Proposals to generalise Dynamic Syntax for wider application

Julian Hough,[1] Arash Eshghi,[2]
Matthew Purver[1] and Graham White[1]

[1]Queen Mary University of London
[2]Heriot-Watt University

2nd DS Conference, April 2018

The problem: academic sociology

- DS is fantastic!
- We deserve more recognition.

The problem: academic sociology

- DS is fantastic!
- We deserve more recognition.
- We model interactive phenomena like no other ‘grammar formalism’.
- We’d like more users!

The problem: academic sociology

- DS is fantastic!
- We deserve more recognition.
- We model interactive phenomena like no other ‘grammar formalism’.
- We’d like more users!
- 5 proposals!

1. Make it clear what DS is all about: Dynamics
2. Generalize the composition calculus: lambda
3. Liberalise permissible node types
4. Liberalise permissible semantic representation languages
5. Use it!

1. Make it clear what DS is all about: Dynamics
2. Generalize the composition calculus: lambda
3. Liberalise permissible node types
4. Liberalise permissible semantic representation languages
5. Use it!

Dynamic Syntax (DS)

- DS is primarily about how a representation is built up over time, with at least a word-by-word granularity, by natural language utterances.
- DS grammar encodes the word-by-word incremental growth of semantic representations directly as *tree building actions*.
- No independent layer of syntactic processing.
- Grammaticality is defined in terms of *left-right parseability*.

Dynamic Syntax (DS)

- *Monotonic* connected tree building- good for dialogue inference.
- DS is bidirectional, i.e. generation is parasitic on parsing. *Self-monitoring* comes for free.
- *Parsing actions* (lexical and computational actions) are first class citizens of the grammar.

DS-TTR

- Recent DS variant uses TTR *record types* on the trees [Purver et al., 2011].
- Record type compilation for *partial trees* [Hough, 2011] allows *strong incremental interpretation* [Milward, 1991].
- Incrementally constructed structures can be compared to domain concepts and generation goals in word-by-word *subtype* relation checking. [Hough, 2011]

DS vs DS-TTR vs DS-X

- Let's focus on the similarities between DS proper, DS-TTR, DS-Tensor etc.
- Let's look at the *parsing dynamics*.

DS vs DS-TTR vs DS-X

- Let's focus on the similarities between DS proper, DS-TTR, DS-Tensor etc.
- Let's look at the *parsing dynamics*.
- Spot the difference...

Parsing *Ruth arrives*:

Incremental DS parsing

Parsing *Ruth arrives*:

$$\diamondsuit ?Ty(t),$$

Incremental DS parsing

Parsing *Ruth arrives*:

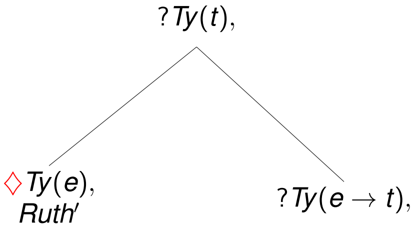
$$?Ty(t),$$

$$\Diamond ?Ty(e), \qquad ?Ty(e \rightarrow t),$$

Incremental DS parsing

Parsing *Ruth arrives*:

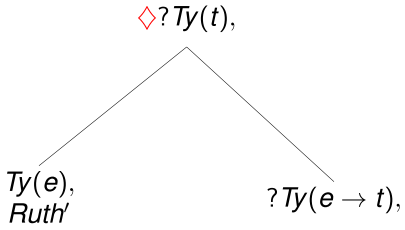
Ruth



Incremental DS parsing

Parsing *Ruth arrives*:

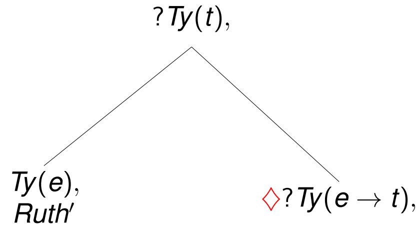
Ruth



Incremental DS parsing

Parsing *Ruth arrives*:

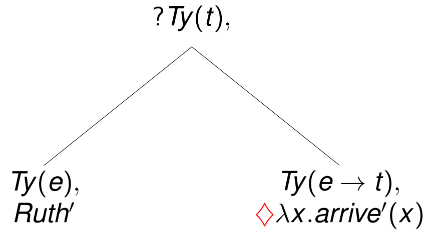
Ruth



Incremental DS parsing

Parsing *Ruth arrives*:

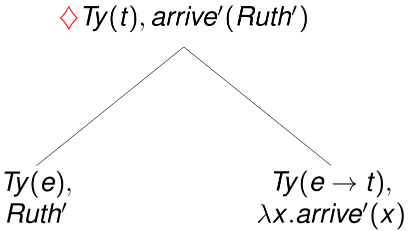
Ruth arrives



Incremental DS parsing

Parsing *Ruth arrives*:

Ruth arrives



Incremental DS-TTR parsing

Parsing *Robin arrives*:

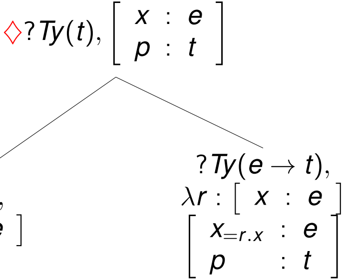
Incremental DS-TTR parsing

Parsing *Robin arrives*:

$$\color{red}{\Diamond}\, ?Ty(t), \left[\ p\ :\ t\ \right]$$

Parsing *Robin arrives*:

Robin



Parsing *Robin arrives*:

Robin

$$?Ty(t), \begin{bmatrix} x & : & e \\ p & : & t \end{bmatrix}$$

$$\diamondsuit Ty(e), \begin{bmatrix} x & : & e \end{bmatrix}$$

$$?Ty(e \rightarrow t), \lambda r : \begin{bmatrix} x & : & e \end{bmatrix} \begin{bmatrix} x_{=r.x} & : & e \\ p & : & t \end{bmatrix}$$

Incremental DS-TTR parsing

Parsing *Robin arrives*:

Robin

$$?Ty(t), \begin{bmatrix} x_{=robin} & : & e \\ p & : & t \end{bmatrix}$$

$$\Diamond Ty(e), \begin{bmatrix} x =_{robin} & : & e \end{bmatrix}$$

$$?Ty(e \rightarrow t), \lambda r : \begin{bmatrix} x & : & e \end{bmatrix} \begin{bmatrix} x_{=r.x} & : & e \\ p & : & t \end{bmatrix}$$

Incremental DS-TTR parsing

Parsing *Robin arrives*:

Robin

$$?Ty(t), \begin{bmatrix} x_{=robin} & : & e \\ p & : & t \end{bmatrix}$$

$$Ty(e), \begin{bmatrix} x =_{robin} & : & e \end{bmatrix}$$

$$\Diamond ?Ty(e \rightarrow t), \lambda r : \begin{bmatrix} x & : & e \end{bmatrix} \begin{bmatrix} x_{=r.x} & : & e \\ p & : & t \end{bmatrix}$$

Incremental DS-TTR parsing

Parsing *Robin arrives*:

Robin arrives

$$\diamondsuit Ty(t), \left[\begin{array}{lll} x_{=robin} & : & e \\ p_{=arrive(x)} & : & t \end{array}\right]$$

$$Ty(e), \left[\begin{array}{lll} x =_{robin} & : & e \end{array}\right]$$

$$Ty(e \rightarrow t), \lambda r : \left[\begin{array}{lll} x & : & e \end{array}\right] \left[\begin{array}{lll} x_{=r.x} & : & e \\ p_{=arrive(x)} & : & t \end{array}\right]$$

Incremental DS-TTR generation

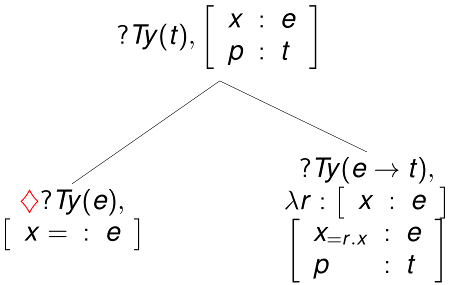
Generating *Robin arrives*:

Generating *Robin arrives*:

GOAL :

$$\begin{bmatrix} x_{=robin} & : & e \\ p_{=arrive(x)} & : & t \end{bmatrix}$$

SUBTYPE



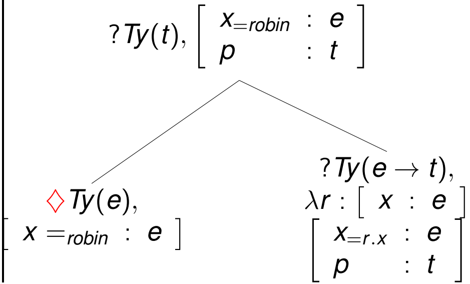
Generating *Robin arrives*:

Robin

GOAL :

$$\begin{bmatrix} x_{=robin} & : & e \\ p_{=arrive(x)} & : & t \end{bmatrix}$$

SUBTYPE



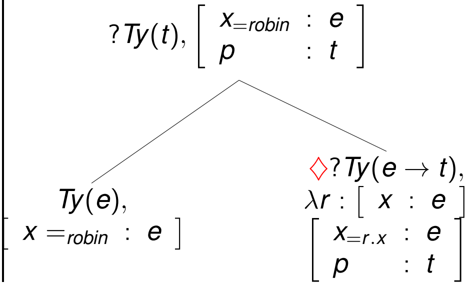
Generating *Robin arrives*:

Robin

GOAL :

$$\begin{bmatrix} x_{=robin} & : & e \\ p_{=arrive(x)} & : & t \end{bmatrix}$$

SUBTYPE



Incremental DS-TTR generation

Generating *Robin arrives*:

Robin arrives

GOAL :

$$\begin{bmatrix} x_{=robin} & : & e \\ p_{=arrive(x)} & : & t \end{bmatrix}$$

MATCHES!

$$\diamondsuit\, Ty(t), \begin{bmatrix} x_{=robin} & : & e \\ p_{=arrive(x)} & : & t \end{bmatrix}$$

$$Ty(e), \begin{bmatrix} x =_{robin} & : & e \end{bmatrix}$$

$$Ty(e \rightarrow t), \lambda r : \begin{bmatrix} x & : & e \end{bmatrix} \begin{bmatrix} x_{=r.x} & : & e \\ p_{=arrive(x)} & : & t \end{bmatrix}$$

DS lexical actions

Michael: ‘Did you burn’

Ruth: ‘myself?’

myself:

IF	$?Ty(e)$
THEN	$\mathtt{put}(Ty(e))$,
	$\mathtt{put}(Ruth')$
ELSE	$\mathtt{abort}$

DS-TTR lexical actions

- Context dependent values can be formally defined now in DS lexical actions [Purver et al., 2010]

myself:

$$
\begin{array}{ll}
\text{IF} & ?Ty(e), r : \left[\begin{array}{lll} ctxt & : & \left[\begin{array}{lll} u & : & utt \\ x & : & e \\ p_{=spkr(u,x)} & : & t \end{array} \right] \end{array} \right], \\
 & \uparrow_0\uparrow_{1*}\downarrow_0 r1 : \left[\begin{array}{lll} cont & : & \left[\begin{array}{lll} x1_{=r.ctxt.x} & : & e \end{array} \right] \end{array} \right] \\
\text{THEN} & \texttt{put}(Ty(e)), \\
 & \texttt{put}(r \boxed{\curlywedge} \left[\begin{array}{lll} cont & : & \left[\begin{array}{lll} x_{=r.ctxt.x} & : & e \end{array} \right] \end{array} \right]) \\
\text{ELSE} & \texttt{abort}
\end{array}
$$

- Use of dependent record types. Use of paths.

What is important for DS?

- Is the difference between the representation language on the nodes important?

What is important for DS?

- Is the difference between the representation language on the nodes important?
- It depends what you do with the representation language– nothing in the representation per se matters.

What is important for DS?

- Is the difference between the representation language on the nodes important?
- It depends what you do with the representation language– nothing in the representation per se matters.
- Perhaps time to get back to the original:

"The emphasis is on the process of establishing some structure as interpretation, rather than just specifying the RESULT, which is the structure itself." [Kempson et al., 2001]

What is important for DS?

- If the result is not the object is of interest, then what is?

What is important for DS?

- If the result is not the object is of interest, then what is?
- *How* we get there, word-by-word.

What is important for DS?

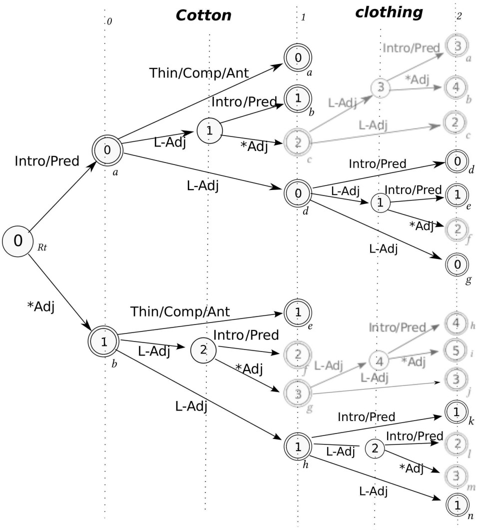
- If the result is not the object is of interest, then what is?
- *How* we get there, word-by-word.
- Processing context characterized as an *action graph*.

What is important for DS?

- If the result is not the object is of interest, then what is?
- *How* we get there, word-by-word.
- Processing context characterized as an *action graph*.
- Inspired by the notion of context as a triple $< T, W, A >$, [Sato, 2011] showed how this could be a Directed Acyclic Graph (DAG) with search.
- Models garden-path sentence processing. ‘Cotton clothing is made of grows in Mississippi’

Action graphs for dynamics

[Sato, 2011]



Examples:

2a. Cotton clothing is made of grows in Mississippi.

2b. Cotton clothing shops buy is made of grows in Mississippi.

2h. Cotton clothing is made of the farmer grows in Mississipi.

2i. Cotton clothing shops buy is made of grows in Mississippi.

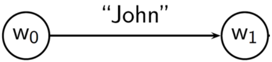
Action graphs for dynamics

- [Purver et al., 2011] defined DyLan which modelled the process as two graphs
- Input *word graph* and the more fine-grained *action graph* grounded in the word graph.
- *Concept graph* [Hough, 2015]

DyLan parsing

John

WORD GRAPH
(INPUT)

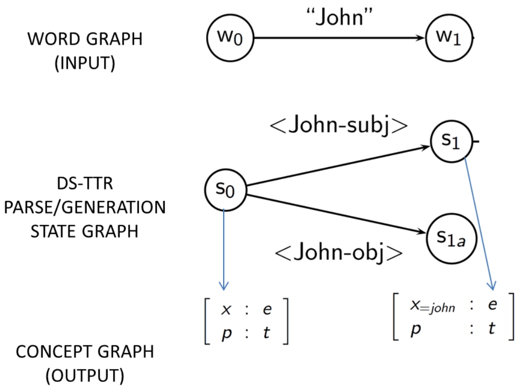


DS-TTR
PARSE/GENERATION
STATE GRAPH

CONCEPT GRAPH
(OUTPUT)

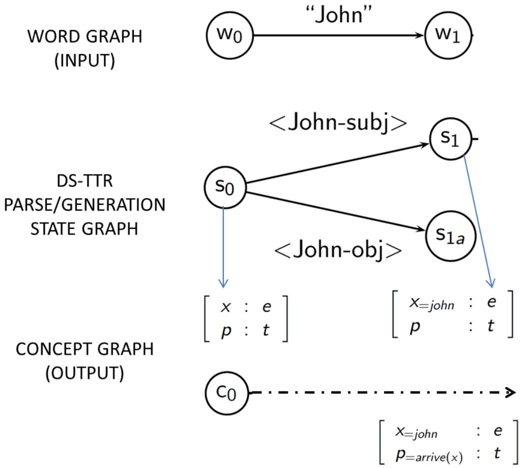
DyLan parsing

John



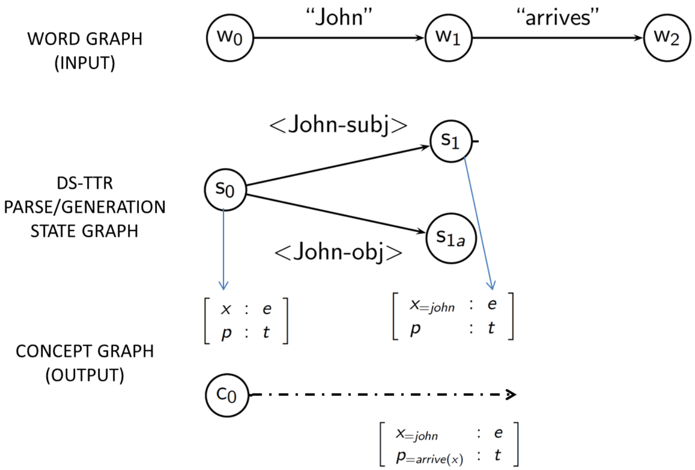
DyLan parsing

John



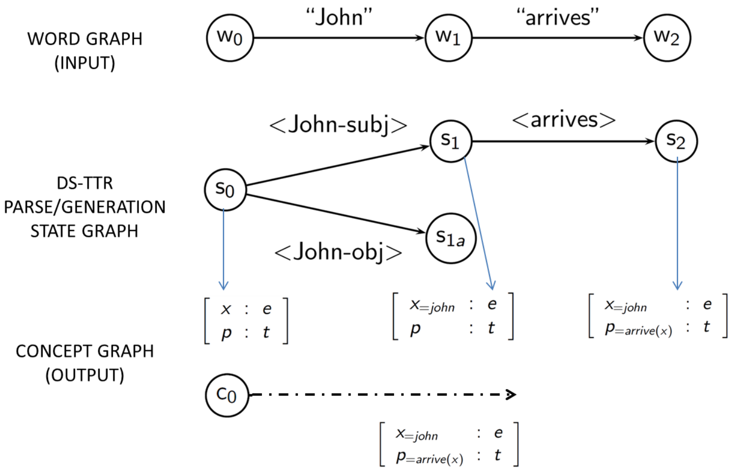
DyLan parsing

John arrives



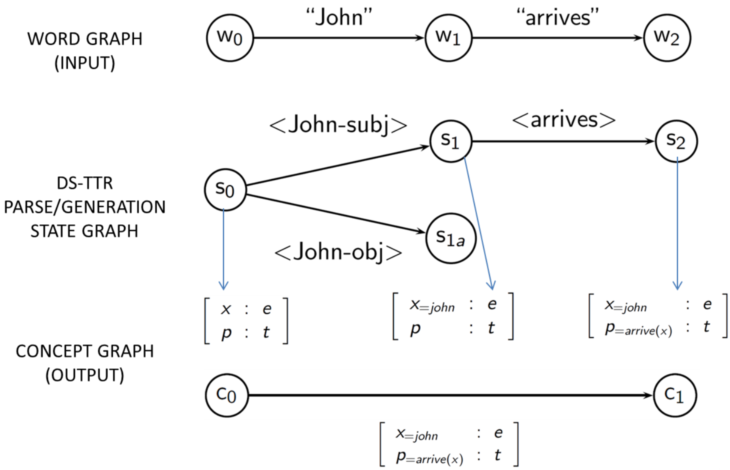
DyLan parsing

John arrives

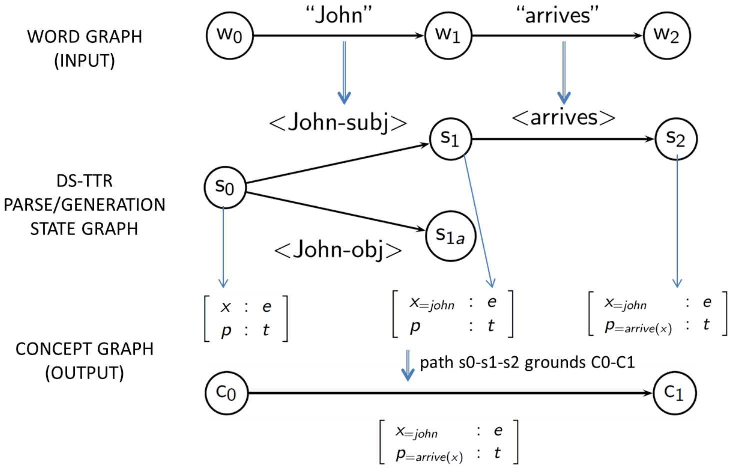


DyLan parsing

John arrives



John arrives



Phenomena DS/DyLan can cover

- [Hough, 2011] modelled *self-repair* in DyLan in terms of backwards-search and re-constructing the right-frontier of the word graph.

Phenomena DS/DyLan can cover

- [Hough, 2011] modelled *self-repair* in DyLan in terms of backwards-search and re-constructing the right-frontier of the word graph.
- [Purver et al., 2014] on compound *contributions/split utterances*.

Phenomena DS/DyLan can cover

- [Hough, 2011] modelled *self-repair* in DyLan in terms of backwards-search and re-constructing the right-frontier of the word graph.
- [Purver et al., 2014] on compound *contributions/split utterances*.
- [Eshghi et al., 2015] modelled *clarification* interaction and *other-repair*, and *backchannels* on this graph but with two graph pointers.

Phenomena DS/DyLan can cover

- [Hough, 2011] modelled *self-repair* in DyLan in terms of backwards-search and re-constructing the right-frontier of the word graph.
- [Purver et al., 2014] on compound *contributions/split utterances*.
- [Eshghi et al., 2015] modelled *clarification* interaction and *other-repair*, and *backchannels* on this graph but with two graph pointers.
- [Kempson et al., 2018] model ellipsis by re-running (copying edges) from the action graph.

Interpreting disfluencies incrementally

- Given we have a graph with a counter *n* as an ID for the last node added, a pointed node *current*, and incoming word *W*:

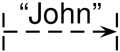
ParseWithSelfRepair(*W*):

IF parse(*W*) from *current* successful
THEN

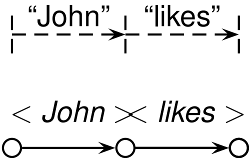
`add` a new edge with new sink node S_n
current := S_n

ELSE:

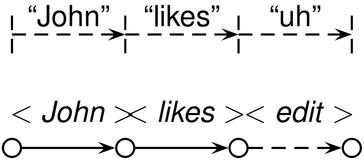
current := *parent*(*current*)
ParseWithSelfRepair(*W*)



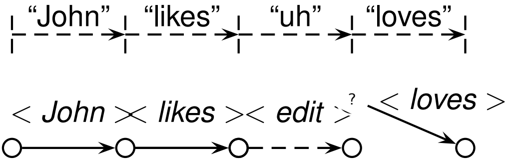
$$
\left[\begin{array}{lcl}
\text{cont} & = & \left[\begin{array}{lcl}
x1 & : & e \\
x_{=John} & : & e \\
e & : & e_s \\
p_{=subj(e,x)} & : & t
\end{array}\right] \\
\text{ctxt} & = & [Assert(User, \text{cont})]
\end{array}\right]
$$



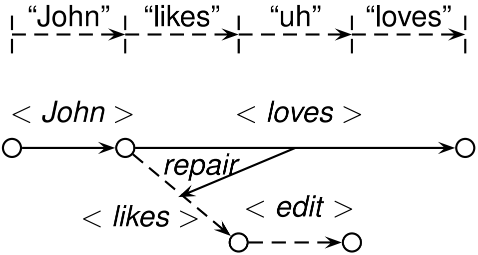
$$\left[\begin{array}{lcl} & & \left[\begin{array}{lcl} x1 & : & e \\ x_{=John} & : & e \\ e_{=likes} & : & e_s \\ p1_{=obj(e,x1)} & : & t \\ p_{=subj(e,x)} & : & t \end{array}\right] \\ \text{cont} & = & \\ \text{ctxt} & = & [\mathit{Assert}(\mathit{User}, \text{cont})] \end{array}\right]$$



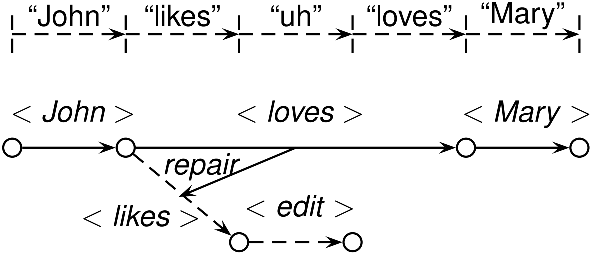
$$\left[\begin{array}{lll} \text{cont} & = & \left[\begin{array}{lll} x1 & : & e \\ x_{=John} & : & e \\ e_{=likes} & : & e_s \\ p1_{=obj(e,x1)} & : & t \\ p_{=subj(e,x)} & : & t \end{array}\right] \\ \text{ctxt} & = & \begin{array}{l} [Assert(User, \text{cont}), \\ FwdProblem(User, \text{cont})] \end{array} \end{array}\right]$$



$$\left[\begin{array}{lcl} \text{cont} & = & \left[\begin{array}{lcl} x1 & : & e \\ x_{=John} & : & e \\ e_{=likes} & : & e_s \\ p1_{=obj(e,x1)} & : & t \\ p_{=subj(e,x)} & : & t \end{array}\right] \\ \text{ctxt} & = & \begin{array}{l}[\mathit{Assert}(\mathit{User}, \text{cont}), \\ \mathit{FwdProblem}(\mathit{User}, \text{cont})]\end{array} \end{array}\right]$$

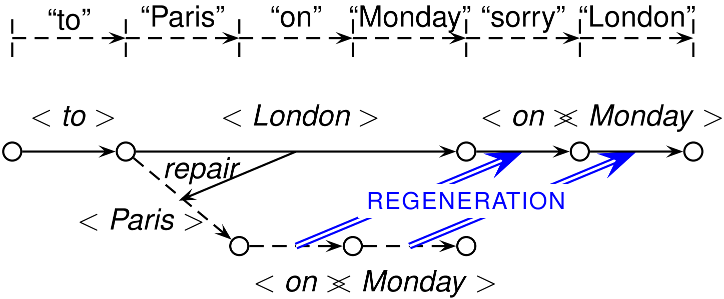


$$\left[\begin{array}{lcl} \text{cont} & = & \left[\begin{array}{lcl} x1 & : & e \\ x_{=John} & : & e \\ e_{=loves} & : & e_s \\ p_{=obj(e,x1)} & : & t \\ p_{=subj(e,x)} & : & t \end{array}\right] \\ \text{ctxt} & = & \begin{array}{l} [Assert(User,\text{cont}), \\ Revoke(User,[\, e_{=likes} : e_s \,] \\ \wedge\neg[\, e_{=loves} : e_s \,])] \end{array} \end{array}\right]$$

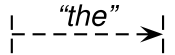


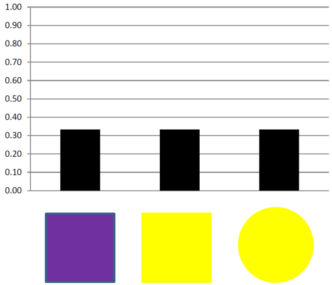
$$\left[\begin{array}{lcl} \text{cont} & = & \left[\begin{array}{lcl} x1_{=Mary} & : & e \\ x_{=John} & : & e \\ e_{=loves} & : & e_s \\ p_{=obj(e,x1)} & : & t \\ p_{=subj(e,x)} & : & t \end{array}\right] \\ \text{ctxt} & = & \begin{array}{l} [Assert(User,\text{cont}), \\ Revoke(User,[\, e_{=likes} : e_s \,] \\ \wedge\neg[\, e_{=loves} : e_s \,])] \end{array} \end{array}\right]$$

Ellipsis in repair though action re-running

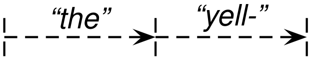


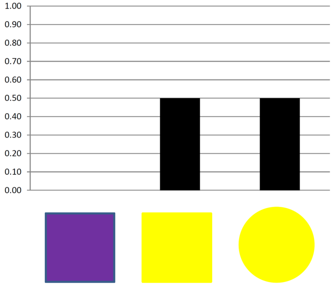
Interpreting disfluencies incrementally



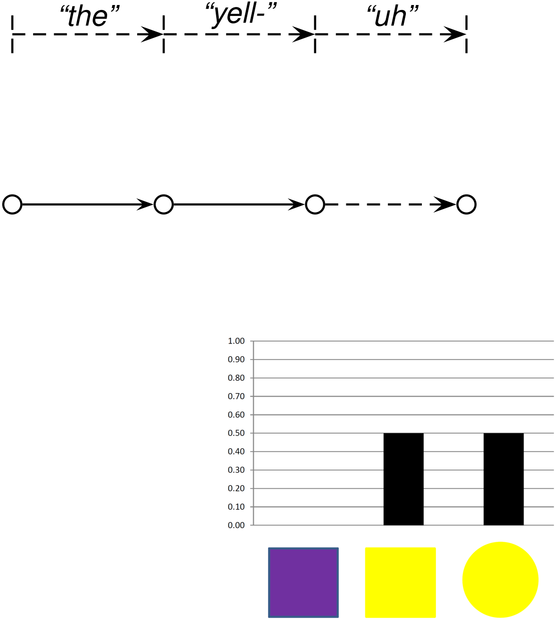


Interpreting disfluencies incrementally

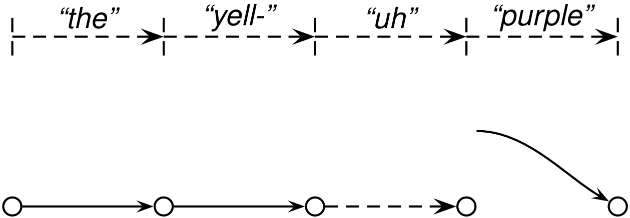


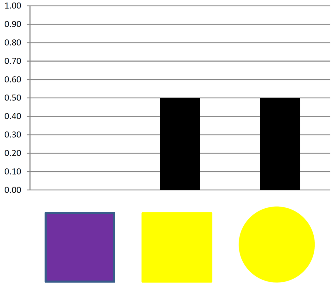


Interpreting disfluencies incrementally

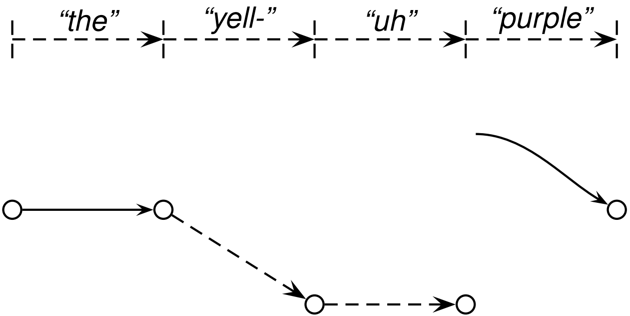


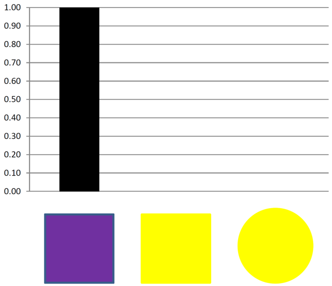
Interpreting disfluencies incrementally



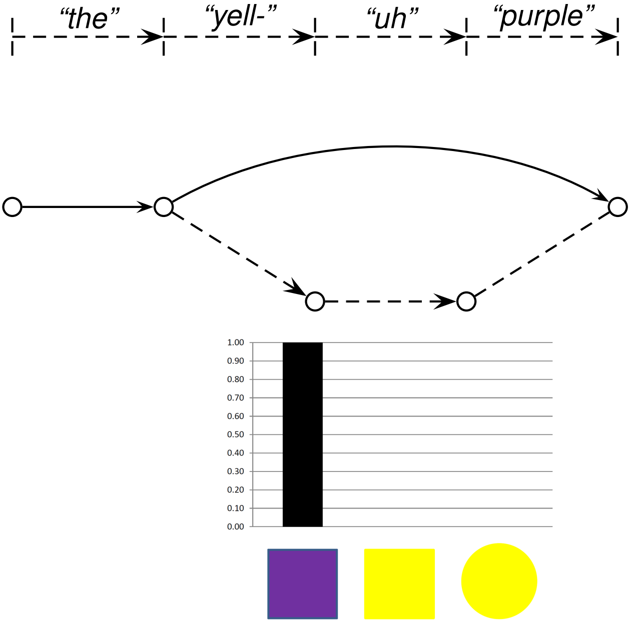


Interpreting disfluencies incrementally

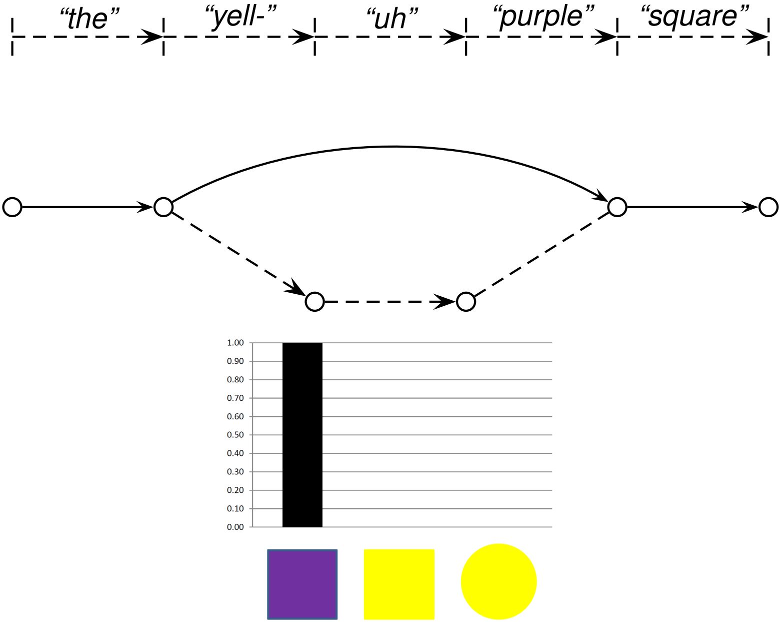




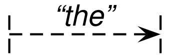
Interpreting disfluencies incrementally

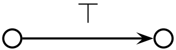


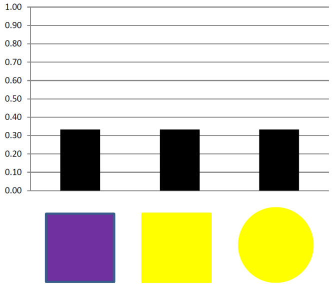
Interpreting disfluencies incrementally



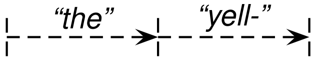
Interpreting disfluencies incrementally

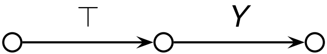


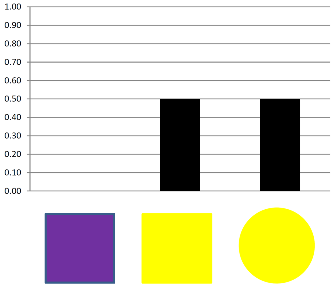




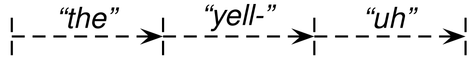
Interpreting disfluencies incrementally

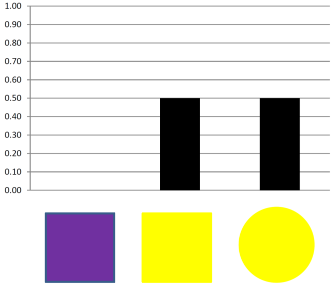




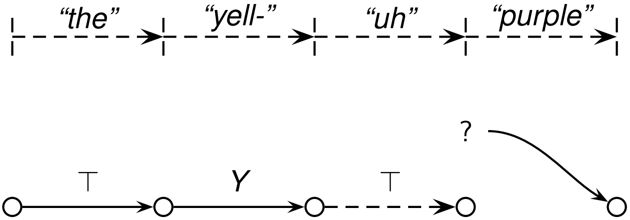


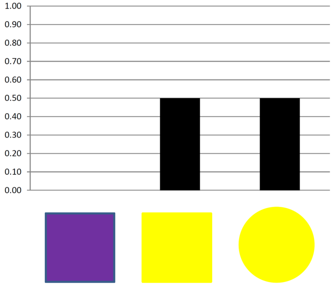
Interpreting disfluencies incrementally



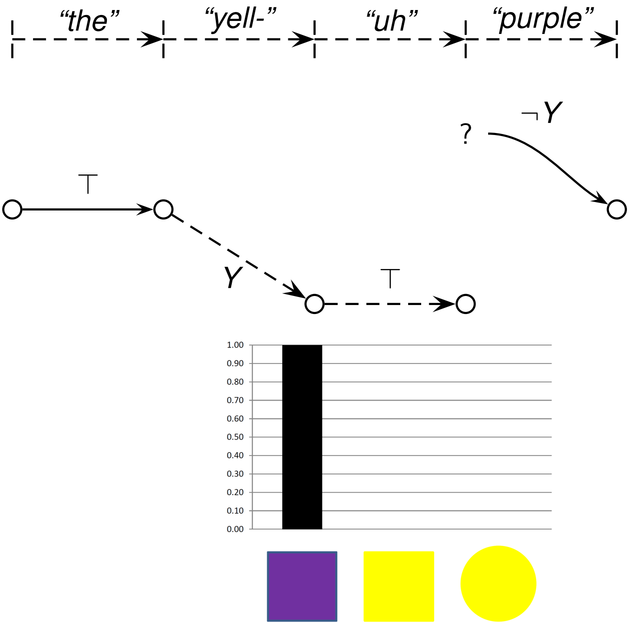


Interpreting disfluencies incrementally

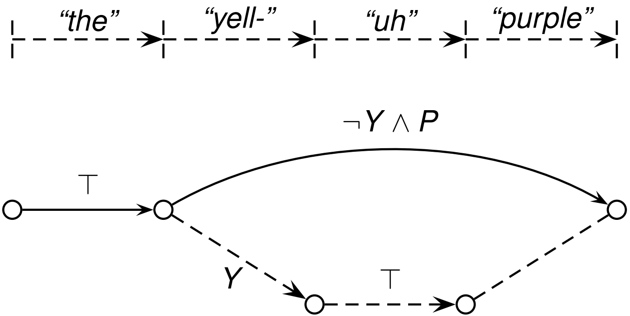


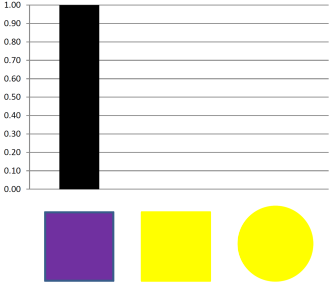


Interpreting disfluencies incrementally

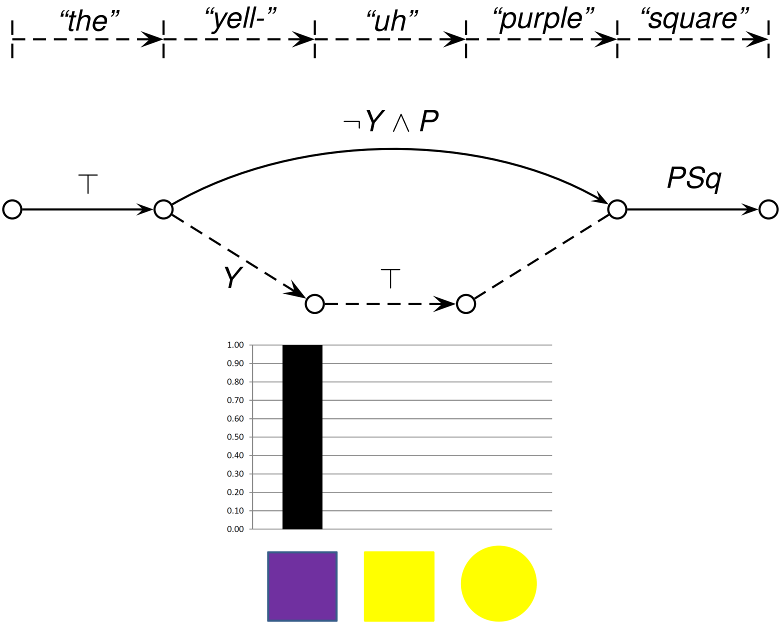


Interpreting disfluencies incrementally





Interpreting disfluencies incrementally



What is important for DS?

- Actions- lexical and computational, the words that triggered them, and their graphs.

What is important for DS?

- Actions- lexical and computational, the words that triggered them, and their graphs.
- Typed trees with under-specification through requirements.
- Functional application and variable renaming in application (β-reduction, α-conversion.)
- The pointer $\diamondsuit$.
- Subsumption $\sqsubseteq$.

1. Make it clear what DS is all about: Dynamics
2. Generalize the composition calculus: lambda
3. Liberalise permissible node types
4. Liberalise permissible semantic representation languages
5. Use it!

What should we compose with?

- λ-calculus is fairly general.
- Functional application through β-reduction a general.
- Variable re-naming with α-conversion gives more flexibility.

What should we compose with?

- λ-calculus is fairly general.
- Functional application through β-reduction a general.
- Variable re-naming with α-conversion gives more flexibility.

- Do we need the ϵ-calculus?
- In DS-TTR we don't really need it as we restrict terms within record types.

1. Make it clear what DS is all about: Dynamics
2. Generalize the composition calculus: lambda
3. **Liberalise permissible node types**
4. Liberalise permissible semantic representation languages
5. Use it!

Go beyond...

- The usual suspects: e, t, cn, e_s

Go beyond...

- The usual suspects: e, t, cn, e_s
- To all and any types.

Go beyond...

- The usual suspects: e, t, cn, e_s
- To all and any types.
- *RecordType*, *Tensor*, *Integer*, *Python* program, *banana* etc.

Go beyond...

- The usual suspects: e, t, cn, e_s
- To all and any types.
- *RecordType*, *Tensor*, *Integer*, *Python* program, *banana* etc.
- In DS (standard) we are building propositions (type t). In DS-TTR we are building record types (not really type t!).
- We should try to be consistent with our typing.

Have we been getting node types right?

Parsing *Robin arrives*:

Robin arrives

$$\diamondsuit\, Ty(t), \begin{bmatrix} x_{=robin} & : & e \\ p_{=arrive(x)} & : & t \end{bmatrix}$$

$$Ty(e), \\ \begin{bmatrix} x =_{robin} & : & e \end{bmatrix}$$

$$Ty(e \rightarrow t), \\ \lambda r : \begin{bmatrix} x & : & e \end{bmatrix} \\ \begin{bmatrix} x_{=r.x} & : & e \\ p_{=arrive(x)} & : & t \end{bmatrix}$$

Have we been getting node types right?

Parsing *Robin arrives*:

Robin arrives

$$\Diamond\, Ty(RecordType), \left[\begin{array}{lcl} x_{=robin} & : & e \\ p_{=arrive(x)} & : & t \end{array}\right]$$

$$Ty(RecordType), \left[\begin{array}{lcl} x =_{robin} & : & e \end{array}\right]$$

$$Ty(RecordType \rightarrow RecordType), \lambda r : \left[\begin{array}{lcl} x & : & e \end{array}\right] \left[\begin{array}{lcl} x_{=r.x} & : & e \\ p_{=arrive(x)} & : & t \end{array}\right]$$

1. Make it clear what DS is all about: Dynamics
2. Generalize the composition calculus: lambda
3. Liberalise permissible node types
4. Liberalise permissible semantic representation languages
5. Use it!

Go beyond...

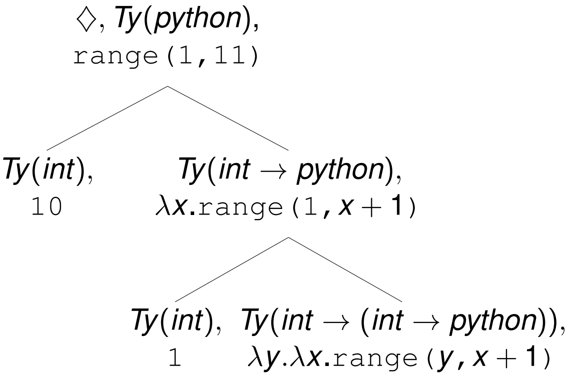
- We've already got:
  - DS-ϵ
  - DS-FOL
  - DS-TTR
  - DS-Tensor

Go beyond...

- We've already got:
  - DS-ϵ
  - DS-FOL
  - DS-TTR
  - DS-Tensor
- Why not DS-Python, DS-G-code, DS-etc.?

'ten'	IF	$?Ty(int)$
	THEN	$\texttt{put}(Ty(int))$
		$\texttt{put}(10)$
	ELSE	$\texttt{abort}$

DS-Python


$\Diamond$, $Ty(python)$,
range(1,11)
$Ty(int)$,
10
$Ty(int \rightarrow python)$,
$\lambda x.$range(1,$x+1$)
$Ty(int)$,
1
$Ty(int \rightarrow (int \rightarrow python))$,
$\lambda y.\lambda x.$range(y,$x+1$)

1. Make it clear what DS is all about: Dynamics
2. Generalize the composition calculus: lambda
3. Liberalise permissible node types
4. Liberalise permissible semantic representation languages
5. Use it!

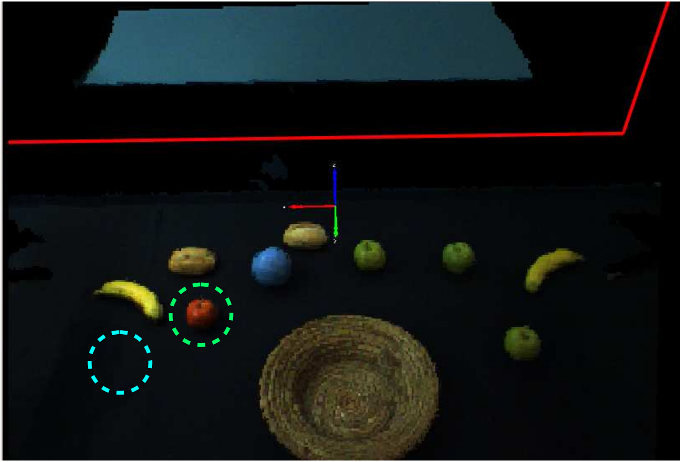
Use it!

- No excuse not to do great stuff!

Use it!

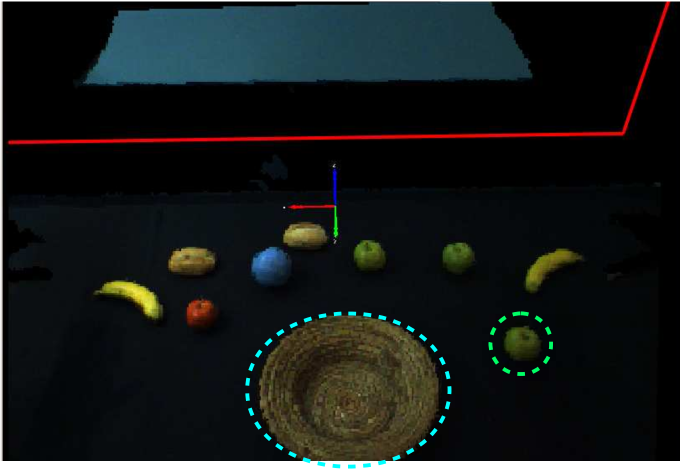
- No excuse not to do great stuff!
- Linguistic analysis!
- Formulae for theorem proving!
- Dialogue systems!
- Human-Robot Interaction systems (embodied)!

Human-Robot interaction live ambiguity resolution



put the apple in front of the banana

Human-Robot interaction live ambiguity resolution



... in the basket

Learning

- We can define arbitrary $< utterance, formula >$ pairs.
- Induce an incremental grammar in the style of [Eshghi et al., 2013].

Learning

- We can define arbitrary $<$ *utterance*, *formula* $>$ pairs.
- Induce an incremental grammar in the style of [Eshghi et al., 2013].
- We could learn useful regularities across domains.
- We could learn *'put'* as a distributional lexical action across different putting situations.
- *'red'* may have a perceptual lexical action grounded in machine vision.
- Otherwise we will always have to go through a pipeline from DS $\rightarrow$ X- why not be more direct?

1. Make it clear what DS is all about: Dynamics
2. Generalize the composition calculus: lambda
3. Liberalise permissible node types
4. Liberalise permissible semantic representation languages
5. Use it!

Thanks!

especially to:

- DUEL project (Bielefeld University and Paris 7, DFG and ANR)

Eshghi, A., Howes, C., Gregoromichelaki, E., Hough, J., and Purver, M. (2015).
Feedback in conversation as incremental semantic update.
In *Proceedings of the 11th International Conference on Computational Semantics*, pages 261–271, London, UK. ACL.

Eshghi, A., Purver, M., and Hough, J. (2013).
Probablistic induction for an incremental semantic grammar.
In *10th International Conference on Computational Semantics (IWCS)*, Potsdam, Germany.

Hough, J. (2011).
Incremental semantics driven natural language generation with self-repairing capability.
In *Proceedings of the Student Research Workshop associated with RANLP 2011*, pages 79–84, Hissar, Bulgaria.

Hough, J. (2015).
Modelling Incremental Self-Repair Processing in Dialogue.
PhD thesis, Queen Mary University of London.

Kempson, R., Gregoromichelaki, E., Edhghi, A., and Hough, J. (2018).
Ellipsis in dynamic syntax.
In *Oxford Handbook of Ellipsis*. OUP.

Kempson, R., Meyer-Viol, W., and Gabbay, D. (2001).
Dynamic Syntax: The Flow of Language Understanding.
Blackwell, Oxford.

Milward, D. (1991).
Axiomatic Grammar, Non-Constituent Coordination and Incremental Interpretation.
PhD thesis, University of Cambridge.

Purver, M., Eshghi, A., and Hough, J. (2011).
Incremental semantic construction in a dialogue system.
In Bos, J. and Pulman, S., editors, *Proceedings of the 9th International Conference on Computational Semantics*, pages 365–369, Oxford, UK.

Purver, M., Gregoromichelaki, E., Meyer-Viol, W., and Cann, R. (2010).
Splitting the 'I's and crossing the 'You's: Context, speech acts and grammar.
In Łupkowski, P. and Purver, M., editors, *Aspects of Semantics and Pragmatics of Dialogue. SemDial 2010, 14th Workshop on the Semantics and Pragmatics of Dialogue*, pages 43–50, Poznań. Polish Society for Cognitive Science.

Purver, M., Hough, J., and Gregoromichelaki, E. (2014).
Dialogue and compound contributions.
In Stent, A. and Bangalore, S., editors, *Natural Language Generation in Interactive Systems*, pages 63–92. Cambridge University Press.

Sato, Y. (2011).
Local ambiguity, search strategies and parsing in Dynamic Syntax.
In Gregoromichelaki, E., Kempson, R., and Howes, C., editors, *The Dynamics of Lexical Interfaces*, pages 205–233. CSLI.