Converting CCGs into TFS Grammars

Hans-Ulrich Krieger and Bernd Kiefer

{krieger,kiefer}@dfki.de

Deutsches Forschungszentrum für Künstliche Intelligenz

18th Conference on Head-Driven Phrase Structure Grammar
August 25, 2011

Goal & Motivation

Goal:

Generate a TFS grammar that is equivalent to a given hand-written CCG

Goal & Motivation

Goal:

Generate a TFS grammar that is equivalent to a given hand-written CCG

Motivation:

- Understand the relation between CCG and Constraint-Based formalisms
  - Could CCG-like syntax embedded into TFS formalisms be helpful?
  - Compactness of formulation vs. flexibility

Goal & Motivation

Goal:

Generate a TFS grammar that is equivalent to a given hand-written CCG

Motivation:

- Understand the relation between CCG and Constraint-Based formalisms
  - Could CCG-like syntax embedded into TFS formalisms be helpful?
  - Compactness of formulation vs. flexibility
- Compare performance of processing systems

Goal & Motivation

Goal:

Generate a TFS grammar that is equivalent to a given hand-written CCG

Motivation:

- Understand the relation between CCG and Constraint-Based formalisms
  - Could CCG-like syntax embedded into TFS formalisms be helpful?
  - Compactness of formulation vs. flexibility
- Compare performance of processing systems
- Uncover implicit constraints in the OpenCCG system (constraints that are visible only in the program code)

Goal & Motivation

Goal:

Generate a TFS grammar that is equivalent to a given hand-written CCG

Motivation:

- Understand the relation between CCG and Constraint-Based formalisms
  - Could CCG-like syntax embedded into TFS formalisms be helpful?
  - Compactness of formulation vs. flexibility
- Compare performance of processing systems
- Uncover implicit constraints in the OpenCCG system (constraints that are visible only in the program code)
- Re-use existing work and implementation for grammar approximation
  - Even better performance?
  - Generate language models restricted to sublanguages

Combinatory Categorial Grammar

- Fully lexicalized
- Very limited set of rule schemata
  Most basic: forward and backward application:
  $(>\mathbf{A}) \quad X/Y \quad Y \quad \Rightarrow \quad X$
  $(<\mathbf{A}) \quad Y \quad X\backslash Y \quad \Rightarrow \quad X$
- Semantics in the basic version defined straightforward by functional application
  $(>\mathbf{A}) \quad X/Y : f \quad Y : a \quad \Rightarrow \quad X : f(a)$
  $(<\mathbf{A}) \quad Y : a \quad X\backslash Y : f \quad \Rightarrow \quad X : f(a)$
- Dialect used in the implementation stems from OpenCCG (Baldridge/Kruijff), greater formal power and different construction of semantics

Additional CCG Rule Schemata

- **Harmonic Composition** ($>$**B**) $X/Y \quad Y/Z \;\Rightarrow\; X/Z$
- **Crossed Composition** ($>$**B**$_\times$) $X/Y \quad Y\backslash Z \;\Rightarrow\; X\backslash Z$
- **Substitution** ($>$**S**) $(X/Y)/Z \quad Y/Z \;\Rightarrow\; X/Z$
- **Type Raising** ($>$**T**) $X \;\Rightarrow\; Y/(Y\backslash X)$
- Functional semantics for these schemata (cf. Steedman)
  - $\mathbf{B} fg \equiv \lambda x.f(gx)$
  - $\mathbf{S} fg \equiv \lambda x.fx(gx)$
  - $\mathbf{T} x \equiv \lambda f.fx$

CCG Examples

Lexicon entries

Transitive verb: *defeat* $\vdash$ (s\np)/np : $\lambda x.\lambda y.\mathbf{defeat}(y, x)$

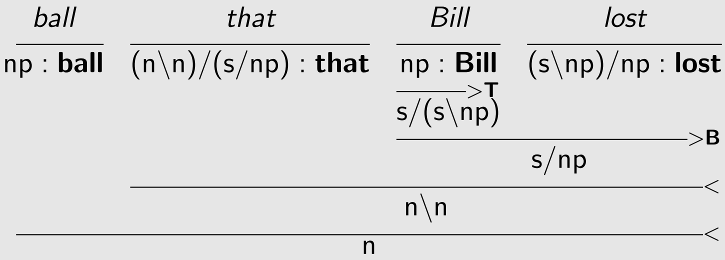
Modal verb: *should* $\vdash$ (s\np)/(s\np) : $\lambda P.\lambda y.\mathbf{should}(Px)$

Derivation

$$\dfrac{\dfrac{\textit{Brazil}}{\mathrm{np} : \mathbf{Brazil}} \quad \dfrac{\dfrac{\textit{defeats}}{(\mathrm{s}\backslash\mathrm{np})/\mathrm{np} : \lambda x.\lambda y.\mathbf{defeat}(y, x)} \quad \dfrac{\textit{Germany}}{\mathrm{np} : \mathbf{Germany}}}{\mathrm{s}\backslash\mathrm{np} : \lambda x.\mathbf{defeat}(y, \mathbf{Germany})}>}{\mathrm{s} : \mathbf{defeat}(\mathbf{Brazil}, \mathbf{Germany})}<$$

CCG Examples

Derivation with type raising and backward substitution



TFS Translation: Lexicon Entry *defeat*

defeat $\vdash$ (s\np)/np : $\lambda x.\lambda y.\mathbf{defeat}(y, x)$

$$\begin{bmatrix} \textit{defeat} & \\ \textit{cat} & \begin{bmatrix} / & \\ \textit{1st} & \begin{bmatrix} \backslash & \\ \textit{1st} & \textit{s} \\ \textit{2nd} & \textit{np} \end{bmatrix} \\ \textit{2nd} & \textit{np} \end{bmatrix} \\ \textit{sem} & \begin{bmatrix} \lambda & \\ \textit{var} & \boxed{x} \\ \textit{body} & \begin{bmatrix} \lambda & \\ \textit{var} & \boxed{y} \\ \textit{body} & \begin{bmatrix} f & \\ \textit{name} & \mathbf{defeat} \\ \textit{args} & \langle \boxed{y}, \boxed{x} \rangle \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

TFS Translation: Lexicon Entry *should*

$\textit{should} \vdash (\mathrm{s}\backslash\mathrm{np})/(\mathrm{s}\backslash\mathrm{np}) : \lambda P.\lambda x.\mathbf{should}(Px)$

$$
\begin{bmatrix} \textit{should} & \\ \textit{cat} & \begin{bmatrix} / & \\ \textit{1st} & \begin{bmatrix} \backslash & \\ \textit{1st} & \textit{s} \\ \textit{2nd} & \textit{np} \end{bmatrix} \\ \textit{2nd} & \begin{bmatrix} \backslash & \\ \textit{1st} & \textit{s} \\ \textit{2nd} & \textit{np} \end{bmatrix} \end{bmatrix} \\ \textit{sem} & \begin{bmatrix} \lambda & \\ \textit{var} & \begin{bmatrix} \lambda & \\ \textit{var} & \boxed{x} \\ \textit{body} & \boxed{b} \end{bmatrix} \\ \textit{body} & \begin{bmatrix} \lambda & \\ \textit{var} & \boxed{x} \\ \textit{body} & \begin{bmatrix} f & \\ \textit{name} & \textbf{should} \\ \textit{args} & \left\langle \boxed{b} \right\rangle \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}
$$

TFS Translation: Forward Application

$$(>\mathbf{A}) \quad X/Y : f \quad Y : a \quad \Rightarrow \quad X : fa$$

$$\begin{bmatrix} >\mathbf{A} \\ \mathit{cat}\ \boxed{X} \\ \mathit{sem}\ \boxed{f} \\ \mathit{dtrs}\ \left\langle \begin{bmatrix} \mathit{cat} & \begin{bmatrix} / \\ \mathit{1st}\ \boxed{X} \\ \mathit{2nd}\ \boxed{Y} \end{bmatrix} \\ \mathit{sem} & \begin{bmatrix} \lambda \\ \mathit{var}\ \boxed{a} \\ \mathit{body}\ \boxed{f} \end{bmatrix} \end{bmatrix}, \begin{bmatrix} \mathit{cat}\ \boxed{Y} \\ \mathit{sem}\ \boxed{a} \end{bmatrix} \right\rangle \end{bmatrix}$$

TFS Translation: Forward Crossed Composition

$$(>\mathbf{B}_\times) \quad X/Y : f \quad Y\backslash Z : g \;\Rightarrow\; X\backslash Z : \lambda x.f(gx)$$

$$\begin{bmatrix} >\mathbf{B}_\times & \\ \textit{cat} & \begin{bmatrix} \backslash \\ \textit{1st}\ \boxed{X} \\ \textit{2nd}\ \boxed{Z} \end{bmatrix} \\ \textit{sem} & \begin{bmatrix} \lambda \\ \textit{var}\ \boxed{x} \\ \textit{body}\ \boxed{f} \begin{bmatrix} \textit{args}|\textit{first}\ \boxed{g} \end{bmatrix} \end{bmatrix} \\ \textit{dtrs} & \left\langle \begin{bmatrix} \textit{cat} & \begin{bmatrix} / \\ \textit{1st}\ \boxed{X} \\ \textit{2nd}\ \boxed{Y} \end{bmatrix} \\ \textit{sem}|\textit{body}\ \boxed{f} & \end{bmatrix}, \begin{bmatrix} \textit{cat} & \begin{bmatrix} \backslash \\ \textit{1st}\ \boxed{Y} \\ \textit{2nd}\ \boxed{Z} \end{bmatrix} \\ \textit{sem} & \begin{bmatrix} \textit{var}\ \boxed{x} \\ \textit{body}\ \boxed{g} \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix}$$

Dollar Convention

- Purpose: Create a rule that transfers an arbitrary number of arguments from a daughter into the mother
- s/\$ represents: $\{s, s/X, (s/X)/Y, \ldots\}$
- Generalized Forward Composition:
  $(>\mathbf{B}^*)\quad X/Y \quad (Y/Z)\,\$ \;\Rightarrow\; (X/Z)\,\$$
- More than one dollar possible: dollar variables:
  $(??)\quad X/Y \quad (Y\,\$_1\,/Z\,)\,\$_2 \;\Rightarrow\; (X\,\$_1\,/Z\,)\,\$_2$
- Often used in type changing (lexical) unary rules
- A dollar specification may include the slash
  → *SLASH* feature needed to be able to coindex slash
- Example from CCG for English: $pp\,\$_1 \;\Rightarrow\; s\,\$_1\backslash s$

Dollar Convention: Translation

Since these rules dig into the argument stack, there is no simple translation as for the other schemata

Implementation alternatives:

- Code in the processing system that matches the (buried) top elements: that's what we want to get rid of

Dollar Convention: Translation

Since these rules dig into the argument stack, there is no simple translation as for the other schemata

Implementation alternatives:

- Code in the processing system that matches the (buried) top elements: that's what we want to get rid of
- Use helper rules
  - Removal rule puts argument from the stack into a scratch stack
  - Finishing rule matches the "real" arguments and restores the stack
  - Ordinary rules are blocked on intermediate results of the removal rule
  - Efficiency questionable: generates a lot of intermediate edges

Dollar Convention: Translation

Since these rules dig into the argument stack, there is no simple translation as for the other schemata

Implementation alternatives:

- Code in the processing system that matches the (buried) top elements: that's what we want to get rid of
- Use helper rules
  - Removal rule puts argument from the stack into a scratch stack
  - Finishing rule matches the "real" arguments and restores the stack
  - Ordinary rules are blocked on intermediate results of the removal rule
  - Efficiency questionable: generates a lot of intermediate edges
- Expand inductively into a set of rules,
  - Each rule transfers a fixed number of arguments
  - Not exact, since expansion can not be infinite
  - In reality, the number of transferred arguments will be finite and small
  - Presumably more efficient, since there are no intermediate results

Dollar Convention: Translation Example

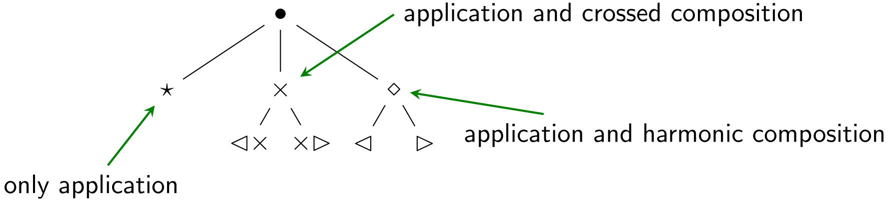
$$(>\mathbf{B}^*) \quad X/Y \quad (Y/Z)\ \$ \quad \Rightarrow \quad (X/Z)\ \$$$

$$\begin{bmatrix} >\mathbf{B}_1 & \\ \textit{cat} & \begin{bmatrix} \textit{slash} & \boxed{S_1} \\ \textit{1st} & \begin{bmatrix} \textit{1st} & \boxed{X} \\ \textit{2nd} & \boxed{Z} \end{bmatrix} \\ \textit{2nd} & \boxed{A_1} \end{bmatrix} \\ \textit{dtrs} & \left\langle \begin{bmatrix} \textit{cat} & \begin{bmatrix} \textit{slash} & / \\ \textit{1st} & \boxed{X} \\ \textit{2nd} & \boxed{Y} \end{bmatrix} \end{bmatrix}, \begin{bmatrix} \textit{cat} & \begin{bmatrix} \textit{slash} & \boxed{S_1} \\ \textit{1st} & \begin{bmatrix} \textit{slash} & / \\ \textit{1st} & \boxed{Y} \\ \textit{2nd} & \boxed{Z} \end{bmatrix} \\ \textit{2nd} & \boxed{A_1} \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix}$$

$$\begin{bmatrix} >\mathbf{B}_2 & \\ \textit{cat} & \begin{bmatrix} \textit{slash} & \boxed{S_1} \\ \textit{1st} & \begin{bmatrix} \textit{slash} & \boxed{S_2} \\ \textit{1st} & \begin{bmatrix} \textit{1st} & \boxed{X} \\ \textit{2nd} & \boxed{Z} \end{bmatrix} \\ \textit{2nd} & \boxed{A_2} \end{bmatrix} \\ \textit{2nd} & \boxed{A_1} \end{bmatrix} \\ \textit{dtrs} & \left\langle \begin{bmatrix} \textit{cat} & \begin{bmatrix} \textit{slash} & / \\ \textit{1st} & \boxed{X} \\ \textit{2nd} & \boxed{Y} \end{bmatrix} \end{bmatrix}, \begin{bmatrix} \textit{cat} & \begin{bmatrix} \textit{slash} & \boxed{S_1} \\ \textit{1st} & \begin{bmatrix} \textit{slash} & \boxed{S_2} \\ \textit{1st} & \begin{bmatrix} \textit{slash} & / \\ \textit{1st} & \boxed{Y} \\ \textit{2nd} & \boxed{Z} \end{bmatrix} \\ \textit{2nd} & \boxed{A_2} \end{bmatrix} \\ \textit{2nd} & \boxed{A_1} \end{bmatrix} \end{bmatrix} \right\rangle \end{bmatrix}$$

Categorical Information and Modes

- Inclusion of more general schemata may lead to spurious ambiguities
- Additional information provided to restrict applicability
  - Categorical information as feature-value pairs (e.g., morphology)
  - binary feature on slashes: *active* or *inert*
  - Hierarchy of slash modes, that control the applicability of rule schemata:



Transformation Process

- Straightforward transformation of type hierarchy, some basic types defined by hand
- Lexicon entries and unary rules:
  - Initial plan: read XML definitions directly
  - But: Some only poorly understood feature inheritance mechanisms found in the code
  - Now: data structures generated by OpenCCG and transformed into TFS
- Binary rule schemata created by hand:
  OpenCCG implements them with Java classes for every schema type

Transformation Process: Problems

- Slash mode type hierarchy and *activity* derived from code, different from definition in papers
- Lots of hard-coded default restrictions found
  - default type raising rules only applicable to NP's
  - slashes in rule schemata are modalized in the constructor depending on type
  - activity changes of argument slashes during unification under special conditions
  - etc., etc.
- Correct transformation took quite some time due to all this hidden constraints and functionality
- It's very likely that we still overlooked some of the traps

Transformation Process: Semantics

- OpenCCG uses hybrid logic dependency semantics (HLDS)

  'you should have the ball'

  $$@_{ascription:e1}(\textbf{have} \wedge \langle \textsc{Mood} \rangle \text{ind} \wedge \langle \textsc{Tense} \rangle \text{pres} \wedge \langle \textsc{Modifier} \rangle m1 \wedge \langle \textsc{Act} \rangle t0 \wedge \langle \textsc{Pat} \rangle t1 \wedge \langle \textsc{Subj} \rangle t0)$$
  $$\wedge @_{person:t0}(\textbf{you} \wedge \langle \textsc{Num} \rangle \text{sg}) \wedge @_{modal:m1}\textbf{should}$$
  $$\wedge @_{thing:t1}(\textbf{ball} \wedge \langle \textsc{Num} \rangle \text{sg} \wedge \langle \textsc{Delim} \rangle \text{unique} \wedge \langle \textsc{Quant} \rangle \text{specific})$$

- Similar in many respects to MRS
- Allows (almost) direct realization as feature structure
  - $\langle \textsc{Modifier} \rangle$ relations may occur multiple times in one structure
  - cyclic structures possible
- Problematic cases avoided by slightly modifying the input grammar
- Different translation (flat list) considered to obtain equivalence

Measurements

- Medium-size hand-written English grammar for interaction with robots
- Small test corpus of 246 sentences coming with the grammar
- Vanilla OpenCCG parsing compared to pet using translated grammar
- Both parsers did not use supertagging (no models available)
- Both parsers used packing
- pet used the standard filter and optimization techniques
- Same number of passive edges → translation seems to be correct
- Platform: MacBook Pro 1,1, Ubuntu 10.10, startup times taken out
  - pet: 9.49 CPUsec, 170MB max
  - OpenCCG: 75.56 CPUsec, 780MB max

Outlook

- Modify original semantics to simplify transformation:
    - Modifier lists
    - remove cyclic formulations
- Restore the families of lexicon entries to get more compact representations
- Application of approximation methods
- Use of approximated grammars to ease learning of language models
- Add additional descriptional apparatus to $\mathcal{TDL}$?
    - more compact representation
    - automatic expansion

Thank you for listening!