Extending and Learning an Incremental Semantic Grammar

Matthew Purver
with Arash Eshghi, Julian Hough, Ruth Kempson et al

Interaction, Media and Communication
School of Electronic Engineering and Computer Science
Queen Mary, University of London

RISER - EPSRC EP/J010383/1
Robust Incremental SEmantic Resources for Dialogue

EPSRC Pioneering research and skills

Queen Mary University of London

IMC

Dialogue is Incremental

We don't always speak in "complete" sentences

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 2038-2044*]

Dialogue is Incremental

We don't always speak in "complete" sentences

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 2038-2044]*

- We're not dealing with individual grammatical *sentences*
- What does this tell us for grammar, parser, generator?
- Can we build a system that handles this?

Outline

Dialogue is Incremental

We don't always speak in "complete" sentences

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 2038-2044*]

Dialogue is Incremental

We don’t always speak in “complete” sentences

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 2038-2044*]

- Fragments and ellipsis (Fernandez & Ginzburg, 2003)
- Nearly 20% of BNC contributions continue another
- Over 70% continue something already apparently complete
- Pauses, role changes, continuations, self/other repair . . .

Incremental Processing

BNC KND 160-164

A: So if you start at the centre [pause] and draw a line and mark off seventy two degrees,
B: Mm.
A: and then mark off another seventy two degrees and another seventy two degrees and another seventy two degrees and join the ends,
B: Yeah.
A: you'll end up with a regular pentagon.

Incremental Processing

BNC KND 160-164

A: So if you start at the centre [pause] and draw a line and mark off seventy two degrees,
B: Mm.
A: and then mark off another seventy two degrees and another seventy two degrees and another seventy two degrees and join the ends,
B: Yeah.
A: you'll end up with a regular pentagon.

- NLG must be suspended and restarted in context
- NLU must be suspended and restarted in context

Parsing $\leftrightarrow$ Generation

BNC KPY 1005-1008

A: And er they X-rayed me, and took a urine sample, took a blood sample. Er, the doctor
B: Chorlton?
A: Chorlton, mhm, he examined me, erm, he, he said now they were on about a slide [unclear] on my heart.

Parsing $\leftrightarrow$ Generation

BNC KPY 1005-1008

A: And er they X-rayed me, and took a urine sample, took a blood sample. Er, the doctor
B: Chorlton?
A: Chorlton, mhm, he examined me, erm, he, he said now they were on about a slide [unclear] on my heart.

- NLG $\rightarrow$ NLU $\rightarrow$ NLG, in context

Parsing $\leftrightarrow$ Generation

BNC KPY 1005-1008

A: And er they X-rayed me, and took a urine sample, took a blood sample. Er, the doctor
B: Chorlton?
A: Chorlton, mhm, he examined me, erm, he, he said now they were on about a slide [unclear] on my heart.

- NLG $\rightarrow$ NLU $\rightarrow$ NLG, in context
- Partial interpretations must be available

Parsing $\leftrightarrow$ Generation

BNC KPY 1005-1008

A: And er they X-rayed me, and took a urine sample, took a blood sample. Er, the doctor
B: Chorlton?
A: Chorlton, mhm, he examined me, erm, he, he said now they were on about a slide [unclear] on my heart.

- NLG $\rightarrow$ NLU $\rightarrow$ NLG, in context
- Partial interpretations must be available
- Linguistic context must be available

Antecedent Completeness

BNC H5H 110-111

A: Before that then if they were ill
B: They get nothing.

- Antecedents often syntactically/semantically incomplete

Antecedent Completeness

BNC H5H 110-111

A: Before that then if they were ill
B: They get nothing.

- Antecedents often syntactically/semantically incomplete
- But sometimes already complete:

BNC FUK 2460-2461

A: The profit for the group is a hundred and ninety thousand pounds.
B: Which is superb.

Antecedent Completeness

BNC H5H 110-111

A: Before that then if they were ill
B: They get nothing.

- Antecedents often syntactically/semantically incomplete
- But sometimes already complete:

BNC FUK 2460-2461

A: The profit for the group is a hundred and ninety thousand pounds.
B: Which is superb.

- Need representations which can be extended incrementally

Syntax, But Not As We Know It

Syntactic Dependencies

A: I'm afraid I burnt the kitchen ceiling
B: But have you
A: burned myself? Fortunately not.

Syntax, But Not As We Know It

Syntactic Dependencies

A: I'm afraid I burnt the kitchen ceiling
B: But have you
A: burned myself? Fortunately not.

- Syntactic dependencies apply (context-dependent)

Syntax, But Not As We Know It

Syntactic Dependencies

A: I'm afraid I burnt the kitchen ceiling
B: But have you
A: burned myself? Fortunately not.

- Syntactic dependencies apply (context-dependent)
- But they can't be defined over strings

Syntax, But Not As We Know It

Syntactic Dependencies

A: I'm afraid I burnt the kitchen ceiling
B: But have you
A: burned myself? Fortunately not.

- Syntactic dependencies apply (context-dependent)
- But they can't be defined over strings

Syntactic Constituents

A: whereas qualitative is [pause] you know what the actual variations
B: entails

Syntax, But Not As We Know It

Syntactic Dependencies

A: I'm afraid I burnt the kitchen ceiling
B: But have you
A: burned myself? Fortunately not.

- Syntactic dependencies apply (context-dependent)
- But they can't be defined over strings

Syntactic Constituents

A: whereas qualitative is [pause] you know what the actual variations
B: entails

- Syntactic constituency not respected

Not Always Collaborative

Lerner (1991)

Daughter:	Oh here dad, a good way to get those corners out
Dad:	is to stick yer finger inside.
Daughter:	well, that's one way.

Not Always Collaborative

Lerner (1991)

Daughter:	Oh here dad, a good way to get those corners out
Dad:	is to stick yer finger inside.
Daughter:	well, that's one way.

- Not just plan recognition and extension

Outline

1. Dialogue & Incrementality
   - Compound Contributions
   - Requirements for Grammar
2. Tools for Incrementality
   - Dynamic Syntax
   - Type Theory with Records
3. DS/TTR: The DYLAN Framework
   - Incremental Interpretation
   - Context and Parse Graphs
   - Compound Contributions: DYLAN
4. Learning Incremental Grammar
   - Problem and Background
   - Learning Lexical Entries

Requirements for Grammar (see Milward, 1991)

- Incrementality
  - Processing language word by word

Requirements for Grammar (see Milward, 1991)

- Incrementality
  - Processing language word by word
- Incremental interpretation
  - Maximal semantic content calculated at each step

Requirements for Grammar (see Milward, 1991)

- Incrementality
  - Processing language word by word
- Incremental interpretation
  - Maximal semantic content calculated at each step
- Incremental representation
  - Contribution of each word/unit to representations built

Requirements for Grammar (see Milward, 1991)

- Incrementality
  - Processing language word by word
- Incremental interpretation
  - Maximal semantic content calculated at each step
- Incremental representation
  - Contribution of each word/unit to representations built
- Incremental context
  - Context added to and read from incrementally

Requirements for Grammar (see Milward, 1991)

- Incrementality
  - Processing language word by word
- Incremental interpretation
  - Maximal semantic content calculated at each step
- Incremental representation
  - Contribution of each word/unit to representations built
- Incremental context
  - Context added to and read from incrementally
- Reversibility
  - Representations common between parsing and generation

Requirements for Grammar (see Milward, 1991)

- Incrementality
  - Processing language word by word
- Incremental interpretation
  - Maximal semantic content calculated at each step
- Incremental representation
  - Contribution of each word/unit to representations built
- Incremental context
  - Context added to and read from incrementally
- Reversibility
  - Representations common between parsing and generation
- Extensibility
  - Representations extendable even for complete antecedents

Previous Approaches - Parsing

- Psycholinguistic Models (Sturt, Crocker)
- Computational Models (Roark, Hale)
  - Efficient, predictive parsing models
  - Based on string-licensing syntactic grammars

Previous Approaches - Parsing

- Psycholinguistic Models (Sturt, Crocker)
- Computational Models (Roark, Hale)
  - Efficient, predictive parsing models
  - Based on string-licensing syntactic grammars
- Categorial Grammar (Steedman, Clark, Milward)
  - Well-defined syntax/semantics interface
  - Incremental parsing by type-raising
  - (although see Hefny et al, 2001)

Previous Approaches - Generation

- Psycholinguistic models (De Smedt, Kempen, Guhe)
  - Modular / parallel generator components
  - Strategic $\rightarrow$ tactical generator components
  - Not left-to-right linguistic processing

Previous Approaches - Generation

- Psycholinguistic models (De Smedt, Kempen, Guhe)
  - Modular / parallel generator components
  - Strategic $\rightarrow$ tactical generator components
  - Not left-to-right linguistic processing
- Self-Monitoring Models (Neumann, van Noord)
  - Interleaved parsing $\leftrightarrow$ generation
  - Not left-to-right linguistic processing

Previous Approaches - Collaborative Completions

- Formal model (Poesio & Rieser)
  - Lexicalised TAG
  - PTT for dialogue/utterance context
  - Detailed plan recognition

Previous Approaches - Collaborative Completions

- Formal model (Poesio & Rieser)
  - Lexicalised TAG
  - PTT for dialogue/utterance context
  - Detailed plan recognition
- String-licensing grammar
- NLU/NLG interface unclear
- Relies on collaborative plan recognition

Previous Approaches - Dialogue

- General abstract model (Schlangen & Skantze)
- Incremental NLU (Schlangen, Buss, Peldszus et al, Aist et al)
  - Faster NLU and reference resolution
- Incremental NLG (Skantze, Hjalmarsson)
  - Faster, more natural generation with repair

Previous Approaches - Dialogue

- General abstract model (Schlangen & Skantze)
- Incremental NLU (Schlangen, Buss, Peldszus et al, Aist et al)
  - Faster NLU and reference resolution
- Incremental NLG (Skantze, Hjalmarsson)
  - Faster, more natural generation with repair
- Lacking NLU/NLG reversibility
- Lacking linguistic structure

What we need. . .

- An incremental grammar formalism for parsing and generation

What we need. . .

- An incremental grammar formalism for parsing and generation
  - *Dynamic Syntax* (Kempson et. al., 2001)

What we need. . .

- An incremental grammar formalism for parsing and generation
    - *Dynamic Syntax* (Kempson et. al., 2001)

- Ideally, a domain general formalism for (sub-propositional) semantic representation (which could interface easily with domain (frame) semantics)

What we need. . .

- An incremental grammar formalism for parsing and generation
  - *Dynamic Syntax* (Kempson et. al., 2001)
- Ideally, a domain general formalism for (sub-propositional) semantic representation (which could interface easily with domain (frame) semantics)
  - *Type Theory with Records (TTR)* (Cooper, 2005)

What we need. . .

- An incremental grammar formalism for parsing and generation
  - *Dynamic Syntax* (Kempson et. al., 2001)
- Ideally, a domain general formalism for (sub-propositional) semantic representation (which could interface easily with domain (frame) semantics)
  - *Type Theory with Records (TTR)* (Cooper, 2005)
- An incremental dialogue framework

What we need. . .

- An incremental grammar formalism for parsing and generation
  - *Dynamic Syntax* (Kempson et. al., 2001)
- Ideally, a domain general formalism for (sub-propositional) semantic representation (which could interface easily with domain (frame) semantics)
  - *Type Theory with Records (TTR)* (Cooper, 2005)
- An incremental dialogue framework
  - *Jindigo* (Schlangen & Skantze, 2009)

Outline

Dynamic Syntax

- An inherently incremental grammatical framework
- Word-by-word incremental construction of semantic interpretation:
  - no autonomous level of syntax
  - “syntax” defined via constraints on incremental semantic structure-building
  - “grammar” is a set of procedures for incremental parsing
  - “trees” are semantic representations defined using LoFT (Blackburn & Meyer-Viol, 1994)
- Monotonic growth with underspecification-plus-enrichment
- Predictivity: requirements for later satisfaction

DS Trees as semantic representations

- End product of parsing is a semantic tree
- Nodes decorated with $Ty()$ type and $Fo()$ formula labels

"John likes Mary":

$$Ty(t),\ Fo(like(john, mary))$$

$$Ty(e),\ Fo(john) \qquad Ty(e \rightarrow t),\ Fo(\lambda x.like(x, mary))$$

$$Ty(e),\ Fo(mary) \qquad Ty(e \rightarrow (e \rightarrow t)),\ Fo(\lambda y \lambda x.like(x, y))$$

- Daughter order does not reflect sentence order!
- Nodes interpretable as terms in the λ-calculus
- NPs map onto terms of type e using the ϵ-calculus.

Actions as tree-building procedures

- Incremental tree growth driven by *requirements* e.g. $?Ty(t)$
- Node under development marked by *pointer* $\diamondsuit$
- Words induce sets of *lexical* actions: *"john"*

IF	$?Ty(e)$
THEN	put($Fo(john)$);
	put($Ty(e)$)
ELSE	ABORT

$?Ty(e)$

Actions as tree-building procedures

- Incremental tree growth driven by *requirements* e.g. $?Ty(t)$
- Node under development marked by *pointer* $\diamondsuit$
- Words induce sets of *lexical* actions: *"john"*

IF	$?Ty(e)$
THEN	put($Fo(john)$);
	put($Ty(e)$)
ELSE	ABORT

$?Ty(e)\ ,Fo(john), Ty(e)$

Actions as tree-building procedures

- Incremental tree growth driven by *requirements* e.g. $?Ty(t)$
- Node under development marked by *pointer* $\diamondsuit$
- Words induce sets of *lexical* actions: *"like"*

IF	$?Ty(e \rightarrow t)$
THEN	make($\langle\downarrow_1\rangle$);go($\langle\downarrow_1\rangle$);
	put($Fo(\lambda y \lambda x.like(x,y))$);
	put($Ty(e \rightarrow (e \rightarrow t))$)
	go($\langle\uparrow_1\rangle$); make($\langle\downarrow_0\rangle$);
	go($\langle\downarrow_0\rangle$); put($?Ty(e)$)
ELSE	ABORT

$?Ty(e \rightarrow t)$

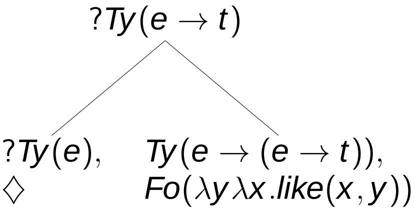
$?Ty(e)$, $\diamondsuit$

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

Actions as tree-building procedures

- Incremental tree growth driven by *requirements* e.g. $?Ty(t)$
- Node under development marked by *pointer* $\diamondsuit$
- Words induce sets of *lexical* actions: *"like"*

IF	$?Ty(e \rightarrow t)$
THEN	make($\langle\downarrow_1\rangle$);go($\langle\downarrow_1\rangle$);
	put($Fo(\lambda y \lambda x.like(x,y))$);
	put($Ty(e \rightarrow (e \rightarrow t))$)
	go($\langle\uparrow_1\rangle$); make($\langle\downarrow_0\rangle$);
	go($\langle\downarrow_0\rangle$); put($?Ty(e)$)
ELSE	ABORT



- General *computational* actions are also available e.g. requirement fulfillment, beta-reduction

Unfolding then building up the tree

Processing *John fainted*

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

Unfolding then building up the tree

Processing *John fainted*

$$?Ty(t)$$

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

Unfolding then building up the tree

Processing *John fainted*

$$?Ty(t)$$

$\diamondsuit$, $Ty(e)$, *john* $\qquad\qquad$ $?Ty(e \rightarrow t)$

Unfolding then building up the tree

Processing *John* *fainted*

$$?Ty(t)$$

$Ty(e)$, $john$ $\qquad$ $?Ty(e \rightarrow t), \diamondsuit$

Unfolding then building up the tree

Processing *John fainted*

$$\begin{array}{ccc} & ?Ty(t) & \\ Ty(e) & & Ty(e \rightarrow t), \diamondsuit \\ john & & \lambda y.faint(y) \end{array}$$

Unfolding then building up the tree

Processing *John fainted*

$$faint(john)$$
$$Ty(t),\ \Diamond$$

$Ty(e)$	$Ty(e \rightarrow t)$
$john$	$\lambda y.faint(y)$

Unfolding then building up the tree

Processing *John fainted*

$\rightsquigarrow faint(john)$

$$\begin{array}{ccc} & \begin{array}{c} faint(john) \\ Ty(t) \end{array} & \\ \begin{array}{c} Ty(e) \\ john \end{array} & & \begin{array}{c} Ty(e \rightarrow t) \\ \lambda y.faint(y) \end{array} \end{array}$$

Generation

- Speakers go through the same tree-growth 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 *John fainted*

GOAL TREE

$Ty(t), \diamondsuit$
$faint(john)$

$Ty(e)$, $john$ — $Ty(e \rightarrow t)$, $\lambda y.faint(y)$

TEST TREE

$?Ty(t), \diamondsuit$

Generation

- Speakers go through the same tree-growth 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 *John fainted*

GOAL TREE

$Ty(t), \diamondsuit$
$faint(john)$

$Ty(e)$, $john$ | $Ty(e \rightarrow t)$, $\lambda y.faint(y)$

TEST TREE

$?Ty(t)$,

$\diamondsuit, ?Ty(e)$ | $?Ty(e \rightarrow t)$

Generation

- Speakers go through the same tree-growth 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 *John fainted*

GOAL TREE

$Ty(t), \diamondsuit$
$faint(john)$

$Ty(e)$, $john$ — $Ty(e \rightarrow t)$, $\lambda y.faint(y)$

TEST TREE

$?Ty(t)$,

$\diamondsuit$, $Ty(e)$, $john$ — $?Ty(e \rightarrow t)$

Gen: “John

Generation

- Speakers go through the same tree-growth 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 *John fainted*

GOAL TREE

$Ty(t), \diamondsuit$
$faint(john)$

$Ty(e)$, $john$ — $Ty(e \rightarrow t)$, $\lambda y.faint(y)$

TEST TREE

$?Ty(t)$,

$Ty(e)$, $john$ — $?Ty(e \rightarrow t), \diamondsuit$

Gen: “John

Generation

- Speakers go through the same tree-growth 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 *John fainted*

GOAL TREE

$Ty(t), \diamondsuit$
$faint(john)$

$Ty(e),$ $john$ — $Ty(e \rightarrow t)$ $\lambda y.faint(y)$

TEST TREE

$?Ty(t),$

$Ty(e)$ $john$ — $?Ty(e \rightarrow t), \diamondsuit$ $\lambda y.faint(y)$

Gen: "John fainted"

Generation

- Speakers go through the same tree-growth 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 *John fainted*

GOAL TREE

$$Ty(t), \diamondsuit$$
$$faint(john)$$

$Ty(e),$ $john$ — $Ty(e \rightarrow t)$ $\lambda y.faint(y)$

TEST TREE

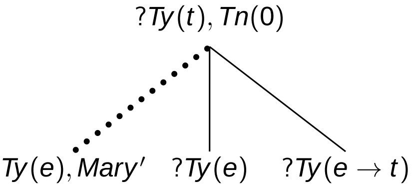
$$Ty(t), \diamondsuit$$
$$faint(john)$$

$Ty(e)$ $john$ — $Ty(e \rightarrow t)$ $\lambda y.faint(y)$

Gen: “John fainted”

Structural Underspecification

- “Unfixed” nodes - building underspecified tree relations

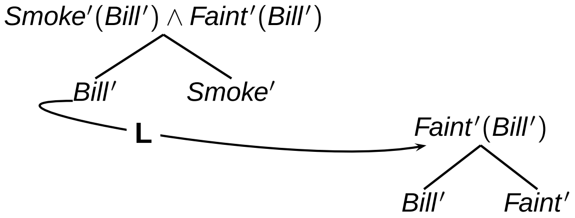


- Left-dislocation “Mary, John likes”

LINKed trees

- **Relative clauses**: pairs of LINKed trees evaluated as conjunction

  e.g. Bill, who fainted, smokes.



Context-dependence

- Pronouns project META-VARIABLES (**U**)
- Substituted by item from context during construction

Context-dependence

- Pronouns project META-VARIABLES (**U**)
- Substituted by item from context during construction

(1) Someone smoked He fainted.

Context-dependence

- Pronouns project META-VARIABLES (**U**)
- Substituted by item from context during construction

(1) Someone smoked He fainted.

TREE AS CONTEXT: TREE UNDER CONSTRUCTION:

$smoke(\epsilon, x, person(x))$

$\epsilon, x, person(x)$ $\lambda y.smoke(y)$

SUBSTITUTION

$?Ty(t)$

U, $?\exists x Fo(x)$ $\diamondsuit$

$\lambda y.faint(y)$

Context-dependence

- Pronouns project META-VARIABLES (**U**)
- Substituted by item from context during construction

(1) Someone smoked He fainted.

TREE AS CONTEXT: TREE UNDER CONSTRUCTION:

$$?Ty(t)$$

$$smoke(\epsilon, x, person(x))$$

$$\epsilon, x, person(x) \quad \lambda y.smoke(y)$$

$$\mathbf{U}, ?\exists x Fo(x) \; \diamondsuit \quad \lambda y.faint(y)$$

SUBSTITUTION

- Context must include trees and action sequences

How are we doing?

- Incrementality
  - Processing language word by word
- Incremental interpretation
  - Maximal semantic content calculated at each step
- Incremental representation
  - Contribution of each word/unit to representations built
- Incremental context
  - Context added to and read from incrementally
- Reversibility
  - Representations common between parsing and generation
- Extensibility
  - Representations extendable even for complete antecedents

How are we doing?

- Incrementality ✓
  - Processing language word by word
- Incremental interpretation
  - Maximal semantic content calculated at each step
- Incremental representation
  - Contribution of each word/unit to representations built
- Incremental context
  - Context added to and read from incrementally
- Reversibility
  - Representations common between parsing and generation
- Extensibility
  - Representations extendable even for complete antecedents

How are we doing?

- Incrementality✓
  - Processing language word by word
- Incremental interpretation?
  - Maximal semantic content calculated at each step
- Incremental representation
  - Contribution of each word/unit to representations built
- Incremental context
  - Context added to and read from incrementally
- Reversibility
  - Representations common between parsing and generation
- Extensibility
  - Representations extendable even for complete antecedents

How are we doing?

- Incrementality ✓
  - Processing language word by word
- Incremental interpretation?
  - Maximal semantic content calculated at each step
- Incremental representation ✗
  - Contribution of each word/unit to representations built
- Incremental context
  - Context added to and read from incrementally
- Reversibility
  - Representations common between parsing and generation
- Extensibility
  - Representations extendable even for complete antecedents

How are we doing?

- Incrementality ✓
  - Processing language word by word
- Incremental interpretation ?
  - Maximal semantic content calculated at each step
- Incremental representation ✗
  - Contribution of each word/unit to representations built
- Incremental context ✗
  - Context added to and read from incrementally
- Reversibility
  - Representations common between parsing and generation
- Extensibility
  - Representations extendable even for complete antecedents

How are we doing?

- Incrementality✓
  - Processing language word by word
- Incremental interpretation?
  - Maximal semantic content calculated at each step
- Incremental representation✗
  - Contribution of each word/unit to representations built
- Incremental context✗
  - Context added to and read from incrementally
- Reversibility✓
  - Representations common between parsing and generation
- Extensibility
  - Representations extendable even for complete antecedents

How are we doing?

- Incrementality ✓
  - Processing language word by word
- Incremental interpretation?
  - Maximal semantic content calculated at each step
- Incremental representation ✗
  - Contribution of each word/unit to representations built
- Incremental context ✗
  - Context added to and read from incrementally
- Reversibility ✓
  - Representations common between parsing and generation
- Extensibility?
  - Representations extendable even for complete antecedents

Some specific shortcomings

- No principled way to incorporate context information
  - e.g. constraints over speaker/hearer identity
- Generation requires a goal *tree*
  - i.e. knowledge of how the LF is to be compiled
- FOL/ϵ-calculus formulae hard to integrate with dialogue systems
  - usually DRT or frame-like constructs

Outline

Type Theory With Records

- (Cooper, 2005; Betarte & Tasistro, 1998), following Martin-Löf
- *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 sequences of label/type pairs:

$$\begin{bmatrix} l_1 & : & T_1 \\ l_2 & : & T_2 \\ l_3 & : & T_3 \end{bmatrix}$$

- Record types are true iff they are *inhabited/witnessed*
  - = there exists at least one record of that type
  - = successful type judgements for each label/value pair:

$$v_1 : T_1,\quad v_2 : T_2,\quad v_3 : T_3$$

Type Theory With Records

- Types can be *dependent* on earlier (higher-up) types:

$$\begin{bmatrix} l_1 & : & T_1 \\ l_2 & : & T_2(l_1) \\ l_3 & : & T_3(l_1, l_2) \end{bmatrix}$$

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

$$\begin{bmatrix} l_1 & : & T_1 \\ l_2 & : & \begin{bmatrix} l'_1 & : & T'_1 \\ l'_2 & : & T'_2 \end{bmatrix} \\ l_3 & : & T_3(l_1, l_2.l'_1, l_2.l'_2) \end{bmatrix}$$

- We can have *functional* record types:

$$\lambda r : \begin{bmatrix} l_1 & : & T_1 \\ l_2 & : & T_2 \end{bmatrix} \left(\begin{bmatrix} l_3 & : & T_3 \\ l_4 & : & T_4(r.l_1, r.l_2) \end{bmatrix} \right)$$

Type Theory With Records

- Subtype-supertype relations:

$$\left[\begin{array}{lll} l_1 & : & T_1 \end{array}\right] \sqsubset \left[\begin{array}{lll} l_1 & : & T_2 \end{array}\right] \quad \text{if} \quad T_1 \sqsubset T_2$$

$$\left[\begin{array}{lll} l_1 & : & T_1 \\ l_2 & : & T_2 \end{array}\right] \sqsubset \left[\begin{array}{lll} l_1 & : & T_1 \end{array}\right]$$

- All records are of type []
- Manifest (singleton) types:

$$\left[\begin{array}{lll} x & : & john \end{array}\right] \sqsubset \left[\begin{array}{lll} x & : & e \end{array}\right] \quad \text{if} \quad john \sqsubset e$$

$$\left[\begin{array}{lll} x_{=john} & : & e \end{array}\right]$$

Type Theory With Records

- Used for sentential semantics, e.g. Cooper (2005)
  - "A man left": $\begin{bmatrix} x & : & man \\ p & : & leave(x) \end{bmatrix}$
  - for truth: x must be a man, p a proof that x left
  - "Every man left":
    $\lambda r : \begin{bmatrix} x & : & man \end{bmatrix} (\begin{bmatrix} p & : & leave(r.x) \end{bmatrix})$
- Similarities to DRT representation:

x
man(x) leave(x)

- Used for dialogue modelling in the information-state-based tradition
  - (Cooper & Ginzburg, 2002; Ranta & Cooper, 2004; Fernandez, 2006; Ginzburg, 2012)

The best of both worlds?

- TTR gives us a type-theoretic framework, applicable to dialogue phenomena
- DS gives us an incremental framework using type theory as an underlying mechanism
- Can we combine the two?

The best of both worlds?

- TTR gives us a type-theoretic framework, applicable to dialogue phenomena
- DS gives us an incremental framework using type theory as an underlying mechanism
- Can we combine the two?

$$\diamondsuit, leave(john), Ty(t)$$

$$john, \quad Ty(e)$$

$$\lambda x.leave(x), \quad Ty(e \rightarrow t)$$

The best of both worlds?

- TTR gives us a type-theoretic framework, applicable to dialogue phenomena
- DS gives us an incremental framework using type theory as an underlying mechanism
- Can we combine the two?

$$\diamondsuit, \textit{leave}(\textit{john}), Ty(t)$$

$$\textit{john}, Ty(e) \qquad \lambda x.\textit{leave}(x), Ty(e \rightarrow t)$$

$$\diamondsuit, \left[\begin{array}{lcl} x & : & \textit{john} \\ e & : & \textit{leave}(x) \end{array}\right]$$

$$\left[\, x : \textit{john} \,\right] \qquad \lambda[x].\left[\, p : \textit{leave}(x) \,\right]$$

Combining DS with TTR

- Replace *Fo*() epsilon-calculus labels with TTR record types

Combining DS with TTR

- Replace *Fo*() epsilon-calculus labels with TTR record types

$\diamondsuit, ?Ty(t)$

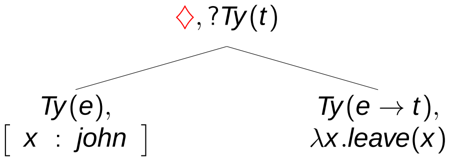
$Ty(e)$, *john*

$Ty(e \rightarrow t)$, $\lambda x.leave(x)$

IF	$?Ty(e)$
THEN	put($Ty(e)$)
	put($Fo(john)$)
ELSE	abort

Combining DS with TTR

- Replace *Fo*() epsilon-calculus labels with TTR record types



IF	$?Ty(e)$
THEN	put($Ty(e)$)
	put($\left[\ x\ :\ john\ \right]$)
ELSE	abort

Combining DS with TTR

- Replace *Fo*() epsilon-calculus labels with TTR record types
- Interpret *Ty*() simple type labels as referring to *final* TTR field type

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

$$Ty(e), \left[\ x : john\ \right] \qquad\qquad Ty(e \rightarrow t), \lambda x.leave(x)$$

IF	$?Ty(e)$
THEN	put($Ty(e)$)
	put($\left[\ x : john\ \right]$)
ELSE	abort

Combining DS with TTR

- Replace *Fo*() epsilon-calculus labels with TTR record types
- Interpret *Ty*() simple type labels as referring to *final* TTR field type

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

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

$$Ty(e \rightarrow t), \quad \lambda \left[\begin{array}{lll} x & : & e \end{array}\right] . \left[\begin{array}{lll} x & : & e \\ p & : & leave(x) \end{array}\right]$$

Combining DS with TTR

- Replace *Fo*() epsilon-calculus labels with TTR record types
- Interpret *Ty*() simple type labels as referring to *final* TTR field type
- Function application as before for DS `elimination` process

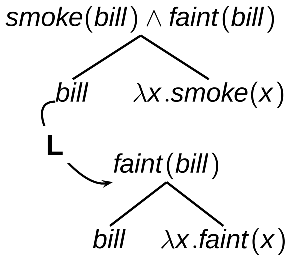
$$\diamondsuit, Ty(t), \left[\begin{array}{lcl} x & : & john \\ p & : & leave(x) \end{array}\right]$$

$$Ty(e), \left[\begin{array}{lcl} x & : & john \end{array}\right] \qquad Ty(e \rightarrow t), \lambda\left[\begin{array}{lcl} x & : & e \end{array}\right].\left[\begin{array}{lcl} x & : & e \\ p & : & leave(x) \end{array}\right]$$

Adding in LINK relations

- For LINKed trees, we need conjunction

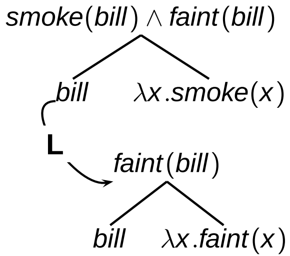
“Bill, who fainted, smokes.”



Adding in LINK relations

- For LINKed trees, we need conjunction
- Use *extension*: $\oplus$ where $r_1 \oplus r_2$ adds r_2 to the end of r_1
  - (for distinct labels; identical fields collapse (Cooper, 1998))

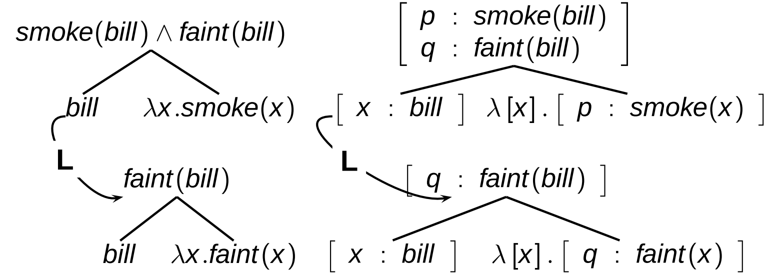
"Bill, who fainted, smokes."



Adding in LINK relations

- For LINKed trees, we need conjunction
- Use *extension*: $\oplus$ where $r_1 \oplus r_2$ adds r_2 to the end of r_1
  - (for distinct labels; identical fields collapse (Cooper, 1998))

"Bill, who fainted, smokes."



DS/TTR

- Recent work integrating DS with TTR (Purver et al, 2011)

$$\left[\begin{array}{lcl} x_{=john} & : & e \\ p_{=leave(x)} & : & t \end{array}\right]$$

$$\left[\begin{array}{lcl} x_{=john} & : & e \end{array}\right] \qquad \lambda \left[\begin{array}{lcl} x & : & e \end{array}\right] . \left[\begin{array}{lcl} x & : & e \\ p_{=leave(x)} & : & t \end{array}\right]$$

DS/TTR

- Recent work integrating DS with TTR (Purver et al, 2011)

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

$$\begin{bmatrix} x_{=john} & : & e \end{bmatrix} \qquad \lambda \begin{bmatrix} x & : & e \end{bmatrix} . \begin{bmatrix} x & : & e \\ p_{=leave(x)} & : & t \end{bmatrix}$$

- TTR *record types* now provide the semantic content of each node of the DS trees

DS/TTR

- Recent work integrating DS with TTR (Purver et al, 2011)

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

$$\begin{bmatrix} x_{=john} & : & e \end{bmatrix} \qquad \lambda \begin{bmatrix} x & : & e \end{bmatrix} . \begin{bmatrix} x & : & e \\ p_{=leave(x)} & : & t \end{bmatrix}$$

- TTR *record types* now provide the semantic content of each node of the DS trees
- LINKed trees for relative clauses and adjuncts are easily incorporated by extending (intersecting) *record types*

DS/TTR

- Recent work integrating DS with TTR (Purver et al, 2011)

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

$$\begin{bmatrix} x_{=john} & : & e \end{bmatrix} \qquad \lambda \begin{bmatrix} x & : & e \end{bmatrix} . \begin{bmatrix} x & : & e \\ p_{=leave(x)} & : & t \end{bmatrix}$$

- TTR *record types* now provide the semantic content of each node of the DS trees
- LINKed trees for relative clauses and adjuncts are easily incorporated by extending (intersecting) *record types*
- Recently, a Davidsonian event-based semantics for tense has been incorporated (Cann, 2010, see next slide)

Outline

Root Node Type Deduction

- Inference of maximal semantic content (Hough, 2011)

$$?Ty(t),$$

$$Ty(e), \quad \left[\; x_{=john} \;:\; e \;\right]$$

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

$$?Ty(e),$$

$$Ty(e \rightarrow e \rightarrow t), \quad \lambda\,[y : e]\,.\lambda\,[x : e] \begin{bmatrix} x & : & e \\ y & : & e \\ p_{=like(x,y)} & : & t \end{bmatrix}$$

Root Node Type Deduction

- Inference of maximal semantic content (Hough, 2011)

$$?Ty(t),$$

$$Ty(e), \quad \left[\, x_{=john} \; : \; e \,\right] \qquad\qquad ?Ty(e \rightarrow t),$$

$$?Ty(e), \quad \left[\, y \; : \; e \,\right] \qquad\qquad Ty(e \rightarrow e \rightarrow t), \quad \lambda[y:e].\lambda[x:e]\begin{bmatrix} x & : & e \\ y & : & e \\ p_{=like(x,y)} & : & t \end{bmatrix}$$

Root Node Type Deduction

- Inference of maximal semantic content (Hough, 2011)

$$?Ty(t),$$

$$Ty(e), \quad \begin{bmatrix} x_{=john} & : & e \end{bmatrix}$$

$$?Ty(e \rightarrow t), \quad \lambda[x:e].\begin{bmatrix} x & : & e \\ y & : & e \\ p_{=like(x,y)} & : & t \end{bmatrix}$$

$$?Ty(e), \quad \begin{bmatrix} y & : & e \end{bmatrix}$$

$$Ty(e \rightarrow e \rightarrow t), \quad \lambda[y:e].\lambda[x:e]\begin{bmatrix} x & : & e \\ y & : & e \\ p_{=like(x,y)} & : & t \end{bmatrix}$$

Root Node Type Deduction

- Inference of maximal semantic content (Hough, 2011)

$$?Ty(t), \begin{bmatrix} x_{=john} & : & e \\ y & : & e \\ p_{=like(x,y)} & : & t \end{bmatrix}$$

$$Ty(e), \begin{bmatrix} x_{=john} & : & e \end{bmatrix}$$

$$?Ty(e \rightarrow t), \quad \lambda [x : e] . \begin{bmatrix} x & : & e \\ y & : & e \\ p_{=like(x,y)} & : & t \end{bmatrix}$$

$$?Ty(e), \begin{bmatrix} y & : & e \end{bmatrix}$$

$$Ty(e \rightarrow e \rightarrow t), \quad \lambda [y : e] . \lambda [x : e] \begin{bmatrix} x & : & e \\ y & : & e \\ p_{=like(x,y)} & : & t \end{bmatrix}$$

Root Node Type Deduction

- Inference of maximal semantic content (Hough, 2011)

$$Ty(t), \begin{bmatrix} x_{=john} & : & e \\ y_{=mary} & : & e \\ p_{=like(x,y)} & : & t \end{bmatrix}$$

$$Ty(e), \begin{bmatrix} x_{=john} & : & e \end{bmatrix}$$

$$Ty(e \rightarrow t), \lambda[x:e].\begin{bmatrix} x & : & e \\ y_{=mary} & : & e \\ p_{=like(x,y)} & : & t \end{bmatrix}$$

$$Ty(e), \begin{bmatrix} y_{=mary} & : & e \end{bmatrix}$$

$$Ty(e \rightarrow e \rightarrow t), \lambda[y:e].\lambda[x:e]\begin{bmatrix} x & : & e \\ y & : & e \\ p_{=like(x,y)} & : & t \end{bmatrix}$$

Generation from Goal Concepts

- We can now generate from a goal *concept* (not *tree*)

GOAL TREE

$Ty(t), \Diamond$
$faint(john)$

$Ty(e)$, $john$ | $Ty(e \rightarrow t)$ $\lambda y.faint(y)$

TEST TREE

$?Ty(t), \Diamond$

Generation from Goal Concepts

- We can now generate from a goal *concept* (not *tree*)

GOAL TREE

$Ty(t), \diamondsuit$
$faint(john)$

$Ty(e)$,
$john$

$Ty(e \rightarrow t)$
$\lambda y.faint(y)$

TEST TREE

$?Ty(t)$,

$\diamondsuit, ?Ty(e)$

$?Ty(e \rightarrow t)$

Generation from Goal Concepts

- We can now generate from a goal *concept* (not *tree*)

GOAL TREE

$Ty(t), \diamondsuit$
$faint(john)$

$Ty(e)$, $john$ — $Ty(e \rightarrow t)$ $\lambda y.faint(y)$

TEST TREE

$?Ty(t)$,

$\diamondsuit$, $Ty(e)$ $john$ — $?Ty(e \rightarrow t)$

Gen: “John

Generation from Goal Concepts

- We can now generate from a goal *concept* (not *tree*)

GOAL TREE

$Ty(t), \diamondsuit$
$faint(john)$

$Ty(e),$ $john$ — $Ty(e \rightarrow t)$ $\lambda y.faint(y)$

TEST TREE

$?Ty(t),$

$Ty(e)$ $john$ — $?Ty(e \rightarrow t), \diamondsuit$

Gen: "John

Generation from Goal Concepts

- We can now generate from a goal *concept* (not *tree*)

GOAL TREE

$Ty(t), \diamondsuit$
$faint(john)$

$Ty(e),$
$john$

$Ty(e \rightarrow t)$
$\lambda y.faint(y)$

TEST TREE

$?Ty(t),$

$Ty(e)$
$john$

$?Ty(e \rightarrow t), \diamondsuit$
$\lambda y.faint(y)$

Gen: "John fainted"

Generation from Goal Concepts

- We can now generate from a goal *concept* (not *tree*)

GOAL TREE

$Ty(t)$, $\diamondsuit$
$faint(john)$

$Ty(e)$, $john$ — $Ty(e \rightarrow t)$ $\lambda y.faint(y)$

TEST TREE

$Ty(t)$, $\diamondsuit$
$faint(john)$

$Ty(e)$ $john$ — $Ty(e \rightarrow t)$ $\lambda y.faint(y)$

Gen: "John fainted"

Generation from Goal Concepts

- We can now generate from a goal *concept* (not *tree*)

GOAL CONCEPT

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

TEST TREE

$?Ty(t), \diamondsuit$

Generation from Goal Concepts

- We can now generate from a goal *concept* (not *tree*)

GOAL CONCEPT

$$\left[\begin{array}{lcl} x_{=john} & : & e \\ p_{=faint(x)} & : & t \end{array}\right]$$

TEST TREE

$$?Ty(t),$$

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

Generation from Goal Concepts

- We can now generate from a goal *concept* (not *tree*)

GOAL CONCEPT

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

TEST TREE

$$?Ty(t),$$

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

$$\begin{bmatrix} x_{=john} & : & e \end{bmatrix}$$

Gen: “John

Generation from Goal Concepts

- We can now generate from a goal *concept* (not *tree*)

GOAL CONCEPT

$$\left[\begin{array}{lll} x_{=john} & : & e \\ p_{=faint(x)} & : & t \end{array}\right]$$

TEST TREE

$$?Ty(t), \left[\begin{array}{lll} x_{=john} & : & e \end{array}\right]$$

$$Ty(e), \left[\begin{array}{lll} x_{=john} & : & e \end{array}\right] \qquad ?Ty(e \rightarrow t), \diamondsuit$$

Gen: “John

Generation from Goal Concepts

- We can now generate from a goal *concept* (not *tree*)

GOAL CONCEPT

$$\left[\begin{array}{lcl} x_{=john} & : & e \\ p_{=faint(x)} & : & t \end{array}\right]$$

TEST TREE

$$?Ty(t), \left[\begin{array}{lcl} x_{=john} & : & e \end{array}\right]$$

$$Ty(e) \quad \left[\begin{array}{lcl} x_{=john} & : & e \end{array}\right]$$

$$?Ty(e \rightarrow t), \diamondsuit \quad \lambda x. \left[\begin{array}{lcl} x & : & e \\ p_{=faint(x)} & : & t \end{array}\right]$$

Gen: “John fainted”

Generation from Goal Concepts

- We can now generate from a goal *concept* (not *tree*)

GOAL CONCEPT

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

TEST TREE

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

$$Ty(e) \quad \begin{bmatrix} x_{=john} & : & e \end{bmatrix} \qquad Ty(e \rightarrow t) \quad \lambda x. \begin{bmatrix} x & : & e \\ p_{=faint(x)} & : & t \end{bmatrix}$$

Gen: “John fainted”

Incremental Semantic Construction with DS-TTR

- Davidsonian semantics, LINKed trees: incremental interpretation

Incremental Semantic Construction with DS-TTR

- Davidsonian semantics, LINKed trees: incremental interpretation

$$\begin{bmatrix} event_{=e1} & : & e_s \\ RefTime & : & e_s \\ p1_{=today(RefTime)} & : & t \\ p2_{=RefTime \bigcirc event} & : & t \end{bmatrix}$$

A: Today

Incremental Semantic Construction with DS-TTR

- Davidsonian semantics, LINKed trees: incremental interpretation

$$\left[\begin{array}{lcl} event_{=e1} & : & e_s \\ RefTime & : & e_s \\ p1_{=today(RefTime)} & : & t \\ p2_{=RefTime \bigcirc event} & : & t \\ x_{=robin} & : & e \\ p_{=arrive(event,x)} & : & t \end{array}\right]$$

A: Today.. Robin arrives

Incremental Semantic Construction with DS-TTR

- Davidsonian semantics, LINKed trees: incremental interpretation

$$\left[\begin{array}{lcl} event_{=e1} & : & e_s \\ RefTime & : & e_s \\ p1_{=today(RefTime)} & : & t \\ p2_{=RefTime \bigcirc event} & : & t \\ x_{=robin} & : & e \\ p_{=arrive(event,x)} & : & t \\ x1 & : & e \\ p3_{=from(event,x1)} & : & t \end{array}\right]$$

A: Today.. Robin arrives
B: From?

Incremental Semantic Construction with DS-TTR

- Davidsonian semantics, LINKed trees: incremental interpretation

$$\left[\begin{array}{lcl} event_{=e1} & : & e_s \\ RefTime & : & e_s \\ p1_{=today(RefTime)} & : & t \\ p2_{=RefTime \bigcirc event} & : & t \\ x_{=robin} & : & e \\ p_{=arrive(event,x)} & : & t \\ x1_{=Sweden} & : & e \\ p3_{=from(event,x1)} & : & t \end{array}\right]$$

A: Today.. Robin arrives
B: From?
A: Sweden

Incremental Semantic Construction with DS-TTR

- Davidsonian semantics, LINKed trees: incremental interpretation

$$\left[\begin{array}{lcl} event_{=e1} & : & e_s \\ RefTime & : & e_s \\ p1_{=today(RefTime)} & : & t \\ p2_{=RefTime \bigcirc event} & : & t \\ x_{=robin} & : & e \\ p_{=arrive(event,x)} & : & t \\ x1_{=Sweden} & : & e \\ p3_{=from(event,x1)} & : & t \\ x2_{=Elisabeth} & : & e \\ p4_{=with(event,x2)} & : & t \end{array}\right]$$

A: Today.. Robin arrives
B: From?
A: Sweden
B: With Elisabeth?

Incremental Semantic Construction with DS-TTR

- Davidsonian semantics, LINKed trees: incremental interpretation

$$\begin{bmatrix} event_{=e1} & : & e_s \\ RefTime & : & e_s \\ p1_{=today(RefTime)} & : & t \\ p2_{=RefTime \bigcirc event} & : & t \\ x_{=robin} & : & e \\ p_{=arrive(event,x)} & : & t \\ x1_{=Sweden} & : & e \\ p3_{=from(event,x1)} & : & t \\ x2_{=} & : & e \\ p4_{=with(event,x2)} & : & t \end{bmatrix}$$

A: Today.. Robin arrives
B: From?
A: Sweden
B: With Elisabeth?

- As far as we are aware, no other formalism can model such online, collaborative and incremental construction of meaning that is diagnostic of dialogue.

Outline

Adding utterance context

- Add minimal utterance context information (see Poesio & Traum/Rieser)
  - Utterance event (for each word)
  - Speaker and addressee for that event

Adding utterance context

- Add minimal utterance context information (see Poesio & Traum/Rieser)
  - Utterance event (for each word)
  - Speaker and addressee for that event

$$\begin{bmatrix} A & : & participantA \\ B & : & participantB \\ u & : & utt\text{-}event \\ s_u & : & e \\ p_s & : & spkr(u, s_u) \\ a_u & : & e \\ p_a & : & addr(u, a_a) \end{bmatrix}$$

Adding utterance context

- Add minimal utterance context information (see Poesio & Traum/Rieser)
  - Utterance event (for each word)
  - Speaker and addressee for that event

$$\begin{bmatrix} A & : & participantA \\ B & : & participantB \\ u & : & utt\text{-}event \\ s_u(=A) & : & e \\ p_s & : & spkr(u, s_u) \\ a_u(=B) & : & e \\ p_a & : & addr(u, a_a) \end{bmatrix}$$

Adding utterance context

- Add minimal utterance context information (see Poesio & Traum/Rieser)
  - Utterance event (for each word)
  - Speaker and addressee for that event

$$\begin{bmatrix} A & : & participantA \\ B & : & participantB \\ u & : & utt\text{-}event \\ s_u(=A) & : & e \\ p_s & : & spkr(u, s_u) \\ a_u(=B) & : & e \\ p_a & : & addr(u, a_a) \end{bmatrix}$$

$$\begin{bmatrix} u & : & utt(s_u, a_u) \end{bmatrix}$$

$$\begin{bmatrix} u & : & utt(A, B) \end{bmatrix}$$

Adding utterance context

- Add minimal utterance context information (see Poesio & Traum/Rieser)
  - Utterance event (for each word)
  - Speaker and addressee for that event

$$\Diamond, Ty(e), \left[\; x \; : \; john \; \right]$$

Adding utterance context

- Add minimal utterance context information (see Poesio & Traum/Rieser)
  - Utterance event (for each word)
  - Speaker and addressee for that event

$$\diamondsuit, Ty(e), \left[\begin{array}{lll} u_0 & : & utt(s_0, a_0) \\ x & : & john \end{array} \right]$$

Adding utterance context

- Add minimal utterance context information (see Poesio & Traum/Rieser)
  - Utterance event (for each word)
  - Speaker and addressee for that event

$$\diamondsuit, Ty(e), \begin{bmatrix} ctxt & : & \begin{bmatrix} u_0 & : & utt(s_0, a_0) \end{bmatrix} \\ cont & : & \begin{bmatrix} x & : & john \end{bmatrix} \end{bmatrix}$$

Adding utterance context

- Add minimal utterance context information (see Poesio & Traum/Rieser)
  - Utterance event (for each word)
  - Speaker and addressee for that event

$$\Diamond, Ty(e), \left[\begin{array}{lcl} ctxt & : & \left[\begin{array}{lcl} u_0 & : & utt(s_0, a_0) \end{array} \right] \\ cont & : & \left[\begin{array}{lcl} x & : & john \end{array} \right] \end{array} \right]$$

- Content can refer to context, but not vice versa
- Assume this information available when parsing a word

Using utterance context

- Can use this to provide identity for indexicals
- "*I*":

$$\begin{array}{ll} \text{IF} & ?Ty(e), \left[\ ctxt\ :\ \left[\ u : utt(s_u, a_u)\ \right]\ \right], \\ \text{THEN} & \mathtt{put}(Ty(e)), \\ & \mathtt{put}\left(\left[\begin{array}{ll} ctxt & : \left[\ u : utt(s_u, a_u)\ \right] \\ cont & : \left[\ s_u : e\ \right] \end{array} \right] \right) \end{array}$$

Using utterance context

- Can use this to provide identity for indexicals
- "*I*":

$$
\begin{array}{ll}
\text{IF} & ?Ty(e), \left[\ ctxt\ :\ \left[\ u\ :\ utt(s_u, a_u)\ \right]\ \right], \\
\text{THEN} & \mathtt{put}(Ty(e)), \\
 & \mathtt{put}\left(\begin{bmatrix} ctxt & : & \left[\ u\ :\ utt(s_u, a_u)\ \right] \\ cont & : & \left[\ s_u\ :\ e\ \right] \end{bmatrix}\right)
\end{array}
$$

$$?Ty(t)$$

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

$$\begin{bmatrix} ctxt & : & \left[\ u_0\ :\ utt(A, B)\ \right] \\ & & \end{bmatrix}$$

Using utterance context

- Can use this to provide identity for indexicals
- "*I*":

$$
\begin{array}{ll}
\text{IF} & ?Ty(e), \left[\ ctxt\ :\ \left[\ u\ :\ utt(s_u, a_u)\ \right]\ \right], \\
\text{THEN} & \mathtt{put}(Ty(e)), \\
& \mathtt{put}\left(\left[\begin{array}{ll} ctxt & : \left[\ u\ :\ utt(s_u, a_u)\ \right] \\ cont & : \left[\ s_u\ :\ e\ \right] \end{array} \right] \right)
\end{array}
$$

$$?Ty(t)$$

$$Ty(e), \Diamond$$

$$\left[\begin{array}{ll} ctxt & : \left[\ u_0\ :\ utt(A, B)\ \right] \\ cont & : \left[\ x\ :\ john\ \right] \end{array} \right]$$

Using utterance context

- Can use this to provide identity for indexicals
- *"I"*:

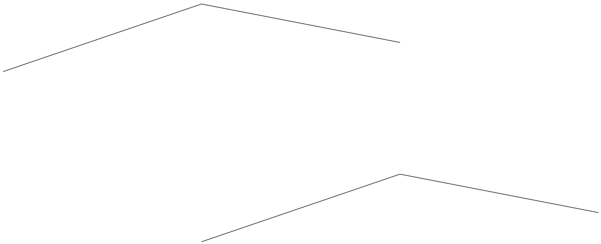
$$\begin{array}{ll} \text{IF} & ?Ty(e), \left[\ ctxt\ :\ \left[\ u\ :\ utt(s_u, a_u)\ \right]\ \right], \\ \text{THEN} & \mathtt{put}(Ty(e)), \\ & \mathtt{put}\left(\left[\begin{array}{ll} ctxt & : \left[\ u\ :\ utt(s_u, a_u)\ \right] \\ cont & : \left[\ s_u\ :\ e\ \right] \end{array}\right]\right) \end{array}$$

- *"myself"*:

$$\begin{array}{ll} \text{IF} & ?Ty(e), \left[\ ctxt\ :\ \left[\ u\ :\ utt(s_u, a_u)\ \right]\ \right], \\ & \uparrow_0\uparrow_{1*}\downarrow_0 \left[\ cont\ :\ \left[\ x(=s_u)\ :\ e\ \right]\ \right] \\ \text{THEN} & \mathtt{put}(Ty(e)), \\ & \mathtt{put}\left(\left[\begin{array}{ll} ctxt & : \left[\ u\ :\ utt(s_u, a_u)\ \right] \\ cont & : \left[\ s_u\ :\ e\ \right] \end{array}\right]\right) \end{array}$$

Split utterances with indexicals

- A: I like . . . B: yourself.



Split utterances with indexicals

- A: I like . . . B: yourself.

$$cx : \left[\begin{array}{lll} u_0 & : & utt(A,B) \end{array} \right]$$

Split utterances with indexicals

- A: I like . . . B: yourself.

$$\begin{bmatrix} cx & : & \begin{bmatrix} u_0 & : & utt(A,B) \end{bmatrix} \\ ct & : & \begin{bmatrix} x & : & A \end{bmatrix} \end{bmatrix}$$

Split utterances with indexicals

- A: I like . . . B: yourself.

$$\begin{bmatrix} cx : \begin{bmatrix} u_0 & : & utt(A,B) \end{bmatrix} \\ ct : \begin{bmatrix} x & : & A \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} cx : \begin{bmatrix} u_1 & : & utt(A,B) \end{bmatrix} \\ \\ \\ \end{bmatrix}$$

Split utterances with indexicals

- A: I like . . . B: yourself.

$$\begin{bmatrix} cx : \begin{bmatrix} u_0 & : & utt(A,B) \end{bmatrix} \\ ct : \begin{bmatrix} x & : & A \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} cx : \begin{bmatrix} u_1 & : & utt(A,B) \end{bmatrix} \\ ct : \lambda \begin{bmatrix} x & : & e \\ y & : & e \end{bmatrix} . \begin{bmatrix} x & : & e \\ y & : & e \\ p & : & like(y,x) \end{bmatrix} \end{bmatrix}$$

Split utterances with indexicals

- A: I like . . . B: yourself.

$$\left[\begin{array}{lcl} cx & : & \left[\begin{array}{lcl} u_0 & : & utt(A,B) \end{array}\right] \\ ct & : & \left[\begin{array}{lcl} x & : & A \end{array}\right] \end{array}\right]$$

$$\left[\begin{array}{lcl} cx & : & \left[\begin{array}{lcl} u_2 & : & utt(B,A) \end{array}\right] \end{array}\right] \qquad \left[\begin{array}{lcl} cx & : & \left[\begin{array}{lcl} u_1 & : & utt(A,B) \end{array}\right] \\ ct & : & \lambda \left[\begin{array}{lcl} x & : & e \\ y & : & e \end{array}\right] . \left[\begin{array}{lcl} x & : & e \\ y & : & e \\ p & : & like(y,x) \end{array}\right] \end{array}\right]$$

Split utterances with indexicals

- A: I like . . . B: yourself.

$$\left[\begin{array}{lcl} cx & : & \left[\begin{array}{lcl} u_0 & : & utt(A,B) \end{array}\right] \\ ct & : & \left[\begin{array}{lcl} x & : & A \end{array}\right] \end{array}\right]$$

$$\left[\begin{array}{lcl} cx & : & \left[\begin{array}{lcl} u_2 & : & utt(B,A) \end{array}\right] \\ ct & : & \left[\begin{array}{lcl} y & : & A \end{array}\right] \end{array}\right] \qquad \left[\begin{array}{lcl} cx & : & \left[\begin{array}{lcl} u_1 & : & utt(A,B) \end{array}\right] \\ ct & : & \lambda\left[\begin{array}{lcl} x & : & e \\ y & : & e \end{array}\right].\left[\begin{array}{lcl} x & : & e \\ y & : & e \\ p & : & like(y,x) \end{array}\right] \end{array}\right]$$

Split utterances with indexicals

- A: I like . . . B: yourself.

$$\left[\begin{array}{lcl} cx & : & \left[\begin{array}{lcl} u_0 & : & utt(A,B) \end{array}\right] \\ ct & : & \left[\begin{array}{lcl} x & : & A \end{array}\right] \end{array}\right]$$

$$\left[\begin{array}{lcl} cx & : & \left[\begin{array}{lcl} u_1 & : & utt(A,B), u_2 \end{array}\right] \\ ct & : & \lambda\left[\begin{array}{lcl} x & : & e \end{array}\right].\left[\begin{array}{lcl} x & : & e \\ y & : & A \\ p & : & like(y,x) \end{array}\right] \end{array}\right]$$

$$\left[\begin{array}{lcl} cx & : & \left[\begin{array}{lcl} u_2 & : & utt(B,A) \end{array}\right] \\ ct & : & \left[\begin{array}{lcl} y & : & A \end{array}\right] \end{array}\right]$$

$$\left[\begin{array}{lcl} cx & : & \left[\begin{array}{lcl} u_1 & : & utt(A,B) \end{array}\right] \\ ct & : & \lambda\left[\begin{array}{lcl} x & : & e \\ y & : & e \end{array}\right].\left[\begin{array}{lcl} x & : & e \\ y & : & e \\ p & : & like(y,x) \end{array}\right] \end{array}\right]$$

Split utterances with indexicals

- A: I like ... B: yourself.

$$Ty(t), \begin{bmatrix} ctxt & : & \begin{bmatrix} u_0 & : & utt(A,B), u_1, u_2 \end{bmatrix} \\ cont & : & \begin{bmatrix} x & : & A \\ y & : & A \\ p & : & like(x,y) \end{bmatrix} \end{bmatrix}$$

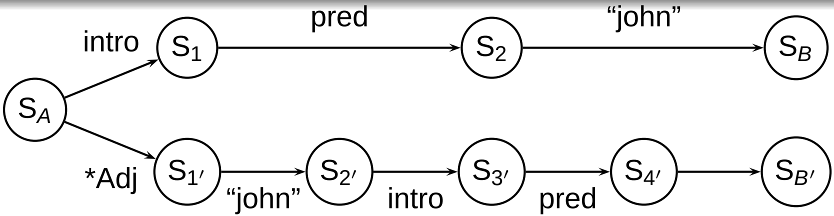
$$\begin{bmatrix} cx & : & \begin{bmatrix} u_0 & : & utt(A,B) \end{bmatrix} \\ ct & : & \begin{bmatrix} x & : & A \end{bmatrix} \end{bmatrix} \qquad \begin{bmatrix} cx & : & \begin{bmatrix} u_1 & : & utt(A,B), u_2 \end{bmatrix} \\ ct & : & \lambda \begin{bmatrix} x & : & e \end{bmatrix} . \begin{bmatrix} x & : & e \\ y & : & A \\ p & : & like(y,x) \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} cx & : & \begin{bmatrix} u_2 & : & utt(B,A) \end{bmatrix} \\ ct & : & \begin{bmatrix} y & : & A \end{bmatrix} \end{bmatrix} \qquad \begin{bmatrix} cx & : & \begin{bmatrix} u_1 & : & utt(A,B) \end{bmatrix} \\ ct & : & \lambda \begin{bmatrix} x & : & e \\ y & : & e \end{bmatrix} . \begin{bmatrix} x & : & e \\ y & : & e \\ p & : & like(y,x) \end{bmatrix} \end{bmatrix}$$

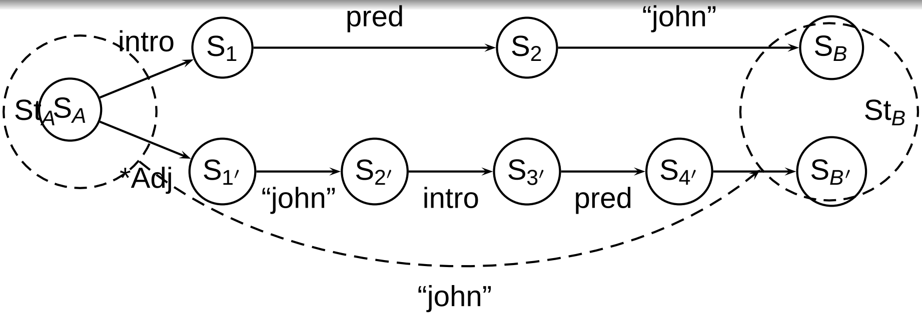
Parsing in Dynamic Syntax

- Parsing starts from some partial tree, proceeds in a time linear manner, reading the words in one by one, applying the corresponding lexical actions, optionally interspersing computational actions.
- This process is modelled on a Directed Acyclic Graph (DAG) (Purver et al. 2011, Sato, 2010) where:
  - Nodes = Trees
  - Edges = actions (lexical or computational)
  - Different Paths represent different parsing strategies.

DS Parse DAG

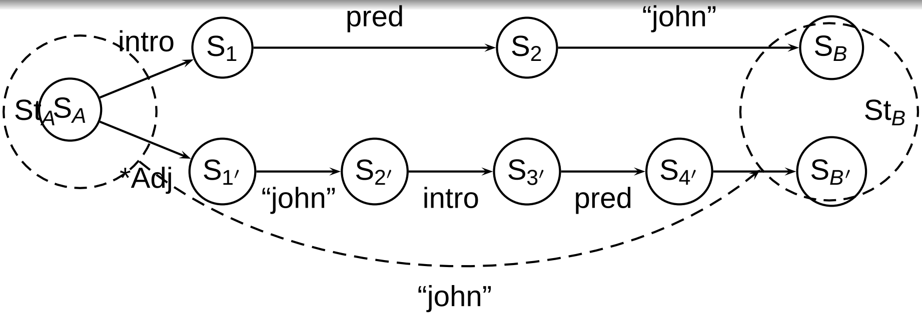


DS Parse DAG



- Integrate with word graph (and ASR "lattice")
  - Nodes = tree sets
  - Edges = word transitions

DS Parse DAG



- Integrate with word graph (and ASR "lattice")
  - Nodes = tree sets
  - Edges = word transitions
- Graph *is* context model: words, trees, action sequences
  - Incremental *representation*

Outline

How are we doing now?

- Incrementality ✓
  - Processing language word by word
- Incremental interpretation
  - Maximal semantic content calculated at each step
- Incremental representation
  - Contribution of each word/unit to representations built
- Incremental context
  - Context added to and read from incrementally
- Reversibility ✓
  - Representations common between parsing and generation
- Extensibility
  - Representations extendable even for complete antecedents

How are we doing now?

- Incrementality ✓
  - Processing language word by word
- Incremental interpretation ✓
  - Maximal semantic content calculated at each step
- Incremental representation
  - Contribution of each word/unit to representations built
- Incremental context
  - Context added to and read from incrementally
- Reversibility ✓
  - Representations common between parsing and generation
- Extensibility ✓
  - Representations extendable even for complete antecedents

How are we doing now?

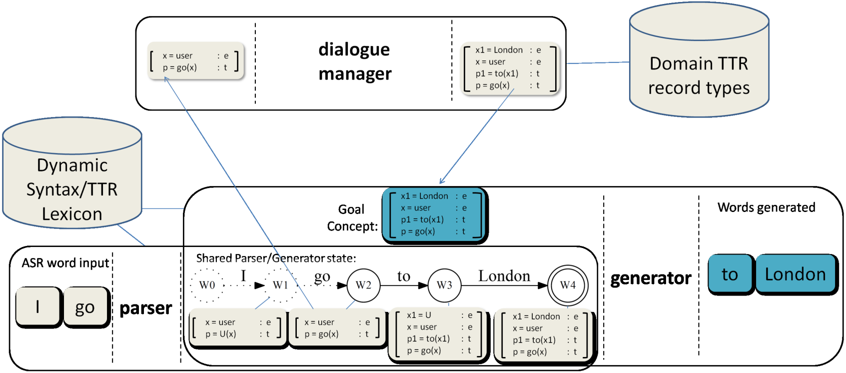
- Incrementality✓
  - Processing language word by word
- Incremental interpretation✓
  - Maximal semantic content calculated at each step
- Incremental representation✓
  - Contribution of each word/unit to representations built
- Incremental context✓
  - Context added to and read from incrementally
- Reversibility✓
  - Representations common between parsing and generation
- Extensibility✓
  - Representations extendable even for complete antecedents

So . . .

- This seems like a suitable framework
- Can we actually do anything with it . . . ?

Jindigo Module interaction: sharing parse state lattices

- Parse state DAG is common to generation and parsing, so can be shared between Interpreter and Generator modules...



But . . .

- What about the coverage?

Outline

Problem: learning incremental semantic grammars

- DS is idiosyncratic: no independent level of syntactic processing, and word-by-word incremental
- Current induction methods developed for grammars that:
  - define syntactic structures over words.
  - are not incremental, i.e. they cannot deal with partial utterances/sentences.

- These methods are therefore hard or impossible to adapt directly.
- For (semi-)supervised learning, we need corpora that are annotated for semantic structure (i.e. logical form), e.g. GeoQuery, PropBank

Previous work on induction

- Supervised: e.g. learning of Context Free Grammar rules from parsed corpora (e.g. PCFGs, Charniak (1996)): These have achieved great success, but the assumptions made are too strong to be plausible as a model of child language acquisition.
- Unsupervised: learning of grammars from raw, unannotated corpora: less successful. The problem is computationally intractable in the worst case (Gold, 1967)
- Latent Variable Supervised: e.g. learn from sentences paired with their meaning, i.e. their compositional semantic structure or Logical Form (LF).

Semantically supervised learning

- Successfully applied in Combinatorial Categorial Grammar (Steedman, 2000), as it tightly couples compositional semantics with syntax (Zettlemoyer& Collins, 2007; Kwiatkowski et al. 2010; Kwiatkowski et al. 2011).
- Our problem of inducing DS lexical actions is in the same spirit . . .
- . . . except that CCG is not word-by-word incremental.
- Approach: hypothesize lexical entries which can be extended to yield the known LF

Semantically supervised learning: cognitive plausibility

- In interaction, (child) learner can often know/infer what an utterance means overall without knowing some of the individual words:
  - Shared focus of attention with others or
  - Constrained directly through ‘helpful’ interaction with others, see e.g. Saxton (2010) for how corrective feedback can facilitate acquisition.
- This remains an open domain for further empirical investigation.

Assumptions

- The data for the induction task consists of sentences paired with the DS tree that expresses their predicate-argument structure (rather than a flat LF).
- Tree operations (DS computational actions, i.e. lambda calculus) are known
- The training data does not contain any context-dependent expressions (no pronouns, no ellipsis).
- We have a seed lexicon reducing the hypothesis space.
- Actions are conditioned solely on the semantic type of the pointed node. This is true of most lexical actions in DS (see IF clause in examples above), but not all.

The problem

Input:
- the set of computational actions in Dynamic Syntax, G.
- a seed lexicon, L_s consisting of a set of known words with their associated lexical actions
- a set of training examples of the form $\langle S_i, T_i \rangle$, where S_i is a sentence of the language and T_i – henceforth referred to as the *target tree* – is the complete semantic tree representing the compositional structure of the meaning of S_i.

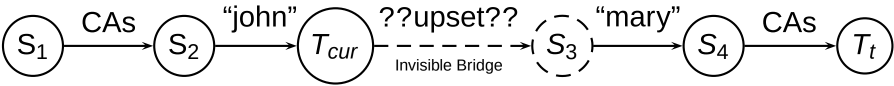
Our task:
- learn the THEN clauses of unknown lexical actions: sequences of DS atomic actions such as *go*, *make*, and *put*.

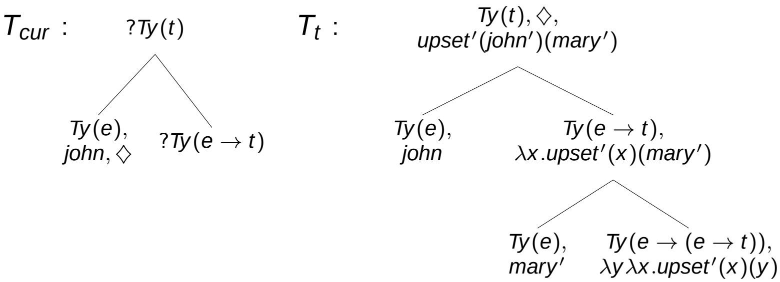
Outline

Method: incremental hypothesis construction

- DS is strictly monotonic:
  - Hypothesising lexical actions $=$ an incremental search through the space of all monotonic extensions of the current tree T_{cur} that subsume the target tree T_t.
- Basic constraints on the structure of DS lexical actions makes the search space tractable.
- Hypothesis construction is integrated with parsing over a parse state DAG as above.
- It proceeds by locally and incrementally extending the current tree, taking it one step closer to the target tree.
- Paths that lead to trees that do not subsume the target tree are blocked.

Learning “upset” from “john upset mary”





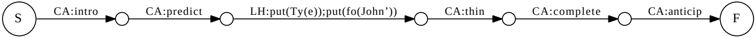
Generalisation through sequence intersection

- The output from each training example is a mapping from words to hypothesis *Candidate Sequences* extracted from the DAG.
- We refine and generalise over Candidate Sequences by *Sequence Intersection* modulo computational actions

Generalisation through sequence intersection

- The output from each training example is a mapping from words to hypothesis *Candidate Sequences* extracted from the DAG.
- We refine and generalise over Candidate Sequences by *Sequence Intersection* modulo computational actions

John (subject)

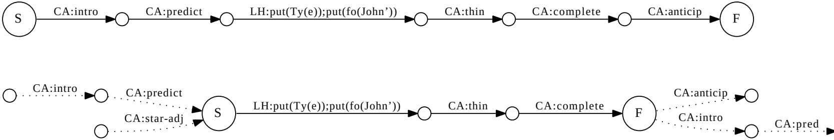


Generalisation through sequence intersection

- The output from each training example is a mapping from words to hypothesis *Candidate Sequences* extracted from the DAG.
- We refine and generalise over Candidate Sequences by *Sequence Intersection* modulo computational actions

John (subject)

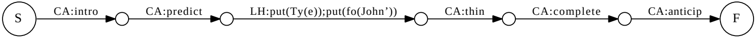
John (unfixed)



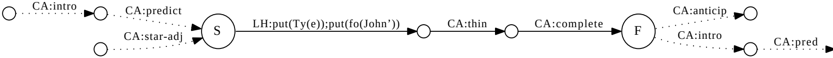
Generalisation through sequence intersection

- The output from each training example is a mapping from words to hypothesis *Candidate Sequences* extracted from the DAG.
- We refine and generalise over Candidate Sequences by *Sequence Intersection* modulo computational actions

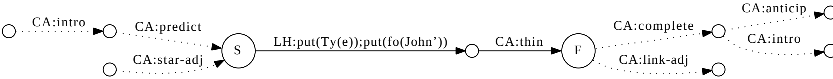
John (subject)



John (unfixed)



John (relative)



- Lexical Ambiguity is postulated when the candidate sequences cannot be intersected in this manner.

Parameter Estimation

- DS lexical entries:

IF	$?Ty(e)$
THEN	put($Fo(john)$);
	put($Ty(e)$)
ELSE	ABORT

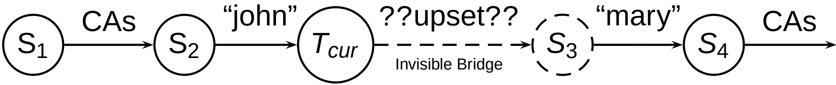
- now become probability distributions $p(S|w,t)$ where:
    - S is a candidate action sequence
    - w is a word
    - t is a type requirement
- With toy examples, can estimate parameters directly . . .
- (providing a model for probabilistic parsing)

Scaling Up

- As with generation, can constrain hypothesis construction by TTR record type (not DS tree)
- Existing corpora (e.g. CHILDES) mean we don't need to use toy examples
- Will this remain tractable?

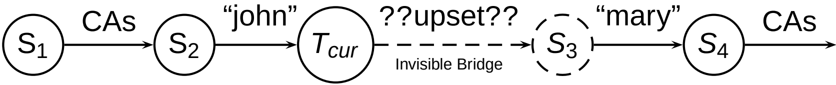
Scaling Up

- As with generation, can constrain hypothesis construction by TTR record type (not DS tree)
- Existing corpora (e.g. CHILDES) mean we don't need to use toy examples
- Will this remain tractable?
- Is this a reinforcement learning problem?



Scaling Up

- As with generation, can constrain hypothesis construction by TTR record type (not DS tree)
- Existing corpora (e.g. CHILDES) mean we don't need to use toy examples
- Will this remain tractable?
- Is this a reinforcement learning problem?



- Any suggestions gratefully received . . .

Thank you

Many people to thank: Arash Eshghi, Julian Hough, Ruth Kempson, Eleni Gregoromichelaki, Yo Sato, Wilfried Meyer-Viol, Graham White, Chris Howes, Pat Healey among others.