مقدمه ای بر

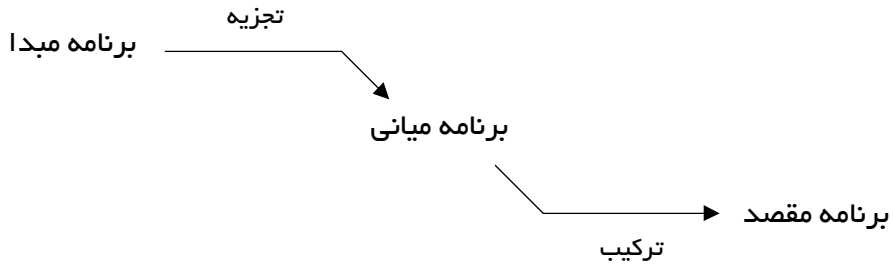
عملکرد کامپایلر

تهیه کننده: سید مصطفی رضاتوفیق

منبع اصلی:
Compilers, Principles, Techniques, and Tools
نویسنده: Alfred V. Aho
مترجم: مهدی آصفی

به نام خدا

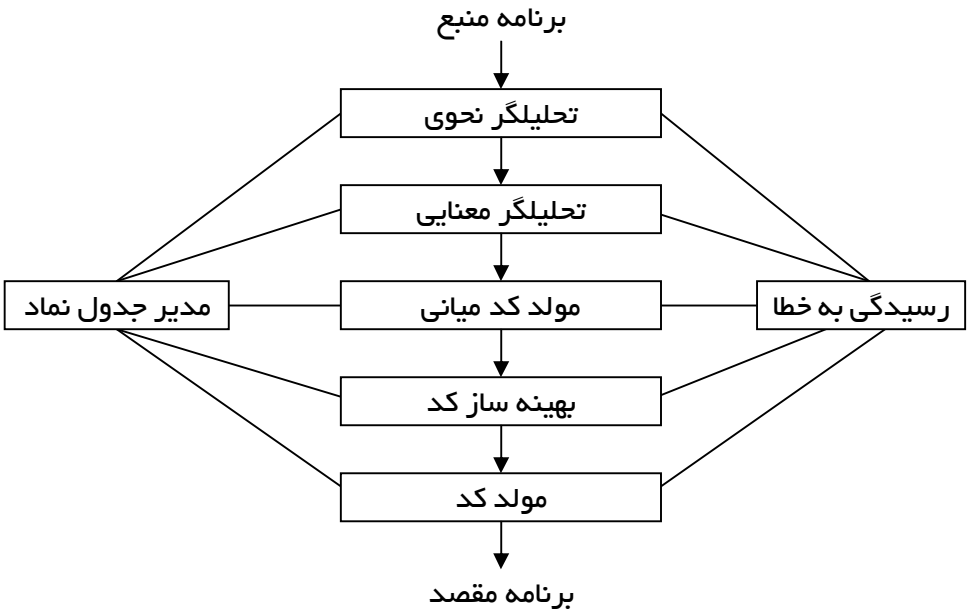
مدل تجزیه-ترکیب در عمل کامپایل



مراحل تجزیه یک برنامه منبع توسط کامپایلر

- تجزیه لغوی (خطی): رشته کاراکترهای برنامه منبع به گروهی از توکن ها (مجموعه ای از کاراکترهای معنی دار) تقسیم می شود.
- تجزیه نحوی (سلسله مراتبی): طبقه بندی توکن ها به بخش های گرامری.
- تجزیه معنایی: اطمینان از اینکه اجزا برنامه در کنار هم معنی صحیح دارند. یک بخش مهم از این تحلیل، کنترل نوع هاست.

فازهای یک کامپایلر (تجزیه و ترکیب)



Front end و Back end

فازهای کامپایلر (در پیاده سازی) به دو گروه طبقه بندی می شوند. Front end شامل فازها یا بخش هایی است که به زبان منبع وابسته اند. این بخش شامل تحلیل های لغوی، نحوی، ایجاد جدول نماد، تحلیل معنایی، بخش اصلی بهینه سازی کد و تولید کد میانی است.

Back end بخش های مربوط به ماشین مقصد را در بر دارد که به زبان میانی وابسته اند. در این بخش قسمتهایی از فازهای بهینه سازی و تولید کد و اعمال مربوط به خطاها و جدول های نماد را می توان یافت.

می توان به منظور ساخت کامپایلر برای یک ماشین با استفاده از کامپایلری مربوط به ماشینی متفاوت (با زبان منبع مشابه) از قسمت Front end آن استفاده نموده و در اینجا فقط نیاز به طراحی یک Back end جدید می باشد.

گرامر مستقل از متن (BNF)

یک زبان برنامه سازی را می توان با شرح چگونگی برنامه های آن (قواعد زبان) و مفهوم برنامه ها (معنای زبان) تعریف نمود. برای مشخص کردن قواعد یک زبان، از گرامر های مستقل از متن و یا BNF استفاده می کنیم. این گرامر ها علاوه بر مشخص کردن قاعده زبان، به عنوان راهنما در تبدیل برنامه ها کاربرد دارند.

اجزاء یک گرامر مستقل از متن (BNF)

- مجموعه پایانه ها (Terminals)
- مجموعه غیر پایانه ها (Variables)
- مجموعه تولیدات (Products)
- یک عنصر غیر پایانه به نام سمبل شروع (Start)

یک کلمه (W) بر مجموعه S دنباله ای است متناهی از عناصر آن.

می نویسیم W => W' اگر بتوان با استفاده از یک تولید W' را از W به دست آورد.

می نویسیم W =>=> W' اگر بتوان با استفاده از دنباله ای متناهی از تولیدات، W' را از W به دست آورد.

زبان G که با L(G) نموده می شود، تمام کلماتی در پایانه هاست که بتوان آنها را از سمبل شروع به دست آورد:

$L(G) = \{W: S =>=> W\}$

مثال:

$V = \{A, B, S\}$
$T = \{a, b\}$
$G = \{S \rightarrow AB, A \rightarrow Aa, B \rightarrow Bb, A \rightarrow a, B \rightarrow b\}$
$L(G) = a^n b^m$

مثلا a^2b^4 یکی از کلمات L(G) است زیرا: $S =>=> a^2b^4$.

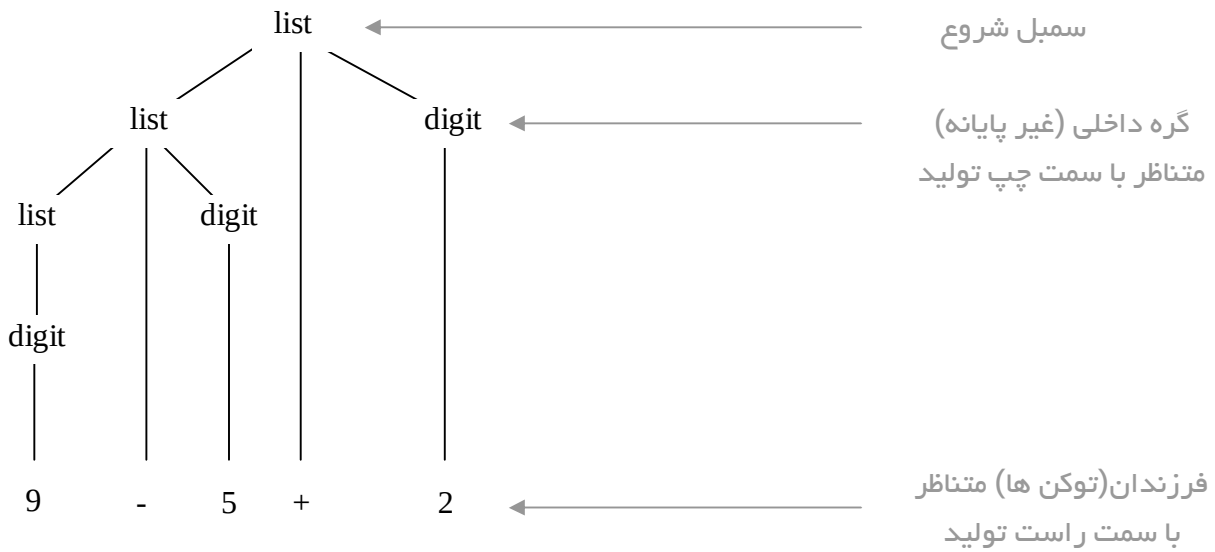
درخت های تجزیه

یک درخت تجزیه نحوه ایجاد یک رشته توسط سمبل شروع در گرامر را به صورت گرافیکی نشان می دهد. پروسه یافتن یک درخت تجزیه برای رشته داده شده از توکن ها، تجزیه آن رشته نامیده می شود.

مثال. گرامر زیر را در نظر می گیریم:

list → list + digit | list – digit | digit
digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

درخت تجزیه مربوط به عبارت 9 – 5 + 2 متناظر با گرامر فوق به صورت زیر است:



ابهام

بعضی گرامر ها می توانند بیشتر از یک درخت تجزیه تولید کننده رشته توکن های مشخصی را داشته باشند. چنین گرامر هایی مبهم اند.

گرامر زیر مبهم است:

num → num + num | num – num | digit
digit → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

زیرا:

num => **num – num** => num – num + num => 9 – 5 + 2
num => **num + num** => num – num + num => 9 – 5 + 2

شرکت پذیری عملگرها

گوییم عملگر + شرکت پذیر از چپ است وقتی عملوندی با علامت جمع در دو طرف آن، توسط عملگر سمت چپ آن تحت تاثیر قرار گیرد:

9 + **5** + 2

جمع، تفریق، ضرب و تقسیم شرکت پذیری از چپ دارند. توان داری شرکت پذیری راست است.

عملگر تخصیص مقدار (=) در C شرکت پذیری راست دارد:

right → letter = right | letter
letter → a | b | ... | z

درخت تجزیه برای شرکت پذیری چپ/راست، از چپ/راست به پایین رشد می کند.

پیاده سازی تقدم عملگر ها

expr → expr + term | expr – term | term
term → term * factor | term / factor | factor
factor → digit | (expr)

ترجمه نحوی

برای ترجمه ساختار یک زبان برنامه سازی، کامپایلر ممکن است نیاز به کنترل بسیاری از مقادیر درون مولد کد مربوط به ساختار (مانند نوع ساختار، محل اولین دستورالعمل درون کد مقصد و ...) داشته باشد. در این بخش یک روش فرموله شده را به نام تعریف نحوی، برای مشخص کردن تبدیل ساختارهای زبان برنامه سازی معرفی می کنیم.

تعریف نحوی

تعریف نحوی برای مشخص کردن ساختمان نحوی ورودی، از یک گرامر مستقل از متن استفاده می کند. هر سمبل گرامر مجموعه ای از صفات دارد و هر تولید مجموعه ای از قوانین معنایی برای محاسبه مقادیر صفات سمبل ظاهر شده در تولید.

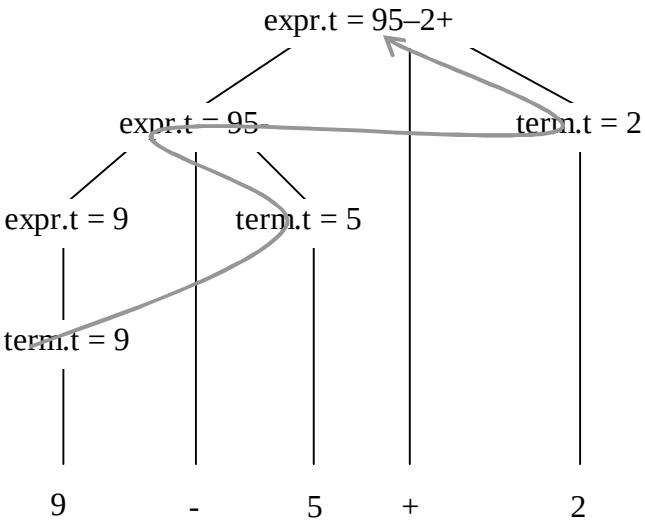
توضیح	قانون معنایی	تولید
عبارت پسوندی معادل تولید	expr.t := expr1.t || term.t || '+'	expr → expr1 + term
عبارت پسوندی معادل تولید	expr.t := expr1.t || term.t || '-'	expr → expr1 – term
	expr.t := term.t	expr → term
	term.t := '0'	term → 0
	term.t:= '1'	term → 1
	…	…
	term.t := '9'	term → 9

صفات ترکیبی

یک صفت، ترکیبی خوانده می شود اگر مقدار آن در یک گره درخت تجزیه توسط مقادیر فرزند آن گره تشخیص داده شود. این صفات دارای این حسن هستند که می توان آنها را در طی یک پیمایش پایین به بالا در درخت تجزیه مورد ارزیابی قرار داد.

درخت تجزیه مربوط به تولیدات جدول فوق در زیر آمده است:

پیکان، مسیر پیمایش عمقی درخت را برای ارزیابی صفت ترکیبی t نشان می دهد

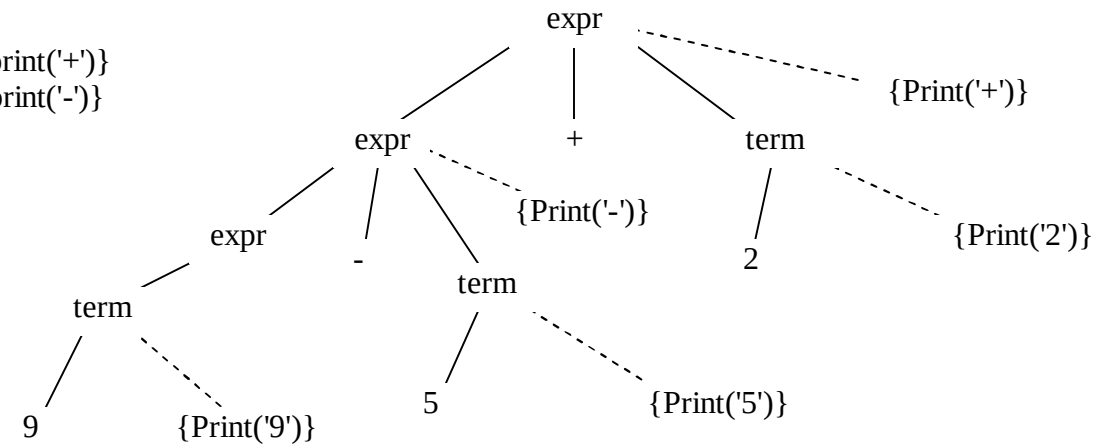


رویه های ترجمه

یک رویه ترجمه، گرامری مستقل از متن است که در آن بخش های منفصل برنامه که جریان معنایی نام دارند در سمت راست تولیدات جای می گیرند. در هنگام رسم درخت تجزیه برای یک رویه تبدیل، با ایجاد فرزندی خارجی برای آن که با خط چین به گره آن متصل شده است جریان را مشخص می نماییم.

اگر ترتیب غیر پایانه های سمت راست تولید در قانون معنایی حفظ شود، می گوییم تعریف نحوی ساده است. تعاریف نحوی ساده را می توان با کمک رویه تبدیل به نحوی که جریان ها، رشته هایی اضافه را به ترتیب ظاهر شده در تعریف چاپ نمایند، پیاده سازی نمود.

expr → expr + term {print('+')}
expr → expr + term {print('-')}
expr → term
term → 0 {print('0')}
term → 1 {print('1')}
…
term → 9 {print('9')}



نکته: در یک رویه تبدیل ساده (که از یک تعریف نحوی ساده حاصل می شود) جریان ها به ترتیب چپ به راست اجرا می شوند. بنابر این برای پیاده سازی یک رویه تبدیل ساده می توانیم جریان های معنایی را در حین تجزیه اجرا کنیم و ایجاد درخت تجزیه در همه موارد ضروری نیست.

عمل تجزیه پیشگویی پذیر

عمل تجزیه بازگشتی تسلسلی (recursive descent) یک روش بالا به پایین (در روش بالا به پایین، درخت تجزیه از ریشه به سمت پایین تولید می شود) از تحلیل نحوی است که مجموعه ای از روال های بازگشتی را برای پردازش ورودی تعریف می کند. متناظر با هر غیر پایانه از گرامر، یک روال وجود دارد.

فرمی خاص از تجزیه بازگشتی تسلسلی به نام تجزیه پیشگویی پذیر را در نظر می گیریم که در آن سمبل lookahead به صورت غیر مبهم روالی انتخاب شده برای هر غیر پایانه را مشخص می نماید. ترتیب روال های فراخوانی شده در پردازش ورودی به صورت شفاف یک درخت تجزیه را برای ورودی مشخص می کند.

هر روال تصمیم می گیرد که کدام تولید توسط جستجوی سمبل lookahead به کار می رود. تولید با سمت راست a به کار می رود اگر سمبل lookahead در First(a) باشد. First(a) مجموعه اولین عناصر سمت چپ در تولیدات a است.

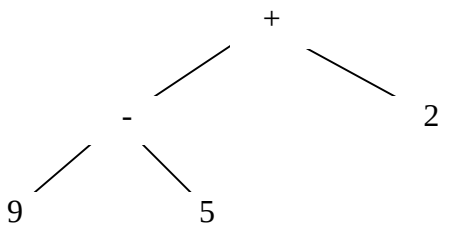
type → simple | array [simple] of type First(type) = {integer, char, num, array}
simple → integer | char | num .. num First(simple) = {integer, char, num}

نکته: مجموعه های First را وقتی A → a و A → b وجود داشته باشند در نظر می گیریم. تجزیه بازگشتی تسلسلی بدون عمل بازگشت (back-tracking) نیاز به آن دارد که First(a) و First(b) از هم جدا باشند.

ساختمان مجرد (Abstract) و ذاتی (Concrete)

در درخت نحوی مجرد، هر گره یک عملگر و فرزندان گره عملوندها را نمایش می دهند. در مقابل، درخت تجزیه، درخت نحوی ذاتی نامیده شده و گرامر مشخص شده توسط آن ساختمان، جمله ذاتی گرامر نامیده می شود.

درخت نحوی عبارت 9 – 5 + 2 که قبلا ذکر شد به این صورت است:



چپ بازگشتی

تولیدات expr در زیر چپ بازگشتی اند و باعث می شوند روال expr در حلقه بینهایت بیفتد:

expr → expr + term | expr – term | term
term → 0 | 1 | … | 9

معادل تولیدات فوق با حذف چپ بازگشتی در زیر آمده است (E: پوچ):

expr → term rest
rest → + expr | - expr | E
term → 0 | 1 | … | 9

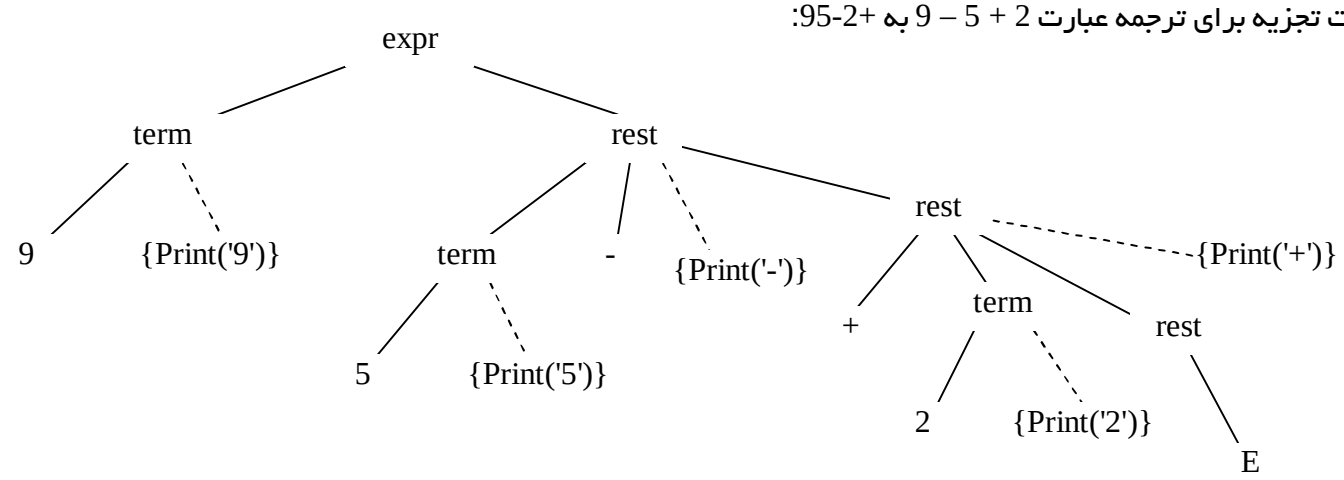
اشکال تولیدات فوق این است که در آن عملوند های مربوط به عملگرها از تولیدات مشخص نمی شوند. به علاوه نمی توان فرم پسوندی را با این روش به دست آورد. با توجه به اینکه فرم پسوندی عبارت 9 - 5 + 2 به صورت 95-2+ است:

rest → - expr {rest.t := '-' || expr.t} ➔ 9 – 5 + 2 (فرم میانوندی به دست می آید)
rest → - expr {rest.t := expr.t || '-'} ➔ 952+- (غلط است)

با استفاده از جریان ها، مشکل برطرف می شود:

expr → term rest
rest → + term {print('+')} rest | - term {print('-')} rest | E
term → 0 {print('0')}
…

درخت تجزیه برای ترجمه عبارت 2 + 5 – 9 به +2-95:



روال های بازگشتی مربوط به گرامر قبل:

```
expr()
{
	term(); rest();
}
rest()
{
	if (lookahead == '+') {
		match('+'); term(); putchar('+'); rest();
	}
	else if (lookahead) == '-') {
		match('-'); term(); putchar('-'); rest();
	}
	else ;
}
term()
{
	if (isdigit(lookahead) {
		putchar(lookahead); match(lookahead);
	}
	else  error();
}
```

فرم غیر بازگشتی روال های فوق به صورت زیر است:

```
expr()
{
	term();
	while (1)
		if (lookahead == '+')
			match('+'); term(); putchar('+');
		}
		else if (lookahead == '-') {
			match('-'); term(); putchar('-');
		}
		else break;
}
```