Dialogue Systems and Type Theory

Aarne Ranta

aarne@cs.chalmers.se

Universität Tübingen, 29 November 2007.

Based on joint work with Björn Bringert, Robin Cooper, Peter Ljunglöf, Nadine Perera, and European collaboration in the TALK project (FP-6, 2004-2006)

Modified from talk at Rencontre "Pragmatique, Ludique et Continuations", Carry le Rouet, 13-15 June 2007

I

Introduction

Dialogue systems

A system that permits human-computer interaction in spoken language.

Examples:

- operatorless call centre (saves 1 EUR per call)
- hands-free controls of devices in a car
- smart houses
- travel booking

Dialogue model 1: sequence of questions

S. Where do you want to go from?

U. From Berlin.

S. Where do you want to go to?

U. To Paris.

S. Do you want to take a flight, a train, or a bus?

U. A flight.

S. Which class do you want?

U. Business class.

S. When do you want to travel?

U. On Wednesday.

S. Ok, you want a flight from Berlin to Paris in business class on Wednesday.

Dialogue model 2: information state updates

S. What can I do for you?

U. I want a flight from here to Paris on Wednesday.

S. Which class do you want?

U. Business class.

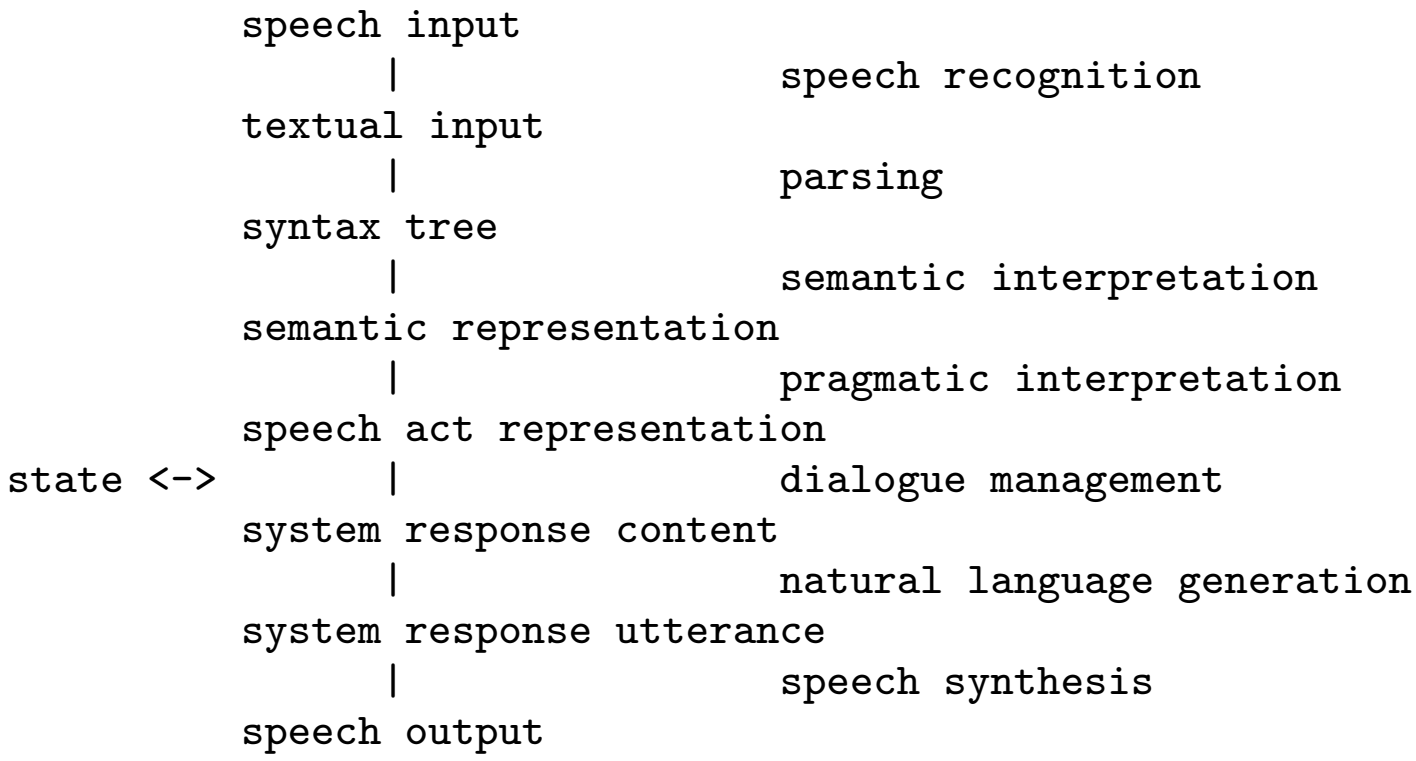
S. Ok, you want a flight from Berlin to Paris in business class on Wednesday.

Dialogue systems and linguistics

A perfect field for logically oriented linguistics

- ”all” aspects of linguistic processing
- deep semantics and pragmatics
- limited fragments of language

Dialogue system phases



Further dimension 1: multilinguality

To make the same system work in different languages

- for instance, in cars exported to different countries

Language-independent:

- semantic representation
- dialogue management

Language-dependent:

- speech recognition (acoustic model, language model)
- syntactic parser
- response generator
- speech synthesis

Further dimension 2: multimodality

User input: speech completed with pointing

I want to go from **this city** *to* **that city**

System output: speech completed with graphics

Multimodal fusion: combine speech and clicks to one semantic object.

Multimodal fission: generate adequate combination of speech and graphics

Research problems

Speech recognition: the worst bottleneck in practice

- ambiguity
- noisy environments
- demand of computing resources

Dialogue management

- understand what the user means
- give intelligent response

Engineering challenges

Formats and skills for the different components

- speech: language model formats
- parsing: grammar formats
- dialogue management: general-purpose programming languages, specialized dialogue languages

Synchrony between the components

- each component must deliver what the next one needs to receive

Portability of systems

- different OS's, small devices, cars

Result: huge amounts of human work needed!

Aspects covered in this talk

Dialogue management via information state updates

Multilinguality

Multimodal fusion

Language models from grammars

- grammar-based models in standard formats
- statistical models from synthesized corpora

Single-source production of dialogue system components

GF = Grammatical Framework

grammar = abstract syntax + concrete syntax(es)

abstract syntax = theory, in the sense of ALF logical framework

abstract syntax tree = lambda term

concrete syntax = homomorphism from trees to concrete syntax objects

concrete syntax object = record of strings and features

Abstract syntax example

```
abstract Travel = {

cat
  Itinerary ; Mode ; Class (m : Mode) ;
  Date ; City ;

fun
  Itin : City -> City -> (m : Mode) -> Class m -> Date -> Itinerary ;

  Flight, Train, Bus : Mode ;
  Business, Economy : Class Flight ;
  First, Second : Class Train ;
  Unique : Class Bus ;

  Mon, Tue, Wed, Thu, Fri, Sat, Sun : Date ;
  Paris, Stockholm, Berlin : City ;
}
```

Example of itinerary

Abstract syntax tree

```
Itin Stockholm Berlin Train First Sun
```

Concrete syntax record

```
{s = ["a train from Stockholm to Berlin in first class on Sunday"]}
```

Concrete syntax example

```
concrete TravelEng0 of Travel = open Prelude in {

lincat
  Itinerary, Mode, Class, Date, City = {s : Str} ;

lin
  Itin dep dest mod cls dat = {
    s = "a" ++ mod.s ++ "from" ++ dep.s ++ "to" ++ dest.s ++ cls.s ++ dat.s
    } ;
```

```
lin
  Flight = ss "flight" ;
  Train = ss "train" ;
  Bus = ss "bus" ;

  Business = inClass "business" ;
  Economy = inClass "econony" ;
  First = inClass "first" ;
  Second = inClass "second" ;
  Unique = ss [] ;

  Mon = onDay "Monday" ;
  Tue = onDay "Tuesday" ;
  -- etc
  Paris = ss "Paris" ;
  -- etc

oper
-- Prelude.ss : Str -> {s : Str} = \x -> {s = x} ;
  inClass : Str -> {s : Str} = \c -> ss ("in" ++ c ++ "class") ;
  onDay   : Str -> {s : Str} = \d -> ss ("on" ++ d) ;
```

Semantics and pragmatics

The type-theoretical semantics really gives the "pragmatic value" in the "language game" of travel booking:

```
lin
  Itin dep dest mod cls dat = {
    s =
      variants {
        ["I want to have"] ;
        ["can I get"] ;
        ["can you give me"] ;
        []
      } ++
      "a" ++ mod.s ++ "from" ++ dep.s ++ "to" ++ dest.s ++ cls.s ++ dat.s ++
      variants {"please" ; []}
    } ;
```

Concrete syntax example: French

```
concrete TravelFre0 of Travel = open Prelude in {
lin
  Itin dep dest mod cls dat = {
    s = "un" ++ mod.s ++ "de" ++ dep.s ++ "à" ++ dest.s ++ cls.s ++ dat.s
    } ;
  Flight = ss "vol" ;
  Train = ss "train" ;
  Bus = ss "bus" ;
  Business = inClass ["classe d'affaires"] ;
  Economy = inClass ["classe économique"] ;
  First = inClass ["première classe"] ;
  Second = inClass ["deuxième classe"] ;
  Unique = ss [] ;
  Mon = onDay "lundi" ;
  Tue = onDay "mardi" ;
  -- etc
```

```
  Paris = ss "Paris" ;
  -- etc
oper
  inClass : Str -> SS = \c -> ss ("dans" ++ "la" ++ c) ;
  onDay   : Str -> SS = \d -> ss d ;
}
```

Concrete syntax example: German

```
concrete TravelGer0 of Travel = open Prelude in {
lin
  Itin dep dest mod cls dat = {
    s = "ein" ++ mod.s ++ "von" ++ dep.s ++ "nach" ++ dest.s ++ cls.s ++ dat.s
    } ;
  Flight = ss "Flug" ;
  Train = ss "Zug" ;
  Bus = ss "Buss" ;
  Business = inClass "business" ;
  Economy = inClass "economy" ;
  First = inClass "ersten" ;
  Second = inClass "zweiten" ;
  Unique = ss [] ;
  Mon = onDay "Montag" ;
  Tue = onDay "Dienstag" ;
  -- etc
```

```
  today = ss "heute" ;
  tomorrow = ss "morgen" ;
  Paris = ss "Paris" ;
  -- etc
  here = ss "hier" ;
oper
  inClass : Str -> SS = \c -> ss ("in" ++ "der" ++ c ++ "Klasse") ;
  onDay   : Str -> SS = \d -> ss ("am" ++ d) ;
}
```

Porting a system to different languages

With the abstract-concrete architecture, is is *possible* to port a system without changing its semantic core.

But grammar writing is a lot of work, even with small grammars.

Solution: the GF Resource Grammar Library, large-coverage linguistically motivated grammars for 10 languages.

Nadine Perera and Aarne Ranta. "Dialogue System Localization with the GF Resource Grammar Library". *SPEECHGRAM 2007: ACL Workshop on Grammar-Based Approaches to Spoken Language Processing*, June 29, 2007, Prague. 2007.

Using the GF resource grammar library

The resource API: functions for building syntactic structures

```
Itin dep dest mod cls dat =
  mkS
    i_Pron
    want_VV
    (mkVP
       (mkVP go_V3 dep dest)
       (mkAdv bymeans_Prep (mkNP (indefNP mod) cls))
       dat) ;
```

To create a semantically complete new grammar, it is enough to write the domain lexicon, i.e. define the linearizations of `go_V3`, `train_N`, `class_N` etc.

GF functionalities

linearization: from tree to string

parsing: from string to tree

translation: parsing followed by linearization

random and exhaustive generation

syntax editing: we show a demo

code generation: speech recognizers, JavaScript, VoiceXML, C,...

II

Dialogue management

Desiderata for a "natural" system

Transition from linear questionnaires to information state update

From Trindi Tick List (Bohlin & al, 1999)

Q1: The system is sensitive to context

When do you want to leave?

Tomorrow.

When do you want to return?

Two days later.

3 senses of context sensitivity:

- slot being filled (departure)
- environment of dialogue (*tomorrow*)
- history of dialogue (*two days later*)

Q2 The system accepts more information than requested

Where do you want to go?

To Stockholm, tomorrow.

Q3 The system accepts an answer to a different question

Where do you want to go?

I want to leave tomorrow.

Ok. And where do you want to go?

Q4 The system accepts an answer with incomplete information

Where do you want to go?

To France.

Q5 Ambiguity is accepted

London.

London U.K. or London Ontario?

Q6 Negatively specified information is handled properly

When do you want to leave?

Not on Friday anyway.

This rules out **keyword spotting**.

Q7 The system can live with no information at all

Where do you want to go?

—

The phone service should e.g. hang up or repeat the question instead of waiting for ever.

Q8 The input can be noisy

Where do you want to go?

ZXXXXXXZZZZZZZKkK6 to Stockholm you #/%/##

This **robustness** is often achieved by keyword spotting.

Q9 The user can initiate help dialogues

How do you want to travel?

What alternatives are there?

Flight, train and bus

Q10 The user can initiate non-help dialogues

How do you want to travel?

Is there a train?

Yes.

(Has to do with the database rather than the capabilities of the system.)

Q11 The system only asks appropriate follow-up questions

How do you want to travel?

By train.

Which class?

Second.

When?

...

How do you want to travel?

By bus.

When?

Q12 The system can deal with inconsistent information

I want to leave tomorrow, Friday.

Tomorrow is Thursday. Do you mean Thursday or Friday?

Dialogue system as a proof editor

Aarne Ranta and Robin Cooper, "Dialogue Systems as Proof Editors". *Journal of Logic, Language and Information*, 13, pp. 225-240, 2004.

Information state: proof term with metavariables.

User input: editing commands.

Proof editor commands

Refinement

Focus shift

Local undo

Derived refinements

Interaction in a proof editor

Basic datastructures and commands:

```
State       = (Term, Subgoals, Constraints, SubtermPtr)
Subgoals    = [(Meta,Type)]
Constraints = [(Term,Term)]

refine  : Term        -> State -> State
delete  :               State -> State
move    : SubtermPtr  -> State -> State
```

User moves are interpreted as sequences of these basic commands.

How to give a term

Basic ways

- Write a term
- Choose from a menu
- Parse a string to a term (GF)

In general, this term contains metavariables.

Partial information

Simple model: suppress any constituents to create metavariables.

```
lin Itin dep dest mod cls dat = ss (
    optStr ("a" ++ mod.s) ++
    optStr ("from" ++ dep.s) ++
    optStr ("to" ++ dest.s) ++
    optStr cls.s ++
    optStr dat.s
    ) ;

oper Prelude.optStr : Str -> Str = \s -> variants {s ; []} ;
```

(show a demo with possible combinations)

Proof editors evaluated as dialogue systems

Q	requirement	ALF/GF	how
1	context-sensitivity	yes	actual meta; global constant
2	more information	yes	multiple refinements
3	different information	yes	move + refine
4	less information	yes	refine with incomplete term
5	ambiguity	yes	parse input + display alternatives
6	negative information	near	real parsing; add constraint
7	no answer	no	set time limit?
8	noise	near?	parse with fault tolerance
9	help menu	yes	refinement menu
10	non-help menu	yes	ask type checker
11	appropriate follow-up	yes	dependent types
12	inconsistency	yes	unsolvable constraints

Demo

Generated from a GF grammar:

- SRGS for speech recognition
- VoiceXML for dialogue management
- JavaScript for linearization
- SVG graphics for image display

W3C Standards supported by the Opera browser under MS Windows.

Björn Bringert, "Generating Dialogue Systems from Grammars". SIGDIAL 2007.

Abstract syntax of the Pizza demo

```
abstract Pizza = {

cat
  Order ;
  Item ;
  [Item] ;
fun
  order : [Item] -> Order ;
  pizza : PizzaNumber -> PizzaSize -> Toppings -> Item ;
  drink : DrinkNumber -> DrinkSize -> DrinkType -> Item ;
cat
  Output ;
fun
  confirm : Order -> Number -> Output ;
-- ... 64 lines altogether
}
```

III

Multimodality

Parallel multimodality: SVG pictures in Pizza

```
concrete PizzaDraw of Pizza = {
lin
  order xs = { s = "[" ++ xs.s ++ "]" } ;
  BaseItem = { s = empty } ;
  ConsItem x xs = { s = x.s ++ "," ++ xs.s } ;

  pizza n s ts =
    { s = call2 "number" n.s (call2 "size" s.s
            (call2 "above" ts.s (image "pizza").s)) } ;

  small  = { s = "0.33" } ;
  medium = { s = "0.67" } ;
  large  = { s = "1.00" } ;
  cheese    = image "cheese" ;
  anchovies = image "anchovies" ;
-- ... 73 lines altogether
}
```

Integrated multimodality

Idea: include **demonstratives** in the grammar.

Speech and pointing data are stored in different record fields.

```
{speech : Str ; click : Str}
```

Asynchronous parsing + multimodal fusion: the order of clicks is preserved, not exact timing.

Compositionality is preserved when combining demonstratives into a sequence.

Björn Bringert, Robin Cooper, Peter Ljunglöf, and Aarne Ranta, "Multimodal Dialogue System Grammars". *Proceedings of DIALOR'05, Ninth Workshop on the Semantics and Pragmatics of Dialogue*, Nancy, France, June 9-11, 2005, 2005.

Demonstratives

```
cat
  Click ;

lincat
  Click = {s : Str} ;

  City = {s : Str ; click : Str} ; -- new lincat

lin
  Paris = {s = "Paris" ; click = []} ; -- non-demonstrative

fun
  here : Click -> City ;
lin
  Paris c = {s = "here" ; click = c.s} ;  -- demonstrative
```

Multimodal fusion

The speech and the click components are concatenated (with a separator)

```
lin
  Itin dep dest mod cls dat = ss (
    "a" ++ mod.s ++ "from" ++ dep.s ++ "to" ++ dest.s ++ cls.s ++ dat.s ++
    ";" ++
    dep.click ++ dest.click
    ) ;
```

This is what the parser receives:

```
  a bus from here to here on Monday ; Paris Berlin
```

Asynchronicity is confirmed in user experiments: clicks are not aligned with the "here"s.

Demonstrative vs. indexical "here"

Indexical "here" refers to the place where the user is in, and has no associated click:

```
fun hereInd : City ;
lin hereInd = {s = "here" ; click = []} ;
```

Because of asynchronicity, the following is ambiguous:

```
a bus from here to here on Monday ; Paris
```

(Demo parsing in GF)

Demo film

Trams and buses in Gothenburg.

6 languages + multimodality

Multimodality, parsing, generation, and language models implemented by GF grammars

Dialogue management implemented in GoDiS (Larsson 2001) (with Prolog code generated from GF)

Beware: there are some grammar errors in the French version!

Joint work of the Gothenburg group in TALK (2006)

IV

Language model generation

Speech recognizer components

Language-independent: signal processing, statistical computation, . . .

Language-dependent:

acoustic model: possible phoneme sequences

lexicon: mapping phoneme sequences to written words

language model: filtering "meaningful" word sequences

The need of a language model

How to choose the correct alternative?

```
je n'aime que cette femme

je n'aime que cette femmes

je n'aimes que cette femme

jeune aime que cette femme

je n'aime que sept femmes
```

Grammaticality, in a general sense, is not enough (and might be too inefficient, too)

In practice, domain and context dependent filtering are needed

Two kinds of language models

Grammar-based: use a context-free or regular grammar as filter

Statistical: compute n-gram probabilities from a corpus

Both can be restricted to a domain.

Both are labour-intensive to create.

Grammar-based language models from GF

It is easy to write domain grammars.

What remains is to generate them in formats required by speech recognition systems

- context-free approximations: GSL, JSG, SRGS,. . .
- regular approximations: HTK/ATK

Björn Bringert. Speech Recognition Grammar Compilation in Grammatical Framework *SPEECHGRAM 2007: ACL Workshop on Grammar-Based Approaches to Spoken Language Processing*, June 29, 2007, Prague. 2007.

(demo?)

Statistical language models from GF

It is easy to generate sentences from a grammar: one can build a huge artificial corpus.

The resulting SLM has a wider coverage and a lower word-error rate than the grammar-based model generated from the same grammar.

But it requires robust parsing to analyse the speech recognition output.

Rebecca Jonson. ''Generating statistical language models from interpretation grammars in dialogue system''. *Proceedings of EACL'06, Trento, Italy*. 2006.

SLM with dependent types

An "ordinary" grammar is likely to generate sentences that would never appear in a natural corpus

```
  a train from Paris to Berlin in the second class
 *a bus from Paris to Berlin in the second class
 *a flight from Paris to Berlin in the second class
```

By the use of dependent type checking, GF can filter such sentences out from the corpus.

```
> i TravelEng0.gf
> gt | l | ? wc -l
    4320
> gt | pt -transform=typecheck | l | ? wc -l
    1440
```

(demo?)

V

Conclusions

Benefits of type theory

Formalization of semantic model ("ontology")

Proof-editor model of interaction

We haven't even mentioned:

- context dependencies via dependent types
- the inherent pragmatics of type theory (judgements vs. propositions)

Benefits of GF

Abstract syntax providing language- and modality-independent representation

Concrete syntax permitting different languages and modalities

Compilers generating many different formats

Resource grammar library helping in localization

How good are our dialogue systems?

Limited domain

Limited grammar (the user has to learn "commands")

Speech recognition is still unreliable

Systems are still not very portable

Future work

Our goal: push-button method to generate limited-domain multimodal dialogue systems in multiple languages, for multiple platforms

Others' goals: more AI in dialogue management, closer to free speech input

References

TALK project, FP-6, 2004-2006: `http://www.talk-project.org/`

Björn Bringert, "Generating Dialogue Systems from Grammars". SIGDIAL 2007.

Björn Bringert. Speech Recognition Grammar Compilation in Grammatical Framework ACL/SPEECHGRAM 2007.

Nadine Perera and Aarne Ranta. "Dialogue System Localization with the GF Resource Grammar Library". ACL/SPEECHGRAM 2007.

Rebecca Jonson. Generating statistical language models from interpretation grammars in dialogue system. EACL 2006.

Björn Bringert, Robin Cooper, Peter Ljunglöf, and Aarne Ranta, Multimodal Dialogue System Grammars. DIALOR 2005.

Aarne Ranta and Robin Cooper. Dialogue Systems as Proof Editors. *Journal of Logic, Language and Information* **13**, pp. 225-240, 2004.

Aarne Ranta, "Grammatical Framework. A Type Theoretical Grammar Formalism", *The Journal of Functional Programming* **14**(2), pp. 145-189, 2004.