Grammar Induction in an Incremental Type-Theoretic Framework

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

Cognitive Science Group
School of Electronic Engineering and Computer Science
Queen Mary, University of London

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

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 (or learn) a suitable grammar?

Outline

1. Dialogue & Incrementality
   - Compound Contributions
   - Requirements for Grammar
2. Tools for Incrementality: DS and TTR
   - Dynamic Syntax (Kempson et al, 2001)
   - Type Theory with Records
   - DS/TTR: The DYLAN Framework
   - Filling the Gaps
3. Learning Incremental Grammar
   - Problem and Background
   - Hypothesising Lexical Entries
   - Learning Lexical Entries

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*]

- Nearly 20% of BNC contributions continue another
- Over 70% continue something already apparently complete
- Pauses, role changes, continuations, self/other repair . . .
- Incremental parsing & generation, highly coordinated

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

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 - requires look-ahead
  - (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, 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, Aist et al)
  - Faster NLU and reference resolution
- Incremental NLG (Skantze, Hjalmarsson)
  - Faster, more natural generation with repair
- NLU/NLG reversibility?
- Linguistic structure, constraints?
- Linguistic context?

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 construction of semantic interpretation:
  - "trees" = semantic representations defined using LoFT (Blackburn & Meyer-Viol, 1994)
    - nodes interpretable as terms in the λ-calculus
  - "syntax" = constraints on semantic structure-building
  - "grammar" = set of procedures for incremental parsing
    - *computational* and *lexical* actions
- Trees decorated with $Ty()$ type and $Fo()$ formula labels
  - Monotonic growth driven by *requirements* $?Ty(e)$
  - NPs map onto terms of type *e* using the ϵ-calculus.
  - Daughter order does not reflect sentence order!

Unfolding then building up the tree

Parsing *John fainted*

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

Unfolding then building up the tree

Parsing *John fainted*

$$
\begin{array}{ccc}
 & ?Ty(t), \diamondsuit & \\
?Ty(e) & & ?Ty(e \rightarrow t)
\end{array}
$$

INTRODUCTION

Unfolding then building up the tree

Parsing *John fainted*

$$\begin{array}{ccc} & ?Ty(t) & \\ \diamondsuit, ?Ty(e) & & ?Ty(e \rightarrow t) \end{array}$$

PREDICTION

Unfolding then building up the tree

Parsing *John fainted*

$$?Ty(t)$$

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

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

Unfolding then building up the tree

Parsing *John fainted*

$$?Ty(t)$$

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

$?Ty(e \rightarrow t)$

Unfolding then building up the tree

Parsing *John fainted*

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

THINNING

Unfolding then building up the tree

Parsing *John fainted*

$?Ty(t), \diamondsuit$

$Ty(e)$
john

$?Ty(e \rightarrow t)$

COMPLETION

Unfolding then building up the tree

Parsing *John fainted*

$$?Ty(t)$$

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

PREDICTION

Unfolding then building up the tree

Parsing *John fainted*

$?Ty(t)$

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

john

IF	$?Ty(e \rightarrow t)$
THEN	put($Fo(\lambda y.faint(y))$);
	put($Ty(e \rightarrow t)$)
ELSE	ABORT

Unfolding then building up the tree

Parsing *John fainted*

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

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

THINNING, COMPLETION

Unfolding then building up the tree

Parsing *John fainted*

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

ELIMINATION

Unfolding then building up the tree

Parsing *John fainted*

$\rightsquigarrow$ $faint(john)$

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

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

Unfolding then building up the tree

Generating *John fainted*: generate & test subsumption

GOAL TREE:

$$faint(john)$$
$$Ty(t)$$

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

$?Ty(t), \diamondsuit$

Unfolding then building up the tree

Generating *John fainted*: generate & test subsumption

GOAL TREE:

$faint(john)$
$Ty(t)$

$Ty(e)$
$john$

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

$?Ty(t), \diamondsuit$

$?Ty(e)$

$?Ty(e \rightarrow t)$

Unfolding then building up the tree

Generating *John fainted*: generate & test subsumption

GOAL TREE:

$faint(john)$
$Ty(t)$

$Ty(e)$
$john$

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

$?Ty(t)$

$\Diamond, ?Ty(e)$

$?Ty(e \rightarrow t)$

Unfolding then building up the tree

Generating *John fainted*: generate & test subsumption

GOAL TREE:

$faint(john)$
$Ty(t)$

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

$?Ty(t)$

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

Unfolding then building up the tree

Generating *John fainted*: generate & test subsumption

GOAL TREE:

$$faint(john)$$
$$Ty(t)$$

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

$$?Ty(t)$$

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

Unfolding then building up the tree

Generating *John fainted*: generate & test subsumption

GOAL TREE:

$$faint(john)$$
$$Ty(t)$$

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

$$?Ty(t)$$

$\Diamond$, $Ty(e)$, $john$ | $?Ty(e \rightarrow t)$

Unfolding then building up the tree

Generating *John* *fainted*: generate & test subsumption

GOAL TREE:

$faint(john)$
$Ty(t)$

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

$?Ty(t), \Diamond$

$Ty(e)$ $\quad$ $?Ty(e \rightarrow t)$
$john$

Unfolding then building up the tree

Generating *John fainted*: generate & test subsumption

GOAL TREE:

$faint(john)$
$Ty(t)$

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

$?Ty(t)$

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

Unfolding then building up the tree

Generating *John fainted*: generate & test subsumption

GOAL TREE:

$$faint(john)$$
$$Ty(t)$$

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

$$?Ty(t)$$

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

Unfolding then building up the tree

Generating *John fainted*: generate & test subsumption

GOAL TREE:

$$\begin{array}{c} faint(john) \\ Ty(t) \end{array}$$

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

$?Ty(t), \diamondsuit$

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

Unfolding then building up the tree

Generating *John fainted*: generate & test subsumption

GOAL TREE:

$faint(john)$
$Ty(t)$

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

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

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

There's more . . .

- "Unfixed" nodes - building underspecified tree relations
  - e.g. for left-dislocation "Mary, John likes"
- LINKed trees evaluated as conjunction
  - e.g. for relative clauses "John, who snores, arrived"
- Metavariables for anaphoric elements
  - to be resolved from items/actions in context
  - intrasentential too: relative clauses as above

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$\checkmark$
  - 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

- FOL/ϵ-calculus formulae
  - how do we extend complete formulae?
  - dialogue systems tend to prefer DRT/frames
- Generation requires a goal *tree*
  - i.e. knowledge of how the LF is to be compiled
- No principled way to incorporate context information
  - e.g. constraints over speaker/hearer identity

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*
- But you guys know this stuff.

Type Theory With Records

- Well-defined subtype-supertype relations:

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

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

- Manifest (singleton) types:

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

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

- Dialogue modelling in the information state 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?

$\Diamond, \mathit{leave}(\mathit{john}), Ty(t)$

$\mathit{john},\ Ty(e)$

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

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?

$$\Diamond, \mathit{leave}(\mathit{john}), Ty(t)$$

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

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

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

Outline

Combining DS with TTR

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

$\Diamond, ?Ty(t)$

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

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

IF	$?Ty(e)$
THEN	put($Ty(e)$)
	put($\left[\ x_{=john}\ :\ e\ \right]$)
ELSE	abort

- See (Purver et al, SemDial 2010; IWCS 2011)

Combining DS with TTR

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

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

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

- See (Purver et al, SemDial 2010; IWCS 2011)

Combining DS with TTR

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

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

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

- See (Purver et al, SemDial 2010; IWCS 2011)

Combining DS with TTR

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

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

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

- See (Purver et al, SemDial 2010; IWCS 2011)

Combining DS with TTR

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

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

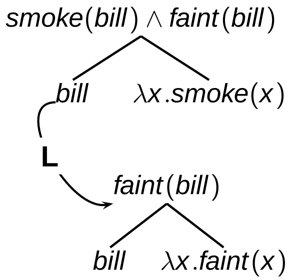
$$Ty(e), \quad \left[\begin{array}{lcl} x & : & john \end{array}\right] \qquad\qquad Ty(e \rightarrow t), \quad \lambda \left[\begin{array}{lcl} x & : & e \end{array}\right] . \left[\begin{array}{lcl} x & : & e \\ p_{=leave(x)} & : & t \end{array}\right]$$

- See (Purver et al, SemDial 2010; IWCS 2011)

Adding in LINK relations

- For LINKed trees, we need conjunction

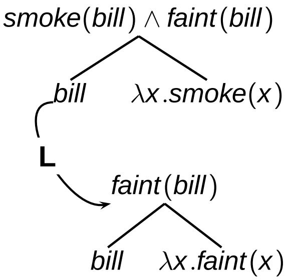
"Bill, who fainted, smokes."



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."

$$smoke(bill) \wedge faint(bill)$$

$$bill \qquad \lambda x.smoke(x)$$

L

$$faint(bill)$$

$$bill \qquad \lambda x.faint(x)$$

$$\begin{bmatrix} x_{=bill} & : & e \\ p_{=smoke(bill)} & : & t \\ q_{=faint(bill)} & : & t \end{bmatrix}$$

$$\begin{bmatrix} x_{=bill} & : & e \end{bmatrix} \qquad \lambda[x].\begin{bmatrix} p_{=smoke(x)} & : & t \end{bmatrix}$$

L

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

$$\begin{bmatrix} x_{=bill} & : & e \end{bmatrix} \qquad \lambda[x].\begin{bmatrix} q_{=faint(x)} & : & t \end{bmatrix}$$

Outline

Root Node Type Deduction

- Inference of maximal semantic content (Hough, 2011)

$?Ty(t),$

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

$?Ty(e \rightarrow t),$

$?Ty(e),$

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

Root Node Type Deduction

- Inference of maximal semantic content (Hough, 2011)

$?Ty(t),$

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

$?Ty(e \rightarrow t),$

$?Ty(e),$
$\left[\ y\ :\ e\ \right]$

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

Root Node Type Deduction

- Inference of maximal semantic content (Hough, 2011)

$$?Ty(t),$$

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

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

$$?Ty(e), \quad \left[\begin{array}{lcl} y & : & e \end{array}\right]$$

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

Root Node Type Deduction

- Inference of maximal semantic content (Hough, 2011)

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

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

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

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

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

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

$$\begin{array}{c} Ty(t), \diamondsuit \\ faint(john) \end{array}$$

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

TEST TREE

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

Generation from Goal Concepts

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

GOAL TREE

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

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

TEST TREE

Root: $?Ty(t)$,

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

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

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

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

TEST TREE

$$?Ty(t),$$

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

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

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

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

TEST TREE

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

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

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

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

TEST TREE

$$?Ty(t),$$

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

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

$?Ty(t),$

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

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

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

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

TEST TREE

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

$$\begin{array}{cc} Ty(e) & ?Ty(e \rightarrow t), \diamondsuit \\ \begin{bmatrix} x_{=john} & : & e \end{bmatrix} & \lambda x . \begin{bmatrix} x & : & e \\ p_{=faint(x)} & : & t \end{bmatrix} \end{array}$$

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

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

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

Gen: "John fainted"

Incremental Semantic Construction with DS-TTR

- Davidsonian semantics, LINKed trees:

Incremental Semantic Construction with DS-TTR

- Davidsonian semantics, LINKed trees:

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

A: Today

Incremental Semantic Construction with DS-TTR

- Davidsonian semantics, LINKed trees:

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

A: Today.. Robin arrives

Incremental Semantic Construction with DS-TTR

- Davidsonian semantics, LINKed trees:

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

A: Today.. Robin arrives
B: From?

Incremental Semantic Construction with DS-TTR

- Davidsonian semantics, LINKed trees:

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

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

Incremental Semantic Construction with DS-TTR

- Davidsonian semantics, LINKed trees:

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

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

Incremental Semantic Construction with DS-TTR

- Davidsonian semantics, LINKed trees:

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

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

- $\rightarrow$ incremental interpretation

Adding utterance context

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

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

Adding utterance context

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

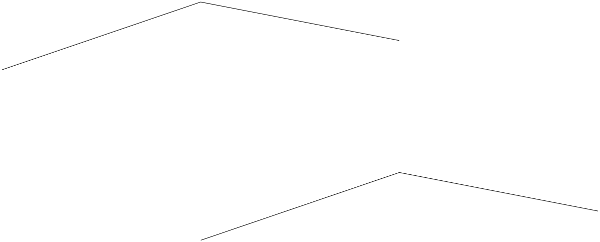
$$\diamondsuit, Ty(e), \left[\begin{array}{lll} ctxt & : & \left[\begin{array}{lll} u_0 & : & utt(s_0, a_0) \end{array}\right] \\ cont & : & \left[\begin{array}{lll} x & : & john \end{array}\right] \end{array}\right]$$

- *"myself"*:

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

Split utterances with indexicals

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



Split utterances with indexicals

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

$$cx : \left[\ u_0 \ : \ utt(A, B)\ \right]$$

Split utterances with indexicals

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

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

Split utterances with indexicals

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

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

$$\left[\begin{array}{lcl} cx & : & \left[\begin{array}{lcl} u_1 & : & utt(A,B) \end{array}\right] \\ & & \\ & & \end{array}\right]$$

Split utterances with indexicals

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

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

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

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.

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

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

Split utterances with indexicals

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

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

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

Split utterances with indexicals

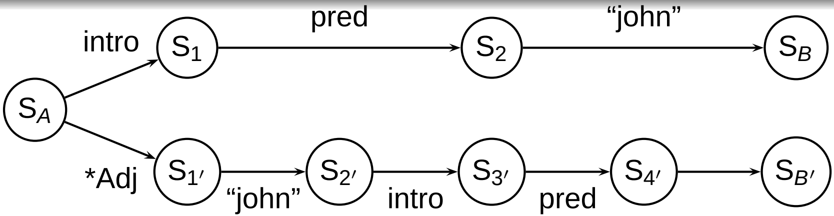
- A: I like . . . B: yourself.

$$
Ty(t), \left[\begin{array}{lcl} ctxt & : & \left[\begin{array}{lcl} u_0 & : & utt(A,B), u_1, u_2 \end{array}\right] \\ cont & : & \left[\begin{array}{lcl} x & : & A \\ y & : & A \\ p & : & like(x,y) \end{array}\right] \end{array}\right]
$$

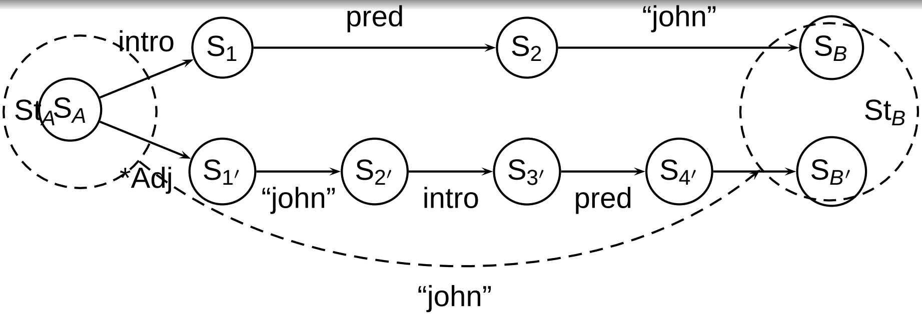
$$
\left[\begin{array}{lcl} cx & : & \left[\begin{array}{lcl} u_0 & : & utt(A,B) \end{array}\right] \\ ct & : & \left[\begin{array}{lcl} x & : & A \end{array}\right] \end{array}\right]
\qquad
\left[\begin{array}{lcl} cx & : & \left[\begin{array}{lcl} u_1 & : & utt(A,B), u_2 \end{array}\right] \\ ct & : & \lambda \left[\begin{array}{lcl} x & : & e \end{array}\right] . \left[\begin{array}{lcl} x & : & e \\ y & : & A \\ p & : & like(y,x) \end{array}\right] \end{array}\right]
$$

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

Parsing in DS/TTR (Sato 2010; Purver et al 2011)

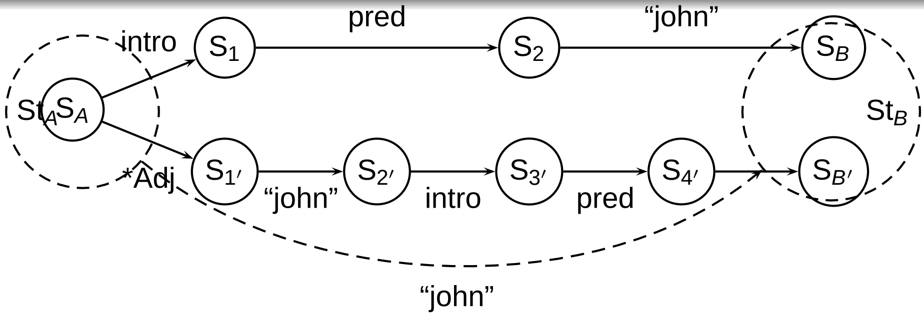


Parsing in DS/TTR (Sato 2010; Purver et al 2011)



- Integrate with word graph (and ASR “lattice”)
  - Nodes = tree sets (and TTR record types)
  - Edges = word transitions (lexical/computational actions)

Parsing in DS/TTR (Sato 2010; Purver et al 2011)



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

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?

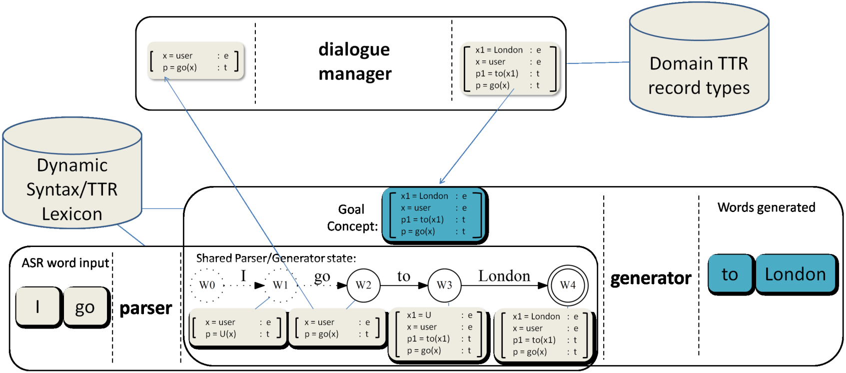
- Incrementality ✓
  - Processing language word by word
- Incremental interpretation ✓
  - Maximal semantic content calculated at each step
- Incremental representation ✓
  - Contribution of each word/unit to representations built
- Incremental context ✓
  - Context added to and read from incrementally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

DYLAN Dialogue System – via Jindigo

- Incremental dialogue, compound contributions, self-repair . . .
- (see Hough, 20 mins time)



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
- Increasing coverage manually is unrealistic . . .
- We need to learn from data!

Problem: learning incremental semantic grammars

- DS is idiosyncratic: no independent level of syntactic processing, and word-by-word incremental
- Increasing coverage manually is unrealistic . . .
- We need to learn from data!
- Current induction methods developed for grammars that:
  - define syntactic structures over words
  - are not incremental, i.e. cannot deal with partial utterances/sentences
- Therefore hard or impossible to adapt directly

Previous work on induction

- Supervised: e.g. learning PCFGs from parsed corpora (e.g. Charniak, 1996)
  - successful for PSGs, but cognitively implausible
  - no data available for us
- Unsupervised: learning from raw, unannotated corpora
  - less successful: computationally intractable in the worst case (Gold, 1967)
  - not clear how to apply to semantic problem

Previous work on induction

- Supervised: e.g. learning PCFGs from parsed corpora (e.g. Charniak, 1996)
  - successful for PSGs, but cognitively implausible
  - no data available for us
- Unsupervised: learning from raw, unannotated corpora
  - less successful: computationally intractable in the worst case (Gold, 1967)
  - not clear how to apply to semantic problem
- Lightly supervised (latent variable supervised)
  - e.g. learn from sentences paired with Logical Form (LF)
- Plausible?
  - Shared focus of attention with others
  - 'Helpful' interaction e.g. corrective feedback (Saxton, 2010)

Semantically supervised learning

- Successfully applied to 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.
- Existing corpora annotated e.g. GeoQuery, PropBank, CHILDES
- Approach: hypothesize lexical entries which can be extended to yield the known LF

The problem

Input:

- the set of computational actions in Dynamic Syntax, G.
- 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 is the complete semantic tree representing the compositional structure of the meaning of S_i
- (we will call T_i the *target tree*)

Output:

- a grammar consisting of the possible lexical actions for each word w
- probability distributions θ_w over possible lexical actions specifying $p(a|w, T)$ in the context of a partial tree T

Simplifying Assumptions

- Assume tree operations (i.e. lambda calculus) known

Simplifying Assumptions

- Assume tree operations (i.e. lambda calculus) known
- Assume T_i is a *tree*, not a flat logical form
  - not a syntactic phrase-structure tree
  - correspondence of words *arrive* to LF elements $\lambda x.arrive'(x)$ unknown

Simplifying Assumptions

- Assume tree operations (i.e. lambda calculus) known
- Assume T_i is a *tree*, not a flat logical form
  - not a syntactic phrase-structure tree
  - correspondence of words *arrive* to LF elements $\lambda x.arrive'(x)$ unknown
- Assume lexical action probabilities conditioned only on pointed node type, and apply to only one type
  - θ_w specifies $p(a|w,T) \to p(a|w)$
  - (i.e. assume IF $?Ty(X)$; learn THEN clause as sequence of atomic actions *go*, *make*, *put*)

Outline

Lexical Actions

- Our task is to learn lexical actions:

Action

IF	$?Ty(e)$
THEN	$\mathtt{put}(Ty(e))$
	$\mathtt{put}(Fo(John'))$
	$\mathtt{put}(\langle\downarrow\rangle\bot)$
ELSE	ABORT

Input tree

$?Ty(t)$

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

$\xrightarrow{John}$

Output tree

$?Ty(t)$

$Ty(e), ?Ty(e)$, $John', \langle\downarrow\rangle\bot, \diamondsuit$ $?Ty(e \rightarrow t)$

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.
- Splitting and generalisation into possible lexical action subsequences.
- Probability estimation to keep most probable hypotheses.

Hypothesis construction

- Hypothesise extensions which subsume the target tree:

Current

$?Ty(t)$

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

Hypothesis

$?Ty(t)$

$Ty(e), ?Ty(e), John', \diamond$ $?Ty(e \rightarrow t)$

Target tree

$Ty(t), arrive'(John')$

$Ty(e), John'$ $Ty(e \rightarrow t), \lambda x.arrive'(x)$

- This is just one of many possible hypotheses . . .

Constraining hypotheses

- Constraints imposed by tree logic, lambda calculus, type constraints
- Mother nodes compatible with daughter types, formulae
- No formula decoration without type decoration
- Finite type set
- Words add semantic formulae at one node only

Constraining hypotheses

- Constraints imposed by tree logic, lambda calculus, type constraints
- Mother nodes compatible with daughter types, formulae
- No formula decoration without type decoration
- Finite type set
- Words add semantic formulae at one node only
- Package these as possible hypothesis macros:

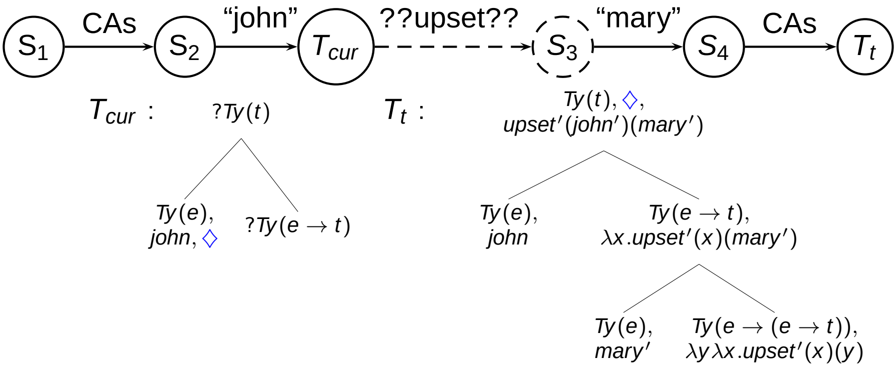
$?Ty(X), \diamondsuit$

$?Ty(e)$ $\quad$ $?Ty(e \rightarrow X)$

IF	$?Ty(X)$
	$X \neq e$
THEN	$make(\langle\downarrow_0\rangle); go(\langle\downarrow_0\rangle)$
	$put(?Ty(e)); go(\langle\uparrow\rangle)$
	$make(\langle\downarrow_1\rangle); go(\langle\downarrow_1\rangle)$
	$put(?Ty(e \rightarrow X)); go(\uparrow)$
ELSE	ABORT

Constraining hypotheses

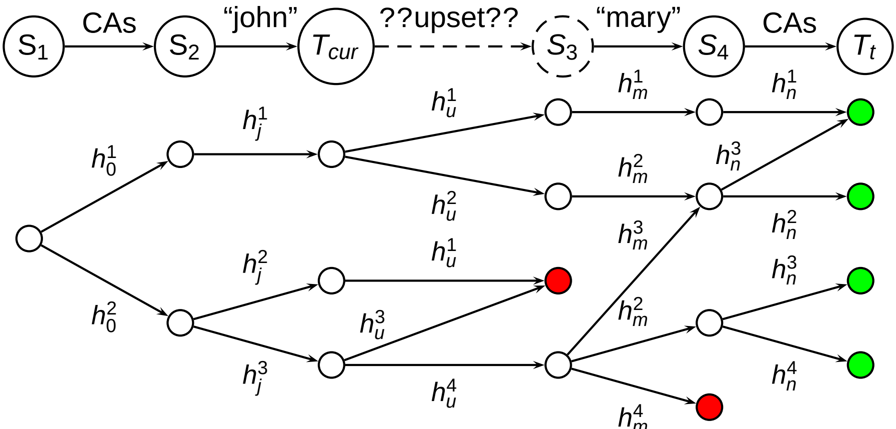
- Constrain hypotheses within DAG paths:



- Hypotheses themselves form a (finite, bounded) DAG

Constraining hypotheses

- Constrain hypotheses within DAG paths:



Outline

Splitting lexical hypotheses

- Split DAG edges into possible word sequences
  - hypothesise possible set of split points
  - constraints: one semantic decoration subsequence per word, kept to the right
- DAG edges combine lexical and computational actions
- Lexical entries should be *general*
  - apply in all desired (tree) contexts
  - consign variation in start/end point to computational actions
- Lexical entries should be *efficient*
  - constrain possible context to those observed
  - i.e. lexicalising computational actions where possible

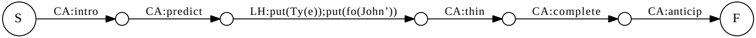
Generalisation through sequence intersection

- 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

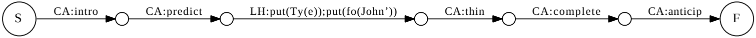
John (subject)



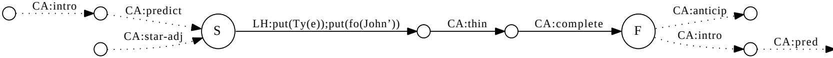
Generalisation through sequence intersection

- 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)

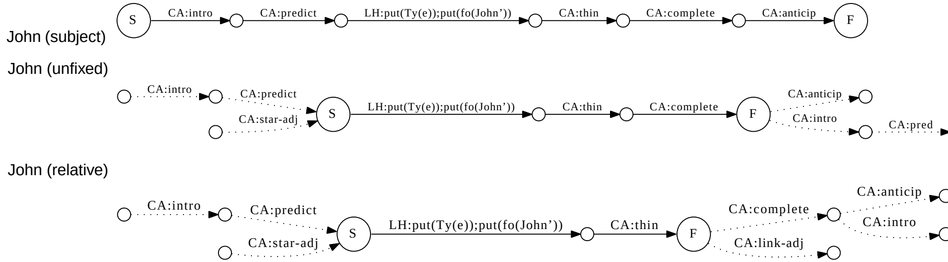


John (unfixed)



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



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

Parameter Estimation

- Assume we have a prior estimate of θ'_w giving $p(h|w)$
- Probability of DAG path sequence $p(HT_j|S)$:

$$p(HT_j|S) = \prod_{i=1}^{n} p(h_i^j|w_i) = \prod_{i=1}^{n} \theta'_{w_i}(h_i^j)$$

- Posterior estimate of $p(h|w)$:
  (summing over sequences HT_j containing h)

$$\theta''_w(h) = p(h|w) = \frac{1}{Z} \sum_{HT_j \in HT^h} p(HT_j|S) = \frac{1}{Z} \sum_{HT_j \in HT^h} \prod_{i=1}^{n} \theta'_{w_i}(h_i^j)$$

- $\theta'_w \neq \theta''_w$ – new information from hypothesis DAG

Parameter Estimation

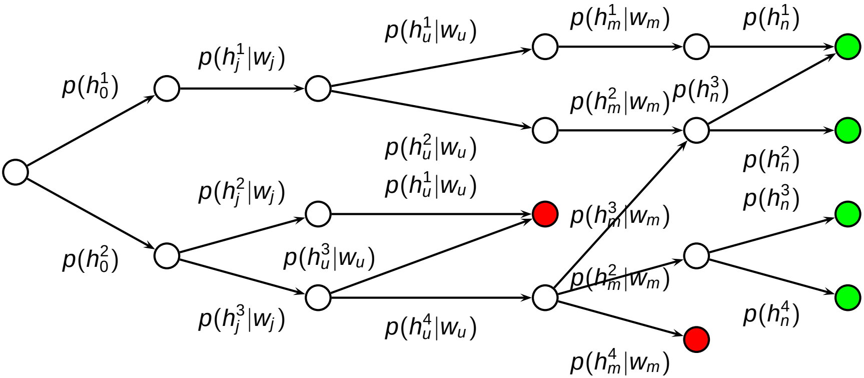
- Incremental version of Expectation-Maximisation
  - Expectation step: DAG paths from prior estimate
  - Maximisation step: re-estimate from path distribution
- Apply this *incrementally*
  - Update distributions at each training example
- Update probability distributions at each step:

$$\theta_w^N(h) = \frac{N-1}{N}\theta_w^{N-1}(h) + \frac{1}{N}\theta_w''(h)$$

- Reserve probability mass for unseen h in same way

Probabilistic Parsing

- This model will provide a probabilistic parser:



Evaluation: Artificial corpus

- Need a corpus annotated with target trees
- Easiest way: generate one using a known grammar, and try to learn it back (see e.g. Pulman & Cussens, 2001)
- Use PoS type and token distributions from CHILDES

Evaluation: Artificial corpus

- Need a corpus annotated with target trees
- Easiest way: generate one using a known grammar, and try to learn it back (see e.g. Pulman & Cussens, 2001)
- Use PoS type and token distributions from CHILDES

- 200 sentence set: 90% as training, 10% for test:

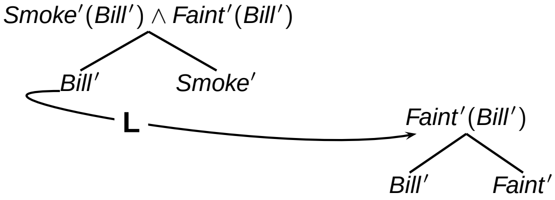
	Parsing Coverage	Same Formula
Top one	26%	77%
Top two	77%	79%
Top three	100%	80%

Evaluation: lexical ambiguity

- 10% of word types ambiguous between 2 or 3 senses
  - 57% learned both senses in top 3 hypotheses
  - but only one with both in top 2
- Data sparsity

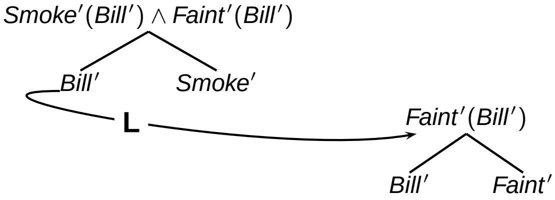
Evaluation: anaphoricity

- Allow free "copy-from-context" computational action
  - can be hypothesised at any time
- Relative pronouns: conjoined (linked) trees



Evaluation: anaphoricity

- Allow free “copy-from-context” computational action
  - can be hypothesised at any time
- Relative pronouns: conjoined (linked) trees



- Learned constraints identical to manual grammars:

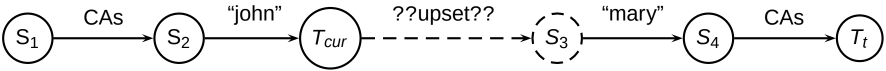
who	**IF**	$?Ty(e)$
		$\langle\uparrow_*\uparrow_L\rangle Fo(X)$
	THEN	$put(Ty(e))$
		$put(Fo(X))$
		$put(\langle\downarrow\rangle\bot)$
	ELSE	ABORT

Scaling Up

- We need to apply this to real data . . .
- Can we do it without target *trees*?
    - incremental TTR compilation allows same method

Scaling Up

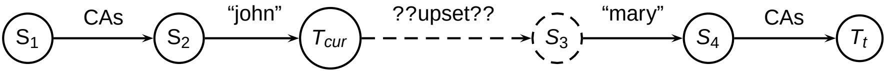
- We need to apply this to real data . . .
- Can we do it without target *trees*?
  - incremental TTR compilation allows same method



$$T_{cur} : \left[\begin{array}{lll} x_{=john} & : & e \\ p & : & t \end{array}\right] \qquad T_t : \left[\begin{array}{lll} x_{=john} & : & e \\ y_{=mary} & : & e \\ p_{=upset(x,y)} & : & t \end{array}\right]$$

Scaling Up

- We need to apply this to real data . . .
- Can we do it without target *trees*?
  - incremental TTR compilation allows same method



$$T_{cur} : \begin{bmatrix} x_{=john} & : & e \\ p & : & t \end{bmatrix} \qquad T_t : \begin{bmatrix} x_{=john} & : & e \\ y_{=mary} & : & e \\ p_{=upset(x,y)} & : & t \end{bmatrix}$$

- Can convert existing corpora (e.g. CHILDES) to TTR
- But search space increases . . .

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. Including, of course, Robin Cooper.