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, ...
- 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 $t_1 : T_1$, ..., $t_n : T_n$

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

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°?"
- 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°
- function $T_1 \times \cdots \times T_n \to T \iff$ implication $T_1^\circ \wedge \ldots \wedge T_n^\circ \to T^\circ$
- term $t : T \iff$ proof of T°
- building a term $t : T \iff$ proving a proposition T°

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°?"
- term containing metavariables $\iff$ incomplete proof tree

Specifying a theory

Specifying a theory consists of giving:

- the basic types (*Action*, *Price*, *Event*, *Date*, *City*)
- the constants (*sthlm* : *City*, *today* : *Date*, €450 : *Price*)
- the functions (*book* : *Event* $\longrightarrow$ *Action*, *hotel* : *City* $\times$ *Date* $\longrightarrow$ *Event*)

Specifying the utterances

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

- system questions corresponding to basic types
  ($?Action \mapsto$ "What do you want to do?",
  $?Date \mapsto$ "What date do you mean?")
- utterances corresponding to constants:
  ($sthlm \mapsto$ "Stockholm", €450 $\mapsto$ "fourhundred and fifty Euros")
- complex utterances for functions:
  ($book(x) \mapsto$ "book (an event | x), please",
  $hotel(x, y) \mapsto$ "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

Terms and Trees

An example term

book(oneway(route(to(sthlm), from(lon), boat), tomorrow))) : Action

Terms and Trees

A fully typed variant

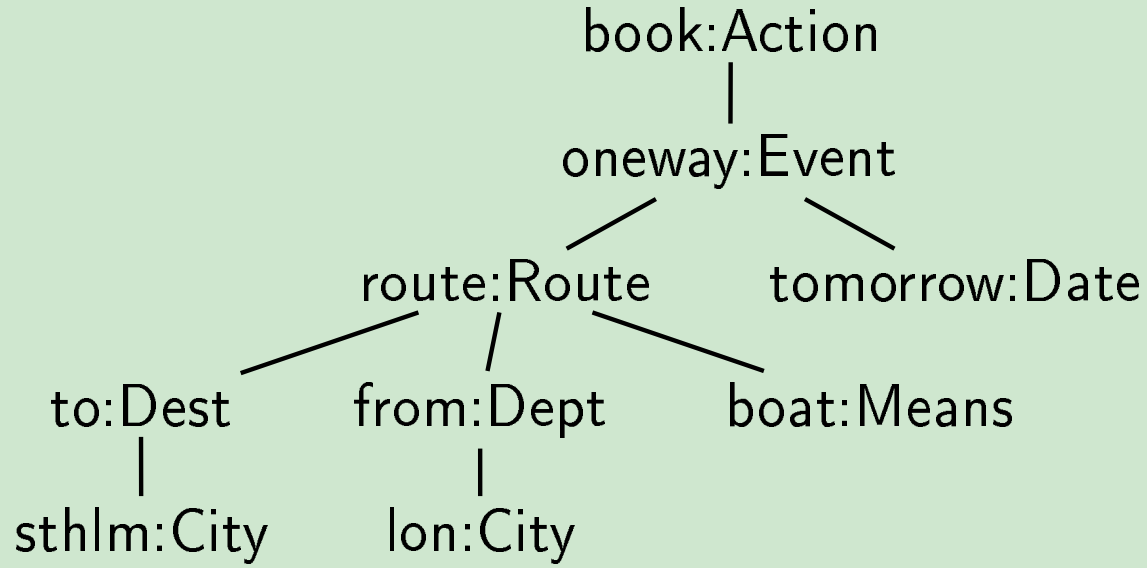
book(oneway(route(to(sthlm:City):Dest, from(lon:City):Dept,
boat:Means), tomorrow:Date):Route):Event):Action

Terms and Trees

A fully typed variant

book(oneway(route(to(sthlm:City):Dest, from(lon:City):Dept,
boat:Means), tomorrow:Date):Route):Event):Action

The corresponding tree



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 $?f_1 \vee \ldots \vee f_n : T$ 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, refinining metavariables, ...

The initial tree is the single focused node ?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 $?f_1 \vee \ldots \vee f_n : T$ 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, refinining metavariables, ...

The initial tree is the single focused node ?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 $?f_1 \vee \ldots \vee f_n : T$ 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, refinining metavariables, ...

The initial tree is the single focused node ?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 $?f_1 \vee \ldots \vee f_n : T$ 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, refinining metavariables, ...

The initial tree is the single focused node ?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 $?f_1 \vee \ldots \vee f_n : T$ 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, refinining metavariables, ...

The initial tree is the single focused node ?Action.

System-driven dialogue

?Action

S: ask(?Action) $\Rightarrow$
"What do you want to do?"

System-driven dialogue

?price∨book:Action

`refine-down`

System-driven dialogue

?price∨book:Action

S: ask(priceIssue(?Price):Action ∨ book(?Event):Action) ⇒
"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)

System-driven dialogue

`integrate` book(?Event):Action

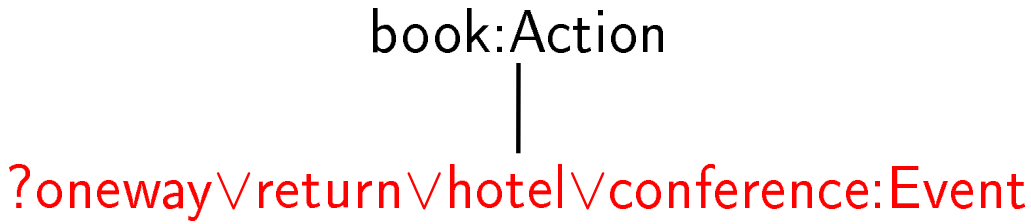
System-driven dialogue

`select-next`

System-driven dialogue

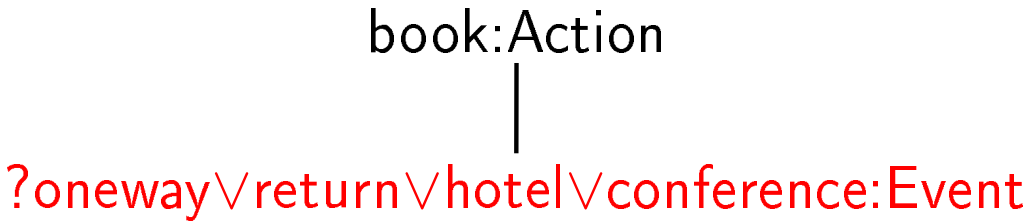
S: ask(?Event)⇒
"What event are you interested in?"

System-driven dialogue



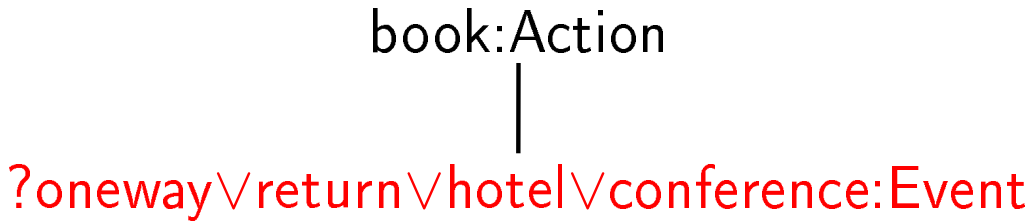
`refine-down`

System-driven dialogue



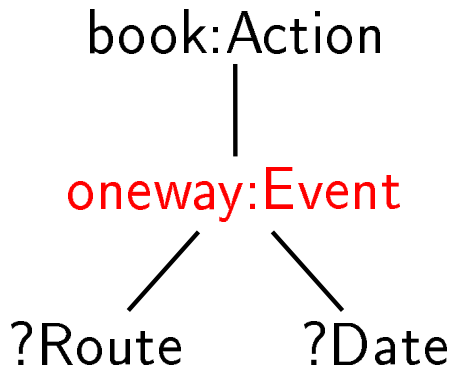
S: ask(oneway ∨ return ∨ hotel ∨ conference) ⇒
"Oneway trip, return trip, hotel or conference?"

System-driven dialogue



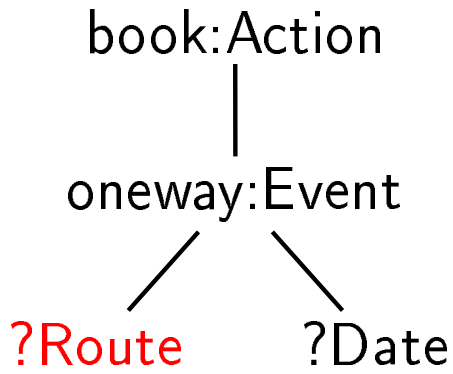
U: “oneway trip” ⇒
answer(oneway(?Route,?Date):Event)

System-driven dialogue



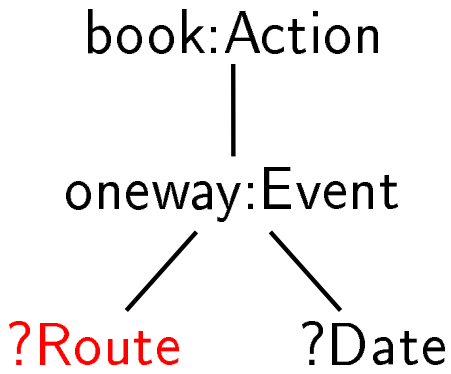
`integrate` oneway(?Route,?Date):Event

System-driven dialogue



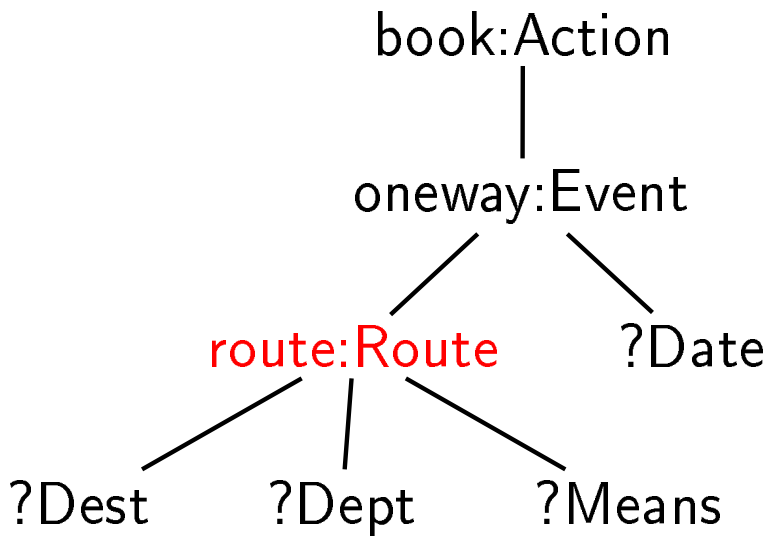
`select-next`

System-driven dialogue



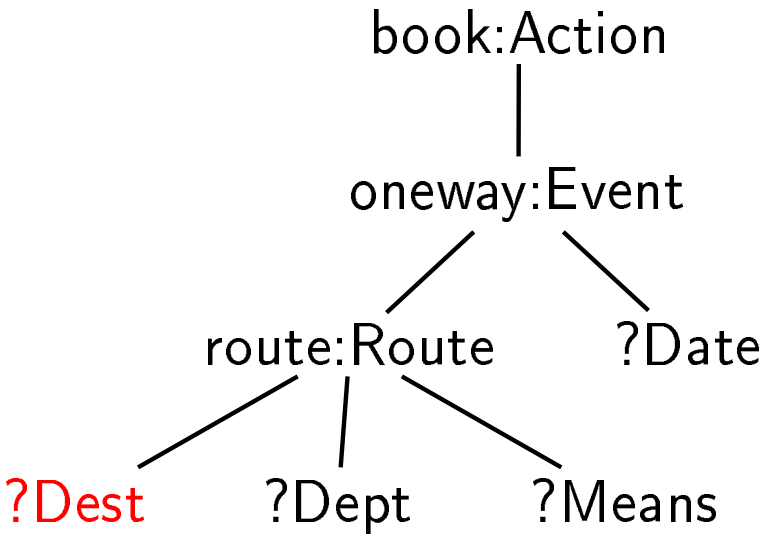
S: ask(?Route) ⇒
"Which route do you want?"

System-driven dialogue



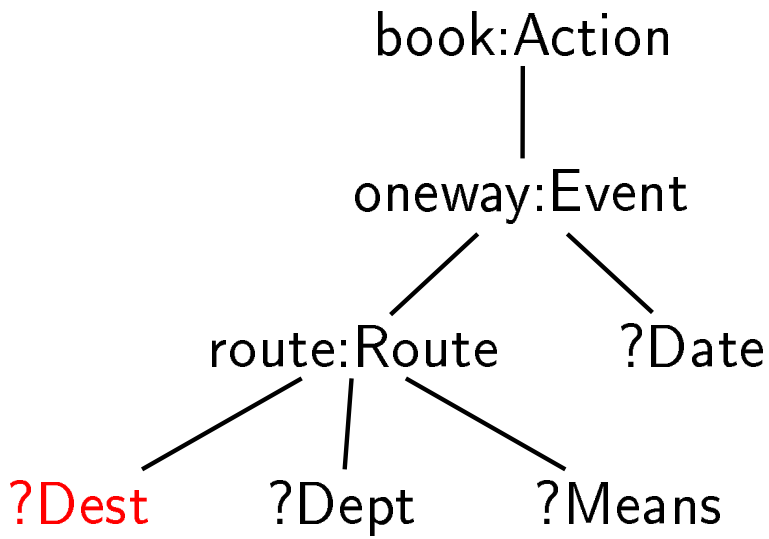
`refine-down`

System-driven dialogue



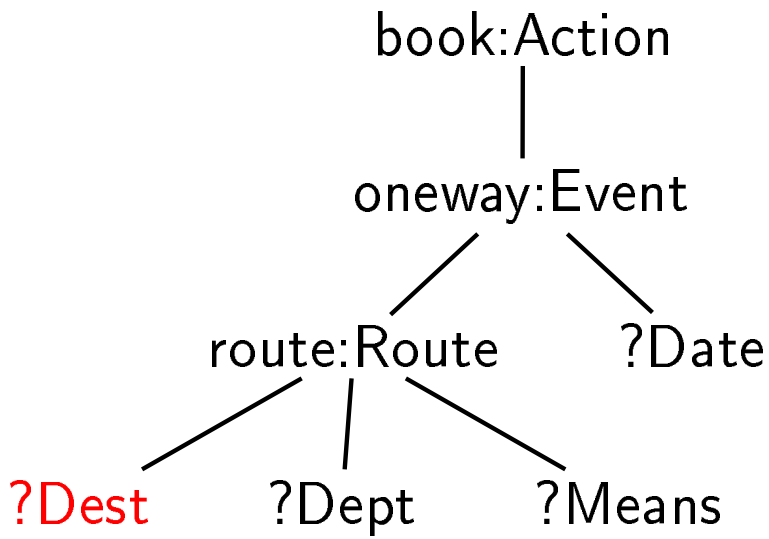
`select-next`

System-driven dialogue



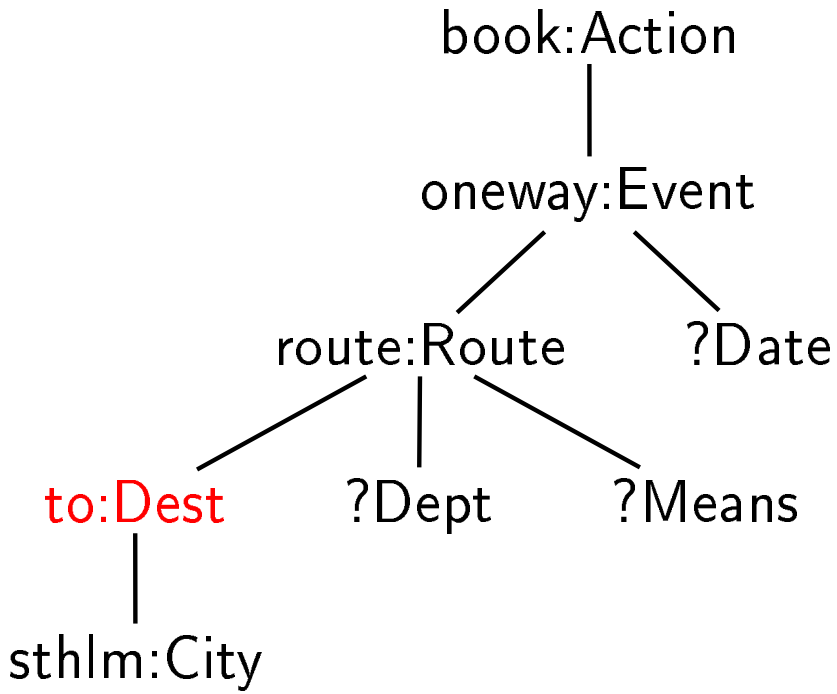
S: ask(?Dest) ⇒
"To which city are you heading?"

System-driven dialogue



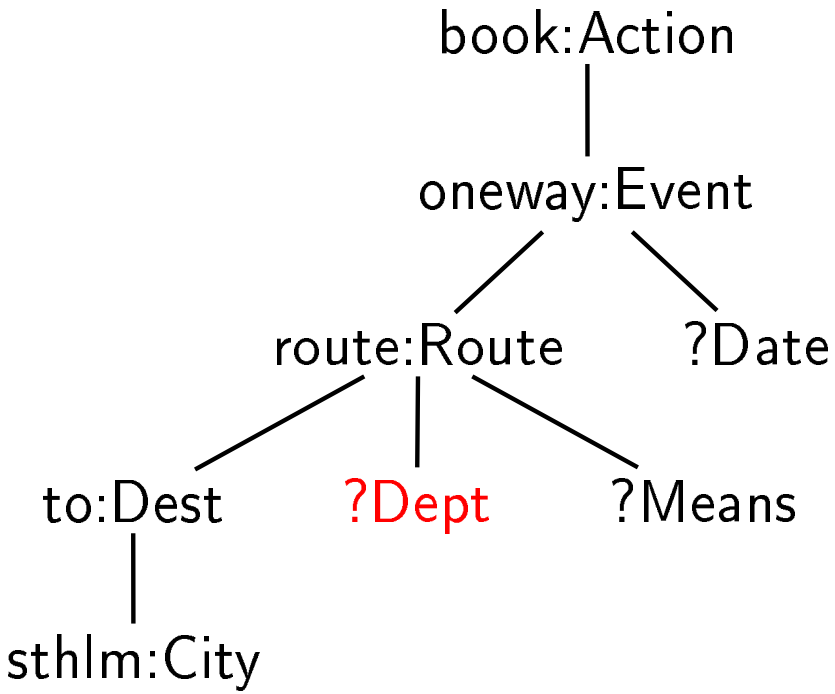
U: “to Stockholm” ⇒
answer(to(sthlm:City):Dest)

System-driven dialogue



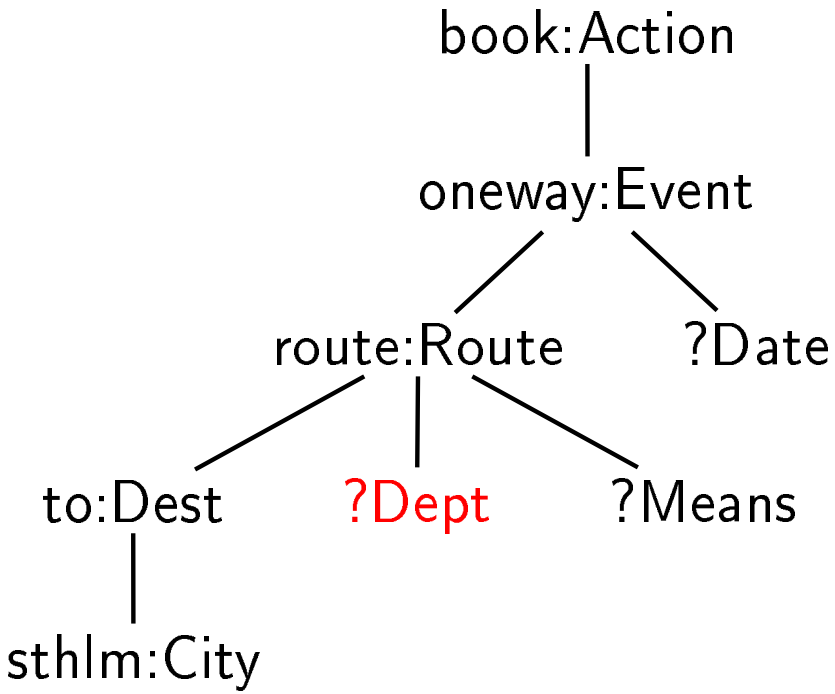
`integrate` to(sthlm:City):Dest

System-driven dialogue



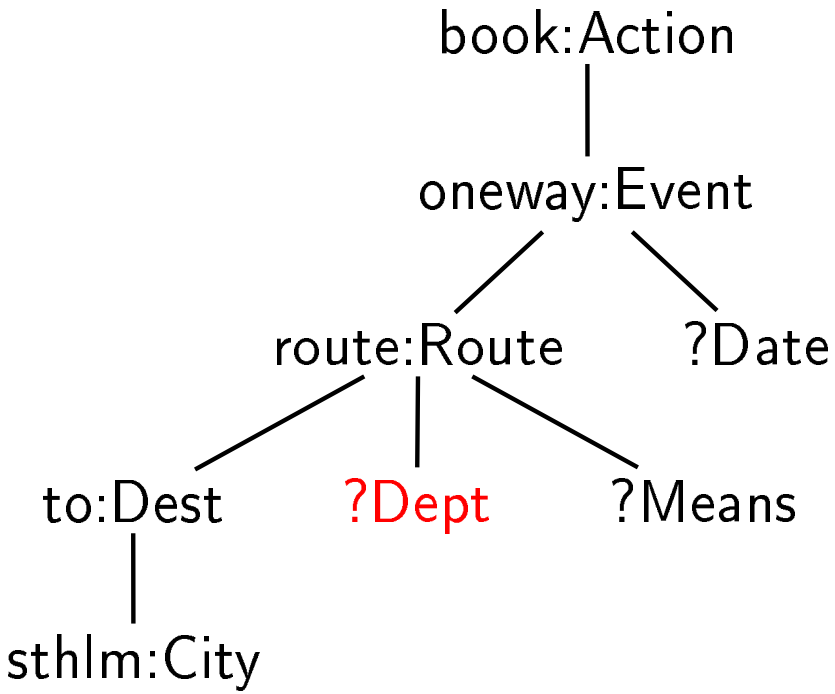
`select-next`

System-driven dialogue



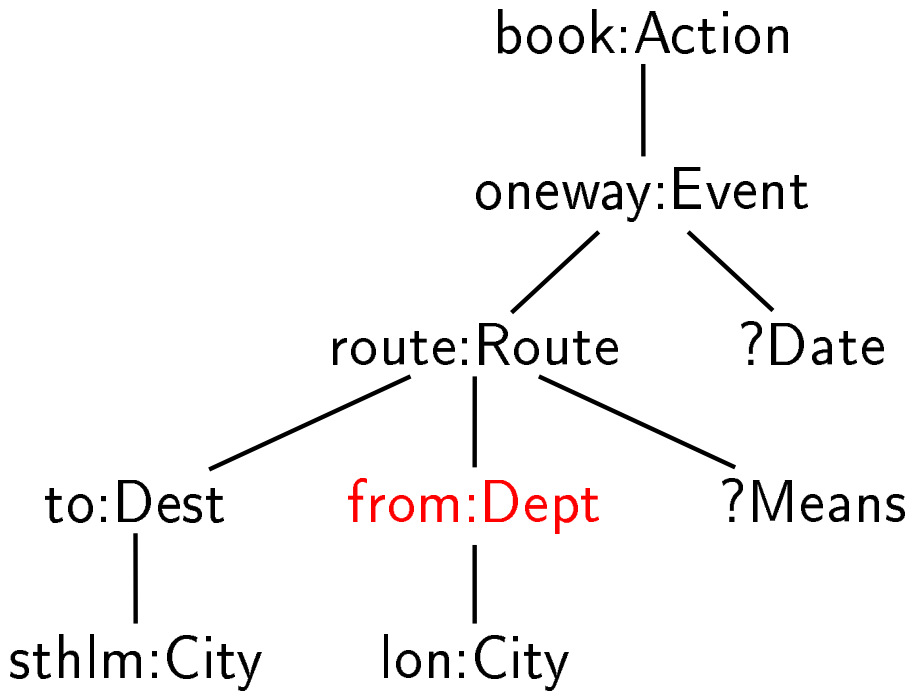
S: ask(?Dept) ⇒
"From where are you leaving?"

System-driven dialogue



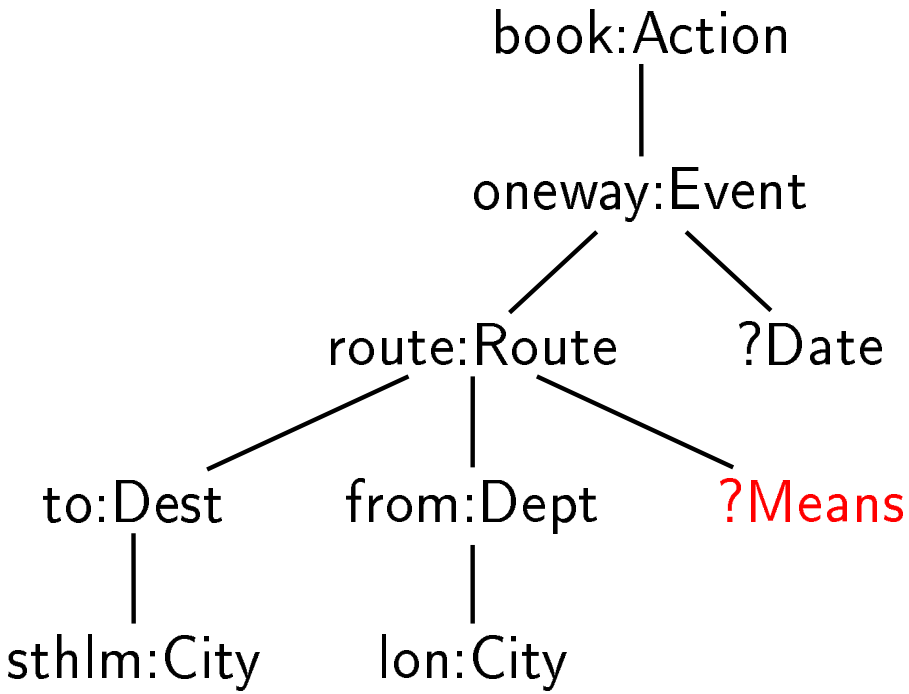
U: “from London” $\Rightarrow$
answer(from(lon:City):Dept)

System-driven dialogue



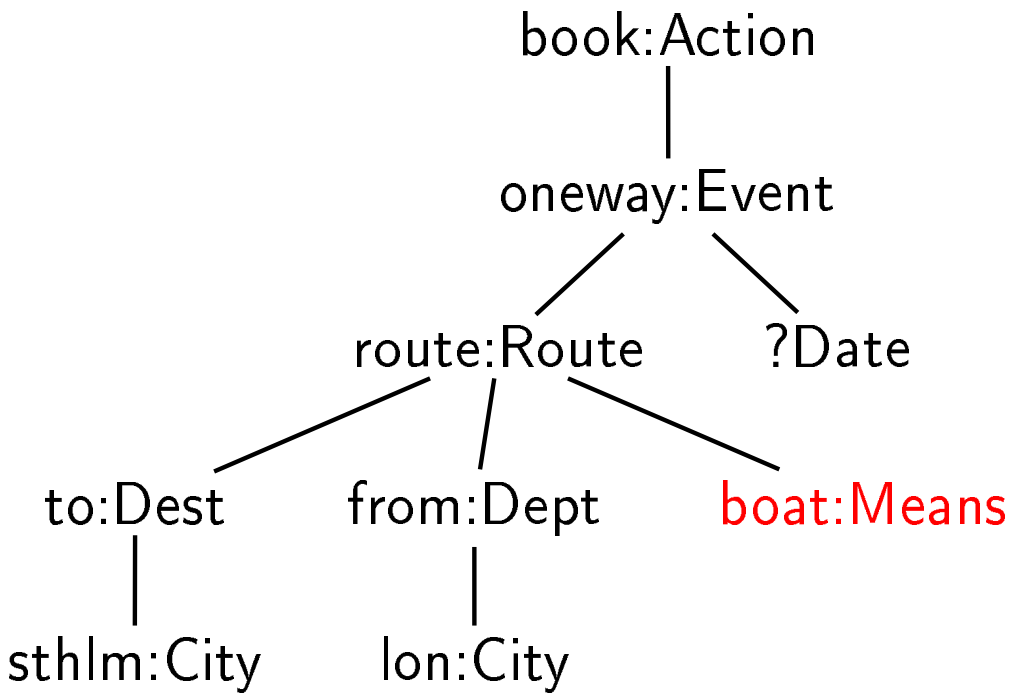
`integrate` from(lon:City):Dept

System-driven dialogue



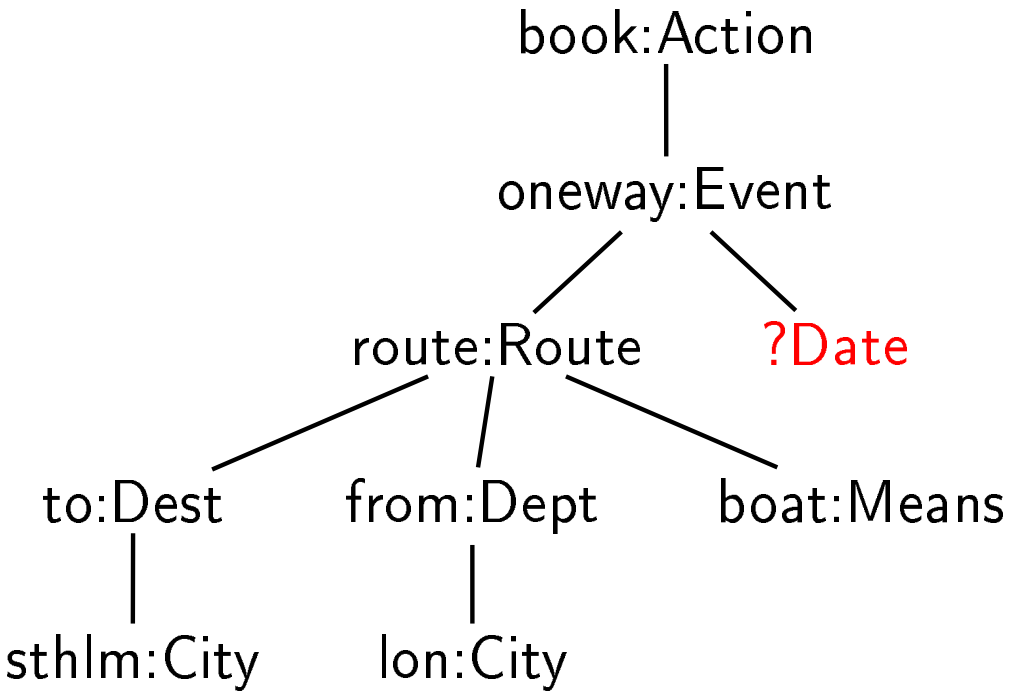
(etcetera. . .)

System-driven dialogue



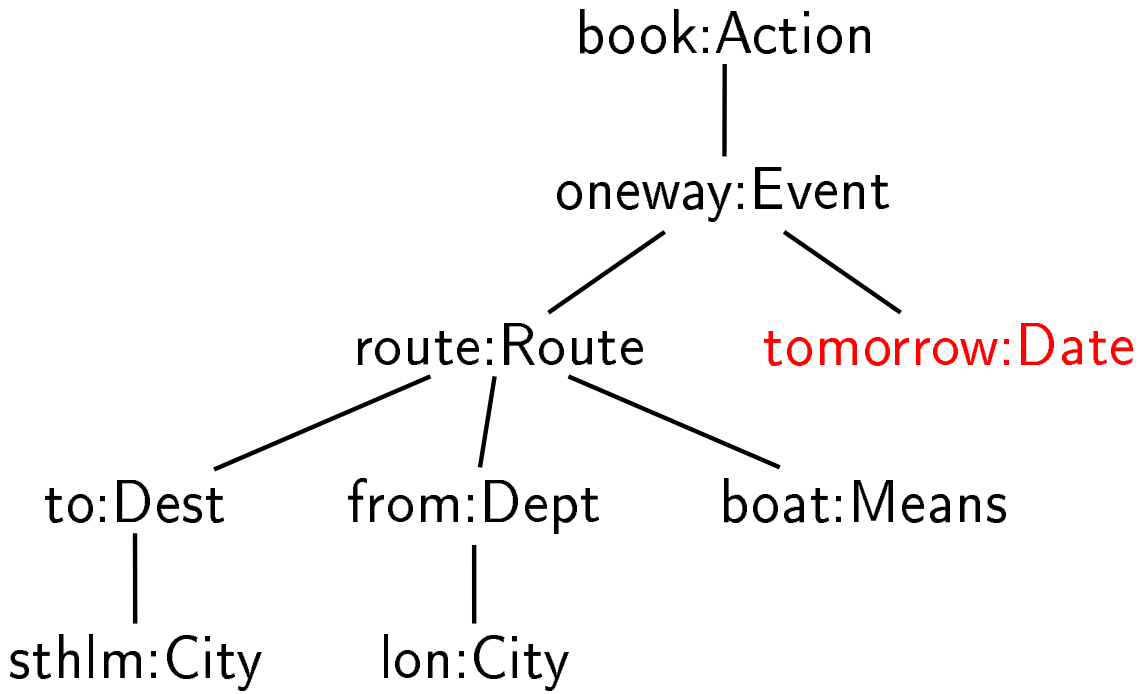
(etcetera. . .)

System-driven dialogue



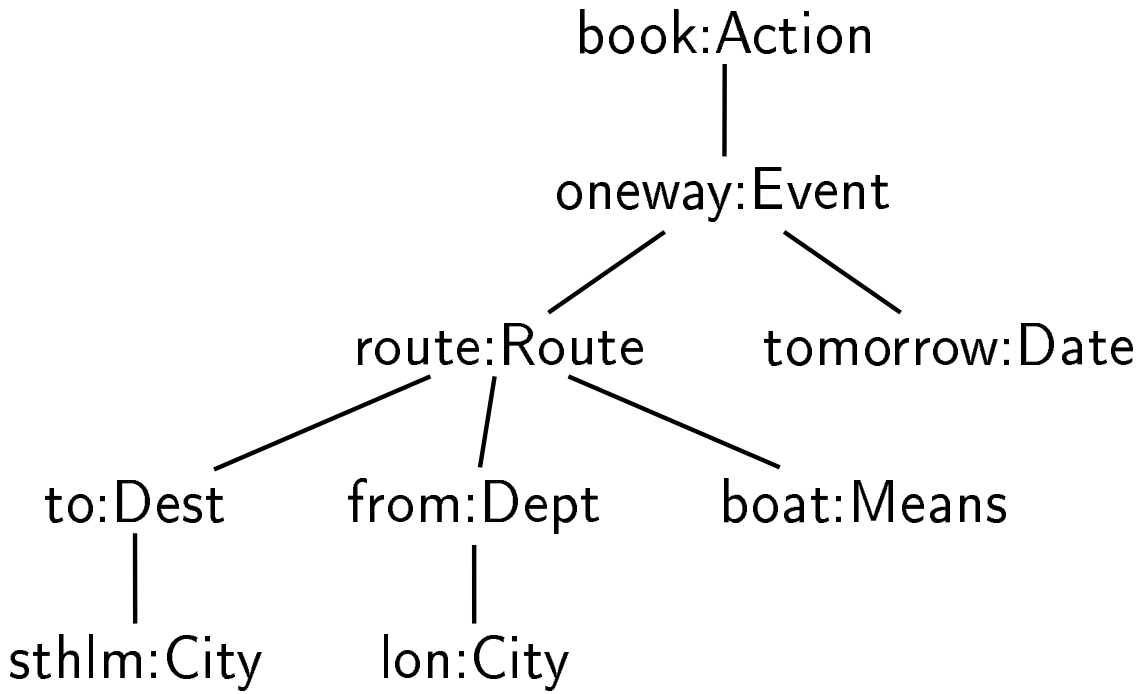
(etcetera. . .)

System-driven dialogue



(etcetera. . .)

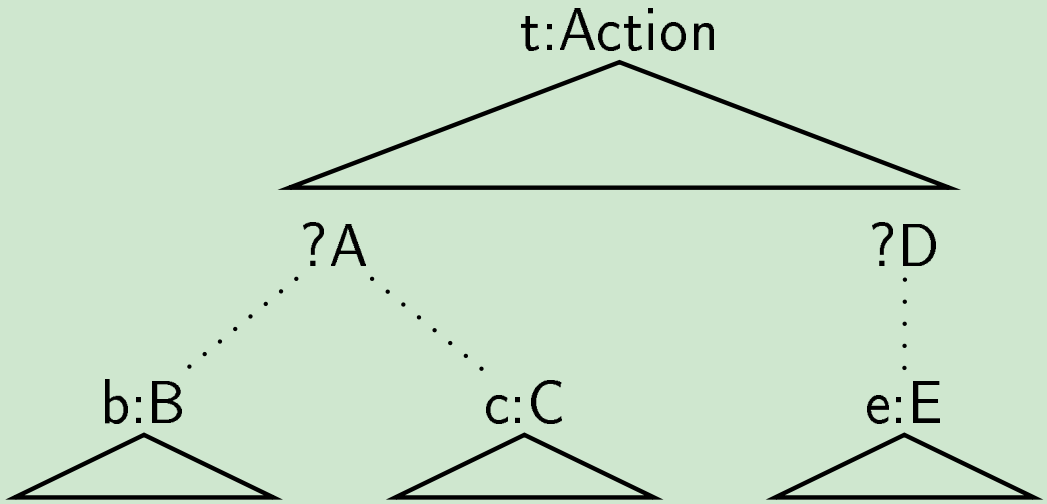
System-driven dialogue



Voilà!

Underspecified tree nodes

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 $\Rightarrow^*$ αBβ, ...
- 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.

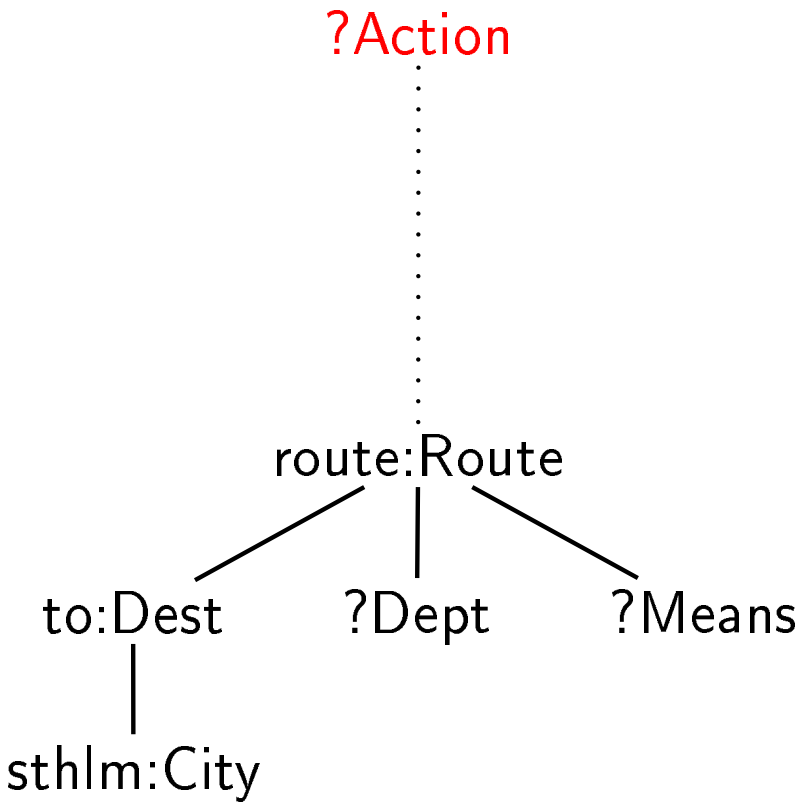
- Correspond to known dialogue strategies?

Strategy 1: Top-down refinement

?Action

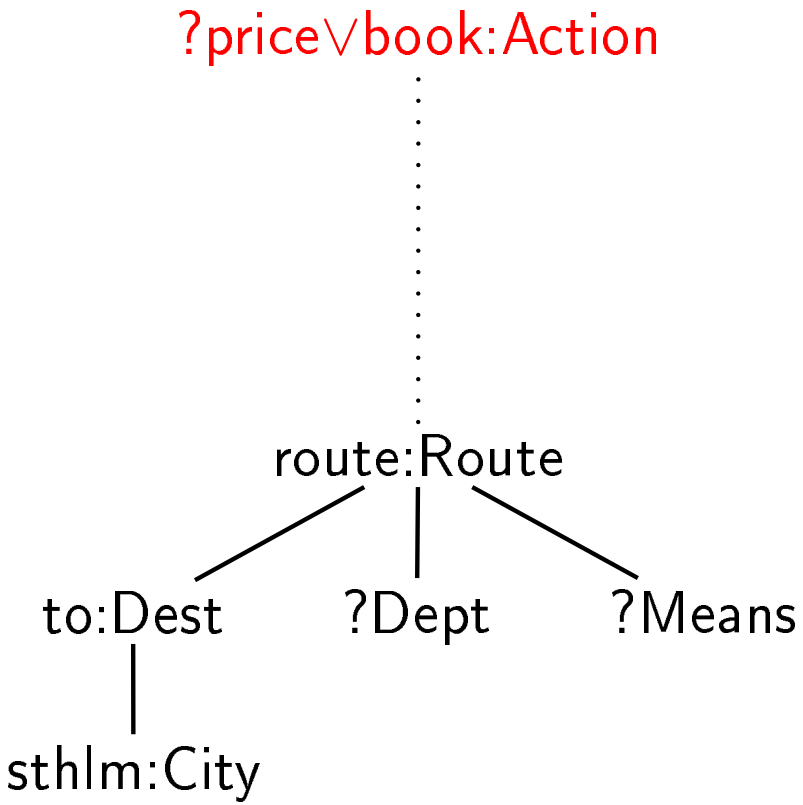
S: ask(?Action) $\Rightarrow$
"What do you want to do?"

Strategy 1: Top-down refinement



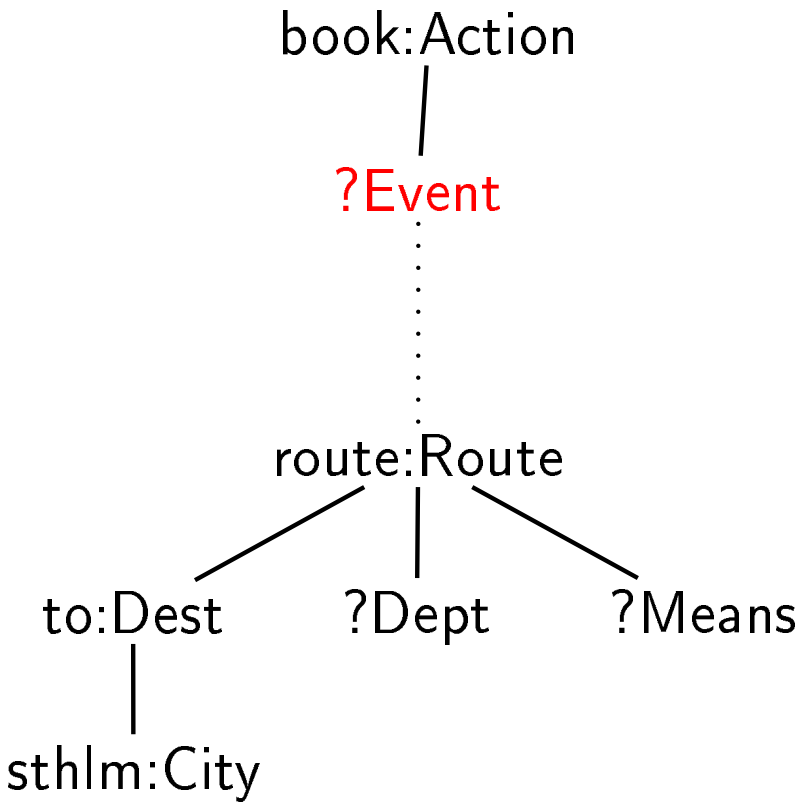
U: "go to Stockholm"⇒
answer(route(to(sthlm),?City,?Means):Route)

Strategy 1: Top-down refinement



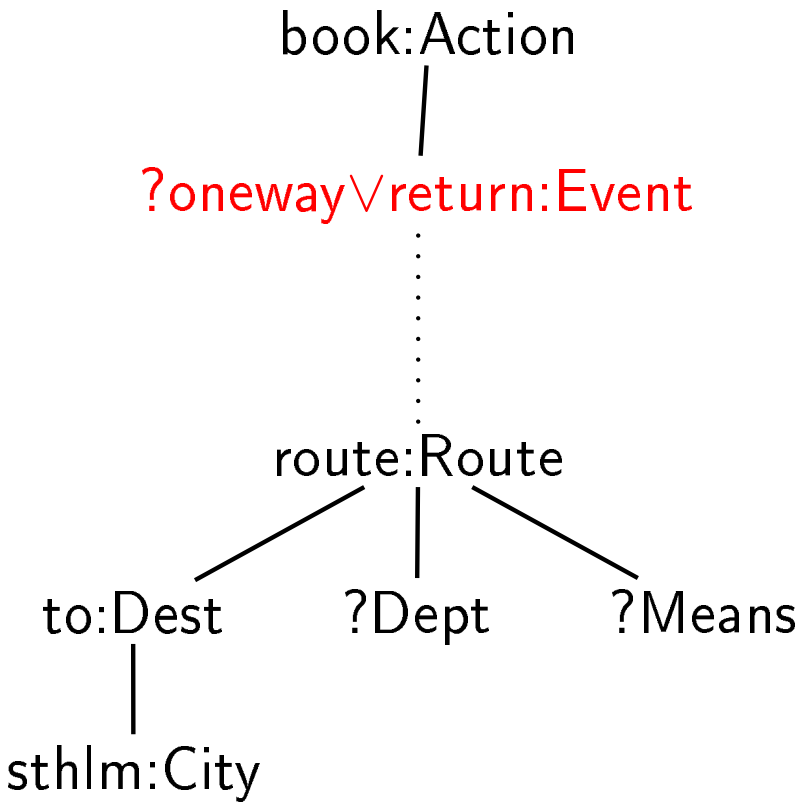
S: ask(priceIssue(?):Action ∨ book(?):Action) ⇒
"Do you want to ask for the price or book an event?"

Strategy 1: Top-down refinement



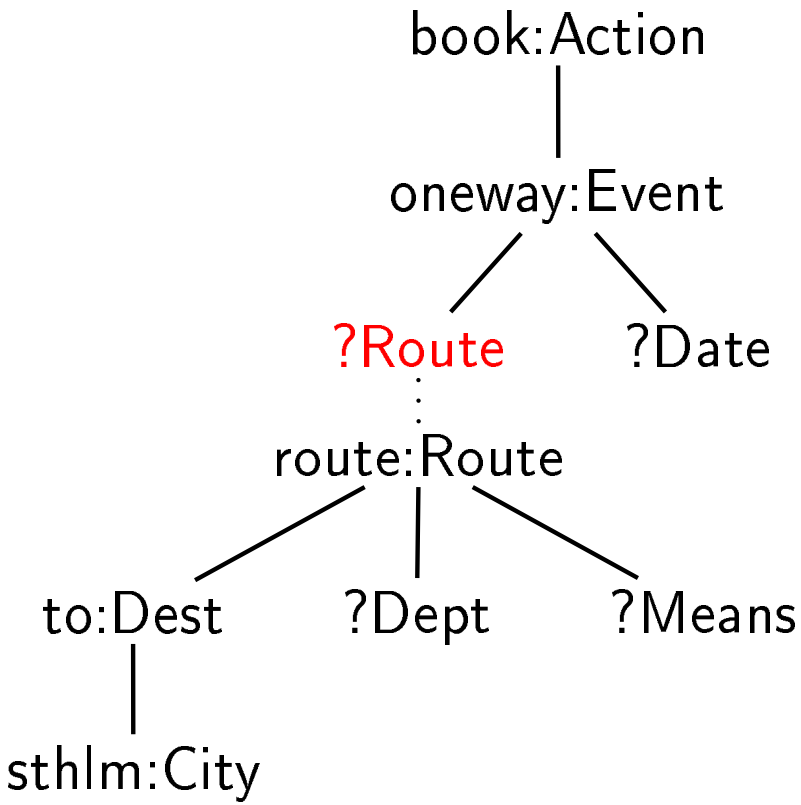
U: "book an event"⇒
answer(book(?Event):Action)

Strategy 1: Top-down refinement



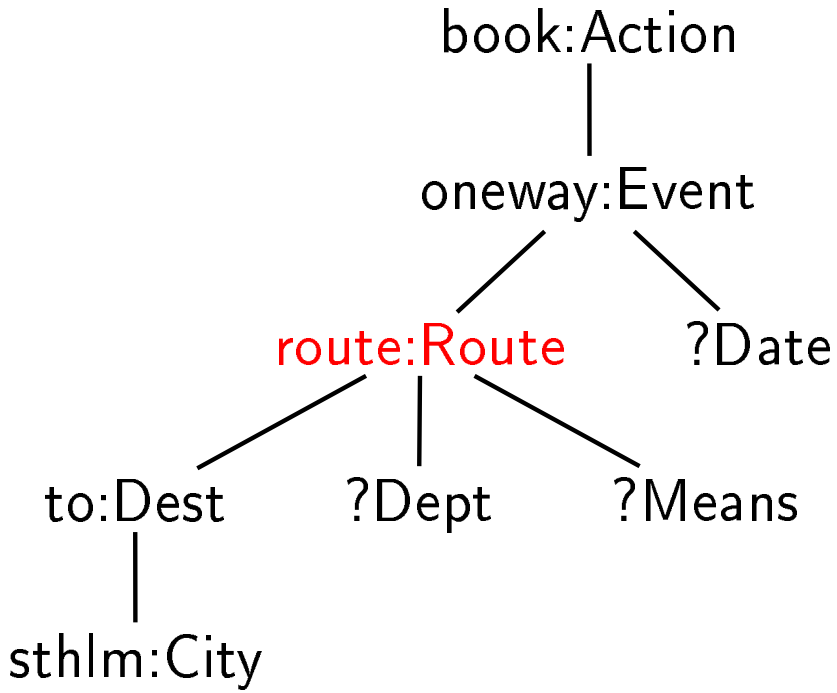
S: ask(oneway(?,?):Event ∨ return(?,?,?):Event)⇒
"Do you want a oneway trip or a return trip?"

Strategy 1: Top-down refinement



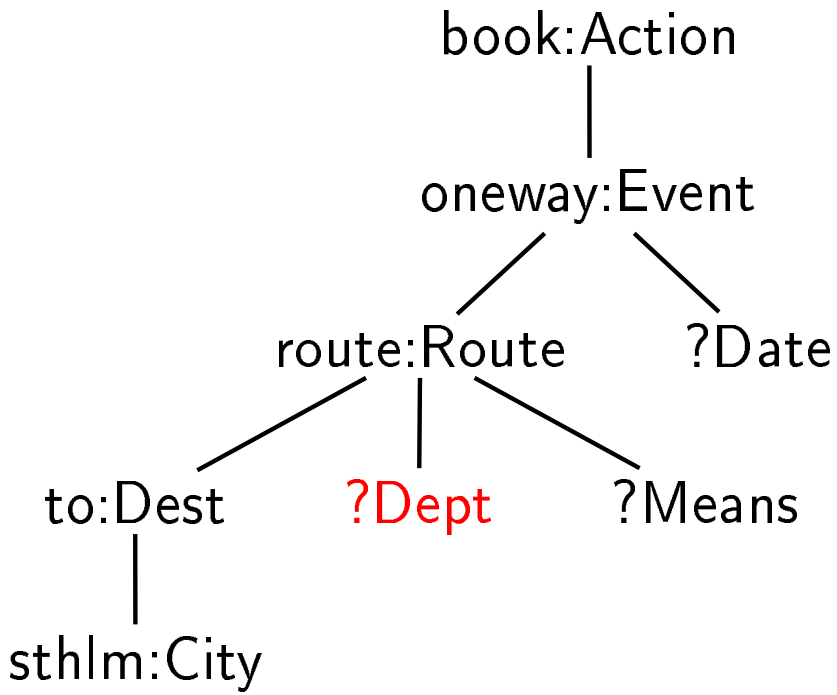
U: “oneway”⇒
answer(oneway(?Route,?Date):Event)

Strategy 1: Top-down refinement



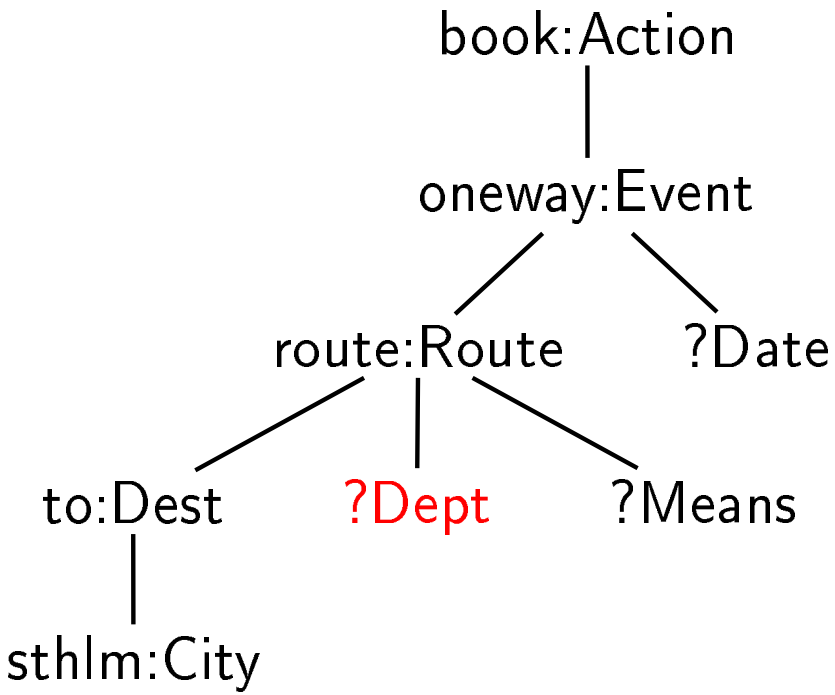
`refine-down`

Strategy 1: Top-down refinement



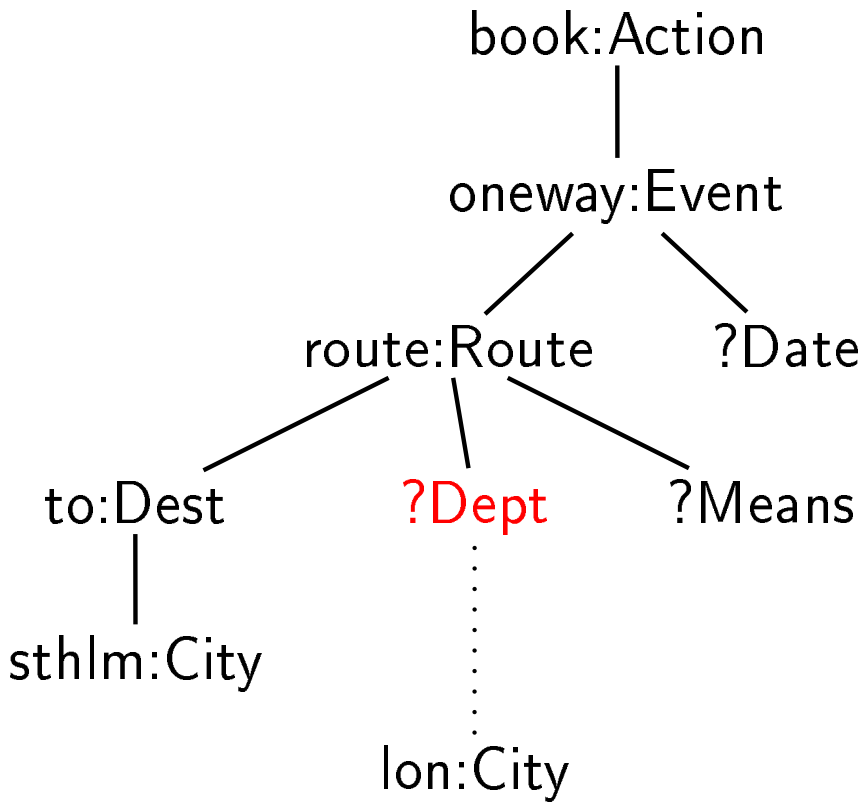
`select-next`

Strategy 1: Top-down refinement



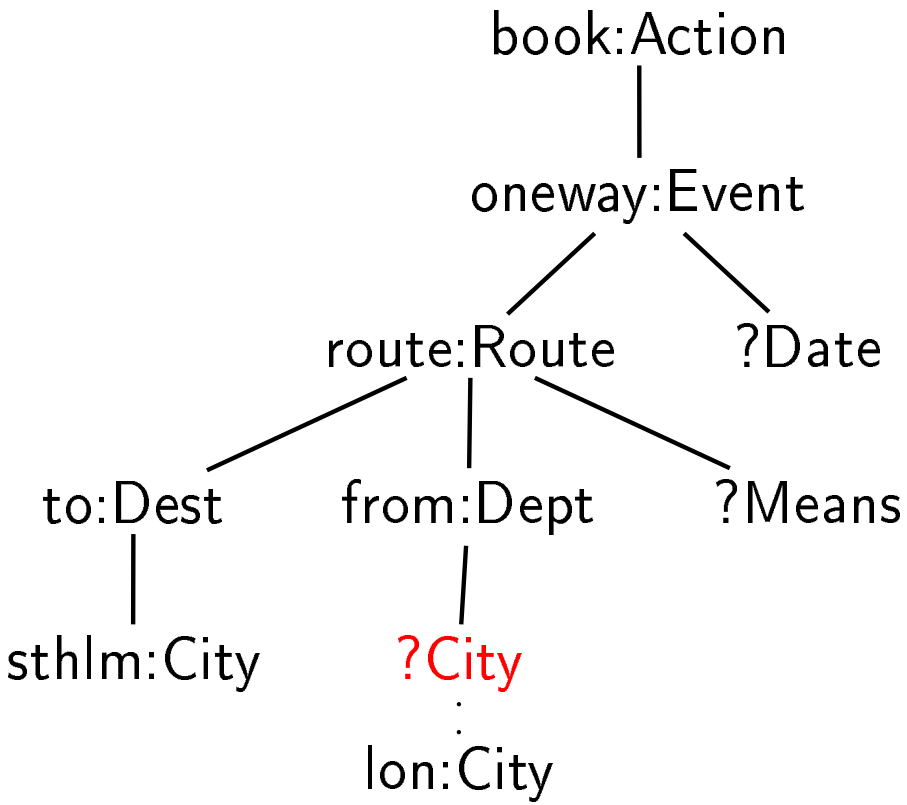
S: ask(?Dept) ⇒
"From where are you leaving?"

Strategy 1: Top-down refinement



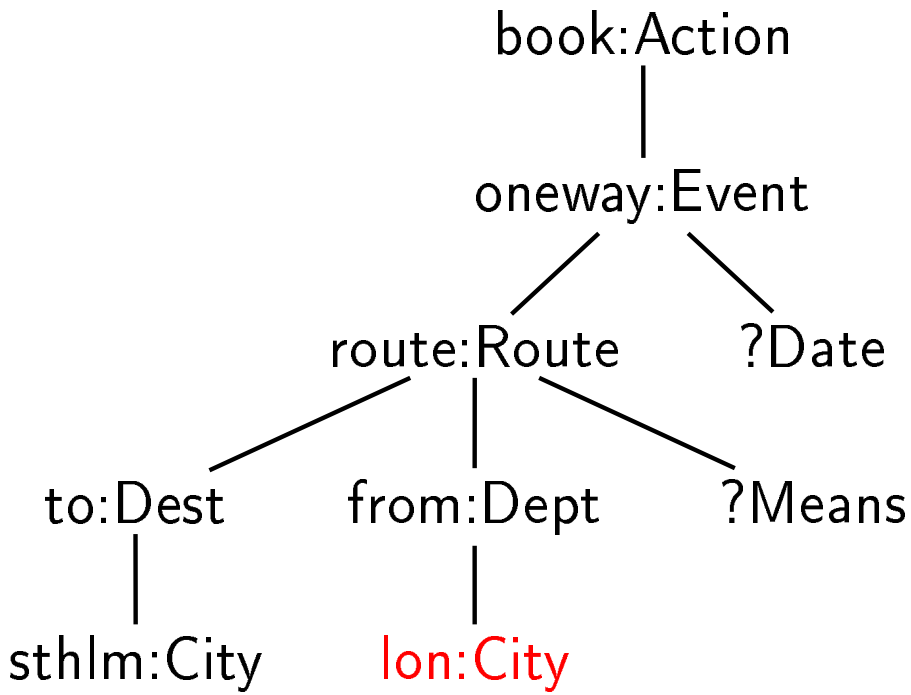
U: "London"⇒
answer(lon:City)

Strategy 1: Top-down refinement



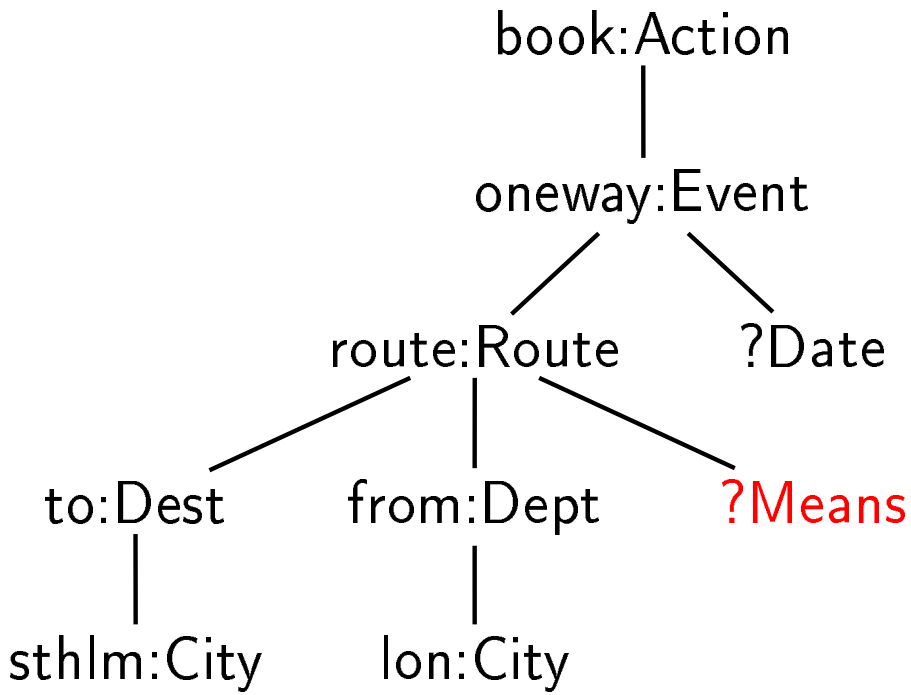
`refine-down`

Strategy 1: Top-down refinement



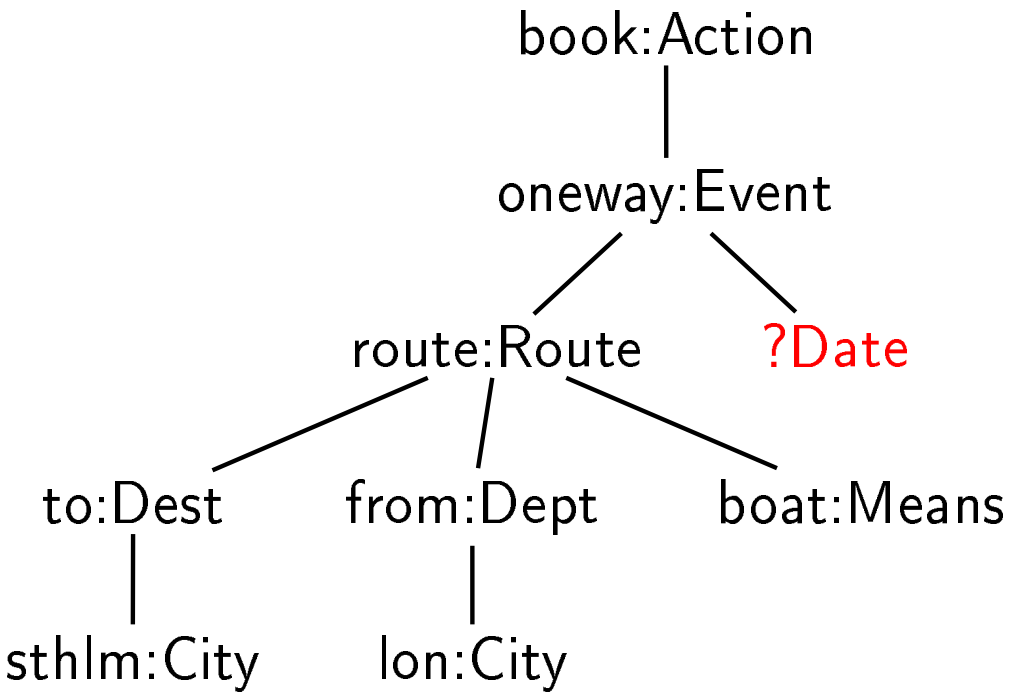
`refine-down`

Strategy 1: Top-down refinement



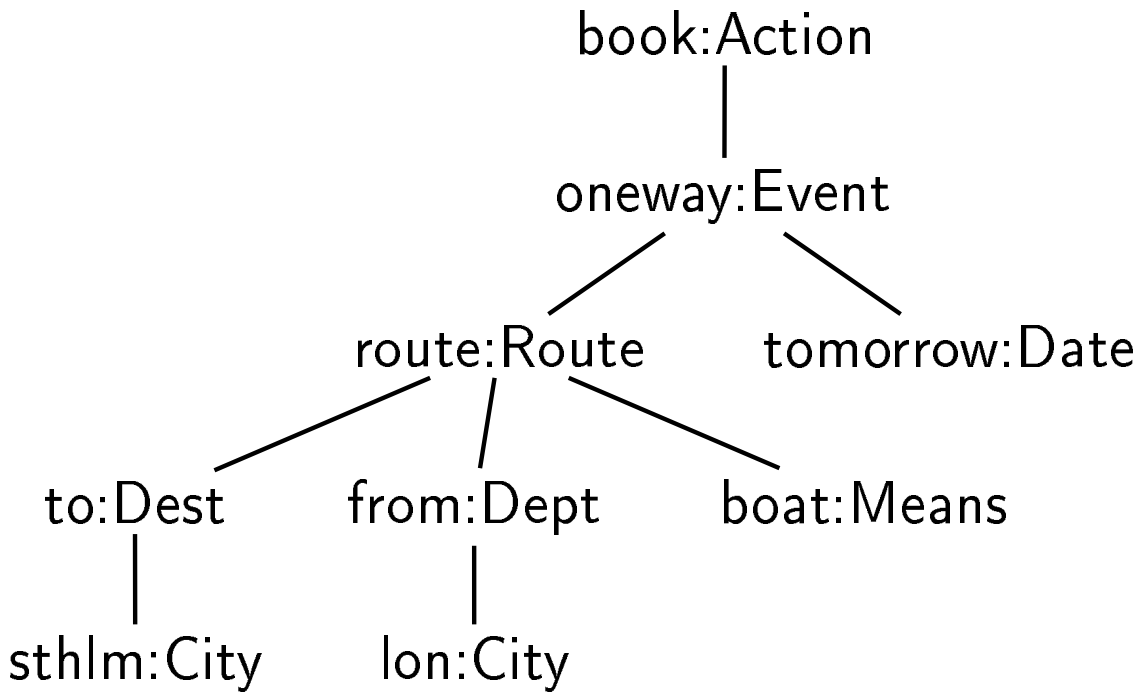
(etcetera...)

Strategy 1: Top-down refinement



(etcetera. . .)

Strategy 1: Top-down refinement



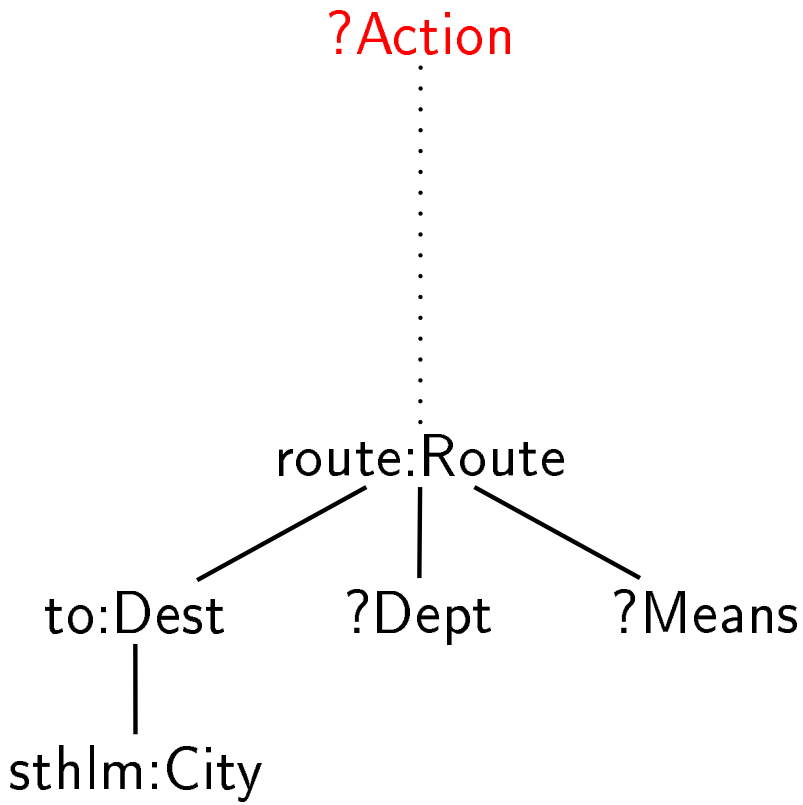
(etcetera. . .)

Strategy 2: Bottom-up refinement

?Action

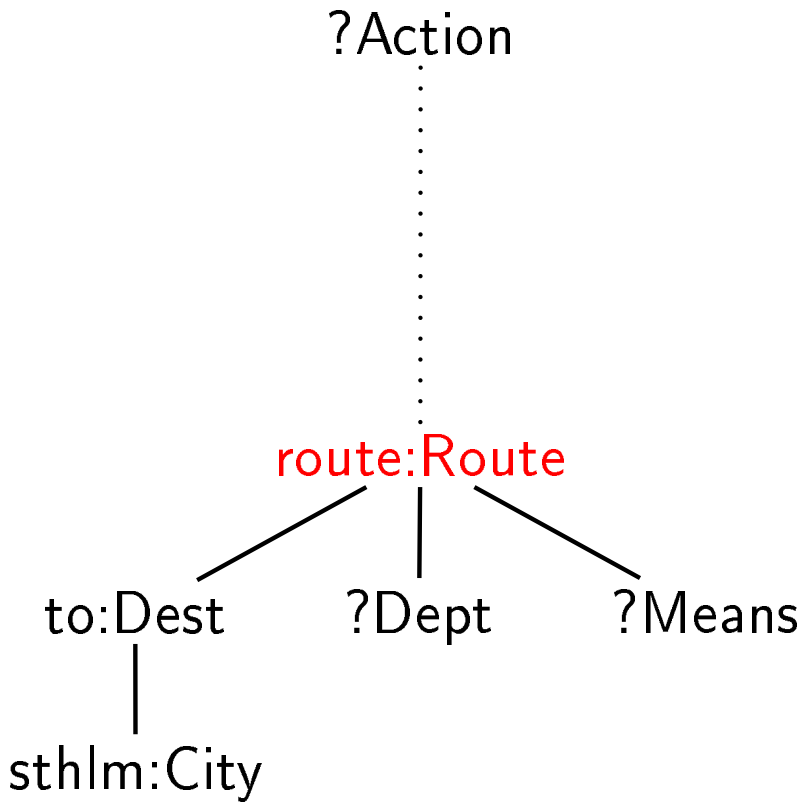
S: ask(?Action) ⇒
"What do you want to do?"

Strategy 2: Bottom-up refinement



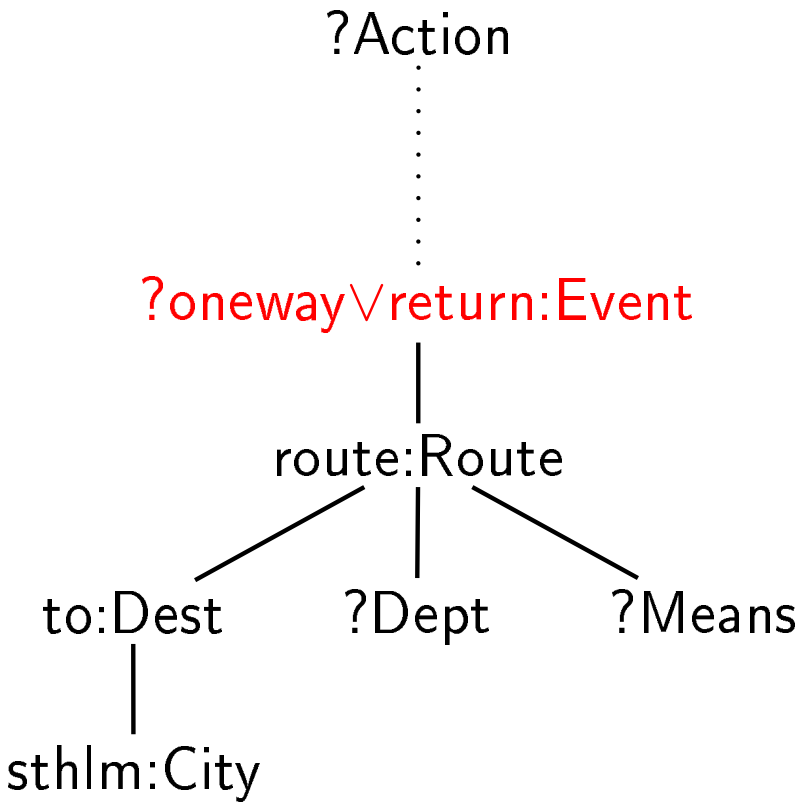
U: “go to Stockholm”⇒
answer(route(to(sthlm),?Dept,?Means):Route)

Strategy 2: Bottom-up refinement



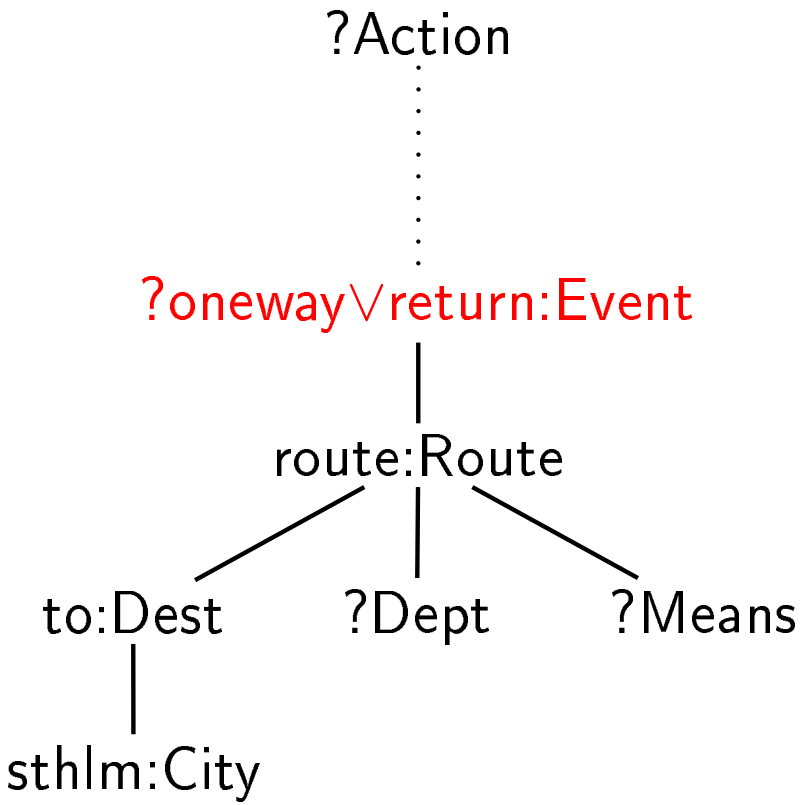
`focus-down`

Strategy 2: Bottom-up refinement



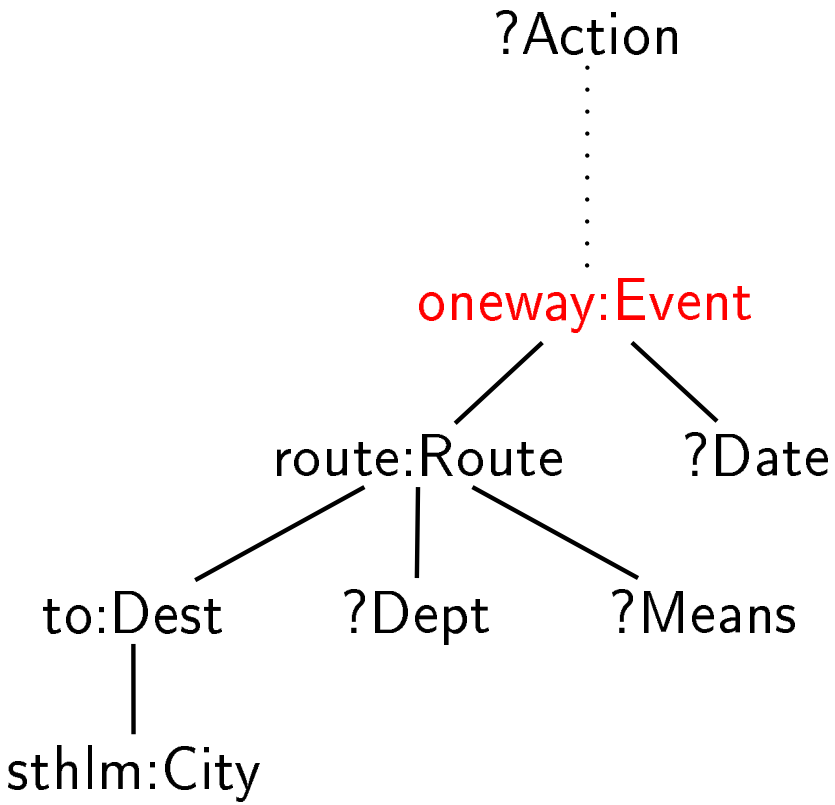
`refine-up`

Strategy 2: Bottom-up refinement



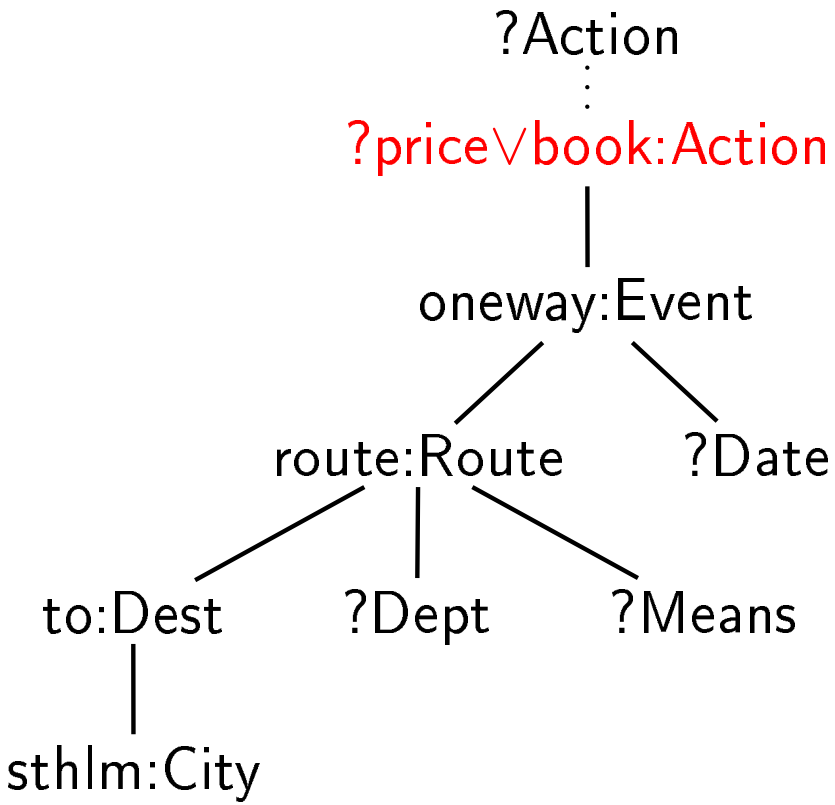
S: ask(oneway(?,?):Event ∨ return(?,?,?):Event)⇒
"Do you want a oneway trip or a return trip?"

Strategy 2: Bottom-up refinement



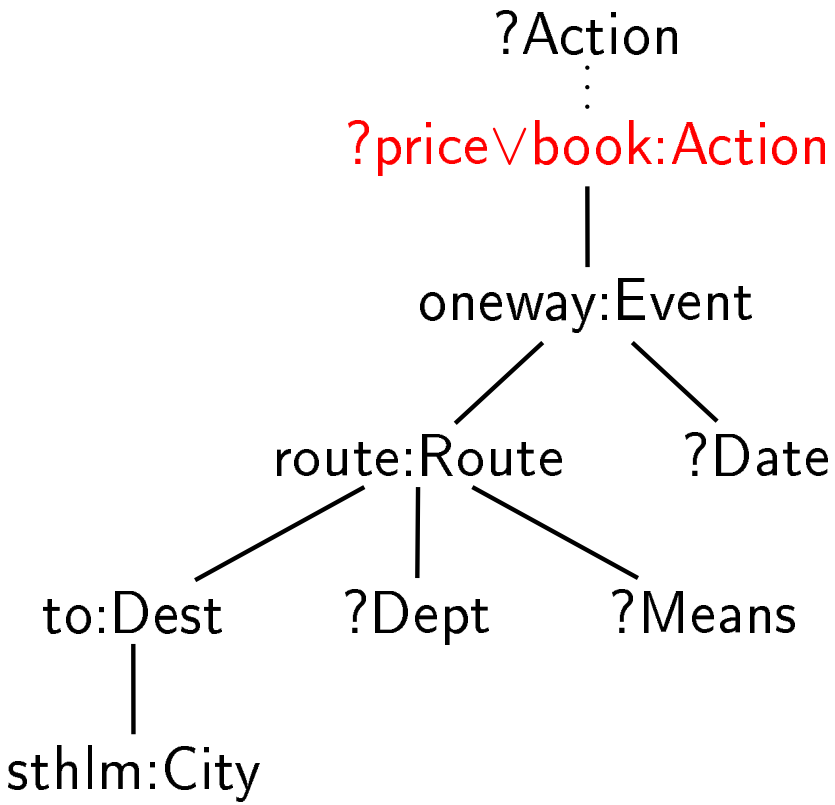
U: “oneway”⇒
answer(oneway(?Route,?Date):Event)

Strategy 2: Bottom-up refinement



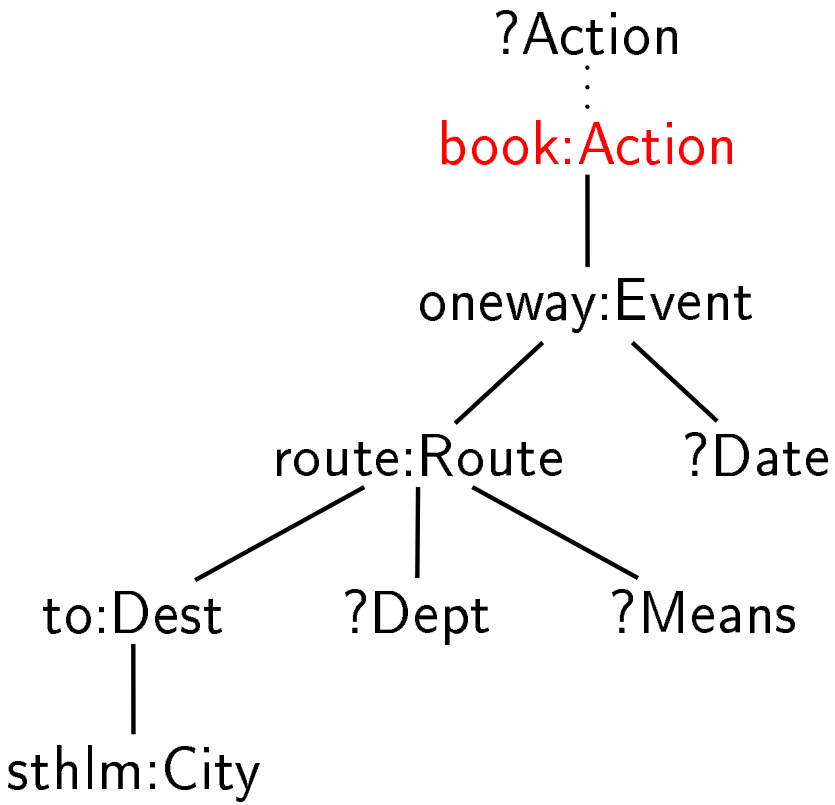
`refine-up`

Strategy 2: Bottom-up refinement



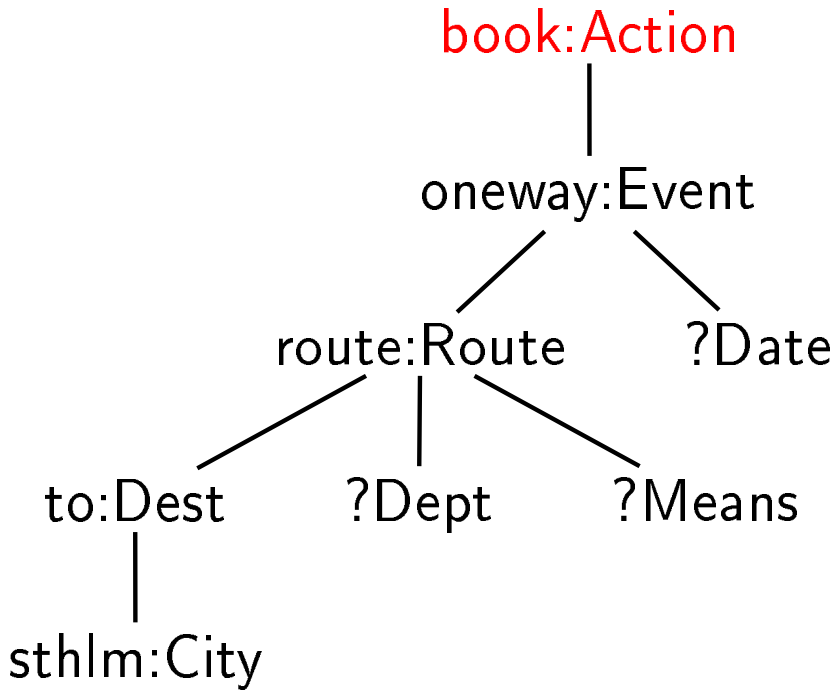
S: ask(priceIssue(?):Action ∨ book(?):Action) ⇒
"Do you want to ask for the price or book an event?"

Strategy 2: Bottom-up refinement



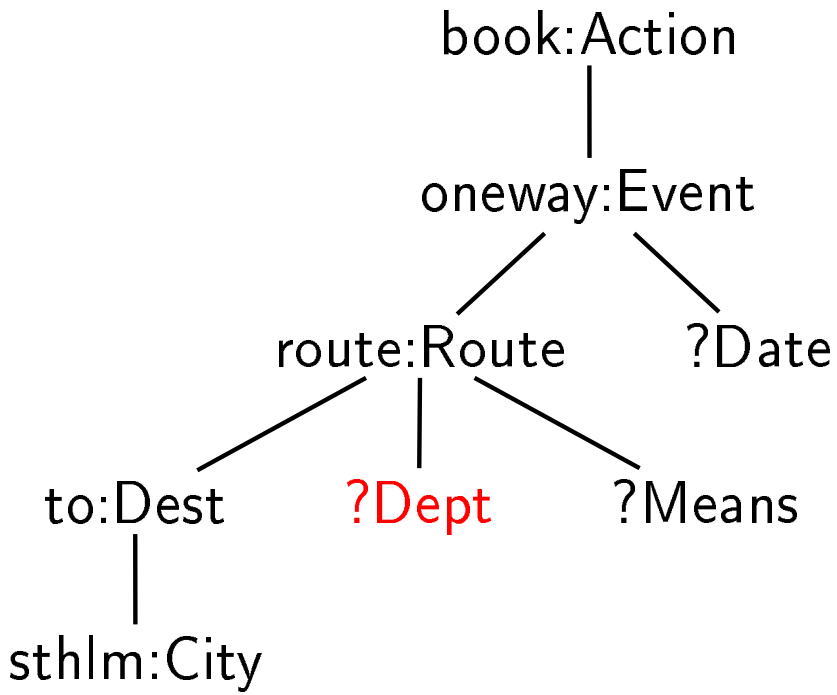
U: "book an event"⇒
answer(book(?Event):Action)

Strategy 2: Bottom-up refinement



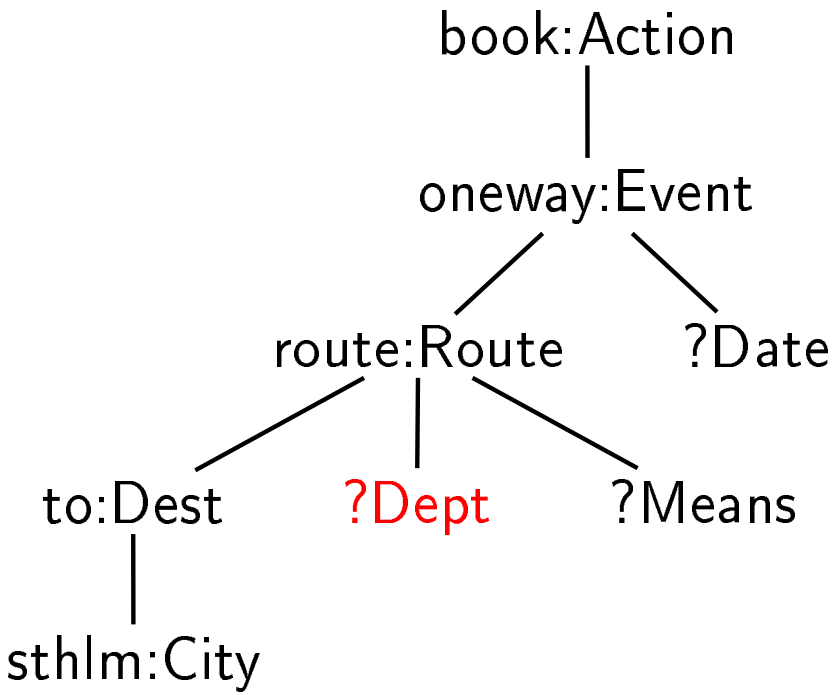
`refine-up`

Strategy 2: Bottom-up refinement



`select-next`

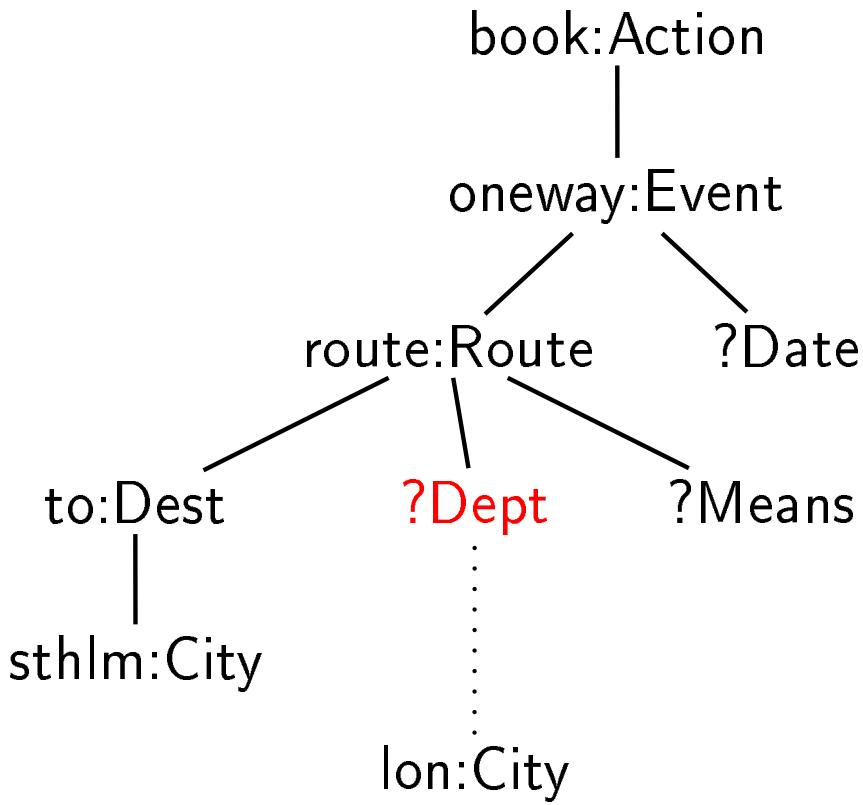
Strategy 2: Bottom-up refinement



S: ask(?Dept) ⇒

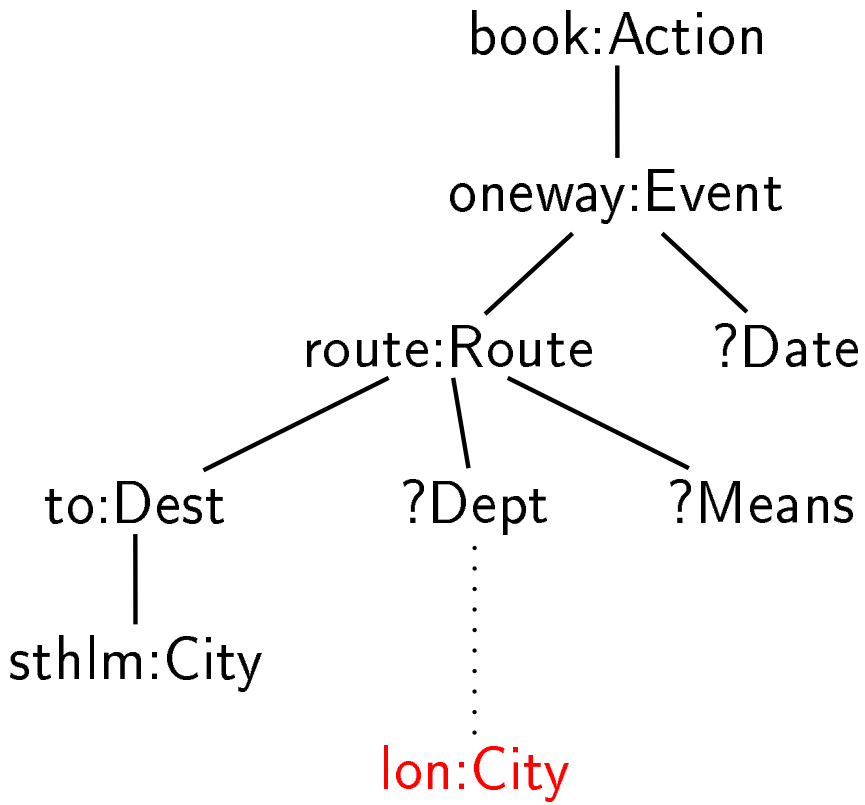
"From where are you leaving?"

Strategy 2: Bottom-up refinement



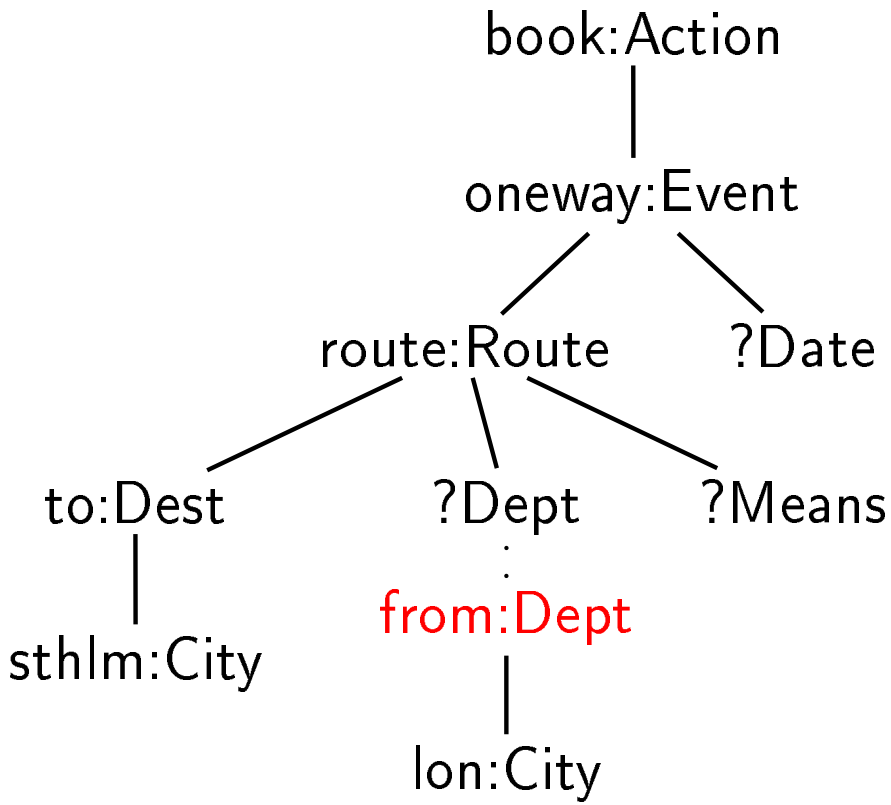
U: "London"⇒
answer(lon:City)

Strategy 2: Bottom-up refinement



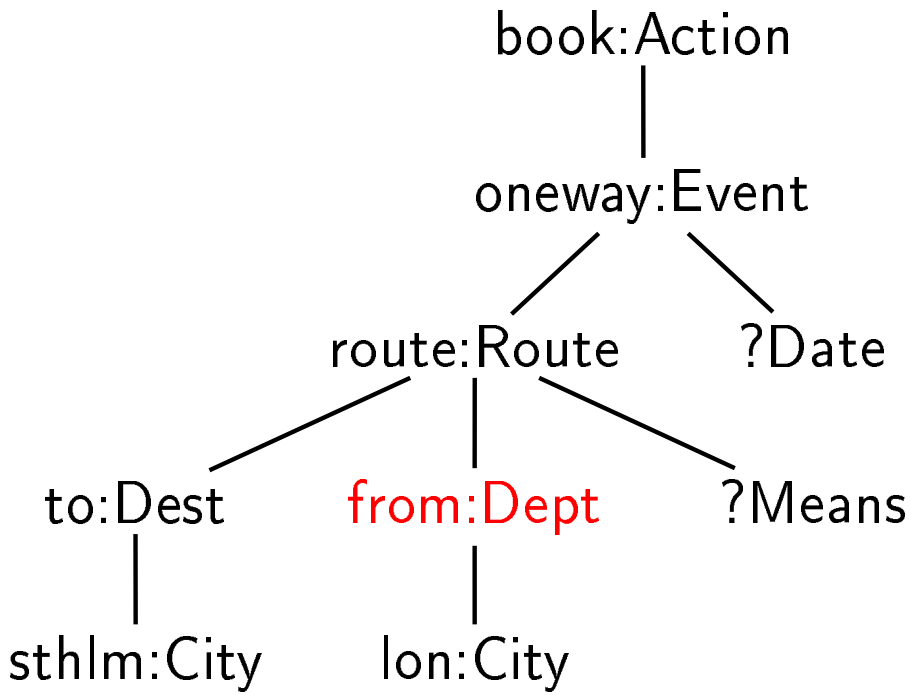
`focus-down`

Strategy 2: Bottom-up refinement



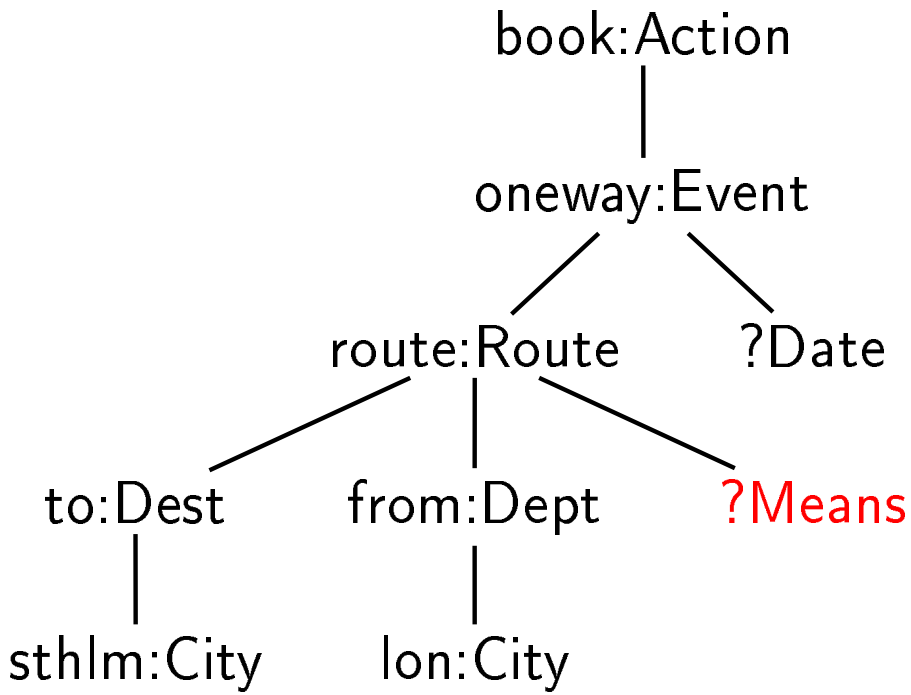
`refine-up`

Strategy 2: Bottom-up refinement



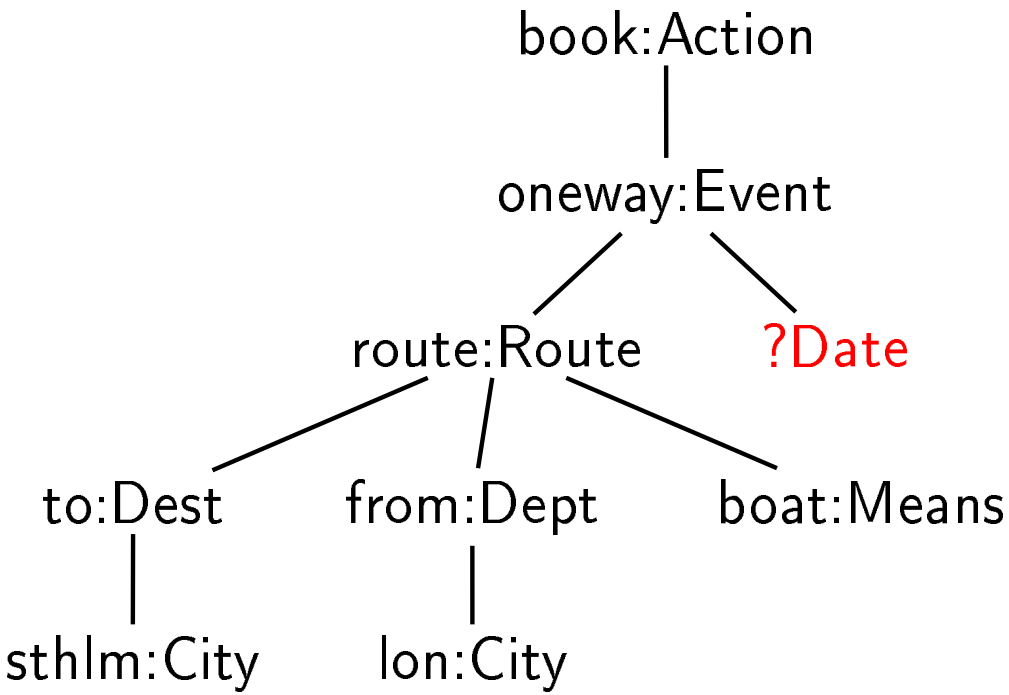
`refine-up`

Strategy 2: Bottom-up refinement



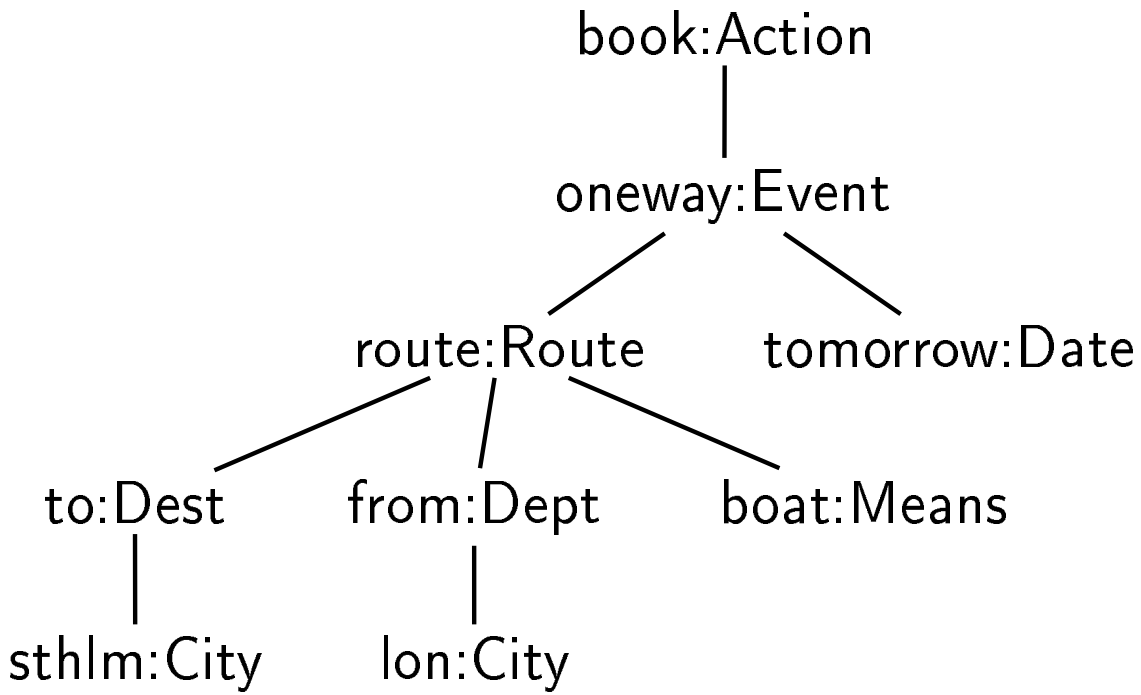
(etcetera. . .)

Strategy 2: Bottom-up refinement



(etcetera. . .)

Strategy 2: Bottom-up refinement

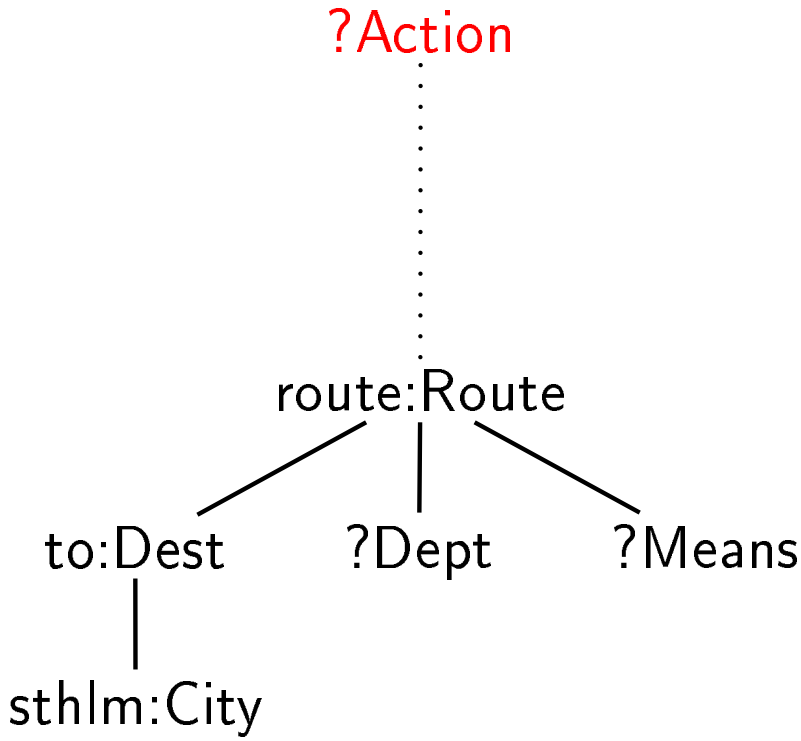


(etcetera. . .)

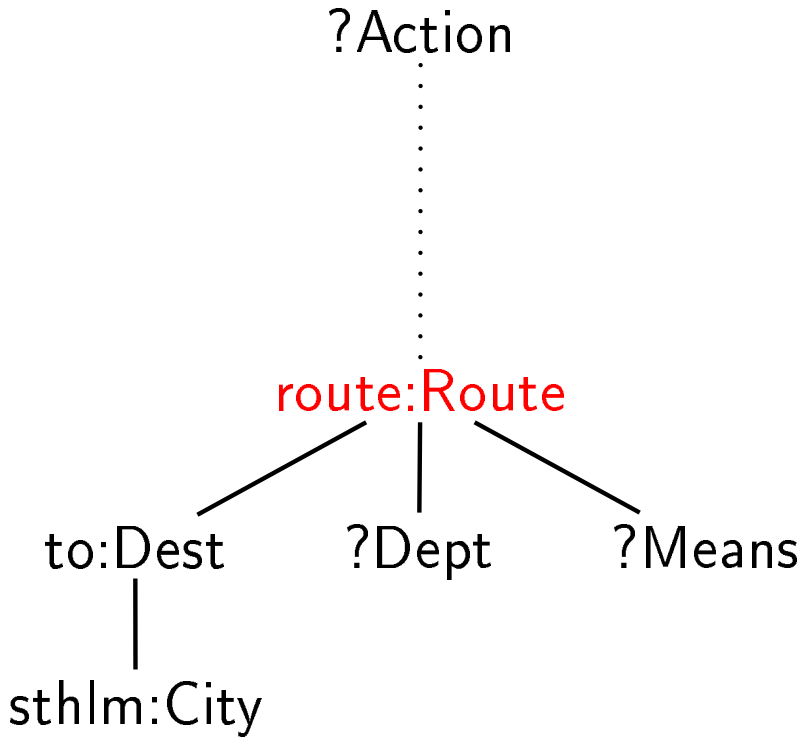
Strategy 3: "Bottom-down" refinement

?Action

Strategy 3: "Bottom-down" refinement

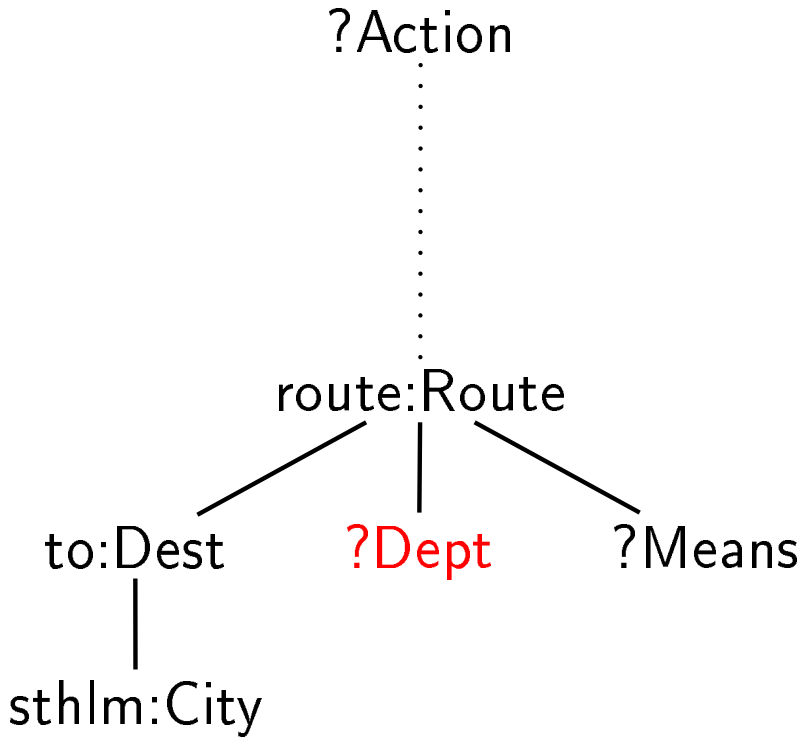


Strategy 3: "Bottom-down" refinement



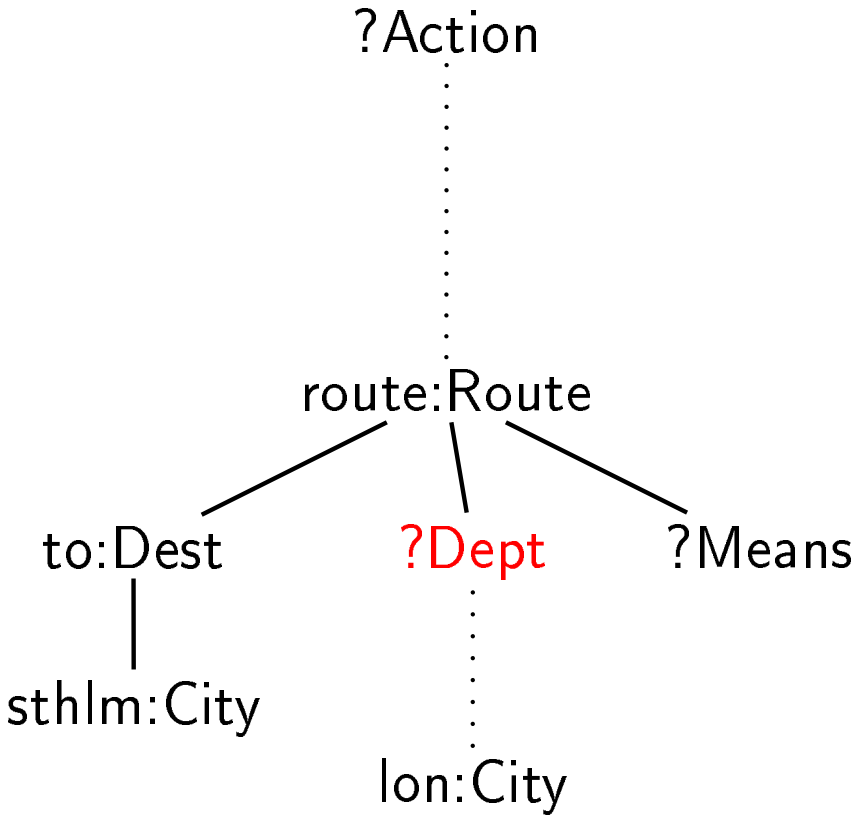
`focus-down`

Strategy 3: "Bottom-down" refinement

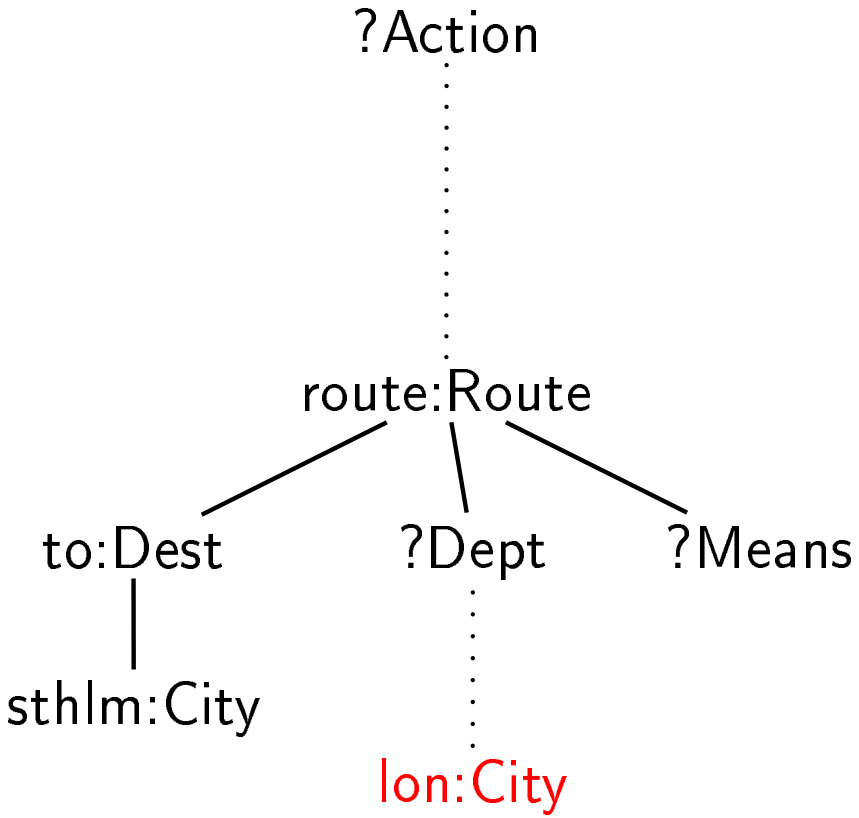


`select-next`

Strategy 3: "Bottom-down" refinement

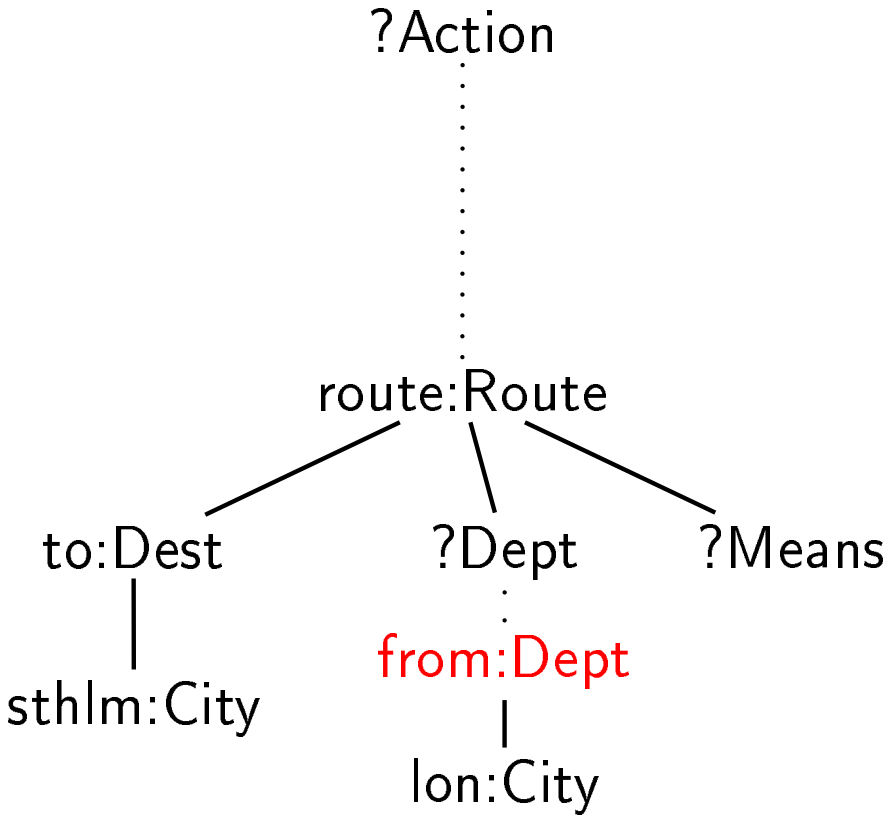


Strategy 3: "Bottom-down" refinement



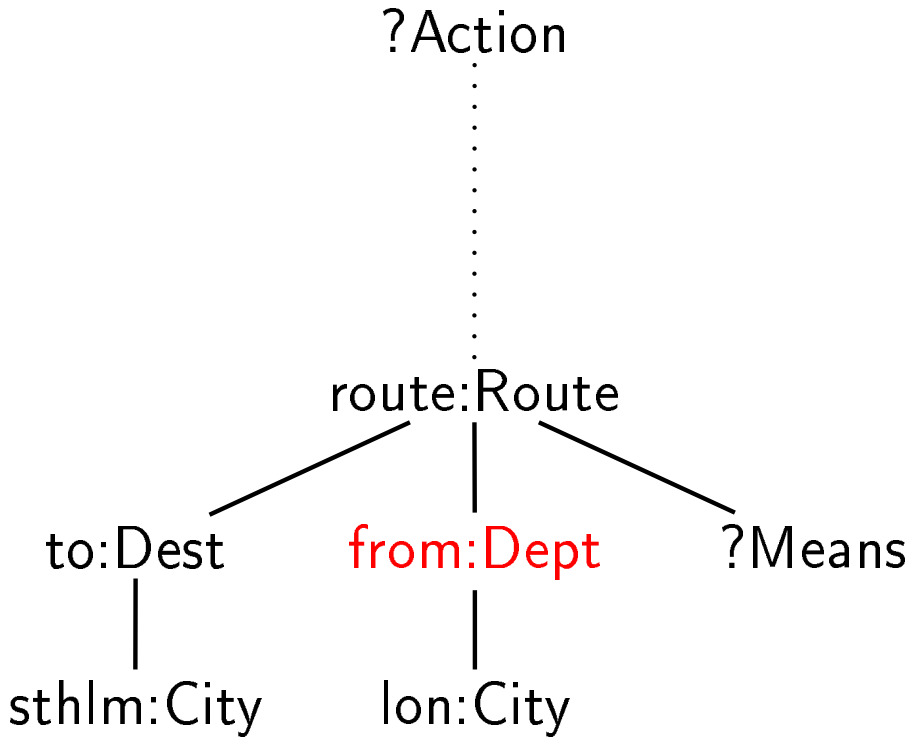
`focus-down`

Strategy 3: "Bottom-down" refinement



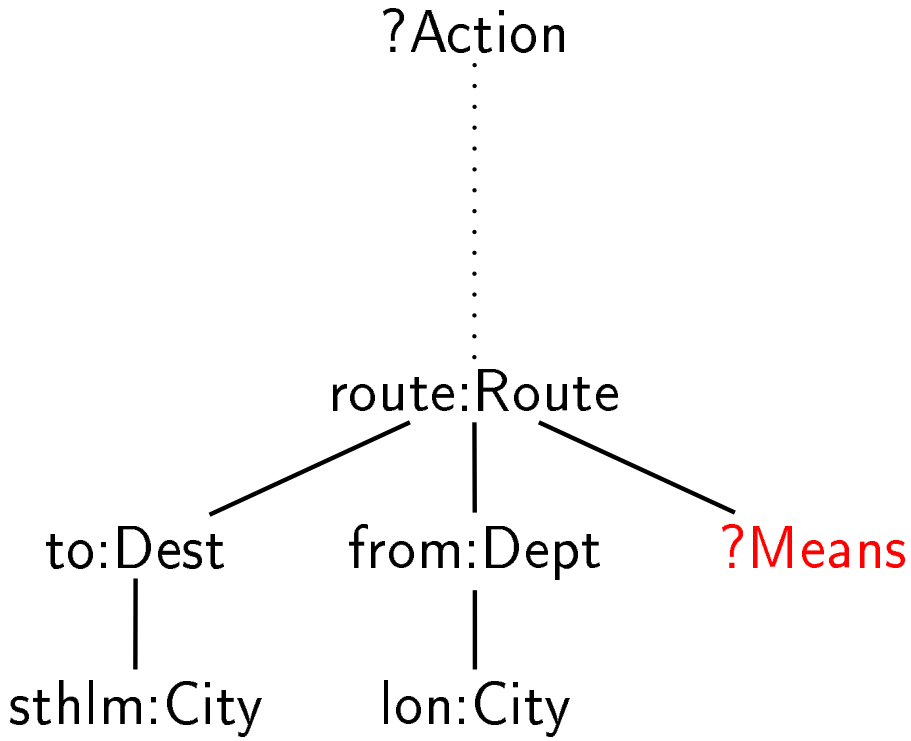
`refine-up`

Strategy 3: "Bottom-down" refinement

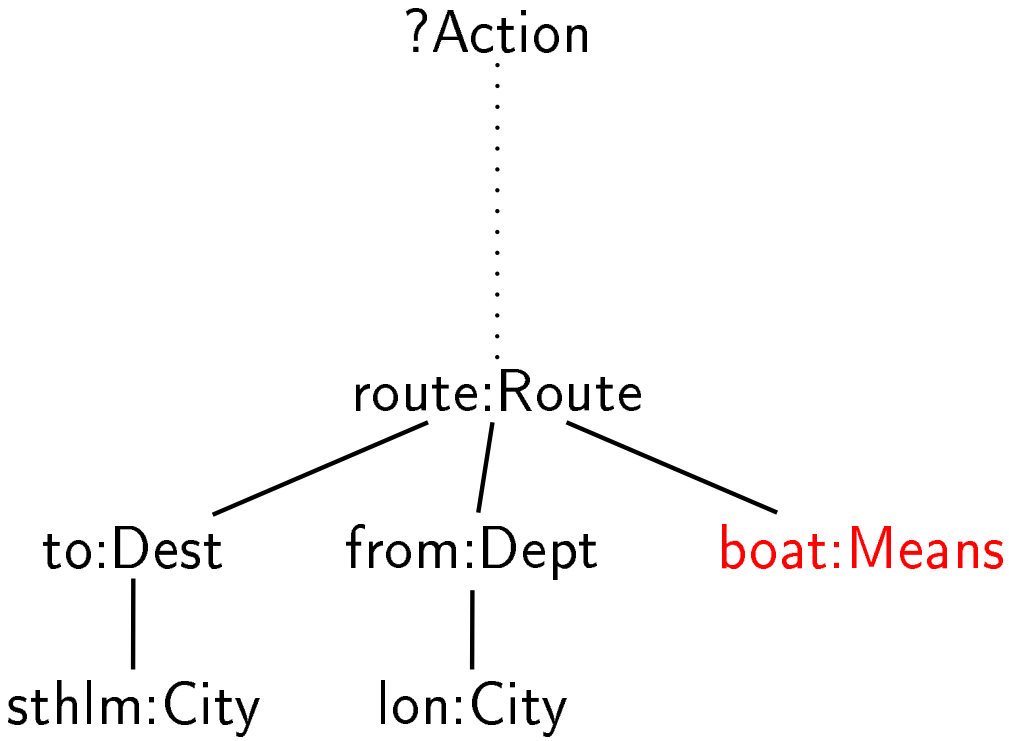


`refine-up`

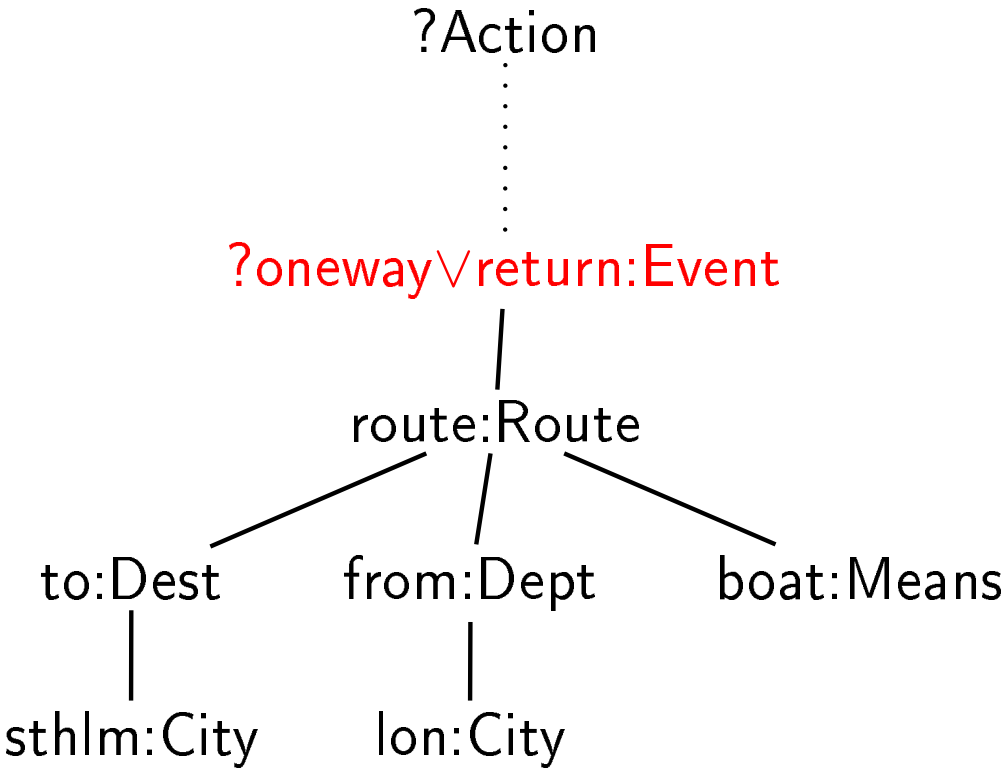
Strategy 3: "Bottom-down" refinement



Strategy 3: "Bottom-down" refinement

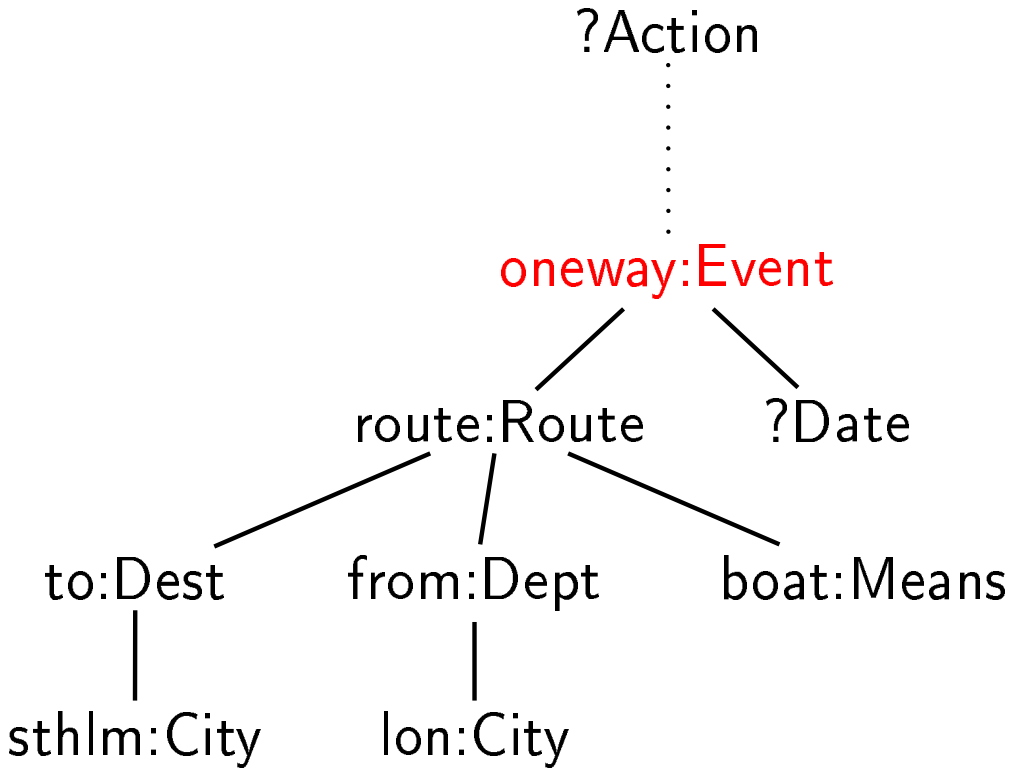


Strategy 3: "Bottom-down" refinement

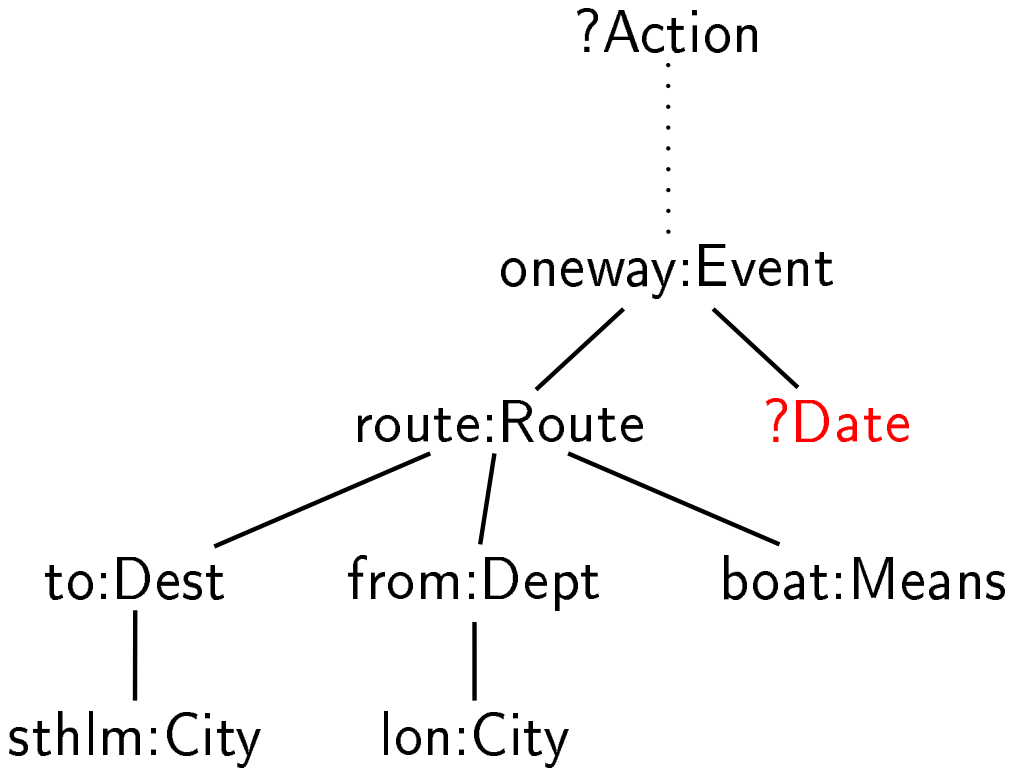


`refine-up`

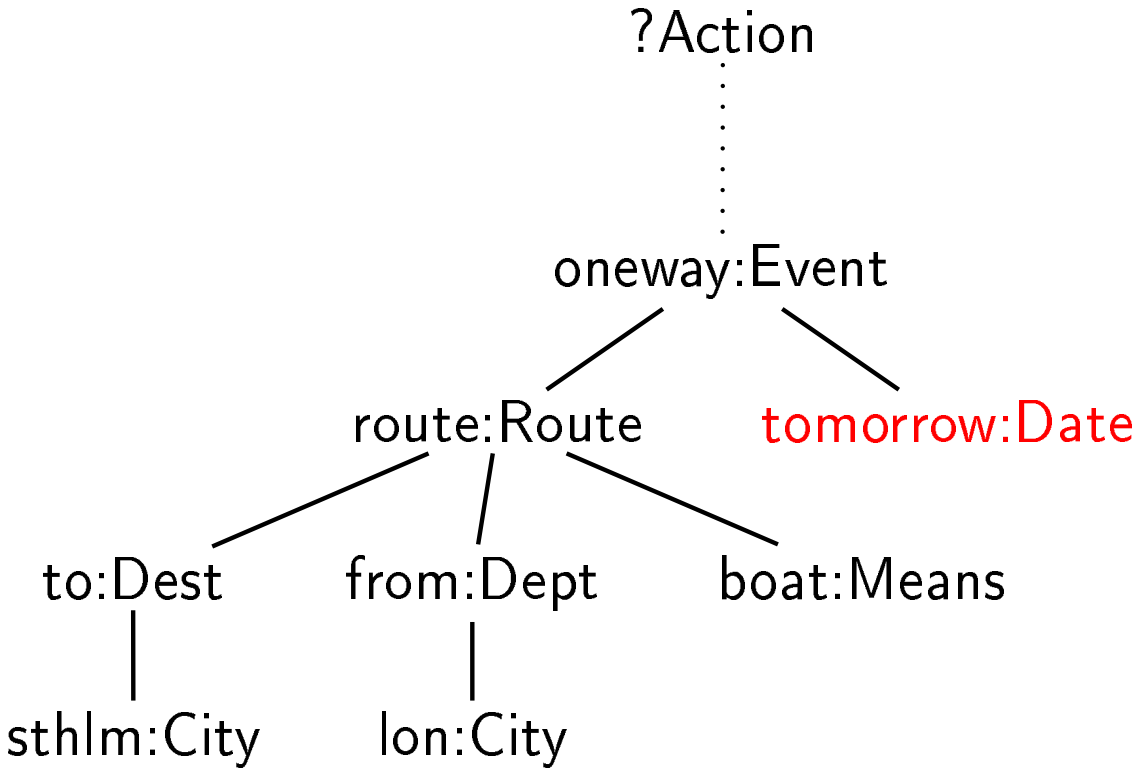
Strategy 3: "Bottom-down" refinement



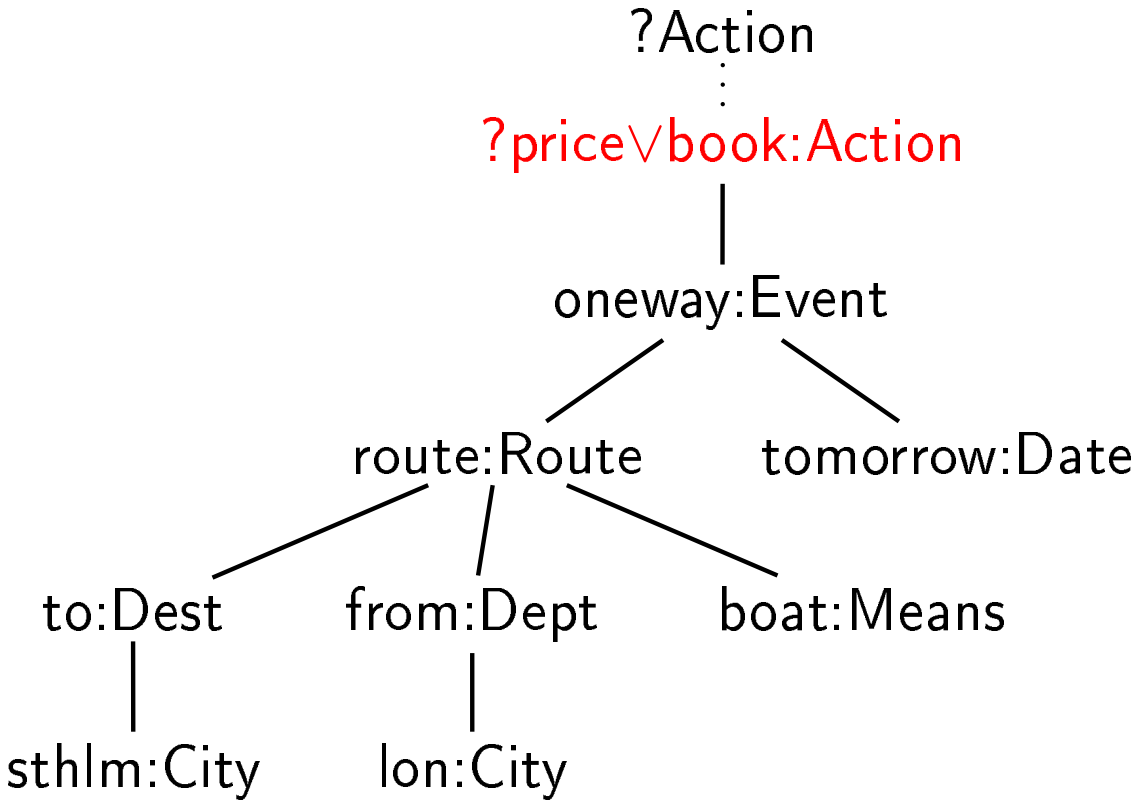
Strategy 3: "Bottom-down" refinement



Strategy 3: "Bottom-down" refinement

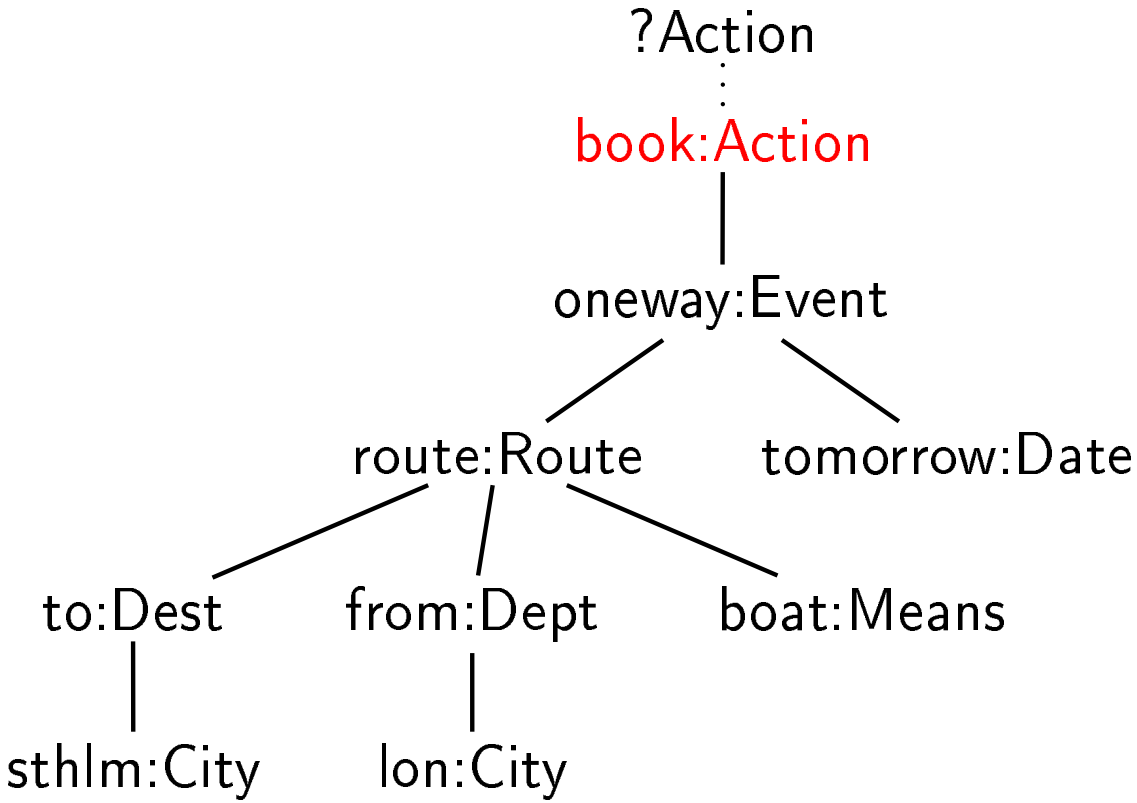


Strategy 3: "Bottom-down" refinement

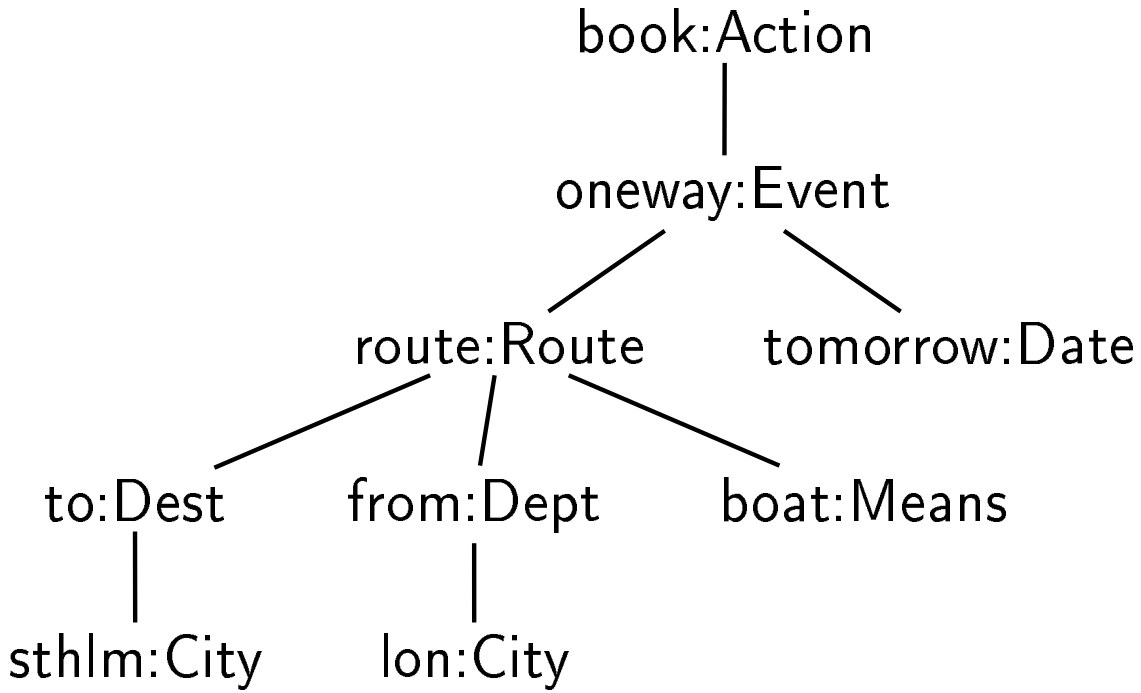


`refine-up`

Strategy 3: "Bottom-down" refinement



Strategy 3: "Bottom-down" refinement



`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"
$\Rightarrow$ 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"
$\Rightarrow$ 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"
$\Rightarrow$ 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"
$\Rightarrow$ 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"
$\Rightarrow$ 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"
$\Rightarrow$ when(conf-date(semdial,2009)) $\Rightarrow$ when(date(jun,24)) $\Rightarrow$
S: "SemDial 2009 starts 24th June."

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

Linked trees

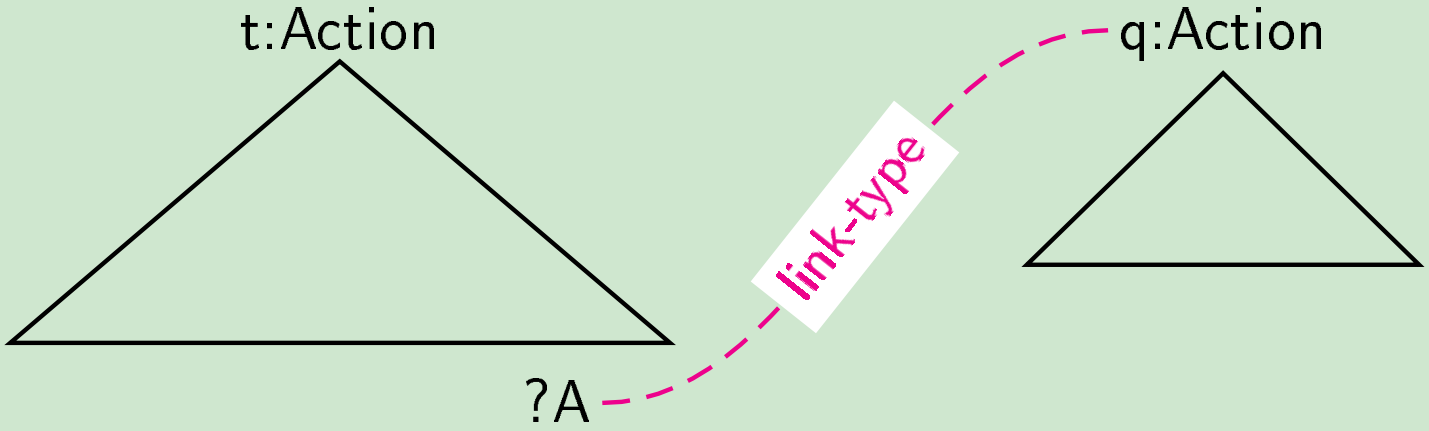
An underspecified node is (will be) a subtree of its parent node

- 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

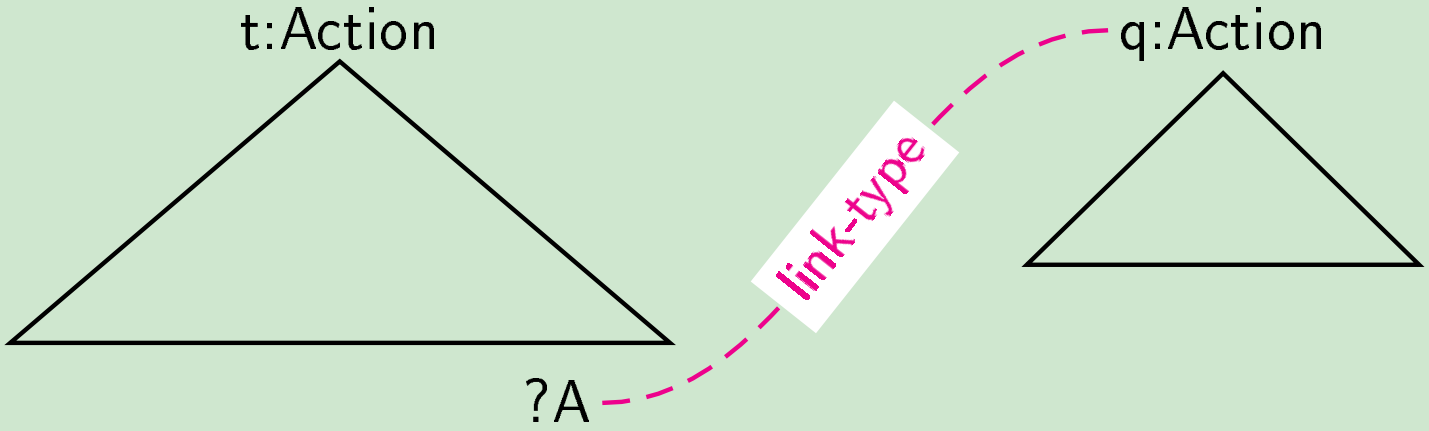
Linked trees



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

Linked trees



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

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

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

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

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

Example sub-dialogue

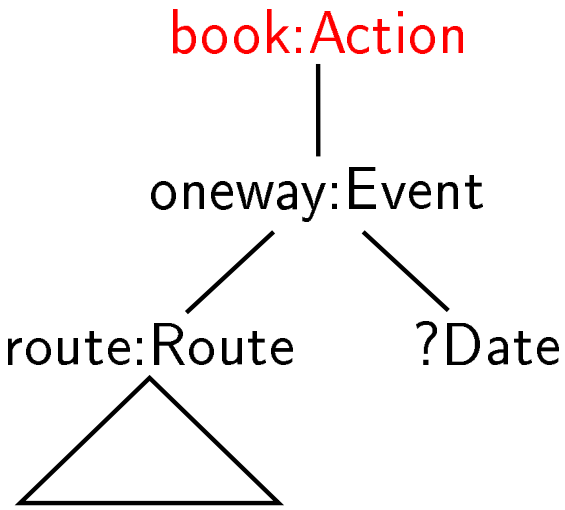
U: "I'd like to book a boat to Stockholm"
S: "When do you want to leave?"
| U: "when is SemDial?"
| S: "Which year do you mean?"
| U: "this year"
| S: "SemDial starts 24th June."
U: "ok, I'll leave the day before"
S: "I have booked a boat to Stockholm on 23rd June."

Sub-dialogues

?Action

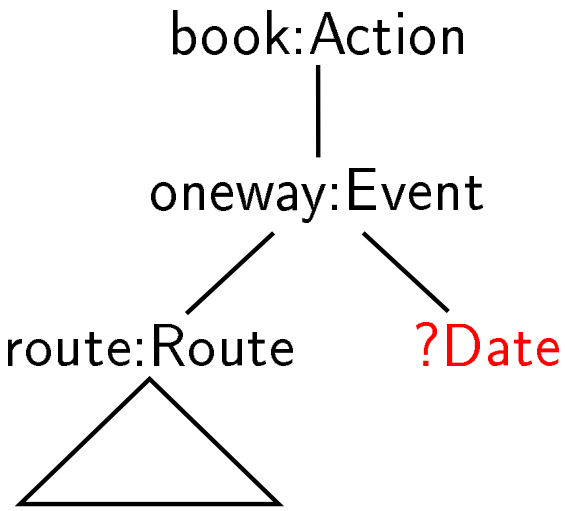
S: “What can I do for you?”

Sub-dialogues



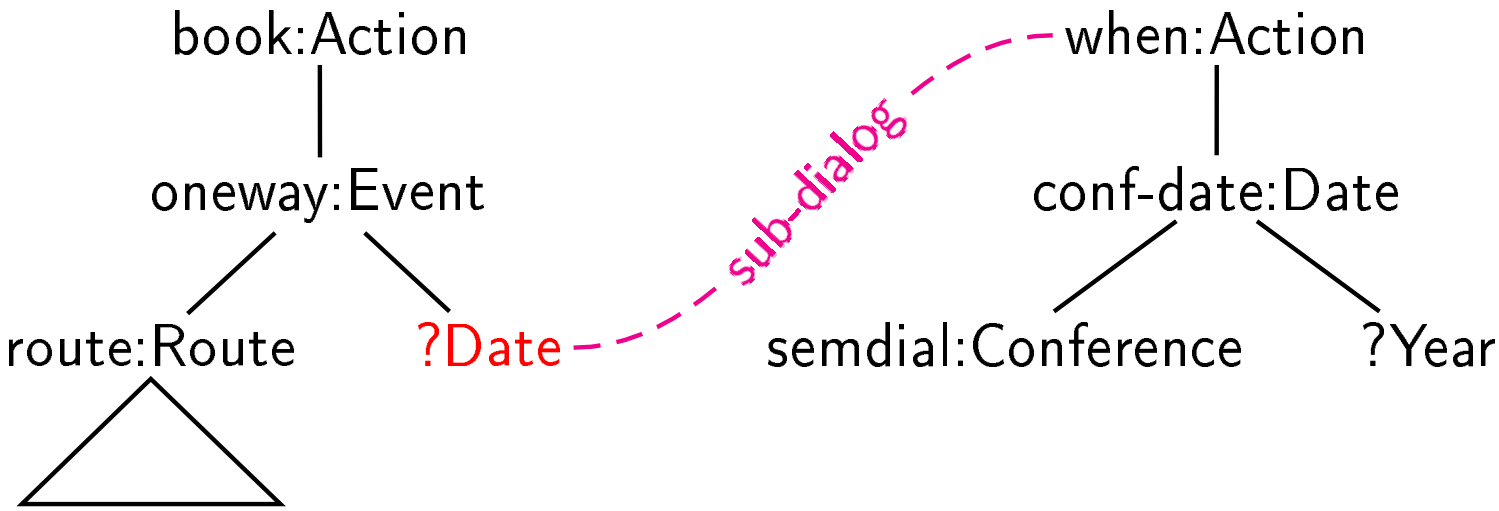
U: “I want to book a boat to Stockholm”

Sub-dialogues



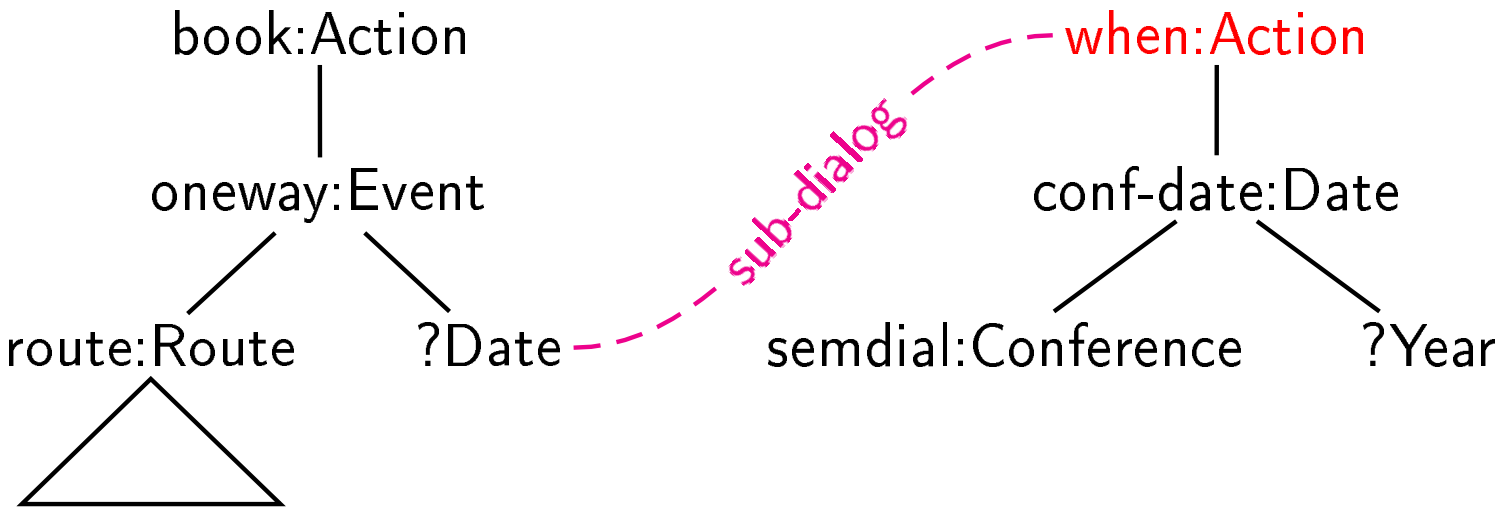
S: “When do you want to leave?”

Sub-dialogues



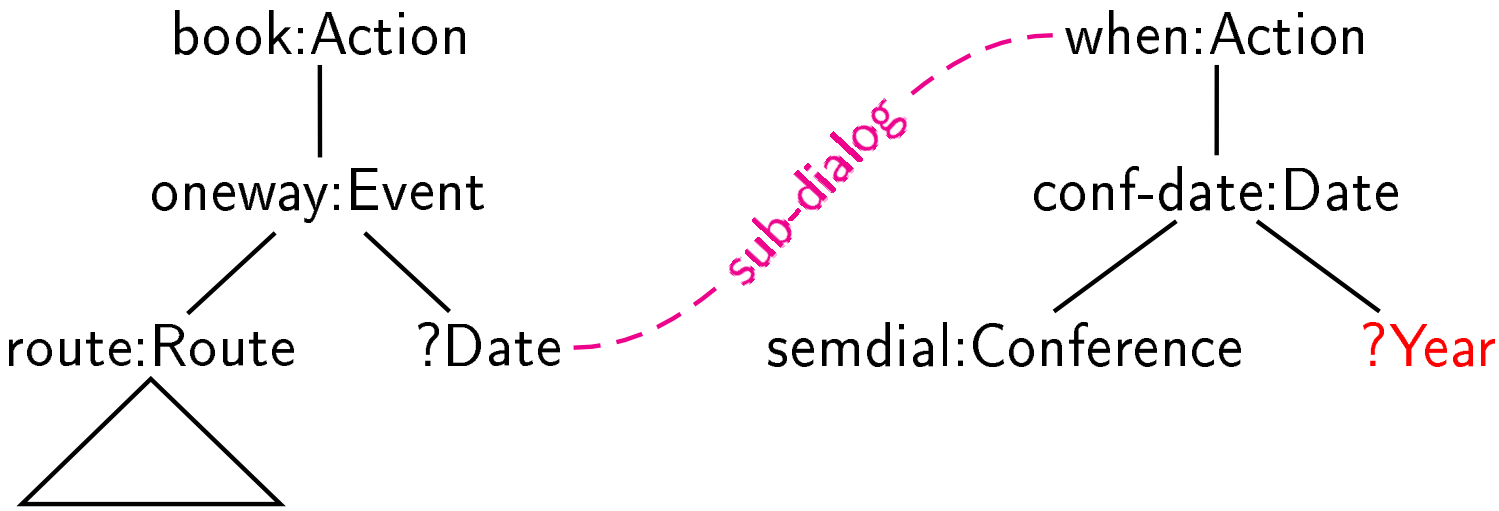
U: “when is SemDial?”

Sub-dialogues



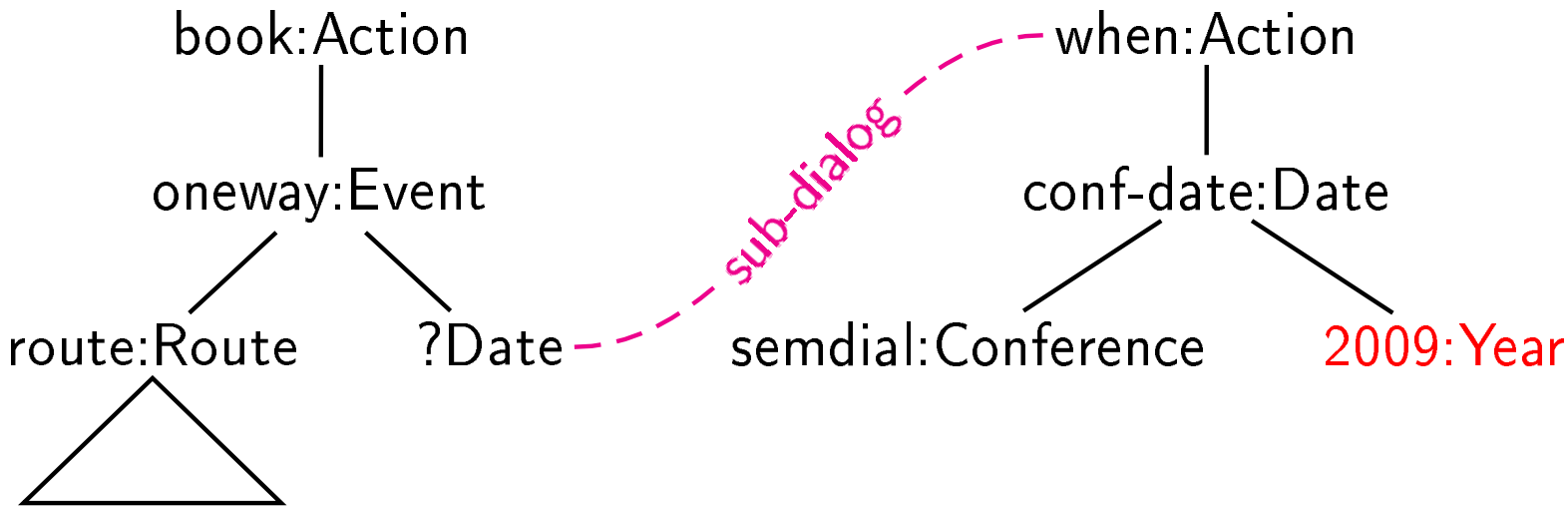
U: “when is SemDial?”

Sub-dialogues



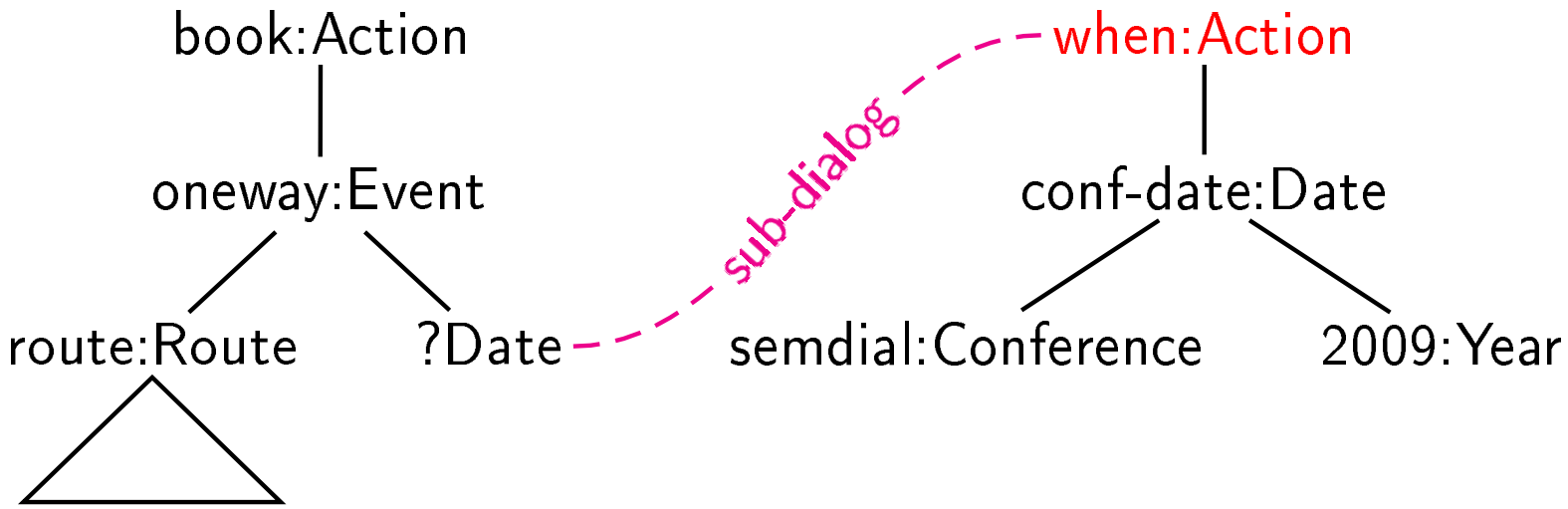
S: “Which year do you mean?”

Sub-dialogues



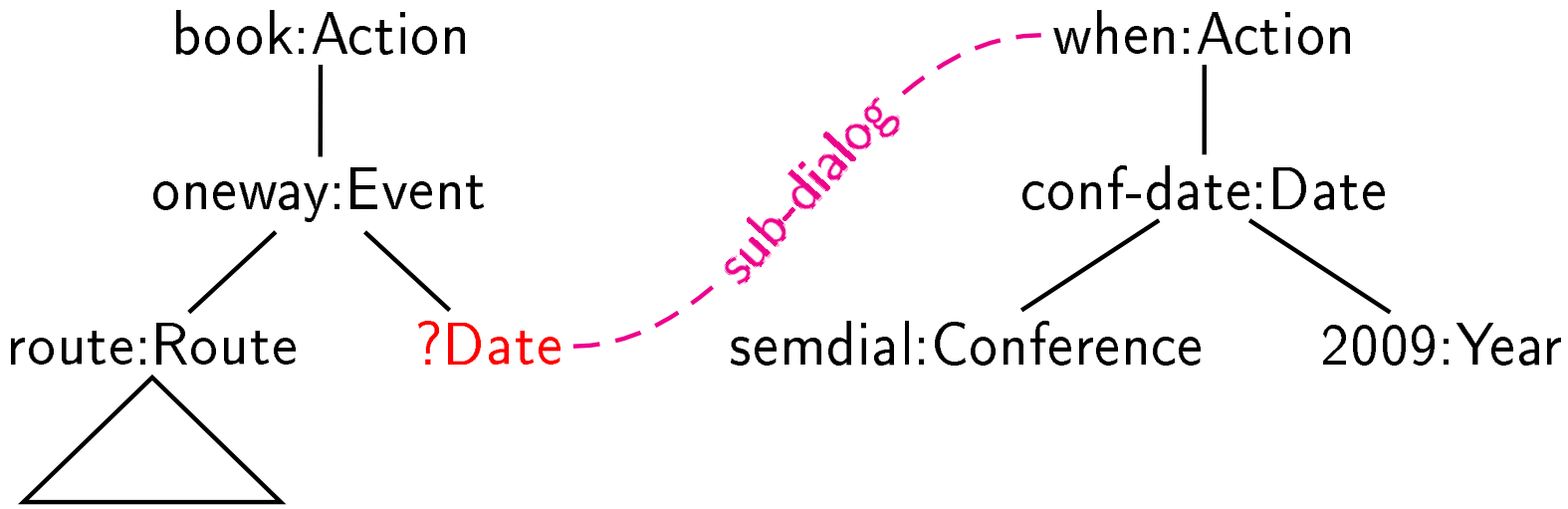
U: “this year”

Sub-dialogues



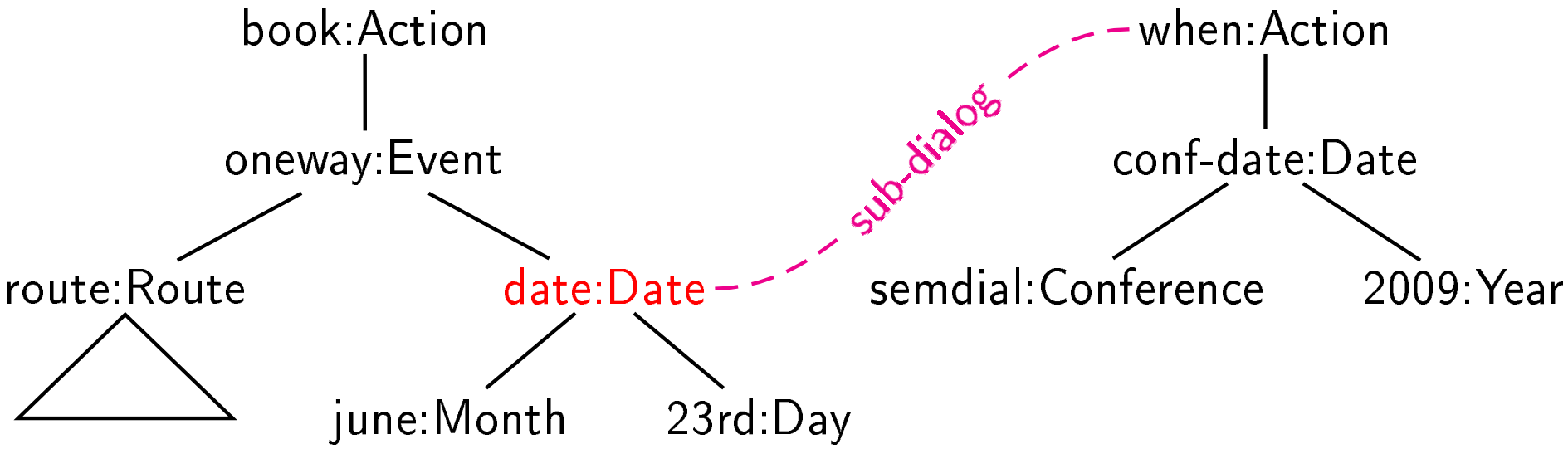
S: “SemDial starts 24th June.”

Sub-dialogues



S: “SemDial starts 24th June.”

Sub-dialogues



U: “ok, I’ll leave the day before”

Anaphoric expressions

?Action

S: “What can I do for you?”

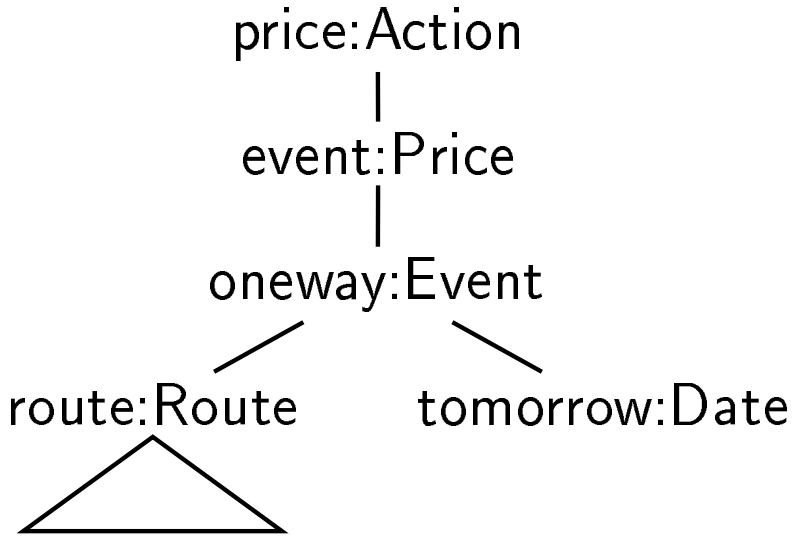
U: “how much is a flight to Stockholm tomorrow?”

S: “It costs €450.”

U: “ok, book it”

S: “I have booked a flight to Stockholm tomorrow.”

Anaphoric expressions



S: "What can I do for you?"

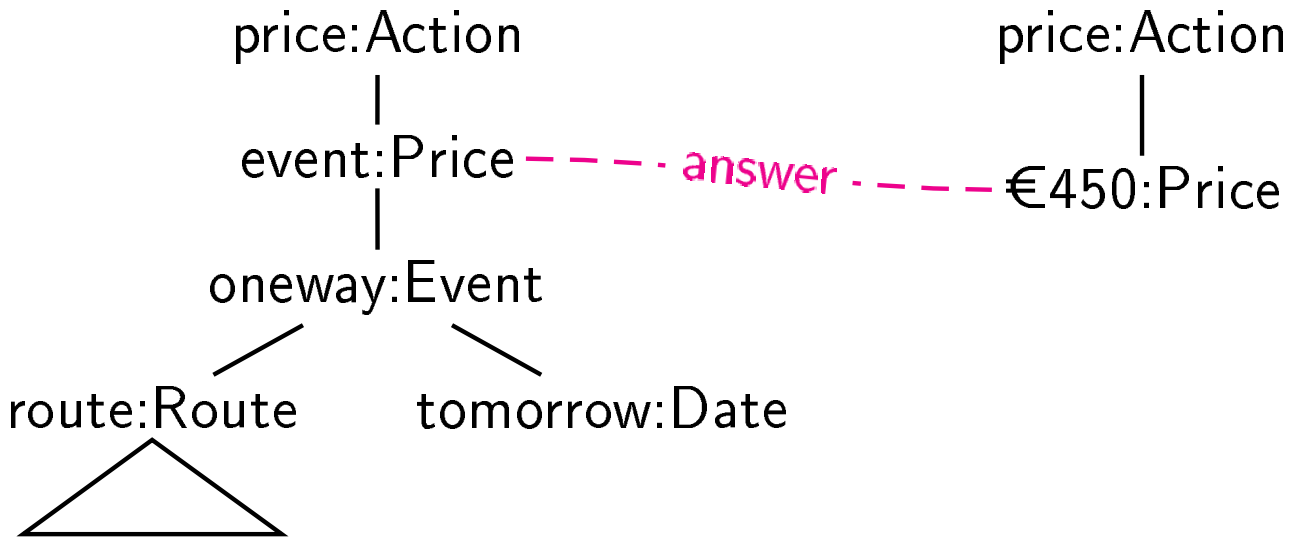
U: "how much is a flight to Stockholm tomorrow?"

S: "It costs €450."

U: "ok, book it"

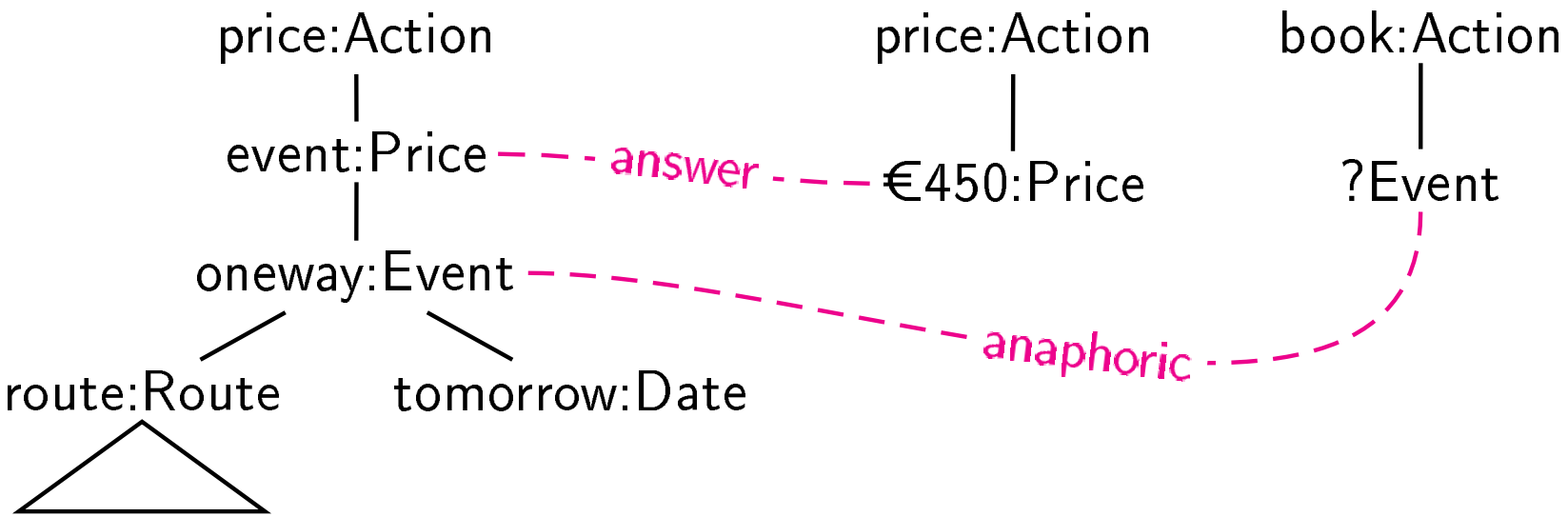
S: "I have booked a flight to Stockholm tomorrow."

Anaphoric expressions



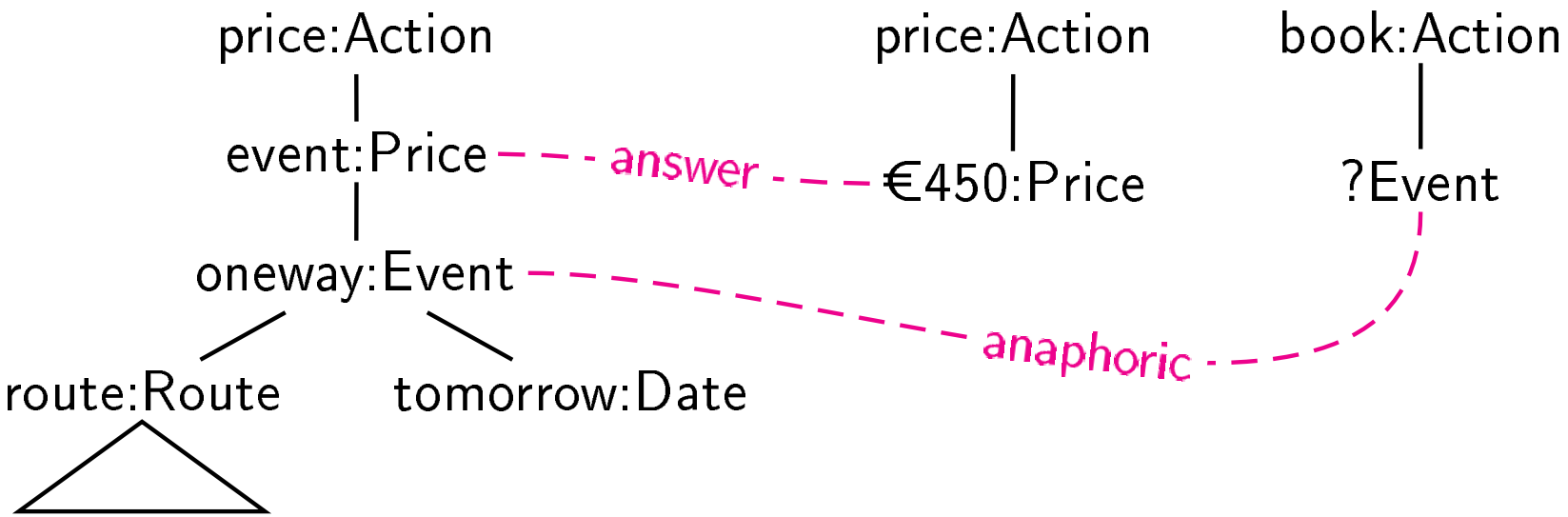
S: "What can I do for you?"
U: "how much is a flight to Stockholm tomorrow?"
S: "It costs €450."
U: "ok, book it"
S: "I have booked a flight to Stockholm tomorrow."

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."

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?”

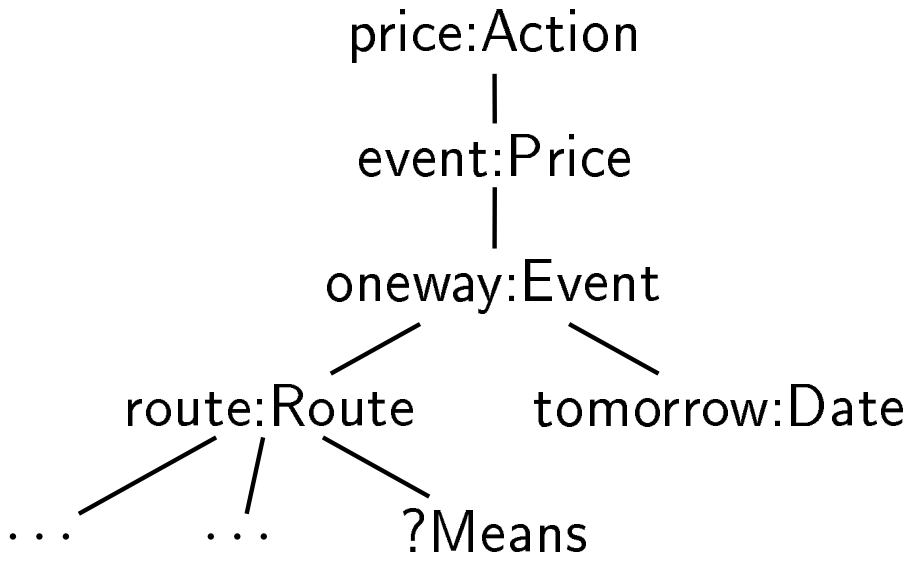
S: “By plane or by boat?”

U: “both”

S: “Plane costs €450 and boat €300.”

U: “ok, book the cheapest one”

Several alternatives



S: "What can I do for you?"

U: "how much is a trip to Stockholm tomorrow?"

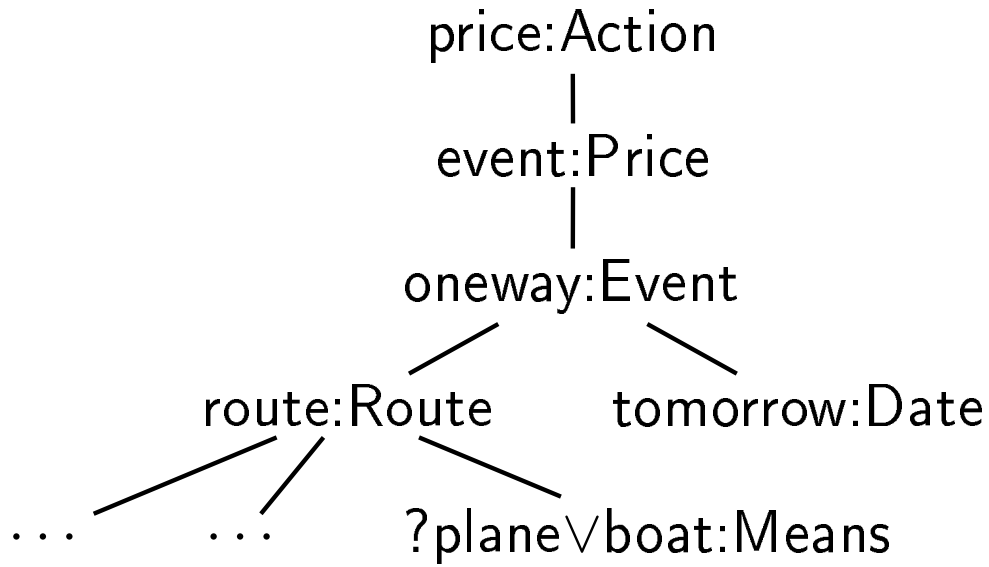
S: "By plane or by boat?"

U: "both"

S: "Plane costs €450 and boat €300."

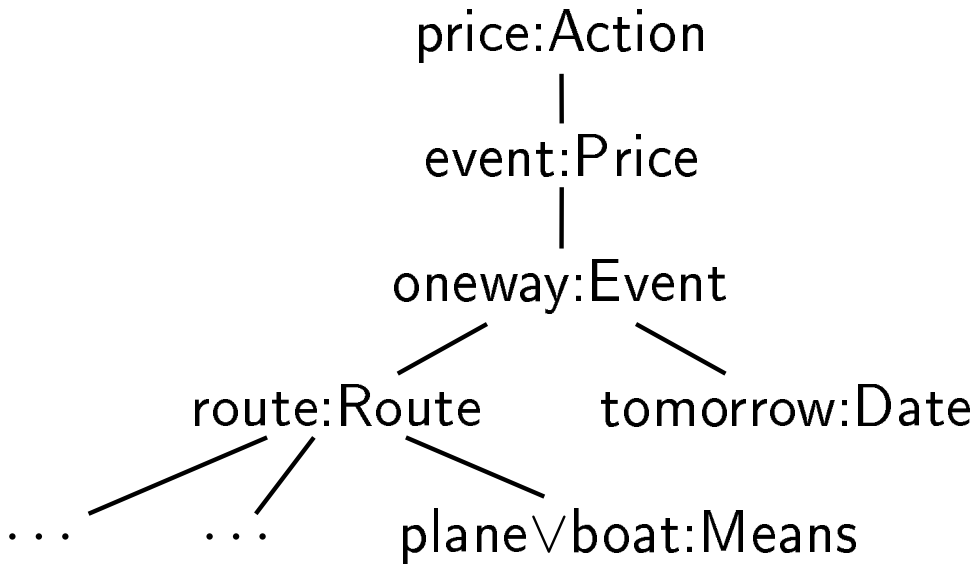
U: "ok, book the cheapest one"

Several alternatives



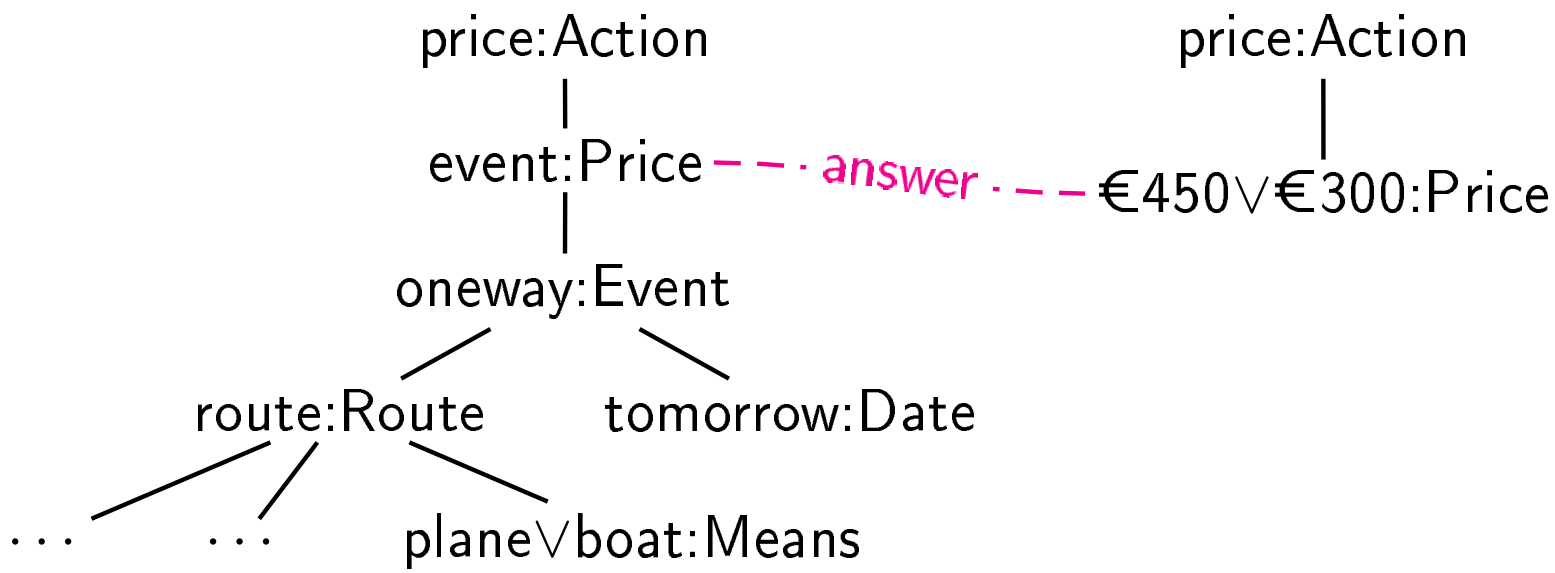
S: "What can I do for you?"
U: "how much is a trip to Stockholm tomorrow?"
S: "By plane or by boat?"
U: "both"
S: "Plane costs €450 and boat €300."
U: "ok, book the cheapest one"

Several alternatives



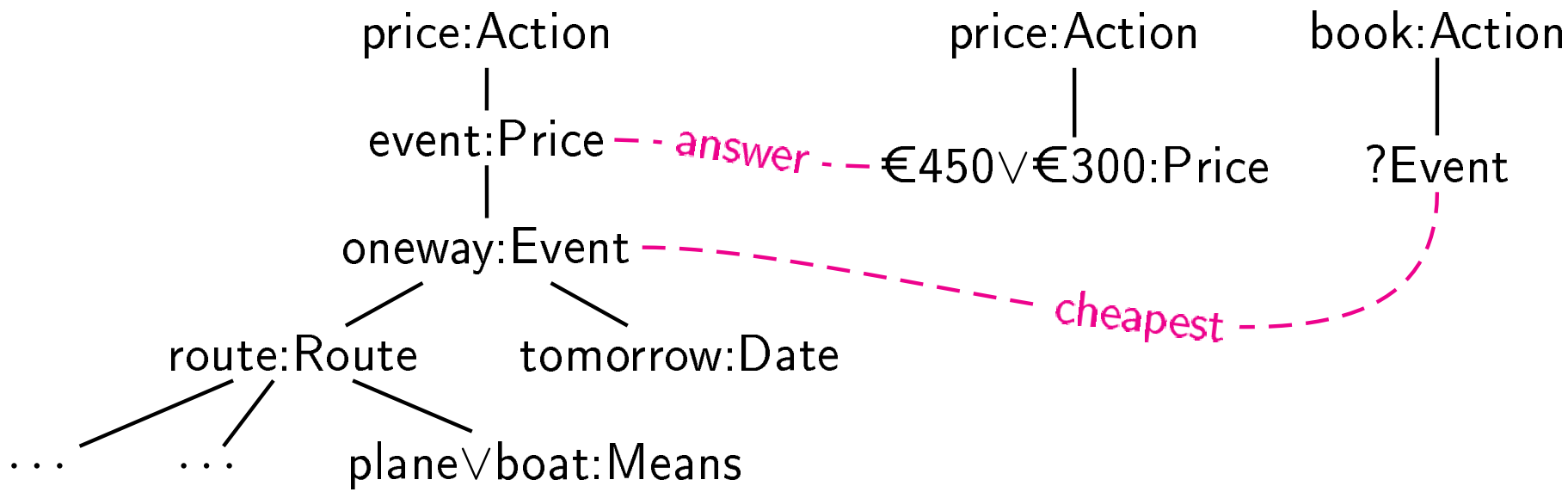
S: "What can I do for you?"
U: "how much is a trip to Stockholm tomorrow?"
S: "By plane or by boat?"
U: "both"
S: "Plane costs €450 and boat €300."
U: "ok, book the cheapest one"

Several alternatives



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



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"

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

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