Dialogue Management as Interactive Tree Building

Peter Ljunglöf

Philosophy, Linguistics and Theory of Science
University of Gothenburg

DiaHolmia, 24–26 June 2009
Abstract

We introduce a new dialogue manager for limited-domain dialogue systems:

- the dialogue domain is specified in type theory
- the user and system utterances are specified as a type-theoretical grammar

The dialogue manager tries to build a complete type-correct tree by successive refinement.

- similar to how Dynamic Syntax builds an analysis of a sentence
Abstract

We introduce a new dialogue manager for limited-domain dialogue systems:

- the dialogue domain is specified in type theory
- the user and system utterances are specified as a type-theoretical grammar

The dialogue manager tries to build a complete type-correct tree by successive refinement.

- similar to how Dynamic Syntax builds an analysis of a sentence
Yet another dialogue manager

So, why a new dialogue manager?

- well-defined underlying logic (type-theory)
- all-in-one specification: the whole domain, both syntax and semantics, is specified within the same framework
- can use type-checking to ensure consistency
Simple type theory

The types that are used in this talk are:

- **basic types**: $A$, $B$, $C$, ...
- **functions**: $T_1 \times \cdots \times T_n \rightarrow T$

The corresponding terms are:

- **constants**: $a_1, a_2, \ldots : A$, $b_1, \ldots : B$, ...
- **functions**: $f : T_1 \times \cdots \times T_n \rightarrow T$
  
  iff $f(t_1, \ldots, t_n) : T$ whenever $t_1 : T_1$, $\ldots$, $t_n : T_n$

Note: in this framework we only use atomic functions (i.e., no lambdas).
Simple type theory

The types that are used in this talk are:

- basic types: $A, B, C, \ldots$
- functions: $T_1 \times \cdots \times T_n \rightarrow T$

The corresponding terms are:

- constants: $a_1, a_2, \ldots : A, b_1, \ldots : B, \ldots$
- functions: $f : T_1 \times \cdots \times T_n \rightarrow T$
  
  iff $f(t_1, \ldots, t_n) : T$ whenever $t_1 : T_1, \ldots, t_n : T_n$

Note: in this framework we only use atomic functions (i.e., no lambdas).
Dialogue as proof editing

Type Theory is based on the Curry-Howard isomorphism:

- type $T \iff$ proposition $T^\circ$
- function $T_1 \times \cdots \times T_n \rightarrow T \iff$ implication $T_1^\circ \land \cdots \land T_n^\circ \rightarrow T^\circ$
- term $t : T \iff$ proof of $T^\circ$
- building a term $t : T \iff$ proving a proposition $T^\circ$

An interactive proof editor builds a term interactively:

- metavariable $?T \iff$ type (proposition) that has no term (proof)
  $\iff$ question (from the system):
  “what is the proof for $T^\circ$?”
- term containing metavariables $\iff$ incomplete proof tree
Dialogue as proof editing

Type Theory is based on the Curry-Howard isomorphism:

- type $T \iff$ proposition $T^\circ$
- function $T_1 \times \cdots \times T_n \to T \iff$ implication $T_1^\circ \land \cdots \land T_n^\circ \to T^\circ$
- term $t : T \iff$ proof of $T^\circ$
- building a term $t : T \iff$ proving a proposition $T^\circ$

An interactive proof editor builds a term interactively:

- metavariable $?T \iff$ type (proposition) that has no term (proof)
  \iff question (from the system):
  “what is the proof for $T^\circ$?”
- term containing metavariables \iff incomplete proof tree
Specifying a theory consists of giving:

- the basic types \((\text{Action}, \text{Price}, \text{Event}, \text{Date}, \text{City})\)
- the constants \((\text{sthlm} : \text{City}, \text{today} : \text{Date}, \text{€450} : \text{Price})\)
- the functions \((\text{book} : \text{Event} \rightarrow \text{Action}, \text{hotel} : \text{City} \times \text{Date} \rightarrow \text{Event})\)
Specifying the utterances

To each type, constant and function we have to specify utterances:

- system questions corresponding to basic types
  
  \( ?Action \mapsto \text{“What do you want to do?”}, \)
  
  \( ?Date \mapsto \text{“What date do you mean?”} \)

- utterances corresponding to constants:
  
  \( sthlm \mapsto \text{“Stockholm”, €450 \mapsto “fourhundred and fifty Euros”} \)

- complex utterances for functions:
  
  \( book(x) \mapsto \text{“book (an event | x), please”}, \)
  
  \( hotel(x, y) \mapsto \text{“a hotel ?(in x) ?(y)”} \)

This means that “book a hotel, please” is interpreted as

\( book(hotel(?City, ?Date)) \).
An example domain

A travel agency: specified as incomplete trees

book(?Event),
price(?Price),
when(?Date) : Action

event(?Event) : Price
oneway(?Route, ?Date),
return(?Route, ?Date, ?RDate),
hotel(?City, ?Date),
conference(?Conference) : Event

route(?Dest, ?Dept, ?Means) : Route
returnDate(?Date) : RDate
to(?City) : Dest
from(?City) : Dept

2009, 2010, ... : Year
jan, feb, ... : Month
1st, 2nd, ... : Day
lon, sthlm, ... : City
flight, boat, ... : Means
semdial, acl, ... : Conference
€450, €600, ... : Price
today, tomorrow,
date(?Month, ?Day),
conf-date(?Conference, ?Year) : Date
An example term

\texttt{book(oneway(route(to(sthlm), from(lon), boat), tomorrow))) : Action}
A fully typed variant

Terms and Trees

A fully typed variant


The corresponding tree

```
book:Action
  \|-- oneway:Event
      \|-- route:Route
          \|-- tomorrow:Date
              \|-- to:Dest
                  \|-- sthlm:City
              \|-- from:Dept
                  \|-- lon:City
              \|-- boat:Means
```

Dialogue management by successive refinement

The dialogue system builds a complete tree by successive refinement.

- similar (but not equivalent) to how Dynamic Syntax works

Uninstantiated nodes in the tree are represented with typed metavariables:

- a metavariable of type $T$ is written $?T$ as a wh-question
- or $?f_1 \lor \ldots \lor f_n : T$ for the corresponding alt-question

There is always one active node in the current tree:

- it is called the focus node (and is highlighted)

The tree is operated with commands:

- moving focus, inserting subtrees, refining metavariables, ...

The initial tree is the single focused node $?\text{Action}$. 
Dialogue management by successive refinement

The dialogue system builds a complete tree by successive refinement.

- similar (but not equivalent) to how Dynamic Syntax works

Uninstantiated nodes in the tree are represented with typed metavariables:

- a metavariable of type $T$ is written $?T$ as a wh-question
- or $?f_1 \lor \ldots \lor f_n : T$ for the corresponding alt-question

There is always one active node in the current tree:

- it is called the focus node (and is highlighted)

The tree is operated with commands:

- moving focus, inserting subtrees, refining metavariables, ...

The initial tree is the single focused node $?\text{Action}$. 
The dialogue system builds a complete tree by successive refinement.

- similar (but not equivalent) to how Dynamic Syntax works

Uninstantiated nodes in the tree are represented with typed metavariables:

- a metavariable of type $T$ is written $\ ? T \ ?$ as a wh-question
- or $\ ? f_1 \lor \ldots \lor f_n : T$ for the corresponding alt-question

There is always one active node in the current tree:

- it is called the focus node (and is highlighted)

The tree is operated with commands:

- moving focus, inserting subtrees, refining metavariables, ...

The initial tree is the single focused node $\ ? \text{Action}$. 
Dialogue management by successive refinement

The dialogue system builds a complete tree by successive refinement.

- similar (but not equivalent) to how Dynamic Syntax works

Uninstantiated nodes in the tree are represented with typed metavariables:

- a metavariable of type $T$ is written $\text{?} T$ as a wh-question
- or $\text{?} f_1 \lor \ldots \lor f_n : T$ for the corresponding alt-question

There is always one active node in the current tree:

- it is called the focus node (and is highlighted)

The tree is operated with commands:

- moving focus, inserting subtrees, refining metavariables, ...

The initial tree is the single focused node $\text{?} \text{Action}$. 
Dialogue management by successive refinement

The dialogue system builds a complete tree by successive refinement.
- similar (but not equivalent) to how Dynamic Syntax works

Uninstantiated nodes in the tree are represented with typed metavariables:
- a metavariable of type $T$ is written $\textit{?}\, T$ as a wh-question
- or $\textit{?}\, f_1 \lor \ldots \lor f_n : T$ for the corresponding alt-question

There is always one active node in the current tree:
- it is called the \textit{focus} node (and is highlighted)

The tree is operated with commands:
- moving focus, Inserting subtrees, refining metavariables, ...

The initial tree is the single focused node $\textit{?}\textit{Action}$. 
System-driven dialogue

?Action

S: ask(?Action) ⇒
“What do you want to do?”
System-driven dialogue

?price\book:Action

refine-down
“Do you want to ask for the price or book an event?”
System-driven dialogue

?-price^book:Action

U: “book an event” ⇒
answer(book(?Event):Action)
book:Action

?Event

integrate book(?Event):Action
System-driven dialogue

book:Action

?Event

select-next
S: ask(?Event)⇒
“What event are you interested in?”
book:Action

?oneway
return
hotel
conference:Event

refine-down
System-driven dialogue

book:Action

?oneway ∨ return ∨ hotel ∨ conference:Event

S: ask(oneway ∨ return ∨ hotel ∨ conference) ⇒
"Oneway trip, return trip, hotel or conference?"
System-driven dialogue

book: Action

?oneway ∨ return ∨ hotel ∨ conference: Event

U: “oneway trip” ⇒
answer(oneway(?Route, ?Date): Event)
System-driven dialogue

book:Action
   /
  oneway:Event
   /     
?Route  ?Date

integrate oneway(?Route,?Date):Event
System-driven dialogue

book:Action
    
    oneway:Event
    
    ?Route  ?Date

select-next
System-driven dialogue

\[
\text{book:Action} \\
\text{oneway:Event} \\
\text {?Route} \quad \text {?Date}
\]

S: ask(?Route) ⇒
“Which route do you want?”
System-driven dialogue

```
book:Action
  oneway:Event
    route:Route
      ?Date
        ?Dest
        ?Dept
        ?Means

refine-down
```
System-driven dialogue

book:Action

oneway:Event

route:Route

?Date

?Dest

?Dept

?Means

select-next
System-driven dialogue

```
book:Action
   
oneway:Event
   |       
route:Route ?Date
       |       |
?Dest ?Dept ?Means
```

S: ask(?Dest) ⇒
“To which city are you heading?”
System-driven dialogue

```
book: Action
 progress
  oneway: Event
    route: Route
      dept: ?Dept
      dest: ?Dest
      means: ?Means
      date: ?Date

U: “to Stockholm” ⇒
answer(to(sthlm: City): Dest)
```
System-driven dialogue

book:Action

oneway:Event

route:Route

?Date

to:Dest

?Dept

?Means

sthlm:City

---

integrate to(sthlm:City):Dest
System-driven dialogue

book:Action
  oneway:Event
    route:Route
      ?Date
        to:Dest
          ?Dept
            ?Means
              sth lm:City

select-next
S: ask(?Dept) ⇒
“From where are you leaving?”
System-driven dialogue

```
book:Action
   /oneway:Event
     /route:Route
       /?Date
         /to:Dest
           /?Dept
             /?Means
               /sthIm:City
```

U: “from London” ⇒
answer(from(lon:City):Dept)
System-driven dialogue

book:Action
  oneway:Event
    route:Route
      ?Date
      to:Dest
      from:Dept
        sthlm:City
        lon:City
      ?Means

integrate from(lon:City):Dept
System-driven dialogue

(etcetera...)

Diagram:

- **book**: Action
  - **oneway**: Event
    - **route**: Route
      - **to**: Dest
        - **sthlm**: City
      - **from**: Dept
        - **lon**: City
      - **?Date**
      - **?Means**
System-driven dialogue

(etcetera... )
System-driven dialogue

```
book:Action
  oneway:Event
    route:Route
      to:Dest
        sthlm:City
      from:Dept
        lon:City
      boat:Means
        ?Date
```

(etcetera...)
System-driven dialogue

```
book:Action
  oneway:Event
    route:Route
      tomorrow:Date
        to:Dest
          sthlm:City
        from:Dept
          Lon:City
      boat:Means
```

(etcetera...)
System-driven dialogue

Voilà!
I have borrowed the idea of underspecified (or unfixed) nodes from Dynamic Syntax (or rather the Logic of Finite Trees):

- The type A (D) must dominate B, C (E); i.e.: $A \Rightarrow^* \alpha B \beta$, ...
- All dominating nodes (A, D) must be uninstantiated
Underspecified information

We use underspecified tree nodes for incorporating underspecified information; when the user says something which the system cannot integrate into the current tree.

- This is similar to how Dynamic Syntax does it:
  - if the syntactic function of a phrase is unknown, its node/tree becomes underspecified
  - e.g., a noun in initial position can be subject or object

- Corresponds to issue/action clarification in GoDiS
  - within plans or between plans

There are (at least) three different refinement strategies.

- Correspond to known dialogue strategies?
Strategy 1: Top-down refinement

?Action

S: ask(?Action) ⇒
“What do you want to do?”
U: “go to Stockholm” ⇒
answer(route(to(sthlm), ?City, ?Means):Route)
Strategy 1: Top-down refinement

S: \text{ask(\text{priceIssue(?):Action} \lor \text{book(?):Action})} \Rightarrow
“Do you want to ask for the price or book an event?”
Strategy 1: Top-down refinement

book: Action
  |?
  ?Event

route: Route
  to: Dest
  ?Dept
  ?Means

sthlm: City

U: “book an event” ⇒
answer(book(?Event):Action)
Strategy 1: Top-down refinement

S: ask(oneway(?,?):Event ∨ return(?,?,?):Event)⇒
“Do you want a oneway trip or a return trip?”
Strategy 1: Top-down refinement

U: “oneyway” ⇒
answer(oneyway(ROUTE,?Date):Event)
Strategy 1: Top-down refinement

```
book:Action
    |     |
    v     v
oneway:Event
    |     |
    v     v
route:Route
    |     |     |
    v     v     v
  to:Dest  ?Dept  ?Means
    |      |
    v      v
sthIn:City
```

refine-down
Strategy 1: Top-down refinement

```
book:Action
  
oneway:Event
     
route:Route   ?Date

  
to:Dest   ?Dept   ?Means

  
sthIm:City
```

select-next
Strategy 1: Top-down refinement

book:Action
  oneway:Event
    route:Route
      ?Date
        to:Dest
        ?Dept
        ?Means
      sthIm:City

S: ask(?Dept) ⇒
“From where are you leaving?”
Strategy 1: Top-down refinement

U: “London” ⇒
answer(lon:City)
Strategy 1: Top-down refinement

```
book:Action
  |           |
  oneway:Event
    |
    route:Route
      |   ?Date
      to:Dest
        |
        sthlm:City
            ?City
                lon:City
```

refine-down
Strategy 1: Top-down refinement

```
book:Action
  oneway:Event
    route:Route
      to:Dest
      from:Dept
      ?Date
        sthlm:City
        lon:City
      ?Means
```

refine-down
Strategy 1: Top-down refinement

(etcetera...)

```
book:Action
   |
   oneway:Event
 |
route:Route  ?Date
 |
to:Dest
 |
sthlm:City

from:Dept
 |
lon:City
 |
?Means
```

---

Peter Ljunglöf (Univ. Gothenburg)  Tree-Based Dialogue Management  DiaHolmia 2009  14 / 26
Strategy 1: Top-down refinement

```
book:Action
   |
oneway:Event
   |
route:Route
   |  ?Date
   |
   |  to:Dest
   |    |  sthlm:City
   |    |
   |  from:Dept
   |    |  lon:City
   |
```

(etcetera...)

Strategy 1: Top-down refinement

book:Action
  |
  oneway:Event
    |
    route:Route       tomorrow:Date
      |
      to:Dest       from:Dept
        |
        sthlm:City       Ion:City
          |
          boat:Means

(etcetera...)

Peter Ljunglöf (Univ. Gothenburg)
Strategy 2: Bottom-up refinement

?Action

S: ask(?Action) ⇒ “What do you want to do?”
Strategy 2: Bottom-up refinement

U: “go to Stockholm” ⇒ answer(route(to(sthlm), ?Dept, ?Means):Route)
Strategy 2: Bottom-up refinement

?Action

route: Route

to: Dest

sthlm: City

?Dept

?Means

focus-down
Strategy 2: Bottom-up refinement

?Action

?oneway ∨ return: Event

route: Route

to: Dest

?Dept

?Means

sthlm: City

refine-up
Strategy 2: Bottom-up refinement

?Action

?oneway ∨ return: Event

route: Route

to: Dest

?Dept

?Means

sthlm: City

S: ask(oneway(?,?): Event ∨ return(?,?,?): Event) ⇒
“Do you want a oneway trip or a return trip?”
Strategy 2: Bottom-up refinement

U: “oneway” ⇒
answer(oneway(?Route,?Date):Event)
Strategy 2: Bottom-up refinement

?Action
  ?price -> book:Action
  oneway:Event
  route:Route ?Date
  to:Dest ?Dept ?Means
  sthIm:City

refine-up
Strategy 2: Bottom-up refinement

\[
\text{?Action} \\
\uparrow \\
\text{?price} \lor \text{book:Action} \\
\uparrow \\
\text{oneway:Event} \\
\uparrow \\
\text{route:Route} \quad \text{?Date} \\
\uparrow \\
\text{to:Dest} \quad \text{?Dept} \quad \text{?Means} \\
\uparrow \\
\text{sthIm:City}
\]

S: ask(pricelssue(?):Action \lor book(?):Action) \Rightarrow 
“Do you want to ask for the price or book an event?”
U: “book an event” ⇒
answer(book(?Event):Action)
Strategy 2: Bottom-up refinement

book: Action

oneway: Event

route: Route

?Date

to: Dest

?Dept

?Means

sthln: City

refine-up
Strategy 2: Bottom-up refinement

```
book:Action
  
  one-way:Event
    
    route:Route
      
      to:Dest
        ?Dept
      
      ?Means
    
    sthIm:City

select-next
```
Strategy 2: Bottom-up refinement

S: ask(?Dept) ⇒
"From where are you leaving?"
Strategy 2: Bottom-up refinement

```
book:Action
  |     |
  oneway:Event
  |           |
route:Route  ?Date
  |         |
to:Dest ?Dept ?Means
  sthlm:City

lon:City
```

U: “London” ⇒
answer(lon:City)
Strategy 2: Bottom-up refinement

```
book:Action
  oneway:Event
    route:Route
      to:Dest
      ?Date
      ?Dept
      ?Means
    sthlm:City

lon:City
```

focus-down
Strategy 2: Bottom-up refinement

book:Action
  └── oneway:Event
      ├── route:Route
      │    └── ?Date
      │        ├── to:Dest
      │        │    └── sthlm:City
      │        └── ?Dept
      │              └── ?Means
      │                  └── from:Dept
      │                          └── Ion:City

refine-up
Strategy 2: Bottom-up refinement

```
book: Action
  
oneway: Event
    
route: Route
  
from: Dept
  
?Date
  
?Means
  
to: Dest
  
sthlm: City
  
lon: City
```

refine-up
Strategy 2: Bottom-up refinement

Book: Action

Oneway: Event

Route: Route

?Date

to: Dest

from: Dept

sthlm: City

lon: City

?Means

(etcetera... )
Strategy 2: Bottom-up refinement

```
book:Action
  oneway:Event
    route:Route
      ?Date
        to:Dest
        from:Dept
        boat:Means
          sthlm:City
          lon:City

(etcetera...)```
Strategy 2: Bottom-up refinement

(book:Action)

(owenay:Event)

(route:Route)

(tomorrow:Date)

(to:Dest)

(from:Dept)

(boat:Means)

(sthlm:City)

(Ion:City)

(etcetera... )
Strategy 3: “Bottom-down” refinement

?Action
Strategy 3: “Bottom-down” refinement

?Action

route: Route

to: Dest  ?Dept  ?Means

sthlm: City
Strategy 3: “Bottom-down” refinement

\[
\text{?Action} \\
\text{route:Route} \\
\text{to:Dest} \quad \text{?Dept} \quad \text{?Means} \\
\text{sthlm:City}
\]

focus-down
Strategy 3: “Bottom-down” refinement

```
?Action

route: Route

<table>
<thead>
<tr>
<th>to: Dest</th>
<th>?Dept</th>
<th>?Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>sthlm: City</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

select-next
Strategy 3: “Bottom-down” refinement

```
?Action

route: Route

<table>
<thead>
<tr>
<th>to: Dest</th>
</tr>
</thead>
<tbody>
<tr>
<td>sthlm: City</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?Dept</th>
</tr>
</thead>
<tbody>
<tr>
<td>lon: City</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?Means</th>
</tr>
</thead>
</table>
```
Strategy 3: “Bottom-down” refinement

```
?Action
  
route: Route

  to: Dest
  sthlm: City

  ?Dept

  ?Means

lon: City

focus-down
```
Strategy 3: “Bottom-down” refinement

```
?Action
  
route: Route
  
to: Dest
     
sthlm: City

from: Dept

lon: City
```

refine-up
Strategy 3: “Bottom-down” refinement
Strategy 3: “Bottom-down” refinement

```
?Action
  
route:Route
  
  to:Dest
  sthlm:City
  lon:City

  from:Dept

?Means
```
Strategy 3: “Bottom-down” refinement

```
?Action

route:Route
  to:Dest
    sthlm:City
  from:Dept
    lon:City
  boat:Means
```
Strategy 3: “Bottom-down” refinement

![](image-url)

```
?Action

?oneway\ or \ return: Event

route: Route

to: Dest  from: Dept  boat: Means

sthlm: City  lon: City
```

refine-up
Strategy 3: “Bottom-down” refinement

?Action

oneway:Event

route:Route

?Date

to:Dest

sthlm:City

from:Dept

lon:City

boat:Means
Strategy 3: “Bottom-down” refinement

?Action

oneway:Event

route:Route

?tDate

to:Dest

from:Dept

boat:Means

sthlm:City

lon:City
Strategy 3: “Bottom-down” refinement

?Action

oneway:Event

route:Route

<table>
<thead>
<tr>
<th>to:Dest</th>
<th>from:Dept</th>
<th>boat:Means</th>
</tr>
</thead>
<tbody>
<tr>
<td>sthIm:City</td>
<td>lon:City</td>
<td></td>
</tr>
</tbody>
</table>

tomorrow:Date
Strategy 3: “Bottom-down” refinement

?-Action

?price∧book:Action

oneway:Event

route:Route	tomorrow:Date

to:Dest

from:Dept

boat:Means

sthlm:City

lon:City

refine-up
Strategy 3: “Bottom-down” refinement

?Action
  ↓
book:Action
  ↓
oneway:Event
    ↓
route:Route	 tomorrow:Date
      ↓
to:Dest	 from:Dept
        ↓
sthlm:City	 lon:City
        ↓
boat:Means
Strategy 3: “Bottom-down” refinement

book:Action

oneway:Event

route:Route

tomorrow:Date

to:Dest

from:Dept

boat:Means

sthlm:City

lon:City

refine-up
Three different refinement strategies

So, there are (at least) the following refinement strategies:

- top-down refinement
- bottom-up refinement
- “bottom-down” refinement

Of course, these strategies can be combined, e.g.:

- strategy depends on the type of the dominating node
- strategy depends on the maximum/minimum distance between the dominating and dominated nodes
Three different refinement strategies

So, there are (at least) the following refinement strategies:

- top-down refinement
- bottom-up refinement
- “bottom-down” refinement

Of course, these strategies can be combined, e.g.:

- strategy depends on the type of the dominating node
- strategy depends on the maximum/minimum distance between the dominating and dominated nodes
Answering user questions

We use function definitions for finding answers to user questions.

```
when(?Date) : Action
conf-date(?Conference, ?Year) : Date
eacl, semdial, ... : Conference
def conf-date(semdial,2009) = date(jun,24)
def conf-date(semdial,2010) = ...
def conf-date(acl,2009) = ...
```

U: “when is SemDial?”
S: “Which year do you mean?”
U: “this year”

⇒ when(conf-date(semdial,2009)) ⇒ when(date(jun,24)) ⇒
S: “SemDial 2009 starts 24th June.”
Answering user questions

We use function definitions for finding answers to user questions.

```quote
when(\?Date) : Action
conf-date(\?Conference, \?Year) : Date
eacl, semdial, ... : Conference
def conf-date(semdial,2009) = date(jun,24)
def conf-date(semdial,2010) = ... 
def conf-date(acl,2009) = ...
```

U: “when is SemDial?”
S: “Which year do you mean?”
U: “this year”

```quote
⇒ when(conf-date(semdial,2009)) ⇒ when(date(jun,24)) ⇒
S: “SemDial 2009 starts 24th June.”
```
Answering user questions

We use function definitions for finding answers to user questions.

```
when(?

conf-date(?

eacl, semdial, ... : Conference
def conf-date(semdial,2009) = date(jun,24)
def conf-date(semdial,2010) = ...
def conf-date(acl,2009) = ...
```

U: “when is SemDial?”
S: “Which year do you mean?”
U: “this year”

⇒ when(conf-date(semdial,2009)) ⇒ when(date(jun,24)) ⇒
S: “SemDial 2009 starts 24th June.”
Answering user questions

We use function definitions for finding answers to user questions.

```plaintext
when(?Date) : Action
conf-date(?Conference, ?Year) : Date
eacl, semdial, ... : Conference
def conf-date(semdial,2009) = date(jun,24)
def conf-date(semdial,2010) = ...
def conf-date(acl,2009) = ...
```

U: “when is SemDial?”
S: “Which year do you mean?”
U: “this year”

⇒ when(conf-date(semdial,2009)) ⇒ when(date(jun,24)) ⇒
S: “SemDial 2009 starts 24th June.”
Answering user questions

We use function definitions for finding answers to user questions.

\[
\text{when(\textbf{?Date}) : Action} \\
\text{conf-date(\textbf{?Conference}, \textbf{?Year}) : Date} \\
\text{eacl, semdial, \ldots : Conference} \\
\textbf{def} \text{ conf-date(semdial,2009) = date(jun,24)} \\
\textbf{def} \text{ conf-date(semdial,2010) = \ldots} \\
\textbf{def} \text{ conf-date(acl,2009) = \ldots}
\]

U: “when is SemDial?”
S: “Which year do you mean?”
U: “this year”
   \[\Rightarrow \text{when(conf-date(semdial,2009)) \Rightarrow when(date(jun,24)) \Rightarrow}\]
S: “SemDial 2009 starts 24th June.”
Answering user questions

We use function definitions for finding answers to user questions.

```
when(?Date) : Action
conf-date(?Conference, ?Year) : Date
eacl, semdial, ... : Conference
def conf-date(semdial,2009) = date(jun,24)
def conf-date(semdial,2010) = ...
def conf-date(acl,2009) = ...
```

U: “when is SemDial?”
S: “Which year do you mean?”
U: “this year”

⇒ when(conf-date(semdial,2009)) ⇒ when(date(jun,24)) ⇒
S: “SemDial 2009 starts 24th June.”
An underspecified node is (will be) a subtree of its parent node
- in DS, underspecified nodes are used when their function is unknown
- here, they are used for underspecified/ambiguous user answers

A linked tree is *not* dominated by its “parent”: it is a different tree
- in DS, linked trees are used for relative clauses, PPs, definite, anaphoric expressions and such things
  - links are *hard*: they determine the semantics of the “parent”
- here, links are used for answers, sub-dialogues, anaphoric expressions
  - links are *soft*: the user is free to skip the result of the sub-dialogue
Linked trees

An underspecified node is (will be) a subtree of its parent node

- in DS, underspecified nodes are used when their function is unknown
- here, they are used for underspecified/ambiguous user answers

A linked tree is not dominated by its “parent”: it is a different tree

- in DS, linked trees are used for relative clauses, PPs, definites, anaphoric expressions and such things
  - links are hard: they determine the semantics of the “parent”
- here, links are used for answers, sub-dialogues, anaphoric expressions
  - links are soft: the user is free to skip the result of the sub-dialogue
The information state of our proposed dialogue model is:

- a collection of linked trees,
- where one tree is active, and has an active focus node.
The information state of our proposed dialogue model is:

- a collection of linked trees,
- where one tree is active, and has an active focus node
Example sub-dialogue

U: “I’d like to book a boat to Stockholm”
S: “When do you want to leave?”
   U: “when is SemDial?”
   S: “Which year do you mean?”
   U: “this year”
   S: “SemDial starts 24th June.”
U: “ok, I’ll leave the day before”
S: “I have booked a boat to Stockholm on 23rd June.”
Sub-dialogues

Example sub-dialogue

U: “I’d like to book a boat to Stockholm”
S: “When do you want to leave?”
  U: “when is SemDial?”
  S: “Which year do you mean?”
  U: “this year”
  S: “SemDial starts 24th June.”
U: “ok, I’ll leave the day before”
S: “I have booked a boat to Stockholm on 23rd June.”
Example sub-dialogue

U: “I’d like to book a boat to Stockholm”
S: “When do you want to leave?”
   U: “when is SemDial?”
      S: “Which year do you mean?”
      U: “this year”
         S: “SemDial starts 24th June.”
U: “ok, I’ll leave the day before”
S: “I have booked a boat to Stockholm on 23rd June.”
Sub-dialogues

Example sub-dialogue

U: “I’d like to book a boat to Stockholm”
S: “When do you want to leave?”
   U: “when is SemDial?”
   S: “Which year do you mean?”
   U: “this year”
   S: “SemDial starts 24th June.”
U: “ok, I’ll leave the day before”
S: “I have booked a boat to Stockholm on 23rd June.”
Example sub-dialogue

U: “I’d like to book a boat to Stockholm”
S: “When do you want to leave?”
   U: “when is SemDial?”
   S: “Which year do you mean?”
   U: “this year”
      S: “SemDial starts 24th June.”
U: “ok, I’ll leave the day before”
S: “I have booked a boat to Stockholm on 23rd June.”
Sub-dialogues

Example sub-dialogue

U: “I’d like to book a boat to Stockholm”
S: “When do you want to leave?”
   U: “when is SemDial?”
   S: “Which year do you mean?”
   U: “this year”
   S: “SemDial starts 24th June.”
U: “ok, I’ll leave the day before”
S: “I have booked a boat to Stockholm on 23rd June.”
Example sub-dialogue

U: “I’d like to book a boat to Stockholm”
S: “When do you want to leave?”
  U: “when is SemDial?”
  S: “Which year do you mean?”
  U: “this year”
  S: “SemDial starts 24th June.”
U: “ok, I’ll leave the day before”
S: “I have booked a boat to Stockholm on 23rd June.”
Sub-dialogues

**Example sub-dialogue**

**U:** “I’d like to book a boat to Stockholm”  
**S:** “When do you want to leave?”  
  **U:** “when is SemDial?”  
  **S:** “Which year do you mean?”  
  **U:** “this year”  
  **S:** “SemDial starts 24th June.”  
**U:** “ok, I’ll leave the day before”  
**S:** “I have booked a boat to Stockholm on 23rd June.”
Sub-dialogues

S: “What can I do for you?”
book:Action
  └── oneway:Event
      └── route:Route
          └── ?Date

U: “I want to book a boat to Stockholm”
S: “When do you want to leave?”
U: “when is SemDial?”
Sub-dialogues

book:Action

oneway:Event

route:Route

?Date

sub-dialog

when:Action

conf-date:Date

semdial:Conference

?Year

U: “when is SemDial?”
S: “Which year do you mean?”
Sub-dialogues

book:Action

onenway:Event

route:Route
?

?Date

semdialog:Conference

2009:Year

U: “this year”
Sub-dialogues

S: “SemDial starts 24th June.”
S: “SemDial starts 24th June.”
U: “ok, I’ll leave the day before”
Anaphoric expressions

?Action

S: “What can I do for you?”
U: “how much is a flight to Stockholm tomorrow?”
S: “It costs €450.”
U: “ok, book it”
S: “I have booked a flight to Stockholm tomorrow.”
Anaphoric expressions

\[
\begin{align*}
\text{price: Action} \\
\text{event: Price} \\
\text{oneway: Event} \\
\text{route: Route} & \quad \text{tomorrow: Date}
\end{align*}
\]

S: “What can I do for you?”
U: “how much is a flight to Stockholm tomorrow?”
S: “It costs €450.”
U: “ok, book it”
S: “I have booked a flight to Stockholm tomorrow.”
Anaphoric expressions

\[
\begin{align*}
\text{price}\!: \text{Action} & \quad \quad \quad \text{price}\!: \text{Action} \\
\text{event}\!: \text{Price} & \quad \quad \quad \quad \text{answer} \quad \quad \quad \quad \quad \text{€450}\!: \text{Price} \\
\text{oneway}\!: \text{Event} & \quad \quad \quad \quad \quad \quad \text{oneway}\!: \text{Event} \\
\text{route}\!: \text{Route} & \quad \quad \quad \quad \quad \quad \quad \text{route}\!: \text{Route} \\
& \quad \quad \quad \quad \quad \quad \quad \quad \quad \quad \text{tomorrow}\!: \text{Date} \\
\end{align*}
\]

S: “What can I do for you?”
U: “how much is a flight to Stockholm tomorrow?”
S: “It costs €450.”
U: “ok, book it”
S: “I have booked a flight to Stockholm tomorrow.”
Anaphoric expressions

price:Action
  | event:Price
  | oneway:Event
route:Route

price:Action

€450:Price

book:Action

?Event

answer

anaphoric

S: “What can I do for you?”
U: “how much is a flight to Stockholm tomorrow?”
S: “It costs €450.”
U: “ok, book it”
S: “I have booked a flight to Stockholm tomorrow.”
Anaphoric expressions

S: “What can I do for you?”
U: “how much is a flight to Stockholm tomorrow?”
S: “It costs €450.”
U: “ok, book it”
S: “I have booked a flight to Stockholm tomorrow.”
Several alternatives

?Action

S: “What can I do for you?”
U: “how much is a trip to Stockholm tomorrow?”
S: “By plane or by boat?”
U: “both”
S: “Plane costs €450 and boat €300.”
U: “ok, book the cheapest one”
Several alternatives

price: Action
    |
  event: Price
    |
  oneway: Event
  /
route: Route    tomorrow: Date
  /
  . . . ?Means

S: “What can I do for you?”
U: “how much is a trip to Stockholm tomorrow?”
S: “By plane or by boat?”
U: “both”
S: “Plane costs €450 and boat €300.”
U: “ok, book the cheapest one”
Several alternatives

\[
\text{price: Action} \\
\quad | \quad \text{event: Price} \\
\quad | \quad \text{oneway: Event} \\
\text{route: Route} \\
\quad | \quad \text{tomorrow: Date} \\
\quad | \\
\quad ?\text{plane} \lor \text{boat: Means}
\]

S: “What can I do for you?”
U: “how much is a trip to Stockholm tomorrow?”
S: “By plane or by boat?”
U: “both”
S: “Plane costs €450 and boat €300.”
U: “ok, book the cheapest one”
Several alternatives

```
price:Action
   event:Price
      oneway:Event
          route:Route
tomorrow:Date
equals... plane\lor boat:Means
```

S: “What can I do for you?”
U: “how much is a trip to Stockholm tomorrow?”
S: “By plane or by boat?”
U: “both”
S: “Plane costs €450 and boat €300.”
U: “ok, book the cheapest one”
Several alternatives

price:Action

| event:Price  --· answer  --· €450 ∨ €300:Price
| oneway:Event

route:Route

...  ...  plane ∨ boat:Means

tomorrow:Date

---

S: “What can I do for you?”
U: “how much is a trip to Stockholm tomorrow?”
S: “By plane or by boat?”
U: “both”
S: “Plane costs €450 and boat €300.”
U: “ok, book the cheapest one”
Several alternatives

price:Action
  | event:Price
  | answer - - $450\lor $300:Price
  | ?Event
  | oneway:Event
route:Route
  | tomorrow:Date
  | plane\lor boat:Means
...
...

S: “What can I do for you?”
U: “how much is a trip to Stockholm tomorrow?”
S: “By plane or by boat?”
U: “both”
S: “Plane costs €450 and boat €300.”
U: “ok, book the cheapest one”
The dialogue system domain is specified in type theory

- A dialogue manager can be implemented using interactive tree building
- Unfixed tree nodes represent underspecified information
- Linked trees are used for sub-dialogue and anaphoric expressions
- Function definitions represent system answers to user questions
- User and system utterances are specified in type-theoretical grammar, ensuring consistency between surface form and internal representation
The dialogue system domain is specified in type theory
A dialogue manager can be implemented using interactive tree building
Unfixed tree nodes represent underspecified information
Linked trees are used for sub-dialogue and anaphoric expressions
Function definitions represent system answers to user questions
User and system utterances are specified in type-theoretical grammar, ensuring consistency between surface form and internal representation
Summary

- The dialogue system domain is specified in type theory
- A dialogue manager can be implemented using interactive tree building
- Unfixed tree nodes represent underspecified information
- Linked trees are used for sub-dialogue and anaphoric expressions
- Function definitions represent system answers to user questions
- User and system utterances are specified in type-theoretical grammar, ensuring consistency between surface form and internal representation
Summary

- The dialogue system domain is specified in type theory
- A dialogue manager can be implemented using interactive tree building
- Unfixed tree nodes represent underspecified information
- Linked trees are used for sub-dialogue and anaphoric expressions
- Function definitions represent system answers to user questions
- User and system utterances are specified in type-theoretical grammar, ensuring consistency between surface form and internal representation
Summary

- The dialogue system domain is specified in type theory
- A dialogue manager can be implemented using interactive tree building
- Unfixed tree nodes represent underspecified information
- Linked trees are used for sub-dialogue and anaphoric expressions
- Function definitions represent system answers to user questions
- User and system utterances are specified in type-theoretical grammar, ensuring consistency between surface form and internal representation
Summary

- The dialogue system domain is specified in type theory
- A dialogue manager can be implemented using interactive tree building
- Unfixed tree nodes represent underspecified information
- Linked trees are used for sub-dialogue and anaphoric expressions
- Function definitions represent system answers to user questions
- User and system utterances are specified in type-theoretical grammar, ensuring consistency between surface form and internal representation
What have I not discussed?

- **Commands**: which they are and how to define them
- **Feedback**: how it can be incorporated
- ** Corrections**: commands for deleting nodes in the tree
- **Dependent types**: can be used for info depending on other info
- **Implementation**: the dialogue model is *not* implemented yet
What have I not discussed?

- **Commands**: which they are and how to define them
- **Feedback**: how it can be incorporated
- **Corrections**: commands for deleting nodes in the tree
- **Dependent types**: can be used for info depending on other info
- **Implementation**: the dialogue model is *not* implemented yet
What have I not discussed?

- **Commands**: which they are and how to define them
- **Feedback**: how it can be incorporated
- **Corrections**: commands for deleting nodes in the tree
- **Dependent types**: can be used for info depending on other info
- **Implementation**: the dialogue model is *not* implemented yet
What have I not discussed?

- Commands: which they are and how to define them
- Feedback: how it can be incorporated
- Corrections: commands for deleting nodes in the tree
- Dependent types: can be used for info depending on other info
- Implementation: the dialogue model is *not* implemented yet
What have I not discussed?

- **Commands**: which they are and how to define them
- **Feedback**: how it can be incorporated
- **Corrections**: commands for deleting nodes in the tree
- **Dependent types**: can be used for info depending on other info
- **Implementation**: the dialogue model is *not* implemented yet