Issue Based Dialogue Management, part 2

Staffan Larsson
Dialogue Systems II
Master in Language Technology programme
University of Gothenburg

September 25, 2018

Basic issue-based dialogue management, cont'd

- Total Information State in IBiS1
- IBiS1 update module
- IBiS1 selection module
- Adapting IBiS1 to the travel information domain
- Sample dialogue with IBiS1
- Reraising issues and sharing information

QUD, ISSUES and ACTIONS

Question Accommodation

Outline

IBiS: an issue-based dialogue system

- Formulated using the ISU approach and implemented using TrindiKit, a toolkit for building dialogue systems using the ISU approach
- Explores and implements Issue-based dialogue management (Larsson 2002)
  - Based on Ginzburg's notion of a dialogue gameboard involving Questions Under Discussion (QUD)
  - Uses (mostly pre-scripted) dialogue plans
- Extends theory to more flexible dialogue
  - Multiple tasks, information sharing between tasks
  - Interactive Communication Management: feedback and grounding, sequencing
  - Question accommodation, re-raising, clarification

A sample inquiry-oriented dialogue

S> Welcome to the travel agency!
U> hello
U> price information please
S> How do you want to travel?
U> a flight please
S> What city do you want to go to?
U> paris
S> What city do you want to go from?
U> from london
S> What month do you want to leave?
U> in april
S> What day do you want to leave?
U> as cheap as possible
S> What day do you want to leave?
U> the fifth
S> What class did you have in mind?
U> as cheap as possible
S> It will cost 123 crowns.
U> bye

S> Thank you for your visit!

Outline

Total Information State

- Information State (IS)
- resource interface variables
- module interface variables

Dialogue state in IBiS (simplified)

$$\begin{bmatrix} \text{PRIVATE} : \begin{bmatrix} \text{AGENDA} : \text{OpenQueue(Action)} \\ \text{PLAN} : \text{OpenStack(PlanConstruct)} \\ \text{BEL} : \text{Set(Prop)} \end{bmatrix} \\ \text{SHARED} : \begin{bmatrix} \text{COM} : \text{Set(Prop)} \\ \text{QUD} : \text{OpenStack(Question)} \\ \text{LU} : \begin{bmatrix} \text{SPEAKER} : \text{Participant} \\ \text{MOVES} : \text{Set(Move)} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Resource interfaces

- DOMAIN (and in TDM ONTOLOGY)
  - goals and plans
  - actions
  - sorts
  - individuals and their sorts
  - predicates and the sorts of their argument
- DATABASE / DEVICE
  - device actions
  - device queries
- LEXICON (in TDM GRAMMAR)

Module interface variables

Handle the interaction between modules:

- INPUT : String
- LATEST_SPEAKER : Participant
- LATEST_MOVES : Set(Move)
- NEXT_MOVES : Set(Move)
- OUTPUT : String
- PROGRAM_STATE : ProgramState

Module interface variables, cont'd

- INPUT: Stores a string representing the latest user utterance. It is written to by the Input module and read by the Interpretation module
- LATEST-SPEAKER: The speaker of the latest utterance; read and written as the input variable, but also written to by the Output module (when the system made the last utterance).
- LATEST-MOVES: A representation of the moves performed in the latest utterance, as interpreted by the Interpret module.
- NEXT-MOVES: The next system moves to be performed, and the input to the Generate module
- OUTPUT: A string representing the system utterance, as generated by the Generate module
- PROGRAM-STATE: Whether IBiS should quit; either "run" or "quit". Read by the Control module.

Outline

IBiS1 update module

- Rules for dealing with ask and answer moves, based on Ginzburg's protocols for raising and resolving issues in dialogue, modified to a dialogue systems setting
- Some simple rules for handling greet and quit moves
- Rules for processing plans

Update rule for getting the latest utterance

- IBiS1 assumes perfect communication, in the sense that all system utterances are correctly understood by the user, and all interpretations of user utterances produced by the **interpret** module are correct.

RULE: **getLatestMove**

CLASS: grounding

PRE:
- $LATEST_MOVES=*Moves*
- $LATEST_SPEAKER=*DP*

EFF:
- IS.SHARED.LU.MOVES := *Moves*
- IS.SHARED-LU.SPEAKER := *DP*

Integrating ask moves

- To integrate an ask move with content q, make q topmost on QUD

RULE: **integrateSysAsk**
CLASS: integrate
PRE: { $IS.SHARED.LU.SPEAKER==sys
in($IS.SHARED.LU.MOVES, ask($Q$)) }
EFF: { push(IS.SHARED.QUD, Q) }

RULE: **integrateUsrAsk**
CLASS: integrate
PRE: { $IS.SHARED.LU.SPEAKER==usr
in($IS.SHARED.LU.MOVES, ask($Q$)) }
EFF: { push(IS.SHARED.QUD, Q)
push(IS.PRIVATE.AGENDA, respond(Q)) }

Integrating answer moves

RULE: **integrateAnswer**

CLASS: integrate

PRE:
- in($IS.SHARED.LU.MOVES, answer(A))
- fst($IS.SHARED.QUD, Q)
- $DOMAIN :: relevant($A$, Q)

EFF:
- ! $DOMAIN :: combine($Q$, A, P)
- add(IS.SHARED.COM, P)

QUD downdate

If Q is on /SHARED/QUD and there is a proposition p in /SHARED/COM such that P resolves Q, remove Q from QUD

RULE: **downdateQUD**

CLASS: downdate_qud

PRE:
- fst($IS.SHARED.QUD, Q)
- in($IS.SHARED.COM, P)
- $DOMAIN :: resolves($P$, Q)

EFF:
- pop(IS.SHARED.QUD)

Integrating greet and quit moves

RULE: **integrateGreet**
CLASS: integrate
PRE: { in($IS.SHARED.LU.MOVES, greet)
EFF: {

RULE: **integrateUsrQuit**
CLASS: integrate
PRE: { $IS.SHARED.LU.SPEAKER==usr
in($IS.SHARED.LU.MOVES, quit)
EFF: { push(IS.PRIVATE.AGENDA, quit)

RULE: **integrateSysQuit**
CLASS: integrate
PRE: { $IS.SHARED.LU.SPEAKER==sys
in($IS.SHARED.LU.MOVES, quit)
EFF: { PROGRAM_STATE := quit

Finding and loading a plan

When integrating a user ask move with content Q, the action respond(Q) is pushed on the agenda, thus enabling this rule to trigger and load a plan for dealing with Q:

RULE: **findPlan**

CLASS: find_plan

PRE:
- fst($IS.PRIVATE.AGENDA, respond(Q))
- $DOMAIN :: plan($Q$, *Plan*)
- not in($IS.PRIVATE.BEL, P) and $DOMAIN :: resolves($P$, Q)

EFF:
- pop(IS.PRIVATE.AGENDA)
- set(IS.PRIVATE.PLAN, *Plan*)

Executing the plan

One should not ask a question whose answer is already shared. On an individual level, each DP should make sure to not ask such questions.

RULE: **removeFindout**

CLASS: exec_plan

PRE:
- fst($IS.PRIVATE.PLAN, findout(Q))
- in($IS.SHARED.COM, P)
- $DOMAIN :: resolves($P$, Q)

EFF:
- pop(IS.PRIVATE.PLAN)

Executing the plan, cont'd

If there is a consultDB action topmost on the plan, this rule will trigger a database search. (Replaced by corresponding rules for device queries and actions in TDM.)

RULE: **exec_consultDB**

CLASS: exec_plan

PRE: { fst($IS.PRIVATE.PLAN, consultDB(Q))

EFF:
- ! $IS.SHARED.COM=$COM$
- ! $DATABASE :: consultDB($Q$, COM, P)
- add(IS.PRIVATE.BEL, P)
- pop(IS.PRIVATE.PLAN)

Update algorithm for IBiS1

if not LATEST_MOVES == failed
then ⟨ apply clear(IS.PRIVATE.AGENDA),
getLatestMove,
integrate,
try downdate_qud,
try load_plan,
repeat exec_plan⟩

Outline

IBiS1 selection module

Two steps:

- First, select an agenda item by either selecting to respond to an issue or taking the topmost action from the plan and move it to the agenda
- Second, select a move which realizes this action.

Selecting an action from the plan

RULE: **selectFromPlan**

CLASS: select_action

PRE: { is_empty($IS.PRIVATE.AGENDA)
fst($IS.PRIVATE.PLAN, *Action*)

EFF: { push(IS.PRIVATE.AGENDA, *Action*)

- Note that this rule does not remove the action from the plan
- when this should be done depends on the action; for example, an action findout(Q) is not removed until Q is resolved.

Selecting the ask move

RULE: **selectAsk**

CLASS: select_move

PRE:
- fst($IS.PRIVATE.AGENDA, findout(Q)) or
- fst($IS.PRIVATE.AGENDA, raise(Q))

EFF:
- add(NEXT_MOVES, ask(Q))
- if_do(fst($IS.PRIVATE.PLAN, raise($Q$)), pop(IS.PRIVATE.PLAN))

Selecting to respond to a question

- We will assume that for a DP A to select an answer move with (propositional) content p, there must be a question Q topmost on QUD and a proposition P which is relevant to Q such that A privately believes that P is true.
- Also, p must not be in what A takes to be the shared commitments; if it were, the answer move would be irrelevant.

RULE: **selectRespond**

CLASS: select_action

PRE:
- is_empty($IS.PRIVATE.AGENDA)
- is_empty($IS.PRIVATE.PLAN)
- fst($IS.SHARED.QUD, Q)
- in($IS.PRIVATE.BEL, P)
- not in($IS.SHARED.COM, P)
- $DOMAIN :: relevant($P$, Q)

EFF:
- push(IS.PRIVATE.AGENDA, respond(Q))

Selecting the answer move

Given that a respond(Q) action is on the agenda, and the system knows a relevant answer P to Q which is not yet shared, this rule selects an answer move with content P to be pushed on NEXT_MOVES.

RULE: **selectAnswer**

CLASS: select_move

PRE:
- fst($IS.PRIVATE.AGENDA, respond(Q))
- in($IS.PRIVATE.BEL, P)
- not in($IS.SHARED.COM, P)
- $DOMAIN :: relevant($P$, Q)

EFF:
- add(NEXT_MOVES, answer(P))

Outline

Domain: dialogue plans

ISSUE : $?x.\text{price}(x)$

PLAN: $\langle$

findout($?x.\text{how}(x)$),
findout($?x.\text{dest\_city}(x)$),
findout($?x.\text{depart-city}(x)$),
findout($?x.\text{depart-month}(x)$),
findout($?x.\text{depart-day}(x)$),
raise($?x.\text{class}(x)$),
consultDB($?x.\text{price}(x)$)

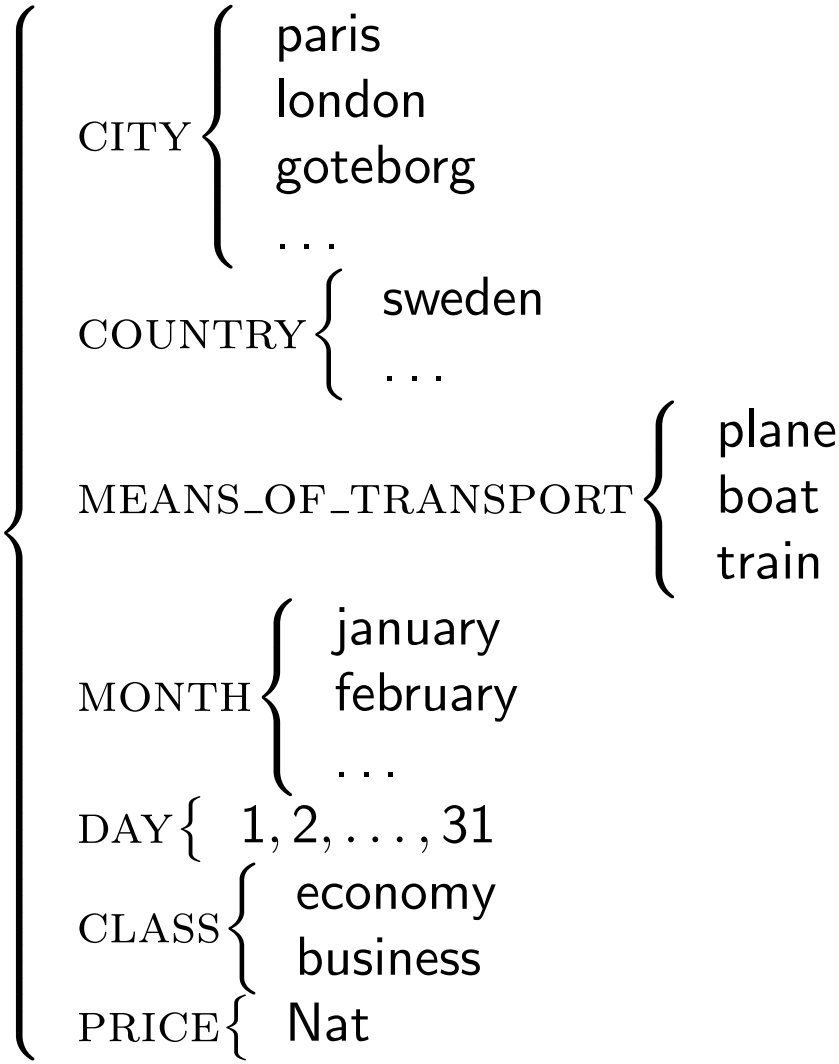
$\rangle$

Domain: dialogue plans, cont'd

ISSUE : ?need_visa

PLAN: ⟨
  findout(?x.dest_city(x)),
  findout(?x.citizenship(x)),
  consultDB(?need_visa),
⟩

Domain/ontology: sorts and individuals



Domain/ontology: sortal restrictions

proposition	**restriction**
dest_city(X)	$X \in$ CITY
depart_city(X)	$X \in$ CITY
how(X)	$X \in$ MEANS_OF_TRANSPORT
depart_month(X)	$X \in$ MONTH
depart_day(X)	$X \in$ DAY
class(X)	$X \in$ CLASS
citizenship(X)	$X \in$ COUNTRY
price(X)	$X \in$ PRICE

Outline

Sample dialogue with IBiS1

selectOther

{ add(NEXT_MOVES, greet)

S> Welcome to the travel agency!

getLatestMove

{ set(IS.SHARED.LU.MOVES, set([greet]))
{ set(IS.SHARED.LU.SPEAKER, sys)

integrateGreet

$$\begin{bmatrix} \text{PRIVATE} = \begin{bmatrix} \text{AGENDA} = \langle\rangle \\ \text{PLAN} = \langle\rangle \\ \text{BEL} = \{\} \end{bmatrix} \\ \text{SHARED} = \begin{bmatrix} \text{COM} = \{\} \\ \text{QUD} = \langle\rangle \\ \text{LU} = \begin{bmatrix} \text{SPEAKER} = \text{sys} \\ \text{MOVES} = \{ \text{ greet } \} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

U> `price information please`

getLatestMove

$$\left\{\begin{array}{l} \text{set}(\text{IS.SHARED.LU.MOVES}, \text{set}([\text{ask}(?A.\text{price}(A))])) \\ \text{set}(\text{IS.SHARED.LU.SPEAKER}, \text{usr}) \end{array}\right.$$

integrateUsrAsk

$$\left\{\begin{array}{l} \text{push}(\text{IS.SHARED.QUD}, ?A.\text{price}(A)) \\ \text{push}(\text{IS.PRIVATE.AGENDA}, \text{respond}(?A.\text{price}(A))) \end{array}\right.$$

findPlan

$$\left\{\begin{array}{l} \text{pop}(\text{IS.PRIVATE.AGENDA}) \\ \text{set}(\text{IS.PRIVATE.PLAN}, \text{stack}([\text{findout}(?C.\text{how}(C)), \text{findout}(?D.\text{dest\_city}(D)), \ldots])) \end{array}\right.$$

$$
\left[\begin{array}{lll}
\text{PRIVATE} & = & \left[\begin{array}{lll}
\text{AGENDA} & = & \langle \\
\text{PLAN} & = & \left\langle \begin{array}{l} \text{findout}(?A.\text{how}(A)) \\ \text{findout}(?B.\text{dest\_city}(B)) \\ \text{findout}(?C.\text{dept\_city}(C)) \\ \text{findout}(?D.\text{month}(D)) \\ \text{findout}(?E.\text{dept\_day}(E)) \\ \text{raise}(?F.\text{class}(F)) \\ \text{consultDB}(?G.\text{price}(G)) \end{array} \right\rangle \\
\text{BEL} & = & \{\}
\end{array}\right] \\
\text{SHARED} & = & \left[\begin{array}{lll}
\text{COM} & = & \{\} \\
\text{QUD} & = & \langle\ ?H.\text{price}(H)\ \rangle \\
\text{LU} & = & \left[\begin{array}{lll} \text{SPEAKER} & = & \text{usr} \\ \text{MOVES} & = & \{\ \text{ask}(?H.\text{price}(H))\ \} \end{array}\right]
\end{array}\right]
\end{array}\right]
$$

selectFromPlan

{ push(IS.PRIVATE.AGENDA, raise(?A.how(A)))

selectAsk

{ add(NEXT_MOVES, ask(?A.how(A)))

{ if_do(fst($IS.PRIVATE.PLAN, raise(?$A$.how($A$))), pop(IS.PRIVATE.PLAN))

S> How do you want to travel?

getLatestMove

integrateSysAsk

{ push(IS.SHARED.QUD, ?A.how(A))

$$
\left[\begin{array}{lll}
\text{PRIVATE} & = & \left[\begin{array}{lll}
\text{AGENDA} & = & \langle\rangle \\
\text{PLAN} & = & \left\langle \begin{array}{l} \text{findout}(?A.\text{how}(A)) \\ \text{findout}(?A.\text{dest\_city}(A)) \\ \text{findout}(?B.\text{dept\_city}(B)) \\ \text{findout}(?C.\text{month}(C)) \\ \text{findout}(?D.\text{dept\_day}(D)) \\ \text{raise}(?E.\text{class}(E)) \\ \text{consultDB}(?F.\text{price}(F)) \end{array} \right\rangle \\
\text{BEL} & = & \{\}
\end{array}\right] \\
\text{SHARED} & = & \left[\begin{array}{lll}
\text{COM} & = & \{\} \\
\text{QUD} & = & \left\langle \begin{array}{l} ?G.\text{how}(G) \\ ?H.\text{price}(H) \end{array} \right\rangle \\
\text{LU} & = & \left[\begin{array}{lll} \text{SPEAKER} & = & \text{sys} \\ \text{MOVES} & = & \{\ \text{ask}(?G.\text{how}(G))\ \} \end{array}\right]
\end{array}\right]
\end{array}\right]
$$

U> `a flight`

getLatestMove

integrateAnswer

$$\begin{cases} ! \ \$\text{DOMAIN} :: \text{combine}(?A.\text{how}(A), \text{plane}, B) \\ \text{add}(\text{IS.SHARED.COM}, B) \end{cases}$$

removeFindout

downdateQUD

$$\left[\begin{array}{lll} & \text{AGENDA} & = \ \langle\rangle \\ \text{PRIVATE} = & \text{PLAN} & = \left\langle \begin{array}{l} \text{findout}(?A.\text{dest\_city}(A)) \\ \text{findout}(?B.\text{dept\_city}(B)) \\ \text{findout}(?C.\text{month}(C)) \\ \text{findout}(?D.\text{dept\_day}(D)) \\ \text{raise}(?E.\text{class}(E)) \\ \text{consultDB}(?F.\text{price}(F)) \end{array} \right\rangle \\ & \text{BEL} & = \ \{\} \end{array}\right.$$

$$\left.\begin{array}{lll} \text{SHARED} = & \text{COM} & = \ \{ \ \text{how}(\text{plane}) \ \} \\ & \text{QUD} & = \ \langle \ ?G.\text{price}(G) \ \rangle \\ & \text{LU} & = \left[\begin{array}{lll} \text{SPEAKER} & = & \text{usr} \\ \text{MOVES} & = & \{ \ \text{answer}(\text{plane}) \ \} \end{array}\right] \end{array}\right]$$

selectFromPlan
selectAsk
S> What city do you want to go to?
U> paris
S> What city do you want to go from?
U> london
S> What month do you want to leave?
U> in april
S> What day do you want to leave?
U> the fifth

selectAsk

$$\left\{ \begin{array}{l} \text{add}(\text{NEXT\_MOVES}, \text{ask}(?A.\text{how}(A))) \\ \text{if\_do}(\text{fst}(\$\text{IS.PRIVATE.PLAN}, \text{raise}(?A.\text{class}(A))), \text{pop}(\text{IS.PRIVATE.PLAN})) \end{array} \right.$$

S> `What class did you have in mind?`

getLatestMove

integrateSysAsk

$$\left\{ \text{push}(\text{IS.SHARED.QUD}, ?E.\text{class}(E)) \right.$$

$$\left[\begin{array}{lll} \text{PRIVATE} & = & \left[\begin{array}{lll} \text{AGENDA} & = & \langle\rangle \\ \text{PLAN} & = & \langle\ \text{consultDB}(?F.\text{price}(F))\ \rangle \\ \text{BEL} & = & \{\} \end{array} \right] \\ \text{SHARED} & = & \left[\begin{array}{lll} \text{COM} & = & \left\{ \begin{array}{l} \text{dept\_day(fifth)} \\ \text{month(april)} \\ \text{dept\_city(london)} \\ \text{dest\_city(paris)} \\ \text{how(plane)} \end{array} \right\} \\ \text{QUD} & = & \left\langle \begin{array}{l} ?E.\text{class}(E) \\ ?F.\text{price}(F) \end{array} \right\rangle \\ \text{LU} & = & \left[\begin{array}{lll} \text{SPEAKER} & = & \text{usr} \\ \text{MOVES} & = & \{\ \text{ask}(?E.\text{class}(E))\ \} \end{array} \right] \end{array} \right] \end{array} \right]$$

U> as cheap as possible

getLatestMove

integrateAnswer

downdateQUD

exec_consultDB

$$\begin{cases} !\ \$\text{IS.SHARED.COM}{=}B \\ !\ \$\text{DATABASE} :: \text{consultDB}(?A.\text{price}(A),\ B,\ C) \\ \text{add}(\text{IS.PRIVATE.BEL},\ C) \\ \text{pop}(\text{IS.PRIVATE.PLAN}) \end{cases}$$

$$\left[\begin{array}{lll} & & \left[\begin{array}{lll} \text{AGENDA} & = & \langle\rangle \\ \text{PLAN} & = & \langle\rangle \\ \text{BEL} & = & \{\ \text{price(123)}\ \} \end{array}\right] \\ \text{PRIVATE} & = & \\ & & \left[\begin{array}{lll} \text{COM} & = & \left\{\begin{array}{l} \text{class(economy)} \\ \text{dept\_day(fifth)} \\ \text{month(april)} \\ \text{dept\_city(london)} \\ \text{dest\_city(paris)} \\ \text{how(plane)} \end{array}\right\} \\ \text{QUD} & = & \langle\ ?A.\text{price}(A)\ \rangle \\ \text{LU} & = & \left[\begin{array}{lll} \text{SPEAKER} & = & \text{usr} \\ \text{MOVES} & = & \{\ \text{answer(class(economy))}\ \} \end{array}\right] \end{array}\right] \\ \text{SHARED} & = & \end{array}\right]$$

selectRespond

{ push(IS.PRIVATE.AGENDA, respond(?A.price(A)))

selectAnswer

{ add(NEXT_MOVES, answer(price(123)))

S> `The price is 123 crowns.`

getLatestMove

integrateAnswer

downdateQUD

$$
\left[\begin{array}{lll}
\text{PRIVATE} & = & \left[\begin{array}{lll}
\text{AGENDA} & = & \langle\rangle \\
\text{PLAN} & = & \langle\rangle \\
\text{BEL} & = & \{\ \text{price(123)}\ \}
\end{array}\right] \\
\text{SHARED} & = & \left[\begin{array}{lll}
\text{COM} & = & \left\{\begin{array}{l}
\text{price(123)} \\
\text{class(economy)} \\
\text{dept\_day(fifth)} \\
\text{month(april)} \\
\text{dept\_city(london)} \\
\text{dest\_city(paris)} \\
\text{how(plane)}
\end{array}\right\} \\
\text{QUD} & = & \langle\rangle \\
\text{LU} & = & \left[\begin{array}{lll}
\text{SPEAKER} & = & \text{sys} \\
\text{MOVES} & = & \{\ \text{answer(price(123))}\ \}
\end{array}\right]
\end{array}\right]
\end{array}\right]
$$

Outline

Reraising issues and sharing information

- IBiS1 is able to answer any number of queries which the resources are set up to handle, as long as each issue is resolved before moving on to the next one.
- If the user asks q and then asks q' before q has been resolved, IBiS1 will forget its plan for dealing with q and instead load the plan for dealing with q'.
- When q' has been resolved, IBiS1 will wait for a new question from the user.
- However, q is still on QUD, and by adding a simple rule we could make IBiS1 reload the plan for dealing with q and get back to work on it.

Recovering the plan

This rule will pick up any question Q lying around on QUD when the plan and agenda is empty, check if there is a plan for resolving it, and if so load this plan.

RULE: **recoverPlan**

CLASS: exec_plan

PRE:
- fst($IS.SHARED.QUD, Q)
- is_empty($IS.PRIVATE.AGENDA)
- is_empty($IS.PRIVATE.PLAN)
- $DOMAIN :: plan($Q$, *Plan*)

EFF:
- set(IS.PRIVATE.PLAN, *Plan*)

Recovering the plan, cont'd

- However, this solution has a problem.
- If, when dealing with a question q, the system asks a question q_u and the user does not answer this question but instead raises a new question q_1, both q and q_u will remain on QUD when q_1 has been resolved.
- Now, if the user simply answers q_u immediately after q_1 has been resolved, everything is fine and the system will reload the plan for dealing with q.
- However, if the user does not answer q_u, this question will be topmost on QUD and block **recoverPlan** from triggering.
- Because of the simple structure of QUD, IBiS1 sees no reason to ask q_u again; after all, it is already under discussion, and the user is expected to provide an answer.
- The **reraiseIssue** rule on the next slide provides a solution to this problem

Rule for reraising issues

This rule reraises any questions on QUD which are not associated with any plan (i.e., which have been raised previously by the system).

RULE: **reraiseIssue**

CLASS: select_action

PRE:
- fst($IS.SHARED.QUD, Q)
- not $DOMAIN :: plan($Q$, _*SomePlan*)

EFF:
- push(IS.PRIVATE.AGENDA, raise(Q))

Sample dialogue: reraising issues

S> Welcome to the travel agency!
U> price information please
S> How do you want to travel?
U> plane
S> What city do you want to go to?
U> paris
S> What city do you want to go from?
U> do i need a visa
getLatestMove
integrateUsrAsk
findPlan
removeFindout
selectFromPlan
selectAsk

S> What country are you from?
U> sweden
S> Yes, you need a Visa.
U>
reraiseIssue
{ push(IS.PRIVATE.AGENDA, raise(?A.dept_city(A)))
selectAsk
S> What city do you want to go from?

U> `london`
getLatestMove
integrateAnswer
downdateQUD
recoverPlan
removeRaise
removeFindout
removeFindout

$$
\left[\begin{array}{lll}
\text{PRIVATE} & = & \left[\begin{array}{lll}
\text{AGENDA} & = & \langle\rangle \\
\text{PLAN} & = & \left\langle \begin{array}{l} \text{findout}(?A.\text{month}(A)) \\ \text{findout}(?B.\text{dept\_day}(B)) \\ \text{findout}(?C.\text{class}(C)) \\ \text{consultDB}(?D.\text{price}(D)) \end{array} \right\rangle \\
\text{BEL} & = & \{\ \text{need\_visa}\ \}
\end{array}\right] \\
\text{SHARED} & = & \left[\begin{array}{lll}
\text{COM} & = & \left\{ \begin{array}{l} \text{dept\_city(london)} \\ \text{need\_visa} \\ \text{citizenship(sweden)} \\ \text{dest\_city(paris)} \\ \text{how(plane)} \end{array} \right\} \\
\text{QUD} & = & \langle\ ?E.\text{price}(E)\ \rangle \\
\text{LU} & = & \left[\begin{array}{lll} \text{SPEAKER} & = & \text{usr} \\ \text{MOVES} & = & \{\ \text{answer(london)}\ \} \end{array}\right]
\end{array}\right]
\end{array}\right]
$$

selectFromPlan
selectAsk
S> `What month do you want to leave?`

Incidentally, this dialogue also demonstrates information sharing between dialogue plans; when the user asks about visa, the system already knows what the destination city is and thus does not ask this again.

Outline

A problem with QUD

- Example
  - A: Who won the world cup?
  - B: Where?
  - A: in Italy
  - B: Are you sure it wasn't in France?
  - A: Yes
  - B: Where did you say it was again?
  - A: Italy
  - B: Germany
- If QUD=$< q_1, q_2 >$ and q1 is resolved, q_2 is available for resolution of short answers
- Takes no account of how many turns since q_2 was raised
- But: short answers a long distance away from the question are not as easily processed as an adjacent answer

Issues and QUD

- We extend Ginzburg's DGB by adding ISSUES of type Stack(Question)
- ISSUES contains all raised but unresolved questions
- ISSUES determines relevance of user answers
- QUD used for resolving short answers
  - Questions drop off QUD after N turns
  - A short answer to a question that's on ISSUES but not QUD requires adjusting QUD by copying a question on ISSUES

Action-oriented dialogue

- In many applications, we do not only search for information but we want to get things done (calling someone, playing a song, ..)
- Genre: Action-oriented dialouge
- New moves: request(α), report(α, Status) where α is an action
- New field SHARED.ACTIONS of type OpenStack(Action) has a similar role to ISSUES
- Integrating a move request(α) results in pushing α on SHARED.ACTIONS
- Integrating a move report(α, done) results in α being popped off ACTIONS

Extended dialogue state in IBiS (simplified)

$$
\begin{bmatrix}
\textsc{private} : \begin{bmatrix}
\textsc{agenda} : \mathsf{OpenQueue(Action)} \\
\textsc{plan} : \mathsf{OpenStack(PlanConstruct)} \\
\textsc{bel} : \mathsf{Set(Prop)}
\end{bmatrix} \\
\textsc{shared} : \begin{bmatrix}
\textsc{com} : \mathsf{Set(Prop)} \\
\textsc{issues} : \mathsf{OpenStack(Question)} \\
\textsc{qud} : \mathsf{OpenStack(Question)} \\
\textsc{actions} : \mathsf{OpenStack(Action)} \\
\textsc{lu} : \begin{bmatrix}
\textsc{speaker} : \mathsf{Participant} \\
\textsc{moves} : \mathsf{Set(Move)}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix}
$$

Action oriented dialogue (TV)

- S: What can I do for you?
- U: Change channel
  - request(change_channel)
- S: What channel?
- U: Five
- S: The channel was changed
  - report(change_channel, done)

Outline

Accommodation

- Lewis (1979): If someone says something at t which requires X to be in the conversational scoreboard, and X is not in the scoreboard at t, then (under certain conditions) X will become part of the scoreboard at t
- Example
  - I'm sorry I'm late, my car broke down
  - >> The speaker has a car
  - B did not know that A had a car, but accommodates and adds "A has a car" to (her view of) the conversational scoreboard
- Conversational scoreboard is information assumed to be shared by all dialogue participants; a.k.a. dialogue gameboard, (shared) dialogue state, common ground
- Presuppositions of an utterance are required to be in the conversational scoreboard

Accommodation, cont'd

- The concept of accommodation enables a theory of how deviant usages are brought back into line by a cooperative interlocutor
- A "tacit extension" is made to the discourse context to allow for an update with otherwise unfulfilled presuppositions
- What is accommodated should be non-controversial and consistent with all the propositions already placed in the context; otherwise, some kind of repair may be needed
  - I'm sorry I'm late, my fire-engine broke down
  - >> The speaker has a fire-engine
  - I'm sorry I'm late, my spaceship broke down
- New information can be conveyed by presuppositions
- Accommodation is essentially a mechanism for updating the discourse context with new, uncontroversial assumptions

Typical human-human dialogue

S(alesman), C(ustomer)

- S: hi
- C: flights to paris
- S: when do you want to travel?
- C: april, as cheap as possible

Accommodation

- Accommodation has been applied to referents and propositions, as parts of the conversational scoreboard / information state
- Question accommodation
  - If questions are part of the information state, they too can be accommodated
  - If the latest move was an answer, and there is an action in the plan to ask a matching question, then put that question on issues (and QUD if it is a short answer)
  - Requires that the number of possible matching questions is not too large (or can be narrowed down by asking clarification question)

General strategy: question accommodation

- Issue accommodation
- QUD accommodation
- Dependent issue accommodation

Issue accommodation: PLAN ⇒ ISSUES

- Name: AccommodatePlanToIssues
- If
  - in(SHARED.lu.moves, answer(A))
  - no Q in ISSUES such that DOMAIN.relevant(A, Q)
- then
  - find findout(Q) in PLAN such that DOMAIN.relevant(A, Q)
  - push Q on ISSUES
- Used when prevously unraised question (available in plan) is answered using a short or full answer
- Takes care of overanswering and other-answering

QUD accommodation: ISSUES ⇒ QUD

- AccommocateIssueToQud
- If
  - in(SHARED.lu.moves, answer(A))
  - A is a short answer
  - no Q in QUD s.t. DOMAIN.relevant(A, Q)
- then
  - find Q in ISSUES s.t. relevant(A, Q)
  - push Q on QUD
  - raise Q in ISSUES (make Q topmost)
- Used when a previously raised question has dropped off QUD, but is answered using a short answer
- Previously unraised question is answered using short answer (in which case it needs to be preceded by accommodatePlanToIssues)
- Takes care of e.g. corrections by user (part of grounding, see Lecture 3)

Dependent issue accommodation: DOMAIN ⇒ ISSUES + PLAN

- AccommodateDependent
- If
  - LM=answer(A)
  - no Q in ISSUES s.t. DOMAIN.relevant(A, Q)
  - no findout(Q) in PLAN s.t. relevant(A, Q)
- then
  - find plan Π for some Q' in DOMAIN s.t. findout(Q) or raise(Q) in Π and DOMAIN.relevant(A, Q)
  - push Q' on ISSUES
  - set PLAN to Π
- Used when previously unraised question, unavailable in PLAN but available in some plan in the domain knowledge resource, is answered using full or short answer (a.k.a. "task accomodation")
- Takes care of Task recognition
- A variant of this rule instead generates a clarification question if there are several matching plans (Task clarification)

Sample dialogue: accommodation

- S: Welcome to the travel agency.
- U: From London to Paris in April
  - Not relevant to any question that has been raised, or to any current plan
  - Look in domain knowledge for a plan (for dealing with some question Q) with matching questions
  - Load this plan, push Q on ISSUES (dependent accommodation)
  - Find in PLAN the question(s) matching the user's answer, and push them onto ISSUES (issue accommodation)
  - Integrate answer (requres matching question on ISSUES)
- S: How do you want to travel?
  - SHARED.ISSUES=<?x.how(x), ?x.price(x)>

Task clarification (VCR application)

- S: What can I do for you?
- U: Channel five
  - Answer matches questions in several plans $\Pi_1, \ldots \Pi_n$ with goals $\alpha_1, \ldots, \alpha_n$
  - Construct a clarification question ?{issue(α_1), ..., issue(α_n)} (task clarification)
- S: Do you want to add a program or change channel?
- U: Change channel
- U: Yes
- S: The channel was changed

Task clarification with grounding (VCR application)

- S: What can I do for you?
- U: Channel five
  - Answer matches questions in several plans $\Pi_1, \ldots \Pi_n$ with goals $\alpha_1, \ldots, \alpha_n$
  - Construct a clarification question ?{issue(α_1), ..., issue(α_n)} (task clarification)
- S: Channel five. I don't quite understand. Do you want to add a program or change channel?
- U: Change channel
- S: Okay. Change channel.
- S: Lets see. Channel five?
- U: Yes
- S: The channel was changed

All utterances are answers!

- In IBDM, all utterances are regarded as answers to some question
  - This can be regarded as an implementation of the Maxim of relation
  - The assumption that each utterance is relevant (to some goal) is reinterpreted as saying that each utterance addresses some question

- "Normal" answers require a matching question to be on ISSUES
- request and ask moves are also answers, addressing and resolving questions like "What can I do for you?", "How can I help you", "Do you want price or visa information?", "Do you want to call a contact or browse the phonebook"
- However, request and ask moves are special in that they do not *require* a matching question to be on ISSUES, and thus do not trigger accommodation
  - This encodes the fact that they are mainly forward-looking rather than backward-looking
  - This is the reason for having them as separate types of moves