Semântica Formal N - INF05516

Ciência da Computação - UFRGS

2006/2

Departamento de Informática Teórica - Instituto de Informática - UFRGS

Sintaxe Concreta I

- Existem princípios sólidos para sintaxe concreta.
  - Autômatos finitos e gramáticas livres do contexto
  - Geradores automáticos de *parsers*
- Disciplinas:
  - Linguagens Formais e Autômatos
  - Compiladores

Linguagem ARITH

Veículo para apresentar conceitos básicos sobre

- Sintaxe Abstrata (brevemente)
- Semântica Operacional
- Semântica Denotacional
- Propriedades

Sintaxe Concreta II

- Sintaxe Concreta
  - As regras pelas quais programas podem ser expressos como sequências de caracteres
  - Lida com questões como palavras-chaves, identificadores, separadores ou terminadores de comandos, comentários, identação, etc
- Sintaxe concreta é importante na prática
  - para legibilidade, familiaridade, rapidez no *parsing*, eficácia da recuperação de erros, claridade das mensagens de erros de sintaxe

Sintaxe Abstrata

Vamos ignorar *parsing* e estudaremos construções na forma de *árvores de sintaxe abstrata* (produzidas por algum parser)

Uma árvore de sintaxe abstrata é

- conveniente para manipulação formal e por algoritmos
- independente da sintaxe concreta

Sintaxe Abstrata para ARITH I

Conjunto de árvores de sintaxe abstrata pode ser descrito através de uma *gramática abstrata*

$$\begin{array}{lcll} a & ::= & n & \textit{literais inteiros} \\ & | & a_1 + a_2 & \textit{soma} \\ & | & a_1 * a_2 & \textit{produto} \end{array}$$

note que a gramática descreve conjunto de árvores, logo não é necessário parênteses

inconveniente desenhar árvores

parênteses poderão ser necessários quando algumas árvores forem linearizadas

Sintaxe Abstrata para ARITH II

- Outra definição para a sintaxe de ARITH pode ser

$$\begin{array}{lcll} a & ::= & n & \textit{literais inteiros} \\ & | & SUM(a_1, a_2) & \textit{soma} \\ & | & PROD(a_1, a_2) & \textit{produto} \end{array}$$

- ambas as formas podem ser representadas pelo seguinte tipo em OCAML

```
type arith =
      Nrl of int
    | Sum of arith * arith
    | Prod  of arith * arith
```


Análise de ARITH

- Questões a serem respondidas:
  - Qual o "significado"de uma dada expressão de ARITH?
  - Como é a avaliação de expressões de ARITH?
  - Como o avaliador e o significado estão relacionados ?

Semântica Operacional *Big Step*

$a \Downarrow n$ quer dizer que a avalia para n

- afirma uma relação entre a e n
- $\Downarrow \subseteq \mathsf{ARITH} \times \mathbb{Z}$

regras de avaliação expressas na forma de regras de inferência

- Também chamadas regras de avaliação

Em geral, temos uma regra para cada construção da linguagem

Semântica Operacional *Big-Step*

$$\frac{}{n \Downarrow n} \quad \text{(NUM)}$$

$$\frac{a_1 \Downarrow n_1 \qquad a_2 \Downarrow n_2 \qquad n = n_1 + n_2}{a_1 + a_2 \Downarrow n} \quad \text{(SOM)}$$

$$\frac{a_1 \Downarrow n_1 \qquad a_2 \Downarrow n_2 \qquad n = n_1 + n_2}{a_1 \times a_2 \Downarrow n} \quad \text{(MUL)}$$

Também chamada de semântica operacional natural

- Parecidas com regras de inferência do cálculo de dedução natural

Como Ler as Regras?

- De cima para baixo, como regras de inferência:
  - Se as premisas são verdadeiras podemos inferir a conclusão
  - Ex.: se $a_1 \Downarrow 5$ e que $a_2 \Downarrow 7$ então podemos inferir que $a_1 + a_2 \Downarrow 12$
- Essas regras não impõem ordem de avaliação !!

Como Ler as Regras ?

- De baixo para cima:
  - Supor que queremos avaliar $a_1 + a_2$, ou seja queremos encontrar n tal que $a_1 + a_2 \Downarrow n$ é derivável pelas regras de avaliação
  - Por inspeção das regras notamos que o último passo na derivação de $a_1 + a_2 \Downarrow n$ **deve** ser a regra da adição
    - A conclusão das demais regras não "casam"com $a_1 + a_2 \Downarrow n$
    - Assim, devemos encontrar n_1 tal que $a_1 \Downarrow n_1$ é derivável pelas regras de avaliação, e n_2 tal que $a_2 \Downarrow n_2$ é derivável pelas regras de avaliação (recursivamente)

Como Ler as Regras?

Como há exatamente uma regra para cada tipo de expressão de ARITH dizemos que as regras são dirigidas a sintaxe

- A cada passo, no máximo uma regra é aplicável.
- Isto é o que permite um procedimento de avaliação simples como o descrito acima (ver também implementação em OCAML abaixo)

Exemplo de Avaliador *big-step* em OCAML

```
type arith =
      Nrl of int
    | Sum of arith * arith
    | Prod of arith * arith;;

let rec eval a =
    match a with
      Nrl n -> n
    | Sum (a1,a2)  -> eval(a1) + eval(a2)
    | Prod (a1,a2) -> eval(a1) * eval(a2);;

let anexp = Prod(Sum(Nrl 3, Nrl 5), Sum(Nrl 4, Nrl 2));;

eval anexp;;
```


Semântica Operacional *Small Step*

- $a \rightarrow a'$ quer dizer que expressão a reduz em um passo para expressão a'
- $\rightarrow \subseteq \text{ARITH} \times \text{ARITH}$
- também chamada de semântica operacional estrutural
- também usa regras de inferência mas, em um processo de redução de uma expressão até uma expressão completamente reduzida pode ser necessário construir mais do que uma árvore de derivação (cada árvore de derivação também semelhante a uma árvore de dedução natural)
- importante a definição de elementos de ARITH completamente reduzidos

Semântica Operacional *Small-Step*

Regras para Adição

$$\frac{a_1 \rightarrow a_1'}{a_1 + a_2 \rightarrow a_1' + a_2} \quad (\text{SOM1})$$

$$\frac{a_2 \rightarrow a_2'}{n_1 + a_2 \rightarrow n_1 + a_2'} \quad (\text{SOM2})$$

$$\frac{n \textit{ é a soma de } n_1 \textit{ e } n_2}{n_1 + n_2 \rightarrow n} \quad (\text{SOM3})$$

Regras de Avaliação *Small-Step* (cont.)

gras para Multiplicação

$$\frac{a_1 \longrightarrow a_1'}{a_1 * a_2 \longrightarrow a_1' * a_2} \quad \text{(MUL1)}$$

$$\frac{a_2 \longrightarrow a_2'}{n_1 * a_2 \longrightarrow n_1 * a_2'} \quad \text{(MUL2)}$$

$$\frac{n \textit{ é o produto de } n_1 \textit{ e } n_2}{n_1 * n_2 \longrightarrow n} \quad \text{(MUL3)}$$

Regras de Avaliação *Small-Step*

Também é dirigida a sintaxe

Note que, nesse caso, a ordem de avaliação foi fixada (esquerda para direita)

em geral mais do que uma regra para cada estrutura sintática

- regras de redução (ou reescrita), e
- regras de computação

não há regra para expressões que já estão completamente reduzidas

Small-Step em OCAML

```
exception Cannot_step;;

let rec step a =
  match a with
      Nrl n                  -> raise Cannot_step
    | Sum  (Nrl n1, Nrl n2)  -> Nrl (n1+n2)
    | Sum  (Nrl n1, a2 )     -> Sum (Nrl n1, step a2)
    | Sum  (a1, a2)          -> Sum (step a1, a2)
    | Prod (Nrl n1, Nrl n2)  -> Nrl (n1*n2)
    | Prod (Nrl n1, a2 )     -> Prod(Nrl n1, step a2)
    | Prod (a1, a2)          -> Prod(step a1, a2);;

let rec stepstar a =
   a :: (try stepstar (step a)
         with Cannot_step -> []);;

stepstar anexp;;
```


Semântica Denotacional

- Semântica Denotacional associa um significado matemático a cada construção da linguagem.
- Para ARITH, um significado apropriado para uma expressão é um número inteiro
- Notação: $[\![a]\!]$ é o inteiro denotado por a
- $[\![a]\!]$ é a aplicação de uma função ao argumento a
- Definição da função de denotação $[\![]\!] : \mathsf{ARITH} \rightarrow \mathbb{Z}$:

$$\begin{aligned} [\![n]\!] &= n \\ [\![a_1 + a_2]\!] &= [\![a_1]\!] + [\![a_2]\!] \\ [\![a_1 * a_2]\!] &= [\![a_1]\!] * [\![a_2]\!] \end{aligned}$$

Semântica Denotacional

É dirigida a sintaxe

- Sempre, não somente nesse caso

É composicional

- A denotação de uma expressão é expressa como uma função da denotação de suas sub-expressões.
- Isso nem sempre ocorre com outras formas de semântica

Semântica Denotacional

Para ARITH, a semântica denotacional é muito parecida com a semântica operacional natural

O significado de uma expressão é um objeto matemático, não uma entidade sintática

- em $[\![n]\!] = n$, o primeiro n é sintático e o segundo é um inteiro
- em $[\![a_1 + a_2]\!] = [\![a_1]\!] + [\![a_2]\!]$, o primeiro $+$ é sintático e o segundo é uma função matemática sobre inteiros

Propriedades da Semântica

- podemos afirmar (e provar) diversos fatos sobre nossa semântica
- A semântica operacional natural e a *small-step* concordam:
  - $\forall a \forall n.\ a \Downarrow n \quad \text{sse} \quad a \rightarrow^* n$
- $\rightarrow^*$ é o fecho reflexivo e transitivo de $\rightarrow$, ou seja $a \rightarrow^* a'$ sse $a = a'$ ou $\exists a''\ tq.\ a \rightarrow a'' \ \wedge\ a'' \rightarrow^* a'$

Propriedades da Semântica

- A semântica operacional e denotacional concordam
  - $\forall a \forall n.\ a \Downarrow n$ sse $[\![a]\!] = n$
  - ou, de forma equivalente, $\forall a.\ a \Downarrow [\![a]\!]$
  - ou, ainda $\forall a, a'.\ a \rightarrow a' \Rightarrow [\![a]\!] = [\![a']\!]$
- Essa propriedade é as vezes chamada de segurança da semântica operacional

Propriedades da Semântica

Expressões ARITH sempre tem um valor

- $\forall a \exists n.\ a \Downarrow n$

ARITH é uma linguagem determinística

- todas as expressões avaliam para no máximo um valor
  $\forall a. \forall n \forall n'.\ a \Downarrow n \wedge a \Downarrow n' \Rightarrow n = n'$

Para ARITH, esses dois fatos podem ser provados diretamente da semântica denotacional pois

- a função de denotação é uma função total de expressões para inteiros

Exercícios

1. Prove, usando as regras da semântica que
   (a) $(7 + 3) * 2 \Downarrow 20$
   (b) $(7 + 3) * 2 \rightarrow^* 20$
2. Usando a semântica operacional *big-step* de ARITH
   (a) encontre o numeral n tal que $(4 + 1) + (2 + 2) \Downarrow n$
   (b) prove que $(3 + 2) \times (1 + 4) \Downarrow 25$
3. Usando a semântica Operacional *small-step* de ARITH
   (a) prove que $(1 + 2) + (4 + 3) \rightarrow^* 10$. Mostre a derivação completa de cada passo da avaliação.
4. As regras *small step* para ARITH apresentadas especificam uma avaliação da esquerda para direita. Defina um conjunto de regras que especifiquem avaliação da direita para esquerda
5. Implemente os avaliadores *small-step* e *big-step* de ARITH em OCAML
6. Mostre que $[\![(1 + 2) + (4 + 3)]\!] = 10$.

Propriedades da Semântica

A avaliação *small-step* de um não valor sempre progride

- $\forall a.\ a$ *não é um valor* $\Rightarrow \exists a'\ tq\ a \rightarrow a'$

Teoremas de progresso são uma das formas de afirmar segurança de linguagens de programação

- aqui progresso é trivial, mas nem sempre é assim

A seguir veremos linguagens mais interessantes com técnicas de prova para tais teoremas (várias formas de indução)