Theory of Programming Languages
程序设计语言理论

张昱

Department of Computer Science and Technology
University of Science and Technology of China

September, 2008

第3章　一种简单的语言$\mathcal{L}$\{num, str\}

3.1 概述-1

- $\mathcal{L}$**{num, str}**
  - 支持在自然数上的基本算术运算以及字符串上的简单计算
  - 包含一种能在特定作用域内将表达式值绑定到一个变量的语言构造
- 具体语法 (**Concrete Syntax**) [PFPL, 7]
  - 将表达式表示成字符串的一种手段(写在纸上或用键盘输入)
  - 通常希望有好的可读性并且没有二义性.

3.1 概述-2

- 抽象语法(Abstract Syntax) [PFPL, 8]
  - 揭示语言的层次结构和绑定结构
    e.g. 抽象语法树(abstract syntax tree, AST), 抽象绑定树(abstact binding tree, ABT)
- 语法对象(Syntactic Objects) [PFPL, 5,6]
  - 字符串, 名字, AST, ...
- 分析 (Parsing)
  - 将具体语法翻译成抽象语法的过程

3.1 概述-3

- 静态语义(Static Semantics) [PFPL, 9]
  - 由一组用来约束程序形成的规则组成，称为类型系统.
- 动态语义(Dynamic Semantics) [PFPL, 10,12]
  - 描述程序将如何执行
  - 表示方法
    - 结构语义(Structural semantics) [PFPL, 10]
      - 上下文语义(Contextual semantics)
    - 求值语义(Evaluation semantics), [PFPL, 12]

3.2 语法对象

3.2.1　符号和字符串 [PFPL, 5.1 5.2]

3.2.2　抽象语法树 [PFPL, 5.3]

3.2.3　抽象绑定树 [PFPL, 6]

3.2.1 符号和字符串-1

- 符号(symbols): 字符、变量名、域名等等.
  - 断言

    x sym ：x是一个符号

    $x \# y$，其中 x sym 且 y sym，x和y是不同的符号
- 字符串(strings): 字符、变量名、域名等等.
  - 字母表(alphabet)Σ ：一组字符的集合.
  - 断言

    c char ：c是一个字符.

    $\Sigma \vdash s$ str：在Σ上定义字符串，由以下规则归纳定义

$$\frac{}{\Sigma \vdash \varepsilon \text{ str}} \qquad \frac{\Sigma \vdash c \text{ char} \qquad \Sigma \vdash s \text{ str}}{\Sigma \vdash c \cdot s \text{ str}} \qquad \textbf{(5.1)}$$

一个字符串本质上是一个字符序列，空串是空序列。

3.2.1 符号和字符串-2

- 字符串的归纳原理
  - To show P s whenever s str, it is enough to show

    1) P ε, and

    2) if P s and c sym, then P(c · s)

- 字符串的连接
  - 断言 $s_1 \hat{} s_2 = s \text{ str}$： s是字符串$s_1$和$s_2$连接组成的串.
  - 归纳定义:

$$\frac{}{\varepsilon \hat{} s = s} \qquad \frac{s_1 \hat{} s_2 = s}{(c \cdot s_1) \hat{} s_2 = c \cdot s} \qquad \textbf{(5.2)}$$

该断言具有模式 (∀, ∀, ∃!)

3.2.2 抽象语法树-1

❖ 抽象语法树**Abstract Syntax Tree(AST)** [PFPL, 5.3]

➢ an ordered tree in which certain symbols (operators) label the nodes

➢ Each operator is assigned an arity (number of children)

❖ **Operator signature, Ω**

➢ 是一组形如 $\mathrm{ar}(o) = n$ 的断言，其中 o sym、n nat
如果 $\Omega \vdash \mathrm{ar}(o) = n$ 和 $\Omega \vdash \mathrm{ar}(o) = n'$，则 $n = n'$ nat .

➢ 归纳定义

$$\frac{\Omega \vdash \mathrm{ar}(o) = \mathrm{zero}}{o()\ \mathrm{ast}} \qquad \frac{\Omega \vdash \mathrm{ar}(o) = n \quad a_1\ \mathrm{ast}\ \cdots\ a_n\ \mathrm{ast}}{o(a_1, \cdots, a_n)\ \mathrm{ast}} \qquad \textbf{(5.3)}$$

3.2.2 抽象语法树-2

- 结构归纳原理(Principle of Structural Induction)
  - To show $\mathcal{P}(a \text{ ast})$, it is enough to show that $\mathcal{P}$ is closed under Rules **(5.3)**. That is,

    if $\Omega \vdash \text{ar}(o) = n$, then we are to show that

    if $\mathcal{P}(a_1 \text{ ast}), \cdots, \mathcal{P}(a_n \text{ ast})$

    then $\mathcal{P}(o(a_1, \cdots, a_n) \text{ ast})$
  - 例如，AST高度(断言模式为$(\forall, \exists!)$)的归纳定义

$$\frac{\text{hgt}(a_1) = h_1 \quad \cdots \quad \text{hgt}(a_n) = h_n \quad \max(h_1, \cdots, h_n) = h}{\text{hgt}(o(a_1, \cdots, a_n)) = \text{succ}(h)} \quad \textbf{(5.4)}$$

3.2.2 抽象语法树-3

- 变量与代换(Variables and Substitution)
  - Variables are represented by names, and are given meaning by substitution.
  - 假设 $\mathcal{X} = x_1 \text{ ast}, \cdots, x_n \text{ ast}$ 是参数集 $\{x_1, \cdots, x_n\} | x_1 \text{ ast}, \cdots, x_n \text{ ast}$ ($x_1 \text{ sym}, \cdots, x_n \text{ sym}$)是一组假设序列；$x \# \mathcal{X}$表示 $x \notin \{x_1, \cdots, x_n\}$
  - 断言 $\mathcal{X} \vdash a \text{ ast}$ 由以下规则归纳定义

$$\frac{}{\mathcal{X}, x \text{ ast} \vdash x \text{ ast}}$$

$$\frac{\Omega \vdash \text{ar}(o) = n \quad \mathcal{X} \vdash a_1 \text{ ast} \quad \cdots \quad \mathcal{X} \vdash a_n \text{ ast}}{\mathcal{X} \vdash o(a_1, \cdots, a_n) \text{ ast}} \qquad \textbf{(5.5)}$$

3.2.2 抽象语法树-4

❖ 变量与代换(Variables and Substitution)

➢ 归纳原理: To show $\mathcal{P}(\mathcal{X} \vdash a \text{ ast})$, it is enough to show

1) $\mathcal{P}(\mathcal{X}, x \text{ ast} \vdash x \text{ ast})$.

2) If $\Omega \vdash \text{ar}(o) = n$, and if

$$\mathcal{P}(\mathcal{X} \vdash a_1 \text{ ast}), \cdots, \mathcal{P}(\mathcal{X} \vdash a_n \text{ ast})$$

then $\mathcal{P}(\mathcal{X} \vdash o(a_1, \cdots, a_n) \text{ ast})$.

$\mathcal{X}$中的参数都被当成原子对象，每个参数有自己的AST.

3.2.2 抽象语法树-5

变量代换

- 断言 $\mathcal{X} \vdash [a/x]b = c$：c是用a代换b中的x所得的结果
- 归纳定义

$$\frac{}{\mathcal{X}, x \text{ ast} \vdash [a/x]x = a} \quad (5.6a)$$

$$\frac{x \# y}{\mathcal{X}, x \text{ ast}, y \text{ ast} \vdash [a/x]y = y} \quad (5.6b)$$

$$\frac{\mathcal{X} \vdash [a/x]b_1 = c_1 \quad \cdots \quad \mathcal{X} \vdash [a/x]b_n = c_n}{\mathcal{X} \vdash [a/x]o(b_1, \cdots, b_n) = o(c_1, \cdots, c_n)} \quad (5.6c)$$

- **Theorem 5.1.** 如果$\mathcal{X} \vdash a \text{ ast}$且$\mathcal{X}, x \text{ ast} \vdash b \text{ ast}$($x \# \mathcal{X}$)
则存在一个唯一的c使得$\mathcal{X} \vdash [a/x]b = c$且$\mathcal{X} \vdash c \text{ ast}$

3.2.2 抽象语法树-6

证明(Theorem 5.1)：

在上下文$\mathcal{X}, x$ ast 上对**b**进行结构归纳:

1. 由于$\mathcal{X}, x \text{ ast} \vdash x \text{ ast}$，需要证明存在唯一的**c**使得:

$$\mathcal{X} \vdash [a/x]x = c$$

考虑规则**(5.6a)**，故选择**c**为**a**是充分且必要的.

2. 如果$\mathcal{X}, x \text{ ast}, y \text{ ast}$ ($x \# \mathcal{X}$),则由规则**(5.6b)**，选择**c**为**y**是充分且必要的.

3. 如果$b = o(b_1, \cdots, b_n)$，则由归纳假设，存在唯一的使得

$$\mathcal{X} \vdash [a/x]b_1 = c_1, \cdots, \mathcal{X} \vdash [a/x]b_n = c_n$$

由规则**(5.6c)**，**c**只能取 $o(c_1, \cdots, c_n)$

3.2.3 抽象绑定树-1

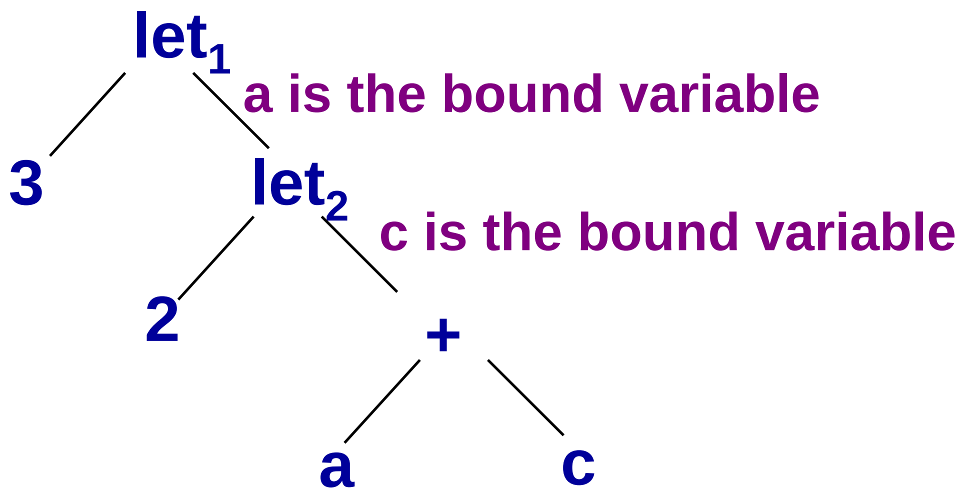
抽象语法树：反映了语法的层次结构

抽象绑定树(Abstract binding trees, ABT)：增加了绑定(binding)和作用域(scope)的概念.

- ABT
  - ABT: extends AST with an abstractor
  - Abstractor x.a: 将变量 x 绑定到ABT a, a 称为绑定的作用域. 受约束的变量x仅在a内是有意义的
  - arity of an operator
    - 是自然数的有限序列$(n_1, \cdots, n_k)$, k表示参数的个数，n_i 表示第i个参数中约束变量的数目(valence).
    - e.g. let x be exp1 in exp2   ar(let)=(0,1)
      //exp2 has a variable named x.

3.2.3 抽象绑定树-2

let$_1$ a be 3 in let$_2$ c be 2 in a+c   ar(let)=(0,1)



- operator signature Ω

  一组有限的形如 $\mathrm{ar}(o) = (n_1, \cdots, n_k)$ 的断言

- Ω上的良形(well-formed)ABT由参数化假言断言描述

  $\{x_1, \cdots, x_k\} | x_1\ \mathrm{abt}^0, \cdots, x_k\ \mathrm{abt}^0 \vdash a\ \mathrm{abt}^n$

  $x_1, \cdots, x_k$ 是a中的自由变量. $\mathcal{X} | \mathcal{A} \vdash a\ \mathrm{abt}^n$ $\mathcal{A} \vdash a\ \mathrm{abt}^n$

3.2.3 抽象绑定树-3

➢ 良形abt的归纳定义

$$\frac{}{\mathcal{X}, x \mid \mathcal{A}, x \text{ abt}^0 \vdash x \text{ abt}^0} \quad \textbf{(6.1a)}$$

$$\frac{\text{ar}(o) = (n_1, \cdots, n_k) \quad \mathcal{X} \mid \mathcal{A} \vdash a_1 \text{ abt}^{n_1} \quad \cdots \quad \mathcal{X} \mid \mathcal{A} \vdash a_k \text{ abt}^{n_k}}{\mathcal{X} \mid \mathcal{A} \vdash o(a_1, \cdots, a_k) \text{ abt}^0} \quad \textbf{(6.1b)}$$

$$\frac{\mathcal{X}, x' \mid \mathcal{A}, x' \text{ abt}^0 \vdash [x' \leftrightarrow x]a \text{ abt}^n \quad (x' \notin \mathcal{X})}{\mathcal{X} \mid \mathcal{A} \vdash x.a \text{ abt}^{n+1}} \quad \textbf{(6.1c)}$$

$\mathcal{X}, x' \mid \mathcal{A}, x' \text{ abt}^0 \vdash [x' \leftrightarrow x]a \text{ abt}^n \quad (x' \notin \mathcal{X})$ 表示用**x'**替换**a**中的约束变量**x**所得的体是良形的.

则 $\mathcal{X} \mid \mathcal{A} \vdash x.a \text{ abt}^{n+1}$，即抽象子**x.a**相对于 $\mathcal{A}$ 是良形的

3.2.3 抽象绑定树-4

- 结构归纳原理

  To show that $\mathcal{P}(\mathcal{X}|\mathcal{A} \vdash a \text{ abt}^n)$ whenever $\mathcal{X}|\mathcal{A} \vdash a \text{ abt}^n$ it suffices to show the following:

1) $\mathcal{P}(\mathcal{X}, x|\mathcal{A}, x \text{ abt}^0 \vdash x \text{ abt}^0)$.

2) For any operator, o, of arity $(m_1, \ldots, m_k)$, if

$$\mathcal{P}(\mathcal{X}|\mathcal{A} \vdash a_1 \text{ abt}^{m_1}), \cdots, \mathcal{P}(\mathcal{X}|\mathcal{A} \vdash a_k \text{ abt}^{m_k})$$

   then $\mathcal{P}(\mathcal{X}|\mathcal{A} \vdash o(a_1, \cdots, a_k) \text{ abt}^0)$

3) If $\mathcal{P}(\mathcal{X}, x'|\mathcal{A}, x' \text{ abt}^0 \vdash [x' \leftrightarrow x]a \text{ abt}^n)$ for some/any $x' \notin \mathcal{X}$, then $\mathcal{P}(\mathcal{X}|\mathcal{A} \vdash x.a \text{ abt}^{n+1})$.

3.2.3 抽象绑定树-5

- 例，**abt的size s用断言** $|a\ \mathrm{abt}^n| = s$ 定义
  一般地，参数化假言断言
  $$|x_1\ \mathrm{abt}^0| = 1, \cdots, |x_k\ \mathrm{abt}^0| = 1 \vdash |a\ \mathrm{abt}^n| = s$$
  由以下规则归纳定义

$$\frac{}{\mathcal{S}, |x\ \mathrm{abt}^0| = 1 \vdash |x\ \mathrm{abt}^0| = 1} \quad \textbf{(6.2a)}$$

$$\frac{\mathcal{S} \vdash |a_1\ \mathrm{abt}^{n_1}| = s_1 \quad \cdots \quad \mathcal{S} \vdash |a_m\ \mathrm{abt}^{n_m}| = s_m \quad s = s_1 + \cdots + s_m + 1}{\mathcal{S} \vdash |o(a_1, \cdots, a_m)\ \mathrm{abt}^0| = s} \quad \textbf{(6.2b)}$$

$$\frac{\mathcal{S}, |x'\ \mathrm{abt}^0| = 1 \vdash |[x' \leftrightarrow x]a\ \mathrm{abt}^n| = s}{\mathcal{S} \vdash |x.a\ \mathrm{abt}^{n+1}| = s + 1} \quad \textbf{(6.2c)}$$

Theorem 6.1.每个良形的abt有唯一的size.

3.2.3 抽象绑定树-6

- **Apartness judgement:** $\mathcal{A} \vdash x \# a\ \mathrm{abt}^n$ ($\mathcal{A} \vdash a\ \mathrm{abt}^n$)

  the abt a does not involve the variable x except possibly as a bound variable.

Rules

$$\frac{x \# y}{\mathcal{A} \vdash x \# y\ \mathrm{abt}^0} \quad (6.3a)$$

$$\frac{\mathcal{A} \vdash x \# a_1\ \mathrm{abt}^{n_1} \quad \cdots \quad \mathcal{A} \vdash x \# a_k\ \mathrm{abt}^{n_k}}{\mathcal{A} \vdash x \# o(a_1, \cdots, a_k)\ \mathrm{abt}^0} \quad (6.3b)$$

$$\frac{\mathcal{A}, y\ \mathrm{abt}^0 \vdash x \# a\ \mathrm{abt}^n}{\mathcal{A} \vdash x \# y.a\ \mathrm{abt}^{n+1}} \quad (6.3c)$$

- **x is free in an abt, a, written $x \in a\ \mathrm{abt}$, iff it is not the case that $x \# a\ \mathrm{abt}$.**

3.2.3 抽象绑定树-7

- **Renaming of Bound Variables (α-equivalence)**

  两个**abt**是α-等价的，当且仅当它们只在约束变量的选取上有不同之处. $\mathcal{A} \vdash a =_\alpha b \text{ abt}^n$

Rules

$$\overline{\mathcal{A}, x \text{ abt}^0 \vdash x =_\alpha x \text{ abt}^0} \quad \textbf{(6.4a)}$$

$$\frac{\mathcal{A} \vdash a_1 =_\alpha b_1 \text{ abt}^{n_1} \quad \cdots \quad \mathcal{X} \vdash a_k =_\alpha b_k \text{ abt}^{n_k}}{\mathcal{X} \vdash o(a_1, \cdots, a_k) =_\alpha o(b_1, \cdots, b_k) \text{ abt}^0} \quad \textbf{(6.4b)}$$

$$\frac{\mathcal{A}, z \text{ abt}^0 \vdash [z \leftrightarrow x]a =_\alpha [z \leftrightarrow y]b \text{ abt}^n}{\mathcal{A} \vdash x.a =_\alpha y.b \text{ abt}^{n+1}} \quad \textbf{(6.4c)}$$

- **Theorem 6.3.**α-等价是自反的、对称的和传递的.

3.2.3 抽象绑定树-8

❖ Substitution:

代换是将一个abt中一个自由变量的所有出现替换为另一个abt

$$\mathcal{A} \vdash [a/x]b = c \ \text{abt}^n$$

Rules

$$\frac{}{\mathcal{A} \vdash [a/x]x = a \ \text{abt}^0} \quad (6.5a)$$

$$\frac{x \# y}{\mathcal{A} \vdash [a/x]y = y \ \text{abt}^0} \quad (6.5b)$$

$$\frac{\mathcal{A} \vdash [a/x]b_1 = c_1 \ \text{abt}^{n_1} \quad \cdots \quad \mathcal{X} \vdash [a/x]b_k = c_k \ \text{abt}^{n_k}}{\mathcal{X} \vdash [a/x]o(b_1, \cdots, b_k) = o(c_1, \cdots, c_k) \ \text{abt}^0} \quad (6.5c)$$

$$\frac{\mathcal{A}, y' \ \text{abt}^0 \vdash [a/x]([y' \leftrightarrow y]b) = b' \ \text{abt}^n \quad y' \# \mathcal{A} \quad y' \neq x}{\mathcal{A} \vdash [a/x]y.b = y'.b' \ \text{abt}^n} \quad (6.5d)$$

- 由规则(6.5d)，有 $y.[y \leftrightarrow y']b' =_\alpha y'.b'$

3.2.3 抽象绑定树-9

❖ Substitution

Theorem 6.4.

1. If $\mathcal{A} \vdash a \text{ abt}^0$ and $\mathcal{A}, x \text{ abt}^0 \vdash b \text{ abt}^n$, then there exists $\mathcal{A} \vdash c \text{ abt}^n$ such that $\mathcal{A} \vdash [a/x]b = c \text{ abt}^n$.

2. If $\mathcal{A} \vdash a \text{ abt}^0$, $\mathcal{A} \vdash [a/x]b = c \text{ abt}^n$ and $\mathcal{A} \vdash [a/x]b = c' \text{ abt}^n$, then $\mathcal{A} \vdash c =_\alpha c' \text{ abt}^n$.

证明： 1. 对$\mathcal{A}, x \text{ abt}^0 \vdash b \text{ abt}^n$ 归纳证明

2. 对 $\mathcal{A} \vdash [a/x]b = c \text{ abt}^n$和$\mathcal{A} \vdash [a/x]b = c' \text{ abt}^n$ 联立归纳证明。

3.2.3 抽象绑定树-10

❖ Substitution

Theorem 6.5.

If $\mathcal{A} \vdash a =_\alpha a' \ \mathrm{abt}^0$, $\mathcal{A}, x \ \mathrm{abt}^0 \vdash b =_\alpha b' \ \mathrm{abt}^n$,

$\mathcal{A} \vdash [a/x]b = c \ \mathrm{abt}^n$ and $\mathcal{A} \vdash [a'/x]b' = c' \ \mathrm{abt}^n$,

then $\mathcal{A} \vdash c =_\alpha c' \ \mathrm{abt}^n$.

Proof. By rule induction on $\mathcal{A}, x \ \mathrm{abt}^0 \vdash b =_\alpha b' \ \mathrm{abt}^n$

3.2.3 抽象绑定树-11

若假设所有断言关于abt α-等价，则抽象子的形成规则可以简写为**(x#A)**:

$$\frac{\mathcal{A}, x\ \mathrm{abt}^0 \vdash a\ \mathrm{abt}^n}{\mathcal{A} \vdash x.a\ \mathrm{abt}^{n+1}} \qquad (6.6)$$

3.3 具体语法(Concrete Syntax)

- a means of representing expressions as strings (written on a page or entered using a keyboard)
- usually designed to enhance readability and to eliminate ambiguity.

3.3.1 Lexical Structure

3.3.2 Context-Free Grammars

3.3.3 Grammatical Structure

3.3.4 Ambiguity

3.3.5 Informal Conventions

3.3.1 Lexical Structure-1

❖ Lexical analysis (lexing)
  ➢ characters ➔ symbols (tokens)
    – white space (spaces, tabs, newlines, comments, ...)
      -- discarded by the lexical analyzer

❖ Lexical structure of $\mathcal{L}\{\text{num}, \text{str}\}$

Item	itm	::=	kwd | id | num | lit | spl
Keyword	kwd	::=	$\texttt{l}\cdot\texttt{e}\cdot\texttt{t}\cdot\epsilon$ | $\texttt{b}\cdot\texttt{e}\cdot\epsilon$ | $\texttt{i}\cdot\texttt{n}\cdot\epsilon$
Identifier	id	::=	ltr (ltr | dig)*
Numeral	num	::=	dig dig*
Literal	lit	::=	qum (ltr | dig)*qum
Special	spl	::=	+ | * | ^ | (|) | |
Letter	ltr	::=	a | b | ...
Digit	dig	::=	0 | 1 | ...
Quote	qum	::=	"

3.3.1 Lexical Structure-2

❖ **Rules for translating lexical items into tokens**

$$\frac{s\ \text{str}}{\mathtt{ID}[s]\ \text{tok}} \qquad \frac{}{\mathtt{LET}\ \text{tok}} \qquad \frac{}{\mathtt{ADD}\ \text{tok}} \qquad \frac{}{\mathtt{VB}\ \text{tok}}$$

$$\frac{n\ \text{nat}}{\mathtt{NUM}[n]\ \text{tok}} \qquad \frac{}{\mathtt{BE}\ \text{tok}} \qquad \frac{}{\mathtt{MUL}\ \text{tok}} \qquad \frac{}{\mathtt{LP}\ \text{tok}}$$

$$\frac{s\ \text{str}}{\mathtt{LIT}[s]\ \text{tok}} \qquad \frac{}{\mathtt{IN}\ \text{tok}} \qquad \frac{}{\mathtt{CAT}\ \text{tok}} \qquad \frac{}{\mathtt{RP}\ \text{tok}}$$

(7.1)

3.3.1 Lexical Structure-3

❖ Judgements for lexical analysis

- $s \text{ inp} \longleftrightarrow t \text{ tokstr}$ — Scan input
- $s \text{ itm} \longleftrightarrow t \text{ tok}$ — Scan an item
- $s \text{ kwd} \longleftrightarrow t \text{ tok}$ — Scan a keyword
- $s \text{ id} \longleftrightarrow t \text{ tok}$ — Scan an identifier
- $s \text{ num} \longleftrightarrow t \text{ tok}$ — Scan a number
- $s \text{ spl} \longleftrightarrow t \text{ tok}$ — Scan a symbol
- $s \text{ lit} \longleftrightarrow t \text{ tok}$ — Scan a string literal
- $s \text{ whs}$ — Skip white space

e.g. let a be 34 in a*12

$be \text{ itm} \longleftrightarrow \texttt{BE} \text{ tok}$ $\qquad$ $a \text{ id} \longleftrightarrow \texttt{ID}[a] \text{ tok}$

3.3.1 Lexical Structure-4

❖ **Rules for lexical analysis**

$$\frac{}{\epsilon\ \mathsf{inp} \longleftrightarrow \epsilon\ \mathsf{tokstr}}$$

$$\frac{s = s_1 \hat{\ } s_2 \hat{\ } s_3\ \mathsf{str} \quad s_1\ \mathsf{whs} \quad s_2\ \mathsf{itm} \longleftrightarrow t\ \mathsf{tok} \quad s_3\ \mathsf{inp} \longleftrightarrow ts\ \mathsf{tokstr}}{s\ \mathsf{inp} \longleftrightarrow t \cdot ts\ \mathsf{tokstr}}$$

$$\frac{s\ \mathsf{kwd} \longleftrightarrow t\ \mathsf{tok}}{s\ \mathsf{itm} \longleftrightarrow t\ \mathsf{tok}} \qquad \frac{s\ \mathsf{id} \longleftrightarrow t\ \mathsf{tok}}{s\ \mathsf{itm} \longleftrightarrow t\ \mathsf{tok}} \qquad \frac{s\ \mathsf{num} \longleftrightarrow t\ \mathsf{tok}}{s\ \mathsf{itm} \longleftrightarrow t\ \mathsf{tok}}$$

$$\frac{s\ \mathsf{lit} \longleftrightarrow t\ \mathsf{tok}}{s\ \mathsf{itm} \longleftrightarrow t\ \mathsf{tok}} \qquad \frac{s\ \mathsf{spl} \longleftrightarrow t\ \mathsf{tok}}{s\ \mathsf{itm} \longleftrightarrow t\ \mathsf{tok}} \qquad \frac{s = \mathtt{l} \cdot \mathtt{e} \cdot \mathtt{t} \cdot \varepsilon\ \mathsf{str}}{s\ \mathsf{kwd} \longleftrightarrow \mathtt{LET}\ \mathsf{tok}}$$

$$\frac{s = \mathtt{b} \cdot \mathtt{e} \cdot \varepsilon\ \mathsf{str}}{s\ \mathsf{kwd} \longleftrightarrow \mathtt{BE}\ \mathsf{tok}} \qquad \frac{s = \mathtt{i} \cdot \mathtt{n} \cdot \varepsilon\ \mathsf{str}}{s\ \mathsf{kwd} \longleftrightarrow \mathtt{IN}\ \mathsf{tok}}$$

$$\frac{s = s_1 \hat{\ } s_2\ \mathsf{str} \quad s_1\ \mathsf{ltr} \quad s_2\ \mathsf{lord}}{s\ \mathsf{id} \longleftrightarrow \mathtt{ID}[s]\ \mathsf{tok}} \qquad \textbf{(7.2)}$$

3.3.1 Lexical Structure-5

❖Rules for lexical analysis(cont'd)

$$\frac{s = s_1 \hat{\ } s_2 \ \mathsf{str} \quad s_1 \ \mathsf{dig} \quad s_2 \ \mathsf{dgs} \quad s \ \mathsf{num} \longleftrightarrow n \ \mathsf{nat}}{s \ \mathsf{num} \longleftrightarrow \mathtt{NUM}[n] \ \mathsf{tok}}$$

$$\frac{s = s_1 \hat{\ } s_2 \hat{\ } s_3 \ \mathsf{str} \quad s_1 \ \mathsf{qum} \quad s_2 \ \mathsf{lord} \quad s_3 \ \mathsf{qum}}{s \ \mathsf{lit} \longleftrightarrow \mathtt{LIT}[s_2] \ \mathsf{tok}}$$

$$\frac{s = + \cdot \varepsilon \ \mathsf{str}}{s \ \mathsf{spl} \longleftrightarrow \mathtt{ADD} \ \mathsf{tok}} \qquad \frac{s = * \cdot \varepsilon \ \mathsf{str}}{s \ \mathsf{spl} \longleftrightarrow \mathtt{MUL} \ \mathsf{tok}} \qquad \frac{s = \hat{\ } \cdot \epsilon \ \mathsf{str}}{s \ \mathsf{spl} \longleftrightarrow \mathtt{CAT} \ \mathsf{tok}}$$

$$\frac{s = (\cdot \varepsilon \ \mathsf{str}}{s \ \mathsf{spl} \longleftrightarrow \mathtt{LP} \ \mathsf{tok}} \qquad \frac{s =) \cdot \varepsilon \ \mathsf{str}}{s \ \mathsf{spl} \longleftrightarrow \mathtt{RP} \ \mathsf{tok}} \qquad \frac{s = | \cdot \varepsilon \ \mathsf{str}}{s \ \mathsf{spl} \longleftrightarrow \mathtt{VB} \ \mathsf{tok}}$$

(7.2)

3.3.1 Lexical Structure-6

❖ **e.g.** **a*12**

$$\dfrac{\dfrac{\dfrac{a\ \text{str} \quad a\ \text{ltr}}{a\ \text{id} \longleftrightarrow \texttt{ID}[a]\ \text{tok}}}{a\ \text{itm} \longleftrightarrow \texttt{ID}[a]\ \text{tok}} \qquad \dfrac{\dfrac{\dfrac{*\ \text{str}}{*\ \text{spl} \longleftrightarrow \texttt{MUL}\ \text{tok}}}{*\ \text{itm} \longleftrightarrow \texttt{MUL}\ \text{tok}} \quad \dfrac{\dfrac{\dfrac{\dfrac{1\ \text{dig} \quad 2\ \text{dgs} \quad 12\ \text{num} \longleftrightarrow 12\ \text{nat}}{12\ \text{num} \longleftrightarrow \texttt{NUM}[12]\ \text{tok}}}{12\ \text{itm} \longleftrightarrow \texttt{NUM}[12]\ \text{tok}}}{12\ \text{inp} \longleftrightarrow \texttt{NUM}[12]\ \text{tokstr}}}{*12\ \text{inp} \longleftrightarrow \texttt{MUL} \cdot \texttt{NUM}[12]\ \text{tokstr}}}{a * 12\ \text{inp} \longleftrightarrow \texttt{ID}[a] \cdot \texttt{MUL} \cdot \texttt{NUM}[12]\ \text{tokstr}}$$

3.3.2 Context-Free Grammars-1

- **Components of a Grammar**
  - **tokens, or terminals,**
  - **syntactic classes, or non-terminals,**
  - **rules, or productions,**
    - $A ::= \alpha$,
      A: non-terminal,
      α: a string of terminals and non-terminals
    - $A ::= \alpha_1 \mid \cdots \mid \alpha_n$, (compound production)

3.3.2 Context-Free Grammars-2

❖ Context-free Grammar

- It determines a simultaneous inductive definition of its syntactic classes
- Regard each non-terminal, A, as a judgement, s A, over strings of terminals.
- To each production, $A ::= s_1 \ A_1 \ s_2 \cdots s_n \ A_n \ s_{n+1}$ **(7.3)**

  we associate a rule:

$$\frac{s_1' \ A_1 \quad \cdots \quad s_n' \ A_n}{s_1 \ s_1' \ s_2 \cdots s_n \ s_n' \ s_{n+1} \ A} \qquad \textbf{(7.4)}$$

and it can be rewritten as follows:

$$\frac{s_1' \ A_1 \quad \cdots \quad s_n' \ A_n \quad s = s_1 \hat{\ } s_1' \hat{\ } s_2 \hat{\ } \cdots s_n \hat{\ } s_n' \hat{\ } s_{n+1}}{s \ A} \qquad \textbf{(7.5)}$$

3.3.3 Grammatical Structure-1

- **Grammatical Structure of $\mathcal{L}$\{num, str\}**

$$
\begin{array}{llcl}
\text{Expression} & \mathsf{exp} & ::= & \mathsf{num} \mid \mathsf{lit} \mid \mathsf{id} \mid \mathtt{LP}\ \mathsf{exp}\ \mathtt{RP} \mid \mathsf{exp}\ \mathtt{ADD}\ \mathsf{exp} \mid \\
 & & & \mathsf{exp}\ \mathtt{MUL}\ \mathsf{exp} \mid \mathsf{exp}\ \mathtt{CAT}\ \mathsf{exp} \mid \mathtt{VB}\ \mathsf{exp}\ \mathtt{VB} \mid \\
 & & & \mathtt{LET}\ \mathsf{id}\ \mathtt{BE}\ \mathsf{exp}\ \mathtt{IN}\ \mathsf{exp} \\
\text{Number} & \mathsf{num} & ::= & \mathtt{NUM}[n] \quad (n\ \mathsf{nat}) \\
\text{String} & \mathsf{lit} & ::= & \mathtt{LIT}[s] \quad (s\ \mathsf{str}) \\
\text{Identifier} & \mathsf{id} & ::= & \mathtt{ID}[s] \quad (s\ \mathsf{str})
\end{array}
$$

  - **String**: let a be 3 in a*12
  - **tokstr**: LET ID[*a*] BE NUM[3] IN ID[*a*] MUL NUM[12]

3.3.3 Grammatical Structure-2

- Rules for interpreting a grammar

$$\frac{s\ \text{num}}{s\ \text{exp}} \qquad \frac{s_1\ \text{exp} \quad s_2\ \text{exp}}{s_1\ \texttt{ADD}\ s_2\ \text{exp}} \qquad \frac{s\ \text{exp}}{\texttt{VB}\ s\ \texttt{VB}\ \text{exp}}$$

$$\frac{s\ \text{lit}}{s\ \text{exp}} \qquad \boxed{\frac{s_1\ \text{exp} \quad s_2\ \text{exp}}{s_1\ \texttt{MUL}\ s_2\ \text{exp}}} \qquad \frac{s\ \text{exp}}{\texttt{LP}\ s\ \texttt{RP}\ \text{exp}}$$

$$\frac{s\ \text{id}}{s\ \text{exp}} \qquad \frac{s_1\ \text{exp} \quad s_2\ \text{exp}}{s_1\ \texttt{CAT}\ s_2\ \text{exp}} \qquad \frac{s\ \text{str}}{\texttt{LIT}[s]\ \text{lit}} \qquad \frac{s\ \text{str}}{\texttt{ID}[s]\ \text{id}}$$

$$\frac{s_1\ \text{id} \quad s_2\ \text{exp} \quad s_3\ \text{exp}}{\texttt{LET}\ s_1\ \texttt{BE}\ s_2\ \texttt{IN}\ \text{exp}} \qquad \frac{n\ \text{nat}}{\texttt{NUM}[n]\ \text{num}} \qquad (7.6)$$

$$\frac{s = s_1\ \texttt{MUL}\ s_2\ \text{str} \quad s_1\ \text{exp} \quad s_2\ \text{exp}}{s\ \text{exp}} \qquad (7.7)$$

3.3.4 Ambiguity

- **Principal goal of concrete syntax design**: readability, eliminate ambiguity
- **Example**: 1 + 2*3
  NUM[1] ADD NUM[2] MUL NUM[3]
- *Ambiguity is a purely syntactic property of grammars.*
- **Grammatical structure of $\mathcal{L}$\{num str\}** (eliminate ambiguity)

Factor	fct	::=	num | lit | id | LP prg RP
Term	trm	::=	fct | fct MUL trm | VB fct VB
Expression	exp	::=	trm | trm ADD exp | trm CAT exp
Program	prg	::=	exp | LET id BE trm IN prg

3.3.5 Informal Conventions

❖ **The concrete syntax of $\mathcal{L}$\{num str\}**

Expr

$$e ::= n \mid "s" \mid s \mid e_1 + e_2 \mid e_1 * e_2 \mid e_1 \,\hat{}\, e_2 \mid |e| \mid \texttt{let}\ x\ \texttt{be}\ e_1\ \texttt{in}\ e_2$$

3.4 Abstract Syntax

- expose the hierarchical and binding structure of the language
- eliminate ambiguity

3.4.1 Abstract Syntax Trees

3.4.2 Parsing Into Abstract Syntax Trees

3.4.3 Parsing Into Abstract Binding Trees

3.4.4 Informal Conventions

3.4.1 Abstract Syntax Trees-1

Concrete Syntax	Token String	Abstract Syntax
n	NUM[n]	num[n]
"s"	LIT[s]	str[s]
s	ID[s]	id[s]
e_1+e_2	e_1 ADD e_2	plus(e_1; e_2)
e_1*e_2	e_1 MUL e_2	times(e_1; e_2)
e_1^e_2	e_1 CAT e_2	cat(e_1; e_2)
| e |	VB e VB	len(e)
let s be e_1 in e_2	LET ID[s] BE e_1 IN e_2	let[s](e_1; e_2)
(e)	LP e RP	e

3.4.1 Abstract Syntax Trees-2

❖ **Arities to operators(AST of $\mathcal{L}\{\mathbf{num}, \mathbf{str}\}$):**

$$\begin{aligned} \mathrm{ar}(\mathtt{num}[n]) &= 0 \quad (n \text{ nat}) & \mathrm{ar}(\mathtt{plus}) &= 2 \\ \mathrm{ar}(\mathtt{str}[s]) &= 0 \quad (s \text{ str}) & \mathrm{ar}(\mathtt{times}) &= 2 \\ \mathrm{ar}(\mathtt{id}[s]) &= 0 \quad (s \text{ str}) & \mathrm{ar}(\mathtt{cat}) &= 2 \\ \mathrm{ar}(\mathtt{len}) &= 1 & \mathrm{ar}(\mathtt{let}[\mathtt{s}]) &= 2 \end{aligned}$$

❖ **Inductive definition of the abstract syntax**

$$\frac{n \text{ nat}}{\mathtt{num}[n] \text{ ast}} \qquad \frac{a_1 \text{ ast} \quad a_2 \text{ ast}}{\mathtt{plus}(a_1; a_2) \text{ ast}} \qquad \frac{a \text{ ast}}{\mathtt{len}(a) \text{ ast}}$$

$$\frac{s \text{ str}}{\mathtt{str}[s] \text{ ast}} \qquad \frac{a_1 \text{ ast} \quad a_2 \text{ ast}}{\mathtt{times}(a_1; a_2) \text{ ast}} \qquad \frac{s \text{ str} \quad a_1 \text{ ast} \quad a_2 \text{ ast}}{\mathtt{let}[s](a_1; a_2) \text{ ast}}$$

$$\frac{s \text{ str}}{\mathtt{id}[s] \text{ ast}} \qquad \frac{a_1 \text{ ast} \quad a_2 \text{ ast}}{\mathtt{cat}(a_1; a_2) \text{ ast}} \qquad \textbf{(8.1)}$$

3.4.2 Parsing Into ASTs-1

- **Parsing**: translation from concrete to abstract syntax
- **Parsing judgements** for $\mathcal{L}\{\textbf{num}, \textbf{str}\}$

$s \ \mathsf{prg} \longleftrightarrow a \ \mathsf{ast}$	Parse as a program
$s \ \mathsf{exp} \longleftrightarrow a \ \mathsf{ast}$	Parse as an expression
$s \ \mathsf{trm} \longleftrightarrow a \ \mathsf{ast}$	Parse as a term
$s \ \mathsf{fct} \longleftrightarrow a \ \mathsf{ast}$	Parse as a factor
$s \ \mathsf{num} \longleftrightarrow a \ \mathsf{ast}$	Parse as a number
$s \ \mathsf{lit} \longleftrightarrow a \ \mathsf{ast}$	Parse as a literal
$s \ \mathsf{id} \longleftrightarrow a \ \mathsf{ast}$	Parse as a identifier

3.4.2 Parsing Into ASTs-2

- **Inductive definition**

Factor fct ::= num | lit | id | LP prg RP

$$\frac{n \text{ nat}}{\texttt{NUM}[n] \text{ num} \longleftrightarrow \texttt{num}[n] \text{ ast}} \qquad \frac{s \text{ str}}{\texttt{LIT}[s] \text{ lit} \longleftrightarrow \texttt{str}[s] \text{ ast}}$$

$$\frac{s \text{ str}}{\texttt{ID}[s] \text{ id} \longleftrightarrow \texttt{id}[s] \text{ ast}} \qquad \frac{s \text{ num} \longleftrightarrow a \text{ ast}}{s \text{ fct} \longleftrightarrow a \text{ ast}}$$

$$\frac{s \text{ lit} \longleftrightarrow a \text{ ast}}{s \text{ fct} \longleftrightarrow a \text{ ast}} \qquad \frac{s \text{ id} \longleftrightarrow a \text{ ast}}{s \text{ fct} \longleftrightarrow a \text{ ast}} \qquad \frac{s \text{ prg} \longleftrightarrow a \text{ ast}}{\texttt{LP}\ s\ \texttt{RP} \text{ fct} \longleftrightarrow a \text{ ast}}$$

Term trm ::= fct | fct MUL trm | VB fct VB

$$\frac{s \text{ fct} \longleftrightarrow a \text{ ast}}{s \text{ trm} \longleftrightarrow a \text{ ast}} \qquad \frac{s_1 \text{ fct} \longleftrightarrow a_1 \text{ ast} \quad s_2 \text{ trm} \longleftrightarrow a_2 \text{ ast}}{s_1\ \texttt{MUL} s_2 \text{ trm} \longleftrightarrow \texttt{times}(a_1; a_2) \text{ ast}}$$

$$\frac{s \text{ fct} \longleftrightarrow a \text{ ast}}{\texttt{VB}\ s\ \texttt{VB} \text{ trm} \longleftrightarrow \texttt{len}(a) \text{ ast}} \qquad \textbf{(8.2)}$$

3.4.2 Parsing Into ASTs-3

❖ Inductive definition (cont'd)

Expression	exp	::=	trm | trm ADD exp | trm CAT exp

$$\frac{s \text{ trm} \longleftrightarrow a \text{ ast}}{s \text{ exp} \longleftrightarrow a \text{ ast}} \qquad \frac{s_1 \text{ trm} \longleftrightarrow a_1 \text{ ast} \quad s_2 \text{ exp} \longleftrightarrow a_2 \text{ ast}}{s_1 \texttt{ ADD } s_2 \text{ exp} \longleftrightarrow \texttt{plus}(a_1; a_2) \text{ ast}}$$

$$\frac{s_1 \text{ trm} \longleftrightarrow a_1 \text{ ast} \quad s_2 \text{ exp} \longleftrightarrow a_2 \text{ ast}}{s_1 \texttt{ CAT } s_2 \text{ exp} \longleftrightarrow \texttt{cat}(a_1; a_2) \text{ ast}}$$

Program	prg	::=	exp | LET id BE exp IN prg

$$\frac{s \text{ exp} \longleftrightarrow a \text{ ast}}{s \text{ prg} \longleftrightarrow a \text{ ast}}$$

$$\frac{s_1 \text{ fct} \longleftrightarrow \texttt{id}[s] \text{ ast} \qquad s_2 \text{ exp} \longleftrightarrow a_2 \text{ ast} \qquad s_3 \text{ prg} \longleftrightarrow a_3 \text{ ast}}{\texttt{LET } s_1 \texttt{ BE } s_2 \texttt{ IN } s_3 \text{ prg} \longleftrightarrow \texttt{let}[s](a_2; a_3) \text{ ast}}$$

(8.2)

3.4.2 Parsing Into ASTs-4

- **Concrete Syntax**

  let a be 3 in a * 12

  tokstr

  LET ID[*a*] BE NUM[3] IN ID[*a*] MUL NUM[12]

  prg

  id — exp — prg,exp,trm

  trm — fct,id — trm,fct,num

  fct

  num

- **AST**

  let[*a*] (num[3]; times (id[*a*]; num[12])) ast

  2

  2

3.4.2 Parsing Into ASTs-5

- **Theorem 8.1**

  If s prg $\longleftrightarrow$ a ast, *then* s prg *and* a ast,

  ……(其他分析断言Parsing judgements有类似的性质)

  证明：对规则(8.2)归纳证明.

- **Theorem 8.2**

  If s prg, *then there is a unique* a *such that* s prg $\longleftrightarrow$ a ast.

  ……(其他分析断言Parsing judgements有类似的性质)

  分析断言具有模式 $(\forall, \exists!)$

Why introduce ABT ?

manage the binding and scope of variables in a let expression

3.4.3 Parsing Into ABTs-1

Concrete Syntax	Token String	Abstract Syntax
n	NUM[n]	num[n]
"s"	LIT[s]	str[s]
s	ID[s]	id[s]
e_1+e_2	e_1 ADD e_2	plus(e_1; e_2)
e_1*e_2	e_1 MUL e_2	times(e_1; e_2)
e_1^e_2	e_1 CAT e_2	cat(e_1; e_2)
| e |	VB e VB	len(e)
let s be e_1 in e_2	LET ID[s] BE e_1 IN e_2	let(e_1; x.e_2)
(e)	LP e RP	e

3.4.3 Parsing Into ABTs-2

- **Goal**: manage the binding and scope of variables in a let expression
- **Arities to operators** (ABT of $\mathcal{L}\{\textbf{num}, \textbf{str}\}$):

$$
\begin{array}{rclrcl}
\mathrm{ar}(\mathtt{num}[n]) & = & () & \mathrm{ar}(\mathtt{plus}) & = & (0,0) \\
\mathrm{ar}(\mathtt{str}[s]) & = & () & \mathrm{ar}(\mathtt{times}) & = & (0,0) \\
\mathrm{ar}(\mathtt{cat}) & = & (0,0) & \mathrm{ar}(\mathtt{len}) & = & (0) \\
\mathrm{ar}(\mathtt{let}) & = & (0,1) & & &
\end{array}
$$

  - **Identifiers**: not as operators, but as variables.
- **Revised parsing judgements** for $\mathcal{L}\{\textbf{num}, \textbf{str}\}$

$$
\begin{array}{l}
s \ \mathsf{prg} \longleftrightarrow a \ \mathsf{abt} \\
\ldots
\end{array}
$$

3.4.3 Parsing Into ABTs-3

修订后的分析断言可以用与规则(8.1)类似的一套规则来定义，这些规则采用参数化归纳定义，其中规则的前提和结论都是具有如下形式的假言断言:

$$\mathtt{ID}[s_1]\ \mathrm{id} \longleftrightarrow x_1\ \mathrm{abt}, \cdots, \mathtt{ID}[s_n]\ \mathrm{id} \longleftrightarrow x_n\ \mathrm{abt} \vdash s\ \mathrm{prg} \longleftrightarrow a\ \mathrm{abt}$$

其中所有的x_i是互不相同的变量名。

断言的假设部分说明标识符如何分析为变量，它遵循假言断言的自反性质:

$$\Gamma, \mathtt{ID}[s]\ \mathrm{id} \longleftrightarrow x\ \mathrm{abt} \vdash \mathtt{ID}[s]\ \mathrm{id} \longleftrightarrow x\ \mathrm{abt}$$

3.4.3 Parsing Into ABTs-4

在分析let表达式时，为维护标识符与变量之间的关联关系，会更新假设部分，以记录绑定标识符和其对应的变量之间的关联关系：

去掉?

$$\frac{\Gamma \vdash s_1 \ \mathrm{id} \longleftrightarrow x \ \mathrm{abt} \quad \Gamma \vdash s_2 \ \mathrm{exp} \longleftrightarrow a_2 \ \mathrm{abt} \quad \Gamma, s_1 \ \mathrm{id} \longleftrightarrow x \ \mathrm{abt} \vdash s_3 \ \mathrm{prg} \longleftrightarrow a_3 \ \mathrm{abt}}{\Gamma \vdash \mathtt{LET} \ s_1 \ \mathtt{BE} \ s_2 \ \mathtt{IN} \ s_3 \ \mathrm{prg} \longleftrightarrow \mathtt{let}(a_2;\ x.a_3) \ \mathrm{abt}} \quad \textbf{(8.3a)}$$

问题：如果内层let表达式的绑定标识符与外层let表达式的绑定标识符一样，同一id对应不同的变量，如x1和x2. 由于假言断言的交换性，导致会任意选择x1或x2应用到s3中出现的id.

利用假设不能解决标识符同名问题。

3.4.3 Parsing Into ABTs-5

解决办法：显式地维护符号表来记录标识符与其对应的变量，从而实现同名标识符的**shadowing**策略

分析断言的主要变化是：由假言断言

$$\Gamma \vdash s \text{ prg} \longleftrightarrow a \text{ abt}$$

改成直言断言

$$s \text{ prg} \longleftrightarrow a \text{ abt } [\sigma]$$

σ 是符号表，

符号表是断言的参数，而不是在假设下执行推理的隐式机制

关于符号表的断言：

$\sigma \text{ symtab}$	well-formed symbol table
$\sigma' = \sigma[\mathtt{ID}[s] \mapsto x]$	add new association
$\sigma(\mathtt{ID}[s]) = x$	lookup identifier

3.4.3 Parsing Into ABTs-6

用于分析let表达式的规则：

去掉?

$$\frac{s_1 \text{ id} \longleftrightarrow x\ [\sigma] \quad s_2 \text{ exp} \longleftrightarrow a_2 \text{ abt}\ [\sigma] \quad \sigma' = \sigma[s_1 \mapsto x] \quad s_3 \text{ prg} \longleftrightarrow a_3 \text{ abt}\ [\sigma']}{\texttt{LET}\ s_1\ \texttt{BE}\ s_2\ \texttt{IN}\ s_3 \text{ prg} \longleftrightarrow \texttt{let}(a_2;\ x.a_3) \text{ abt}\ [\sigma]} \quad \textbf{(8.4)}$$

该规则与**(8.3a)**的区别在于：必须显式管理符号表

另外必须增加一条分析标识符的规则，而不是依靠假言断言的自反性：

$$\frac{\sigma(\texttt{ID}[s]) = x}{\texttt{ID}[s] \text{ id} \longleftrightarrow x\ [\sigma]} \quad \textbf{(8.5)}$$

σ maps the identifier ID[s] to the variable x.

3.4.3 Parsing Into ABTs-7

- **Concrete Syntax**

let a be 3 in a * (1 + 2)

tokstr

LET ID[*a*] BE NUM[3] IN ID[*a*] MUL LP NUM[1] ADD NUM[2] RP

- **AST**

let [*a*] (num[3]; times (id[*a*]; plus (num[1]; num[2]))) ast

- **ABT**

let(num[3]; *x*.times (id[*a*]; plus (num[1]; num[2]))) abt

3.4.3 Syntactic Conventions

- **The abstract syntax of $\mathcal{L}$\{num str\}**

$$
\begin{array}{llll}
\text{Type} & \tau & ::= & \texttt{num} \mid \texttt{str} \\
\text{Expr} & e & ::= & x \mid \texttt{num}[n] \mid \texttt{str}[s] \mid \texttt{plus}(e_1; e_2) \mid \\
 & & & \texttt{times}(e_1; e_2) \mid \texttt{cat}(e_1; e_2) \mid \texttt{len}(e) \mid \\
 & & & \texttt{let}(e_1; x.e_2)
\end{array}
$$

τ type: **τ is a well-formed type** Ω_{type}

e exp : **e is a well-formed expression** Ω_{exp}

3.5 Static Semantics

- Consist of a collection of rules for imposing constraints on the formation of programs, called a type system.
- the type of a phrase predicts the form of its value
- well-typed: A phrase is constructed consistently with these predictions.
  ill-typed

  $x : \mathrm{nat}$ $\quad x + 3$ well-typed $\quad x +'' 123''$ ill-typed

3.5.1 Static Semantics of L{num,str}

3.5.2 Structural Properties

3.5.1 Static Semantics of $\mathcal{L}\{\text{num}, \text{str}\}$ -1

❖ **Judgement**

e : τ,   where e exp and τ type.

Parametric hypothetical judgements $\mathcal{X} | \Gamma \vdash e : \tau$

$\mathcal{X}$: a finite set of variables, 通常被省去

Γ : a typing context ($x : \tau$, $x \in \mathcal{X}$)

❖ **Typing Rules**

$$\overline{\Gamma, x : \tau \vdash x : \tau} \quad \textbf{(9.1a)}$$

$$\overline{\Gamma \vdash \mathtt{str}[s] : \mathtt{str}} \quad \textbf{(9.1b)}$$

$$\overline{\Gamma \vdash \mathtt{num}[n] : \mathtt{num}} \quad \textbf{(9.1c)}$$

3.5.1 Static Semantics of $\mathcal{L}\{\text{num}, \text{str}\}$ -2

- **Typing Rules**

$$\frac{\Gamma \vdash e_1 : \texttt{num} \quad \Gamma \vdash e_2 : \texttt{num}}{\Gamma \vdash \texttt{plus}(e_1; e_2) : \texttt{num}} \quad (9.1d)$$

$$\frac{\Gamma \vdash e_1 : \texttt{num} \quad \Gamma \vdash e_2 : \texttt{num}}{\Gamma \vdash \texttt{times}(e_1; e_2) : \texttt{num}} \quad (9.1e)$$

$$\frac{\Gamma \vdash e_1 : \texttt{str} \quad \Gamma \vdash e_2 : \texttt{str}}{\Gamma \vdash \texttt{cat}(e_1, e_2) : \texttt{str}} \quad (9.1f)$$

$$\frac{\Gamma \vdash e : \texttt{str}}{\Gamma \vdash \texttt{len}(e) : \texttt{num}} \quad (9.1g)$$

x不在Γ中，若e_1中有x，则通过α-等价将e_1中的x换名

$$\frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let}(e_1, x.e_2) : \tau_2} \quad (9.1h)$$

3.5.1 Static Semantics of $\mathcal{L}\{\mathbf{num}, \mathbf{str}\}$ -3

- **Lemma 9.1 (Unicity for Typing).**

  For every typing context Γ and expression e, there exists at most one τ such that $\Gamma \vdash e : \tau$.

- **Lemma 9.2 (Inversion for Typing).**

  Suppose that $\Gamma \vdash e : \tau$.

  1. if e = x, then $\Gamma \vdash x : \tau$.
  2. if e = num[n], then τ = num.
  3. if e = str[s], then τ = str.
  4. if e = plus(e_1; e_2) or e = times(e_1; e_2), then τ = num, $\Gamma \vdash e_1 : \mathtt{num}$ and $\Gamma \vdash e_2 : \mathtt{num}$.

  ……

3.5.1 Static Semantics of $\mathcal{L}\{\text{num}, \text{str}\}$ -4

- **e.g.Lemma**

  let (str[1], x.cat(str[123], x)) :str

$$\dfrac{\dfrac{}{\Gamma \vdash \texttt{str}[1] : \texttt{str}} \quad \dfrac{\dfrac{}{\Gamma \vdash \texttt{str}[123] : \texttt{str}} \quad \dfrac{}{\Gamma, x : \texttt{str} \vdash x : \texttt{str}}}{\Gamma, a : \texttt{str} \vdash \texttt{cat}(\texttt{str}[123]; a) : \texttt{str}}}{\Gamma \vdash \texttt{let}(\texttt{str}[1]; x.\texttt{cat}(\texttt{str}[123]; x)) : \texttt{str}}$$

3.5.2 Structural Properties-1

静态语义具有假言断言和参数化断言的结构性质。

- **Lemma 9.3 (proliferation)**

  If $\Gamma \vdash e' : \tau'$, then for any $x \# \Gamma$ and any $\tau\ \text{type}$,
  $\Gamma, x : \tau \vdash e' : \tau'$.

- **Lemma 9.4 (substitution)**

  If $\Gamma, x : \tau \vdash e' : \tau'$ and $\Gamma \vdash e : \tau$,
  then $\Gamma \vdash [e/x]e' : \tau'$.

3.5.2 Structural Properties-2

❖ **Lemma 9.3 (proliferation)**

If $\Gamma \vdash e' : \tau'$, then for any $x \# \Gamma$ and any $\tau \text{ type}$, $\Gamma, x : \tau \vdash e' : \tau'$.

Proof

By induction on the derivation of $\Gamma, x : \tau \vdash e' : \tau'$

Suppose $e' = \texttt{let}(e_1, z.e_2)$, where $z \# \Gamma$ and $z \# x$

By induction we have

(A) $\Gamma, x : \tau \vdash e_1 : \tau_1$,

(B) $\Gamma, x : \tau, z : \tau_1 \vdash e_2 : \tau'$,

from which the result follows by Rule (9.1h).

Other cases….

3.5.2 Structural Properties-3

- **Lemma 9.4 (substitution)**

  If $\Gamma, x : \tau \vdash e' : \tau'$ and $\Gamma \vdash e : \tau$,
  then $\Gamma \vdash [e/x]e' : \tau'$.

Proof

By induction on $\Gamma, x : \tau \vdash e' : \tau'$

Suppose $e' = \texttt{let}(e_1, z.e_2)$, where $z \# \Gamma$, $z \# x$ and $z \# e$

By induction we have

(A) $\Gamma \vdash [e/x]e_1 : \tau_1$,

(B) $\Gamma, z : \tau_1 \vdash [e/x]e_2 : \tau'$,

Since **z#e**, we have $[e/x]\texttt{let}(e_1, z.e_2) = \texttt{let}([e/x]e_1, z.[e/x]e_2)$

It follows by Rule (9.1h) that $\Gamma \vdash [e/x]\texttt{let}(e_1, z.e_2) : \tau$

Other cases....

3.5.2 Structural Properties-4

- **Lemma 9.5** (descomposition)

  If $\Gamma \vdash [e/x]e' : \tau'$ then there exists a unique type τ such that $\Gamma \vdash e : \tau$, $\Gamma, x : \tau \vdash e' : \tau'$.

Proof

Directly from the unicity of types (Lemma 9.1)

since τ is the unique type for e in the composite expression [e/x]e'.

3.6 Dynamic Semantics

- Specify how programs are to be executed.
- Methods for specifying dynamic semantics
  - Structural semantics: small-step OS.
    - Contextual semantics
  - Evaluation semantics: big-step OS
    - Environment semantics, cost semantics

3.6.1 Transition Systems [PFPL, 4]

3.6.2 Structural semantics [PFPL, 10]

3.6.3 Contextual semantics [PFPL, 10]

3.6.4 Evaluation semantics [PFPL, 12]

3.6.5 Environment semantics and Cost semantics

3.6.1 Transition Systems-1

❖ Transition system

S: a set of states that are related by a transition judgement,

An transition system is specified by the judgements

$$s \text{ state}, \quad s \text{ final}, \quad s \text{ initial}, \quad s \mapsto s'$$

- A state s is stuck, if there is no $s' \in S$ such that $s \mapsto s'$.

All final states are stuck, but not all stuck states need be final!

3.6.1 Transition Systems-2

- **Transition Sequence:** a sequence of states $s_0, \cdots, s_n$ such that $s_0 \text{ initial}$ and $s_i \mapsto s_{i+1}, 0 \le i < n$

A transition sequence is

- maximal: iff $s_n \not\mapsto$
- complete: iff $s_n \not\mapsto$ and $s_n \text{ final}$.
- deterministic: iff for every state s there exists at most one state s' such that $s \mapsto s'$, otherwise it is non-deterministic.

3.6.1 Transition Systems-3

❖ Iterated Transition

$s \mapsto^* s'$: is the reflexive, transitive closure of $s \mapsto s'$

Rules

$$\frac{}{s \mapsto^* s} \quad (4.1a)$$

$$\frac{s \mapsto s' \qquad s' \mapsto^* s''}{s \mapsto^* s''} \quad (4.1b)$$

Principle of rule induction

To show that P(s,s') holds whenever $s \mapsto^* s'$,

it is enough to show:

1) P(s,s)

2) if $s \mapsto s'$ and P(s',s''), then P(s, s'') .

3.6.1 Transition Systems-4

❖ Iterated Transition

$s \mapsto^n s'$: n-times iterated transition judgement, $n \geq 0$

$$\frac{}{s \mapsto^0 s} \quad (4.2a)$$

$$\frac{s \mapsto s' \quad s' \mapsto^n s''}{s \mapsto^{n+1} s''} \quad (4.2b)$$

Theorem 4.1

For all states s and s', $s \mapsto^* s'$ iff $s \mapsto^k s'$ for some $k \geq 0$.

$\downarrow s$: indicate there exists some s' final such that $s \mapsto^* s'$

3.6.2 Structural semantics-1

- **Structural semantics of $\mathcal{L}\{\text{num, str}\}$**
  - a transition system whose states are closed expressions.
  - Every closed expression is an initial state
  - The final states are the closed values, as defined by

$$\frac{}{\texttt{num}[n]\ \text{val}} \qquad \frac{}{\texttt{str}[s]\ \text{val}} \qquad (10.1)$$

- **Transition judgement** $e \mapsto e'$

Rule

$$\frac{n_1 + n_2 = n\ \text{nat}}{\texttt{plus}(\texttt{num}[n_1];\texttt{num}[n_2]) \mapsto \texttt{num}[n]} \qquad (10.2a)$$

$$\frac{e_1 \mapsto e_1'}{\texttt{plus}(e_1;e_2) \mapsto \texttt{plus}(e_1', e_2)} \qquad (10.2b)$$

3.6.2 Structural semantics-2

❖ **Rule**

$$\frac{e_1 \ \mathsf{val} \quad e_2 \mapsto e_2'}{\texttt{plus}(e_1; e_2) \mapsto \texttt{plus}(e_1; e_2')} \quad \textbf{(10.2c)}$$

$$\frac{s_1 \hat{\ } s_2 = s \ \mathsf{str}}{\texttt{cat}(\texttt{str}[s_1]; \texttt{str}[s_2]) \mapsto \texttt{str}[s]} \quad \textbf{(10.2d)}$$

$$\frac{e_1 \mapsto e_1'}{\texttt{cat}(e_1; e_2) \mapsto \texttt{cat}(e_1'; e_2)} \quad \textbf{(10.2e)}$$

$$\frac{e_1 \ \mathsf{val} \quad e_2 \mapsto e_2'}{\texttt{cat}(e_1; e_2) \mapsto \texttt{cat}(e_1, e_2')} \quad \textbf{(10.2f)}$$

$$\frac{e_1 \ \mathsf{val}}{\texttt{let}(e_1; x.e_2) \mapsto [e_1/x]e_2} \quad \textbf{(10.2g)}$$

$$\frac{e_1 \mapsto e_1'}{\texttt{let}(e_1; x.e_2) \mapsto \texttt{let}(e_1'; x.e_2)} \quad \textbf{(10.2h)}$$

instruction transitions (10.2a, d, g) Primitive steps of evaluation

search transitions Determine the evaluation order

3.6.2 Structural semantics-3

- **Derivation sequence:**

  width: the number of steps in the sequence

$$
\begin{aligned}
&\texttt{let}(\texttt{plus}(\texttt{num}[1];\texttt{num}[2]), x.\texttt{plus}(\texttt{plus}(x;\texttt{num}[3]);\texttt{num}[4]))\\
\mapsto\quad & \texttt{let}(\texttt{num}[3]; x.\texttt{plus}(\texttt{plus}(x;\texttt{num}[3]);\texttt{num}[4]))\\
\mapsto\quad & \texttt{plus}(\texttt{plus}(\texttt{num}[3];\texttt{num}[3]);\texttt{num}[4])\\
\mapsto\quad & \texttt{plus}(\texttt{num}[6];\texttt{num}[4])\\
\mapsto\quad & \texttt{num}[10]
\end{aligned}
$$

  depth: the derivation tree for each step

  e.g. the third transition is

$$
\dfrac{\overline{\texttt{plus}(\texttt{num}[3];\texttt{num}[3]) \mapsto \texttt{num}[6]}\,(10.2a)}{\texttt{plus}(\texttt{plus}(\texttt{num}[3];\texttt{num}[3]);\texttt{num}[4]) \mapsto \texttt{plus}(\texttt{num}[6];\texttt{num}[4])}\,(10.2b)
$$

3.6.2 Structural semantics-4

- **Principle of rule induction**

  To show P(e, e′) holds whenever $e \mapsto e'$, it is sufficient to show that P is closed under the rules defining the transition judgement.

- **Lemma 10.1 (evaluation of exp.s is deterministic)**

  If $e \mapsto e'$ and $e \mapsto e''$, then e′ is e″ .

  Proof. By simultaneous induction on the two premises using Rules (10.2).

  Only one rule applies for a given e.

3.6.3 Contextual semantics-1

- **Contextual semantics:** variant of structural semantics
  - isolate instruction steps as instruction transition judgements $e_1 \leadsto e_2$

$$\frac{m + n = p \text{ nat}}{\texttt{plus}(\texttt{num}[m];\texttt{num}[n]) \leadsto \texttt{num}[p]} \quad \textbf{(10.3a)}$$

$$\frac{s\hat{\ }t = u \text{ str}}{\texttt{cat}(\texttt{str}[s];\texttt{str}[t]) \leadsto \texttt{str}[u]} \quad \textbf{(10.3b)}$$

$$\frac{e_1 \text{ val}}{\texttt{let}(e_1;x.e_2) \leadsto [e_1/x]e_2} \quad \textbf{(10.3c)}$$

redax: LHS of each intruction; contractum: RHS

  - formalize the process of locating the next instruction using an evaluation context.

3.6.3 Contextual semantics-2

Evaluation Context Judgement

$\mathcal{E}$ ectxt : determines the location of the next instruction to execute in a larger expression.

$\circ$:"hole", the position of the next instruction step

e.g. $\circ$ can be one of the following forms

plus(num[n1]; num[n2])

cat(str[s1]; str[s2])

let(e1; x.e2), where e1 val

……

3.6.3 Contextual semantics-3

Evaluation Context Judgement

Rules

$$\overline{\circ \ \text{ectxt}}$$

It means the next instruction may occur “here”, i.e.

$$\frac{\mathcal{E}_1 \ \text{ectxt}}{\texttt{plus}(\mathcal{E}_1; e_2) \ \text{ectxt}} \qquad \frac{e_1 \ \text{val} \quad \mathcal{E}_2 \ \text{ectxt}}{\texttt{plus}(e_1; \mathcal{E}_2) \ \text{ectxt}}$$

$$\frac{\mathcal{E}_1 \ \text{ectxt}}{\texttt{cat}(\mathcal{E}_1; e_2) \ \text{ectxt}} \qquad \frac{e_1 \ \text{val} \quad \mathcal{E}_2 \ \text{ectxt}}{\texttt{cat}(\mathcal{E}_1; e_2) \ \text{ectxt}}$$

$$\frac{\mathcal{E}_1 \ \text{ectxt}}{\texttt{let}(\mathcal{E}_1; x.e_2) \ \text{ectxt}} \qquad (10.4)$$

The remaining rules correspond one-for-one to the search rules of the structural semantics.

3.6.3 Contextual semantics-4

❖ Evaluation Context

- a template instantiated by replacing the hole with an instruction to be executed.

Judgement $e' = \mathcal{E}\{e\}$

e' is the result of filling the hole in $\mathcal{E}$ with e.

(10.5)

$$\frac{}{e = \circ\{e\}}$$

$$\frac{e_1 = \mathcal{E}_1\{e\}}{\texttt{cat}(e_1; e_2) = \texttt{cat}(\mathcal{E}_1; e_2)\{e\}}$$

$$\frac{e_1 = \mathcal{E}_1\{e\}}{\texttt{plus}(e_1; e_2) = \texttt{plus}(\mathcal{E}_1; e_2)\{e\}}$$

$$\frac{e_1 \ \mathsf{val} \quad e_1 = \mathcal{E}_2\{e\}}{\texttt{cat}(e_1; e_2) = \texttt{cat}(e_1; \mathcal{E}_2)\{e\}}$$

$$\frac{e_1 \ \mathsf{val} \quad e_1 = \mathcal{E}_2\{e\}}{\texttt{plus}(e_1; e_2) = \texttt{plus}(e_1; \mathcal{E}_2)\{e\}}$$

$$\frac{e_1 = \mathcal{E}_1\{e\}}{\texttt{let}(e_1; x.e_2) = \texttt{let}(\mathcal{E}_1; x.e_2)\{e\}}$$

3.6.3 Contextual semantics-4

- **Dynamic semantics for $\mathcal{L}$\{num str\}**

$$\frac{e = \mathcal{E}\{e_0\} \quad e_0 \rightsquigarrow e_0' \quad e' = \mathcal{E}\{e_0'\}}{e \mapsto e'} \quad (10.6)$$

A transition from e to e′ consists of

1. decomposing e into an evaluation context and an instruction,
2. execution of that instruction, and
3. replacing the instruction by the result of its execution in the same spot within e to obtain e′.

3.6.3 Contextual semantics-5

The structural and contextual semantics define the same transition relation.

Theorem 10.2. $e \mapsto_s e'$ if, and only if, $e \mapsto_c e'$.

$$\frac{e = \mathcal{E}\{e_0\} \quad e_0 \leadsto e_0' \quad e' = \mathcal{E}\{e_0'\}}{e \mapsto e'} \quad \textbf{(10.6)}$$

$$\frac{e_0 \leadsto e_0'}{\mathcal{E}\{e_0\} \mapsto \mathcal{E}\{e_0'\}} \quad \textbf{(10.7)}$$

3.6.4 Evaluation semantics-1

- **Evaluation semantics(ES)**: a relation between a phrase and its value.
- **Evaluation judgement** $e \Downarrow v$

  specify the value v of a closed expression e

$$\frac{}{\texttt{num}[n] \Downarrow \texttt{num}[n]} \qquad \frac{e_1 \Downarrow \texttt{num}[n_1] \quad e_2 \Downarrow \texttt{num}[n_2] \quad n_1 + n_2 = n \text{ nat}}{\texttt{plus}(e_1; e_2) \Downarrow \texttt{num}[n]}$$

$$\frac{}{\texttt{str}[s] \Downarrow \texttt{str}[s]} \qquad \frac{e_1 \Downarrow \texttt{str}[s_1] \quad e_2 \Downarrow \texttt{str}[s_2] \quad s_1 \hat{\ } s_2 = s \text{ str}}{\texttt{cat}(e_1; e_2) \Downarrow \texttt{str}[s]}$$

$$\frac{e_1 \Downarrow v_1 \quad [v_1/x]e_2 \Downarrow v_2}{\texttt{let}(e_1; x.e_2) \Downarrow v_2} \qquad \textbf{(12.1)}$$

3.6.4 Evaluation semantics-2

❖ **Principle of rule induction**

To show P(e, v) holds, it is enough to show that P is closed under the rules defining the evaluation judgement.

1. Show that P(num[n], num[n]) .
2. Show that P(str[s], str[s]) .
3. Show that P(plus(e_1; e_2), num[n]), assuming $n_1 + n_2 = n$ nat, P(e_1, num[n_1]) and P(e_2, num[n_2]).
4. Show that P(cat(e_1; e_2), str[s]), assuming s_1^s_2 = s str, P(e_1, str[s_1]) and P(e_2, str[s_2]).
5. Show that P(let(e_1; x.e_2), v_2) , assuming P(e_1, v_1) and P([v_1/x]e_2, v_2).

Lemma 12.1 If $e \Downarrow v$, then v val.

3.6.4 Evaluation semantics-3

Theorem 12.2 For all closed expressions e and values v, $e \mapsto^* v$ iff $e \Downarrow v$.

Lemma 12.3 If $e \Downarrow v$, then $e \mapsto^* v$.

Proof. By induction on the definition of the evaluation judgement.

Suppose: $\texttt{plus}(e_1; e_2) \Downarrow \texttt{num}[n]$ by the rule (12.1).

By induction, $e_1 \mapsto^* \texttt{num}[n_1]$ and $e_2 \mapsto^* \texttt{num}[n_2]$

$$
\begin{aligned}
\texttt{plus}(e_1; e_2) &\mapsto^* \texttt{plus}(\texttt{num}[n_1]; e_2) \\
&\mapsto^* \texttt{plus}(\texttt{num}[n_1]; \texttt{num}[n_2]) \\
&\mapsto \texttt{num}[n_1 + n_2]
\end{aligned}
$$

3.6.4 Evaluation semantics-4

Lemma 12.4 If $e \mapsto e'$ and $e' \Downarrow v$, then $e \Downarrow v$

Proof. By induction on the definition of the transition judgement.

Suppose: $\texttt{plus}(e_1; e_2) \mapsto \texttt{plus}(e_1'; e_2)$ where $e_1 \mapsto e_1'$ by the rule (10.2).

Suppose further: $\texttt{plus}(e_1'; e_2) \Downarrow \texttt{num}[n]$, so that $e_1' \Downarrow \texttt{num}[n_1]$ and $e_2 \Downarrow \texttt{num}[n_2]$ and $n_1 + n_2 = n \text{ nat}$

By induction, $e_1 \Downarrow \texttt{num}[n_1]$ and hence

$$\texttt{plus}(e_1; e_2) \Downarrow \texttt{num}[n]$$

3.6.5 Environment S. and Cost S.-1

- **Environment semantics**
  - Substitution: replace let-bound variables by their bindings during evaluation.
    Maintain the invariant that only closed expressions are ever considered

In practice, we do not perform substitution

  - record the bindings of variables in some sort of data structure
  - **environment** $\mathcal{E}$: set of hypotheses of the form $x \Downarrow v$, x is a variable, v is a value

3.6.5 Environment S. and Cost S.-2

❖ Environment semantics

Judgement $\mathcal{E} \vdash e \Downarrow v$

$\mathcal{E}$:an env. governing some finite set of variables

Rules

$$\frac{}{\mathcal{E}, x \Downarrow v \vdash x \Downarrow v}$$

$$\frac{\mathcal{E} \vdash e_1 \Downarrow \texttt{num}[n_1] \qquad \mathcal{E} \vdash e_2 \Downarrow \texttt{num}[n_2]}{\mathcal{E} \vdash \texttt{plus}(e_1; e_2) \Downarrow \texttt{num}[n_1 + n_2]}$$

$$\frac{\mathcal{E} \vdash e_1 \Downarrow \texttt{str}[s_1] \qquad \mathcal{E} \vdash e_2 \Downarrow \texttt{num}[s_2]}{\mathcal{E} \vdash \texttt{cat}(e_1; e_2) \Downarrow \texttt{str}[s_1 \hat{\ } s_2]} \qquad \textbf{(12.2)}$$

$$\frac{\mathcal{E} \vdash e_1 \Downarrow v_1 \qquad \mathcal{E}, x \Downarrow v_1 \vdash e_2 \Downarrow v_2}{\mathcal{E} \vdash \texttt{let}(e_1; x.e_2) \Downarrow v_2}$$

3.6.5 Environment S. and Cost S.-3

❖ Cost semantics

- SS provides time complexity for program,but ES does not provide such a direct notion of complexity

Judgement : $e \Downarrow^n v$, e evaluates to v in n steps

Rules

$$\overline{\texttt{num[n]} \Downarrow^0 \texttt{num[n]}} \qquad \overline{\texttt{str[s]} \Downarrow^0 \texttt{str[s]}}$$

$$\frac{e_1 \Downarrow^{k_1} \texttt{num}[n_1] \quad e_2 \Downarrow^{k_2} \texttt{num}[n_2]}{\texttt{plus}(e_1; e_2) \Downarrow^{k_1+k_2+1} \texttt{num}[n_1 + n_2]}$$

$$\frac{e_1 \Downarrow^{k_1} \texttt{str}[s_1] \quad e_2 \Downarrow^{k_2} \texttt{str}[s_2]}{\texttt{cat}(e_1; e_2) \Downarrow^{k_1+k_2+1} \texttt{str}[s_1 + s_2]}$$

(12.3)

$$\frac{e_1 \Downarrow^{k_1} \quad [v_1/x]e_2 \Downarrow^{k_2} v_2}{\texttt{let}(e_1; x.e_2) \Downarrow^{k_1+k_2+1} v_2}$$

一个例子

- 程序

  let a be 3+3 in let b be 4 in a+b

- 记号串

  LET ID[a] BE NUM[3] ADD NUM[3] IN LET ID[b] BE NUM[4] IN ID[a] ADD ID[b]

- 分析生成抽象绑定树(ABT)

Factor	fct	::=	num | lit | id | LP prg RP
Term	trm	::=	fct | fct MUL trm | VB fct VB
Expression	exp	::=	trm | trm ADD exp | trm CAT exp
Program	prg	::=	exp | LET id BE trm IN prg

一个例子-分析生成ABT

LET ID[a] BE NUM[3] ADD NUM[3] IN LET ID[b] BE NUM[4] IN ID[a] ADD ID[b]

Factor	fct	::=	num | lit | id | LP prg RP
Term	trm	::=	fct | fct MUL trm | VB fct VB
Expression	exp	::=	trm | trm ADD exp | trm CAT exp
Program	prg	::=	exp | LET id BE trm IN prg

$$\Gamma, \mathtt{ID}[s] \text{ id} \longleftrightarrow x \text{ abt} \vdash \mathtt{ID}[s] \text{ id} \longleftrightarrow x \text{ abt}$$

$$\frac{\Gamma \vdash s_2 \text{ exp} \longleftrightarrow a_2 \text{ abt} \qquad \Gamma, s_1 \text{ id} \longleftrightarrow x \text{ abt} \vdash s_3 \text{ prg} \longleftrightarrow a_3 \text{ abt}}{\Gamma \vdash \mathtt{LET}\ s_1\ \mathtt{BE}\ s_2\ \mathtt{IN}\ s_3 \text{ prg} \longleftrightarrow \mathtt{let}(a_2;\ x.a_3) \text{ abt}}$$

$$\overline{\mathcal{A}, x\ abt^0 \vdash x\ abt^0}$$

$$\frac{\text{ar}\ (o) = (n_1, \cdots, n_k) \quad \mathcal{A} \vdash a_1 \text{ abt}^{n_1} \quad \cdots \quad \mathcal{A} \vdash a_k \text{ abt}^{n_k}}{\mathcal{A} \vdash o(a_1, \cdots, a_k) \text{ abt}^0}$$

$$\frac{x \# \mathcal{A} \qquad \mathcal{A}, x \text{ abt}^0 \vdash a \text{ abt}^n}{\mathcal{A} \vdash x.a \text{ abt}^{n+1}}$$

自底向上构造（Γ 被省去）

- 对**ID[a]**和**ID[b]**分析，得到对应的抽象绑定树

$$\frac{a \text{ str}}{\mathtt{ID}[a] \text{ id} \longleftrightarrow x_1 \text{ abt}^0 \vdash \mathtt{ID}[a] \text{ id} \longleftrightarrow x_1 \text{ abt}^0}$$

$$\frac{b \text{ str}}{\mathtt{ID}[b] \text{ id} \longleftrightarrow x_2 \text{ abt}^0 \vdash \mathtt{ID}[b] \text{ id} \longleftrightarrow x_2 \text{ abt}^0}$$

一个例子-分析生成ABT

LET ID[a] BE NUM[3] ADD NUM[3] IN LET ID[b] BE NUM[4] IN ID[a] ADD ID[b]

Factor	fct	::=	num | lit | id | LP prg RP
Term	trm	::=	fct | fct MUL trm | VB fct VB
Expression	exp	::=	trm | trm ADD exp | trm CAT exp
Program	prg	::=	exp | LET id BE trm IN prg

$$\Gamma, \mathtt{ID}[s] \text{ id} \longleftrightarrow x \text{ abt} \vdash \mathtt{ID}[s] \text{ id} \longleftrightarrow x \text{ abt}$$

$$\frac{\Gamma \vdash s_2 \text{ exp} \longleftrightarrow a_2 \text{ abt} \qquad \Gamma, s_1 \text{ id} \longleftrightarrow x \text{ abt} \vdash s_3 \text{ prg} \longleftrightarrow a_3 \text{ abt}}{\Gamma \vdash \mathtt{LET}\ s_1\ \mathtt{BE}\ s_2\ \mathtt{IN}\ s_3 \text{ prg} \longleftrightarrow \mathtt{let}(a_2;\ x.a_3) \text{ abt}}$$

$$\overline{\mathcal{A}, x\ abt^0 \vdash x\ abt^0}$$

$$\frac{\text{ar}\ (o) = (n_1, \cdots, n_k) \qquad \mathcal{A} \vdash a_1 \text{ abt}^{n_1} \quad \cdots \quad \mathcal{A} \vdash a_k \text{ abt}^{n_k}}{\mathcal{A} \vdash o(a_1, \cdots, a_k) \text{ abt}^0}$$

$$\frac{x \# \mathcal{A} \qquad \mathcal{A}, x \text{ abt}^0 \vdash a \text{ abt}^n}{\mathcal{A} \vdash x.a \text{ abt}^{n+1}}$$

- 对**NUM[3]**和**NUM[4]**分析，得到对应的抽象绑定树

$$\cfrac{\cfrac{\cfrac{\cfrac{3 \text{ nat}}{\vdash \mathtt{NUM}[3] \text{ num} \longleftrightarrow \mathtt{num}[3] \text{ abt}^0}}{\vdash \mathtt{NUM}[3] \text{ fct} \longleftrightarrow \mathtt{num}[3] \text{ abt}^0}}{\vdash \mathtt{NUM}[3] \text{ trm} \longleftrightarrow \mathtt{num}[3] \text{ abt}^0}}{\vdash \mathtt{NUM}[3] \text{ exp} \longleftrightarrow \mathtt{num}[3] \text{ abt}^0}$$

$$\cfrac{\cfrac{\cfrac{\cfrac{4 \text{ nat}}{\vdash \mathtt{NUM}[4] \text{ num} \longleftrightarrow \mathtt{num}[4] \text{ abt}^0}}{\vdash \mathtt{NUM}[4] \text{ fct} \longleftrightarrow \mathtt{num}[4] \text{ abt}^0}}{\vdash \mathtt{NUM}[4] \text{ trm} \longleftrightarrow \mathtt{num}[4] \text{ abt}^0}}{\vdash \mathtt{NUM}[4] \text{ exp} \longleftrightarrow \mathtt{num}[4] \text{ abt}^0}$$

一个例子-分析生成ABT

LET ID[a] BE NUM[3] ADD NUM[3] IN LET ID[b] BE NUM[4] IN ID[a] ADD ID[b]

Factor	fct	::=	num | lit | id | LP prg RP
Term	trm	::=	fct | fct MUL trm | VB fct VB
Expression	exp	::=	trm | trm ADD exp | trm CAT exp
Program	prg	::=	exp | LET id BE trm IN prg

$$\Gamma, \mathtt{ID}[s]\ \text{id} \longleftrightarrow x\ \text{abt} \vdash \mathtt{ID}[s]\ \text{id} \longleftrightarrow x\ \text{abt}$$

$$\frac{\Gamma \vdash s_2\ \text{exp} \longleftrightarrow a_2\ \text{abt} \qquad \Gamma, s_1\ \text{id} \longleftrightarrow x\ \text{abt} \vdash s_3\ \text{prg} \longleftrightarrow a_3\ \text{abt}}{\Gamma \vdash \mathtt{LET}\ s_1\ \mathtt{BE}\ s_2\ \mathtt{IN}\ s_3\ \text{prg} \longleftrightarrow \mathtt{let}(a_2;\ x.a_3)\ \text{abt}}$$

$$\frac{}{\mathcal{A}, x\ abt^0 \vdash x\ abt^0}$$

$$\frac{\text{ar}\ (o) = (n_1, \cdots, n_k) \qquad \mathcal{A} \vdash a_1\ \text{abt}^{n_1} \quad \cdots \quad \mathcal{A} \vdash a_k\ \text{abt}^{n_k}}{\mathcal{A} \vdash o(a_1, \cdots, a_k)\ \text{abt}^0}$$

$$\frac{x \# \mathcal{A} \qquad \mathcal{A}, x\ \text{abt}^0 \vdash a\ \text{abt}^n}{\mathcal{A} \vdash x.a\ \text{abt}^{n+1}}$$

- 对**ID[a] ADD ID[b]**分析，得到对应的抽象绑定树

$$\frac{ar(\mathtt{plus}) = (0, 0) \qquad x_1\ \text{abt}^0 \vdash x_1\ \text{abt}^0 \qquad x_2\ \text{abt}^0 \vdash x_2\ \text{abt}^0}{x_1\ \text{abt}^0, x_2\ \text{abt}^0 \vdash \mathtt{plus}(x_1; x_2)\ \text{abt}^0}$$

$$\frac{\dfrac{\dfrac{\dfrac{a\ \text{str}}{\mathtt{ID}[a]\ \text{id} \longleftrightarrow x_1\ \text{abt}^0}}{\mathtt{ID}[a]\ \text{fct} \longleftrightarrow x_1\ \text{abt}^0}}{\mathtt{ID}[a]\ \text{trm} \longleftrightarrow x_1\ \text{abt}^0} \qquad \dfrac{\dfrac{\dfrac{\dfrac{b\ \text{str}}{\mathtt{ID}[b]\ \text{id} \longleftrightarrow x_2\ \text{abt}^0}}{\mathtt{ID}[b]\ \text{fct} \longleftrightarrow x_2\ \text{abt}^0}}{\mathtt{ID}[b]\ \text{trm} \longleftrightarrow x_2\ \text{abt}^0}}{\mathtt{ID}[b]\ \text{exp} \longleftrightarrow x_2\ \text{abt}^0}}{\mathtt{ID}[a]\ \mathtt{ADD}\ \mathtt{ID}[b]\ \text{exp} \longleftrightarrow \mathtt{plus}(x_1; x_2)\ \text{abt}^0}$$

An Example-Parsing into ABT

LET ID[a] BE NUM[3] ADD NUM[3] IN LET ID[b] BE NUM[4] IN ID[a] ADD ID[b]

Factor	fct	::=	num | lit | id | LP prg RP
Term	trm	::=	fct | fct MUL trm | VB fct VB
Expression	exp	::=	trm | trm ADD exp | trm CAT exp
Program	prg	::=	exp | LET id BE trm IN prg

$$\frac{}{\mathcal{A}, x\ abt^0 \vdash x\ abt^0}$$

$$\frac{ar\ (o) = (n_1, \cdots, n_k) \quad \mathcal{A} \vdash a_1\ \mathrm{abt}^{n_1} \quad \cdots \quad \mathcal{A} \vdash a_k\ \mathrm{abt}^{n_k}}{\mathcal{A} \vdash o(a_1, \cdots, a_k)\ \mathrm{abt}^0}$$

$$\frac{x \# \mathcal{A} \quad \mathcal{A}, x\ \mathrm{abt}^0 \vdash a\ \mathrm{abt}^n}{\mathcal{A} \vdash x.a\ \mathrm{abt}^{n+1}}$$

$$\frac{\Gamma \vdash s_2\ \mathrm{exp} \longleftrightarrow a_2\ \mathrm{abt} \quad \Gamma, s_1\ \mathrm{id} \longleftrightarrow x\ \mathrm{abt} \vdash s_3\ \mathrm{prg} \longleftrightarrow a_3\ \mathrm{abt}}{\Gamma \vdash \mathtt{LET}\ s_1\ \mathtt{BE}\ s_2\ \mathtt{IN}\ s_3\ \mathrm{prg} \longleftrightarrow \mathtt{let}(a_2;\ x.a_3)\ \mathrm{abt}}$$

$$\Gamma, \mathtt{ID}[s]\ \mathrm{id} \longleftrightarrow x\ \mathrm{abt} \vdash \mathtt{ID}[s]\ \mathrm{id} \longleftrightarrow x\ \mathrm{abt}$$

- 对**LET ID[b] BE NUM[4] IN ID[a] ADD ID[b]**分析

$$\frac{x_1\ \mathrm{abt}^0, x_2\ \mathrm{abt}^0 \vdash \mathtt{plus}(x_1; x_2)\ \mathrm{abt}^0}{x_1\ \mathrm{abt}^0 \vdash x_2.\mathtt{plus}(x_1; x_2))\ \mathrm{abt}^1}$$

$$\frac{ar(\mathtt{let}) = (0,1) \quad \vdash \mathtt{num}[4]\ \mathrm{abt}^0 \quad x_1\ \mathrm{abt}^0 \vdash x_2.\mathtt{plus}(x_1; x_2)\ \mathrm{abt}^1}{x_1\ \mathrm{abt}^0 \vdash \mathtt{let}(\mathtt{num}[4]; x_2.\mathtt{plus}(x_1; x_2))\ \mathrm{abt}^0}$$

$$\frac{\vdash \mathtt{NUM}[4]\ \mathrm{exp} \longleftrightarrow \mathtt{num}[4]\ \mathrm{abt}^0 \quad \mathtt{ID}[a]\ \mathrm{id} \longleftrightarrow x_1\ \mathrm{abt}^0, \mathtt{ID}[b]\ \mathrm{id} \longleftrightarrow x_2\ \mathrm{abt}^0 \vdash \mathtt{ID}[a]\ \mathtt{ADD}\ \mathtt{ID}[b]\ \mathrm{prg} \longleftrightarrow \mathtt{plus}(x_1; x_2)\ \mathrm{abt}^0}{\mathtt{ID}[a]\ \mathrm{id} \longleftrightarrow x_1\ \mathrm{abt}^0 \vdash \mathtt{LET}\ \mathtt{ID}[b]\ \mathtt{BE}\ \mathtt{NUM}[4]\ \mathtt{IN}\ \mathtt{ID}[a]\ \mathtt{ADD}\ \mathtt{ID}[b]\ \mathrm{exp} \longleftrightarrow \mathtt{let}(\mathtt{num}[4]; x_2.\mathtt{plus}(x_1; x_2))\ \mathrm{abt}^0}$$

一个例子-静态语义

- ABT $\texttt{let(plus(num[3];num[3]);}\ x_1\texttt{.let(num[4];}\ x_2\texttt{.plus(}x_1;\ x_2\texttt{)))}$
- 类型检查 (static semantics)

$$\frac{}{\Gamma, x : \tau \vdash x : \tau} \qquad \frac{}{\Gamma \vdash \texttt{num}[n] : \texttt{num}} \qquad \frac{\Gamma \vdash e_1 : \texttt{num} \quad \Gamma \vdash e_2 : \texttt{num}}{\Gamma \vdash \texttt{plus}(e_1; e_2) : \texttt{num}} \qquad \frac{\Gamma \vdash e_1 : \tau_1 \quad \Gamma, x : \tau_1 \vdash e_2 : \tau_2}{\Gamma \vdash \texttt{let}(e_1; x.e_2) : \tau_2}$$

自底向上的类型检查

$$\frac{x_1 : \texttt{num} \vdash x_1 : \texttt{num} \qquad x_2 : \texttt{num} \vdash x_2 : \texttt{num}}{x_1 : \texttt{num}, x_2 : \texttt{num} \vdash \texttt{plus}(x_1; x_2) : \texttt{num}}$$

$$\frac{\vdash \texttt{num}[4] : \texttt{num} \qquad x_1 : \texttt{num}, x_2 : \texttt{num} \vdash \texttt{plus}(x_1; x_2) : \texttt{num}}{x_1 : \texttt{num} \vdash \texttt{let}(\texttt{num}[4]; x_2.\texttt{plus}(x_1; x_2)) : \texttt{num}}$$

$$\frac{\vdash \texttt{num}[3] : \texttt{num}}{\vdash \texttt{plus}(\texttt{num}[3]; \texttt{num}[3]) : \texttt{num}}$$

$$\frac{\vdash \texttt{plus}(\texttt{num}[3]; \texttt{num}[3]) : \texttt{num} \qquad x_1 : \texttt{num} \vdash \texttt{let}(\texttt{num}[4]; x_2.\texttt{plus}(x_1; x_2)) : \texttt{num}}{\vdash \texttt{let}(\texttt{plus}(\texttt{num}[3]; \texttt{num}[3]); x_1.\texttt{let}(\texttt{num}[4]; x_2.\texttt{plus}(x_1; x_2))) : \texttt{num}}$$

一个例子-结构语义

- ABT let(plus(num[3];num[3]); x_1.let(num[4]; x_2.plus(x_1; x_2)))
- 执行 (Structural Semantics)

$$\frac{}{\texttt{num}[n]\ \mathsf{val}} \qquad \frac{}{\texttt{str}[s]\ \mathsf{val}} \qquad \frac{n_1 + n_2 = n\ \mathsf{nat}}{\texttt{plus}(\texttt{num}[n_1];\texttt{num}[n_2]) \mapsto \texttt{num}[n]} \qquad \frac{e_1\ \mathsf{val} \quad e_2 \mapsto e_2'}{\texttt{plus}(e_1;e_2) \mapsto \texttt{plus}(e_1;e_2')}$$

$$\frac{e_1 \mapsto e_1'}{\texttt{plus}(e_1;e_2) \mapsto \texttt{plus}(e_1';e_2)} \qquad \frac{e_1\ \mathsf{val}}{\texttt{let}(e_1;x.e_2) \mapsto [e_1/x]e_2} \qquad \frac{e_1 \mapsto e_1'}{\texttt{let}(e_1;x.e_2) \mapsto \texttt{let}(e_1';x.e_2)}$$

$$\begin{aligned}
&\texttt{let}(\texttt{plus}(\texttt{num}[3];\texttt{num}[3]);x_1.\texttt{let}(\texttt{num}[4];x_2.\texttt{plus}(x_1;x_2)))\\
\mapsto\quad &\texttt{let}(\texttt{num}[6];x_1.\texttt{let}(\texttt{num}[4];x_2.\texttt{plus}(x_1;x_2)))\\
\mapsto\quad &\texttt{let}(\texttt{num}[4];x_2.\texttt{plus}(\texttt{num}[6];x_2))\\
\mapsto\quad &\texttt{plus}(\texttt{num}[6];\texttt{num}[4])\\
\mapsto\quad &\texttt{num}[10]
\end{aligned}$$

一个例子-上下文语义

- ABT let(plus(num[3];num[3]); x_1.let(num[4]; x_2.plus(x_1; x_2)))
- 执行 (Contextual Semantics)

$$\frac{m+n=p \text{ nat}}{\texttt{plus}(\texttt{num}[m];\texttt{num}[n]) \rightsquigarrow \texttt{num}[p]} \qquad \frac{e_1 \text{ val}}{\texttt{let}(e_1;x.e_2) \rightsquigarrow [e_1/x]e_2} \qquad \frac{}{e = \circ\{e\}}$$

$$\frac{e_1 = \mathcal{E}_1\{e\}}{\texttt{plus}(e_1;e_2) = \texttt{plus}(\mathcal{E}_1;e_2)\{e\}} \qquad \frac{e_1 \text{ val} \quad e_1 = \mathcal{E}_2\{e\}}{\texttt{plus}(e_1;e_2) = \texttt{plus}(e_1;\mathcal{E}_2)\{e\}} \qquad \frac{e_1 = \mathcal{E}_1\{e\}}{\texttt{let}(e_1;x.e_2) = \texttt{let}(\mathcal{E}_1;x.e_2)\{e\}}$$

$$\frac{e = \mathcal{E}\{e_0\} \quad e_0 \rightsquigarrow e_0' \quad e' = \mathcal{E}\{e_0'\}}{e \mapsto e'}$$

$$\begin{array}{ll}
\multicolumn{2}{l}{\texttt{let}(\texttt{plus}(\texttt{num}[3];\texttt{num}[3]);x_1.\texttt{let}(\texttt{num}[4];x_2.\texttt{plus}(x_1;x_2)))} \\
= & \texttt{let}(\circ;x_1.\texttt{let}(\texttt{num}[4];x_2.\texttt{plus}(x_1;x_2)))\{\texttt{plus}(\texttt{num}[3];\texttt{num}[3])\} \\
\mapsto & \texttt{let}(\circ;x_1.\texttt{let}(\texttt{num}[4];x_2.\texttt{plus}(x_1;x_2)))\{\texttt{num}[6]\} \\
= & \circ\{\texttt{let}(\texttt{num}[6];x_1.\texttt{let}(\texttt{num}[4];x_2.\texttt{plus}(x_1;x_2)))\} \\
\mapsto & \circ\{\texttt{let}(\texttt{num}[4];x_2.\texttt{plus}(\texttt{num}[6];x_2))\} \\
\mapsto & \circ\{\texttt{plus}(\texttt{num}[6];\texttt{num}[4])\} \\
\mapsto & \circ\{\texttt{num}[10]\}
\end{array}$$

一个例子-求值语义

- ABT $\text{let}(\text{plus}(\text{num}[3];\text{num}[3]);\ x_1.\text{let}(\text{num}[4];\ x_2.\text{plus}(x_1;\ x_2)))$
- 执行 (Evaluation Semantics)

$$\frac{}{\texttt{num}[n] \Downarrow \texttt{num}[n]} \qquad \frac{e_1 \Downarrow \texttt{num}[n_1] \quad e_2 \Downarrow \texttt{num}[n_2] \quad n_1 + n_2 = n \text{ nat}}{\texttt{plus}(e_1; e_2) \Downarrow \texttt{num}[n]} \qquad \frac{e_1 \Downarrow v_1 \quad [v_1/x]e_2 \Downarrow v_2}{\texttt{let}(e_1; x.e_2) \Downarrow v_2}$$

自底向上

$$\frac{\texttt{num}[3] \Downarrow \texttt{num}[3] \quad 3 + 3 = 6 \text{ nat}}{\texttt{plus}(\texttt{num}[3]; \texttt{num}[3]) \Downarrow \texttt{num}[6]}$$

$$\frac{\texttt{num}[4] \Downarrow \texttt{num}[4] \quad \texttt{plus}(\texttt{num}[6]; \texttt{num}[4]) \Downarrow \texttt{num}[10]}{\texttt{let}(\texttt{num}[4]; x_2.\texttt{plus}(\texttt{num}[6]; x_2)) \Downarrow \texttt{num}[10]}$$

$$\frac{\texttt{plus}(\texttt{num}[3]; \texttt{num}[3]) \Downarrow \texttt{num}[6] \quad \texttt{let}(\texttt{num}[4]; x_2.\texttt{plus}(\texttt{num}[6]; x_2)) \Downarrow \texttt{num}[10]}{\texttt{let}(\texttt{num}[6]; x_1.\texttt{let}(\texttt{num}[4]; x_2.\texttt{plus}(x_1; x_2))) \Downarrow \texttt{num}[10]}$$

3.7 类型和语言

- **类型安全**

  表达静态语义和动态语义之间的一致性

  – 静态语义预测表达式值将具有某种形式，使得表达式的动态语义是良定义的.

3.7.1 类型安全(Type Safety)[PFPL, 11]

3.7.2 运行时错误[PFPL, 11]

3.7.3 阶段上的区别[PFPL, 13]

3.7.4 引入和消去[PFPL, 13]

3.7.5 组合性[PFPL, 13]

3.7.6 变量和值[PFPL, 13]

3.7.1 类型安全-1

- **Types** $\tau ::= \text{num} \mid \text{str}$
- **Values** $v ::= \texttt{num}[n] \mid \texttt{str}[s], \quad n \text{ nat}, \ s \text{ str}$
- **Expr** $e ::= x \mid \texttt{num}[n] \mid \texttt{str}[s] \mid \texttt{plus}(e_1; e_2) \mid \texttt{times}(e_1; e_2) \mid \texttt{cat}(e_1; e_2) \mid \texttt{len}(e) \mid \texttt{let}(e_1; x.e_2)$
- 定型规则**Typing rules**

$$\frac{\Gamma \vdash e_1 : \texttt{num} \quad \Gamma \vdash e_2 : \texttt{num}}{\Gamma \vdash \texttt{plus}(e_1; e_2) : \texttt{num}}$$

- 定型的逆转**Inversion for Typing**

如果 $\Gamma \vdash e : \tau$, **e=plus(e$_1$;e$_2$)**, 那么 τ = **num**,

$\Gamma \vdash e_1 : \texttt{num}$ 且 $\Gamma \vdash e_2 : \texttt{num}$.

3.7.1 类型安全-2

❖ Theorem 11.1 (Type safety for $\mathcal{L}${num str})

1.(preservation) 如果 $e : \tau$ 且 $e \mapsto e'$,则 $e' : \tau$.

2.(progress) 如果 $e : \tau$, 那么或者 e val, 或者存在 e' 使得 $e \mapsto e'$.

➢ 保持性(Preservation): 计算的每一步都保持定型.

➢ 进展性(Progress): 确保良类型的表达式或者是值，或者可以被进一步计算.

e是受阻的(stuck)当且仅当它不是一个值，而且也不存在e'使得 $e \mapsto e'$.

一个受阻的状态必然是不良类型的(ill-typed).

进展性：良类型的程序不会到达受阻状态。

3.7.1 类型安全-保持性-1

❖ 保持性　　如果 $e : \tau$ 并且 $e \mapsto e'$ ，则 $e' : \tau$.

证明：对转换(transition)断言的推导规则进行规则归纳.

情况 1

$$\frac{e_1 \mapsto e_1'}{\mathtt{plus}(e_1; e_2) \mapsto \mathtt{plus}(e_1'; e_2)}$$

假设：　$\mathtt{plus}(e_1; e_2) : \tau$

由定型的逆转引理，有 $\tau = \mathrm{num}$, $e_1 : \mathrm{num}$, $e_2 : \mathrm{num}$

再由归纳原理，有 $e_1' : \mathrm{num}$

从而有　$\mathtt{plus}(e_1'; e_2) : \mathrm{num}$

故得证。

3.7.1 类型安全-保持性-2

❖**保持性**　　如果 $e : \tau$ 并且 $e \mapsto e'$，则 $e' : \tau$.

证明： 根据转换断言规则进行规则归纳证明.

情况 2
$$\frac{e_1 \ \mathsf{val}}{\mathtt{let}(e_1; x.e_2) \mapsto [e_1/x]e_2}$$

假设：　$\mathtt{let}(e_1; x.e_2) : \tau_2$

由定型的逆转引理9.2，对于某些 τ_1 有 $e_1 : \tau_1$

使得 $x : \tau_1 \vdash e_1 : \tau_2$

由置换引理9.4，可得 $[e_1/x]e_2 : \tau_2$

故得证。

…… // 其他情况

3.7.1 类型安全-保持性-3

❖保持性

➢对保持性的证明不能按表达式**e**的结构进行归纳，因为在绝大多数情况下，会有不止一个转换规则适用于一个表达式。

例如： 对于**plus(e$_1$;e$_2$)**,可以有以下转换规则

$$\frac{n_1 + n_2 = n \text{ nat}}{\texttt{plus}(\texttt{num}[n_1]; \texttt{num}[n_2]) \mapsto \texttt{num}[n]}$$

$$\frac{e_1 \mapsto e_1'}{\texttt{plus}(e_1; e_2) \mapsto \texttt{plus}(e_1'; e_2)}$$

$$\frac{e_1 \text{ val} \qquad e_2 \mapsto e_2'}{\texttt{plus}(e_1; e_2) \mapsto \texttt{plus}(e_1; e_2')}$$

3.7.1 类型安全-进展性-1

进展性 如果$e : \tau$，则或者e val ,或者存在 e'使得 $e \mapsto e'$.

引理11.3(范式Canonical Forms):

如果e val且e:τ，那么：

1. 如果τ= num，则e = num[n](n为某一数值)是范式。
2. 如果τ= str，则e = str[s](s为某一串)是范式。

证明 按定型规则(9.1)和值规则(10.1)进行归纳。

没有求值规则可以作用在范式e上。

表达式e求值终止，是指存在某个范式e'，使得 $e \mapsto^* e'$

3.7.1 类型安全-进展性-2

进展性 如果$e : \tau$，则或者e val ,或者存在 e'使得$e \mapsto e'$.

证明： 对定型推导进行归纳.

情况 1 $\dfrac{e_1 : \mathtt{num} \quad e_2 : \mathtt{num}}{\mathtt{plus}(e_1; e_2) : \mathtt{num}}$ 上下文为空表示只考虑闭项

由归纳法，有：

1) e_1 val ：由归纳法有

a) e_2 val：则由范式引理11.3,有$e_1 = \mathtt{num}[n_1]$，$e_2 = \mathtt{num}[n_2]$

从而 $\mathtt{plus}(\mathtt{num}[n_1]; \mathtt{num}[n_2]) \mapsto \mathtt{num}[n_1 + n_2]$

b) 存在e_2'使得 $e_2 \mapsto e_2'$ ：由转换规则，有

$$\mathtt{plus}(e_1; e_2) \mapsto \mathtt{plus}(e_1; e_2')$$

2) 存在 e_1'，使得$e_1 \mapsto e_1'$

$$\mathtt{plus}(e_1; e_2) \mapsto \mathtt{plus}(e_1'; e_2)$$

3.7.1 类型安全-进展性-3

❖ 进展性

- L{num,str}的定型规则是语法制导的，所以进展性就等于按表达式e的结构进行归纳。

- 当定型规则不是语法制导的时候，即可能存在不止一个规则适用于一个给定的表达式。

例如：while b do c

b为假，-

b为真，c; while b do c

3.7.2 运行时错误-1

- 对L{num,str}进行扩展，增加除法div(e_1;e_2)运算

  假设对除法运算没有指定除0的语义

$$\frac{e_1 : \mathtt{num} \quad e_2 : \mathtt{num}}{\mathtt{div}(e_1; e_2) : \mathtt{num}}$$

  这时，div(num[2];num[0])是良类型的，但求值时会受阻

  - 解决办法1：增强类型系统，使得良类型的程序不会执行除0操作

    这要求类型检查器要能证明分母非0

    ➔对多数程序来说，很难确定！

  - 解决办法2：增加动态检查，使得除0会导致求值结果为错误
    - 无需检查的错误(unchecked error):由类型系统来排除
    - 要检查的错误(checked error)：需要定义检查这种错误的动态语义

3.7.2 运行时错误-2

❖ 对动态检查错误的形式化-方法1

➢ 增加断言e err 以及对断言的归纳定义：

引起错误

$$\frac{e_1\ \mathrm{val}}{\mathtt{div}(e_1;\mathtt{num}[0])\ \mathrm{err}}$$

传播错误

$$\frac{e_1\ \mathrm{err}}{\mathtt{plus}(e_1;e_2)\ \mathrm{err}} \qquad \frac{e_1\ \mathrm{val} \quad e_2\ \mathrm{err}}{\mathtt{plus}(e_1;e_2)\ \mathrm{err}} \tag{11.1}$$

➢ 保持性定理不受影响

➢ 带错误检查的进展性：要考虑检查出的错误

Theorem11.5. 如果e:τ,则或者e err,或者e val，或者存在e',使得 $e \mapsto e'$ 。

证明：对定型规则归纳证明，与前面的证明类似，只是现在要考虑三种情况。

3.7.2 运行时错误-3

❖ 对动态检查错误的形式化

方法1：需要一组特殊的求值规则来检查错误

方法2：通过增加error表达式，将求值与错误检查合二为一。

- 定型规则：增加如下规则 $\overline{\texttt{error} : \tau}$ **(11.2)**
- 动态语义：

  增加引起错误的规则 $\dfrac{e_1 \ \text{val}}{\texttt{div}(e_1; \texttt{num}[0]) \mapsto \texttt{error}}$ **(11.3)**

  增加一些规则以传播错误，如

$$\overline{\texttt{plus}(\texttt{error}; e_2) \mapsto \texttt{error}} \qquad \frac{e_1 \ \text{val}}{\texttt{plus}(e_1; \texttt{error}) \mapsto \texttt{error}}$$

3.7.3 阶段上的区别(Phase Distinction)

- 静态语义 vs. 动态语义
  - 静态语义(定型规则)对程序中的结构进行约束，以保证动态语义(求值规则)是良行为的(well-behaved)
- 静态阶段 vs. 动态阶段
  - 静态阶段发生在动态阶段之前，二者相互独立
  - 静态阶段预测表达式在动态阶段所求的值的形式
- 类型安全定理(进展性和保持性)
  - 静态语义所预测的是动态语义中的真集，否则动态语义会到达受阻状态
  - 如何处理安全性的反例：或者增强静态语义保证反例被禁止；或者增强动态语义确保在运行时能对错误条件进行检查

3.7.4 引入和消去-1

- 和类型相关的基本操作
  - 引入形式(Introduction)：构造属于这种类型的值

    例：nat类型的引入形式是数值

    str类型的引入形式是字符串
  - 消去形式(Elimination)：这类值所能进行的运算

    例：nat类型的消去形式是加、乘运算

    str类型的消去形式是连接、求长度运算

  就λ演算而言,引入形式为λ抽象，消去形式为λ应用。
- 动态语义以逆转原理(inversion principle)为基础

  [逆转原理]例:假设$e:\tau$,如果e=num[n]，那么τ=num

3.7.4 引入和消去-2

- 消去形式是引入形式的逆

  引入是构造这种类型的值，而消去则是确定这种类型的值所能进行的运算

- 消去形式所得到的是由传给它的参数来确定

例： plus(e_1;e_2)的结果是一个数值，它由参数e_1和e_2的值来得到，两个参数都是数值。

- 可以将类型安全定理看成是对语言逆转原理的验证

例：对于plus(e_1;e_2)，

类型保持定理确保plus的参数e_1和e_2的类型必须是num，从而由范式引理，可得e_1和e_2的值必须是数值。这保证plus的进展性，它产生一个数值，其类型为num 。

3.7.4 引入和消去-3

❖ 逆转原理可以指导对动态语义的推导，但在多数情况下不能完全决定动态语义

假设对L{num,str}增加ifz($e;e_1;e_2$)条件表达式，

考虑消去形式ifz($e;e_1;e_2$)：e是num类型，如果e计算为0，则结果为e_1，否则为e_2。

e的求值结果将决定是继续计算e_1,还是e_2,而不是对e_1和e_2都计算。

➢ 主要参数 vs. 次要的参数

– 对于ifz($e;e_1;e_2$)来说， e是主要参数，e_1和e_2是次要参数

对一个消去形式来说，其主要参数必须被计算，而次要参数则不必被计算。

3.7.4 引入和消去-4

❖ 引入形式的参数计算

假设将L{num,str}中的数值换成z和s(e)(分别表示0和后继)

z和s(e)均为num类型的引入形式。

s(e)是值的前提是要求e本身是值？还是不管e是否是值？

- 激进eager(严格)的运算 要求引入形式的参数是值
  在动态语义中就必须有关联的search规则
- 惰性lazy(不严格)的运算 不要求引入形式的参数是值
  e.g. s(plus(z;s(z))) 是值

如果语言中所有引入形式的参数是激进的，则该语言是激进的。

如果语言中所有引入形式的参数是惰性的，则该语言是惰性的。

3.7.5 组合性

❖ 组合性(compositionality)

由定型的置换性质和传递性(见3.5.2节)可以得到类型系统的一个基本性质，称为组合性或模块性(modularity)

$$\frac{\Gamma \vdash e : \tau \quad \Gamma, x : \tau \vdash e' : \tau'}{\Gamma \vdash [e/x]e' : \tau'}$$

上述规则体现连接(linking)的本质

- e'中含有τ类型的自由变量x
- linker的任务是通过置换x,将e和e'组合起来，得到一个完整的编译单元
- 客户端e'可以单独进行类型检查，而与共享组件e的实现无关

类型提供了模块性的基础。

3.7.6 变量和值

如果一个变量在执行时永远只绑定到一个值上，则称该变量为**值变量**，否则为**计算变量**。

➢ **计算变量** $x_1 : \tau_1, \ldots, x_n : \tau_n \vdash e : \tau.$

$$\frac{\Gamma \vdash e : \tau \quad \Gamma, x : \tau \vdash e' : \tau'}{\Gamma \vdash [e/x]e' : \tau'}$$

x可以代换为任何类型为τ的表达式

➢ **值变量** $x_1 \text{ val}, \ldots, x_n \text{ val}\ x_1 : \tau_1, \ldots, x_n : \tau_n \vdash e : \tau,$

$$\frac{\Phi\,\Gamma \vdash e : \tau \quad \Phi \vdash e \text{ val} \quad \Phi, x \text{ val}\ \Gamma, x : \tau \vdash e' : \tau'}{\Phi\,\Gamma \vdash [e/x]e' : \tau'}$$

Φ表示一组形如 x_i val的假设

注意：**e**是开放的值(含有自由变量)，即

$$x_1 \text{ val}, \ldots, x_n \text{ val} \vdash e \text{ val}$$

如: $x \text{ val} \vdash \mathsf{s}(\mathsf{s}(x)) \text{ val}$ 与后继运算是**eager/lazy**无关

Thanks!