time\_next\_train(Bclna, }A\text{))]} & :[\mathit{Action}] \\
\text{plan=[]} & :[\mathit{Action}] \\
\text{bel=[]} & :\mathit{RecType}
\end{array}\right] \\
\text{shared:} & \left[\begin{array}{l}
\text{com=[]}:\mathit{RecType} \\
\text{qud=[?}A\text{.time\_next\_train(Bclna, }A\text{)]}:[\mathit{Question}] \\
\text{lu:} \left[\begin{array}{ll}
\text{speaker=usr} & :\mathit{DP} \\
\text{moves=}\{\text{ask(?}A\text{.time\_next\_train(Bclna, }A\text{))}\} & :\{\mathit{Move}\}
\end{array}\right]
\end{array}\right]
\end{array}\right] \\
\ldots &
\end{array}\right]
\end{array}\right]
$$

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_{curr}

The rule only fires if $T_{curr} \sqsubseteq T_1$

We can *instantiate* the rule to $\lambda r : T_{curr}(T_2(r))$

Relabelling of shared paths

For the next step we will need to ensure that T_{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_{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 $a : T$ 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

$$
\lambda r : \left[\begin{array}{lll}
\textcolor{red}{\text{prev}} & : & \left[\text{tis:}\left[\text{is:}\left[\begin{array}{lll}\text{private} & : & \left[\text{agenda=}\textcolor{red}{[]} \;:\; [\mathit{Action}]\right] \\ \text{shared} & : & \left[\text{qud=}\textcolor{red}{[]} \;:\; [\mathit{Question}]\right]\end{array}\right]\right]\right] \\
\text{tis} & : & \left[\text{is:}\left[\text{shared:}\left[\text{lu} \;:\; \left[\begin{array}{lll}\text{speaker=usr} & : & \mathit{DP} \\ \text{moves=}\textcolor{red}{\{\text{ask}(?A.\text{time\_next\_train}(\text{Bclna},A))\}} & : & \{\mathit{Move}\}\end{array}\right]\right]\right]\right] \\
\text{cond:} & & \left[\begin{array}{lll}\text{q} & : & \mathit{Question} \\ \text{c} & : & \text{member(ask(cond.q), tis.is.shared.lu.moves)}\end{array}\right]
\end{array}\right]
$$

$$
(\left[\text{tis:}\left[\text{is:}\left[\begin{array}{ll}\text{private:} & \left[\text{agenda=respond}(r\text{.cond.q})|r.\textcolor{red}{\text{prev}}\text{.tis.is.private.agenda} \;:\; [\mathit{Action}]\right] \\ \text{shared:} & \left[\text{qud=}r\text{.cond.q}|r.\textcolor{red}{\text{prev}}\text{.tis.is.shared.qud} \;:\; [\mathit{Question}]\right]\end{array}\right]\right]\right]
$$

The fixed point type

$$\left[\begin{array}{ll}\text{prev}: & \left[\text{tis}: \left[\text{is}: \left[\begin{array}{lll}\text{private} & : & \left[\text{agenda=[]} \;:\; [\mathit{Action}]\right]\\ \text{shared} & : & \left[\text{qud=[]} \;:\; [\mathit{Question}]\right]\end{array}\right]\right]\right]\\ \text{tis} : & \left[\text{is}: \left[\begin{array}{ll}\text{private}: & \left[\text{agenda=respond(cond.q)}|\text{prev.tis.is.private.agenda} \;:\; [\mathit{Action}]\right]\\ \text{shared}: & \left[\begin{array}{l}\text{qud=cond.q}|\text{prev.tis.is.shared.qud}:[\mathit{Question}]\\ \text{lu}: \left[\begin{array}{l}\text{speaker=usr}:\mathit{DP}\\ \text{moves=}\{\text{ask}(?A.\text{time\_next\_train(Bclna},A))\}:\{\mathit{Move}\}\end{array}\right]\end{array}\right]\end{array}\right]\right]\\ \text{cond}: & \left[\begin{array}{lll}\text{q} & : & \mathit{Question}\\ \text{c} & : & \text{member(ask(cond.q), tis.is.shared.lu.moves)}\end{array}\right]\end{array}\right]$$

Reasoning about unique solutions

$$
\left[\begin{array}{ll}
\text{prev} : & \left[\text{tis:} \left[\text{is:} \left[\begin{array}{lll} \text{private} & : & \left[\begin{array}{lll}\text{agenda=[]} & : & [\mathit{Action}]\end{array}\right] \\ \text{shared} & : & \left[\begin{array}{lll}\text{qud=[]} & : & [\mathit{Question}]\end{array}\right]\end{array}\right]\right]\right] \\
\text{tis} : & \left[\text{is:} \left[\begin{array}{ll} \text{private:} & \left[\begin{array}{lll}\text{agenda=respond(cond.q)}|\text{prev.tis.is.private.agenda} & : & [\mathit{Action}]\end{array}\right] \\ \text{shared:} & \left[\begin{array}{l}\text{qud=cond.q}|\text{prev.tis.is.shared.qud:}[\mathit{Question}] \\ \text{lu:} \left[\begin{array}{l}\text{speaker=usr:}\mathit{DP} \\ \text{moves=}\{\text{ask(?}A\text{.time\_next\_train(Bclna,}A\text{))}\}\text{:}\{\mathit{Move}\}\end{array}\right]\end{array}\right]\end{array}\right]\right] \\
\text{cond:} & \left[\begin{array}{lll}\text{q=?}A\text{.time\_next\_train(Bclna,}A\text{)} & : & \mathit{Question} \\ \text{c:member(ask(cond.q), tis.is.shared.lu.moves)} & & \end{array}\right]
\end{array}\right]
$$

Substituting values of manifest fields

$$\left[\begin{array}{lll}
\text{prev} & : & \left[\text{tis}: \left[\text{is}: \left[\begin{array}{lll} \text{private} & : & \left[\text{agenda}=[\,] : [\mathit{Action}]\right] \\ \text{shared} & : & \left[\text{qud}=[\,] : [\mathit{Question}]\right]\end{array}\right]\right]\right] \\
\text{tis} & : & \left[\text{is}: \left[\begin{array}{ll} \text{private}: & \left[\text{agenda}=[\text{respond}(?A.\text{time_next_train}(\text{Bclna},A))] : [\mathit{Action}]\right] \\ \text{shared}: & \left[\begin{array}{l} \text{qud}=[?A.\text{time_next_train}(\text{Bclna},A)]:[\mathit{Question}] \\ \text{lu}: \left[\begin{array}{l}\text{speaker}=\text{usr}:\mathit{DP}\\ \text{moves}=\{\text{ask}(?A.\text{time_next_train}(\text{Bclna},A))\}:\{\mathit{Move}\}\end{array}\right]\end{array}\right]\end{array}\right]\right] \\
\text{cond} & : & \left[\begin{array}{lll} \text{q}=?A.\text{time_next_train}(\text{Bclna},A) & : & \mathit{Question} \\ \text{c:member(ask(cond.q), tis.is.shared.lu.moves)} & & \end{array}\right]
\end{array}\right]$$

Garbage collection

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

$$
\left[\text{tis:}\left[\text{is:}\left[\begin{array}{ll}
\text{private:} & \left[\ \text{agenda=[respond(?}A\text{.time\_next\_train(Bclna,}A\text{))]} \quad : \quad [\textit{Action}]\ \right] \\
\text{shared:} & \left[\begin{array}{ll}
\multicolumn{2}{l}{\text{qud=[?}A\text{.time\_next\_train(Bclna,}A\text{)]:[}\textit{Question}\text{]}} \\
\text{lu:} & \left[\begin{array}{l}
\text{speaker=usr:}\textit{DP} \\
\text{moves=}\{\text{ask(?}A\text{.time\_next\_train(Bclna,}A\text{))}\}\text{:}\{\textit{Move}\}
\end{array}\right]
\end{array}\right]
\end{array}\right]\right]\right]
$$

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*

$$\left[\begin{array}{l}\text{tis:}\left[\begin{array}{l}\text{is:}\left[\begin{array}{l}\text{shared:}\left[\begin{array}{l}\text{com}=\left[\begin{array}{l}\text{x:}\mathit{Ind}\\ \text{c}_1\text{:named(x, “Sam”)}\\ \text{y:}\mathit{Ind}\\ \text{res:}\mathit{Rec}\\ \text{c}_2\text{:program(y)}\upharpoonright\text{res}\\ \text{c}_3\text{:(}r_1\text{:}\left[\begin{array}{l}\text{z:}\mathit{Ind}\\ \text{c:program(z)}\upharpoonright\text{res}\end{array}\right]\text{)}\left[\text{c:eq(}r_1\text{.z, y, }\mathit{Ind}\text{)}\right]\\ \text{p}=\lambda r_2 : \mathit{Rec}\ (\left[\begin{array}{l}\text{x:}\mathit{Ind}\\ \text{c}_4\text{:bug(x, y)}\end{array}\right]\text{):(}\mathit{Rec}\text{)}\mathit{RecType}\\ \text{c}_5\text{:say(x, p)}\end{array}\right]\text{:}\mathit{RecType}\end{array}\right]\end{array}\right]\end{array}\right]\end{array}\right]$$

After *Pat is looking for it*

$$\ldots \left[\begin{array}{l} \text{com=}\left[\begin{array}{l} \text{x}:\mathit{Ind} \\ \text{c}_1\text{:named(x, “Sam”)} \\ \text{y}:\mathit{Ind} \\ \text{res}:\mathit{Rec} \\ \text{c}_2\text{:program(y)}\upharpoonright\text{res} \\ \text{c}_3\text{:(}r_1\text{:}\left[\begin{array}{l}\text{z}:\mathit{Ind}\\ \text{c:program(z)}\upharpoonright\text{res}\end{array}\right]\text{)}\left[\text{c:eq(}r_1\text{.z, y, }\mathit{Ind}\text{)}\right] \\ \text{p}_1=\lambda r_2 : \mathit{Rec}\ (\left[\begin{array}{l}\text{x}:\mathit{Ind}\\ \text{c}_4\text{:bug(x, y)}\end{array}\right]):(\mathit{Rec})\mathit{RecType} \\ \text{c}_5\text{:say(x, p}_1\text{)} \\ \text{z}:\mathit{Ind} \\ \text{c}_6\text{:named(z, “Pat”)} \\ \text{p}_2=\lambda r : ?\ (\left[\text{y=?}:\mathit{Ind}\right]):(?)\mathit{RecType} \\ \text{c}_2\text{:seek(z, p}_2\text{)} \end{array}\right] :\mathit{RecType} \end{array}\right]$$

LU Dialogue Systems Lab

After anaphora resolution

$$\ldots \left[\begin{array}{l} \text{com}=\left[\begin{array}{l} \text{x}:\mathit{Ind} \\ \text{c}_1:\text{named}(\text{x}, \text{“Sam”}) \\ \text{y}:\mathit{Ind} \\ \text{res}:\mathit{Rec} \\ \text{c}_2:\text{program}(\text{y})\restriction\text{res} \\ \text{c}_3:(r_1:\left[\begin{array}{l}\text{z}:\mathit{Ind} \\ \text{c}:\text{program}(\text{z})\restriction\text{res}\end{array}\right])\left[\text{c}:\text{eq}(r_1.\text{z}, \text{y}, \mathit{Ind})\right] \\ \text{p}_1=\lambda r_2 : \mathit{Rec}\ (\left[\begin{array}{l}\text{x}:\mathit{Ind} \\ \text{c}_4:\text{bug}(\text{x}, \text{y})\end{array}\right]):(\mathit{Rec})\mathit{RecType} \\ \text{c}_5:\text{say}(\text{x}, \text{p}_1) \\ \text{z}:\mathit{Ind} \\ \text{c}_6:\text{named}(\text{z}, \text{“Pat”}) \\ \text{p}_2=\lambda r : \mathcal{F}(\text{p}_1)\ (\left[\text{y}=r.\text{x}:\mathit{Ind}\right]):(\mathcal{F}(\text{p}_1))\mathit{RecType} \\ \text{c}_2:\text{seek}(\text{z}, \text{p}_2) \end{array}\right]:\mathit{RecType} \end{array}\right]$$

i.e.,

$$
\ldots \left[\begin{array}{l}
\text{com}=\left[\begin{array}{l}
\text{x}:\mathit{Ind} \\
\text{c}_1:\text{named(x, “Sam”)} \\
\text{y}:\mathit{Ind} \\
\text{res}:\mathit{Rec} \\
\text{c}_2:\text{program(y)}{\restriction}\text{res} \\
\text{c}_3:(r_1:\left[\begin{array}{l}\text{z}:\mathit{Ind} \\ \text{c}:\text{program(z)}{\restriction}\text{res}\end{array}\right])\left[\text{c}:\text{eq}(r_1.\text{z, y, }\mathit{Ind})\right] \\
\text{p}_1=\lambda r_2 : \mathit{Rec}\ (\left[\begin{array}{l}\text{x}:\mathit{Ind} \\ \text{c}_4:\text{bug(x, y)}\end{array}\right]):(\mathit{Rec})\mathit{RecType} \\
\text{c}_5:\text{say(x, p}_1) \\
\text{z}:\mathit{Ind} \\
\text{c}_6:\text{named(z, “Pat”)} \\
\text{p}_2=\lambda r : \left[\begin{array}{l}\text{x}:\mathit{Ind} \\ \text{c}_4:\text{bug(x, y)}\end{array}\right]\ (\left[\text{y}=r.\text{x}:\mathit{Ind}\right]):(\left[\begin{array}{l}\text{x}:\mathit{Ind} \\ \text{c}_4:\text{bug(x, y)}\end{array}\right])\mathit{RecType} \\
\text{c}_2:\text{seek(z, p}_2)
\end{array}\right]:\mathit{RecType}
\end{array}\right]
$$

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?

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

Did a representative leave?

$$?(\lambda r : [] (\begin{bmatrix} \text{x} & : & \textit{Ind} \\ \text{c}_1 & : & \text{representative(x)} \\ \text{c}_2 & : & \text{leave(x)} \end{bmatrix}))$$

where $[]$ is a variant notation for *Rec*, the type of all records

Types of functions in these examples: $(\left[\text{x}:\textit{Ind}\right])$*RecType*, (*Rec*)*RecType*
Both these functions belong to the *polymorphic* type $\{$*Rec*$\}$*RecType* which can also be thought of as the type of *partial* functions from records to record types

‘?’, 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