Introduction to DS-TTR: a personal view

Eleni Gregoromichelaki
Osnabrück University and
King’s College London

www.kcl.ac.uk/research/groups/ds

July 17, 2017
Introductory Motivation

What is grammar?

TTR to formalise conceptual structure

TTR elements adopted
Dynamic Syntax
DS elements adopted
Dynamic Syntax (DS)

DS-TTR

Hannes Rieser’s Questions

General conclusions

DS-TTR and cognition - abandoning competence vs performance

Appendix 1

Appendix 2
Outline

Introductory Motivation

What is grammar?

TTR to formalise conceptual structure

TTR elements adopted
Dynamic Syntax
DS elements adopted
Dynamic Syntax (DS)

DS-TTR

Hannes Rieser’s Questions

General conclusions

DS-TTR and cognition - abandoning competence vs performance

Appendix 1

Appendix 2
real conversational happens bit by bit, without respecting the boundaries of sentences:

half-starts, suggested add-ons, pauses, interruptions, corrections

(1) [Context: Friends of the Earth club meeting]
A: So what is that? Is that er... booklet or something?
B: It’s a book
C: Book
B: Just ... talking about al you know alternative
D: On erm... renewable yeah
B: energy really I think......
A: Yeah [BNC:D97]
real conversation happens bit by bit, without respecting the boundaries of sentences:

half-starts, suggested add-ons, pauses, interruptions, corrections

(1) [Context: Friends of the Earth club meeting]
A: So what is that? Is that er... booklet or something?
B: It’s a book
C: Book
B: Just ... talking about al you know alternative
D: On erm... renewable yeah
B: energy really I think......
A: Yeah [BNC:D97]
ordinary conversation is highly fragmentary and incremental
ordinary conversation is highly fragmentary and incremental
various actions performed with non-sentential/propositional elements, e.g. repair (clarifications, corrections, etc)
ordinary conversation is highly fragmentary and incremental

various actions performed with non-sentential/propositional elements, e.g. repair (clarifications, corrections, etc)

however, syntactic/semantic dependencies still constrain/direct behaviour:
ordinary conversation is highly fragmentary and incremental

various actions performed with non-sentential/propositional elements, e.g. repair (clarifications, corrections, etc)

however, syntactic/semantic dependencies still constrain/direct behaviour:

- **split-utterances**: syntactic/semantic dependencies hold across change of speakers:

  (7) A: **Have you** read ...
  B: **any** of your chapters?
  cf. *I have read any of your chapters*
ordinary conversation is highly fragmentary and incremental

various actions performed with non-sentential/propositional elements, e.g. repair (clarifications, corrections, etc)

however, syntactic/semantic dependencies still constrain/direct behaviour:

split-utterances: syntactic/semantic dependencies hold across change of speakers:

(9) A: Have you read ...
    B: any of your chapters?
    cf. *I have read any of your chapters

(10) A: Oh, I am so sorry, did you burn
    B: myself? No, its OK.
    cf. # Oh, I am so sorry, did you burn myself?
⇒ the “grammar”, as a holistic model, needs to be able to express
(a) the **incremental** licensing and interpretation of NL strings
⇒ the “grammar”, as a holistic model, needs to be able to express

(a) the **incremental** licensing and interpretation of NL strings
(b) the **context shift** (e.g. change of speaker-roles) within a single clause,
the “grammar”, as a holistic model, needs to be able to express

(a) the incremental licensing and interpretation of NL strings
(b) the context shift (e.g. change of speaker-roles) within a single clause,
(c) while still implementing traditional syntactic/syntactic constraints:

(13) a. John likes **himself** vs. *him

b. John likes everyone [ Mary **does** ] vs.
   *John likes everyone [ Mary admires the man [ who **does** ] ]
conversational data and the nature of grammar: the view from DS-TTR

- no separate syntactic level of representation:
  - no syntactic categories for strings of words;
  - no phrase-structure rules;
  - no “constructions”

- grammatical ontology of processes (rather than representations)
  - incrementality, prediction, and underspecification as properties of the grammar (“syntax”)
Introductory Motivation

What is grammar?

**TTR to formalise conceptual structure**
  - TTR elements adopted
  - Dynamic Syntax
  - DS elements adopted
  - Dynamic Syntax (DS)

**DS-TTR**

Hannes Rieser’s Questions

General conclusions
  - DS-TTR and cognition - abandoning competence vs performance

Appendix 1

Appendix 2
- Martin-Löf Type Theory
- **objects/entities** belong to types
- **propositions** are regarded as types of proofs ("propositions as types" principle)
- **proofs** are objects
  - e.g. the proofs of *there is a prime number between 212 and 222* are the prime numbers between 212 and 222
- Ranta (1984): a proof of
  - (14) John hugged Mary.
  - is some event during which John hugged Mary
type theoretical judgements:
  ▶ a : T  ("object a is of type T")
Type Theory With Records

- type theoretical judgements:
  - \( a : T \) ("object \( a \) is of type \( T \)"")
- types in TTR: not atomic, but complex
Type Theory With Records

- type theoretical **judgements**:
  - \( a : T \) ("object \( a \) is of type \( T \)"")
- **types** in TTR: not atomic, but complex
- **records** are sequences of label/value pairs:
  \[
  \begin{bmatrix}
  l_1 &= v_1 \\
  l_2 &= v_2 \\
  l_3 &= v_3
  \end{bmatrix}
  \]
- type theoretical **judgements**:
  - $a : T$ ("object $a$ is of type $T$")
- **types** in TTR: not atomic, but complex
- **records** are sequences of label/value pairs:
  $\begin{align*}
  l_1 &= v_1 \\
  l_2 &= v_2 \\
  l_3 &= v_3
  \end{align*}$
- **record types** are sequences of label/type pairs:
  $\begin{align*}
  l_1 : T_1 \\
  l_2 : T_2 \\
  l_3 : T_3
  \end{align*}$
records model complex entities,
  e.g., events (including contexts)
- records model complex entities,
  - e.g., events (including contexts)
- record types model categorisations of events/individuals
- records model complex entities,
  - e.g., events (including contexts)
- record types model categorisations of events/individuals
  - classification of a situation to be of a certain type with potential for further elaboration
records are sequences of label/value pairs:

\[
\begin{bmatrix}
    l_1 &= v_1 \\
    l_2 &= v_2 \\
    l_3 &= v_3
\end{bmatrix}
\]
records are sequences of label/value pairs:

\[
\begin{bmatrix}
    l_1 &= v_1 \\
    l_2 &= v_2 \\
    l_3 &= v_3
\end{bmatrix}
\]

record types are true iff they are inhabited/witnessed
types can be dependent on earlier (higher-up) types:

\[
\begin{bmatrix}
  l_1 & : & T_1 \\
  l_2 & : & T_2(l_1) \\
  l_3 & : & T_3(l_1, l_2)
\end{bmatrix}
\]
- Types can be **dependent** on earlier (higher-up) types:

  
  \[
  \begin{array}{c}
  l_1 : T_1 \\
  l_2 : T_2(l_1) \\
  l_3 : T_3(l_1, l_2)
  \end{array}
  \]

- **Recursivity**: We can have nested records and record types:

  
  \[
  \begin{array}{c}
  l_1 : T_1 \\
  l_2 : \begin{array}{c}
  l'_1 : T'_1 \\
  l'_2 : T'_2
  \end{array} \\
  l_3 : T_3(l_1, l_2, l'_1, l'_2)
  \end{array}
  \]
we have functional record types:

$$\lambda r : \left[ \begin{array}{c} l_1 : T_1 \\ l_2 : T_2 \end{array} \right] \left( \begin{array}{c} l_3 : T_3 \\ l_4 : T_4(r.l_1, r.l_2) \end{array} \right)$$
TTR (Type Theory with Records) – appealing features

- synthesis of ideas of **frame semantics** and **Montague Grammar**
  - invoked frames as background knowledge
  - integrates standard formal semantic tools like the lambda calculus
- (potentially) constructivist: meanings as programs, as proofs
  (potentially, actions all the way down)
TTR (Type Theory with Records) – appealing features

- synthesis of ideas of **frame semantics** and **Montague Grammar**
  - invoked frames as background knowledge
  - integrates standard formal semantic tools like the lambda calculus
- (potentially) constructivist: meanings as programs, as proofs (potentially, actions all the way down)
- TTRs subtype relation allows complete semantics extraction for any partial tree, and incremental further specification as parsing proceeds
TTR (Type Theory with Records) – appealing features

- synthesis of ideas of frame semantics and Montague Grammar
  - invoked frames as background knowledge
  - integrates standard formal semantic tools like the lambda calculus
- (potentially) constructivist: meanings as programs, as proofs (potentially, actions all the way down)
- TTRs subtype relation allows complete semantics extraction for any partial tree, and incremental further specification as parsing proceeds
- sublexical conceptual structure
  - distributed representations
  - atomic concepts correspond to patterns of activation (not single neurons)
  - complex record structures for single concepts (not atoms as in standard logics)
Probabilistic Type Theory with Records (probTTR)

- types are grounded in classifiers
- interface with perception: NL semantics + perception expressible in the same formalism (TTR)
Probabilistic Type Theory with Records (probTTR)
- types are grounded in classifiers
- interface with perception: NL semantics + perception expressible in the same formalism (TTR)

Connection with action (Cooper(2014; fthcmg))
- judgements as act(ion)s
- modelling of acts of creation of witnesses of types
Probabilistic Type Theory with Records (probTTR)

- types are grounded in classifiers
- interface with perception: NL semantics + perception expressible in the same formalism (TTR)

Connection with action (Cooper(2014; fthcmg))

- judgements as act(ion)s
- modelling of acts of creation of witnesses of types

However, TTR is static
Arrival: holistic logograms
Arrival: holistic logograms
Arrival: holistic logograms
Arrival: holistic logograms

[Image of two figures in orange suits, one holding a sign with the word 'HUMAN']

[Image of a circular pattern with black splatters]

[Image of three human figures in a misty background]
TTR - introducing dynamics: desiderata for incrementalising TTR

- dynamic incremental conceptualisation implemented in DS-TTR to provide actions to:
  - modify, delete, add fields while the rest stay the same (lexical semantics)
  - compute similarity between concepts (record types (e.g. metaphor, quotation)
  - check subsumption incrementally (generation, repair, Hough)
  - extract all available semantic information incrementally (Hough)
  - encompass multimodal aspects of processing (e.g. gesture, affect, Eshghi)
  - model defeasible inference rules as functions from objects of a type to another type (e.g. associative view of reasoning) (Ellen)
  - model frequency and context effects as probabilistic type assignments
DS-TTR:
- conceptualises grammar as a set of actions
- no syntactic level of representation for words
- grammatical/lexical actions build/linearise (ad hoc) conceptual structure
- procedural definitions: constraints on how not what
- single level of operations integrating all aspects of context-dependency
Dynamic Syntax: conceptual structure

- Incrementally building/linearising conceptual structure

- Nodes decorated with $Ty()$ type and $Fo()$ formula labels

John likes Mary:

$$
\begin{align*}
& Ty(t), \\
& Fo(\text{like}(\text{John}, \text{Mary})) \\

& Ty(e), \\
& Fo(\text{John}) \\
& Ty(e \rightarrow t), \\
& Fo(\lambda x. \text{like}(x, \text{Mary})) \\

& Ty(e), \\
& Fo(\text{Mary}) \\
& Ty(e \rightarrow (e \rightarrow t)), \\
& Fo(\lambda y \lambda x. \text{like}(x, y))
\end{align*}
$$
Dynamic Syntax: conceptual structure

- Incrementally building/linearising conceptual structure

- Nodes decorated with $Ty()$ type and $Fo()$ formula labels

John likes Mary:

$\text{Ty}(t)$, $\text{Fo}(\text{like}(\text{John}, \text{Mary}))$

$\text{Ty}(e)$, $\text{Fo}(\text{John})$

$\text{Ty}(e \rightarrow t)$, $\text{Fo}(\lambda x. \text{like}(x, \text{Mary}))$

$\text{Ty}(e)$ $\text{Ty}(e \rightarrow (e \rightarrow t))$, $\text{Fo}(\lambda y \lambda x. \text{like}(x, y))$

- Daughter order does not reflect sentence order

- Nodes interpretable as terms in the $\lambda$-calculus
Multilevel underspecification + update (discontinuity)

- Pronouns, elliptical elements, tree relations can be introduced as **underspecified** and in need of update:

  *Who did Mary upset?*

  Starting with an **unfixed** (underspecified) node

  \[
  Tn(0), \ldots ? Ty(t), \\
  \langle \uparrow * \rangle Tn(0) \\
  ? Ty(e), ? \exists x Tn(x), \diamond
  \]
Processing Who did Mary upset

\[
\langle \uparrow^* \rangle T_n(0) \quad \text{WH : e, } \exists x T_n(x)
\]

\[
T_n(0), \ldots ? T_y(t), \diamondsuit
\]
Multilevel underspecification + incremental update

- Processing *Who did Mary upset*
- Auxiliary projects subject-predicate template
Multilevel underspecification + incremental update

Processing *Who did Mary upset*

\[ Tn(0), ?Ty(t), \diamond \]

\[ \langle \uparrow^* \rangle Tn(0) \]

\[ \text{WH: e, } \exists x Tn(x) \]

\[ \langle \uparrow^* \rangle Tn(0) \]

\[ \langle \uparrow_1 \rangle Tn(0) \]

\[ \text{Mary': e, } \]

\[ \langle \uparrow_0 \rangle \langle \uparrow^* \rangle Tn(0) \]

\[ \exists x Tn(x) \]

\[ s_{PAST} \]

\[ ?Ty(e_s \rightarrow t) \]

\[ U_{AuxDO} \]

\[ ?Ty(e) \]

\[ ?\exists x \text{Fo}(x) \]

\[ Ty(e \rightarrow (e_s \rightarrow t)) \]
Processing **Who did Mary upset**

\[ Tn(0), ?Ty(t) \]

\[ \langle \uparrow* \rangle Tn(0) \]

**WH:** e

\[ ?\exists x Tn(x) \]

\[ SPAST \]

\[ ?Ty(e_s \rightarrow t) \]

\[ U_{AuxDO} \]

\[ ?\exists x Fo(x) \]

\[ Ty(e \rightarrow (e_s \rightarrow t)) \]

\[ Mary' : e \]

\[ ?Ty(e), \Diamond \]

**Upset**
Completing the processing of *Who did Mary upset*

\[ Upset'(WH)(Mary')(s_{PAST}) : t, \diamond \]

\[ s_{PAST} \]

\[ Upset'(WH)(Mary') : e_s \rightarrow t \]

\[ Mary' : e \]

\[ Upset'(WH) : e \rightarrow (e_s \rightarrow t) \]

\[ WH : e \]

\[ Upset' \]
Grammaticalised underspecification

Processing John likes himself

?Ty(t), ♦
Grammaticalised underspecification

Processing John likes himself

\[ ?Ty(t) \]
\[ \diamond, ?Ty(e) \quad ?Ty(e \rightarrow t) \]
Processing *John* likes himself

\[ \text{\checkmark, } Ty(e) \rightarrow Ty(e \rightarrow t) \]
Grammaticalised underspecification

Processing John likes himself

\[ Ty(e) \quad ?Ty(e \rightarrow t), \diamond \]

?Ty(t)
Processing John likes himself

\[ ?Ty(t) \]
\[ Ty(e) \quad ?Ty(e \rightarrow t) \]
\[ John' \]
\[ \diamond, ?Ty(e)\quad Ty(e \rightarrow (e \rightarrow t)) \]
\[ \lambda x.\lambda y.\text{like}(y, x) \]
Grammaticalised underspecification

Processing *John likes himself*

\[
\begin{align*}
&Ty(e) \\
&Ty(e) \rightarrow Ty(e \rightarrow t) \\
&Ty(t) \\
\end{align*}
\]

**IF**

\[\uparrow_0 \uparrow_1 \downarrow_0 \text{Fo}(X), Ty(e)\]

**THEN**

\[\text{put}(\text{Fo}(X)); \text{put}(Ty(e))\]

\[\diamond, ?Ty(e) \rightarrow Ty(e \rightarrow (e \rightarrow t)) \rightarrow \lambda x. \lambda y. \text{like}(y, x)\]
Grammaticalised underspecification

Processing *John likes himself*

```
?Ty(t)
   Ty(e)  ?Ty(e → t)
     Ty(e)  Ty(e → (e → t))
      John'  λx.λy.like(y, x)
```

`Ty(e)` and `Ty(e → (e → t))` correspond to the type of the subject and the verb phrase, respectively. The underspecification is represented by `?Ty(t)`. The notation `λx.λy.like(y, x)` denotes a lambda expression representing the semantic function of the verb phrase.
Grammaticalised underspecification

Processing *John likes himself*

\[ ?Ty(t) \]

\[ Ty(e) \]
\[ Ty(e \rightarrow t), \diamond \]

\[ Ty(e) \]
\[ Ty(e \rightarrow (e \rightarrow t)) \]
\[ \lambda x.\lambda y.\text{like}(y, x) \]
Processing *John likes himself*

\[
?Ty(t), \diamond
\]

\[
Ty(e) \quad Ty(e \to t) \\
John' \quad \lambda y. like(y, John')
\]

\[
Ty(e) \quad Ty(e \to (e \to t)) \\
John' \quad \lambda x. \lambda y. like(y, x)
\]
Grammaticalised underspecification

Processing *John likes himself*

\[
\begin{align*}
like(\text{John}', \text{John}') \\
Ty(t) \\
\text{Ty}(e) & \quad \text{Ty}(e \to t) \\
\text{John}' & \quad \lambda y.\text{like}(y, \text{John}') \\
\text{Ty}(e) & \quad \text{Ty}(e \to (e \to t)) \\
\text{John}' & \quad \lambda x.\lambda y.\text{like}(y, x)
\end{align*}
\]
(13c) It’s possible ... I am wrong

\[ Tn(n), U, ?\exists xFo(x) \]

Possible'
(13c) It’s possible ... I am wrong

\[ Tn(n), U, ?\exists x Fo(x) \]

\[ \langle \uparrow^* \rangle Tn(n), ?\exists x Fo(x), Wrong'(Eleni'), \diamond \]

\[ Eleni' \quad Wrong' \]
(13c) It’s possible ... I am wrong

\[ ?Ty(t) \]

\[ Tn(n), U \]

\[ Possible' \]

\[ \langle \uparrow \ast \rangle Tn(n), ?\exists xFo(x), Wrong'(Eleni'), \diamond \]

\[ Unify \]

\[ Eleni' Wrong' \]
(13c) It’s possible ... I am wrong

\[
?Ty(t) \\
Tn(n), U \\
?\exists x \text{Fo}(x), \text{Wrong'}(\text{Eleni'})
\]

Possible'
e.g. Who upset himself? John did.

Context

TREE: 

\[ \text{Upset}'(\text{WH})(\text{WH}) \]

\[ \text{WH} \quad \text{Upset}'(\text{WH}) \]

\[ \text{WH} \quad \text{Upset}' \]

Incomplete actions: \( \langle \ldots \text{upset, himself}, \text{completion, evaluation} \rangle \)

Complete parse tree

\[ \text{Ty}(t), \]
\[ \text{Upset}'(\text{John'}) (\text{John'}) \]

\[ \text{U}, \quad \text{Upset}'(\text{John'}) \]
\[ \text{Ty}(e \rightarrow t) \]

\[ \text{John'} \quad \text{John'} \quad \text{Upset}' \]
Relative clauses: pairs of \textsc{Linked} trees evaluated as conjunction

e.g. Bill, who fainted, smokes.

\[
\text{smoke}'(\text{bill}') \land \text{faint}'(\text{bill}')
\]

Also used for apposition, clarification and confirmation, implicatures . . .
Re-running actions – ACE

- Antecedent Contained Ellipsis

e.g. Bill saw someone [ that John did ]
Antecedent Contained Ellipsis

e.g. Bill saw someone [ that John did ]

\[ Tn(0), Ty(t) \]

\[ \text{Bill'} \quad \text{?Ty}(e \rightarrow t) \]

\[ \text{?Ty}(e) \quad \text{See'} \]

\[ \text{?Ty}(cn) \quad \lambda P(\epsilon, x, P(x)) \]

\[ x, Ty(e) \]

\[ \text{Person'} \quad \text{Bill saw someone} \]
Antecedent Contained Ellipsis

e.g. Bill saw someone [ that John did ]

\[
Tn(0), \ ?Ty(t)
\]

\[
Bill' \quad \ ?Ty(e \rightarrow t)
\]

\[
?Ty(e) \quad \text{See'}
\]

\[
?Ty(cn) \quad \lambda P(\epsilon, x, P(x))
\]

\[
x, Ty(e) \quad \text{Person'}
\]

Bill saw someone that
Antecedent Contained Ellipsis

e.g. Bill saw someone [ that John did ]

\[ Tn(0), ?Ty(t) \]

\[ Bill' \]

\[ ?Ty(e \rightarrow t) \]

\[ ?Ty(e) \]

\[ See' \]

\[ ?Ty(cn) \]

\[ \lambda P(\epsilon, x, P(x)) \]

\[ x, Ty(e) \]

\[ Person' \]

\[ x \]

\[ John' \]

\[ U \]

Bill saw someone that John did
Antecedent Contained Ellipsis

\[ Tn(0), ?Ty(t) \]

\[ Bill' \]

\[ ?Ty(e \rightarrow t) \]

\[ ?Ty(e) \]

\[ ?Ty(cn) \]

\[ \lambda P(\epsilon, x, P(x)) \]

\[ x, Ty(e) \]

\[ Person' \]

\[ L \]

\[ ?Ty(t) \]

\[ x \]

\[ John' \]

\[ U \]

\[ ?Ty(e) \]

\[ See' \]

Bill saw someone that John did

RE-RUN: see
Re-running actions – ACE

- Antecedent Contained Ellipsis

  e.g. Bill saw someone [ that John did ]

  \[ Tn(0), ?Ty(t) \]

  \[ Bill' \]

  \[ ?Ty(e \rightarrow t) \]

  \[ ?Ty(e) \]

  \[ See' \]

  \[ ?Ty(cn) \]

  \[ \lambda P(\epsilon, x, P(x)) \]

  \[ x, Ty(e) \]

  \[ Person' \]

  \[ John' \]

  \[ ?Ty(t) \]

  \[ U \]

  \[ x \]

  \[ See' \]

  Bill saw someone that John did

  RE-RUN: see

  UNIFICATION
Re-running actions – ACE

- Antecedent Contained Ellipsis
  e.g. Bill saw someone [ that John did ]

\[ Tn(0), ?Ty(t) \]

- Bill'
- ?Ty(e → t)
- ?Ty(e)
- See'
- ?Ty(cn)
- \( \lambda P(\epsilon, x, P(x)) \)
- x, Ty(e)
- Person'

\[ Ty(t) \]

- John'
- U
- x
- See'

Bill saw someone that John did

RE-RUN: see
UNIFICATION
COMPLETION of tree:
e.g. Bill saw someone that John did

\[
\begin{align*}
&\text{Ty}(t) \\
&\text{See}'(\epsilon, x, \text{Person}'(x) \land \text{See}'(x)(John)(Bill'))
\end{align*}
\]

\[
\begin{align*}
&\text{Bill'} \\
&\text{Ty}(e \rightarrow t)
\end{align*}
\]

\[
\begin{align*}
&\text{Ty}(e) \\
&\epsilon, x, \text{Person}'(x) \land \text{See}'(x)(John)
\end{align*}
\]

\[
\begin{align*}
&\text{Ty}(cn) \\
&x, \text{Person}'(x) \land \text{See}'(x)(John')
\end{align*}
\]

\[
\begin{align*}
&\lambda P(\epsilon, x, P(x)) \\
&x, \text{Ty}(e) \land \text{Person'}
\end{align*}
\]

\[
\begin{align*}
&\text{See}' \\
&\text{L} \\
&\text{John'}
\end{align*}
\]

\[
\begin{align*}
&\text{U} \\
&x \land \text{See'}
\end{align*}
\]
Speakers go through the same actions, except they also have a somewhat richer goal tree.

Each word licensed must update partial tree towards the goal tree via subsumption constraint

* Generating **Someone fainted**

**GOAL TREE**

\[
Ty(t), \diamond
\]
\[
faint(\epsilon, \text{person}(x))
\]
\[
Ty(e), \quad Ty(e \rightarrow t)
\]
\[
\epsilon, x, \text{person}(x) \quad \lambda y. faint(y)
\]

**TEST TREE**

\[
?Ty(t), \diamond
\]
Speakers go through the same actions, except they also have a somewhat richer goal tree.

Each word licensed must update partial tree towards the goal tree via *subsumption* constraint

* Generating **Someone fainted**

**GOAL TREE**

\[ Ty(t), \diamond \]
\[ faint(\epsilon, person(x)) \]
\[ Ty(e), Ty(e \rightarrow t) \]
\[ \epsilon, x, person(x) \]
\[ \lambda y. faint(y) \]

**TEST TREE**

\[ ?Ty(t), \]
\[ ?, Ty(e) \]
\[ ?, Ty(e \rightarrow t) \]
Speakers go through the same actions, except they also have a somewhat richer goal tree.

Each word licensed must update partial tree towards the goal tree via \textit{subsumption} constraint

* Generating \textbf{Someone fainted}

\begin{align*}
\text{GOAL TREE} & \\
Ty(t), & \diamond \\
faint(\epsilon, person(x)) & \\
\quad Ty(e), & Ty(e \rightarrow t) \\
\quad \epsilon, x, person(x) & \lambda y. faint(y) \\
\text{TEST TREE} & \\
? Ty(t), & \\
? Ty(e \rightarrow t) & \\
\diamond, Ty(e) & \epsilon, x, person(x)
\end{align*}

Gen: \textbf{“Someone}
Speakers go through the same actions, except they also have a somewhat richer goal tree.

Each word licensed must update partial tree towards the goal tree via subsumption constraint

* Generating Someone fainted

** GOAL TREE **

\[ Ty(t), \diamond \]
\[ faint(\epsilon, person(x)) \]
\[ Ty(e), Ty(e \rightarrow t) \]
\[ \epsilon, x, person(x) \]
\[ Ty(e), Ty(e \rightarrow t) \]
\[ \epsilon, x, person(x) \]

** TEST TREE **

\[ ?Ty(t), Ty(e) \]
\[ ?Ty(e \rightarrow t), \diamond \]
\[ Ty(e), Ty(e \rightarrow t), Ty(e), Ty(e \rightarrow t) \]
\[ Ty(e), Ty(e \rightarrow t), Ty(e), Ty(e \rightarrow t) \]

Gen: “Someone"
Speakers go through the same actions, except they also have a somewhat richer goal tree.

Each word licensed must update partial tree towards the goal tree via *subsumption* constraint

* Generating **Someone fainted**

**GOAL TREE**

\[
Ty(t), \diamond \quad faint(\epsilon, person(x))
\]

\[
Ty(e), \quad Ty(e \rightarrow t) \\
\epsilon, x, person(x) \quad \lambda y. faint(y)
\]

**TEST TREE**

\[
?Ty(t),
\]

\[
Ty(e) \quad ?Ty(e \rightarrow t), \diamond \\
\epsilon, x, person(x) \quad \lambda y. faint(y)
\]

Gen: “Someone fainted”
Speakers go through the same actions, except they also have a somewhat richer goal tree.

Each word licensed must update partial tree towards the goal tree via *subsumption* constraint

* Generating **Someone fainted**

---

**GOAL TREE**

\[
Ty(t), \diamond \\
\text{faint}(\varepsilon, \text{person}(x))
\]

\[
Ty(e), \quad Ty(e \rightarrow t) \\
\varepsilon, x, \text{person}(x), \quad \lambda y. \text{faint}(y)
\]

**TEST TREE**

\[
Ty(t), \diamond \\
\text{faint}(\varepsilon, x, \text{person}(x))
\]

\[
Ty(e), \quad Ty(e \rightarrow t) \\
\varepsilon, x, \text{person}(x), \quad \lambda y. \text{faint}(y)
\]

Gen: “**Someone fainted**”
Alignment: rerunning of actions induces parallelism

- Using actions from context – sloppy readings:

(1) A: John upset his mother.
   B: Harry too.

(2) A: The man [who arrested John] failed to read him his rights.
   B: The man who arrested Tom did too.
Using actions from context – sloppy readings:

(1) A: John upset his mother.
    B: Harry too.

(2) A: The man [who arrested John] failed to read him his rights.
    B: The man who arrested Tom did too.

Also more general parallellism effects, e.g. scope:

(4) A: A consultant interviewed every patient.
    B: A junior doctor too.
Outline

Introductory Motivation
  What is grammar?

TTR to formalise conceptual structure
  TTR elements adopted
  Dynamic Syntax
  DS elements adopted
  Dynamic Syntax (DS)

**DS-TTR**

Hannes Rieser’s Questions

General conclusions
  DS-TTR and cognition - abandoning competence vs performance

Appendix 1

Appendix 2
DS-TTR: parsing and generation

- from strings to conceptual structure (TTR) or vice-versa
DS-TTR: parsing and generation

- from strings to conceptual structure (TTR) or vice-versa
- *John arrived.*
from strings to conceptual structure (TTR) or vice-versa

John arrived.

\[
\begin{align*}
\text{John arrived} & \\
\rightarrow & \\
\diamond, \ Ty(t), & \left[ \begin{array}{c}
x : john' \\
p : \text{arrive}'(x)
\end{array} \right] \\
\text{Ty}(e), & \left[ \begin{array}{c}
x : john'
\end{array} \right] \\
\text{Ty}(e \rightarrow t), & \left[ \begin{array}{c}
x : e \\
p : \text{arrive}'(x)
\end{array} \right] \\
\lambda [ \ x : e \ ]. & \left[ \begin{array}{c}
x : e \\
p : \text{arrive}'(x)
\end{array} \right]
\end{align*}
\]
DS-TTR: Types (simplified)

Type

BasicType
- e(Ind)
- e_s(Event)

PType
- t
  - t(e)
  - t(e_s, e)
  - t(e_s)

RecordType

...
parsing/linearising (syntactic/lexical):
- go [treenode]
- make[treenode]
- put[field/value/label/...]
- IF [value] THEN [actions], ELSE [...] 
- run(list⟨actions⟩[...])
DS-TTR: actions from DS +

- parsing/linearising (syntactic/lexical):
  - go [treenode]
  - make[treenode]
  - put[field/value/label/...]
  - IF [value] THEN [actions], ELSE [...]
  - run(list⟨actions⟩)[...])

- manipulating complex type articulation
  - add[fields]
  - remove[fields]
  - test[subtyping relation]
  - ...

Gregoromichlaki, Eleni
DS-TTR: actions from DS +

- parsing/linearising (syntactic/lexical):
  - go [treenode]
  - make[treenode]
  - put[field/value/label/...]
  - IF [value] THEN [actions], ELSE [...]
  - run(list⟨actions⟩)[...]

- manipulating complex type articulation
  - add[fields]
  - remove[fields]
  - test[subtyping relation]
    ...

- exploring the context:
  - freshput[variable/metavariable]
  - find[value/label/...],
  - substitute[values for metavariables]
    ...
incremental construction

[START] → PREDICTION

◇, \(Ty(t)\)
incremental construction

\[ \text{PREDICTION} \]

\[ ?Ty(t) \]

\[ \Diamond, ?Ty(e) \]

\[ ?Ty(e \rightarrow t) \]
incremental construction

\[ \text{John} \rightarrow \]

\[ ?Ty(t) \]

\[ \Diamond, ?Ty(e) \quad ?Ty(e \rightarrow t) \]

\[ \text{John} \]

IF \[ ?Ty(e) \]

THEN \[ \text{put}(Ty(e)) \]
\[ \text{put}([ x=\text{john} : e ]) \]

ELSE \text{abort}
incremental construction

\[ \text{John, ..., POINTER-MOVEMENT} \]

\[ \begin{array}{c}
?Ty(t) \\
\rightarrow \\
Ty(e), \\
[ x=john : e ] \\
\end{array} \]

John

IF \[ ?Ty(e) \]
THEN \[ put(Ty(e)) \]
\[ put([ x=john : e ]) \]
ELSE abort
\[ \text{arrives} \]

\[ Ty(t) \]

\[ Ty(e \rightarrow t), \Diamond \]

\[ Ty(e), \]
\[ [x_{=john} : e] \]

\[ \lambda r : [x : e]. \]

\[ [x_{=r.x} : e, s_{=\text{arrive}} : e_s, p_{=\text{agent}(s,x)} : t, \ldots : \ldots] \]
incremental construction

...[TENSE, ...], COMPLETION

\[\text{Ty}(t), \quad \Diamond, \quad \text{Ty}(e)\]

\[
\begin{align*}
\text{Ty}(e), & \quad \text{Ty}(e \rightarrow t), \\
\begin{bmatrix}
\text{x}\rightarrow\text{john} & : & e \\
\text{s}\rightarrow\text{arrive} & : & e_s \\
\text{p}\rightarrow\text{agent}(s,x) & : & t \\
... & : & ...
\end{bmatrix} & \quad \lambda \quad \text{r : [ x : e ]}.
\end{align*}
\]
underspecification: structural

- Processing **non-contiguous dependencies**
  - e.g. *Mary, John upset*

  \(? Ty(t), \diamond\)
underspecification: structural

- Processing **non-contiguous dependencies**
  - e.g. *Mary, John upset*

\[ x : mary', \diamond \]

Mary

?Ty(t)
underspecification: structural

- Processing **non-contiguous dependencies**
  - e.g. *Mary, John upset*

```
[ x : mary' ]

Mary

? Ty(t)

? Ty(e)

? Ty(e -> t)
```
Processing **non-contiguous dependencies**

- e.g. *Mary, John upset*
underspecification: structural

> Processing **non-contiguous dependencies**
  > e.g. *Mary, John upset*

Mary, John

```
[ x : mary' ]
```

```
Ty(e), [ y : john' ]
```

```
? Ty(t)
```

```
? Ty(e \rightarrow t), ♦
```

Ty(e) → t, ♦
underspecification: structural

- Processing **non-contiguous dependencies**
  - e.g. *Mary, John upset*

Mary, John upset

\[\begin{align*}
\text{Ty}(t) & \quad ? \\
\text{Ty}(e) & \quad [y : john'] \\
\text{Ty}(e \rightarrow t) & \quad ? \\
\text{Ty}(e) & \quad ? \\
\text{[...upset']} & \quad \diamond \\
\text{[x : mary']} & \quad \text{Ty}(e), [y : john']
\end{align*}\]
underspecification: structural

- Processing **non-contiguous dependencies**
  - e.g. *Mary, John upset*

```
Mary, John upset
```

```
[x : mary']
```

```
Ty(e), [y : john']
```

```
?Ty(t)
```

```
Ty(e → t)
```

```
[x : mary']
```

```
Ty(e)
```

```
?Ty(e)
```

```
[...upset']
```

```
UNIFY
```

```
?Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(e → t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(e → t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(e → t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```

```
Ty(e)
```

```
[...upset']
```

```
Ty(t)
```
underspecification: structural

- Processing **non-contiguous dependencies**
  - e.g. *Mary, John upset*

Mary, John upset

```
?Ty(t)

Ty(e), [y : john']

?Ty(e \rightarrow t), ♦

Ty(e), [x : mary']

[...upset']
```
under specification: structural

- Processing **non-contiguous dependencies**
  - e.g. *Mary, John upset*

Mary, John upset

\[ Tn(0), Ty(t), [upset'(mary')(john')], \diamond \]

\[ Ty(e), [y : john'] \]

\[ Ty(e \rightarrow t), [...upset'(mary')] \]

\[ Ty(e), [x : mary'] \]

[...upset']
utterance micro-events

\[\diamond, Ty(t), \begin{bmatrix}
\text{CONTEXT} & u_1 \oplus u_2 \\
\text{CONTENT} & \begin{cases} x : e \\ p : f(x) \end{cases}
\end{bmatrix}\]

\[Ty(e), \begin{bmatrix}
\text{CONTEXT} & u_2 \\
\text{CONTENT} & \begin{cases} x : e \end{cases}
\end{bmatrix}\]

\[Ty(e \rightarrow t), \begin{bmatrix}
\text{CONTEXT} & u_1 \\
\text{CONTENT} & \begin{cases} \lambda [ x : e ] \cdot \begin{cases} x : e \\ p : f(x) \end{cases} \end{cases}
\end{bmatrix}\]
including contextual parameters

John arrived

\[ \diamondsuit, Ty(t), \]

CONTEXT:

\[ a : participantA \]
\[ b : participantB \]
\[ \ldots : \ldots \]
\[ u_1 : utt \rightarrow event \]
\[ s_{s1} : spkr(u_1, a) \]
\[ s_{a1} : addr(u_1, b) \]
\[ u_2 : utt \rightarrow event \]
\[ s_{s2} : spkr(u_2, a) \]
\[ s_{a2} : addr(u_2, b) \]
\[ \ldots : \ldots \]

CONTENT:

\[ x : john \]
\[ p : arrive(x) \]

CONTEXT:

\[ \lambda[x] \cdot [p : arrive(x)] \]
A: John ...
B: arrives ↦→

CONTEXT :
\[ u_1 \oplus 2 \quad : \quad utt \rightarrow event \]
\[ s_1 \quad : \quad spkr(A, u_1) \]
\[ s_2 \quad : \quad spkr(B, u_2) \]
...

CONTENT :
\[ Ty(t), \]
\[ s_{=\text{now}} \quad : \quad e_s \]
\[ x_{=\text{john}} \quad : \quad e \]
\[ p_{=\text{arrive}(s, x)} \quad : \quad t \]

CONTEXT :
\[ u_1 \quad : \quad utt \rightarrow event \]
\[ s_1 \quad : \quad spkr(A, u_1) \]
\[ s_2 \quad : \quad spkr(B, u_2) \]
...

CONTENT :
\[ Ty(e), \]
\[ x_{=\text{john}} \quad : \quad e \]

\[ \lambda r : [ \quad x \quad : \quad e \quad ] . \]
\[ s_{=\text{now}} \quad : \quad e_s \]
\[ x \quad : \quad e \]
\[ p_{=\text{arrive}(s, x)} \quad : \quad t \]
utterance event parameters - indexicals

$I$:

IF $?Ty(e)$, $\text{CONTEXT} : [ s_s : spkr(u, x) ]$
THEN put($Ty(e)$)
put($x$)
ELSE abort

$myself$:

IF $?Ty(e)$, $\text{CONTEXT} : [ s_s : spkr(u, x) ]$, $\uparrow_0 \uparrow_1 * \downarrow_0 Fo(x)$
THEN put($Ty(e)$)
put($Fo(x)$)
ELSE abort
utterance event parameters - indexicals

I:

IF ?Ty(e), [ CONTEXT : [ ss : spkr(u, x) ] ]
THEN put(Ty(e))
   put((x))
ELSE abort

myself:

IF ?Ty(e), [ CONTEXT : [ ss : spkr(u, x) ] ],
   ↑0↑1*↓0 Fo(x)
THEN put(Ty(e))
   put(Fo(x))
ELSE abort

A: Did you burn ...
B: myself?
split utterances with indexicals

Eleni: I burnt . . .
Bill: yourself! (as usual)
Eleni: I

\[ \text{CXT} : [ s_1 : \text{spk}(Eleni, u_1) ] \]
Eleni: I

IF

?Ty(e), [ CONTEXT : [ s_s : spkr(u, x) ] ]
THEN
put(Ty(e))
put((x))
ELSE
abort
Eleni: *I burnt* ...
split utterances with indexicals

Eleni: I burnt ...
Eleni: I burnt . . .
Bill: yourself! (as usual)

IF \( ?Ty(e), [ \text{CONTEXT} : [s_s : \text{addr}(u, x)] \] \),
\[\uparrow_{0} \uparrow_{1} \uparrow_{0} \downarrow_{0} \text{Fo}(x) \]
THEN
put(\(Ty(e))\)
put(\(Fo(x))\)
ELSE
abort

\[
\begin{array}{l}
\text{CXT:}\ [s_1 : spk(Eleni, u_1)] \\
\text{CNT:}\ [x=Eleni : e]
\end{array}
\]

\[
\begin{array}{l}
\text{CXT:}\ [s_3 : spk(Bill, u_3)] \\
\text{CNT:}\ [s_4 : \text{addr}(Eleni, u_3)]
\end{array}
\]

\[
\begin{array}{l}
\text{CXT:}\ [s_2 : spk(Eleni, u_2)] \\
\text{CNT:}\ \lambda [x : e, y : e, p : \text{burn'}(y, x)]
\end{array}
\]
split utterances with indexicals

Eleni: I burnt ...
Bill: yourself! (as usual)

IF ?Ty(e), [ CONTEXT : [ s_s : addr(u, x) ] ], ↑0↑1*↓0 Fo(x)
put(Ty(e))
put(Fo(x))
ELSE abort

\[
\begin{align*}
\text{CXT: } & \left[ s_1 : spk(Eleni, u_1) \right] \\
\text{CNT: } & \left[ x=Eleni : e \right]
\end{align*}
\]

\[
\begin{align*}
\text{CXT: } & \left[ s_2 : spk(Eleni, u_2) \right] \\
\text{CNT: } & \lambda \left[ x : e, \ y : e \right]. \left[ x : e, \ y : e, \ p : burn'(y, x) \right]
\end{align*}
\]

\[
\begin{align*}
\text{CXT: } & \left[ s_3 : spk(Bill, u_3) \right] \\
\text{CNT: } & \left[ s_4 : addr(Eleni, u_3) \right] \\
\text{CNT: } & \left[ y=Eleni : e \right]
\end{align*}
\]
split utterances with indexicals

Eleni: I burnt . . .
Bill: yourself! (as usual)
Eleni: I burnt . . .
Bill: yourself! (as usual)

\[Ty(t),\]

\[
\text{CONTEXT: } \begin{bmatrix}
...,
U_0 : u_1 \oplus u_2 \oplus u_3...
\end{bmatrix}
\]

\[
\text{CONTENT: } \begin{bmatrix}
x_{=Eleni} : e
y_{=Eleni} : e
p : burn(x, y)
\end{bmatrix}
\]

\[
\text{CXT: } \begin{bmatrix}
s_1 : \text{spk}(Eleni, u_1)
\end{bmatrix}
\]

\[
\text{CNT: } \begin{bmatrix}
x_{=Eleni} : e
\end{bmatrix}
\]

\[
\text{CXT: } \begin{bmatrix}
U_1 : u_2 \oplus u_3
\end{bmatrix}
\]

\[
\text{CNT: } \lambda \begin{bmatrix}
x : e
y : A
p : burn(y, x)
\end{bmatrix}
\]

\[
\text{CXT: } \begin{bmatrix}
s_3 : \text{spk}(Bill, u_3)
s_4 : addr(Eleni, u_3)
\end{bmatrix}
\]

\[
\text{CNT: } \begin{bmatrix}
y_{=Eleni} : e
\end{bmatrix}
\]

\[
\text{CXT: } \begin{bmatrix}
s_2 : \text{spk}(Eleni, u_2)
\end{bmatrix}
\]

\[
\text{CNT: } \lambda \begin{bmatrix}
x : e
y : e
p : burn'(y, x)
\end{bmatrix}
\]
further dialogue phenomena

- **self-repair**
  
  A: Peter went swimming with Susan, um, or rather, surfing, yesterday. ['Peter went surfing with Susan yesterday']

- **other-repair, clarification (echoing)**
  
  A: Peter went swimming with Susan
  
  B: Susan?
Parse/Generation States

\[ \text{Parse/GenIU} = \begin{bmatrix}
\text{words} & : & \text{list}(\text{Words}) \\
\text{actions} & : & \text{list}(\text{Actions}) \\
\text{tree} & : & \text{PointedTree} \\
\text{totalctxt} & : & \text{list}(\text{Tree}) \\
\text{cnt} & : & \text{RT} \\
\text{localctxt} & : & \text{RT}
\end{bmatrix} \]
actions (edges) are transitions between partial trees (nodes)
processing paths probabilistically ranked
DS and TTR issues

- DS features to be maintained:
  - action-based syntax
  - no syntactic representation - grammaticality as constraints on update of semantic structures
  - incremental semantics
  - unified view of anaphora, ellipsis (quotation)
  - treat continuations as continuations
  - speech acts as system updates
DS and TTR issues to be resolved

**DS-TTR: problems of integration**

  - can dispense with *LINK* but island restrictions?
- modality and propositional attitudes: possible worlds vs propositions as types
- monotonicity: multiple parsing paths – predictivity???
- dialogue moves: inferred, represented, encoded, default
Outline

Introductory Motivation
What is grammar?

TTR to formalise conceptual structure
TTR elements adopted
Dynamic Syntax
DS elements adopted
Dynamic Syntax (DS)

DS-TTR

Hannes Rieser’s Questions

General conclusions
DS-TTR and cognition - abandoning competence vs performance

Appendix 1
Appendix 2
What was the original motivation for combining DS and TTR and what was the gain to be expected?

More specifically: What are the concrete interface points for DS and TTR integration?

Update, action and context figure prominently in DS as well as in TTR. Are the notions implied similar and, if so, in which respects?

How are DS tree construction and the build-up of record types related in DS-TTR? It seems that if we use DSs lexical actions as the main integration point of DS and TTR and, consequently, put record types into them, we get in principle two “up-ward working” compositional processes, one for the conceptual structure of DS and the other one for the record type construction. Is this impression wrong?

In more detail: Assume for the sake of discussion that both representations get their own semantics, however expressed, then we would have two different semantic values encoded in one DS-TTR-representation. Again, is this impression wrong?
DS, as I see it, is incremental due to the unfixed-node conventions and the representation of the main verb waiting for input. Are there comparable mechanisms in TTR? Does TTR have different ones from those?

Reconciliation of DS quantifier theory using the epsilon calculus and the Generalized Quantifier approach taken in TTR will require major changes in either the one or the other paradigm, right?
If we look at TTR we see that it is **pragmatics and dialogue based** tout court (see the modelling of turn-exchange using dialogue game boards), plan-based (see the notion of agenda) and relies heavily on **mental sates** (see the labels ”private” and “shared” in the information states).

In contrast, DS relies on **interaction via grammar** defined on LOFT and avoids use of mental states. Does this fact impose a limit on the integratability of DS-TTR?
I see a sort of **division of labour** between DS and TTR in the following way:

- DS can, due to its generation-and-parsing facility cope with, e.g. types of ellipses, split utterances, self- or other-repairs, and across-sentence-clitics.
- TTR can reconstruct dialogue interaction in a very fine-grained way using different types of modal notions.
Both are motivated in different ways and favour different domains of application.

A good deal of action and interaction in dialogue seems to be automatic, take e.g. alignment, hesitation phenomena, mid-turn acknowledgements, repair indicators and similar things. They are not intentional in the sense of “to be reconstructed with an intention operator defined on propositional content”. Hence, this seems to be the “natural ‘mechanistic’ domain” of DS.

On the contrary, modal notion based concepts seem to have their natural site in TTR. It may of course be controversial which phenomena are to be reconstructed using which technology. Is this an acceptable way to fix the divide between DS and TTR?
One gets the idea that **TTR is more directed towards philosophy** (theory of perception and action, allusions to Aristotle, Kant and Russell, semantic puzzles, theory of proper names, reflecting the Montague-Partee-tradition) **DS more towards linguistics** (considering a wealth of natural languages, treating fine-grained data, e.g. morphology). So there is a division of labour in this sense as well. Right or wrong?
Outline

Introductory Motivation
  What is grammar?

TTR to formalise conceptual structure
  TTR elements adopted
  Dynamic Syntax
  DS elements adopted
  Dynamic Syntax (DS)

DS-TTR

Hannes Rieser’s Questions

General conclusions
  DS-TTR and cognition - abandoning competence vs performance

Appendix 1

Appendix 2
NL conceptual representations not domain-specific, common to action/perception

syntax, lexicon = set of actions (affordances) that predict, induce, develop structured contexts

coordinated action (e.g. conversation) relies on:
  - action-oriented predictive simulative processing
  - non-conceptual procedural mechanisms (not high-order inference)

⇒ interaction/coordination is an effect achievable directly from grammar-defined procedures, i.e. from low-level non-conceptual mechanisms

Thanks!

and thanks to:

Outline

Introductory Motivation
  What is grammar?

TTR to formalise conceptual structure
  TTR elements adopted
  Dynamic Syntax
  DS elements adopted
  Dynamic Syntax (DS)

DS-TTR

Hannes Rieser’s Questions

General conclusions
  DS-TTR and cognition - abandoning competence vs performance

Appendix 1

Appendix 2
quotational puns:

(15) The menu says that this restaurant serves “breakfast at any time” so I ordered French toast during the Renaissance. [Steven Wright joke]

(16) ‘Marriage’ is not a word, it’s a sentence.

⇒ the grammar needs to be able to keep track of abandoned parsing paths as well as current viable ones.
actions (edges) are transitions between partial trees (nodes)
processing paths probabilistically ranked
Outline

Introductory Motivation
  What is grammar?

TTR to formalise conceptual structure
  TTR elements adopted
  Dynamic Syntax
  DS elements adopted
  Dynamic Syntax (DS)

DS-TTR

Hannes Rieser’s Questions

General conclusions
  DS-TTR and cognition - abandoning competence vs performance

Appendix 1

Appendix 2