اصول طراحی کامپایلر

درس ۱۸

تحلیل معنایی (۳)

بررسی نوع

Semantic Analysis (3)

Type Checking

کاظم فولادی قلعه

دانشکده مهندسی، پردیس فارابی

دانشگاه تهران

http://courses.fouladi.ir/compiler

اصول طراحی کامپایلر

بررسی نوع

۱

مقدمه

بررسی نوع

TYPE CHECKING

یک کامپایلر باید بررسی کند که آیا برنامه‌ی ورودی از **قواعد نوع** پیروی می‌کند یا خیر.

بررسی‌گر نوع
Type Checker

ماژولی از یک کامپایلر که وظیفه‌ی آن بررسی نوع می‌باشد.

مثال‌هایی از بررسی نوع

عملگر **mod** فقط برای عملوندهای صحیح تعریف شده است.

اندیس‌گذاری فقط برای آرایه معنی دارد و اندیس باید عدد صحیح باشد.

یک تابع باید تعداد دقیقی آرگومان و پارامتر از نوع درست داشته باشد.

...

بررسی نوع

ایستا و پویا

بررسی نوع	
بررسی نوع ایستا *Static Type Checking*	**بررسی‌های نوع پویا** *Dynamic Type Checking*
زمان کامپایل برنامه	زمان اجرای برنامه
زبان‌هایی مانند *C Pascal Java* استفاده از بررسی نوع برای بررسی درستی برنامه پیش از اجرای آن	
مفید برای تعیین میزان حافظه‌ی لازم برای ذخیره‌سازی متغیرها	

بررسی نوع

۲

طراحی بررسی‌گر نوع

عبارت‌های نوع

TYPE EXPRESSIONS

عبارت نوع *Type Expression*
بیانگر نوع یک سازه‌ی زبان
هر عبارت نوع می‌تواند نامی داشته باشد که به آن نسبت داده شود. هر **نام نوع** *(Type Name)* یک عبارت نوع است.

نوع پایه *Basic Type*	**نوع ترکیبی** *Complex Type*
هر نوع پایه یک عبارت نوع است، مثل `int, real, char, bool` نوع `void` مجموعه‌ی تهی را نشان می‌دهد.	هر نوع ترکیبی یک عبارت نوع است که با اعمال یک **سازنده‌ی نوع** روی یک عبارت نوع به‌دست می‌آید.

سازنده‌های نوع

TYPE CONSTRUCTORS

سازنده‌های نوع *Type Constructors*				
ایجادکننده‌ی نوع ترکیبی از روی عبارت‌های نوع				
آرایه *Array*	**ضرب** *Product*	**رکورد** *Record*	**اشاره‌گر** *Pointer*	**تابع** *Function*

سازنده‌های نوع

آرایه

TYPE CONSTRUCTORS

سازنده‌های نوع *Type Constructors*				
ایجادکننده‌ی نوع ترکیبی از روی عبارت‌های نوع				
آرایه *Array*	ضرب *Product*	رکورد *Record*	اشاره‌گر *Pointer*	تابع *Function*

اگر T یک عبارت نوع باشد،

$\Downarrow$

$array(I, T)$ یک عبارت نوع است

که بیانگر

یک آرایه با عناصری از T و گستره‌ی اندیسی در I است.

`array[1..10] of int` $\mapsto$ $array(1..10, \text{int})$

سازنده‌های نوع

ضرب

TYPE CONSTRUCTORS

سازنده‌های نوع *Type Constructors*				
ایجادکننده‌ی نوع ترکیبی از روی عبارت‌های نوع				
تابع *Function*	اشاره‌گر *Pointer*	رکورد *Record*	**ضرب** ***Product***	آرایه *Array*

اگر T_1 و T_2 دو عبارت نوع باشد،

$\Downarrow$

$T_1 \times T_2$ یک عبارت نوع است

که بیانگر

ضرب دکارتی T_1 و T_2 است.

سازنده‌های نوع

رکورد

TYPE CONSTRUCTORS

سازنده‌های نوع *Type Constructors*				
ایجادکننده‌ی نوع ترکیبی از روی عبارت‌های نوع				
تابع *Function*	اشاره‌گر *Pointer*	**رکورد** ***Record***	ضرب *Product*	آرایه *Array*

مشابه ضرب است؛
اما با نام‌هایی برای فیلدهای مختلف
(برای دسترسی به عناصر رکورد)

```
struct{double r; int i;}
```
$\mapsto$ $double \times int$

سازنده‌های نوع

اشاره‌گر

TYPE CONSTRUCTORS

سازنده‌های نوع *Type Constructors*				
ایجادکننده‌ی نوع ترکیبی از روی عبارت‌های نوع				
تابع *Function*	**اشاره‌گر** ***Pointer***	رکورد *Record*	ضرب *Product*	آرایه *Array*

اگر T یک عبارت نوع باشد،

$\Downarrow$

$pointer(T)$ یک عبارت نوع است

که بیانگر

اشاره‌گر به عنصری از نوع T است.

`struct{double r; int i;}` $\mapsto$ $double \times int$

سازنده‌های نوع

تابع

TYPE CONSTRUCTORS

سازنده‌های نوع *Type Constructors*				
ایجادکننده‌ی نوع ترکیبی از روی عبارت‌های نوع				
تابع *Function*	اشاره‌گر *Pointer*	رکورد *Record*	ضرب *Product*	آرایه *Array*

اگر D دامنه و R برد تابع باشد،

$\Downarrow$

$D \to R$ یک عبارت نوع است

که بیانگر

نوع تابع می‌باشد.

عملگر `mod` $\quad \mapsto \quad int \times int \to int$

`function f(a,b:char):int` $\quad \mapsto \quad char \times char \to int$

سیستم نوع

TYPE SYSTEM

سیستم نوع
Type System

مجموعه‌ای از قواعد برای انتساب عبارت‌های نوع به اجزای مختلف یک برنامه

مشخص‌سازی سیستم‌های نوع با **ترجمه‌ی هدایت‌شده با نحو**

یک بررسی‌گر نوع، یک **سیستم نوع** را پیاده‌سازی می‌کند.

زبان‌های برنامه‌نویسی از نظر نوع

زبان‌های برنامه‌سازی	
قوی از لحاظ نوع *Strongly Typed*	**ضعیف از لحاظ نوع** *Weakly Typed*
اگر کامپایلر زبان **بتواند** تضمین کند که برنامه‌ی ورودی آن در صورت پذیرش بدون خطای نوع اجرا می‌شود.	اگر کامپایلر زبان **نتواند** تضمین کند که برنامه‌ی ورودی آن در صورت پذیرش بدون خطای نوع اجرا می‌شود.

مشخص‌سازی یک بررسی‌گر نوع

مثال

یک بررسی‌گر نوع برای یک زبان ساده:
در این زبان به **شناسه‌ها** یک **نوع** نسبت داده می‌شود.

$$P \rightarrow D; E$$

$$D \rightarrow D; D \mid \mathsf{id} : T$$

$$T \rightarrow \mathsf{char} \mid \mathsf{int} \mid \mathsf{array}[\mathsf{num}] \text{ of } T \mid \uparrow T$$

$$E \rightarrow \mathsf{literal} \mid \mathsf{num} \mid \mathsf{id} \mid E \bmod E \mid E[E] \mid E\uparrow$$

مشخص‌سازی یک بررسی‌گر نوع

مثال: تعریف هدایت‌شده با نحو

تعریف هدایت‌شده با نحو

برای انتساب نوع به یک شناسه (id)

PRODUCTION	SEMANTIC RULE
$D \rightarrow \text{id} : T$	addtype(id.entry,T.type)
$T \rightarrow \text{char}$	$T.type := char$
$T \rightarrow \text{int}$	$T.type := int$
$T \rightarrow \uparrow T_1$	$T.type := pointer(T_1.type)$
$T \rightarrow \text{array[num] of } T_1$	$T.type := array(1..\text{num}.val, T_1.type)$

* همه‌ی خصیصه‌ها از نوع ترکیبی هستند.

با توجه به قاعده‌ی $P \rightarrow D;E$ همه‌ی شناسه‌ها نوع خود را دارند
که پیش از بررسی نوع یک عبارت مانند E در جدول نمادها ذخیره شده است.

مشخص‌سازی یک بررسی‌گر نوع

مثال: تعریف هدایت‌شده با نحو

تعریف هدایت‌شده با نحو

برای انتساب نوع به یک عبارت *(Expression)*

PRODUCTION	SEMANTIC RULE
$E \rightarrow$ literal	$E.type := char$
$E \rightarrow$ num	$E.type := int$
$E \rightarrow$ id	$E.type := lookup(\text{id}.entry)$
$E \rightarrow E_1$ mod E_2	$E.type :=$ if $E_1.type = int$ and $E_2.type = int$ then int else type_error
$E \rightarrow E_1[E_2]$	$E.type :=$ if $E_2.type = int$ and $E_1.type =$ array(s,t) then t else type_error
$E \rightarrow E_1 \uparrow$	$E.type :=$ if $E_1.type =$ pointer(t) then t else type_error

مشخص‌سازی یک بررسی‌گر نوع

مثال: تعریف هدایت‌شده با نحو

تعریف هدایت‌شده با نحو

برای انتساب نوع به یک دستور *(Statement)*

PRODUCTION	SEMANTIC RULE
$S \rightarrow \text{id} := E$	$S.type :=$ if id.type = E.type then void else type_error
$S \rightarrow$ if E then S_1	$S.type :=$ if E.type = boolean then S_1.type else type_error
$S \rightarrow$ while E do S_1	$S.type :=$ if E.type = boolean then S_1.type else type_error
$S \rightarrow S_1; S_2$	$S.type :=$ if $S_1.type = void$ and $S_2.type = void$ then void else type_error

* عبارت نوع برای یک دستور، $void$ یا $type\_error$ است.

مشخص‌سازی یک بررسی‌گر نوع

مثال: تعریف هدایت‌شده با نحو

تعریف هدایت‌شده با نحو

برای انتساب نوع به یک تابع *(Function)*

PRODUCTION	SEMANTIC RULE
$D \rightarrow \text{id} : T$	addtype(id.entry,T.type); D.type := T.type
$D \rightarrow D_1; D_2$	$D.type := D_1.type \times D_2.type$
$Fun \rightarrow \text{fun id}(D) : T; B$	addtype(id.entry,D.type:T.type)
$B \rightarrow E$	
$E \rightarrow \text{id}(EList)$	$E.type :=$ if lookup(id.entry)$=t_1 : t_2$ and EList.type$= t_1$ then t_2 else type_error
$EList \rightarrow E$	$EList.type := E.type$
$EList \rightarrow EList, E$	$EList.type := EList_1.type \times E.type$

هم‌ارزی نوع‌ها

TYPE EQUIVALENCE

اهمیت تعریف هم‌ارزی دو نوع: برای زبان‌های دارای نام نوع یا امکان تعریف زیرنوع *(subtype)*

هم‌ارزی نوع	
هم‌ارزی ساختاری *Structural Equivalence*	**هم‌ارزی نامی** *Name Equivalence*
دو نوع با یکدیگر هم‌ارز **ساختاری** هستند اگر و فقط اگر ساختارهای آنها (گراف آنها) یکسان باشد.	دو نوع با یکدیگر هم‌ارز **نامی** هستند اگر و فقط اگر نام آنها یکسان باشد.
هر عبارت تعریف نوع را می‌توان با یک گراف جهت‌دار نمایش داد	
دشوار برای کامپایلر سادگی برای کدنویسی مثال: زبان *C*	ساده برای کامپایلر دشواری برای کدنویسی مثال: زبان *Java*

هم‌ارزی نوع‌ها

مثال

TYPE EQUIVALENCE

```
type vector: array[1..10] of integer;
type sequence: array[1..10] of integer;
var: X,Z: vector, Y: sequence;
```

هم‌ارزی ساختاری *Structural Equivalence*	هم‌ارزی نامی *Name Equivalence*
هر سه متغیر X، Y و Z	متغیرهای X و Z

تبدیل نوع

TYPE CASTING

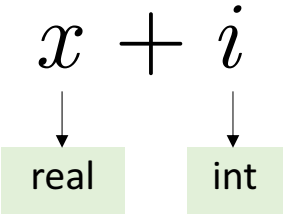
تبدیل نوع
Type Casting

تبدیل نوع یک متغیر توسط کامپایلر برای انجام یک محاسبه

تبدیل نوع

مثال

TYPE CASTING



نوع عبارت؟

بر اساس زبان، باید قواعد مشخصی برای تبدیل نوع توسط کامپایلر به‌کار رود
تا نوع یکی از عملوندهای + تغییر کند.

بررسی‌گر نوع، در یک کامپایلر می‌تواند عملگرهای تبدیل نوع را در کد میانی درج کند.
به این تبدیل نوع ضمنی، *coercion* می‌گویند.

تبدیل نوع ضمنی

مثال

COERCION

ترجمه‌ی هدایت‌شده با نحو
برای **تبدیل نوع ضمنی** از صحیح به حقیقی برای یک عمل محاسباتی کلی *op*

PRODUCTION	SEMANTIC RULE
$E \rightarrow$ num	$E.type := int$
$E \rightarrow$ num.num	$E.type := real$
$E \rightarrow$ id	$E.type := lookup(\text{id}.entry)$
$E \rightarrow E_1$ op E_2	$E.type :=$ if $E_1.type = int$ and $E_2.type = int$ then int else if $E_1.type = int$ and $E_2.type = real$ then real else if $E_1.type = real$ and $E_2.type = int$ then real else if $E_1.type = real$ and $E_2.type = real$ then real else type_error

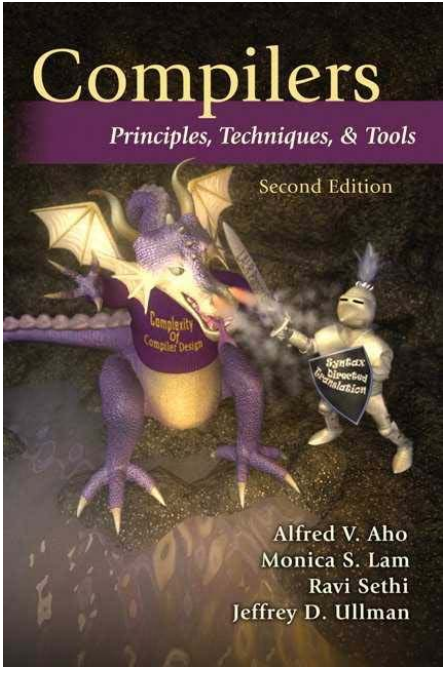
اصول طراحی کامپایلر

بررسی نوع

۳

منابع

منبع اصلی



A. V. Aho, M. S. Lam, R. Sethi, J. D. Ullman,
Compilers: Principles, Techniques and Tools,
Second Edition, Addison-Wesley, 2007.

Chapter 6 (6.5)