习题课

2017-11-23

H7

- 4.12(b) 文法如下：

  $S \rightarrow (L) \mid a$

  $L \rightarrow L, S \mid S$

(1)写一个翻译方案，它打印出每个a在句子中是第几个字符。例如，当句子是(a,(a,(a,a),(a)))时，打印的结果是2， 5， 8， 10， 14。

(2)写出相应的语法制导定义

(3)写出相应的预测翻译器

(4)写出自下而上分析的栈操作代码

概念区分

- 语义规则和产生式相联系的两种方式
  - 语法制导定义
    - 将文法符号和某些属性相关联，并通过语义规则来描述如何计算属性的值，没有描述这些规则的计算时机
  - 语法制导的翻译方案
    - 在产生式的右部的适当位置，插入相应的语义动作，按照分析的进程，执行遇到的语义动作，从而明确了语法分析过程中属性的计算时机。

H7

- 4.12(b) 文法如下：

$S \to (L) \mid a$

$L \to L, S \mid S$

(1)写一个翻译方案，它打印出每个a在句子中是第几个字符。例如，当句子是(a,(a,(a,a),(a)))时，打印的结果是2，5， 8， 10， 14。

- a自身的信息无法确定a在序列中的位置，因此必须要借助继承属性。
- 方法一：
  - 继承属性 in：该文法符号推出的字符序列的前面已经有多少字符
  - 综合属性 out：该文法符号推出的字符序列的最后一个字符在序列中是第几个字符

```
S’ → { S.in = 0; }  S
S  → { L.in = S.in +1; }  (L)  { S.out = L.out + 1; }
S  → a { S.out = S.in + 1; print (S.out); }
L  → { L1.in = L.in; }  L1, { S.in = L1.out + 1; } S {L.out = S.out; }
L  → { S.in = L.in; }  S  { L.out = S.out; }
```

H7

- 4.12(b) 文法如下：

$S \rightarrow (L) \mid a$

$L \rightarrow L, S \mid S$

(1)写一个翻译方案，它打印出每个a在句子中是第几个字符。例如，当句子是(a,(a,(a,a),(a)))时，打印的结果是2，5， 8， 10， 14。

- a自身的信息无法确定a在序列中的位置，因此必须要借助继承属性。
- 方法二：
  - 继承属性 in：该文法符号推出的字符序列的前面已经有多少字符
  - 综合属性 total：该文法符号推出的字符序列所包含的字符总数

S’ → { S.in = 0; } S
S → { L.in = S.in +1; } (L) { S.total = L.total + 2; }
S → a { S.total = 1; print (S.in + 1); }
L → { L1.in = L.in; } L1, { S.in = L1.in + L1.total + 1; } S {L.total = L1.total +S.total + 1; }
L → { S.in = L.in; } S { L.total = S.total; }

H7

- 4.12(b) 文法如下：

  $S \rightarrow (L) \mid a$

  $L \rightarrow L, S \mid S$

  写出相应的语法制导定义

- a自身的信息无法确定a在序列中的位置，因此必须要借助继承属性。
- 方法一的语法制导定义：
  - 继承属性 in：该文法符号推出的字符序列的前面已经有多少字符
  - 综合属性 out：该文法符号推出的字符序列的最后一个字符在序列中是第几个字符

产生式	语义规则
S’ → S	S.in = 0;
S → (L)	L.in = S.in +1; S.out = L.out + 1;
S → a	S.out = S.in + 1; print (S.out);
L → L1, S	L1.in = L.in; S.in = L1.out + 1; L.out = S.out;
L → S	S.in = L.in; L.out = S.out;

H7

- 4.12(b) 文法如下：

  $S \to (L) \mid a$

  $L \to L, S \mid S$

  写出相应的预测翻译器

- 消除左递归

$S' \to S$
$S \to (L)$
$S \to a$
$L \to ST$
$T \to ,ST \mid \varepsilon$

产生式	语义规则
$S' \to S$	S.in = 0;
$S \to (L)$	L.in = S.in +1; S.out = L.out + 1;
$S \to a$	S.out = S.in + 1; print (S.out);
$L \to ST$	S.in = L.in; T.in = S.out; L.out = T.out;
$T \to ,ST_1$	S.in = T.in + 1; T_1.in = S.out; T.out = T_1.out
$T \to \varepsilon$	T.in = T.out

H7

产生式	语义规则
S' → S	S.in = 0;
S → (L)	L.in = S.in +1; S.out = L.out + 1;
S → a	S.out = S.in + 1; print (S.out);
L → ST	S.in = L.in; T.in = S.out; L.out = T.out;
T → ,ST$_1$	S.in = T.in + 1; T$_1$.in = S.out; T.out = T$_1$.out
T → ε	T.in = T.out

```
int S'(){
  return S(0);
}
```

```
int S(int b){
  int in, out;
  if(lookahead == '('){
      in = b + 1;
      match('(');
      out = L(in) + 1;
      match(')')
  }else
  {
       match('a');
       out = b + 1;
       print(out);
  }
   return out;
}
```

H7

产生式	语义规则
S' → S	S.in = 0;
S → (L)	L.in = S.in +1; S.out = L.out + 1;
S → a	S.out = S.in + 1; print (S.out);
L → ST	S.in = L.in; T.in = S.out; L.out = T.out;
T → ,ST_1	S.in = T.in + 1; T_1.in = S.out; T.out = T_1.out
T → ε	T.in = T.out

```
int L(int b){
  int out;
  out = S(b);
  out = T(out);
  return out;
}
```

```
int T(int b)
{
  int out;
  if(lookahead == ‘,’){
     match(‘,’);
     out = S(b+1);
     out = T(out);
   }else{
     out = b;
   }
   return out;
}
```

H7

- 4.12(b) 文法如下：

  $S' \to S$

  $S \to (L)$

  $S \to a$

  $L \to ST$

  $T \to ,ST \mid \varepsilon$

写出自下而上分析的栈操作代码

- 引入标记非终极符M,N,R,P

产生式	语义规则	栈操作代码
$S' \to MS$	S.in = M.out	Stack[top - 1] = Stack[top]
$M \to \varepsilon$	M.out = 0	Stack[top + 1] = 0
$S \to (NL)$	N.in = S.in + 1, L.in = N.out; S.out = L.out + 1;	Stack[top - 3] = Stack[top - 1] + 1
$N \to \varepsilon$	N.out = N.in	Stack[top + 1] = Stack[top - 1] + 1
$S \to a$	S.out = S.in + 1; print (S.out);	Stack[top] = Stack[top - 1] + 1
$L \to SRT$	S.in = L.in; R.in = S.in; T.in = R.out, L.out = T.out;	Stack[top - 2] = Stack[top]
$R \to \varepsilon$	R.out = R.in	Stack[top + 1] = Stack[top - 1]
$T \to ,SPT_1$	S.in = T.in + 1; P.in = S.in; T_1.in = P.out; T_1.out = S.out;	Stack[top - 3] = Stack[top]
$P \to \varepsilon$	P.out = P.in	Stack[top + 1] = Stack[top]
$T \to \varepsilon$	T.out = T.in	Stack[top] = Stack[top - 1]

H8

- 5.5 假如有下列C的声明:

```
typedef struct{
        int a, b;
} CELL, *PCELL;
CELL foo[100];
PCELL bar(x, y) int x; CELL y; {}
```

为变量foo和函数bar的类型写出类型表达式。

H8

- 5.5 假如有下列C的声明:

```
typedef struct{
    int a, b;
} CELL, *PCELL;
CELL foo[100];
PCELL bar(x, y) int x; CELL y; {}
```

为变量foo和函数bar的类型写出类型表达式。

CELL foo[100];

array(Range ?, TypeOfElement ?)

array(0..99, TypeOfElement ?)

array(0..99, CELL)

array(0..99, record((int a) $\times$ (int b)))

array(0..99, record((a $\times$ integer) $\times$ (b $\times$ integer)))

H8

- 5.5 假如有下列C的声明:

```
typedef struct{
      int a, b;
} CELL, *PCELL;
CELL foo[100];
PCELL bar(x, y) int x; CELL y; {}
```

为变量foo和函数bar的类型写出类型表达式。

PCELL bar(x, y) int x; CELL y; {}

TypeOfParameters? -> TypeOfReturnValue?

(int $\times$ CELL) -> PCELL

(integer $\times$ record((a $\times$ integer) $\times$ (b $\times$ integer))) -> PCELL

(integer $\times$ record((a $\times$ integer) $\times$ (b $\times$ integer))) -> pointer(record((a $\times$ integer) $\times$ (b $\times$ integer)))

H8

- 5.12 拓展5.3.3节的类型检查，使之能包含记录。有关记录部分的类型和记录域引用表达式的语法如下：

$$\mathrm{T} \rightarrow \boldsymbol{record}\ fields\ \mathbf{end}$$

$$fields \rightarrow fields; field \mid field$$

$$field \rightarrow \mathbf{id} : T$$

$$E \rightarrow E.\mathbf{id}$$

H8

- 5.12 拓展5.3.3节的类型检查，使之能包含记录。有关记录部分的类型和记录域引用表达式的语法如下：

$T \rightarrow \boldsymbol{record}\ fields\ \mathbf{end}$ {T.type = record(fields.type)}

$fields \rightarrow fields; field$ {fields.type = fields.type × field.type}

$fields \rightarrow field$ {fields.type = field.type}

$field \rightarrow \mathbf{id} : T$ {field.type = id.name × T.type}

$E \rightarrow E_1.\mathbf{id}$ {E.type = if(E1.type == record(t))
lookup(E1.type, id.name)
else
type_error;}

H8

- 5.13在文件stdlib.h中，关于qsort的外部声明如下：

extern void qsort(void *, size_t, size_t, int (*)(const void *, const void *));

用SPARC/Solaris C编译器编译下面的C程序时，错误信息如下：

type.c:24: warning: passing argument 4 of `qsort’ from incompatible pointer type

请你对该程序略作修改，使得该警告错误能消失，并且不改变程序的结果。

```
#include <stdlib.h>

typedef struct{
        int    Ave;
        double  Prob;
}HYPO;

HYPO ∗astHypo;
int  n;

int HypoCompare(HYPO∗stHypo1, HYPO∗stHypo2)
{
   if (stHypo1->Prob>stHypo2->Prob){
     return(-1);
   }else if (stHypo1->Prob<stHypo2->Prob) {
      return(1);
   }else{
      return(0);
   }
}/∗ end of function HypoCompare ∗/

main()
{
  qsort ( astHypo,n,sizeof(HYPO),HypoCompare);
}
```

H8

- 5.13在文件stdlib.h中， 关于qsort的外部声明如下：

extern void qsort(void *, size_t, size_t, int (*)(const void *, const void *));

问题：qsort的第四个形式参数类型与函数调用的传参类型不一致

```
#include <stdlib.h>

typedef struct{
        int    Ave;
        double  Prob;
}HYPO;

HYPO  *astHypo;
int  n;

int HypoCompare(HYPO *stHypo1, HYPO *stHypo2)
{
   if (stHypo1->Prob>stHypo2->Prob){
     return(-1);
   }else if (stHypo1->Prob<stHypo2->Prob) {
     return(1);
   }else{
     return(0);
   }
}/* end of function HypoCompare */

main()
{
  qsort ( astHypo, n, sizeof(HYPO), HypoCompare);
}
```

H8

- 5.13在文件stdlib.h中，关于qsort的外部声明如下：

extern void qsort(void *, size_t, size_t, int (*)(const void *, const void *));

问题：qsort的第四个形式参数类型与函数调用的传参类型不一致

方法一：修改HypoCompare函数形式参数的类型

```
#include <stdlib.h>

typedef struct{
        int    Ave;
        double  Prob;
}HYPO;

HYPO ∗astHypo;
int  n;

int HypoCompare(const void ∗stHypo1, const void∗stHypo2)
{
  if ((HYPO *)stHypo1->Prob>(HYPO *)stHypo2->Prob){
    return(-1);
  }else if ((HYPO *)stHypo1->Prob<(HYPO *)stHypo2->Prob) {
     return(1);
  }else{
     return(0);
  }
}/∗ end of function HypoCompare ∗/

main()
{
  qsort ( astHypo, n, sizeof(HYPO), HypoCompare);
}
```

H8

- 5.13在文件stdlib.h中， 关于qsort的外部声明如下：

extern void qsort(void *, size_t, size_t, int (*)(const void *, const void *));

问题： qsort的第四个形式参数类型与函数调用的传参类型不一致

方法二： 强制修改qsort函数调用中第四个参数的类型

```
#include <stdlib.h>

typedef struct{
        int    Ave;
        double  Prob;
}HYPO;

HYPO  *astHypo;
int  n;

int HypoCompare(HYPO *stHypo1, HYPO *stHypo2)
{
   if (stHypo1->Prob>stHypo2->Prob){
     return(-1);
   }else if (stHypo1->Prob<stHypo2->Prob) {
     return(1);
   }else{
     return(0);
   }
}/* end of function HypoCompare */

main()
{
  qsort ( astHypo, n, sizeof(HYPO), int (*)(const void *, const void *)
HypoCompare);
}
```

H9

- 5.16对下面的每对表达式，

(a) $\alpha_1 \rightarrow (\alpha_2 \rightarrow \alpha_1)$

(b) $array\ (\beta_1) \rightarrow (pointer\ (\beta_1) \rightarrow \beta_2)$

(c) $\gamma_1 \rightarrow \gamma_2$

找出(a) 和 (b) 、(b)和(c)最一般的合一代换：

H9

- (a)与(b)

$S(\alpha 1) = array (\beta 1)$
$S(\alpha 2) = pointer (\beta 1)$
$S(\beta 2) = array (\beta 1)$

- 5.16对下面的每对表达式，

(a) $\alpha_1 \rightarrow (\alpha_2 \rightarrow \alpha_1)$

(b) $array\ (\beta_1) \rightarrow (pointer\ (\beta_1) \rightarrow \beta_2)$

(c) $\gamma_1 \rightarrow \gamma_2$

找出(a) 和 (b) 、(b)和(c)最一般的合一代换：

H9

- (b)与(c)

- 5.16对下面的每对表达式，

(a) $\alpha_1 \rightarrow (\alpha_2 \rightarrow \alpha_1)$

(b) *array* $(\beta_1) \rightarrow$ (*pointer* $(\beta_1) \rightarrow \beta_2)$

(c) $\gamma_1 \rightarrow \gamma_2$

找出(a) 和 (b) 、(b)和(c)最一般的合一代换：

$S(\gamma 1) = \text{array} (\beta 1)$
$S(\gamma 2) = \text{pointer} (\beta 1) \rightarrow \beta 2$

H9

- 5.17效仿例5.5，推导下面map的多态类型：

map : $\forall \alpha. \forall \beta. ((\alpha \to \beta) \times list\ (\alpha)) \to list\ (\beta)$

map的ML定义是

```
fun map ( f, l ) =
   if null (l ) then nil
   else cons (f (hd (l)), map (f, tl (l ) ) );
```

在这个函数体中，内部定义的标识符的类型是：

null : $\forall \alpha. list\ (\alpha) \to$ boolean ;

nil : $\forall \alpha. list\ (\alpha)$;

cons : $\forall \alpha. (\alpha \times list\ (\alpha)) \to list\ (\alpha)$;

hd : $\forall \alpha. list\ (\alpha) \to \alpha$;

tl : $\forall \alpha. list\ (\alpha) \to list\ (\alpha)$;

H9

- 第一步：列出类型声明和要检查的表达式

f : α

l : β

if: $\forall\alpha$. boolean × *list* (α) × *list* (α) → *list* (α)

null : $\forall\alpha$. *list* (α) → boolean ;

nil : $\forall\alpha$. *list* (α) ;

cons : $\forall\alpha$. (α × *list* (α)) → *list* (α) ;

hd : $\forall\alpha$. *list* (α) → α ;

tl : $\forall\alpha$. *list* (α) → *list* (α) ;

```
match(
  map ( f, l ),
  if null (l ) then nil
           else cons (f (hd (l)), map (f, tl (l ) ) );
 )
```

- 5.17效仿例5.5，推导下面map的多态类型：

map : $\forall\alpha$. $\forall\beta$. (($\alpha \rightarrow \beta$) × *list* (α)) → *list* (β)

map的ML定义是

```
fun map ( f, l ) =
  if null (l ) then nil
  else cons (f (hd (l)), map (f, tl (l ) ) );
```

在这个函数体中，内部定义的标识符的类型是：

null : $\forall\alpha$. *list* (α) → boolean ;

nil : $\forall\alpha$. *list* (α) ;

cons : $\forall\alpha$. (α × *list* (α)) → *list* (α) ;

hd : $\forall\alpha$. *list* (α) → α ;

tl : $\forall\alpha$. *list* (α) → *list* (α) ;

H9

- 第二步：代换推导

map : $\forall\alpha.\forall\beta.((\alpha\to\beta)\times list(\alpha))\to list(\beta)$

fun map (f, l) =

if null (l) then nil

else cons (f (hd (l)), map (f, tl (l)));

null : $\forall\alpha.\ list(\alpha)\to$ boolean ;

nil : $\forall\alpha.\ list(\alpha)$;

cons : $\forall\alpha.(\alpha\times list(\alpha))\to list(\alpha)$;

hd : $\forall\alpha.\ list(\alpha)\to\alpha$;   tl : $\forall\alpha.\ list(\alpha)\to list(\alpha)$;

行	定型断言	代换	规则
1	f : α		(Exp Id)
2	l : β		(Exp Id)
3	map : γ		(Exp Id)
4	map (f , l) : δ	γ = (α × β) → δ	(Exp Funcall)
5	null : list (α0) → boolean		(Exp Id Fresh)
6	null (l) : boolean	β = list (α0)	(Exp Funcall + (2))
7	nil : list (α1)		(Exp Id Fresh)
8	l : list (α0)		由（2）可得
9	hd : list (α2) → α2		(Exp Id Fresh)

H9

行	定型断言	代换	规则
9	hd : list (α2) → α2		(Exp Id Fresh)
10	hd (l) : α0	α2 = α0	(Exp Funcall)
11	f (hd (l)) : α3	α = α0→α3	(Exp Id)
12	f : α0→α3		由（1）可得
13	tl : list (α4)→ list (α4)		(Exp Id Fresh)
14	tl (l) : list (α0)	α4 = α0	(Exp Funcall)
15	map : ((α0→α3) ×list(α0))→ δ		由（3）可得
16	map (f , tl (l)) : δ		(Exp Funcall)
17	cons : α5 × list(α5) → list(α5)		(Exp Id Fresh)
18	cons (…) : list (α3)	α5 = α3 , δ=list(α3)	(Exp Funcall)
19	if : boolean × list(α6) × list(α6) →list(α6)		(Exp Id Fresh)
20	if (…) : list (α1)	α6 = α1 , α3 = α1	(Exp Funcall)
21	match : α7 × α7 → α7		(Exp Id Fresh)
22	match (…) : list (α1)	α7 = list (α1)	(Exp Funcall)

map : $\forall\alpha.\forall\beta.((\alpha\rightarrow\beta)\times list\ (\alpha))\rightarrow list\ (\beta)$

```
fun map ( f, l ) =
   if null (l ) then nil
   else cons (f (hd (l)), map (f, tl (l ) ) );
```

null : $\forall\alpha.\ list\ (\alpha)\rightarrow$ boolean ;

nil : $\forall\alpha.\ list\ (\alpha)$;

cons : $\forall\alpha.\ (\alpha\times list\ (\alpha))\rightarrow list\ (\alpha)$;

hd : $\forall\alpha.\ list\ (\alpha)\rightarrow\alpha$;

tl : $\forall\alpha.\ list\ (\alpha)\rightarrow list\ (\alpha)$;

至此有**map : ((α0→α1) × list(α0))→list(α1)**

所以**map : ∀ α . ∀ β . ((α → β) × *list (α)) → list (β)***

H9

- 5.21 使用例5.9的规则，确定下列哪些表达式有唯一类型（假定z是复数）：

  (a) $1*2*3$

  (b) $1 * (z * 2)$

  (c) $(1 * z) * z$

H9

- 5.21 使用例5.9的规则，确定下列哪些表达式有唯一类型（假定z是复数）：

  (a) $1*2*3$

  (b) $1*(z*2)$

  (c) $(1*z)*z$

- 运算规则：
  - int × int -> int
  - int × int -> complex
  - complex × complex -> complex

H9

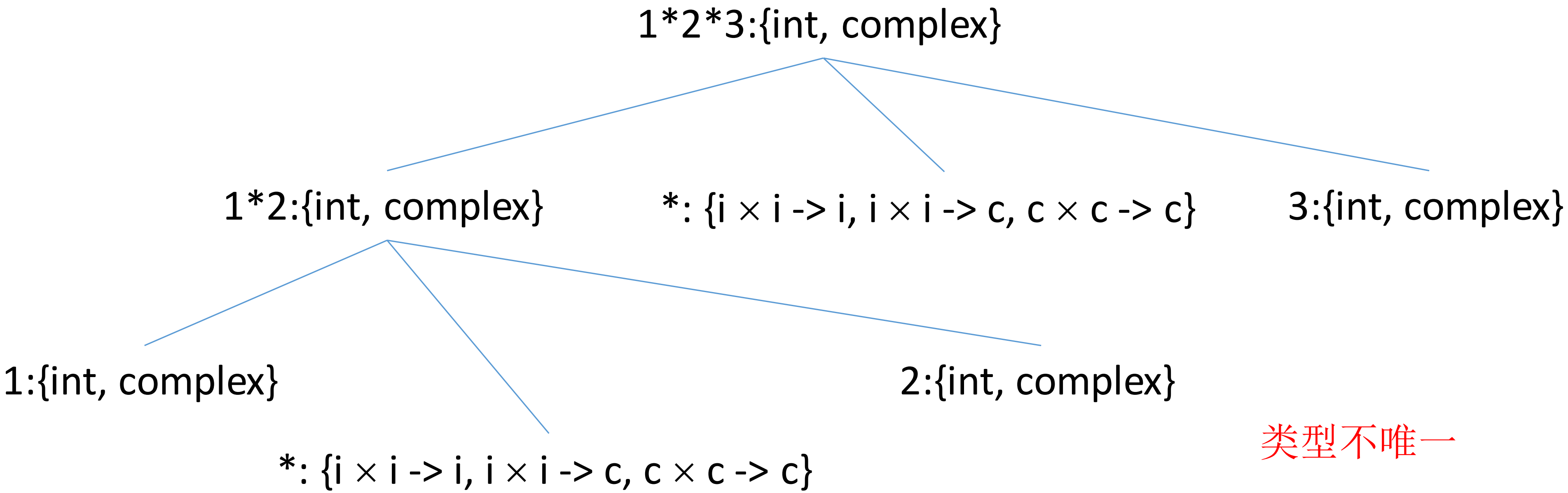
- 5.21 使用例5.9的规则，确定下列哪些表达式有唯一类型（假定z是复数）：

  (a) $1*2*3$

  (b) $1*(z*2)$

  (c) $(1*z)*z$

- 运算规则：
  - int × int -> int
  - int × int -> complex
  - complex × complex -> complex



类型不唯一

H9

- 5.21 使用例5.9的规则，确定下列哪些表达式有唯一类型（假定z是复数）：

  (a) 1∗2∗3

  (b) 1 ∗ (z ∗2)

  (c) (1 ∗ z) ∗ z

- 运算规则：
  - int × int -> int
  - int × int -> complex
  - complex × complex -> complex

1 ∗ (z ∗2) :{complex}

1:{int}

*: {i × i -> i, i × i -> c, c × c -> c}

z*2:{complex}

z:{complex}

*: {i X i -> i, i X i -> c, c X c -> c}

2:{int, complex}

类型唯一

H9

- 5.21 使用例5.9的规则，确定下列哪些表达式有唯一类型（假定z是复数）：

  (a) 1∗2∗3

  (b) 1 ∗ (z ∗2)

  (c) (1 ∗ z) ∗ z

- 运算规则：
  - int × int -> int
  - int × int -> complex
  - complex × complex -> complex

(1 ∗ z) ∗ z:{complex}

1*z:{complex}   *: {i × i -> i, i × i -> c, c × c -> c}   z:{complex}

1:{int, complex}   *: {i × i -> i, i × i -> c, c × c -> c}   z:{complex}

类型唯一