**Representação semântica
em
analisadores gramaticais**

Luiz Arthur Pagani (UFPR)

`arthur@ufpr.br`

1 Introdução

- Analisador gramatical (*parser*) é normalmente apenas analisador sintático: o resultado do processamento é apenas uma representação da estrutura sintática da expressão analisada.
- Objetivo aqui: analisadores gramaticais com interpretação semântica

2 Pereira & Shieber

- Representação semântica de *every*:
 $(X \wedge S1) \wedge (X \wedge S2) \wedge \text{all}(X, S1 \Rightarrow S2)$ (ao invés de
 $P \wedge Q \wedge \text{all}(X, P @ X \Rightarrow Q @ X)$, esperado para
 $\lambda P. \lambda Q. \forall x. (P x) \rightarrow (Q x)$)
- Representação semântica de *Shrdlu*: $(\text{shrdlu} \wedge P) \wedge P$ (ao invés
de $P \wedge P @ \text{shrdlu}$, esperado para $\lambda P. (P \text{ shrdlu})$)
- Representação semântica de *wrote*: $X \wedge Y \wedge \text{wrote}(X, Y)$ (ao invés
de $Y \wedge X \wedge \text{wrote}(X, Y)$, esperado para $\lambda y. \lambda x. \text{wrote}(x, y)$)

Como reconhecem os próprios autores (p. 101):

A forma lógica de *Shrdlu* parece contra-intuitiva, porque não corresponde à codificação de nenhuma expressão- λ . A posição que deveria ser ocupada por uma variável é ocupada pela constante *shrdlu*.

A execução parcial da aplicação pode resultar em expressões bizarras, justamente por causa da execução parcial. Apenas uma parte da tarefa da redução- β é executada, o restante acaba sendo realizado durante o processamento, quando as variáveis envolvidas são instanciadas. Portanto, não devemos nos preocupar tanto com o fato de que a codificação das expressões- λ que estamos usando apresentem algumas propriedades que o cálculo abstratamente não apresenta.

Apesar disto, o programa constrói as representações corretas (que os autores chamam de forma lógica (*logical form*)):

```
?- s(LF, [terry, wrote, shrdlu], []).
```

```
LF=wrote(terry, shrdlu) ?;
```

```
no
```

```
?- s(LF, [every, program, halts], []).
```

```
LF=all(A, program(A)=>halts(A)) ?;
```

```
no
```

```
?- s(LF, [every, student, wrote, a, program], []).
```

```
LF=all(A, student(A)=>exists(B, program(B)&wrote(A, B))) ?;
```

```
no
```

3 Covington

- Quantificadores: conectivos implícitos —
 $\text{some}(A, \text{cat}(A), \text{meow}(A))$ e $\text{all}(A, \text{dog}(A), \text{bark}(A))$, ao invés
de $\text{some}(A, \text{cat}(A) \ \& \ \text{meow}(A))$ e $\text{all}(A, \text{dog}(A) \Rightarrow \text{bark}(A))$
- Nome próprio: como no de Pereira & Shieber
- Verbo transitivo direto: como esperado

Para construir a representação semântica de *every dog chased a cat*, este analisador também funciona adequadamente:

```
?- s(Sem, [every, dog, chased, a, cat], []).  
Sem=all(A, dog(A), some(B, cat(B), chase(A, B))) ?;  
no
```

4 Escopo

Nenhum dos dois analisadores apresenta as duas interpretações de uma sentença ambígua como *every student wrote a program* ou *every dog chased a cat* (uma na qual o quantificador universal tem escopo sobre o existencial e outra na qual o existencial tem escopo sobre o universal).

Em Pereira & Shieber há uma proposta de extensão do analisador para lidar com escopo. Mas Covington, apesar de reconhecer a importância do fenômeno, apenas sugere, como exercício para o leitor, a implementação de um predicado que execute o alçamento do quantificador.

De qualquer maneira, a questão do escopo está fora do âmbito da presente proposta.

5 Incluindo um histórico derivacional

Segundo Pereira & Shieber (ps. 95–96):

o uso de variáveis do Prolog para codificar variáveis da LPO [lógica de primeira ordem] é incorreto, configurando um caso de confusão entre variáveis do objeto (as da forma lógica) e variáveis da metalinguagem (as do Prolog — a metalinguagem usada aqui para descrever a relação entre as cadeias lingüísticas e as formas lógicas). Seria possível evitar esta confusão entre variáveis do objeto e da metalinguagem através de uma descrição um pouco mais complexa. No entanto, quando este abuso de notação é adequadamente compreendido, é provável que ele não cause nenhum problema e ainda traga benefícios substanciais em relação à simplicidade do programa.

Esta opinião faz parecer que a escolha não causa danos, apesar de não ser “logicamente correta”, mas não é bem assim: a adaptação mais evidente para estes analisadores resulta na construção de uma derivação estrutural inadequada em relação às etapas de sua construção.

A adaptação proposta consiste apenas na adaptação do argumento para a representação semântica, documente nele sua derivação.

Assim, a representação de *Pedro* é $[p, []]$, indicando que ele denota o indivíduo p e sua derivação não depende de nada (lista vazia); a representação de *dormiu* fica como $[X^{\text{dormir}}(X), []]$, também nos informando que a denotação $X^{\text{dormir}}(X)$ não foi construída a partir de mais nada. Observando a regra de estruturação do sintagma nominal, para o mesmo *Pedro*, constata-se que ela construiria a representação $[(p^P)^P, [p, []]]$; ela nos diz que a denotação agora é $(p^P)^P$, obtida a partir de $[p, []]$.

Se pedirmos para analisar *Pedro dormiu*, o resultado é bastante estranho:

```
?- analisa([pedro,dormiu]).
```

```
s:
```

```
    dormir(pedro)
```

```
        (pedro^dormir(pedro))^dormir(pedro)
```

```
            pedro
```

```
        pedro^dormir(pedro)
```

```
yes ? ;
```

```
no
```

6 Construindo uma derivação coerente

No novo programa, implementa-se a sugestão de Pereira & Shieber, empregando as variáveis do Prolog apenas para guiar a introdução das variáveis das representações; estas variáveis da metalinguagem da representação terão o formato $x(N)$, para qualquer número inteiro N (as do Prolog, portanto, são variáveis da metalinguagem para a construção da representação; ambas são metavariables, mas de metalinguagens diferentes); além disso, como é exigido pelas teorias de prova, toda variável introduzida na análise não pode ter sido usada antes.

Como exemplo, podemos ver abaixo a análise de *todo menino correu*:

```

analisa([todo, menino, correu]).
qualquer(x(3),m@x(3)=>c@x(3))
<== (x(2)^qualquer(x(3),m@x(3)=>x(2)@x(3)))@c
    x(2)^qualquer(x(3),m@x(3)=>x(2)@x(3))
<== (x(1)^x(2)^qualquer(x(3),x(1)@x(3)=>x(2)@x(3)))@m
    x(1)^x(2)^qualquer(x(3),x(1)@x(3)=>x(2)@x(3))
    m
    c
yes ;
no

```

7 Conclusão

- Derivação composicional
- Definição explícita da redução- β
- Produto $\times$ processo
- Programas disponíveis em <http://www.ufpr.br/~arthur>