

# A record type theoretic approach to information state update in dialogue – is this pragmatics?

Robin Cooper  
Göteborg University

The Department of Linguistics

COMPUTER SCIENCE AND ENGINEERING

# Dialogue Management

- a practical problem from the development of automated dialogue systems
- what should the system say next?
- theoretical development: dialogue games, dialogue moves (Lauri Carlson, Jonathan Ginzburg)

# Information State Update (ISU)

- Complex structure information states
  - gameboard
  - records
- non-monotonic updates
  - e.g. questions disappear from QUD (Questions under discussion)

cf. classical dynamic or update semantics

# ISU – a type-theoretic view

## Updates

$\text{info\_state}_1 \Rightarrow \text{info\_state}_2 \Rightarrow \text{info\_state}_3$

## Reasoning about updates

$\text{type}_1 \Rightarrow \text{type}_2 \Rightarrow \text{type}_3$

*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

# ISU – a record type-theoretic view

## Updates

$\text{record}_1 \Rightarrow \text{record}_2 \Rightarrow \text{record}_3$

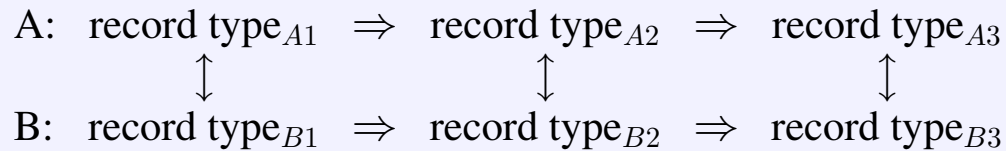
## Reasoning about updates

$\text{record type}_1 \Rightarrow \text{record type}_2 \Rightarrow \text{record type}_3$

$\text{record} \rightsquigarrow \text{information state}$

$\text{record type} \rightsquigarrow \text{information about an information state}$

# Alignment



## 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)

## A simple information state, $r$

Larsson, p. 56 (adapted)

private	=	<table style="border-collapse: collapse; margin-left: 20px;"> <tr> <td style="padding: 5px;">agenda</td> <td style="padding: 5px;">=</td> <td style="padding: 5px;">[]</td> </tr> <tr> <td style="padding: 5px;"></td> <td style="padding: 5px;"></td> <td style="padding: 5px;">[raise(?A.how(A), findout(?B.dest_city(B)), findout(?C.dep_city(C)), findout(?D.month(D), findout(?E.dep_date(E), findout(?F.class(F)), consultDB(?G.price(G))]</td> </tr> <tr> <td style="padding: 5px;">plan</td> <td style="padding: 5px;">=</td> <td style="padding: 5px;">findout(?D.month(D), findout(?E.dep_date(E), findout(?F.class(F)), consultDB(?G.price(G))]</td> </tr> <tr> <td style="padding: 5px;">bel</td> <td style="padding: 5px;">=</td> <td style="padding: 5px;">[]</td> </tr> </table>	agenda	=	[]			[raise(?A.how(A), findout(?B.dest_city(B)), findout(?C.dep_city(C)), findout(?D.month(D), findout(?E.dep_date(E), findout(?F.class(F)), consultDB(?G.price(G))]	plan	=	findout(?D.month(D), findout(?E.dep_date(E), findout(?F.class(F)), consultDB(?G.price(G))]	bel	=	[]
agenda	=	[]												
		[raise(?A.how(A), findout(?B.dest_city(B)), findout(?C.dep_city(C)), findout(?D.month(D), findout(?E.dep_date(E), findout(?F.class(F)), consultDB(?G.price(G))]												
plan	=	findout(?D.month(D), findout(?E.dep_date(E), findout(?F.class(F)), consultDB(?G.price(G))]												
bel	=	[]												
shared	=	<table style="border-collapse: collapse; margin-left: 20px;"> <tr> <td style="padding: 5px;">com</td> <td style="padding: 5px;">=</td> <td style="padding: 5px;">[]</td> </tr> <tr> <td style="padding: 5px;">qud</td> <td style="padding: 5px;">=</td> <td style="padding: 5px;">[?H.price(H)]</td> </tr> <tr> <td style="padding: 5px;">lu</td> <td style="padding: 5px;">=</td> <td style="padding: 5px;">[ speaker = usr moves = {ask(?H.price(H))}]</td> </tr> </table>	com	=	[]	qud	=	[?H.price(H)]	lu	=	[ speaker = usr moves = {ask(?H.price(H))}]			
com	=	[]												
qud	=	[?H.price(H)]												
lu	=	[ speaker = usr moves = {ask(?H.price(H))}]												

## *r* is of type:

Adapted from Larsson, p. 32

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

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

## Type theory with records (TTR)

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

the record:

$$\left[ \begin{array}{l} l_1 = a_1 \\ l_2 = a_2 \\ \dots \\ l_n = a_n \\ \dots \end{array} \right]$$

is of type:

$$\left[ \begin{array}{l} l_1 : T_1 \\ l_2 : T_2(l_1) \\ \dots \\ l_n : T_n(l_1, l_2, \dots, l_{n-1}) \end{array} \right]$$

*a man owns a donkey*

Record type:

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

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  $\text{own}(a, b)$

[⇐ contents](#)

# Manifest fields

Coquand, Pollack and Takeyama

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.

$[ x : T_a ]$

written for convenience as

$[ x=a : T ]$

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.

## *r* is also of type:

A singleton type using manifest fields

private :	agenda=[]	:	[Action]					
	[raise(?A.how(A)), findout(?B.dest_city(B)), findout(?C.dep_city(C)),							
	plan= findout(?D.month(D), findout(?E.dep_date(E)), findout(?F.class(F)), consultDB(?G.price(G))]	:	[Action]					
	bel=[]	:	RecType					
shared :	com=[]	:	RecType					
	qud=[?H.price(H)]	:	[Question]					
	lu	:	<table border="0" style="display: inline-table; vertical-align: middle;"> <tr> <td style="padding-right: 10px;">speaker=usr</td> <td style="padding-right: 10px;">:</td> <td>Participant</td> </tr> <tr> <td style="padding-right: 10px;">moves={ask(?H.price(H))}</td> <td style="padding-right: 10px;">:</td> <td>{Move}</td> </tr> </table>	speaker=usr	:	Participant	moves={ask(?H.price(H))}	:
speaker=usr	:	Participant						
moves={ask(?H.price(H))}	:	{Move}						

## Another type for $r$ :

Not a singleton type, but some fields are manifest

```

private : [
  agenda=[] : [Action]
           [raise(?A.how(A)),
            findout(?B.dest_city(B)),
            findout(?C.dep_city(C)),
            findout(?D.month(D)) : [Action]
            findout(?E.dep_date(E)),
            findout(?F.class(F)),
            consultDB(?G.price(G))]
  bel : RecType
]
shared : [
  com : RecType
  qud=[?H.price(H)] : [Question]
  lu : [
        speaker : Participant
        moves={ask(?H.price(H))} : {Move}
      ]
]

```

## Update rules – the intuition

If  $r_i : T_i$ , then  $r_{i+1} : T_{i+1}(r_i)$

## Update rules – the components

Update rules are triples:

- a record type,  $T$ , against which the current information state is checked
- a condition,  $\mathcal{C}$ , a function from records of type  $T$  to a record type
- a update function  $\mathcal{U}$ , a function from records of type  $\left[ \begin{array}{l} \text{cntxt} : T_{[\pi \rightarrow \text{cntxt}.\pi]} \\ \text{cond} : (\mathcal{C}@\text{cntxt})_{[\pi \rightarrow \text{cond}.\pi]} \end{array} \right]$  to a record type

$T_{[\pi \rightarrow \alpha]}$  is the result of replacing each path-reference  $\pi$  in dependent types in  $T$  with  $\alpha$ . We will suppress such subscripts in what follows.

## Applying update rules

Suppose that we have hypothesized a type  $T_{curr}$  for the current information state.

Then an update rule,  $\langle T, \mathcal{C}, \mathcal{U} \rangle$  will *fire* only if  $T_{curr} \sqsubseteq T$  and any  $r : T_{curr}$  will be such that  $\mathcal{C}@r$  is a non-empty type (for any model of basic types and predicates).

The type of the new information state is then computed to be:

$$\left[ \begin{array}{l} \text{prev} \\ \text{current} \end{array} : \left[ \begin{array}{l} \text{cntxt} : T_{curr} \\ \text{cond} : \mathcal{C}@prev.cntxt \end{array} \right] \right]$$

$$\text{current} : T_{curr} \uplus (\mathcal{U}@prev)$$

## Priority union with memory

$$T_1 \uplus T_2 = T_1' \cup T_2$$

where  $T_1'$  is the result of replacing all paths  $\pi$  in  $T_1$  which are

**either** of the form  $\text{prev}.\pi'$ , for some path  $\pi'$   
**or** shared with  $T_2$

with  $\text{prev}.\pi$  (including replacing all occurrences of  $\pi$  in dependent types with  $\text{prev}.\pi$ ).

## An update rule

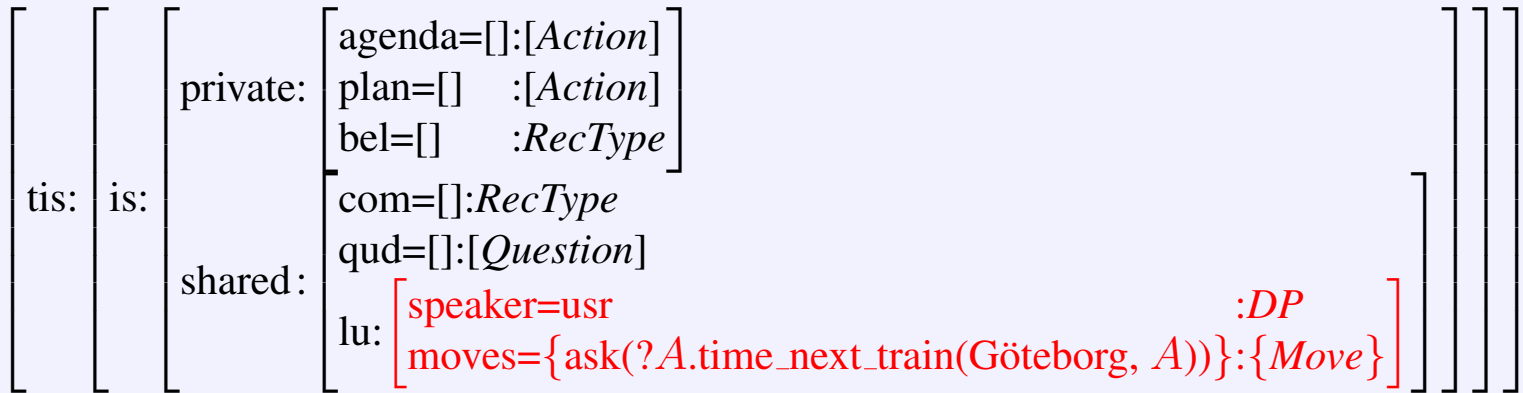
integrateUsrAsk, class: integrate, Larsson, p. 40

$$T = \left[ \text{tis:} \left[ \text{is:} \left[ \begin{array}{l} \text{private} : \left[ \text{agenda} : [Action] \right] \\ \text{shared} : \left[ \begin{array}{l} \text{lu} : \left[ \begin{array}{l} \text{speaker=usr} : DP \\ \text{moves} : \{Move\} \end{array} \right] \\ \text{qud} : [Question] \end{array} \right] \end{array} \right] \right] \right] \right]$$

$$C = \lambda r : T \left( \begin{array}{l} \text{q} : Question \\ \text{c} : \text{member}(\text{ask}(\text{cond.q}), r.\text{tis.is.shared.lu.moves) \end{array} \right)$$

$$U = \lambda r : \left[ \begin{array}{l} \text{cntxt} : T \\ \text{cond} : C@cntxt \end{array} \right] \left( \left[ \text{tis:} \left[ \text{is:} \left[ \begin{array}{l} \text{private:} \left[ \text{agenda}=\text{respond}(r.\text{cond.q}) | r.\text{cntxt.tis.is.private.agenda:} [Action] \right] \\ \text{shared:} \left[ \text{qud}=r.\text{cond.q} | r.\text{cntxt.tis.is.shared.qud:} [Question] \right] \end{array} \right] \right] \right] \right)$$

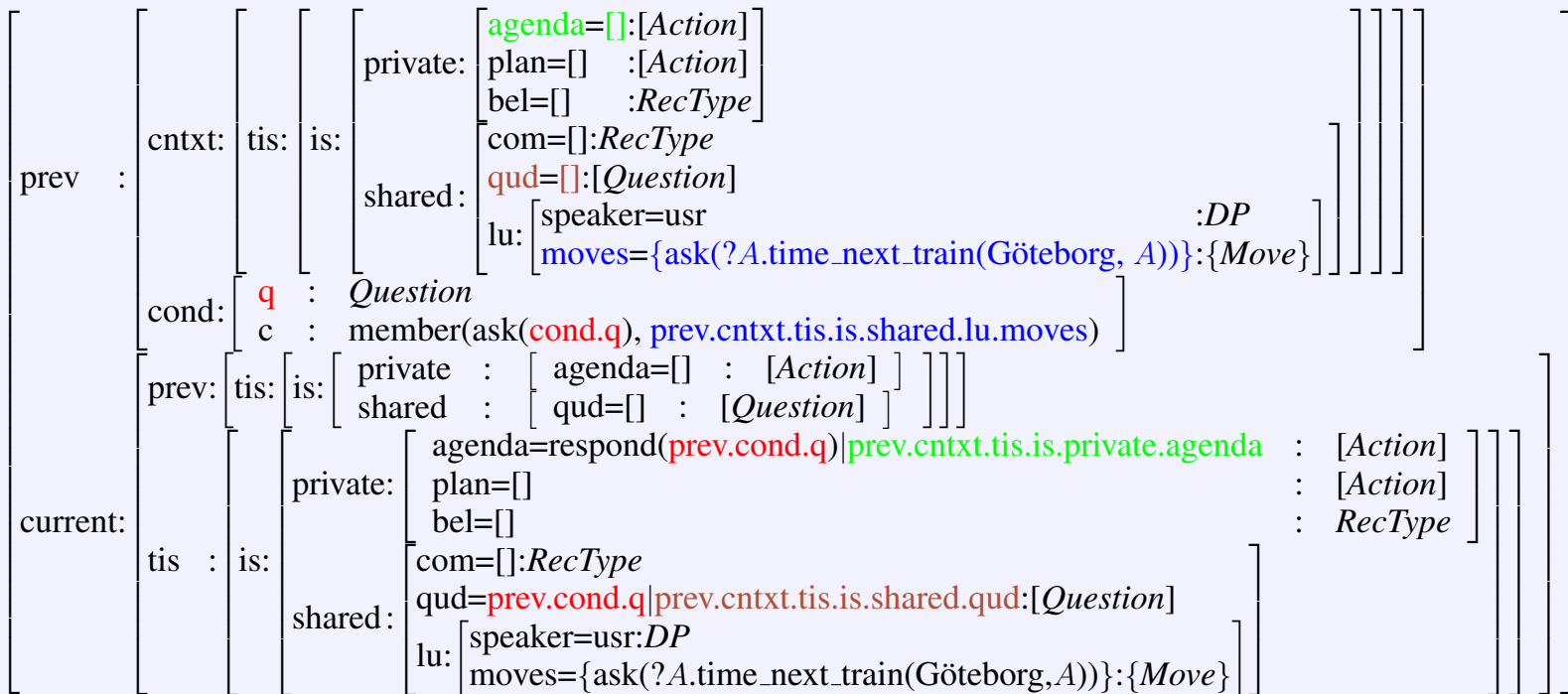
## Suppose current info state is of type



This is a subtype of  $T$ .

It also fulfils condition  $\mathcal{C}$ .

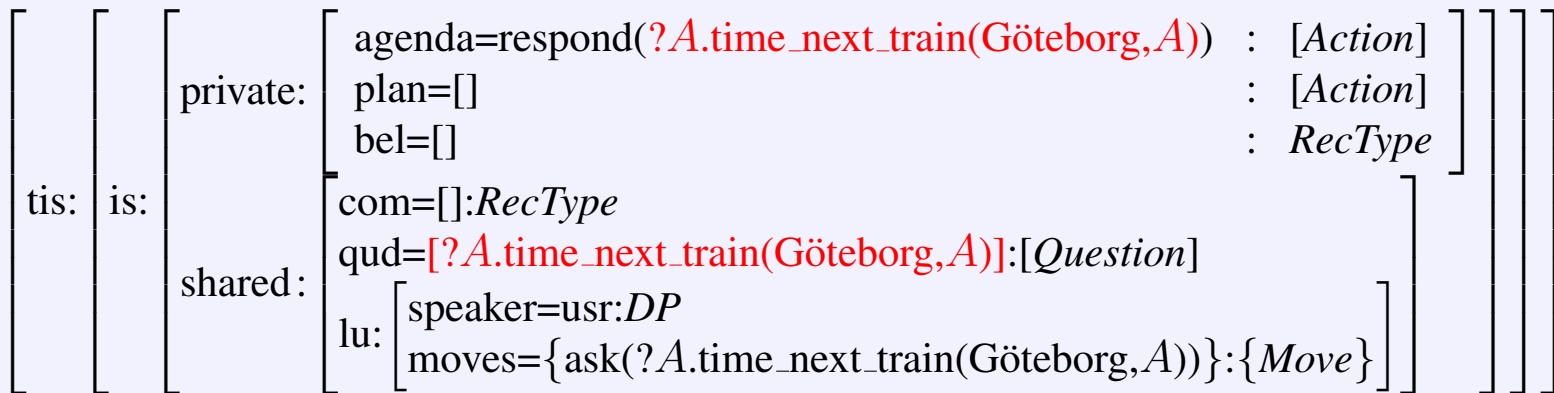
## Result of the update



⇐ contents

# Result of dependency simplification and garbage collection

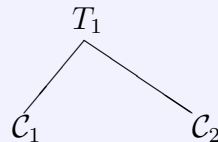
Projection on current



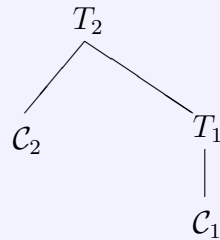
## A potential computational advantage

Update rules of the form  $\langle \textit{Type}, \textit{Condition}, \textit{Update} \rangle$  can be organized into trie-structures based on the type hierarchy so that a rule which will fire can be found with a minimum of type-checking.

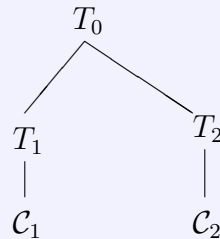
If we have two rules  $\langle T_1, C_1, U_1 \rangle$  and  $\langle T_2, C_2, U_2 \rangle$   
 if  $T_1 = T_2$ , then



else if  $T_1 \sqsubseteq T_2$ , then



else find the lowest common factor,  $T_0$ , of  $T_1$  and  $T_2$



# Dialogue management – semantics or pragmatics?

Is this an important question?