Introduction

Theory: Incremental Semantic Construction

Application: Incremental Reference Feedback

References

Incremental Semantic Interpretation of Dialogue Contributions

Andreas Peldszus

Institut für Linguistik, Universität Potsdam

Kolloquium Linguistik, Universität Bielefeld, 07.11.2012
Outline

1. Introduction

2. Theory: Incremental Semantic Construction

3. Application: Incremental Reference Feedback
Introduction: Spoken Dialogue Systems

ASR → NLU

DM

NLU:
- linguistic knowledge
- world knowledge
- discourse knowledge

TTS → NLG
Introduction: Incremental Processing

Incrementality, Schlangen and Skantze [2009], citing Levelt [1989]

Each processing component is triggered into activity by a minimal amount of its characteristic input.

Information flow between two modules
Introduction: Incremental Processing

Incrementality, Schlangen and Skantze [2009], citing Levelt [1989]

Each processing component is triggered into activity by a minimal amount of its characteristic input.

non-incremental information flow
Introduction: Incremental Processing

Incrementality, Schlangen and Skantze [2009], citing Levelt [1989]

Each processing component is triggered into activity by a minimal amount of its characteristic input.

incremental information flow: *early propagation*
Introduction: Incremental Processing

Incrementality, Schlangen and Skantze [2009], citing Levelt [1989]

Each processing component is triggered into activity by a minimal amount of its characteristic input.

incremental information flow: *early propagation*
Introduction: Incremental Processing

Incrementality, Schlangen and Skantze [2009], citing Levelt [1989]

Each processing component is triggered into activity by a minimal amount of its characteristic input.

incremental information flow: *early propagation*
Introduction: Incremental Processing

Incrementality, Schlangen and Skantze [2009], citing Levelt [1989]

Each processing component is triggered into activity by a minimal amount of its characteristic input.

incremental information flow: *early propagation*
Introduction: Incremental Processing

Incrementancy, Schlangen and Skantze [2009], citing Levelt [1989]

Each processing component is triggered into activity by a minimal amount of its characteristic input.

incremental information flow: *interaction / feedback*
Introduction: Incremental Processing

Incrementality, Schlangen and Skantze [2009], citing Levelt [1989]

Each processing component is triggered into activity by a minimal amount of its characteristic input.

Incremental information flow: interaction / feedback
Introduction: Incremental Processing

Incrementality, Schlangen and Skantze [2009], citing Levelt [1989]

Each processing component is triggered into activity by a minimal amount of its characteristic input.

incremental information flow: *interaction / feedback*
Introduction: Motivating Incremental SDS

1. psycho-linguistic plausibility Tanenhaus and Brown-Schmidt [2008]
   - reference resolution, early disambiguation etc

2. descriptive adequacy
   - backchannel feedback
   - intervening corrections, clarifications, completions

3. computational benefits
   - speed (by early propagation): why wait with processing?
   - accuracy (by interaction): concentrate on useful things
     - but communicative overhead
   - efficiency (by interaction): reduce workload
     - but risk of increased workload with certain utterances
Introduction: Motivating Incremental SDS

1. psycho-linguistic plausibility Tanenhaus and Brown-Schmidt [2008]
   - reference resolution, early disambiguation etc

2. descriptive adequacy
   - backchannel feedback
   - intervening corrections, clarifications, completions

3. computational benefits

But communicative overhead
- speed (by early propagation): why wait with processing?
- accuracy (by interaction): concentrate on useful things
- efficiency (by interaction): reduce workload
  - but risk of increased workload with certain utterances
Introduction: Motivating Incremental SDS

1. psycho-linguistic plausibility Tanenhaus and Brown-Schmidt [2008]
   - reference resolution, early disambiguation etc

2. descriptive adequacy
   - backchannel feedback
   - intervening corrections, clarifications, completions

3. computational benefits
Introduction: Motivating Incremental SDS

1. psycho-linguistic plausibility Tanenhaus and Brown-Schmidt [2008]
   - reference resolution, early disambiguation etc

2. descriptive adequacy
   - backchannel feedback
   - intervening corrections, clarifications, completions

3. computational benefits
   - speed (by early propagation): why wait with processing?
   - communicative overhead
   - accuracy (by interaction): concentrate on useful things
   - risk of garden-pathing
   - efficiency (by interaction): reduce workload
   - risk of increased workload with certain utterances
Introduction: Motivating Incremental SDS

1 psyco-linguistic plausibility Tanenhaus and Brown-Schmidt [2008]
   • reference resolution, early disambiguation etc

2 descriptive adequacy
   • backchannel feedback
   • intervening corrections, clarifications, completions

3 computational benefits
   • speed (by early propagation): why wait with processing?
     but communicative overhead
   • accuracy (by interaction): concentrate on useful things
     but risk of gardenpathing
   • efficiency (by interaction): reduce workload
     but risk of increased workload with certain utterances
Introduction: Motivating Incremental SDS

1. psycho-linguistic plausibility Tanenhaus and Brown-Schmidt [2008]
   - reference resolution, early disambiguation etc

2. descriptive adequacy
   - backchannel feedback
   - intervening corrections, clarifications, completions

3. computational benefits
   - speed (by early propagation): why wait with processing? but communicative overhead
   - accuracy (by interaction): concentrate on useful things but risk of gardenpathing
   - efficiency (by interaction): reduce workload but risk of increased workload with certain utterances
Introduction: Motivating Incremental SDS

1. psycho-linguistic plausibility Tanenhaus and Brown-Schmidt [2008]
   - reference resolution, early disambiguation etc

2. descriptive adequacy
   - backchannel feedback
   - intervening corrections, clarifications, completions

3. computational benefits
   - speed (by early propagation): why wait with processing?
     but communicative overhead
   - accuracy (by interaction): concentrate on useful things
     but risk of gardenpathing
   - efficiency (by interaction): reduce workload
     but risk of increased workload with certain utterances
Introduction: Motivating Incremental SDS

1. psycho-linguistic plausibility Tanenhaus and Brown-Schmidt [2008]
   - reference resolution, early disambiguation etc

2. descriptive adequacy
   - backchannel feedback
   - intervening corrections, clarifications, completions

3. computational benefits
   - speed (by early propagation): why wait with processing? but communicative overhead
   - accuracy (by interaction): concentrate on useful things but risk of gardenpathing
   - efficiency (by interaction): reduce workload but risk of increased workload with certain utterances
Introduction: Motivating Incremental SDS

1. psycho-linguistic plausibility Tanenhaus and Brown-Schmidt [2008]
   - reference resolution, early disambiguation etc

2. descriptive adequacy
   - backchannel feedback
   - intervening corrections, clarifications, completions

3. computational benefits
   - speed (by early propagation): why wait with processing? but communicative overhead
   - accuracy (by interaction): concentrate on useful things but risk of garden-pathing
   - efficiency (by interaction): reduce workload but risk of increased workload with certain utterances
Introduction: Aims of the Approach

Goal: Prune semantically implausible syntactic readings, i.e. readings without referential success.
Related Work

- Purver et al. [2011] - incremental semantic construction within Dynamic Syntax, centered on grammaticality, rather monolithic
- Poesio and Traum [1997], Poesio and Rieser [2010] - incremental semantic construction with \( \lambda \) calculus for TAG, focus on discourse structure, no implementation yet?
- Lison [2008] - incremental semantic construction with CCG, incremental statistical parse selection, no interaction, word- versus phrase-incremental?
- Hassan et al. [2009] - very fast semi-incremental (two-pass) CCG parser, no semantics given, but could be combined with Bos [2005] CCG semantics.
- Stoness et al. [2005] - incremental semantic construction with a PCFG, bottom-up parser, reference feedback, tested on only one dialogue transcript
Related Work

- Purver et al. [2011] - incremental semantic construction within Dynamic Syntax, centered on grammaticality, rather monolithic
- Poesio and Traum [1997], Poesio and Rieser [2010] - incremental semantic construction with $\lambda$ calculus for TAG, focus on discourse structure, no implementation yet?
- Lison [2008] - incremental semantic construction with CCG, incremental statistical parse selection, no interaction, word-versus phrase-incremental?
- Hassan et al. [2009] - very fast semi-incremental (two-pass) CCG parser, no semantics given, but could be combined with Bos [2005] CCG semantics.
- Stoness et al. [2005] - incremental semantic construction with a PCFG, bottom-up parser, reference feedback, tested on only one dialogue transcript
Related Work

- Purver et al. [2011] - incremental semantic construction within Dynamic Syntax, centered on grammaticality, rather monolithic.
- Poesio and Traum [1997], Poesio and Rieser [2010] - incremental semantic construction with λ calculus for TAG, focus on discourse structure, no implementation yet?
- Lison [2008] - incremental semantic construction with CCG, incremental statistical parse selection, no interaction, word- versus phrase-incremental?
- Hassan et al. [2009] - very fast semi-incremental (two-pass) CCG parser, no semantics given, but could be combined with Bos [2005] CCG semantics.
- Stoness et al. [2005] - incremental semantic construction with a PCFG, bottom-up parser, reference feedback, tested on only one dialogue transcript.
Related Work

- Purver et al. [2011] - incremental semantic construction within Dynamic Syntax, centered on grammaticality, rather monolithic
- Poesio and Traum [1997], Poesio and Rieser [2010] - incremental semantic construction with $\lambda$ calculus for TAG, focus on discourse structure, no implementation yet?
- Lison [2008] - incremental semantic construction with CCG, incremental statistical parse selection, no interaction, word-versus phrase-incremental?
- Hassan et al. [2009] - very fast semi-incremental (two-pass) CCG parser, no semantics given, but could be combined with Bos [2005] CCG semantics.
- Stoness et al. [2005] - incremental semantic construction with a PCFG, bottom-up parser, reference feedback, tested on only one dialogue transcript
Related Work

- Purver et al. [2011] - incremental semantic construction within Dynamic Syntax, centered on grammaticality, rather monolithic
- Poesio and Traum [1997], Poesio and Rieser [2010] - incremental semantic construction with \( \lambda \) calculus for TAG, focus on discourse structure, no implementation yet?
- Lison [2008] - incremental semantic construction with CCG, incremental statistical parse selection, no interaction, word- versus phrase-incremental?
- Hassan et al. [2009] - very fast semi-incremental (two-pass) CCG parser, no semantics given, but could be combined with Bos [2005] CCG semantics.
- Stoness et al. [2005] - incremental semantic construction with a PCFG, bottom-up parser, reference feedback, tested on only one dialogue transcript
Theory: Incremental Semantic Construction
Incrementality in Semantic Construction

- Goal: for an input find a semantic representation that is maximally informative and open to combination with further semantic increments
- three factors that influence *a syntax driven* incremental semantic construction
  1. type of syntactic structure building
     (top-down-parsing, bottom-up-parsing)
  2. type of syntactic structure
     (left- or right-branching)
  3. type of semantic structure interpretation
     (bottom-up, top-down)
Incrementality in Semantic Construction

- **Goal**: for an input find a semantic representation that is maximally informative and open to combination with further semantic increments

- **three factors** that influence *a syntax driven* incremental semantic construction
  1. type of syntactic structure building
     (top-down-parsing, bottom-up-parsing)
  2. type of syntactic structure
     (left- or right-branching)
  3. type of semantic structure interpretation
     (bottom-up, top-down)
Incrementality in Semantic Construction

- Goal: for an input find a semantic representation that is maximally informative and open to combination with further semantic increments
- three factors that influence a syntax driven incremental semantic construction

1. type of syntactic structure building
   (top-down-parsing, bottom-up-parsing)
2. type of syntactic structure
   (left- or right-branching)
3. type of semantic structure interpretation
   (bottom-up, top-down)
Incrementality in Semantic Construction

• Goal: for an input find a semantic representation that is maximally informative and open to combination with further semantic increments

• three factors that influence *a syntax driven* incremental semantic construction

1. type of syntactic structure building
   (top-down-parsing, bottom-up-parsing)
2. type of syntactic structure
   (left- or right-branching)
3. type of semantic structure interpretation
   (bottom-up, top-down)
Incrementality in Semantic Construction

- Goal: for an input find a semantic representation that is maximally informative and open to combination with further semantic increments
- three factors that influence *a syntax driven* incremental semantic construction
  1. type of syntactic structure building (top-down-parsing, bottom-up-parsing)
  2. type of syntactic structure (left- or right-branching)
  3. type of semantic structure interpretation (bottom-up, top-down)
Incrementality in Semantic Construction

- Goal: for an input find a semantic representation that is maximally informative and open to combination with further semantic increments
- three factors that influence *a syntax driven* incremental semantic construction
  1. type of syntactic structure building
     (top-down-parsing, bottom-up-parsing)
  2. type of syntactic structure
     (left- or right-branching)
  3. type of semantic structure interpretation
     (bottom-up, top-down)
Incrementality in Semantic Construction

- Roark [2001]: *rooted & connected* structures are essential for incremental interpretation
- usually only provided by top-down-parsers
- drawbacks: many hypotheses, problems with left-recursion
- Roark presents solutions for both problems:
Incrementality in Semantic Construction

- Roark [2001]: *rooted & connected* structures are essential for incremental interpretation
- usually only provided by top-down-parsers
- drawbacks: many hypotheses, problems with left-recursion
- Roark presents solutions for both problems:
  - grammar transformations to delay hypotheses as long as possible (left-factorisation, left-corner transform)
  - probabilistic beam search to reduce the search space
Incrementality in Semantic Construction

- Roark [2001]: *rooted & connected* structures are essential for incremental interpretation
- usually only provided by top-down-parsers
- drawbacks: many hypotheses, problems with left-recursion
- Roark presents solutions for both problems:
  - grammar transformations to delay hypotheses as long as possible (left-factorisation, left-corner transform)
  - probabilistic beam-search to reduce the search space
Incrementality in Semantic Construction

- Roark [2001]: *rooted & connected* structures are essential for incremental interpretation
- usually only provided by top-down-parsers
- drawbacks: many hypotheses, problems with left-recursion
- Roark presents solutions for both problems:
  - grammar transformations to delay hypotheses as long as possible (left-factorisation, left-corner transform)
  - probabilistic beam-search to reduce the search space

```
  S
  |  VP
  vvimp  V1
  |  NP  V1
  lösche  Art
  |  N1
  art  AdjP  N1
  |  |  |
  das  Adj  N1
  |  |  |  |
  adja  rote
```
Incrementality in Semantic Construction

- Roark [2001]: *rooted & connected* structures are essential for incremental interpretation
- usually only provided by top-down-parsers
- drawbacks: many hypotheses, problems with left-recursion
- Roark presents solutions for both problems:
  - grammar transformations to delay hypotheses as long as possible (left-factorisation, left-corner transform)
  - probabilistic beam-search to reduce the search space
Incrementality in Semantic Construction

- Roark [2001]: *rooted & connected* structures are essential for incremental interpretation
- usually only provided by top-down-parsers
- drawbacks: many hypotheses, problems with left-recursion
- Roark presents solutions for both problems:
  - grammar transformations to delay hypotheses as long as possible (left-factorisation, left-corner transform)
  - probabilistic beam-search to reduce the search space
Tree-Interpretation: Bottom up?

Tree-interpretation bottom-up, inside-out:

- need to underspecify open nodes
- need to re-interpret the whole tree every time it expands
Tree-Interpretation: Bottom up?

Tree-interpretation bottom-up, inside-out:

- need to underspecify open nodes
- need to re-interpreter the whole tree every time it expands
Tree-interpretation bottom-up, inside-out:
- need to underspecify open nodes
- need to re-interpret the whole tree every time it expands
Tree-Interpretation: Top-Down!

Tree-interpretation top-down, left-to-right:

- linearise semantic combination in parallel to the top-down expansion of the tree
- every expansion of a non-terminal node adds a rule-specific construction semantic increment (parser action: predict)
- every recognition of a terminal node adds a lexical semantic increment (parser action: match)
- no need to underspecify open nodes
- no need to re-interpret the tree
- monotonic growth, synchronised with parsing
Tree-Interpretation: Top-Down!

Tree-interpretation top-down, left-to-right:

- linearise semantic combination in parallel to the top-down expansion of the tree
- every expansion of a non-terminal node adds a rule-specific construction semantic increment (parser action: predict)
- every recognition of a terminal node adds a lexical semantic increment (parser action: match)
- no need to underspecify open nodes
- no need to re-interpret the tree
- monotonic growth, synchronised with parsing
Tree-Interpretation: Top-Down!

Tree-interpretation top-down, left-to-right:

- linearise semantic combination in parallel to the top-down expansion of the tree
- every expansion of a non-terminal node adds a rule-specific construction semantic increment (parser action: predict)
- every recognition of a terminal node adds a lexical semantic increment (parser action: match)
- no need to underspecify open nodes
- no need to re-interpret the tree
- monotonic growth, synchronised with parsing
Tree-interpretation top-down, left-to-right:

- linearise semantic combination in parallel to the top-down expansion of the tree
- every expansion of a non-terminal node adds a rule-specific construction semantic increment (parser action: predict)
- every recognition of a terminal node adds a lexical semantic increment (parser action: match)
- no need to underspecify open nodes
- no need to re-interpret the tree
- monotonic growth, synchronised with parsing
Tree-Interpretation: Top-Down!

Tree-interpretation top-down, left-to-right:

- linearise semantic combination in parallel to the top-down expansion of the tree
- every expansion of a non-terminal node adds a rule-specific construction semantic increment (parser action: predict)
- every recognition of a terminal node adds a lexical semantic increment (parser action: match)
- no need to underspecify open nodes
- no need to re-interpret the tree
- monotonic growth, synchronised with parsing
Tree-Interpretation: Top-Down!

Tree-interpretation top-down, left-to-right:

- linearise semantic combination in parallel to the top-down expansion of the tree
- every expansion of a non-terminal node adds a rule-specific construction semantic increment (parser action: predict)
- every recognition of a terminal node adds a lexical semantic increment (parser action: match)
- no need to underspecify open nodes
- no need to re-interpret the tree
- monotonic growth, synchronised with parsing
Tree-interpretation top-down, left-to-right:

- linearise semantic combination in parallel to the top-down expansion of the tree
- every expansion of a non-terminal node adds a rule-specific construction semantic increment (parser action: predict)
- every recognition of a terminal node adds a lexical semantic increment (parser action: match)
- no need to underspecify open nodes
- no need to re-interpret the tree
- *monotonic growth, synchronised with parsing*
RMRS introduction

- representation language: first order predicate logic with generalized quantifiers, sortal variables
- scope underspecification: break up into elementary predications with labels and holes, standard conjunction, add scope constraints
- underspecification of predicate argument structure: break up further into elementary predications with anchors, characteristic variable and argument-relations

Every dog chases a cat.
\(\text{every}(x, \text{dog}(x), \text{exist}(y, \text{cat}(y), \text{chase}(x,y)))\)
\(\text{exist}(y, \text{cat}(y), \text{every}(x, \text{dog}(x), \text{chase}(x,y)))\)
RMRS introduction

- representation language: first order predicate logic with
generalized quantifiers, sortal variables
- scope underspecification: break up into elementary predications
  with labels and holes, standard conjunction, add scope constraints
- underspecification of predicate argument structure: break up
  further into elementary predications with anchors, characteristic
  variable and argument-relations

\[
\langle h_0, \{ \ell_1: \text{every}(x, h_1, h_2), \\
\ell_2: \text{dog}(x), \\
\ell_3: \text{chase}(e, x, y), \\
\ell_4: \text{some}(y, h_3, h_4), \\
\ell_5: \text{cat}(y) \}, \\
\{ h_1 =_q \ell_2, h_3 =_q \ell_5 \}\rangle
\]
RMRS introduction

- representation language: first order predicate logic with generalized quantifiers, sortal variables
- scope underspecification: break up into elementary predications with labels and holes, standard conjunction, add scope constraints
- underspecification of predicate argument structure: break up further into elementary predications with anchors, characteristic variable and argument-relationships

⟨h_0,
\{\ell_1: every(x, h_1, h_2),
\ell_2: dog(x),
\ell_3: chase(e, x, y),
\ell_4: some(y, h_3, h_4),
\ell_5: cat(y) \},
\{h_1 =_q \ell_2, h_3 =_q \ell_5\}\rangle
RMRS introduction

- representation language: first order predicate logic with generalized quantifiers, sortal variables
- scope underspecification: break up into elementary predications with labels and holes, standard conjunction, add scope constraints
- underspecification of predicate argument structure: break up further into elementary predications with anchors, characteristic variable and argument-relations

\[ \langle h_0, \\
\{ \ell_1: \text{every}(x, h_1, h_2), \\
\ell_2: \text{dog}(x), \\
\ell_3: \text{chase}(e, x, y), \\
\ell_4: \text{some}(y, h_3, h_4), \\
\ell_5: \text{cat}(y) \}, \\
\{ h_1 =_q \ell_2, h_3 =_q \ell_5 \} \rangle \]
RMRS introduction

- representation language: first order predicate logic with generalized quantifiers, sortal variables
- scope underspecification: break up into elementary predications with labels and holes, standard conjunction, add scope constraints
- underspecification of predicate argument structure: break up further into elementary predications with anchors, characteristic variable and argument-relations

\[ \langle h_0, \{ \ell_1:\text{every}(x, h_1, h_2), \ell_2:\text{dog}(x), \ell_3:\text{chase}(e, x, y), \ell_4:\text{some}(y, h_3, h_4), \ell_5:\text{cat}(y) \}, \{ h_1 = q \ell_2, h_3 = q \ell_5 \} \rangle \]
RMRS introduction

- representation language: first order predicate logic with generalized quantifiers, sortal variables
- scope underspecification: break up into elementary predications with labels and holes, standard conjunction, **add scope constraints**
- underspecification of predicate argument structure: break up further into elementary predications with anchors, characteristic variable and argument-relations

\[
\langle h_0, \\
\{ \ell_1 : \text{every}(x, h_1, h_2), \\
\ell_2 : \text{dog}(x), \\
\ell_3 : \text{chase}(e, x, y), \\
\ell_4 : \text{some}(y, h_3, h_4), \\
\ell_5 : \text{cat}(y) \}, \\
\{ h_1 =_q \ell_2, h_3 =_q \ell_5 \} \rangle
\]
RMRS introduction

- representation language: first order predicate logic with generalized quantifiers, sortal variables
- scope underspecification: break up into elementary predications with labels and holes, standard conjunction, add scope constraints
- underspecification of predicate argument structure: break up further into elementary predications with anchors, characteristic variable and argument-relations

\[
\langle h_0, \\
\{ \ell_1:a_1:every(), BV(a_1, x_1), RSTR(a_1, h_1), BODY(a_1, h_2), \ell_2:a_2:dog(x_1), \ell_3:a_3:chase(e_1), ARG_1(a_3, x_1), ARG_2(a_3, x_2), \ell_4:a_4:some(), BV(a_4, x_2), RSTR(a_4, h_3), BODY(a_4, h_4), \ell_5:a_5:cat(x_2) \}, \\
\{ h_1 =_q \ell_2, h_3 =_q \ell_5 \} \rangle
\]
RMRS introduction

- representation language: first order predicate logic with generalized quantifiers, sortal variables
- scope underspecification: break up into elementary predications with labels and holes, standard conjunction, add scope constraints
- underspecification of predicate argument structure: break up further into elementary predications with anchors, characteristic variable and argument-relations

\[
\langle h_0, \\
\{ \ell_1:a_1:every(), BV(a_1, x_1), RSTR(a_1, h_1), BODY(a_1, h_2), \\
\ell_2:a_2:dog(x_1), \\
\ell_3:a_3:chase(e_1), ARG_1(a_3, x_1), ARG_2(a_3, x_2), \\
\ell_4:a_4:some(), BV(a_4, x_2), RSTR(a_4, h_3), BODY(a_4, h_4), \\
\ell_5:a_5:cat(x_2) \}, \\
\{ h_1 =_q \ell_2, h_3 =_q \ell_5 \} \rangle
\]
RMRS introduction

- representation language: first order predicate logic with generalized quantifiers, sortal variables
- scope underspecification: break up into elementary predications with labels and holes, standard conjunction, add scope constraints
- underspecification of predicate argument structure: break up further into elementary predications with anchors, characteristic variable and argument-relations

\[
\langle h_0, \\
\{ \ell_1 : a_1 : \text{every}(), \ BV(a_1, x_1), \ RSTR(a_1, h_1), \ BODY(a_1, h_2), \\
\ell_2 : a_2 : \text{dog}(x_1), \\
\ell_3 : a_3 : \text{chase}(e_1), \ ARG_1(a_3, x_1), \ ARG_2(a_3, x_2), \\
\ell_4 : a_4 : \text{some}(), \ BV(a_4, x_2), \ RSTR(a_4, h_3), \ BODY(a_4, h_4), \\
\ell_5 : a_5 : \text{cat}(x_2) \}, \\
\{ h_1 = q \ell_2, \ h_3 = q \ell_5 \} \rangle
\]
RMRS introduction

- representation language: first order predicate logic with generalized quantifiers, sortal variables
- scope underspecification: break up into elementary predications with labels and holes, standard conjunction, add scope constraints
- underspecification of predicate argument structure: break up further into elementary predications with anchors, characteristic variable and argument-relations

\[
\langle h_0, \{ \ell_1:a_1:every(), BV(a_1, x_1), RSTR(a_1, h_1), BODY(a_1, h_2), \ell_2:a_2:dog(x_1), \ell_3:a_3:chase(e_1), ARG_1(a_3, x_1), ARG_2(a_3, x_2), \ell_4:a_4:some(), BV(a_4, x_2), RSTR(a_4, h_3), BODY(a_4, h_4), \ell_5:a_5:cat(x_2) \}, \{ h_1 =_q \ell_2, h_3 =_q \ell_5 \} \rangle
\]
RMRS forward composition

- classic RMRS Copestake [2007] uses a set of named slot to represent the open positions, the syntax-semantic interface picks the right slot to fill
  - only one slot of each sort open
  - however, left recursive rules in incremental processing may require to have multiple slots of the same sort open
  - therefore we reinterpret the slots as a stack of unnamed slots, where only the top element can be filled
RMRS forward composition

- classic RMRS Copestake [2007] uses a set of named slot to represent the open positions, the syntax-semantic interface picks the right slot to fill
- only one slot of each sort open
  - however, left recursive rules in incremental processing may require to have multiple slots of the same sort open
  - therefore we reinterpret the slots as a stack of unnamed slots, where only the top element can be filled
RMRS forward composition

- classic RMRS Copestake [2007] uses a set of named slot to represent the open positions, the syntax-semantic interface picks the right slot to fill
- only one slot of each sort open
  - however, left recursive rules in incremental processing may require to have multiple slots of the same sort open
  - therefore we reinterpret the slots as a stack of unnamed slots, where only the top element can be filled

\[
\]

\[
l_1:a_1:every(), BV(a_1,x_1), RSTR(a_1,h_1), BODY(a_1,h_2), h_1 = q l_2,
\]

\[
l_2:a_2:dog(x_1),
\]

\[
l_3:a_3:chase(e_1), ARG_1(a_3,x_1), ARG_2(a_3,x_2),
\]

\[
l_4:a_4:some(), BV(a_4,x_2), RSTR(a_4,h_3), BODY(a_4,h_4), h_3 = q l_5
\]

\[
\]
RMRS forward composition

- classic RMRS Copestake [2007] uses a set of named slot to represent the open positions, the syntax-semantic interface picks the right slot to fill
- only one slot of each sort open
- however, left recursive rules in incremental processing may require to have multiple slots of the same sort open
- therefore we reinterpret the slots as a stack of unnamed slots, where only the top element can be filled

\[
\begin{align*}
\ell_3 &: a_3 : e_1 & \{ & \} \\
\ell_1 &: a_1 : every() & , & BV(a_1, x_1) , & RSTR(a_1, h_1) , & BODY(a_1, h_2) , & h_1 = q \ \ell_2 , \\
\ell_2 &: a_2 : \text{dog}(x_1) , \\
\ell_3 &: a_3 : \text{chase}(e_1) , & ARG_1(a_3, x_1) , & ARG_2(a_3, x_2) , \\
\ell_4 &: a_4 : \text{some}() , & BV(a_4, x_2) , & RSTR(a_4, h_3) , & BODY(a_4, h_4) , & h_3 = q \ \ell_5 \\
\ell_6 &: a_6 : \text{cat}(x_6) , & \ell_6 = \ell_5 , & a_6 = a_5 , & x_6 = x_2
\end{align*}
\]
RMRS forward composition

- classic RMRS Copestake [2007] uses a *set of named slot* to represent the open positions, the syntax-semantic interface picks the right slot to fill
- only one slot of each sort open
- however, left recursive rules in incremental processing may require to have multiple slots of the same sort open
- therefore we reinterpret the slots as a stack of unnamed slots, where only the top element can be filled

\[
[\ell_3: a_3: e_1] \{ \}
\ell_1: a_1: every(), BV(a_1, x_1), RSTR(a_1, h_1), BODY(a_1, h_2), h_1 = q \ell_2,
\ell_2: a_2: dog(x_1),
\ell_3: a_3: chase(e_1), ARG_1(a_3, x_1), ARG_2(a_3, x_2),
\ell_4: a_4: some(), BV(a_4, x_2), RSTR(a_4, h_3), BODY(a_4, h_4), h_3 = q \ell_5
\ell_5: a_5: cat(x_2)
\]
RMRS forward composition

- classic RMRS Copestake [2007] uses a set of named slot to represent the open positions, the syntax-semantic interface picks the right slot to fill
- only one slot of each sort open
- however, left recursive rules in incremental processing may require to have multiple slots of the same sort open
- therefore we reinterpret the slots as a stack of unnamed slots, where only the top element can be filled
RMRS forward composition

- classic RMRS Copestake [2007] uses a set of named slot to represent the open positions, the syntax-semantic interface picks the right slot to fill
- only one slot of each sort open
- however, left recursive rules in incremental processing may require to have multiple slots of the same sort open
- therefore we reinterpret the slots as a stack of unnamed slots, where only the top element can be filled
**RMRS forward composition**

**RMRS structure under construction with a stack of slots**

An RMRS structure under construction is a 6-tuple \( \langle GT, H, S, R, C, E \rangle \),

- with \( GT \) the global top hole \( h_0 \),
- with \( H \) the hook \([\ell:a:i]\) consisting of the local top label \( \ell \), the anchor \( a \) and the index \( i \),
- with \( S \) the stack of slots of the form \([\ell_n:a_n:i_n]\),
- with \( R \) the bag of EPS and argument relations,
- with \( C \) the bag of constraints and
- with \( E \) the set of variable equalities.
RMRS forward composition

Forward slot filling combination

Given two RMRSs, one being the functor
\[ rmrs_f = \langle GT_f, H_f, S_f, R_f, C_f, E_f \rangle \]
with the top slot \( \text{top}(S_f) = [\ell_f:a_f:i_f] \) and one being the argument
\[ rmrs_a = \langle GT_a, H_a, S_a, R_a, C_a, E_a \rangle \]
with its hook \( H_a = [\ell_a:a_a:i_a] \), the slot filling combination \( rmrs_f \triangleleft rmrs_a \) yields an RMRS
\[ rmrs = \langle GT, H, S, R, C, E \rangle \]

- \( GT = GT_f = GT_a \)
- \( H = H_f \)
- \( S = \text{merge-stacks}(S_a, \text{pop}(S_f)) \)
- \( R = R_f \cup R_a \)
- \( C = C_f \cup C_a \)
- \( E = E_f \cup E_a \cup \{\ell_f = \ell_a, a_f = a_a, i_f = i_a\} \)
Basic slotfilling combinators

\[
\begin{align*}
[\cdot] &= \llbracket \ell:a:u \rrbracket \{ \} . \\
[\circ] &= \llbracket \ell:a:u \rrbracket \{ \llbracket \ell:a:u \rrbracket \} . \\
[\equiv] &= \llbracket \ell:a:u \rrbracket \{ \llbracket \ell:a:u \rrbracket \llbracket \ell:a:u \rrbracket \} . \\
[+] &= \llbracket \ell:a:u \rrbracket \{ \llbracket \ell:a:u \rrbracket \llbracket \ell:a:u \rrbracket \} . \\
[\vdash] &= \llbracket \ell:a:u \rrbracket \{ \llbracket \ell:a:u \rrbracket \llbracket \ell:a:u \rrbracket \} . \\
[+\ell] &= \llbracket \ell:a:u \rrbracket \{ \llbracket \ell:a:u \rrbracket \llbracket \ell:a:u \rrbracket \} .
\end{align*}
\]
### A worked example: a toy grammar

<table>
<thead>
<tr>
<th>Non-terminal</th>
<th>Rule 1</th>
<th>Rule 2</th>
<th>Rule 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>$\rightarrow$ S kon S</td>
<td>[Conj]</td>
<td></td>
</tr>
<tr>
<td>S</td>
<td>$\rightarrow$ VP</td>
<td>[Arg1] $\triangleleft$ [adr]</td>
<td></td>
</tr>
<tr>
<td>VP</td>
<td>$\rightarrow$ vvimp V1</td>
<td>[=]</td>
<td></td>
</tr>
<tr>
<td>V1</td>
<td>$\rightarrow$ NP0 V1</td>
<td>[Arg2]</td>
<td></td>
</tr>
<tr>
<td>V1</td>
<td>$\rightarrow$ V1 AdvP</td>
<td>[+]</td>
<td></td>
</tr>
<tr>
<td>V1</td>
<td>$\rightarrow$ V1 PP</td>
<td>[+]</td>
<td></td>
</tr>
<tr>
<td>V1</td>
<td>$\rightarrow$ $\epsilon$</td>
<td>[-]</td>
<td></td>
</tr>
<tr>
<td>PP</td>
<td>$\rightarrow$ appr NP</td>
<td>[PP]</td>
<td></td>
</tr>
<tr>
<td>AdvP</td>
<td>$\rightarrow$ adv</td>
<td>[Adv]</td>
<td></td>
</tr>
<tr>
<td>AdjP</td>
<td>$\rightarrow$ adja</td>
<td>[Adj]</td>
<td></td>
</tr>
<tr>
<td>NP0</td>
<td>$\rightarrow$ PP NP0</td>
<td>[+ ]</td>
<td></td>
</tr>
<tr>
<td>NP0</td>
<td>$\rightarrow$ NP</td>
<td>[○ ]</td>
<td></td>
</tr>
<tr>
<td>NP</td>
<td>$\rightarrow$ pper</td>
<td>[○ ]</td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>$\rightarrow$ art N2</td>
<td>[Q]</td>
<td></td>
</tr>
<tr>
<td>N2</td>
<td>$\rightarrow$ N1 NP</td>
<td>[+]</td>
<td></td>
</tr>
<tr>
<td>N1</td>
<td>$\rightarrow$ AdjP N1</td>
<td>[+l]</td>
<td></td>
</tr>
<tr>
<td>N1</td>
<td>$\rightarrow$ N1 PP</td>
<td>[+l]</td>
<td></td>
</tr>
<tr>
<td>N1</td>
<td>$\rightarrow$ N1 AdvP</td>
<td>[+l]</td>
<td></td>
</tr>
<tr>
<td>N1</td>
<td>$\rightarrow$ nn</td>
<td>[○ ]</td>
<td></td>
</tr>
</tbody>
</table>
A worked example: semantic macros

\[
\begin{align*}
[\text{Arg1}] &= [l:a:u] \{ [l_1:a_1:x_1][l:a:u] \} \text{ARG}_1(a, x_1) \\
[\text{Arg2}] &= [l:a:u] \{ [l_1:a_1:x_1][l:a:u] \} \text{ARG}_2(a, x_1) \\
[\text{Arg3}] &= [l:a:u] \{ [l_1:a_1:x_1][l:a:u] \} \text{ARG}_3(a, x_1) \\
[\text{adr}] &= [l:a:x] \{ \} \ l:a:\text{addressee}(x) \\
[\text{Q}] &= [l:a:x] \{ [l:a:e_1][l_2:a_2:x] \} \text{BV}(a, x), \\
& \quad \text{RSTR}(a, h_1), \text{BODY}(a, h_2), h_1 =_q \ell_2 \\
[\text{PP}] &= [l:a:u] \{ [l_1:a_1:e_1][l_2:a_2:x_2] \} \text{ARG}_1(a_1, u), \\
& \quad \text{ARG}_2(a_1, x_2) \\
[\text{Adj}] &= [l:a:x] \{ [l:a:e_1] \} \text{ARG}_1(a_1, x) \\
[\text{Adv}] &= [l:a:u] \{ [l_1:a_1:e_1] \} \text{ARG}_1(a_1, u) \\
& \quad \text{LEFT}_i(a, u_1), \text{LEFT}_\ell(a, h_1), \\
& \quad \text{RIGHT}_i(a, u_3), \text{RIGHT}_\ell(a, h_3), \\
& \quad h_1 =_q \ell_1, h_3 =_q \ell_3
\end{align*}
\]
A worked example: generic lexical entries

<table>
<thead>
<tr>
<th>Part</th>
<th>Entry</th>
<th>Lemma</th>
</tr>
</thead>
<tbody>
<tr>
<td>adja</td>
<td>([l: a: e])</td>
<td>(\ell:a:\text{lemma}(e))</td>
</tr>
<tr>
<td>adv</td>
<td>([l: a: e])</td>
<td>(\ell:a:\text{lemma}(e))</td>
</tr>
<tr>
<td>appr</td>
<td>([l: a: e])</td>
<td>(\ell:a:\text{lemma}(e))</td>
</tr>
<tr>
<td>art</td>
<td>([l: a: u])</td>
<td>(\ell:a:\text{lemma}())</td>
</tr>
<tr>
<td>kon</td>
<td>([l: a: u])</td>
<td>(\ell:a:\text{lemma}(u))</td>
</tr>
<tr>
<td>nn</td>
<td>([l: a: x])</td>
<td>(\ell:a:\text{lemma}(x))</td>
</tr>
<tr>
<td>pper</td>
<td>([l: a: x])</td>
<td>(\ell:a:\text{pper}(x))</td>
</tr>
<tr>
<td>vvimp</td>
<td>([l: a: e])</td>
<td>(\ell:a:\text{lemma}(e))</td>
</tr>
</tbody>
</table>
A worked example: derivation

[\circ]
\triangleright [\text{Arg1}] \triangleright [\text{adr}] \triangleright [\] \triangleright [\text{lösche}]
\triangleright [\text{Arg2}] \triangleright [\circ] \triangleright [\text{Q}] \triangleright [\text{das}]
\triangleright [\circ] \triangleright [+\ell] \triangleright [\text{Adj}] \triangleright [\text{rote}]
\triangleright [\circ] \triangleright [\text{kreuz}]
\triangleright [-]

[\ell_0:a_0:e_0]\{\}
\ell_0:a_0:_löschen(e_0), \text{ARG}_1(a_0, x_2), \text{ARG}_2(a_0, x_4),
\ell_2:a_2:_adresssee(x_2),
\ell_4:a_4:_def_q(), \text{BV}(a_4, x_4), \text{RSTR}(a_4, h_1), \text{BODY}(a_4, h_2), h_1 =_q \ell_7,
\ell_7:a_{10}:_rot(e_{10}), \text{ARG}_1(a_{10}, x_4),
\ell_7:a_{7}:_kreuz(x_4)
A worked example: derivation

\[
\begin{align*}
S \quad & \quad [\circ] \\
\cdots \\
[l_0 : a_0 : u_0] \quad & \quad \{ [l_0 : a_0 : u_0] \} .
\end{align*}
\]
A worked example: derivation

\[ S \]
\[ \mathit{VP} \]
\[ \ldots \]

\[
\begin{array}{l}
\circ \\
\backslash \text{Arg1}\\
\{ l_0: a_0: u_0 \} \{ l_2: a_2: x_2 \} \}
\end{array}
\]

\[ \text{ARG}_1(a_0, x_2) \]
A worked example: derivation

\[
\begin{align*}
S & \quad [\circ] \\
VP & \quad [\text{Arg1}] \quad [\text{adr}] \\
\ldots & \\
[\ell_0:a_0:u_0] & \{ [\ell_0:a_0:u_0] \} \\
\text{ARG}_1(a_0, x_2), \quad \ell_2:a_2:\text{addressee}(x_2)
\end{align*}
\]
A worked example: derivation

\[ S \]

\[ V \]

\[ vvimp \]

\[ V1 \]

\[ [\circ] \]

\[ \triangleleft [\text{Arg1}] \triangleleft [\text{adr}] \triangleleft [\text{=}] \]

\[ [\ell_0:a_0:u_0] \{ [\ell_0:a_0:u_0]\ell_0:a_0:u_0] \}

\[ \text{ARG}_1(a_0, x_2), \ell_2:a_2:\text{addressee}(x_2) \]
A worked example: derivation

\[
S \\
\quad \Downarrow \text{VP} \\
\quad \quad \text{vvimp} \quad \text{V1} \\
\quad \quad \quad \text{lösche} \quad \ldots
\]

\[
[\circ] \\
\Downarrow [\text{Arg1}] \Downarrow [\text{adr}] \Downarrow [=] \Downarrow [\text{lösche}]
\]

\[
[\ell_0:a_0:e_0] \{ [\ell_0:a_0:e_0] \}
\ell_0:a_0:_löschen(e_0), \text{ARG}_1(a_0,x_2),
\ell_2:a_2:\text{addressee}(x_2)
\]
A worked example: derivation

[S]
    \[\triangleright [\text{Arg1}] \triangleright [\text{adr}] \triangleright [=] \triangleright [\text{lösche}]\]
    \[\triangleright [\text{Arg2}]\]

\[\ell_0:a_0:e_0\] \{ \[\ell_4:a_4:x_4\][\ell_0:a_0:e_0]\}\]
\[\ell_0:a_0:löschen(e_0), \text{ARG}_1(a_0, x_2), \text{ARG}_2(a_0, x_4), \]
\[\ell_2:a_2:addresssee(x_2)\]
A worked example: derivation

\[ S \]
\[ \text{vpimp} \]
\[ \text{lösche} \]
\[ \text{VP} \]
\[ \text{V1} \]
\[ \text{NP} \]
\[ \ldots \]

\[ [\circ] \]
\[ \triangleright [\text{Arg1}] \triangleright [\text{adr}] \triangleright \ldots \triangleright [\text{lösche}] \]
\[ \triangleright [\text{Arg2}] \triangleright [\circ] \]

\[ [l_0:a_0:e_0] \{ [l_4:a_4:x_4][l_0:a_0:e_0] \} \]
\[ l_0:a_0:_löschen(e_0), \text{ARG}_1(a_0,x_2), \text{ARG}_2(a_0,x_4), \]
\[ l_2:a_2:adresssee(x_2) \]
A worked example: derivation

\[
[S \\
| VP \\
  \vdash vvim \quad V1 \\
  | NP0 \quad V1 \\
  \vdash lösch \quad NP \quad N2 \\
  \vdots]
\]

\[
[\circ] \\
\vdash [\text{Arg1}] \vdash [\text{adr}] \vdash [=] \vdash [\text{lösche}] \\
\vdash [\text{Arg2}] \vdash [\circ] \vdash [Q]
\]

\[
[l_0:a_0:e_0] \{ [l_4:a_4:e_6][l_7:a_7:x_4][l_0:a_0:e_0] \} \\
l_0:a_0:_löschen(e_0), \text{ARG}_1(a_0,x_2), \text{ARG}_2(a_0,x_4), \\
l_2:a_2:_addressee(x_2), \\
BV(a_4,x_4), \text{RSTR}(a_4,h_1), \text{BODY}(a_4,h_2), h_1 = q \; l_7
\]
A worked example: derivation

\[
S \xrightarrow{VP} vvimp \xrightarrow{lösehe} \xrightarrow{NP0} \xrightarrow{V1} \xrightarrow{NP} art \xrightarrow{V1} \xrightarrow{NP} ...
\]

\[
[\circ] \\
≡ [Arg1] \equiv [adr] \equiv [\_] \equiv [lösehe] \\
≡ [Arg2] \equiv [\circ] \equiv [Q] \equiv [das]
\]

\[
[l_0:a_0:e_0] \{ [l_7:a_7:x_4][l_0:a_0:e_0] \} \\
l_0:a_0:_löschen(e_0), ARG_1(a_0, x_2), ARG_2(a_0, x_4), \\
l_2:a_2:adresse(x_2), \\
l_4:a_4:_def_q(), BV(a_4, x_4), RSTR(a_4, h_1), BODY(a_4, h_2), h_1 =_q l_7
\]
A worked example: derivation

\[ S \]
\[ VP \]
\[ \text{vvimp} \]
\[ lösche \]
\[ NP0 \]
\[ NP \]
\[ \text{art} \]
\[ das \]
\[ V1 \]
\[ ... \]

\[ [ \circ ] \]
\[ \text{Arg1} \]
\[ \text{adr} \]
\[ = \]
\[ [lösche] \]
\[ \text{Arg2} \]
\[ [ \circ ] \]
\[ [Q] \]
\[ [das] \]
\[ [ \circ ] \]

\[ [l_0:a_0:e_0] \{ [l_7:a_7:x_4][l_0:a_0:e_0] \} \]
\[ l_0:a_0:\text{löschen}(e_0), \text{ARG}_1(a_0,x_2), \text{ARG}_2(a_0,x_4), \]
\[ l_2:a_2:\text{addressee}(x_2), \]
\[ l_4:a_4:\text{def_q}(), \text{BV}(a_4,x_4), \text{RSTR}(a_4,h_1), \text{BODY}(a_4,h_2), h_1 =_q l_7 \]
A worked example: derivation

\[
S \\
| \\
VP \\
| \\
nvimp
\]

\[
VVNP0 \\
| \\
V1
\]

\[
NP \\
| \\
NP \\
| \\
NP0
\]

\[
lösch\mathcal{e} \\
| \\
N2
\]

\[
\text{art} \\
| \\
N1
\]

\[
\text{AdjP} \\
| \\
\ldots
\]

\[
[\circ] \\
\triangleleft [\text{Arg1}] \triangleleft [\text{adr}] \triangleleft [\text{=}] \triangleleft [[lösch\mathcal{e}]] \\
\triangleleft [\text{Arg2}] \triangleleft [\circ] \triangleleft [\text{Q}] \triangleleft [[d\mathcal{a}s]] \\
\triangleleft [\circ] \triangleleft [+\ell]
\]

\[
[\ell_0:a_0:e_0] \{ [\ell_7:a_8:x_4][\ell_7:a_7:x_4][\ell_0:a_0:e_0] \} \\
\ell_0:a_0:_{\text{l"osen}}(e_0), \text{ARG}_1(a_0,x_2), \text{ARG}_2(a_0,x_4), \\
\ell_2:a_2:_{\text{address}}(x_2), \\
\ell_4:a_4:_{\text{def}_q}(), \text{BV}(a_4,x_4), \text{RSTR}(a_4,h_1), \text{BODY}(a_4,h_2), h_1 =_q \ell_7
\]
A worked example: derivation

\[
S \\
\text{VP} \\
\text{vvimp} \quad \text{V1} \\
\text{lösch} \quad \text{NP0} \quad \text{V1} \\
\text{NP} \\
\text{art} \quad \text{N2} \\
\text{das} \quad \text{N1} \\
\text{AdjP} \quad \text{N1} \\
\text{adja} \\
\ldots
\]

\[
[\circ] \quad \Delta [\text{Arg1}] \quad [\text{adr}] \quad [\text{adr}] \quad [\text{lösch}] \\
\Delta [\text{Arg2}] \quad [\circ] \quad [\text{Q}] \quad [\text{das}] \\
\Delta [\circ] \quad [+\ell] \quad [\text{Adj}]
\]

\[
[\ell_0:a_0:e_0] \quad \{ [\ell_7:a_{10}:e_{10}][\ell_7:a_7:x_4][\ell_0:a_0:e_0] \} \\
\ell_0:a_0:\text{lösch}en(e_0), \text{ARG}_1(a_0, x_2), \text{ARG}_2(a_0, x_4), \text{ARG}_1(a_{10}, x_4)
\]

\[
[\ell_0:a_0:e_0] \quad \{ [\ell_7:a_{10}:e_{10}][\ell_7:a_7:x_4][\ell_0:a_0:e_0] \} \\
\ell_0:a_0:\text{lösch}en(e_0), \text{ARG}_1(a_0, x_2), \text{ARG}_2(a_0, x_4), \text{ARG}_1(a_{10}, x_4)
\]

\[
\ell_0:a_0:\text{lösch}en(e_0), \text{ARG}_1(a_0, x_2), \text{ARG}_2(a_0, x_4), \text{ARG}_1(a_{10}, x_4)
\]

Incremental Semantic Interpretation of Dialogue Contributions

Peldszus
A worked example: derivation

\[
\begin{align*}
S & \\
| VP & \\
\vdash vvimp & V1 \\
| lösche & NP0 \quad V1 \\
| NP & \\
\vdash art & N2 \\
| das & N1 \\
| AdjP & N1 \\
| adj & rote
\end{align*}
\]

\[
[\circ] \\
\setminus [\text{Arg1}] \setminus [\text{adr}] \setminus [\text{=}] \setminus [\text{lösch}] \\
\setminus [\text{Arg2}] \setminus [\circ] \setminus [\text{Q}] \setminus [\text{das}] \\
\setminus [\circ] \setminus [+\ell] \setminus [\text{Adj}] \setminus [\text{rote}]
\]

\[
\begin{align*}
[l_0:a_0:e_0] & \{ [l_7:a_7:x_4][l_0:a_0:e_0] \} \\
l_0:a_0:_{löschen}(e_0), & \text{ARG}_1(a_0, x_2), \text{ARG}_2(a_0, x_4), \\
l_2:a_2:_{adresssee}(x_2), & \\
l_4:a_4:_{def.q()}, & \text{BV}(a_4, x_4), \text{RSTR}(a_4, h_1), \text{BODY}(a_4, h_2), h_1 = q \ell_7, \\
l_7:a_{10:rot}(e_{10}), & \text{ARG}_1(a_{10}, x_4)
\end{align*}
\]
A worked example: derivation

[○]
\[\text{Arg1} \triangleleft \text{adr} \triangleleft [=] \triangleleft [\text{lösche}]\]
\[\text{Arg2} \triangleleft [\circ] \triangleleft [\text{Q}] \triangleleft [\text{das}]\]
\[\circ \triangleleft [+\ell] \triangleleft \text{[Adj]} \triangleleft [\text{rote}]\]
\[\circ\]

\[\ell_0:a_0:e_0 \} \{ \ell_7:a_7:x_4[\ell_0:a_0:e_0] \}
\ell_0:a_0:_{\text{löschen}}(e_0), \text{ARG}_1(a_0,x_2), \text{ARG}_2(a_0,x_4),
\ell_2:a_2:_{\text{addressee}}(x_2),
\ell_4:a_4:_{\text{def}_q}(), \text{BV}(a_4,x_4), \text{RSTR}(a_4,h_1), \text{BODY}(a_4,h_2), h_1 = q \ell_7,
\ell_7:a_{10}:_{\text{rot}}(e_{10}), \text{ARG}_1(a_{10},x_4)\]
A worked example: derivation

\begin{itemize}
  \item \[ [\circ] \]
  \item \[ [\text{Arg1}] \triangleleft [\text{adr}] \triangleleft [=] \triangleleft [\text{lösche}] \]
  \item \[ [\text{Arg2}] \triangleleft [\circ] \triangleleft [Q] \triangleleft [\text{das}] \]
  \item \[ [\circ] \triangleleft [+\ell] \triangleleft [\text{Adj}] \triangleleft [\text{rote}] \]
  \item \[ [\circ] \triangleleft [\text{kreuz}] \]
\end{itemize}

\[ [\ell_0:a_0:e_0] \{ [\ell_0:a_0:e_0] \} \]
\[ \ell_0:a_0:löschen(e_0), \text{ARG}_1(a_0,x_2), \text{ARG}_2(a_0,x_4), \]
\[ \ell_2:a_2:addresssee(x_2), \]
\[ \ell_4:a_4:.getvalue(), \text{BV}(a_4,x_4), \text{RSTR}(a_4,h_1), \text{BODY}(a_4,h_2), h_1 = q \ell_7, \]
\[ \ell_7:a_{10}:\text{rot}(e_{10}), \text{ARG}_1(a_{10},x_4), \]
\[ \ell_7:a_7:\text{kreuz}(x_4) \]
A worked example: derivation

```
[ vvimpt ]
  ▲ [Arg1] ▲ [adr] ▲ [=] ▲ [lösche]  ▲ [Arg2] ▲ [o] ▲ [Q] ▲ [das]  ▲ [o] ▲ [Adj] ▲ [rote]  ▲ [-]  

[\ell_0:a_0:e_0] \{ \}  
  \ell_0:a_0:_löschenn(e_0), ARG_1(a_0,x_2), ARG_2(a_0,x_4),  
  \ell_2:a_2:adressee(x_2),  
  \ell_4:a_4:_def_q(), BV(a_4,x_4), RSTR(a_4,h_1), BODY(a_4,h_2), h_1=Q\ell_7,  
  \ell_7:a_{10}:_rot(e_{10}), ARG_1(a_{10},x_4),  
  \ell_7:a_7:_kreuz(x_4)  
```
Further work

Grammar transformation can increase the parsing performance:

1. **left-factored grammars**
   - delay rule identification for rules with equal first child
   - transform semantic rules accordingly (in theory yes, automatically no)
   - no further refinements to the semantic construction needed

2. **left-corner transformed grammars**
Further work

Grammar transformation can increase the parsing performance:

1. left-factored grammars
   - delay rule identification for rules with equal first childs
   - transform semantic rules accordingly (in theory yes, automatically no)
   - no further refinements to the semantic construction needed

2. left-corner transformed grammars
Further work

Grammar transformation can increase the parsing performance:

1. left-factored grammars
   - delay rule identification for rules with equal first child
   - transform semantic rules accordingly (in theory yes, automatically no)
   - no further refinements to the semantic construction needed

2. left-corner transformed grammars
   - delay attachment decisions
   - transform semantic rules accordingly (in theory perhaps, automatically no)
   - augment semantic construction with backward function composition in nested evaluation contexts (done)
Further work

Grammar transformation can increase the parsing performance:

1. left-factored grammars
   - delay rule identification for rules with equal first childs
   - transform semantic rules accordingly (in theory yes, automatically no)
   - no further refinements to the semantic construction needed

2. left-corner transformed grammars
   - delay attachment decisions
   - transform semantic rules accordingly (in theory perhaps, automatically no)
   - augment semantic construction with backward function composition in nested evaluation contexts (done)
Further work

Grammar transformation can increase the parsing performance:

1. left-factored grammars
   - delay rule identification for rules with equal first child
   - transform semantic rules accordingly (in theory yes, automatically no)
   - no further refinements to the semantic construction needed

2. left-corner transformed grammars
   - delay attachment decisions
   - transform semantic rules accordingly (in theory perhaps, automatically no)
   - augment semantic construction with backward function composition in nested evaluation contexts (done)
Further work

Grammar transformation can increase the parsing performance:

1. left-factored grammars
   - delay rule identification for rules with equal first childs
   - transform semantic rules accordingly (in theory yes, automatically no)
   - no further refinements to the semantic construction needed

2. left-corner transformed grammars
   - delay attachment decisions
   - transform semantic rules accordingly (in theory perhaps, automatically no)
   - augment semantic construction with backward function composition in nested evaluation contexts (done)
Grammar transformation can increase the parsing performance:

1. **left-factored grammars**
   - delay rule identification for rules with equal first child
   - transform semantic rules accordingly (in theory yes, automatically no)
   - no further refinements to the semantic construction needed

2. **left-corner transformed grammars**
   - delay attachment decisions
   - transform semantic rules accordingly (in theory perhaps, automatically no)
   - augment semantic construction with backward function composition in nested evaluation contexts (done)
Grammar transformation can increase the parsing performance:

1. **left-factored grammars**
   - delay rule identification for rules with equal first childs
   - transform semantic rules accordingly (in theory yes, automatically no)
   - no further refinements to the semantic construction needed

2. **left-corner transformed grammars**
   - delay attachment decisions
   - transform semantic rules accordingly (in theory perhaps, automatically no)
   - augment semantic construction with backward function composition in nested evaluation contexts (done)
Recapitulation

- incrementality: early propagation
- incrementality: interaction / feedback
- top-down left-to-right tree interpretation
- monotonous, incremental construction
Application: Incremental Reference Feedback
Joint work with Timo Baumann, Okko Buß, and David Schlangen
Incremental top-down parser, RMRS construction and reference resolution have been implemented in the InproTK Schlangen et al. [2010] for the Pentamino domain Fernández and Schlangen [2007].
Pentamino puzzle domain:

- Task oriented dialogue, instructor gives commands to a constructor in order to realise a desired state of the world
- Selecting, moving, deleting, rotating and mirroring of the puzzle pieces
Setting: Domain

A corpus collected in a WOz experiment:

- 20 participants, 284 games
- about 3000 utterances (each with dumped world state) of which...
- about 1600 have semantic annotation from the wizards next action, of which...
- about 1000 are free of pronouns.
- small command language, yet free and spontaneous speech
Setting: Domain

A corpus collected in a WOz experiment:

- 20 participants, 284 games
- about 3000 utterances (each with dumped world state) of which...
- about 1600 have semantic annotation from the wizards next action, of which...
- about 1000 are free of pronouns.
- small command language, yet free and spontaneous speech

Beispiele

in der ersten reihe das rote teil löschen
äh und jetzt nimm das kreuz .. das grüne in der linken spalte
und leg es nach ganz unten
nimm [das [blaue teil] [links] [oben] [neben der treppe]]
und im uhrzeigersinn drehen
Setting: Domain

A corpus collected in a WOz experiment:

- 20 participants, 284 games
- about 3000 utterances (each with dumped world state) of which...
- about 1600 have semantic annotation from the wizards next action, of which...
- about 1000 are free of pronouns.
- small command language, yet free and spontaneous speech

Beispiele

in der ersten reihe das rote teil löschen
äh und jetzt nimm das kreuz .. das grüne in der linken spalte
und leg es nach ganz unten
nimm [das [blaue teil] [links] [oben] [neben der treppe]]
und im uhrzeigersinn drehen
Setting: Modules

1. ASR
   - leave-one-out language model training
   - run transcript versus offline recognition

2. Tagger

3. Grammar

4. Parser
   - simplified version of Roark [2001]
   - incremental top-down beam-search, without conditioning functions
   - extended with robust parsing operations (deletions, insertions, tag-repair)

5. Reference Resolution
   - derives variable assignments for all nominal predicate structures
   - calculates the set of domain objects
   - definite NPs receive a resolution value, depending on whether they resolve to a singleton set, a set with more elements or an empty set
Setting: Modules

1. ASR
   - leave-one-out language model training
   - run transcript versus offline recognition

2. Tagger
   - simple lookup tagger with handmade dictionary

3. Grammar

4. Parser
   - simplified version of Roark [2001]
   - incremental top-down beam-search, without conditioning functions
   - extended with robust parsing operations (deletions, insertions, tag-repair)

5. Reference Resolution
   - derives variable assignments for all nominal predicate structures
   - calculates the set of domain objects
   - definite NPs receive a resolution value, depending on whether they resolve to a singleton set, a set with more elements or an empty set
Setting: Modules

1. **ASR**
   - leave-one-out language model training
   - run transcript versus offline recognition

2. **Tagger**
   - simple lookup tagger with handmade dictionary

3. **Grammar**

4. **Parser**

5. **Reference Resolution**
   - derives variable assignments for all nominal predicate structures
   - calculates the set of domain objects
   - definite NPs receive a resolution value, depending on whether they resolve to a singleton set, a set with more elements or an empty set
Setting: Modules

1. **ASR**
   - leave-one-out language model training
   - run transcript versus offline recognition

2. **Tagger**
   - simple lookup tagger with handmade dictionary

3. **Grammar**
   - small handmade core-grammar, weights set by hand, 35 rules

4. **Parser**

5. **Reference Resolution**
   - derives variable assignments for all nominal predicate structures
   - calculates the set of domain objects
   - definite NPs receive a resolution value, depending on whether they resolve to a singleton set, a set with more elements or an empty set
Setting: Modules

1. ASR
   - leave-one-out language model training
   - run transcript versus offline recognition

2. Tagger
   - simple lookup tagger with handmade dictionary

3. Grammar
   - small handmade core-grammar, weights set by hand, 35 rules

4. Parser

5. Reference Resolution
   - derives variable assignments for all nominal predicate structures
   - calculates the set of domain objects
   - definite NPs receive a resolution value, depending on whether they resolve to a singleton set, a set with more elements or an empty set
Setting: Modules

1. **ASR**
   - leave-one-out language model training
   - run transcript versus offline recognition

2. **Tagger**
   - simple lookup tagger with handmade dictionary

3. **Grammar**
   - small handmade core-grammar, weights set by hand, 35 rules

4. **Parser**
   - simplified version of Roark [2001]
   - incremental top-down beam-search, without conditioning functions
   - extended with robust parsing operations (deletions, insertions, tag-repair)

5. **Reference Resolution**
   - derives variable assignments for all nominal predicate structures
   - calculates the set of domain objects
   - definite NPs receive a resolution value, depending on whether they resolve to a singleton set, a set with more elements or an empty set
Setting: Modules

1. **ASR**
   - leave-one-out language model training
   - run transcript versus offline recognition

2. **Tagger**
   - simple lookup tagger with handmade dictionary

3. **Grammar**
   - small handmade core-grammar, weights set by hand, 35 rules

4. **Parser**
   - simplified version of Roark [2001]
   - incremental top-down beam-search, without conditioning functions
   - extended with robust parsing operations (deletions, insertions, tag-repair)

5. **Reference Resolution**
   - derives variable assignments for all nominal predicate structures
   - calculates the set of domain objects
   - definite NPs receive a resolution value, depending on whether they resolve to a singleton set, a set with more elements or an empty set
Setting: Modules

1. ASR
   - leave-one-out language model training
   - run transcript versus offline recognition

2. Tagger
   - simple lookup tagger with handmade dictionary

3. Grammar
   - small handmade core-grammar, weights set by hand, 35 rules

4. Parser
   - simplified version of Roark [2001]
   - incremental top-down beam-search, without conditioning functions
   - extended with robust parsing operations (deletions, insertions, tag-repair)

5. Reference Resolution
   - derives variable assignments for all nominal predicate structures
   - calculates the set of domain objects
   - definite NPs receive a resolution value, depending on whether they resolve to a singleton set, a set with more elements or an empty set
Setting: Modules

1. ASR
   - leave-one-out language model training
   - run transcript versus offline recognition

2. Tagger
   - simple lookup tagger with handmade dictionary

3. Grammar
   - small handmade core-grammar, weights set by hand, 35 rules

4. Parser
   - simplified version of Roark [2001]
   - incremental top-down beam-search, without conditioning functions
   - extended with robust parsing operations (deletions, insertions, tag-repair)

5. Reference Resolution

Incremental Semantic Interpretation of Dialogue Contributions

Peldszus
Setting: Modules

1. **ASR**
   - leave-one-out language model training
   - run transcript versus offline recognition

2. **Tagger**
   - simple lookup tagger with handmade dictionary

3. **Grammar**
   - small handmade core-grammar, weights set by hand, 35 rules

4. **Parser**
   - simplified version of Roark [2001]
   - incremental top-down beam-search, without conditioning functions
   - extended with robust parsing operations (deletions, insertions, tag-repair)

5. **Reference Resolution**
   - derives variable assignments for all nominal predicate structures
   - calculates the set of domain objects
definite NPs receive a resolution value, depending on whether they resolve to a singleton set, a set with more elements or an empty set
Setting: Modules

1 ASR
   - leave-one-out language model training
   - run transcript versus offline recognition

2 Tagger
   - simple lookup tagger with handmade dictionary

3 Grammar
   - small handmade core-grammar, weights set by hand, 35 rules

4 Parser
   - simplified version of Roark [2001]
   - incremental top-down beam-search, without conditioning functions
   - extended with robust parsing operations (deletions, insertions, tag-repair)

5 Reference Resolution
   - derives variable assignments for all nominal predicate structures
   - calculates the set of domain objects
   - definite NPs receive a resolution value, depending on whether they resolve to a singleton set, a set with more elements or an empty set
Setting: Modules

1. **ASR**
   - leave-one-out language model training
   - run transcript versus offline recognition

2. **Tagger**
   - simple lookup tagger with handmade dictionary

3. **Grammar**
   - small handmade core-grammar, weights set by hand, 35 rules

4. **Parser**
   - simplified version of Roark [2001]
   - incremental top-down beam-search, without conditioning functions
   - extended with robust parsing operations (deletions, insertions, tag-repair)

5. **Reference Resolution**
   - derives variable assignments for all nominal predicate structures
   - calculates the set of domain objects
   - definite NPs receive a resolution value, depending on whether they resolve to a singleton set, a set with more elements or an empty set
Setting: Modules

1. ASR
   - leave-one-out language model training
   - run transcript versus offline recognition

2. Tagger
   - simple lookup tagger with handmade dictionary

3. Grammar
   - small handmade core-grammar, weights set by hand, 35 rules

4. Parser
   - simplified version of Roark [2001]
   - incremental top-down beam-search, without conditioning functions
   - extended with robust parsing operations (deletions, insertions, tag-repair)

5. Reference Resolution
   - derives variable assignments for all nominal predicate structures
   - calculates the set of domain objects
   - definite NPs receive a resolution value, depending on whether they resolve to a singleton set, a set with more elements or an empty set
Setting: Modules

1. ASR
   - leave-one-out language model training
   - run transcript versus offline recognition

2. Tagger
   - simple lookup tagger with handmade dictionary

3. Grammar
   - small handmade core-grammar, weights set by hand, 35 rules

4. Parser
   - simplified version of Roark [2001]
   - incremental top-down beam-search, without conditioning functions
   - extended with robust parsing operations (deletions, insertions, tag-repair)

5. Reference Resolution
   - derives variable assignments for all nominal predicate structures
   - calculates the set of domain objects
   - definite NPs receive a resolution value, depending on whether they resolve to a singleton set, a set with more elements or an empty set

Incremental Semantic Interpretation of Dialogue Contributions

Peldszus
Setting: Modules

1. ASR
   - leave-one-out language model training
   - run transcript versus offline recognition

2. Tagger
   - simple lookup tagger with handmade dictionary

3. Grammar
   - small handmade core-grammar, weights set by hand, 35 rules

4. Parser
   - simplified version of Roark [2001]
   - incremental top-down beam-search, without conditioning functions
   - extended with robust parsing operations (deletions, insertions, tag-repair)

5. Reference Resolution
   - derives variable assignments for all nominal predicate structures
   - calculates the set of domain objects
   - definite NPs receive a resolution value, depending on whether they resolve to a singleton set, a set with more elements or an empty set
Baseline / Variants

Compare the gold semantic annotation with the resolving world object according to... 

1. Just Syntax (JS)
   - single-best derivation of only syntax

2. External Filtering (EF)
   - the 5-best derivations are filtered externally for the best reference resolution value

3. Syntax Pragmatic Interaction (SPI)

4. Combined Interaction and Filtering (CIF)
Baseline / Variants

Compare the gold semantic annotation with the resolving world object according to . . .

1. Just Syntax (JS)
   - single-best derivation of only syntax

2. External Filtering (EF)
   - the 5-best derivations are filtered externally for the best reference resolution value

3. Syntax Pragmatic Interaction (SPI)

4. Combined Interaction and Filtering (CIF)
Baseline / Variants

Compare the gold semantic annotation with the resolving world object according to...  

1. Just Syntax (JS)  
   - single-best derivation of only syntax

2. External Filtering (EF)  
   - the 5-best derivations are filtered externally for the best reference resolution value

3. Syntax Pragmatic Interaction (SPI)  
   - single-best derivation with reference feedback

4. Combined Interaction and Filtering (CIF)
Baseline / Variants

Compare the gold semantic annotation with the resolving world object according to . . .

1. Just Syntax (JS)
   - single-best derivation of only syntax

2. External Filtering (EF)
   - the 5-best derivations are filtered externally for the best reference resolution value

3. Syntax Pragmatic Interaction (SPI)
   - single-best derivation with reference feedback

4. Combined Interaction and Filtering (CIF)
Baseline / Variants

Compare the gold semantic annotation with the resolving world object according to...

1. **Just Syntax (JS)**
   - single-best derivation of only syntax

2. **External Filtering (EF)**
   - the 5-best derivations are filtered externally for the best reference resolution value

3. **Syntax Pragmatic Interaction (SPI)**
   - single-best derivation with reference feedback

4. **Combined Interaction and Filtering (CIF)**
   - best of 5-best derivations with reference feedback
Baseline / Variants

Compare the gold semantic annotation with the resolving world object according to...

1. Just Syntax (JS)
   - single-best derivation of only syntax

2. External Filtering (EF)
   - the 5-best derivations are filtered externally for the best reference resolution value

3. Syntax Pragmatic Interaction (SPI)
   - single-best derivation with reference feedback

4. Combined Interaction and Filtering (CIF)
   - best of 5-best derivations with reference feedback
**Baseline / Variants**

Compare the gold semantic annotation with the resolving world object according to . . .

1. **Just Syntax (JS)**
   - single-best derivation of only syntax

2. **External Filtering (EF)**
   - the 5-best derivations are filtered externally for the best reference resolution value

3. **Syntax Pragmatic Interaction (SPI)**
   - single-best derivation with reference feedback

4. **Combined Interaction and Filtering (CIF)**
   - best of 5-best derivations with reference feedback
Baseline / Variants

Compare the gold semantic annotation with the resolving world object according to . . .

1. Just Syntax (JS)
   - single-best derivation of only syntax

2. External Filtering (EF)
   - the 5-best derivations are filtered externally for the best reference resolution value

3. Syntax Pragmatic Interaction (SPI)
   - single-best derivation with reference feedback

4. Combined Interaction and Filtering (CIF)
   - best of 5-best derivations with reference feedback
## Results: Accuracy

<table>
<thead>
<tr>
<th></th>
<th>JS</th>
<th>EF</th>
<th>SPI</th>
<th>CIF</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>563</td>
<td>518</td>
<td>364</td>
<td>363</td>
</tr>
<tr>
<td>0</td>
<td>197</td>
<td>198</td>
<td>267</td>
<td>268</td>
</tr>
<tr>
<td>1</td>
<td>264</td>
<td>308</td>
<td>392</td>
<td>392</td>
</tr>
<tr>
<td>str.acc.</td>
<td>25.7%</td>
<td>30.0%</td>
<td>38.2%</td>
<td>38.2%</td>
</tr>
<tr>
<td>rel.acc.</td>
<td>44.9%</td>
<td>49.3%</td>
<td>64.2%</td>
<td>64.3%</td>
</tr>
<tr>
<td>incr.scr</td>
<td>-1567.66</td>
<td>-1248.26</td>
<td>-535.862</td>
<td>-503.75</td>
</tr>
<tr>
<td>avg.incr.scr</td>
<td>-1.52</td>
<td>-1.22</td>
<td>-0.52</td>
<td>-0.49</td>
</tr>
<tr>
<td>-1</td>
<td>362</td>
<td>348</td>
<td>254</td>
<td>255</td>
</tr>
<tr>
<td>0</td>
<td>122</td>
<td>121</td>
<td>173</td>
<td>173</td>
</tr>
<tr>
<td>1</td>
<td>143</td>
<td>158</td>
<td>196</td>
<td>195</td>
</tr>
<tr>
<td>str.acc.</td>
<td>22.6%</td>
<td>25.0%</td>
<td>31.0%</td>
<td>30.8%</td>
</tr>
<tr>
<td>rel.acc.</td>
<td>41.2%</td>
<td>44.1%</td>
<td>58.3%</td>
<td>58.1%</td>
</tr>
<tr>
<td>incr.scr</td>
<td>-1905.76</td>
<td>-1729.77</td>
<td>-1105.43</td>
<td>-1075.62</td>
</tr>
<tr>
<td>avg.incr.scr</td>
<td>-1.86</td>
<td>-1.69</td>
<td>-1.01</td>
<td>-1.05</td>
</tr>
</tbody>
</table>
## Results: Efficiency

<table>
<thead>
<tr>
<th></th>
<th>readings</th>
<th>expansions</th>
<th>degrades</th>
<th>pruned deriv.</th>
<th>survived deriv.</th>
</tr>
</thead>
<tbody>
<tr>
<td>no interaction</td>
<td>2186</td>
<td>247052</td>
<td>0</td>
<td>0</td>
<td>69510</td>
</tr>
<tr>
<td>interaction</td>
<td>1934</td>
<td>210137</td>
<td>140626</td>
<td>7310</td>
<td>60654</td>
</tr>
<tr>
<td></td>
<td>88%</td>
<td>85%</td>
<td></td>
<td>87%</td>
<td></td>
</tr>
</tbody>
</table>

(for the transcript only, with robust parser operations activated)
Discussion

- grammar does not cover all that is wanted
- yet, there’ll always be strange constructions in spontaneous speech
- semantic gold featured a discourse-history oracle: 30% of the corpus utterances had a semantic annotation without including a proper NP ("und nochmal drehen bitte"). -30% strict -10% relaxed max accuracy
- first experiments, only hand-set parameters
Discussion

• grammar does not cover all that is wanted
• yet, there’ll always be strange constructions in spontaneous speech
• semantic gold featured a discourse-history oracle: 30% of the corpus utterances had a semantic annotation without including a proper NP ("und nochmal drehen bitte"). -30% strict -10% relaxed max accuracy
• first experiments, only hand-set parameters
Discussion

- grammar does not cover all that is wanted
- yet, there’ll always be strange constructions in spontaneous speech
- semantic gold featured a discourse-history oracle: 30% of the corpus utterances had a semantic annotation without including a proper NP ("und nochmal drehen bitte"). -30% strict -10% relaxed max accuracy
- first experiments, only hand-set parameters
Discussion

• grammar does not cover all that is wanted
• yet, there’ll always be strange constructions in spontaneous speech
• semantic gold featured a discourse-history oracle: 30% of the corpus utterances had a semantic annotation without including a proper NP ("und nochmal drehen bitte"). -30% strict -10% relaxed max accuracy
• first experiments, only hand-set parameters
Recapitulation

- incrementality: early propagation
- incrementality: interaction / feedback
- top-down left-to-right tree interpretation
- monotonous, incremental construction
- feedback improves NLUs accuracy and efficiency
Future Work

• write better grammar, train grammar
• probabilistic reference resolution
• optimise parameters
• automatic semantic rule acquisition
• VP advisoring
• experiment in a larger domain, more complex language
Future Work

- write better grammar, train grammar
- probabilistic reference resolution
- optimise parameters
- automatic semantic rule acquisition
- VP advising
- experiment in a larger domain, more complex language
Future Work

- write better grammar, train grammar
- probabilistic reference resolution
- optimise parameters
- automatic semantic rule acquisition
- VP advisoring
- experiment in a larger domain, more complex language
Future Work

- write better grammar, train grammar
- probabilistic reference resolution
- optimise parameters
- automatic semantic rule acquisition
- VP advisoring
- experiment in a larger domain, more complex language
Future Work

- write better grammar, train grammar
- probabilistic reference resolution
- optimise parameters
- automatic semantic rule acquisition
- VP advising
- experiment in a larger domain, more complex language
Future Work

- write better grammar, train grammar
- probabilistic reference resolution
- optimise parameters
- automatic semantic rule acquisition
- VP advisoring
- experiment in a larger domain, more complex language
Thank You!


