Dynamic Logic in ACG: discourse anaphora and scoping islands

Logical Methods for Discourse
Nancy, December 15, 2009

To analyze scoping islands within the Abstract Categorial Grammar (ACG) formalism we propose an enhancement to ACG along the lines of dynamic logic. The enhanced ACG explains not only the distinct scopes of universals and indefinites and clause-boundness of universals. We can also apply our ACG to anaphoric indefinite descriptions in discourse. We explain how an indefinite can scope inside negation, yet cannot scope outside negation and create definitedness presuppositions.

Our enhancement to ACG affects only the mapping from abstract language to semantics. We retain all ACG's benefits of parsing from the surface form. Crucially, by avoiding type lifting we keep the order of the abstract signature low, so that parsing remains tractable.

We regard the mapping from abstract language to semantics partial: some sentences, albeit well-formed, just don't make sense. We model this partial mapping as a potentially failing computation in a call-by-value language with multi-prompt delimited control. The evaluation and type inference rules of the language are simple and deterministic. Control prompts may be regarded as loci of binding or quantification, used by quantified phrases and pronouns and set by context. We arrive at the mechanism of interaction of a phrase with its context that determines the scope.

Puzzles

(1) A donkey enters. It brays.

(2) Every donkey enters. ⋆It brays.

(3) It-is-not-the-case-that a donkey enters. ⋆It brays.

(4) A donkey and a mule enter. ⋆It brays.

(5) A donkey and a mule enter. The donkey brays.

(6) A donkey enters. It-is-not-the-case-that it brays.

(7) Every donkey denies it brays.

Thanks to Carl Pollard

Thanks to Carl Pollard for these examples.

Puzzles

(1) A donkey enters. It brays.

(2) Every donkey enters. ⋆It brays.

(3) It-is-not-the-case-that a donkey enters. ⋆It brays.

(4) A donkey and a mule enter. ⋆It brays.

(5) A donkey and a mule enter. The donkey brays.

(6) A donkey enters. It-is-not-the-case-that it brays.

(7) Every donkey denies it brays.

- quantification and binding

Thanks to Carl Pollard

A quantifier can bind a variable within its scope, (7), (1).

Puzzles

(1) A donkey enters. It brays.

(2) Every donkey enters. ⋆It brays.

(3) It-is-not-the-case-that a donkey enters. ⋆It brays.

(4) A donkey and a mule enter. ⋆It brays.

(5) A donkey and a mule enter. The donkey brays.

(6) A donkey enters. It-is-not-the-case-that it brays.

(7) Every donkey denies it brays.

- quantification and binding
- different scope of different quantifiers

Thanks to Carl Pollard

Different quantifiers have different scope abilities. Universals are clause- or sentence-bound, (2), but indefinites can scope out of a clause or a sentence (1).

Puzzles

(1) A donkey enters. It brays.

(2) Every donkey enters. ⋆It brays.

(3) It-is-not-the-case-that a donkey enters. ⋆It brays.

(4) A donkey and a mule enter. ⋆It brays.

(5) A donkey and a mule enter. The donkey brays.

(6) A donkey enters. It-is-not-the-case-that it brays.

(7) Every donkey denies it brays.

- quantification and binding
- different scope of different quantifiers
- islands: coordinated structures

Thanks to Carl Pollard

But indefinites can’t scope out of a coordinated structure, (4).

Puzzles

(1) A donkey enters. It brays.

(2) Every donkey enters. ⋆It brays.

(3) It-is-not-the-case-that a donkey enters. ⋆It brays.

(4) A donkey and a mule enter. ⋆It brays.

(5) A donkey and a mule enter. The donkey brays.

(6) A donkey enters. It-is-not-the-case-that it brays.

(7) Every donkey denies it brays.

- quantification and binding
- different scope of different quantifiers
- islands: coordinated structures
- binding into but not out of negation

Thanks to Carl Pollard

An indefinite can bind into negation (6), yet cannot bind out of negation and create definitedness presuppositions, (3).

Results

- Combination of ACG with dynamic semantics
- No type lifting: low complexity of parsing
- Explaining the puzzles
- Uniform mechanism for binding and quantification and their scope

Even with type-lifting, ACG currently has trouble explaining scoping islands, e.g., why universals are clause-bounded.

Outline

Running example

Pedro beats a donkey.

First we need to explain why we use ACG, and how to combine with dynamic logic. For this explanation, which would probably take most of the time, we use a simpler example: a politically incorrect sentence, also suggested by Carl Pollard.I don't need to explain ACG to this audience at all. I only give a brief introduction for the sake of terminology and to clarify the points where we will extend ACG.

Abstract signature

A higher-order signature

A collection of atomic types, constants, and type assignments to constants

Signature Σ_{abs}

Atomic types	N, NP, S, D
Pedro	$: NP$
donkey	$: N$
a	$: N \to NP$
beat	$: NP \to NP \to S$
fullstop	$: S \to D$

Abstract signature

A higher-order signature

A collection of atomic types, constants, and type assignments to constants

Signature Σ_{abs}

Atomic types	N, NP, S, D
Pedro	$: NP$
donkey	$: N$
a	$: N \rightarrow NP$
beat	$: NP \rightarrow NP \rightarrow S$
fullstop	$: S \rightarrow D$

The only uncommon parts here are the type D, for the complete discourse, and ’fullstop’, the end of the discourse (or sentence, in this case).

Abstract terms

Terms over Σ_{abs}

$e ::= x \mid c \mid ee \mid \lambda x.\, e, \qquad c \in \Sigma_{\text{abs}}$

A sample term

$t_{donkey} \overset{\text{def}}{=} \mathsf{fullstop}\ (\mathsf{beat}\ (\mathsf{a}\ \mathsf{donkey})\ \mathsf{Pedro})$

One can verify that the term t_{donkey} is well-typed and so it is in the set of typed lambda-terms over the abstract signature.

String signature

	Atomic type	string
	`"Pedro"`	: string
	`"donkey"`	: string
Signature Σ_{str}	`"a"`	: string
	`"beat"`	: string
	`"."`	: string
	$\diamondsuit$	: $\text{string} \to \text{string} \to \text{string}$

The operation $\diamondsuit$ denotes string concatenation.

Lexicon

$\mathcal{L}_{\text{str}}$: mapping of constants of Σ_{abs} to terms over Σ_{str}

N, NP, S, and D	$\mapsto$	string
Pedro	$\mapsto$	`"Pedro"`
donkey	$\mapsto$	`"donkey"`
a	$\mapsto$	$\lambda x.\, \texttt{"a"} \diamondsuit x$
beat	$\mapsto$	$\lambda o.\, \lambda s.\, s \diamondsuit \texttt{"beat"} \diamondsuit o$
fullstop	$\mapsto$	$\lambda x.\, x \diamondsuit \texttt{"."}$

Lexicon *interprets* constants of the abstract signature in, here, the surface language. The interpretation of constants homomorphically extends to the interpretation of the whole abstract language in terms of the surface language.

Lexicon

$\mathcal{L}_{\text{str}}$: mapping of constants of Σ_{abs} to terms over Σ_{str}

N, NP, S, and D	$\mapsto$	string
Pedro	$\mapsto$	`"Pedro"`
donkey	$\mapsto$	`"donkey"`
a	$\mapsto$	$\lambda x.\,\texttt{"a"} \diamondsuit x$
beat	$\mapsto$	$\lambda o.\,\lambda s.\,s \diamondsuit \texttt{"beat"} \diamondsuit o$
fullstop	$\mapsto$	$\lambda x.\,x \diamondsuit \texttt{"."}$

the surface form

$$\begin{aligned}
&\mathcal{L}_{\text{str}}(t_{donkey}) \\
= \quad & (\lambda x.\,x \diamondsuit \texttt{"."})((\lambda o.\,\lambda s.\,s \diamondsuit \texttt{"beat"} \diamondsuit o) \\
& \qquad ((\lambda x.\,\texttt{"a"} \diamondsuit x)\texttt{"donkey"})\ \texttt{"Pedro"}) \\
\hookrightarrow \quad & \texttt{"Pedro"} \diamondsuit \texttt{"beat"} \diamondsuit \texttt{"a"} \diamondsuit \texttt{"donkey"} \diamondsuit \texttt{"."}
\end{aligned}$$

I must emphasize a point that becomes very important later. If we just substitute for the constants in the sample donkey term their lexicon-mapped terms, we get this long phrase on the second line in the table.

Lexicon

$\mathcal{L}_{\text{str}}$: mapping of constants of Σ_{abs} to terms over Σ_{str}

N, NP, S, and D	$\mapsto$	string
Pedro	$\mapsto$	`"Pedro"`
donkey	$\mapsto$	`"donkey"`
a	$\mapsto$	$\lambda x.\,\texttt{"a"} \diamondsuit x$
beat	$\mapsto$	$\lambda o.\,\lambda s.\,s \diamondsuit \texttt{"beat"} \diamondsuit o$
fullstop	$\mapsto$	$\lambda x.\,x \diamondsuit \texttt{"."}$

Computing the surface form

$$\begin{aligned}
&\mathcal{L}_{\text{str}}(t_{donkey}) \\
= \quad & (\lambda x.\,x \diamondsuit \texttt{"."})((\lambda o.\,\lambda s.\,s \diamondsuit \texttt{"beat"} \diamondsuit o) \\
& \qquad ((\lambda x.\,\texttt{"a"} \diamondsuit x)\texttt{"donkey"})\ \texttt{"Pedro"}) \\
\hookrightarrow \quad & \texttt{"Pedro"} \diamondsuit \texttt{"beat"} \diamondsuit \texttt{"a"} \diamondsuit \texttt{"donkey"} \diamondsuit \texttt{"."}
\end{aligned}$$

When we *normalize* that term we get what looks like a string, the surface form of our sentence. In ACG tutorials that I read, the fact that we have to normalize, or reduce, the result of the lexicon substitution is hardly ever mentioned. There is a good reason: there is little to say: The calculus here is simply-typed lambda calculus and is strongly normalizing. Every term has the normal form; the normalization is as uneventful as it could ever get. But that would change, in our extension to ACG.

Outline

What are Abstract Categorial Grammars (ACG)?

▶ **Why ACGs**

Direct dynamic logic meta-calculus

What about the original puzzles?

Live demo

Why ACG?

Abstract signature

Atomic types	N, NP, S, D
Pedro	$: NP$
donkey	$: N$
a	$: N \rightarrow NP$
beat	$: NP \rightarrow NP \rightarrow S$
fullstop	$: S \rightarrow D$

Abstract and surface forms

fullstop (beat (a donkey) Pedro)

$\hookrightarrow$ `"Pedro"` $\Diamond$ `"beat"` $\Diamond$ `"a"` $\Diamond$ `"donkey"` $\Diamond$ `"."`

So, what attracts me to ACG: the notion of mapping of languages, of interpretations, of a hidden, abstract (I almost said, logical) form. We have seen how the abstract phrase maps to the surface form.

Why ACG?

Abstract signature

Atomic types	N, NP, S, D
Pedro	$: NP$
donkey	$: N$
a	$: N \rightarrow NP$
beat	$: NP \rightarrow NP \rightarrow S$
fullstop	$: S \rightarrow D$

Abstract and surface forms

fullstop (beat (a donkey) Pedro)

$\hookrightarrow$ `"Pedro"` $\Diamond$ `"beat"` $\Diamond$ `"a"` $\Diamond$ `"donkey"` $\Diamond$ `"."`

The mapping took care of the word order. English is SVO language; the surface lexicon switched the words around. A Japanese lexicon would have moved the verb at the end.

Why ACG?

Abstract signature

Atomic types	N, NP, S, D
Pedro	$: NP$
donkey	$: N$
a	$: N \longrightarrow NP$
beat	$: NP \longrightarrow NP \longrightarrow S$
fullstop	$: S \longrightarrow D$

Abstract and surface forms

fullstop (beat (a donkey) Pedro)

$\hookrightarrow$ `"Pedro"` $\Diamond$ `"beat"` $\Diamond$ `"a"` $\Diamond$ `"donkey"` $\Diamond$ `"."`

It is quite easy to extend the surface lexicon to handle subject-verb agreement, so that the result reads “beats” rather than “beat”. We could’ve taken care of cases, declination, verb conjugations, etc.

Why ACG?

Abstract signature

Atomic types	N, NP, S, D
Pedro	$: NP$
donkey	$: N$
a	$: N \rightarrow NP$
beat	$: NP \rightarrow NP \rightarrow S$
fullstop	$: S \rightarrow D$

Abstract and surface forms

fullstop (beat (a donkey) Pedro)

$\hookrightarrow$ `"Pedro"` $\diamondsuit$ `"beat"` $\diamondsuit$ `"a"` $\diamondsuit$ `"donkey"` $\diamondsuit$ `"."`

Abstract and semantic forms

fullstop (beat (a donkey) Pedro)

$\hookrightarrow$ $\exists_j \, (donkey\, j) \wedge (beat\, j\, Pedro)$

If we are interested in semantics, we would map the abstract form (rather than the surface form) to a logical formula. The abstract form is simpler than the surface form. Since the word order is taken care of, there is no need to distinguish left application from the right application.

Why ACG?

Abstract signature

Atomic types	N, NP, S, D
Pedro	$: NP$
donkey	$: N$
a	$: N \rightarrow NP$
beat	$: NP \rightarrow NP \rightarrow S$
fullstop	$: S \rightarrow D$

Abstract and surface forms

fullstop (beat (a donkey) Pedro)

$\xleftarrow{\text{parse}}$ `"Pedro"` $\diamondsuit$ `"beat"` $\diamondsuit$ `"a"` $\diamondsuit$ `"donkey"` $\diamondsuit$ `"."`

Abstract and semantic forms

fullstop (beat (a donkey) Pedro)

$\hookrightarrow$ $\exists_j\, (donkey\, j) \wedge (beat\, j\, Pedro)$

As to the surface form, we certainly are more interested in *parsing to* to the abstract form, rather than the mapping from the abstract form. ACG parsing has been the subject of intense research, great strides have been made and good techniques developed. All these advances fully benefit us: on the syntactic side, our approach is identical to ACG.

Why ACG?

Abstract signature

Atomic types	N, NP, S, D
Pedro	$: NP$
donkey	$: N$
a	$: N \rightarrow NP$
beat	$: NP \rightarrow NP \rightarrow S$
fullstop	$: S \rightarrow D$

Abstract and surface forms

fullstop (beat (a donkey) Pedro)

$\xleftarrow{\text{parse}}$ `"Pedro"` $\diamondsuit$ `"beat"` $\diamondsuit$ `"a"` $\diamondsuit$ `"donkey"` $\diamondsuit$ `"."`

Abstract and semantic forms

fullstop (beat (a donkey) Pedro)

$\hookrightarrow \quad \exists_j\,(donkey\, j) \wedge (beat\, j\, Pedro)$

The complexity of parsing depends on the order of the abstract signature, which is the maximal nesting depth of arrows. In our case, the order is 1: the parsing is therefore tractable and relatively easy.

Why ACG?

Abstract signature

Atomic types	N, NP, S, D
Pedro	$: NP$
donkey	$: N$
a	$: N \multimap NP$
beat	$: NP \multimap NP \multimap S$
fullstop	$: S \multimap D$

Abstract and surface forms

fullstop (beat (a donkey) Pedro)

$\overset{\text{parse}}{\hookleftarrow}$ `"Pedro"` $\Diamond$ `"beat"` $\Diamond$ `"a"` $\Diamond$ `"donkey"` $\Diamond$ `"."`

Abstract and semantic forms

fullstop (beat (a donkey) Pedro)

$\hookrightarrow$ $\exists_j\, (donkey\, j) \wedge (beat\, j\, Pedro)$

We’ve been sloppy, overlooking the fact that the arrow types in the signature should have been linear arrow types (lollipops). Linear types are crucial for parsing. In semantics, non-linear terms are common (e.g., in our logical formula, *j* appears twice). Besides, we don’t usually need to parse a logical formula into the abstract form.

Why ACG?

Abstract and semantic forms

fullstop (beat (a donkey) Pedro)

$\hookrightarrow \quad \exists_j \, (donkey\, j) \wedge (beat\, j\, Pedro)$

In the rest of the talk we'll be dealing only with the semantic mapping, from an abstract form to a logical formula. In particular, we will explain how to map the sample donkey term in the first line of the table to its corresponding logical formula. As in case of the surface form, the mapping is the lexicon substitution plus normalization. I emphasize normalization, which will become prominent.

Outline

What are Abstract Categorial Grammars (ACG)?

Why ACGs

▶ **Direct dynamic logic meta-calculus**

What about the original puzzles?

Live demo

Semantic signature

Signature Σ_{sem}

Atomic types	e, t
$Pedro$	$: e$
$donkey$	$: e \to t$
$beat$	$: e \to e \to t$
$\wedge$	$: t \to t \to t$
$\exists_i$	$: t \to t$
i	$: e$

We start as before, with a signature. It has the expected atomic types e and t, domain constants such as $Pedro$, and logical connectives. There are also logical variables i with an unlimited supply. A quantifier such as $\exists_i$ is indexed by the logical variable that it binds. There are deep reasons for such an unusual setup; I'll be happy to discuss them afterwards. (First of all, if we use a higher-order abstract syntax, we would not be able to compute the body of a quantifier since we can't evaluate under lambda. Mainly, we would like to keep the the meta-language distinct from the target language. Both languages are higher-order and involve binding. We would like to keep the bindings, and variables, separate.)

Turning point

Emulate or build in?

We now want to establish a mapping from an abstract phrase to a formula built with the constants of the semantic signature.

Our sample sentence is a simple example of a *scopal* expression (quantification), which “contributes meaning where it is not seen or heard.” (Carl’s phrase). We would like to take advantage of dynamic logic to analyze this and more complex sentences. One may interpret “dynamic” as an information update, a particular accessibility relation among possible worlds, or just as a desk drawer, into which we can put things and look them up later.

Turning point

Emulate or build in?

- Church numerals vs. native numbers
- CPS (type lifting) vs. direct calculus of effects

We have a choice to make. We can use the ordinary lambda-calculus to express this dynamic aspect. After all, the normalization relation can encode arbitrary computations (given the right type system). We can *encode* our desk drawer. CPS, or type lifting, is such an encoding, which to me appears less than satisfactory. Type lifting raises the order of the types in the abstract signature: parsing becomes more complex or even undecidable. Aesthetically, this encoding appears like doing arithmetic using Church numerals. It is doable, but we won't like doing our taxes this way.

Turning point

Emulate or build in?

- Church numerals vs. native numbers
- CPS (type lifting) vs. direct calculus of effects

Our choice: dynamic effects directly in the calculus

We would like to talk about laws of numbers no matter how numbers are actually represented. Likewise, we want to talk about effects directly. In our calculus, we want effects built-in as primitives.

The $\leftrightarrows$ calculus over Σ_{sem}

Expressions

$$e, k \quad ::= x \mid w \mid () \mid \lambda x.\, e \mid ee \mid \text{reset}_{\text{p}}\, e \mid \widehat{e}_p e$$

Logical Formulae

$$w \quad ::= c \mid ww \qquad c \in \Sigma_{\text{sem}}$$

Values

$$v \quad ::= x \mid w \mid () \mid \lambda x.\, e$$

Reduction rules

$$\begin{array}{lll} (\beta_v) & (\lambda x.\, e)v & \rightsquigarrow\ e\{x := v\} \\ (E_r) & \text{reset}_{\text{p}}\, v & \rightsquigarrow\ v \\ (E_s) & \text{reset}_{\text{p}}\, \widehat{e}_p k & \rightsquigarrow\ \text{reset}_{\text{p}}\ e\ (\lambda x.\, \text{reset}_{\text{p}}\, kx) \\ (\cong) & (\widehat{e}_p k)e_2 & \rightsquigarrow\ \widehat{e}_p(\lambda x.(kx)e_2) \\ & v_1(\widehat{e}_p k) & \rightsquigarrow\ \widehat{e}_p(\lambda x.\, v_1(kx)) \\ & \text{reset}_{\text{q}}\, \widehat{e}_p k & \rightsquigarrow\ \widehat{e}_p(\lambda x.\, \text{reset}_{\text{q}}\, kx) \qquad p \neq q \end{array}$$

This is our calculus over the semantic signature. It includes logical formulas built from the semantic signature, a special value unit, regular abstractions and applications, and reset and shift (or, a continuation bubble). Resets and bubbles are indexed by *prompts*, of which there is an unlimited supply. We define a subset of expression to call values.

The $\leftrightarrows$ calculus over Σ_{sem}

Expressions

$$e, k \quad ::= x \mid w \mid () \mid \lambda x.\, e \mid ee \mid \text{reset}_{\text{p}}\, e \mid \widehat{e}_p e$$

Logical Formulae

$$w \quad ::= c \mid ww \qquad c \in \Sigma_{\text{sem}}$$

Values

$$v \quad ::= x \mid w \mid () \mid \lambda x.\, e$$

Reduction rules

$$\begin{array}{lllll}
(\beta_v) & (\lambda x.\, e)v & \rightsquigarrow & e\{x := v\} & \\
(E_r) & \text{reset}_{\text{p}}\, v & \rightsquigarrow & v & \\
(E_s) & \text{reset}_{\text{p}}\, \widehat{e}_p k & \rightsquigarrow & \text{reset}_{\text{p}}\ e\ (\lambda x.\, \text{reset}_{\text{p}}\, kx) & \\
(\cong) & (\widehat{e}_p k)e_2 & \rightsquigarrow & \widehat{e}_p(\lambda x.(kx)e_2) & \\
 & v_1(\widehat{e}_p k) & \rightsquigarrow & \widehat{e}_p(\lambda x.\, v_1(kx)) & \\
 & \text{reset}_{\text{q}}\, \widehat{e}_p k & \rightsquigarrow & \widehat{e}_p(\lambda x.\, \text{reset}_{\text{q}}\, kx) & p \neq q
\end{array}$$

We restrict the ordinary β-rule: only values can be substituted: see β_v.

The $\leftrightarrows$ calculus over Σ_{sem}

Expressions

$$e, k \quad ::= x \mid w \mid () \mid \lambda x.\, e \mid ee \mid \mathrm{reset}_{\mathrm{p}}\, e \mid \widehat{e}_p e$$

Logical Formulae

$$w \quad ::= c \mid ww \qquad c \in \Sigma_{\mathrm{sem}}$$

Values

$$v \quad ::= x \mid w \mid () \mid \lambda x.\, e$$

Reduction rules

$$\begin{array}{llcl}
(\beta_v) & (\lambda x.\, e)v & \rightsquigarrow & e\{x := v\} \\
(E_r) & \mathrm{reset}_{\mathrm{p}}\, v & \rightsquigarrow & v \\
(E_s) & \mathrm{reset}_{\mathrm{p}}\, \widehat{e}_p k & \rightsquigarrow & \mathrm{reset}_{\mathrm{p}}\;\; e\; (\lambda x.\, \mathrm{reset}_{\mathrm{p}}\, kx) \\
(\cong) & (\widehat{e}_p k)e_2 & \rightsquigarrow & \widehat{e}_p(\lambda x.(kx)e_2) \\
 & v_1(\widehat{e}_p k) & \rightsquigarrow & \widehat{e}_p(\lambda x.\, v_1(kx)) \\
 & \mathrm{reset}_{\mathrm{q}}\, \widehat{e}_p k & \rightsquigarrow & \widehat{e}_p(\lambda x.\, \mathrm{reset}_{\mathrm{q}}\, kx) \qquad p \neq q
\end{array}$$

The continuation bubble expands, devouring terms in its context . . .

The $\leftrightarrows$ calculus over Σ_{sem}

Expressions

$$e, k \quad ::= x \mid w \mid () \mid \lambda x.\, e \mid ee \mid \mathrm{reset}_{\mathrm{p}}\, e \mid \widehat{e}_p e$$

Logical Formulae

$$w \quad ::= c \mid ww \qquad c \in \Sigma_{\mathrm{sem}}$$

Values

$$v \quad ::= x \mid w \mid () \mid \lambda x.\, e$$

Reduction rules

$$\begin{array}{lll}
(\beta_v) & (\lambda x.\, e)v & \rightsquigarrow \; e\{x := v\} \\
(E_r) & \mathrm{reset}_{\mathrm{p}}\, v & \rightsquigarrow \; v \\
(E_s) & \mathrm{reset}_{\mathrm{p}}\, \widehat{e}_p k & \rightsquigarrow \; \mathrm{reset}_{\mathrm{p}}\;\; e\; (\lambda x.\, \mathrm{reset}_{\mathrm{p}}\, kx) \\
(\cong) & (\widehat{e}_p k)e_2 & \rightsquigarrow \; \widehat{e}_p(\lambda x.(kx)e_2) \\
 & v_1(\widehat{e}_p k) & \rightsquigarrow \; \widehat{e}_p(\lambda x.\, v_1(kx)) \\
 & \mathrm{reset}_{\mathrm{q}}\, \widehat{e}_p k & \rightsquigarrow \; \widehat{e}_p(\lambda x.\, \mathrm{reset}_{\mathrm{q}}\, kx) \qquad p \neq q
\end{array}$$

. . . unless stopped, or pricked, by reset$_{\mathrm{p}}$ indexed by the same prompt.

The $\leftrightarrows$ calculus over Σ_{sem}

Expressions

$$e, k \quad ::= x \mid w \mid () \mid \lambda x.\, e \mid ee \mid \mathrm{reset_p}\, e \mid \widehat{e}_p e$$

Logical Formulae

$$w \quad ::= c \mid ww \qquad c \in \Sigma_{\mathrm{sem}}$$

Values

$$v \quad ::= x \mid w \mid () \mid \lambda x.\, e$$

Reduction rules

$$\begin{array}{llcl}
(\beta_v) & (\lambda x.\, e)v & \rightsquigarrow & e\{x := v\} \\
(E_r) & \mathrm{reset_p}\, v & \rightsquigarrow & v \\
(E_s) & \mathrm{reset_p}\, \widehat{e}_p k & \rightsquigarrow & \mathrm{reset_p}\ e\ (\lambda x.\, \mathrm{reset_p}\, kx) \\
(\cong) & (\widehat{e}_p k)e_2 & \rightsquigarrow & \widehat{e}_p(\lambda x.(kx)e_2) \\
 & v_1(\widehat{e}_p k) & \rightsquigarrow & \widehat{e}_p(\lambda x.\, v_1(kx)) \\
 & \mathrm{reset_q}\, \widehat{e}_p k & \rightsquigarrow & \widehat{e}_p(\lambda x.\, \mathrm{reset_q}\, kx) \qquad p \neq q
\end{array}$$

$$\mathrm{shift_p}\, e \overset{\mathrm{def}}{=} \widehat{e}_p(\lambda x.\, x)$$

We introduce `shift` as an abbreviation for the initial bubble.

Features of $\leftrightarrows$

- Evaluation order is built-in
- Can evaluate top-to-bottom or bottom-up
  (top-to-bottom evaluation is deterministic)
- Not even weakly normalizing

Imperfect analogy: `LetTac`, the language of Coq tactics

The slide notes several features of the calculus. When talking about dynamics, the order of effects become important: think of the order of deposits and withdrawals and the consequences of withdrawals before deposits. In CPS, the order is encoded in data dependencies of continuations. In our calculus, the evaluation order is declared more directly.

Intuition of shift and prompt

Prompt marks the spot

$$\begin{array}{rl} & \ldots \mathrm{reset}_{\mathrm{p}} \ldots\ \mathrm{reset}_{\mathrm{q}} \ldots\ \mathrm{shift}_{\mathrm{p}}(\lambda z.\, \exists_i\ (z i)) \ldots \\ \rightsquigarrow^* & \ldots \mathrm{reset}_{\mathrm{p}}\ \exists_i\ \ \mathrm{reset}_{\mathrm{p}} \ldots\ \mathrm{reset}_{\mathrm{q}} \ldots i \ldots \end{array}$$

Let's consider the term on the slides, where insignificant parts are elided, with .... We assume that none of the elided parts contain reset_p with exactly the prompt p.

On the second line is the result of evaluating (normalizing) the term.

Intuition of shift and prompt

Prompt marks the spot

$$\begin{array}{rl} & \ldots \mathrm{reset}_\mathrm{p} \ldots\ \mathrm{reset}_\mathrm{q} \ldots\ \mathrm{shift}_\mathrm{p}(\lambda z.\, \exists_i\ (zi)) \ldots \\ \rightsquigarrow^* & \ldots \mathrm{reset}_\mathrm{p}\ \exists_i\ \mathrm{reset}_\mathrm{p} \ldots\ \mathrm{reset}_\mathrm{q} \ldots i \ldots \end{array}$$

In effect, $\exists_i$ moved to the place marked by reset_{p}. So, prompt marks the locus of movement.

Intuition of shift and prompt

Prompt marks the spot

$$
\begin{array}{ll}
 & \ldots \mathrm{reset}_{\mathrm{p}} \ldots\ \mathrm{reset}_{\mathrm{q}} \ldots\ \mathrm{shift}_{\mathrm{p}}(\lambda z.\, \exists_i\, (z i))\ \ldots \\
\rightsquigarrow^{*} & \ldots \mathrm{reset}_{\mathrm{p}}\, \exists_i\ \mathrm{reset}_{\mathrm{p}} \ldots\ \mathrm{reset}_{\mathrm{q}} \ldots i \ldots
\end{array}
$$

And the whole shift expression is replaced by the quantified variable i.

Intuition of shift and prompt

Prompt marks the spot

$$\begin{aligned} & \ldots \mathrm{reset}_{\mathrm{p}} \ldots\ \mathrm{reset}_{\mathrm{q}} \ldots\ \mathrm{shift}_{\mathrm{p}}(\lambda z.\, \exists_i\ (z i)) \ldots \\ \rightsquigarrow^{*} \quad & \ldots \mathrm{reset}_{\mathrm{p}}\ \exists_i\ \ \mathrm{reset}_{\mathrm{p}} \ldots\ \mathrm{reset}_{\mathrm{q}} \ldots i \ldots \end{aligned}$$

Resets with the prompts other than p may appear in-between (see $\mathrm{reset}_{\mathrm{q}}$). They are “passed through.”

Semantic lexicon

$\mathcal{L}_{\text{sem}}$: mapping of constants of Σ_{abs} to $\leftrightarrows$ over Σ_{sem}

NP	$\mapsto$	$\text{unit} \to e$
N	$\mapsto$	$(\text{unit} \to e) \to (\text{unit} \to t)$
S	$\mapsto$	$\text{unit} \to t$
D	$\mapsto$	t
Pedro	$\mapsto$	$\lambda u.\, \mathit{Pedro}$
donkey	$\mapsto$	$\lambda x.\, \lambda u.\, \mathit{donkey}(xu)$
beat	$\mapsto$	$\lambda o.\, \lambda s.\, \lambda u.(\lambda x.\, \lambda y.\, \mathit{beat}\, yx)\ (su)\ (ou)$
a	$\mapsto$	$\lambda x.\, \lambda u.\, \text{shift}_{\text{q}}(\lambda z.\, \exists_j\ (x(\lambda u.\, j)u) \wedge zj)$
fullstop	$\mapsto$	$\lambda x.\, \text{reset}_{\text{q}}\ x()$

Semantic lexicon

$\mathcal{L}_{\text{sem}}$: mapping of constants of Σ_{abs} to $\leftrightarrows$ over Σ_{sem}

$$
\begin{array}{lll}
\mathsf{NP} & \mapsto & \text{unit} \to e \\
\mathsf{N} & \mapsto & (\text{unit} \to e) \to (\text{unit} \to t) \\
\mathsf{S} & \mapsto & \text{unit} \to t \\
\mathsf{D} & \mapsto & t \\
\mathsf{Pedro} & \mapsto & \lambda u.\, \mathit{Pedro} \\
\mathsf{donkey} & \mapsto & \lambda x.\, \lambda u.\, \mathit{donkey}(xu) \\
\mathsf{beat} & \mapsto & \lambda o.\, \lambda s.\, \lambda u.(\lambda x.\, \lambda y.\, \mathit{beat}\, yx)\ (su)\ (ou) \\
\mathsf{a} & \mapsto & \lambda x.\, \lambda u.\, \text{shift}_{\text{q}}(\lambda z.\, \exists_j\ (x(\lambda u.\, j)u) \wedge zj) \\
\mathsf{fullstop} & \mapsto & \lambda x.\, \text{reset}_{\text{q}}\ x()
\end{array}
$$

Abstract and semantic forms

$$
\begin{array}{ll}
 & \mathcal{L}_{\text{sem}}(\mathsf{fullstop}\ (\mathsf{beat}\ (\mathsf{a}\ \mathsf{donkey})\ \mathsf{Pedro})) \\
\rightsquigarrow^{*} & \exists_j\ (\mathit{donkey}\, j) \wedge (\mathit{beat}\, j\, \mathit{Pedro})
\end{array}
$$

$\mathcal{L}_{\mathrm{sem}}(t_{donkey})$ is a term in $\leftrightarrows$, which includes shift and reset. If we normalize the term by applying the reduction rules described above, we end up with the logical formula at the bottom. This is a value, and it is taken to be the semantic denotation of our sentence.

What about types?

- Our calculus is typed
- Types are very useful
- All types are inferred
- Types are only an approximation of dynamic behavior (the typed calculus is still not normalizing)

Alas, this slide is all I have time now to say about types.

Outline

Hypotheses

Prompts

$p_\forall, p_\exists, p_{it}, p_s$

Ordering of prompts

- Universals cannot scope wider than a sentence:
  $p_\forall$ is set at the sentence boundary.
- Indefinites, pronouns may scope discourse-wide:
  $p_\exists$, p_{it} are not set at the sentence boundary.
- Coordinator limits the scope of quantification and binding:
  `and` sets $p_\forall, p_\exists, p_{it}$
- Negation sets $p_\exists$.

Let us come back to the happy donkeys, our original puzzles. To analyze them, we will pose four prompts. The prompt p_s, the sentence prompt, is the target for the coordination movement, among other things. It is set at each sentence's boundary. The other prompts are self-explanatory.

We make several assumptions, listed on the slide, about the setting of the prompts.

Hypotheses

Prompts

$p_\forall, p_\exists, p_{it}, p_s$

Ordering of prompts

- Universals cannot scope wider than a sentence:
  $p_\forall$ is set at the sentence boundary.
- Indefinites, pronouns may scope discourse-wide:
  $p_\exists$, p_{it} are not set at the sentence boundary.
- Coordinator limits the scope of quantification and binding:
  `and` sets $p_\forall, p_\exists, p_{it}$
- Negation sets $p_\exists$.
- To prevent "scope extrusion," quantifiers *must* set p_{it}.

The last assumption is forced upon us by general considerations, to prevent quantified variables from “leaking out.” It is this assumption that is responsible indefinites’ binding into negation but not out of negation. Negation restricts the scope of indefinites, and indefinites limit the scope of binding.

Outline

What are Abstract Categorial Grammars (ACG)?

Why ACGs

Direct dynamic logic meta-calculus

What about the original puzzles?

▶ **Live demo**

Live demo

(2) Every donkey enters. $\star$It brays.

(3) It-is-not-the-case-that a donkey enters. $\star$It brays.

(6) A donkey enters. It-is-not-the-case-that it brays.

Conclusions

Combination of dynamic logic with ACG

- Denotations are computed
- Computation is expressed in a meta-language with built-in effects
  (no type lifting, no CPS)
- Delimited control gives us both semantic power and easy parsing
- Delimited dynamic binding (and quantifier scope)

The scope of quantification/binding is the interplay

- A quantifier phrase's targeting a particular prompt
- The context's setting the prompts

`http://okmij.org/ftp/Computation/gengo/symantics2.ml`

We have described a computational ACG, emphasizing evaluation as a process to produce a (semantic) logical formula. Computational ACG gives us a principled way to assign different quantifiers different scope-taking abilities, maintaining consistency with Minimalism and avoiding free-wheeling Quantifier Raising.

Computational ACG let us relate quantification and binding: the same mechanism controls the scope of both.

We have implemented Computational ACG by embedding them in OCaml. Using the (unmodified) OCaml system, we can compute ACG yields and, more importantly, denotations. We can do that interactively, using OCaml top-level (interpreter). There is no longer any need to computing denotations by hand. We (computer, actually) can thus handle more complex examples.