# Dialogue Management as Interactive Tree Building

**Peter Ljunglöf**
Department of Philosophy, Linguistics and Theory of Science
University of Gothenburg, Sweden
peb@ling.gu.se

## Abstract

We introduce a new dialogue model and a formalism for limited-domain dialogue systems, which works by interactively building dialogue trees. The model borrows its fundamental ideas from type theoretical grammars and Dynamic Syntax. The resulting dialogue theory is a simple and light-weight formalism, which is still capable of advanced dialogue behaviour.

## 1 Background

### 1.1 Dialogue models beyond finite-state

A finite-state dialogue system employs dialogue states, connected by transitions, which represent where the dialogue participants are in the progress of the dialogue. This is a very low-level formalism, which only is feasible for very limited dialogue domains. The dialogues become system-driven – there is not much room for the user to take initiatives. A number of formalisms have been introduced to improve on this, that are based on richer, more powerful models of dialogue structure. Here are a few examples:

**Form-based dialogue systems** A form-based dialogue system divides different tasks into forms, similar to web forms, containing slots to be filled. VoiceXML (Oshry, 2007) is a W3C standard for writing form-based systems. This is a more powerful formalism than finite-state, but it too becomes difficult to manage when the complexity of the domain increases. One main reason for this is that form-based systems cannot handle underspecified or ambiguous information in a good way. Although the user is allowed to take some initiative within a form, it is the system that drives the dialogue on a higher level.

**Dialogue grammars** The idea of modelling dialogue in terms of a grammar is based on the idea of adjacency pairs, which describe facts such as that questions are generally followed by answers, proposals by acceptances, etc. Grammar-based dialogue systems were quite popular in the 1990's (Jönsson, 1997; Gustafson et al., 1998), but tend to be better at representing the surface linguistic expression involved in dialogue rather than the semantic content and its relation to context which is very often of central importance in determining the range of options available to a dialogue participant at a given point in a dialogue.

**Plan- and logic-based approaches** Plan-based dialogue systems construct or infer plans for fulfilling the goals of the dialogue participants. This is accomplished by using AI techniques such as planning and plan recognition. The related logic-based approach represents dialogue and dialogue context in some logical formalism. These systems tend to be computationally complex, since they perform general AI reasoning or theorem proving. For examples see, e.g., Allen et al. (2001), Sadek et al. (1997) and Smith et al. (1995).

**The information state update approach** To overcome the limitations of form-based systems, a theory of dialogue modelling was introduced, known as the information state update (ISU) approach (Larsson and Traum, 2000). It is based on a structured information state to keep track of dialogue context information. The information state is updated by update rules which are triggered by dialogue moves performed by the participants in the dialogue. The ISU approach enables a modular architecture which allows generic solutions for dialogue technology.

However, there are problems with ISU-based dialogue systems, such as the GoDiS dialogue manager (Larsson et al., 2000; Larsson, 2002). The update rules tend to get very complicated, making it difficult to foresee the side effects of changing a rule, or adding a new one.

**Dynamic Syntax** Dynamic Syntax is a combined syntactic and semantic grammatical theory (Kempson et al., 2001; Cann et al., 2005), which takes into account dialogue phenomena such as clarifications, reformulations, corrections, and acknowledgements.

The idea is that syntactic trees represent simple propositional sentences, and trees can be connected by links to form complex utterances. Dynamic Syntax can be seen as a kind of ISU for-



*Proceedings of the 13th Workshop on the Semantics and Pragmatics of Dialogue, June 24–26, 2009, Stockholm, Sweden.*

malism; the trees are built incrementally word-by-word, where an incomplete tree corresponds to an incomplete utterance. Words whose function is not determined yet (e.g., whether a noun in initial position should act as a subject or an object), are added as unfixed nodes below the current tree. Further on, when the interpreter has read some more words and its function has been determined, an unfixed node becomes a fixed part of the tree.

Since the minimal linguistic units in Dynamic Syntax are words, it is in practise only used for analysing single sentences or short dialogue exchanges. For full-size dialogues, the input resolution is too fine-grained.

**Dialogue as proof editing** Ranta and Cooper (2004) describe how a dialogue system can be implemented in a syntactical proof editor based on type theory, originally developed for editing mathematical proofs. Metavariables in the proof term represents questions that needs to be answered by the user so that the system can calculate a final answer. This is very similar to the Prolog-style proof-searching dialogue of Smith et al. (1995), but with a foundation in type-theory. However, Ranta and Cooper only support information-seeking dialogues, and the backbone is a fairly simple form-based dialogue system. Furthermore, there is no account for underspecified answers, anaphoric expressions, or ambiguous answers.

Our proposed dialogue model can be seen as a development of the approaches of Ranta and Cooper, and Smith et al., using ideas from Dynamic Syntax and ISU to make the system more flexible.

### 1.2 The Logic of Finite Trees

Dynamic Syntax is based on the underlying Logic of Finite Trees (Blackburn and Meyer-Viol, 1994), a logical theory which makes it possible to interactively build a tree in a logically well-founded manner. We will use two concepts from this logic; unfixed nodes and linked trees:

**Unfixed nodes** An unfixed node is a subtree which we know should be attached somewhere below a given node, but we do not yet know exactly where. Figure 1 contains three unfixed nodes: the A node dominates the B node, while the C node dominates both the D and E nodes. This means that in the final tree, C will contain both D and E as descendants. Note that this doesn't say anything about the order between D and E, it can even be the case that one of them will dominate the other in the final tree.

In Dynamic Syntax, unfixed nodes are used when the syntactic function of a phrase is unknown. E.g., a noun in initial position can function as a subject or an object, depending on the context. In the framework we introduce, unfixed nodes will be used for representing underspecified and/or ambiguous information.

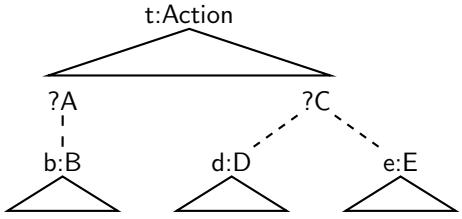


Figure 1: Unfixed nodes in a tree.

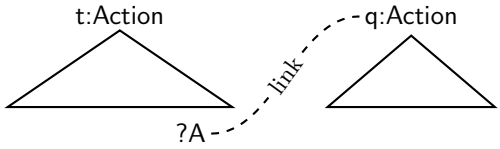


Figure 2: Two linked trees.

**Linked trees** Any node in a tree can have links to other trees. A link between two trees does not say that one of them dominates the other, it is merely a link between tree nodes. We assume that all links are labelled, as in figure 2.

In Dynamic Syntax, linked trees are used for, e.g., relative clauses, prepositional phrases, definites, anaphoric expressions and such things, whereas we will used them for question answering, sub-dialogues, and anaphoric references.

### 1.3 Type-theoretical grammar

Type theory is based on the Curry-Howard correspondence – "formulae-as-types" – where types correspond to propositions and terms correspond to proofs. To prove a proposition $T$ we have to build a syntactic term $t : T$. An interactive proof editor builds a term interactively, where a metavariable $?T$ is used for an unknown subterm of type $T$. As Ranta and Cooper (2004) noted, $?T$ can be seen as a question posed by the system, "Which term of type $T$ do you want to put here?".

Grammatical Framework (GF) is a grammar formalism based on type theory (Ranta, 2004). The main feature is the separation of abstract and concrete syntax, which makes it very suitable for writing multilingual grammars. The abstract part of a GF grammar defines a set of abstract syntactic structures, called abstract terms or trees; and the concrete part defines a relation between abstract structures and concrete structures. This separation of abstract and concrete syntax is crucial for our treatment of dialogue systems. A rich module system also facilitates grammar writing as an engineering task, by reusing common grammars.

**Abstract syntax** The abstract theory of GF is a version of Martin-Löf's (1984) type theory.

A grammar consists of declarations of categories and functions. In figure 3 is an example grammar, which we will use as our example domain. With the declaration route(?Dest,?Dept):Route, we mean that route($x$,$y$):Route whenever $x$:Dest and $y$:Dept.[1] Furthermore, the grammar can contain function definitions, which we will use for calculating dialogue actions.

**Concrete syntax** GF has a *linearization* perspective to grammar writing, where the relation between abstract and concrete is viewed as a compositional mapping from abstract to concrete structures, called linearization terms. Linearizations are written as terms in a typed functional programming language, which is limited to ensure decidability in generation and in parsing.

It is possible to define several concrete syntaxes for one particular abstract syntax. Multilingual grammars can be used as a model for interlingua translation, but also to simplify localization of language technology applications such as dialogue systems.

Since this article is about the abstract dialogue model, and not about parsing and generation, we will not give any examples of linearization definitions. Examples of GF linearizations for dialogue systems can be found in Bringert et al. (2005) and Ljunglöf and Larsson (2008).

## 2 A tree-based ISU dialogue model

Our proposed dialogue model is an ISU model in that it operates on an information state which is modified by update rules. However, the information state is not a flat representation of plans and questions under discussion, as in, e.g., the GoDiS dialogue manager (Larsson, 2002). Instead the information state is represented by an incomplete tree in a similar way as is done in Dynamic Syntax, where incomplete nodes in the tree correspond to information that remains to be given.

In contrast with Dynamic Syntax, the minimal linguistic units are user and system utterances, and not single words. This makes it possible to model practical full-length dialogues, instead of being restricted to single sentences or short dialogue exchanges.

The goal of the dialogue is to build a tree, and when this tree is completed, it represents a task which the user wants the system to perform in some way. This is similar to a form in a form-based system, and a dialogue plan in an ISU system such as GoDiS, but it has a hierarchical, tree-based, structure instead of being flat. Using a tree-based information state means among other things that we can treat tasks, issues, plans and forms in exactly the same way as we treat the ontology of individuals, properties and predicates, thus simplifying the underlying logic. The use of trees, here, is related to the use of dialogue trees in, for example, work by Lemon et al. (2002), and are also found in dialogue grammar approaches. However, the kinds of trees we are using and the relationships we express between them are more complex. The main difference is that we used unfixed nodes and linked trees, which adds flexibility to the dialogue which has been a problem for grammar-based systems.

### 2.1 Specifying the dialogue domain

Simliar to our previous work (Bringert et al., 2005; Ljunglöf and Larsson, 2008), we use a type theoretical grammar to specify all aspects of the dialogue domain – tasks, issues, plans and forms, as well as individuals, properties and predicates. We can then make use of type checking for constraining the dialogue trees, and type checking can also be used when interpreting user utterances and when providing the user with suggestions of what to say next.

Another advantage with using a type theoretical grammar formalism, is that it is a multiple-level formalism, which can be used to specify the concrete user and system utterances which correspond to the tree structures that are used in the information state. Furthermore, Grammatical Framework is a multiple-language formalism, meaning that we can specify the dialogue domain as the language-independent part of the grammar, which is shared with all different language-dependent parts. Finally, type-checking is used to ensure that the different grammar instances are sound with respect to the dialogue domain.

To specify a dialogue domain, we have to declare all possible ways of forming trees. As already mentioned, an example travel agency domain is shown in figure 3, where with the declaration route(?Dest,?Dept):Route, we mean that route($x$,$y$):Route whenever $x$:Dest and $y$:Dept. In this domain the user can book an event, ask for the price of an event, and ask when something happens. The possible events are oneway and round trips, hotel stays and conferences.

An example dialogue tree according to the specification is book(oneway(route(to(lon),from(gbg)), tomorrow)), which is also shown in figure 4. The concrete syntax defines translations between trees and utterances, and one possible translation of the example tree is "Book a oneway trip tomorrow from Gothenburg to London". We assume that the concrete syntax also defines translations of shorter phrases, such as "Book a oneway trip tomorrow", book(oneway(route(?,?),tomorrow)), and "A trip to London", route(to(lon),?).

[1] Note that we use a different GF grammar syntax than is common, to emphasise the similarities with tree-building and incomplete trees.

book(?Event), how-much(?Price), when(?Date) : Action
event-price(?Event) : Price
oneway(?Route,?Date), return(?Route,?Date,?Date),
hotel(?City,?Date), conf(?Conference) : Event
today, tomorrow, date(?Month,?Day),
conf-date(?Conference,?Year) : Date
route(?Dest,?Date) : Route
to(?City) : Dest
from(?City) : Dept
2008, 2009, ... : Year
jan, feb, ... : Month
1st, 2nd, ... : Day
lon, gbg, ... : City
acl, diaholmia, ... : Conference
€450, €600, ... : Price

Figure 3: Example domain

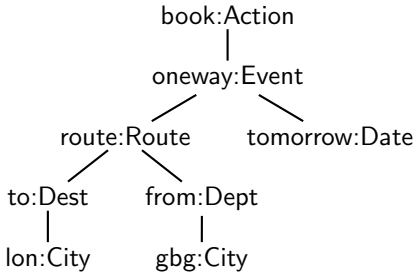


Figure 4: A completed dialogue tree

### 2.2 Dialogue as interactive tree building

The dialogue system tries to build a complete tree by successive refinement. In the middle of the dialogue, we represent the uninstantiated parts of the tree with metavariables. In this framework the metavariables are typed (which we write as $?T$) – when a new variable is created we can always infer its type from the types of the constants in the tree.

During the dialogue there can be several active dialogue trees, but there is always one current tree, and in that tree there is one single node which has focus. The focus node is highlighted like ⋆this⋆ in our example trees. The dialogue tree and its focus are operated with commands, such as changing focus to another node, inserting subtrees below the focus node, refining metavariables, etc.

The general idea is that the system moves the focus to a metavariable node, and asks the user to refine that node. User utterances are translated to incomplete subtrees, which the system tries to incorporate. If the user utterance is of the same type as the focused metavariable, the tree can be extended directly. Otherwise the system tries to add the utterance as an unfixed node below the focus, if possible, or tries to change focus to another metavariable which has the correct type.

### 2.3 System-driven dialogue

The dialogue starts with an incomplete tree, with only one metavariable stating the final type of the tree. In the example domain this final type is Action, so the initial tree is ?Action. The system then asks the question "What do you want to do?".

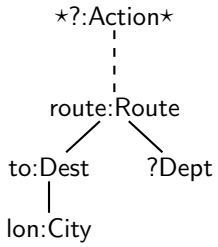


Figure 5: An incomplete dialogue tree.

**Direct answer** If the user gives a direct answer "Book a oneway trip tomorrow", book(oneway(route(?,?),tomorrow)), it is inserted at the focus node.

**Being helpful** If the user asks for help, or remains silent for a while, the system tries to refine the focus node itself. According to the specification, there are three possible actions, so the node is refined to the disjunction ?(book∨how-much∨when):Action. This is interpreted as an alternative question, "Do you want to book an event, ask for the price, or know a date?".

### 2.4 Handling underspecified information

The user is not required to always give direct answers to the system's questions; (s)he can, e.g., give underspecified answers. For incorporating underspecified information we use unfixed tree nodes, which is similar to how Dynamic Syntax does it: If the syntactic function of a word is unknown, its corresponding node in the tree becomes underspecified; e.g., a noun in initial position can be used as subject or object, and we cannot know which until more words are incorporated. This also corresponds to clarifications in GoDiS, within a single plan or between different plans.

If the user answers "A trip to London" (route(to(lon),?)), it is not a direct answer to the question ?Action. But since the answer type Route is dominated by Action, the system adds the answer tree as an unfixed node to the focus node. This is shown in figure 5.

Now, there are (at least) three different refinement strategies, depending on how the system searches for new metavariable nodes. We call these strategies top-down, bottom-up and "bottom-down".

**Top-down refinement** After this the system tries to refine the focus using the dominated tree as a constraint. Of the three possible Action refinements, only book and price can dominate a Route, so the focus node is refined to ?(book∨how-much):Action. This is shown in figure 6.

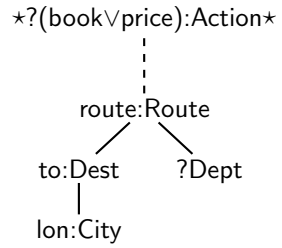


Figure 6: Top-down refinement of figure 5

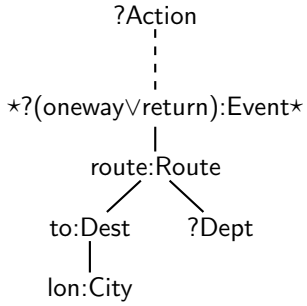


Figure 7: Bottom-up refinement of figure 5

The same thing will happen later in the dialogue, when the system wants to know which event to book (assuming that was what the user intended). When trying to refine the ?Event metavariable, only two of the four possible events can dominate a Route, so the node is refined to ?(oneway∨return):Event.

**Bottom-up refinement** Top-down refinement tries to connect a metavariable node with its unfixed tree by successively refining the dominating node. An alternative strategy is to instead connect the nodes by refining the dominated node. We call this strategy bottom-up refinement. The idea is that whenever the focus node has an unfixed child, the focus is moved to that child and refinement is done upwards. This means that when bottom-up refining the user answer "A trip to London", the system asks whether the user meant a oneway or a return trip, as shown in figure 7.

Furthermore, there are two different flavours of this dialogue strategy – non-aligned and aligned refinement. The most straight-forward variant of bottom-up refinement is to collect the possible immediate parents of the dominated node in the alternative question. Now, assume that the user only answered "London" to the initial ?Action question. There are three possible parents to a City – to:Dest, from:Dept and hotel:Event – which means that the system will have to ask the question ?(to:Dest∨from:Dept∨hotel:Event), which could be translated as "Do you mean to London, from London or a hotel in London?".

If it feels awkward to ask alternative questions about terms of different types, we can use *aligned* bottom-up refinement instead. In this variant, we collect the closest possible parents *all having the same type*. Since both Dest and Dept are dominated by Event, the question we get in our example is ?(oneway∨return∨hotel):Event.

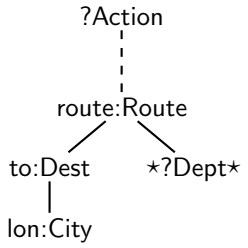


Figure 8: Bottom-down refinement of figure 5

**"Bottom-down" refinement** A third possible dialogue strategy, which we call "bottom-down" refinement, is to immediately dig into the tree that the user provided and try to complete that tree, before returning to the original top-level question. This means that after the system has attached the user answer "A trip to London" as an unfixed child of the ?Action node, focus is moved to the first metavariable in the given tree. The next question will therefore be "From where do you want to go?", as shown in figure 8. When the dominated tree is completed, the system can either proceed by top-down refining the dominating Action node, or by bottom-up refining the dominated Route node.

### 2.5 When the dialogue tree is complete

After hopefully a successful interaction, the dialogue tree is completed and represents an action that the user wants the system to execute. We model this with functional definitions, mapping the trees into action descriptions that the system can execute. In our example domain we can distinguish two kinds of actions:

**Answering a question** Some of the trees in the dialogue domain represent questions asked by the user. In our example both how-much(?Price) and when(?Date) represent user questions. To answer the question the system needs to consult a database, which can be encoded as function definitions in the domain:[2]

```
def conf-date(acl, 2009) = date(aug, 2)
def conf-date(...) = ...
def event-price(oneway(route(gbg,lon),
                       tomorrow) = €450
def event-price(...) = ...
```

After the dialogue tree is completed, the system evalutes it into an answer which then can be told to the user. The evaluated tree is added as a new

[2]Note that we are allowed to generate these function definitions automatically from the database in advance, or even on demand.

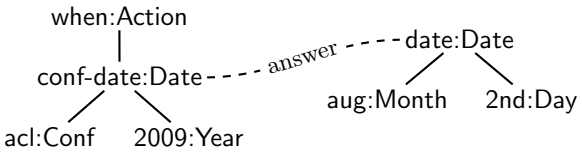


Figure 9: Answering a question

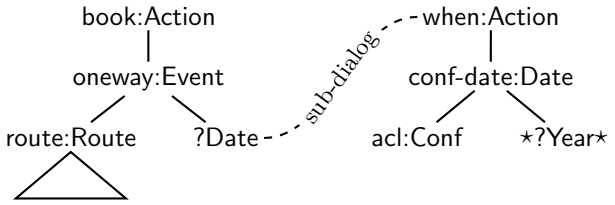


Figure 10: Engaging in a sub-dialogue

tree with a link to the original completed tree. The exact phrasing of the answer is specified in the concrete syntax.

For example, suppose the user asks "When is ACL?", which is recognised as when(conf-date(acl,?)). The system moves the focus to the metavariable, asking "Which year do you mean?", to which the user answers "2009". Now we get the final tree when(conf-date(acl,2009)), which is reduced by the function definitions to when(date(aug,2)). The system uses the concrete syntax to translate this into the answer "ACL starts 2nd August". The final dialogue state contains the question tree and the answer tree, connected by a link, as shown in figure 9.

**Performing an action** If the final tree is a booking of an event, the system needs a way of communicating with the outside world. Our simple solution is to attach side-effects to the type-theoretical function definitions. The problem with this is that the side-effects will reside outside the logical framework, which means that we cannot rely on type-checking or logical reasoning, for outside world interaction.[3] For example, the result of applying the term book(*event*) could be success(booking(*id*)) if the booking succeeds with booking number *id*, or failure(*reason*) otherwise, where *reason* is some explanation of why the booking failed.

### 2.6 Sub-dialogues

One important property of a flexible dialogue system is the possibility to engage in a sub-dialogue, and when that dialogue is finished, to return to the original dialogue. Here is an example from our example domain:

**U:** "I'd like to book a trip from Gothenburg to Singapore"

**S:** "When do you want to leave?"

**U:** "When is ACL?"

**S:** "Which year do you mean?"

**U:** "This year"

**S:** "ACL starts 2nd August."

**U:** "Okay, I'll leave the day before"

**S:** "I have booked a trip to Singapore on 1st August."

We treat sub-dialogues in a similar way as Dynamic Syntax treats relative clauses: Since a relative clause is a statement of its own, it is interpreted as a complete tree, which is connected with a link to the referring node in the main tree. And in the same way a sub-dialogue is a tree of its own, with a link to the node where the sub-dialogue was introduced in the main dialogue tree. Figure 10 shows the dialogue state when the system asks the question "Which year do you mean?". After the user has answered this question, the system will create an answer tree just as in figure 9, after which focus is returned to the remaining ?Date metavariable in the main tree.

To be able to recognise the final user utterance correctly, the system needs to be able to handle anaphoric references ("the day before") by following sub-dialogue links.

### 2.7 Anaphoric expressions

In Dynamic Syntax, linked trees are also used for anaphoric references. A pronoun, or a definite noun phrase, suggests that there is a matching reference somewhere in the context. We treat anaphora in a similar way, by linking the anaphoric node to a previous dialogue tree.

**U:** "How much is a flight from Gothenburg to London tomorrow?"

**S:** "It costs €450."

**U:** "Okay, book it"

**S:** "I have booked a flight to London tomorrow."

After the first two utterances we have two dialogue trees – one representing the user question which is linked to the answer tree. Since these trees are completed, the next user utterance creates a new dialogue tree. The pronoun "it" is translated to a special constant it:Event, which triggers a lookup in the dialogue context for a matching subtree of the same type. The system finds a matching tree and creates an anaphoric link, as is shown in figure 11. When executing the booking, the system can use the event referred to by the link.

[3]Note that this is a problem that is shared with other existing dialogue frameworks. A more logically sound solution would be to, e.g., use a *state monad* for modeling the outside world (Wadler, 1997).

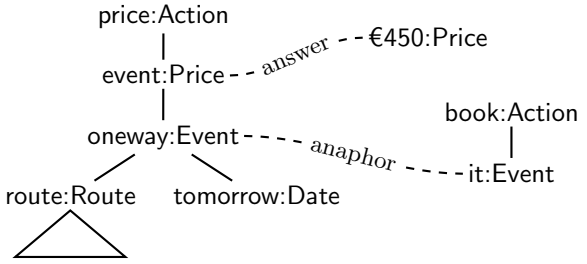


Figure 11: Handling anaphoric expressions

## 3 Discussion

We have introduced a dialogue model which works by interactively building dialogue trees. The model is a development of the "dialogue as proof editing" idea by Ranta and Cooper (2004), enhanced with a treatment of underspecification and references inspired from Dynamic Syntax.

**Specifying user and system utterances** By using a type-theoretical grammar formalism such as Grammatical Framework, we can specify all user and system utterances together with the abstract specification. The type checker can be used for catching errors in the specification, and the modular features of GF can be used for reusing grammar resources.

**Questions under discussion** The metavariables in the active dialogue tree correspond to the QUD (Questions Under Discussion), introduced by Ginzburg (1996). The QUD is a partial ordered set, and its topmost element corresponds to the focus node in our framework. The partial ordering of the QUD is implicit in our model, in the domain specification together with the order in which the algorithm searches the tree for metavariables.

**Unified treatment of plans and items** If we look at the domain specification in figure 3, we see that there is no conceptual difference between the plans (e.g., asking for the price or specifying an event) and the individual entities (e.g., the cities, dates or conferences). In fact, a declaration of a function such as route(?Dest,?Dept), both defines a dialogue plan (asking for a destination and a departure city) and the resulting individual (a specific route between two cities). This is in contrast with traditional form-based systems such as VoiceXML, and ISU systems such as GoDiS, where plans and individuals are separate concepts.

**Unfixed tree nodes** We use unfixed tree nodes for representing underspecified information, much in the same way as Dynamic Syntax does. The underlying Logic of Finite Trees automatically uses these nodes as constraints on the dominating nodes. We have described several different strategies for handling underspecified information (top-down, bottom-up and "bottom-down"), which then correspond to different strategies for accommodation in existing ISU dialogue models.

**Links between trees** Similar to Dynamic Syntax, we use links between trees for question answering, sub-dialogues and anaphoric expressions. The GoDiS dialogue manager handles sub-dialogues by having a stack of active plans, but it has no treatment of anaphoric references.

**Function definitions for system replies** Type-theoretical function definitions represent system replies to user questions and requests. This corresponds to external database calls in other formalisms, the difference being that we are using a well-founded logical theory, hopefully making it easier to reason logically about the properties of the system.

### 3.1 Future work

There are some issues that we have not addressed in this article, which are necessary for a working dialogue system.

**Feedback** We have not described how feedback should be treated. The reason is that since feedback cannot be defined in terms of the dialogue tree, its treatment is an orthogonal matter. Our aim is to incorporate the Interactive Communications Management (ICM) of Larsson (2002) into the system. This means that we need to add feedback information to the dialogue state, in parallel with the linked dialogue trees.

**Corrections** We have not described how the user can correct erroneous information in the dialogue tree. To be able to do this we need commands for deleting and changing tree nodes, as well as a functioning feedback system for clarifying the corrections.

**Implementation** We have rudimentary implementations in the programming languages Haskell and Python, but they need much more work to be useable as dialogue systems.

## Acknowledgments

I am grateful to Staffan Larsson and four anonymous referees for constructive comments and suggestions.

## References

James Allen, Donna K. Byron, Myroslava Dzikovska, George Ferguson, Lucian Galescu, and Amanda Stent. 2001. Toward conversational human-computer interaction. *AI Magazine*, 22(4):27–37.

Patrick Blackburn and Wilfried Meyer-Viol. 1994. Linguistics, logic, and finite trees. CWI Report CS-R9412, Centrum voor Wiskunde en Informatica, Amsterdam, Netherlands.

Björn Bringert, Robin Cooper, Peter Ljunglöf, and Aarne Ranta. 2005. Multimodal dialogue system grammars. In *Proc. Dialor'05, 9th Workshop on the Semantics and Pragmatics of Dialogue*, Nancy, France.

Ronnie Cann, Ruth Kempson, and Lutz Marten. 2005. *The Dynamics of Language*. Elsevier.

Jonathan Ginzburg. 1996. Dynamics and the semantics of dialogue. In J. Seligman, editor, *Language, Logic and Computation, volume 1*, CSLI Lecture Notes. CSLI Publications.

Joakim Gustafson, Patrik Elmberg, Rolf Carlson, and Arne Jönsson. 1998. An educational dialogue system with a user controllable dialogue manager. In *Proc. ICSLP'98, 5th International Conference on Spoken Language Processing*, Sydney, Australia.

Arne Jönsson. 1997. A model for habitable and efficient dialogue management for natural language interaction. *Natural Language Engineering*, 3(2-3):103–122.

Ruth Kempson, Wilfried Meyer-Viol, and Dov Gabbay. 2001. *Dynamic Syntax: The Flow of Language Understanding*. Blackwell.

Staffan Larsson and David Traum. 2000. Information state and dialogue management in the TRINDI Dialogue Move Engine Toolkit. *Natural Language Engineering*, 6(3–4):323–340.

Staffan Larsson, Peter Ljunglöf, Robin Cooper, Elisabet Engdahl, and Stina Ericsson. 2000. GoDiS – an accommodating dialogue system. In *Proc. ANLP–NAACL'00 Workshop on Conversational Systems*, Seattle, Washington.

Staffan Larsson. 2002. *Issue-based Dialogue Management*. Ph.D. thesis, Department of Linguistics, University of Gothenburg.

Oliver Lemon, Alexander Gruenstein, and Stanley Peters. 2002. Collaborative activities and multi-tasking in dialogue systems. *TAL: Traitment Automatique des Langues*, 43(2):131–154.

Peter Ljunglöf and Staffan Larsson. 2008. A grammar formalism for specifying ISU-based dialogue systems. In *Proc. GoTAL'08, 6th International Conference on Natural Language Processing*, number 5221 in Springer-Verlag LNCS/LNAI, Gothenburg, Sweden.

Per Martin-Löf. 1984. *Intuitionistic Type Theory*. Bibliopolis, Napoli.

Matt Oshry, editor. 2007. *Voice Extensible Markup Language (VoiceXML) 2.1*. W3C Recommendation, `http://www.w3.org/TR/voicexml21/`.

Aarne Ranta and Robin Cooper. 2004. Dialogue systems as proof editors. *Journal of Logic, Language and Information*, 13(2):225–240.

Aarne Ranta. 2004. Grammatical Framework, a type-theoretical grammar formalism. *Journal of Functional Programming*, 14(2):145–189.

M. David Sadek, Philippe Bretier, and Franck Panaget. 1997. Artimis: Natural dialogue meets rational agency. In *Proc. IJCAI'97, 15th International Joint Conference on Artificial Intelligence*, Nagoya, Japan.

Ronnie W. Smith, Alan W. Biermann, and D. Richard Hipp. 1995. An architecture for voice dialog systems based on prolog-style theorem proving. *Computational Linguistics*, 21(3):281–320.

Philip Wadler. 1997. How to declare an imperative. *ACM Computing Surveys*, 29(3):240–263.