اصول طراحی کامپایلر

درس‌نامه‌ی ۱۴

کاظم فولادی
http://kazim.fouladi.ir

ویراست اول: ۱۳۸۸
ویراست دوم: ۱۳۹۳

فهرست مطالب

۱۴ بررسی نوع

TYPE CHECKING

۱-۱۴ مقدمه

- یک کامپایلر باید بررسی کند که آیا برنامه‌ی ورودی از قواعد نوع پیروی می‌کند یا خیر.
- اطلاعات مربوط به انواع داده‌ای (Data Types) توسط کامپایلر نگهداری و محاسبه می‌شود.
- یک بررسی‌گر نوع (Type Checker) ماژولی از یک کامپایلر است که وظیفه‌ی آن، بررسی نوع می‌باشد.

مثال

مثال‌هایی از وظایف بررسی‌گر نوع:

۱) عملگر mod تنها برای عملوندهای صحیح تعریف شده است.

۲) اندیس‌گذاری تنها برای یک آرایه معنا دارد و این اندیس باید یک عدد صحیح باشد.

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

۴) ...

- بررسی نوع می‌تواند به صورت **ایستا (Static)** یا **پویا (Dynamic)** باشد.
  - بررسی نوعی که در زمان کامپایل انجام می‌شود، بررسی **ایستا** نام دارد.
  - بررسی نوعی که در زمان اجرا انجام می‌شود، بررسی **پویا** نام دارد.
- زبان‌هایی مانند پاسکال، Java و C اساساً از بررسی نوع ایستا استفاده می‌کنند و از آن برای بررسی درستی برنامه پیش از اجرای آن بهره می‌برند. زبان‌هایی مانند Lisp، Python، Perl و Scheme از بررسی نوع پویا استفاده می‌کنند.
- بررسی نوع ایستا، برای تعیین میزان حافظه‌ی لازم برای ذخیره‌سازی متغیرها نیز مفید است.
- طراحی یک بررسی‌گر نوع وابسته به ساختار نحوی سازه‌های زبان، عبارت‌های نوع زبان و قواعد انتساب نوع‌ها به سازه‌هاست.

۲-۱۴ عبارت‌های نوع

یک **عبارت نوع**، نوع یک سازه‌ی زبان را نشان می‌دهد.
یک عبارت نوع، یک **نوع پایه** است یا با بکارگیری **سازنده‌های نوع** از دیگر نوع‌ها ساخته می‌شود:

۱) یک **نوع پایه (Basic Type)** یک عبارت نوع است، مانند `int`، `real`، `boolean`، `char`، ...
نوع پایه‌ی `void` یک مجموعه‌ی تهی را نشان می‌دهد.

۲) عبارت نوع می‌تواند دارای نامی باشد که به آن نسبت داده می‌شود.
هر **نام نوع (Type Name)** یک عبارت نوع را مشخص می‌کند.

۳) یک **سازنده‌ی نوع (Type Constructor)** که بر روی یک عبارت نوع اعمال می‌شود، یک عبارت نوع است.

۱-۲-۱۴ سازنده‌های نوع در عبارت‌های نوع

سازنده‌های نوع عبارتند از:

۱) **آرایه (Array)**: اگر T یک عبارت نوع باشد، در این صورت $array(I,T)$ یک عبارت نوع است که به یک آرایه با عناصری از T و گستره‌ی اندیسی در I دلالت می‌کند.
مانند: $array[1..10]\ of\ int == array(1..10, int)$

۲) **ضرب (Product)**: اگر T_1 و T_2 دو عبارت نوع باشند، آنگاه ضرب دکارتی آنها $T_1 \times T_2$ نیز یک عبارت نوع است.

۳) **رکورد (Record)**: رکورد مشابه ضرب است، اما با نام‌هایی برای فیلدهای مختلف (برای دسترسی به عناصر رکورد).
مانند: $struct\{double\ r;\ int\ i;\} == double \times int$

۴) **اشاره‌گر (Pointer)**: اگر T یک عبارت نوع باشد، آنگاه $pointer(T)$ یک عبارت نوع است: «اشاره‌گر به عنصری از نوع T».

۵) **تابع (Function)**: اگر D دامنه و R برد تابع باشد، در این صورت نوع آن را با عبارت نوع $D \to R$ نشان می‌دهیم.
مانند: عبارت نوع برای عملگر mod به صورت $int \times int \to int$
مانند: عبارت نوع برای تابع `function f(a,b:char):int` به صورت $char \times char \to int$

نکته. کلاس (class) در زبان‌های شیئ‌گرا، مشابه رکوردها تعریف می‌شود، با این تفاوت که نوع داده‌های کلاس، مشابه رکوردها و نوع متدها از نوع تابع تعریف می‌شود. با تعریف یک کلاس، نام آن به عنوان نام یک نوع در C++ یا Java استفاده می‌شود.

۲-۲-۱۴ نوع بازگشتی

نوع بازگشتی (recursive type) نوعی است که در تعریف آن از خودش استفاده شده باشد. برای مثال، در قطعه کد زیر، `Node` را در نظر بگیرید:

```
public class Node {...}
...
public Node n;
```

نام‌ها می‌توانند برای تعریف نوع‌های بازگشتی استفاده شوند که برای ساختمان‌داده‌هایی چون لیست‌های پیوندی لازم هستند. شبه کد برای عنصری از لیست به صورت

```
class Cell {int info; Cell next; ...}
```

نوع بازگشتی `Cell` را به صورت کلاسی تعریف می‌کند که حاوی فیلد `info` و فیلد `next` از نوع `Cell` است. نوع‌های بازگشتی مشابه در زبان C با استفاده از رکوردها و اشاره‌گرها تعریف می‌شوند.

۳-۱۴ سیستم نوع

یک **سیستم نوع**، مجموعه‌ای از قواعد برای انتساب عبارت‌های نوع به اجزای مختلف یک برنامه.

- سیستم‌های نوع با استفاده از ترجمه‌های هدایت شده با نحو مشخص‌سازی می‌شوند.
- یک **بررسی‌گر نوع** یک سیستم نوع را پیاده‌سازی می‌کند.

تعریف: یک زبان **قوی از لحاظ نوع (Strongly Typed)** نامیده می‌شود، اگر کامپایلر آن بتواند تضمین کند که برنامه‌ی ورودی آن در صورت پذیرش، بدون خطای نوع اجرا می‌شود.

۴-۱۴ مشخص‌سازی یک بررسی‌گر نوع

یک بررسی‌گر نوع برای یک زبان ساده که در آن به شناسه‌ها یک نوع نسبت داده شده است.

$$P \rightarrow D; E$$
$$D \rightarrow D; D \mid \mathsf{id} : T$$
$$T \rightarrow \mathsf{char} \mid \mathsf{int} \mid \mathsf{array}[\mathsf{num}] \text{ of } T \mid \ \uparrow T$$
$$E \rightarrow \mathsf{literal} \mid \mathsf{num} \mid \mathsf{id} \mid E \bmod E \mid E[E] \mid E\uparrow$$

۱-۴-۱۴ تعریف هدایت شده با نحو برای انتساب نوع به یک شناسه (id)

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

- همه‌ی خصیصه‌ها از نوع ترکیبی هستند.
- از آنجا که $P \rightarrow D; E$، همه‌ی شناسه‌ها نوع خود را خواهند داشت که پیش از بررسی نوع یک عبارت مانند E در جدول نمادها ذخیره شده است.

۱۴-۴-۲ تعریف هدایت شده 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*

۱۴-۴-۳ تعریف هدایت شده با نحو برای انتساب نوع به یک دستور (Statement)

PRODUCTION	SEMANTIC RULE
$S \rightarrow$ 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.

۴-۴-۱۴ تعریف هدایت شده با نحو برای انتساب نوع به یک تابع (Function)

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

۵-۱۴ هم‌ارزی نوع‌ها

برای زبان‌هایی که دارای نام‌های نوع هستند و یا زیرنوع (sub-type) را ممکن می‌دانند، باید تعریف کنیم که چه زمانی دو نوع با هم **هم‌ارز** هستند.

هم‌ارزی نوع‌ها می‌تواند با دو گونه تعریف بیان شود:

- **هم‌ارزی ساختاری (Structural Equivalence)**
  باید عبارت تعریف نوع را با یک گراف جهت‌دار نمایش دهیم.
  دو نوع با هم هم‌ارز هستند اگر و فقط اگر ساختارهای آنها (گراف‌های آنها) یکسان باشد.
  (این کار برای کامپایلر دشوار است، مثال: زبان C).
- **هم‌ارزی اسمی (Name Equivalence)**
  دو نوع با هم هم‌ارز هستند اگر و فقط اگر نام آنها یکسان باشد.
  این کار برای کامپایلر ساده است، اما موجب می‌شود کدنویسی دشوار شود.

مثال

متغیرهای X و Z هم‌ارزی نوع اسمی دارند و هر سه متغیر X,Y,Z هم‌ارزی ساختاری دارند:

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

۶-۱۴ تبدیل نوع

نوع $x + i$ چه خواهد بود؟ اگر:

۱) x از نوع $real$ باشد.

۲) i از نوع int باشد.

۳) دستورالعمل‌های مختلف ماشین برای عمل بر روی اعداد حقیقی و صحیح استفاده شود.

پاسخ، وابسته به زبان خواهد بود. در هر صورت، باید قواعد مشخصی برای تبدیل نوع توسط کامپایلر به کار گرفته شود تا نوع یکی از عملوندهای $+$ تغییر کند.

- بررسی‌گر نوع در یک کامپایلر می‌تواند این عملگرهای تبدیل را درکد میانی درج کند.
- تبدیل نوع ضمنی از این دست، Coercion نام دارد.

۱۴-۶-۱ تبدیل نوع ضمنی در عبارت‌ها

به عنوان نمونه، ترجمه‌ی هدایت شده با نحو برای تبدیل نوع ضمنی از صحیح به حقیقی برای یک عمل محاسباتی کلی op به صورت زیر است:

PRODUCTION	SEMANTIC RULE
$E \rightarrow$ num	$E.type := int$
$E \rightarrow$ num.num	$E.type := real$
$E \rightarrow$ id	$E.type := lookup($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*