Fluency project tutorial

Jose F Quesada & Jose Luis Pro

Current state

What do we have now?

- With lekta we can create dialogue systems based upon a certain domain.
- We create lexicons, grammar rules, mind structures and generation strategies in order to get that domain fully implemented.
- But all that things are created *ad-hoc* for that domain.
- If we want to implement another different dialogue system we must start again.
- Without being possible to reuse lexicons or grammar rules.
- It's necessary to have a good level in programming skills and some experience in the implementation of grammars.

Dialogue systems implemented in the exercises

Examples

- Session 04: Exercise 01.
  - In integer calculator exercise we have "english numbers" grammar.
  - But mind structure was an "Expression object" (not reusable).
- Session 04: Exercise 02.
  - In domotics assistant exercise we had grammars and lexicon for actions and devices.
  - And mind structures were basically boolean flags to represent devices context state.

Dialogue systems implemented in the exercises

Examples

- Session 04: Exercise 01.
  - In integer calculator exercise we have "english numbers" grammar.
  - But mind structure was an "Expression object" (not reusable).
- Session 04: Exercise 02.
  - In domotics assistant exercise we had grammars and lexicon for actions and devices.
  - And mind structures were basically boolean flags to represent devices context state.

What do we want?

- **General purpose DM:** We would like a dialogue manager reasoning engine domain-independent.
- **Reusability:** Grammar for some parameters should be reusable. For example, a grammar for english dates or numbers can be used in all imaginable domains with almost no difference between them.
- **Script model:** We must simplify the creation of tasks in a dialogue system by means of script templates, easier to implement and debug.
- **Interface friendly:** If we have a large parameter database it's possible for an inexperienced user (in either programming or linguistic skills) to implement a dialogue system that satisfies his needs.

What is "Fluency"?

Fluency features

- Is a lekta based framework for the easy creation of task-oriented dialogue systems.
- Intended to be domain-independent.
- With generic dialogue manager mind structures and strategies.
- Currently, in a very first production stage.
- Subject to design decisions changes if desired.
- Designed to be translated into any language in a simple and comfortable way.
- Implemented in order to have a GUI designing application that can automatically generate fluency compatible code.

Dialogue act annotation definition

Dialogue act annotation is the activity of marking up stretches of dialogue with information about the dialogue acts performed, [...] focused on marking up their communicative functions.*

- So we must classify and mark up all possible user proferences.
- In order to get its communicative function so we can have different dialogue strategies.
- Bunt taxonomy is so exhaustive and complex.
- We have used at first a simplified taxonomy but it can be expanded when needed.

* *Harry Bunt et al. Towards an ISO standard for dialogue act annotation (2010).*

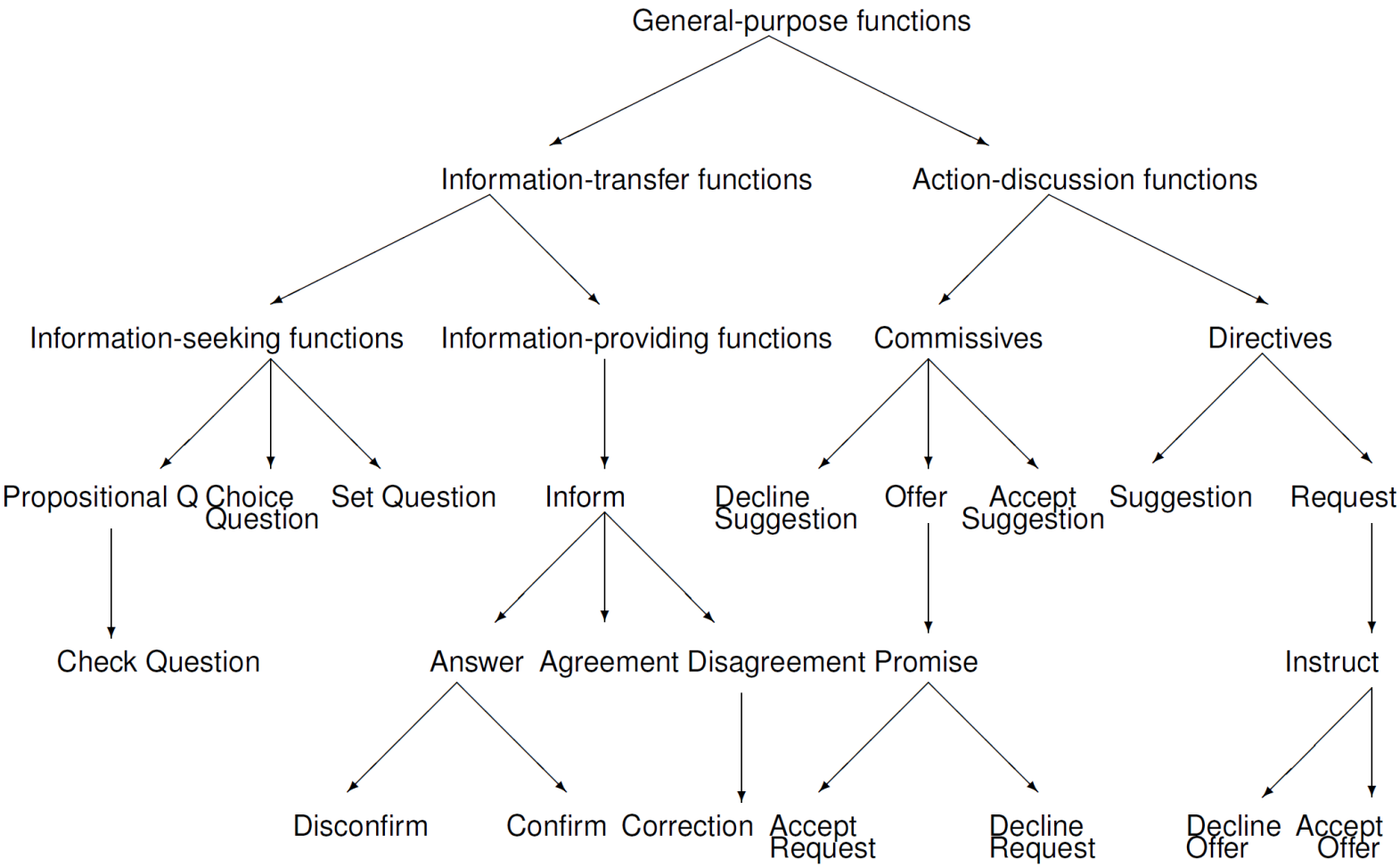
Dialogue act annotation definition

Dialogue act annotation is the activity of marking up stretches of dialogue with information about the dialogue acts performed, [...] focused on marking up their communicative functions.*

- So we must classify and mark up all possible user proferences.
- In order to get its communicative function so we can have different dialogue strategies.
- Bunt taxonomy is so exhaustive and complex.
- We have used at first a simplified taxonomy but it can be expanded when needed.

* *Harry Bunt et al. Towards an ISO standard for dialogue act annotation (2010).*

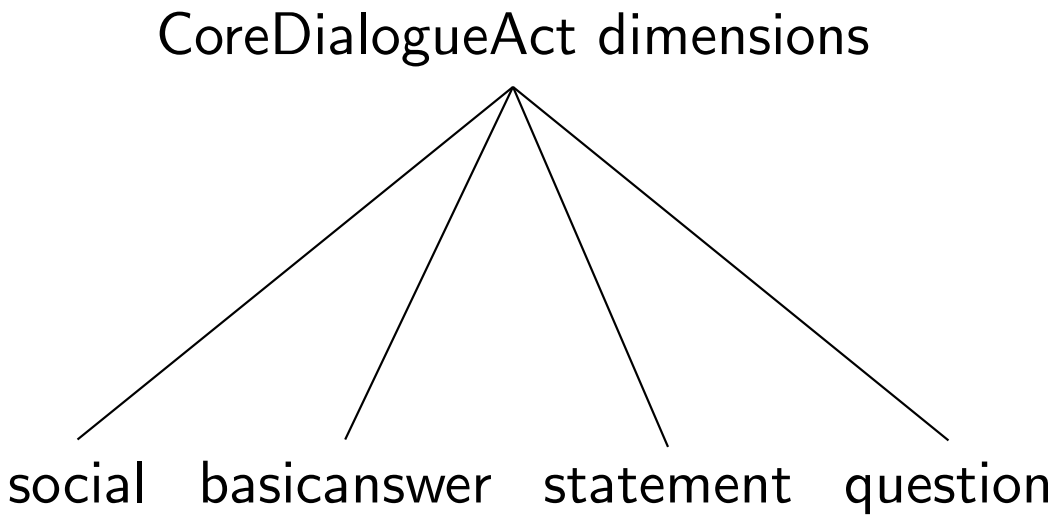
Bunt dialogue act taxonomy



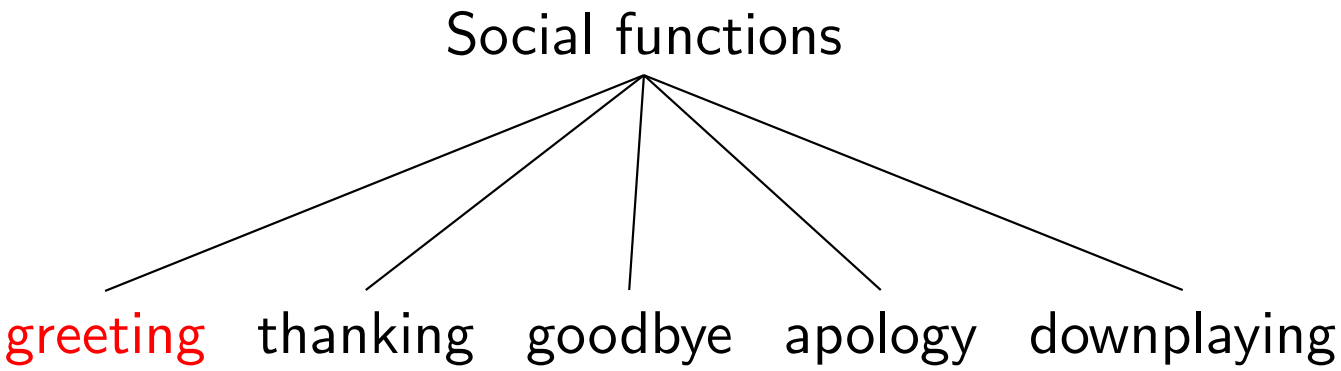
Harry Bunt et al. Towards an ISO standard for dialogue act annotation (2010).

Fluency dialogue act taxonomy

```
classDef:StructureComplex
(
    CoreDialogueAct :
    (
        Dimension,
        Function
    )
)
```



Social dimension



U: Good morning.

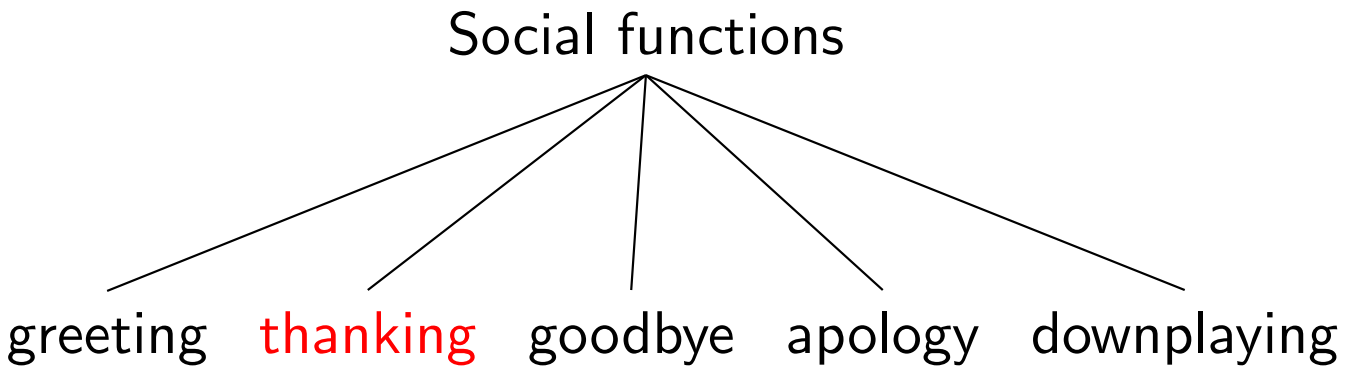
U: Nice to meet you.

U: Hello!

CoreDialogueAct:

$$\begin{bmatrix} \text{Dimension:} & \text{social} \\ \text{Function:} & \text{greeting} \end{bmatrix}$$

Social dimension



U: Thanks!

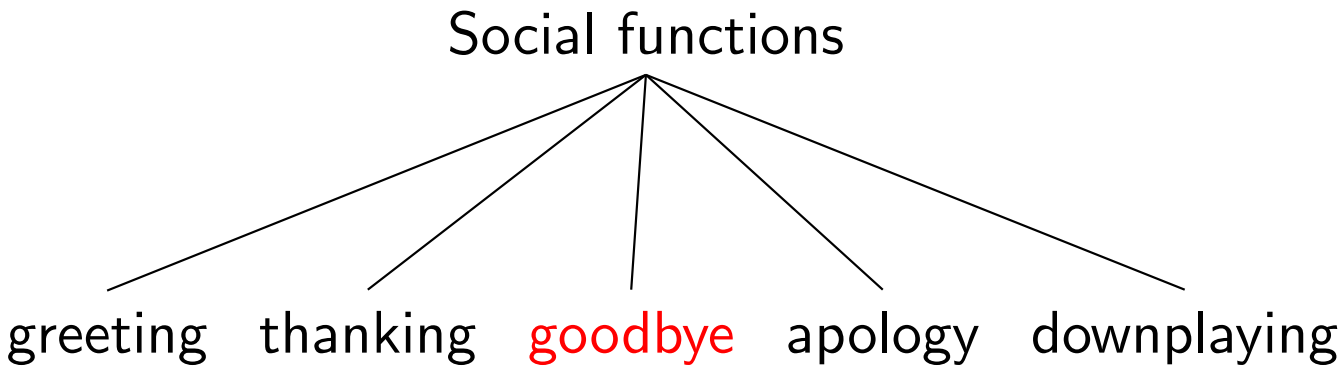
U: Thank you very much.

U: I'm so thankful!

CoreDialogueAct:

$$\begin{bmatrix} \text{Dimension:} & \text{social} \\ \text{Function:} & \text{thanking} \end{bmatrix}$$

Social dimension



U: Have a good day!

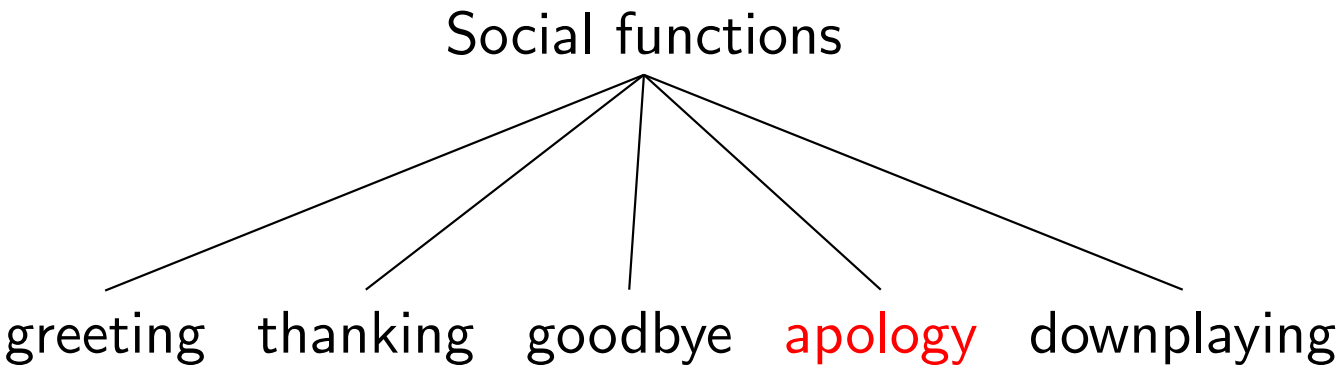
U: See you later.

U: Bye!

CoreDialogueAct:

$$\begin{bmatrix} \text{Dimension:} & \text{social} \\ \text{Function:} & \text{goodbye} \end{bmatrix}$$

Social dimension



U: I'm sorry!

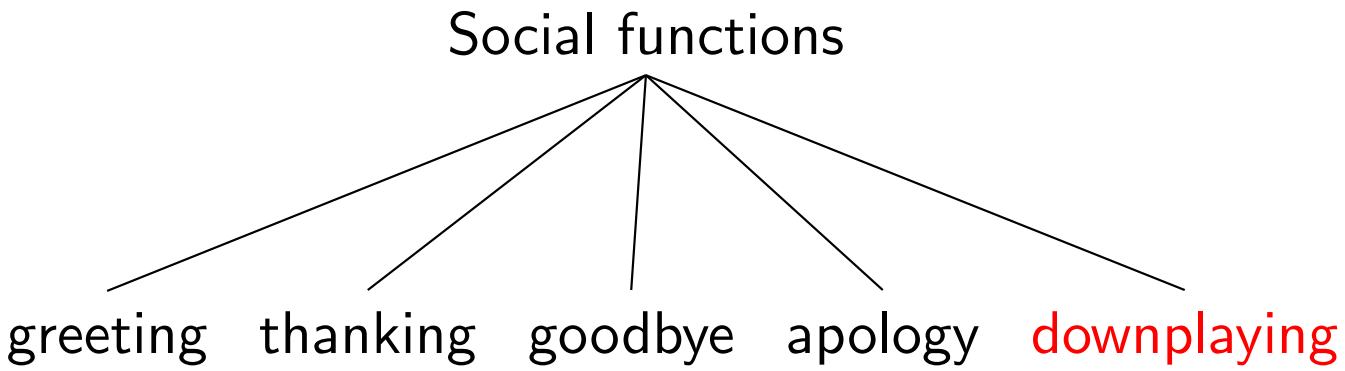
U: Excuse me.

U: My sincere apologies.

CoreDialogueAct:

$$\begin{bmatrix} \text{Dimension:} & \text{social} \\ \text{Function:} & \text{apology} \end{bmatrix}$$

Social dimension



U: You're welcome!

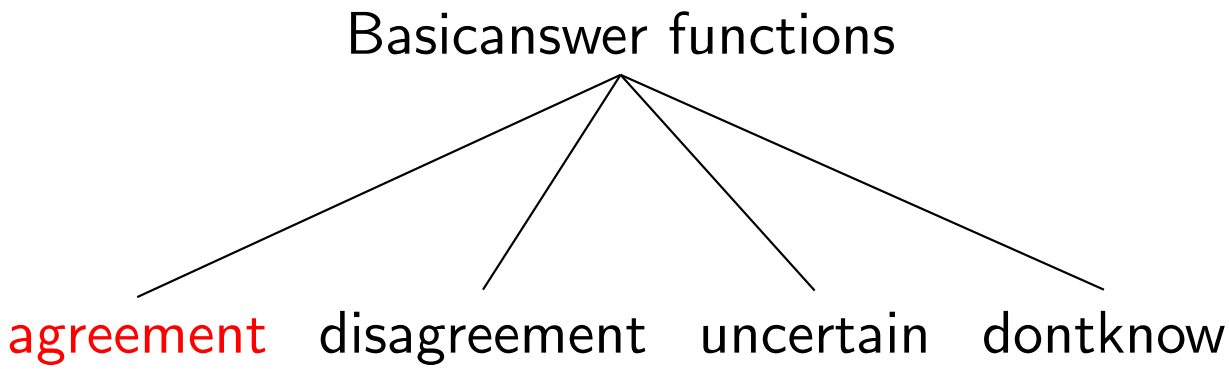
U: Not at all.

U: Don't mention it.

CoreDialogueAct:

$$\begin{bmatrix} \text{Dimension:} & \text{social} \\ \text{Function:} & \text{downplaying} \end{bmatrix}$$

Basicanswer dimension



U: Yes.

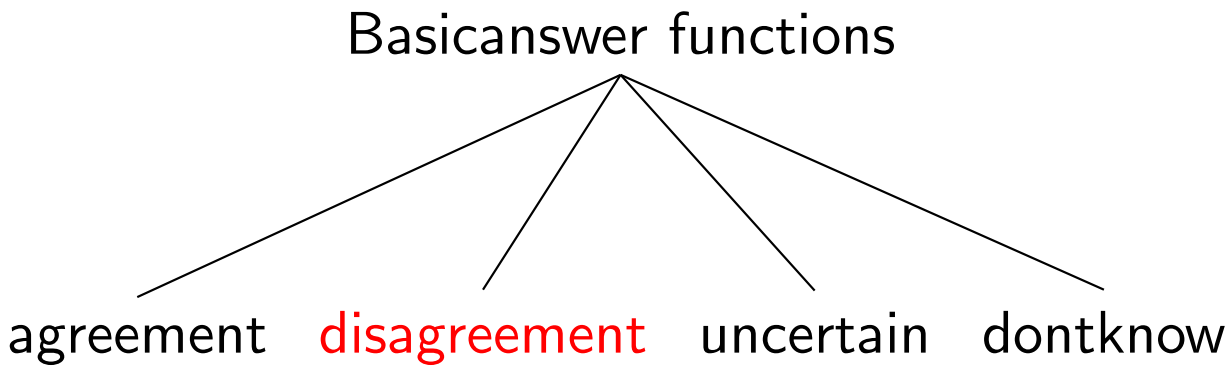
U: Ok.

U: That's right.

CoreDialogueAct:

$$\begin{bmatrix} \text{Dimension:} & \text{basicanswer} \\ \text{Function:} & \text{agreement} \end{bmatrix}$$

Basicanswer dimension



U: No.

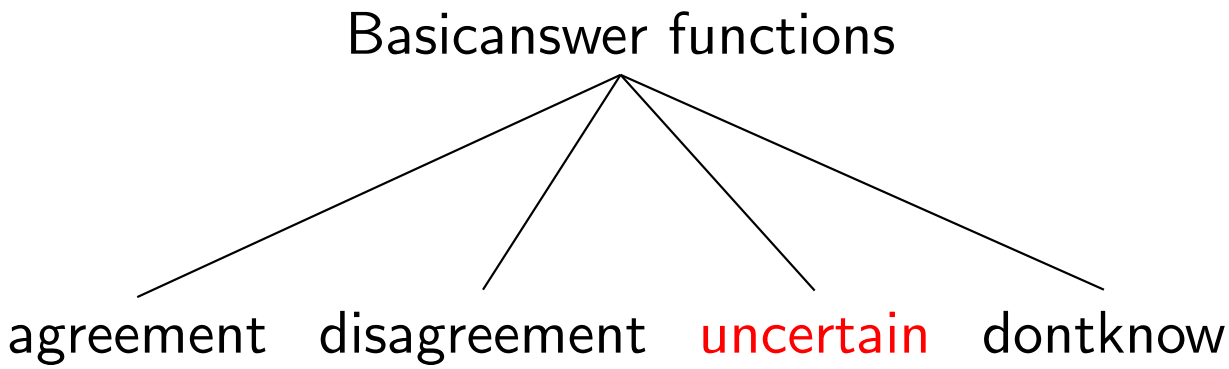
U: No way.

U: I can't agree.

CoreDialogueAct:

$$\begin{bmatrix} \text{Dimension:} & \text{basicanswer} \\ \text{Function:} & \text{disagreement} \end{bmatrix}$$

Basicanswer dimension



U: Maybe.

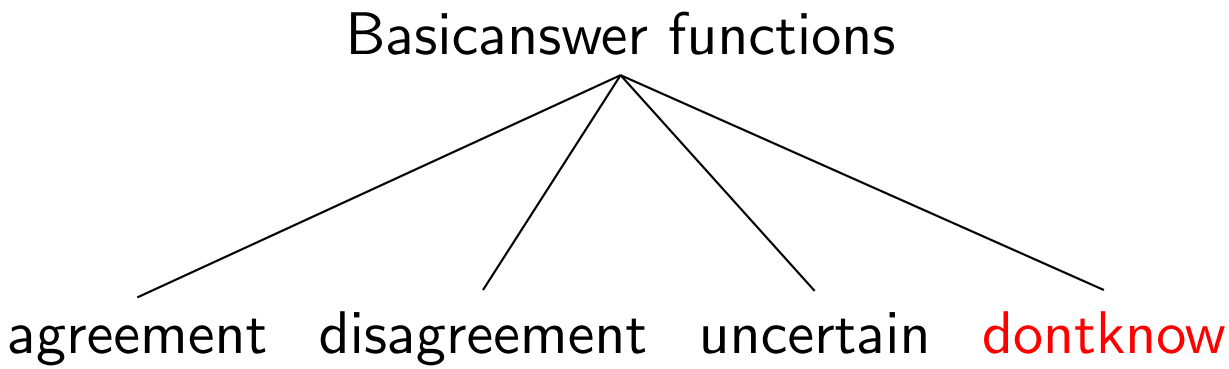
U: Perhaps.

U: Probably.

CoreDialogueAct:

$$\begin{bmatrix} \text{Dimension:} & \text{basicanswer} \\ \text{Function:} & \text{uncertain} \end{bmatrix}$$

Basicanswer dimension



U: I don't know.

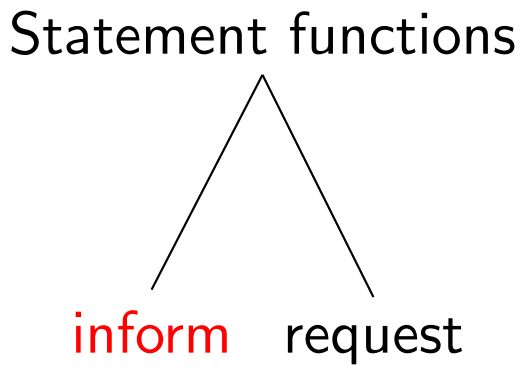
U: I don't remember.

U: I have no idea.

CoreDialogueAct:

$$\begin{bmatrix} \text{Dimension:} & \text{basicanswer} \\ \text{Function:} & \text{dontknow} \end{bmatrix}$$

Statement dimension



U: My name is ...

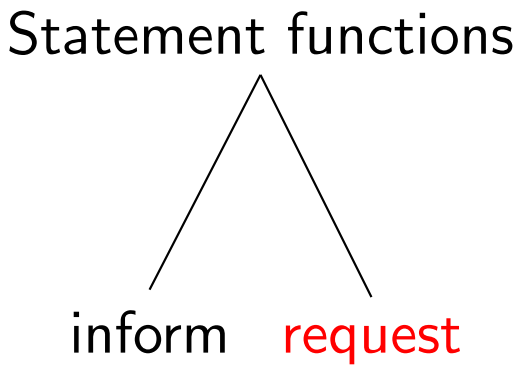
U: I live in ...

U: I don't want nothing else.

CoreDialogueAct:

$$\begin{bmatrix} \text{Dimension:} & \text{statement} \\ \text{Function:} & \text{inform} \end{bmatrix}$$

Statement dimension



U: I want ...

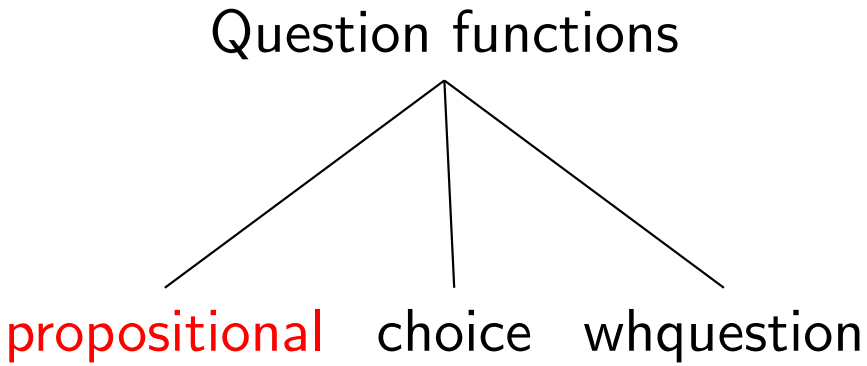
U: I would need ...

U: Would you mind if ...?

CoreDialogueAct:

$$\begin{bmatrix} \text{Dimension:} & \text{statement} \\ \text{Function:} & \text{request} \end{bmatrix}$$

Question dimension



S: Do you agree with the date of the appointment?

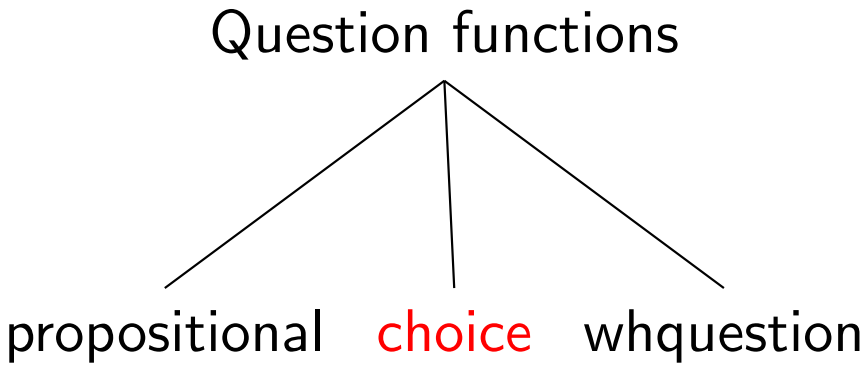
U: Do you like basketball?

U: Do you I am right?

CoreDialogueAct:

$$\begin{bmatrix} \text{Dimension:} & \text{question} \\ \text{Function:} & \text{propositional} \end{bmatrix}$$

Question dimension



S: What do you want to do first, make a bank transfer or locate the nearest ATM?

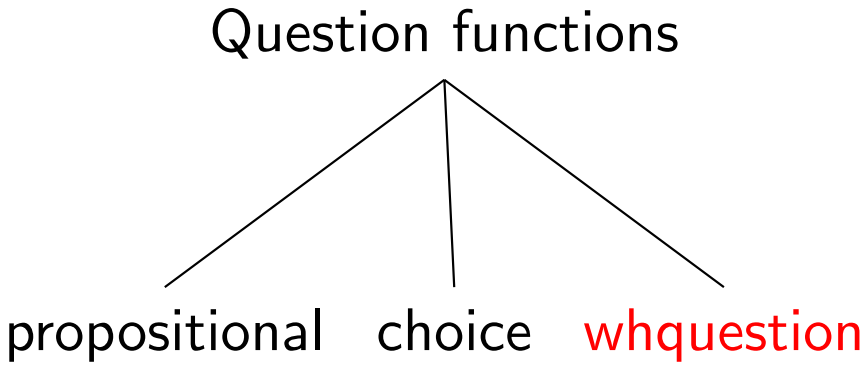
S: Do you prefer coffee or tea?

S: When do you want the appointment, in the morning or in the afternoon?

CoreDialogueAct:

$$\begin{bmatrix} \text{Dimension:} & \text{question} \\ \text{Function:} & \text{choice} \end{bmatrix}$$

Question dimension



U: What time is it?

U: How much is the doctor appointment?

U: Where is the nearest ATM?

CoreDialogueAct:

$$\begin{bmatrix} \text{Dimension:} & \text{question} \\ \text{Function:} & \text{whquestion} \end{bmatrix}$$

TaskDialogueAct structure

- Dialogue act annotation is used to know the communicative function of user proferences.
- This doesn't depend on domain.
- But it lacks of semantic information. For example: In `statement-request` pair we wish to know what user wants to do and the object of his desire.
- This kind of information may depend on domain.

```
1 classDef:StructureComplex (
2    TaskDialogueAct : ( Action, Scope ) )
3
4 classDef:StructureBatch (  Action : ( ActionDomain ) )
5
6 classDef:ElementLiteral (  ActionDomain )
7 classDef:ElementLiteral (  Scope )
```

TaskDialogueAct structure

- Dialogue act annotation is used to know the communicative function of user proferences.
- This doesn't depend on domain.
- But it lacks of semantic information. For example: In `statement-request` pair we wish to know what user wants to do and the object of his desire.
- This kind of information may depend on domain.

```
1 classDef:StructureComplex (
2    TaskDialogueAct : ( Action, Scope ) )
3
4 classDef:StructureBatch (  Action : ( ActionDomain ) )
5
6 classDef:ElementLiteral (  ActionDomain )
7 classDef:ElementLiteral (  Scope )
```

Domains implemented

So we have implemented a couple of domains to do some testings:

Medical appointment

- Task 1: `ActionDomain = 'book'. Scope = 'appointment'`

Banking management

- Task 1: `ActionDomain = 'consult'. Scope = 'bankaccount'`
- Task 2: `ActionDomain = 'locate'. Scope = 'atm'`
- Task 3: `ActionDomain = 'execute'. Scope = 'transfer'`

Domains implemented

So we have implemented a couple of domains to do some testings:

Medical appointment

- Task 1: `ActionDomain = 'book'. Scope = 'appointment'`

Banking management

- Task 1: `ActionDomain = 'consult'. Scope = 'bankaccount'`
- Task 2: `ActionDomain = 'locate'. Scope = 'atm'`
- Task 3: `ActionDomain = 'execute'. Scope = 'transfer'`

Verb lemmas

To detect actions in understanding stage, we associate some verbs lemmas to a certain action:

ActionDomain = 'book'

book, establish, have, **make**, get, schedule, ask, set up, ...

```
U: I want to get an appointment.
U: I would like to make medical appointment.
```

ActionDomain = 'execute'

make, move, execute, perform, do, accomplish, fulfill, effectuate, carry out, complete, ...

```
U: I want to perform a bank transfer.
U: I would like to make a transfer.
```

Please note that a verb lemma can be associated with more than one `ActionDomain` (ambiguities everywhere!).

Verb lemmas

To detect actions in understanding stage, we associate some verbs lemmas to a certain action:

ActionDomain = 'book'

book, establish, have, **make**, get, schedule, ask, set up, ...

```
U: I want to get an appointment.
U: I would like to make medical appointment.
```

ActionDomain = 'execute'

make, move, execute, perform, do, accomplish, fulfill, effectuate, carry out, complete, ...

```
U: I want to perform a bank transfer.
U: I would like to make a transfer.
```

Please note that a verb lemma can be associated with more than one `ActionDomain` (ambiguities everywhere!).

Verb lemmas

To detect actions in understanding stage, we associate some verbs lemmas to a certain action:

ActionDomain = 'book'

book, establish, have, **make**, get, schedule, ask, set up, ...

```
U: I want to get an appointment.
U: I would like to make medical appointment.
```

ActionDomain = 'execute'

make, move, execute, perform, do, accomplish, fulfill, effectuate, carry out, complete, ...

```
U: I want to perform a bank transfer.
U: I would like to make a transfer.
```

Please note that a verb lemma can be associated with more than one `ActionDomain` (ambiguities everywhere!).

Parameters

A ‘parameter’ is some kind of useful information to complete a task. For example a ‘datetime’ or an ‘accountnumber’.

```
classDef:StructureComplex
(
    Parameter :
    (
        ParameterCategory, // 'terminal', 'and', 'or', ...
        ParameterType,     // 'datetime', 'accountnumber', ...
        ParameterValue,    // Similar to math expressions
        ParameterOperand1,
        ParameterOperand2
    )
)
```

Example of ParameterCategory

U: I want an appointment for tomorrow or the day after tomorrow.

U: My telephone numbers are 1234 and 5678.

Parameters

A 'parameter' is some kind of useful information to complete a task. For example a 'datetime' or an 'accountnumber'.

```
classDef:StructureComplex
(
    Parameter :
    (
        ParameterCategory, // 'terminal', 'and', 'or', ...
        ParameterType,     // 'datetime', 'accountnumber', ...
        ParameterValue,    // Similar to math expressions
        ParameterOperand1,
        ParameterOperand2
    )
)
```

Example of ParameterCategory

U: I want an appointment for tomorrow or the day after tomorrow.

U: My telephone numbers are 1234 and 5678.

Parameters

This information is provided by the user and may be compulsory (red) or optative (green)

Examples

- `BookAppointment` task:
  - `medicalspeciality`
  - `countryplace`
  - `phonenumber`
  - `peselnumber` (Ok, it's a polish medical appointment!)
  - `datetime`
- `ConsultBankaccount` task:
  - `accountnumber`
- `LocateAtm` task:
  - `countryplace`
- `ExecuteTransfer` task:
  - `accountnumber`
  - `moneyamount`

Parameters classification

- Parameteres can be classified depending upon its domain.
- If it's domain-independent we say that the parameter belongs to "kernel" domain.
- But take into account that we can move a parameter from its domain to kernel domain in order to make it **reusable**.

Kernel domain implemented parameters

- `countryplace`
- `datetime`
- `letter`
- `moneyamount`
- `number`
- `ordinal`
- `phonenumber`
- `signchunk`

Parameters classification

- Parameteres can be classified depending upon its domain.
- If it's domain-independent we say that the parameter belongs to "kernel" domain.
- But take into account that we can move a parameter from its domain to kernel domain in order to make it **reusable**.

Kernel domain implemented parameters

- `countryplace`
- `datetime`
- `letter`
- `moneyamount`
- `number`
- `ordinal`
- `phonenumber`
- `signchunk`

Parameters classification

Medical appointment domain implemented parameters

- `medicalspeciality`
- `peselnumber`

Banking management domain implemented parameters

- `accountnumber`

Parameters example: datetime

```
classDef:StructureComplex (
        DateTime: (
        BaseDate,
        OffsetDate,
        MinDate,
        MaxDate,
        GeneralTime
    )
)

classDef:StructureComplex (
        GeneralTime: (
        BaseTime,
        OffsetTime,
        MinTime,
        MaxTime
    )
)
```

Parameters example: datetime

U: Starting next monday until 3pm to the day after 25 of august from 10am to a quarter to nine in the afternoon.

```
(DateTime:(MinDate:(GeneralTime:(MaxTime:(BaseTime:(Hour:15))),
                    OffsetDate :(DirectionOfTime:'forward',
Date             :(DayInWeek:1),
DayInWeekOffset:1)),
MaxDate:(GeneralTime:(MinTime:(BaseTime:(Hour:10)),
MaxTime:(BaseTime:(Hour  :20,
Minute:45))),
OffsetDate :(DirectionOfTime:'forward',
Date             :(Day:1)),
BaseDate   :(Day  :25,
Month:8))))))
```