Lecture 24: Continuations

CS 181O
Spring 2016
Kim Bruce

Midterm

- Lambda calculus
  - Define semantics using lambda calculus
- Propositional & predicate logic
  - Syntax & Semantics
  - Natural language
- Intensional/Modal logic
- Parsing
- Extend programs

Midterm

- Pick up before Thursday (9 a.m - 5 p.m.)
- Due 24 hours after pick up, but no later than Thursday at 5 p.m.

Ambiguity

- How do we interpret "someone saw everyone"
  - (someone saw) everyone ⇒ $\exists x\ \forall y\ (saw(x,y))$
  - someone (saw everyone) ⇒ $\forall y\ \exists x\ (saw(x,y))$
  - *Assuming domains is all persons, otherwise more complex!*
- Examine using continuations
  - Tool for understanding (and compiling) programming languages.
  - Applied to natural languages in early 2000's by Barker and Shan (independently).

Parse trees

- Harder:
  - [[John saw everyone]] = [[John]]([[saw everyone]])
  - [[John]]: $(e \to t) \to t$
  - [[saw]]: $e \to e \to t$
  - [[Everyone]]: $(e \to t) \to t$
- Now what???
  - should be $\forall x(saw(john,x)$ or
  - $\forall x\ (Person(x) \to saw(john,x))$

From noun phrases to ...?

- Rather than interpreting NP's as entities: e
  - Instead as properties satisfying the entity: $(e \to t) \to t$
  - Did it to make sense of quantifiers
  - Why stop there?
  - Do same with other grammatical forms!

Continuations

- Invented (many times) in computer science.
- Play a role in compiling functional programming languages, especially exceptions.
  - More recently replaced by A-normal form — bit simpler.
- Continuation passing style, do computation, but keep track of context that will finish the rest of the program.

Meanings

- Interpreted "Dorothy" as $\lambda P.\ (P\ d)$: $(e \to t) \to t$
- Interpret "Dorothy admired Alice"
  - Context of "Dorothy" is "_ admired Alice"
  - Context of "Alice" is "Dorothy admired _"
    - Type of "Alice" is also $(e \to t) \to t$
  - Context of "admired" is "Dorothy _ Alice"
    - Interpret "Dorothy _ Alice" as $\lambda P.\ (P\ d\ a)$: $(e \to e \to t) \to t$
    - Thus interpret "admired" as $\lambda P.\ (P\ admired)$: $((e \to e \to t) \to t) \to t$

Continuation

- Meaning of linguistic context of expression called its *continuation*
- Expressions can have lots of continuations
  - Hence we'll make the continuation a parameter of the meaning.
  - We can think of providing an argument to a function
  - ... or a function to an argument!

What is a continuation?

- Continuation is provided to an expression so can get meaning of sentence.
- Continuation is function type returning type t
  - Continuation of NP is of type $e \to t$
  - Continuation of intransitive verb is $(e \to t) \to t$
  - Continuation of transitive verb is $(e \to e \to t) \to t$
- Meaning functions will now take continuations as an argument to get meaning.

Computations

- Computations are functions that take a continuation and give a meaning (of type r)
  - type Cont a r = a -> r
  - type Comp a r = Cont a r -> r
- Examples:
  - meaning of NP: $(e \to t) \to t$
  - meaning of IV, CN: $((e \to t) \to t) \to t$
  - meaning of TV: $((e \to e \to t) \to t) \to t$
  - meaning of ADJ: ?

Continuation-Passing Semantics

- Make all meaning take a continuation parameter k
  - Constant [[c]] $\Rightarrow \lambda k.\ k\ c$
    - cpsConst:: a -> Comp a r
    - cpsConst c = \ k -> k c
  - *Trick to check work*: Get the original meaning by applying to identity function
    - (cpsConst c) (\x -> x) = (\ k -> k c) (\x -> x) = (\x -> x) c = c

Application?

- [[Dorothy cheered]]
  - [[Dorothy]] = λk. k dorothy:: Comp e t
    - where Comp e t = (e → t) → t
  - [[cheered]] = λk. k cheered:: Comp (e → t) t
    - where Comp (e → t) t = ((e → t) → t) → t
  - [[Dorothy cheered]]: Comp t t
    - where Comp t t = (t → t) → t
    - [[Dorothy cheered]] = λk. ... ???

CpsApply

- cpsApply m n = λk . n (λb. m (λa. k (a b)))
  - result is a function that takes a continuation k.
  - To use k, must:
    - evaluate n with a continuation that takes the value b of n, and then
    - evaluates m with a continuation that takes the value a of m, and
    - finally applies k to the result of evaluating (a b)

CpsApply

- intSent_CPS (Sent np vp) =
  cpsApply (intVP_CPS vp) (intNP_CPS np)
  - Given continuation k:
  - Compute intNP_CPS np, call it b
  - Compute intVP_CPS vp, call it a
  - Apply k to (a b)
  - *Work out intSent_CPS(Sent Dorothy Cheered)*

CpsApply

- (intDET_CPS Every) (intCN_CPS Boy)
  - Given continuation k:
  - Compute intNP_CPS np, call it b
  - Compute intVP_CPS vp, call it a
  - Apply k to (a b)

More CPS

- What about quantifiers?
  - [[everyone]] = λk. ∀x ((Person x) → k x)
  - [[someone]] = λk. ∃x ((Person x) ∧ k x)
  - *What is scope of x? Includes k!*
- Abstract to quantifiers:
  - [[every]] = λk λP. k(λQ.∀x ((Q x) → P x)
  - [[some]] = λk λP. k(λQ.∃x (Q x) ∧ P x)
  - [[the]] = λk λP. k(λQ.∃x ((...Q x) ∧ P x)
  - [[no]] = λk λP. k(λQ.¬∃x ((Q x) ∧ P x)

Example

[[every person]]

= (λk λP. k(λQ.∀x ((Q x) → P x))))(λk'. k' Person)

= (λP. (λk'. k' Person)(λQ.∀x ((Q x) → P x))

= (λP.(λQ.(∀x (Q x) → P x)Person)

= (λP.((∀x (Person x) → P x)

has type (e → t) → t

Same value as everyone, as expected!

Questions?