

به نام خدا

طراحی کامپایلر

آرش شفیعی



# تحليل معنایی

- در این فصل در مورد ترجمه زبان از طریق انتساب معانی به قوانین گرامرهای مستقل از متن صحبت خواهیم کرد. به این نوع ترجمه، ترجمه نحوی<sup>1</sup> گفته می‌شود.
- از این روش برای تحلیل معنایی و بررسی نوع و همچنین تولید کدمیانی استفاده می‌شود.
- در این روش برای هریک از نمادهای گرامر یک یا چند ویژگی<sup>2</sup> (صفت) در نظر گرفته می‌شود.
- در ترجمه نحوی مقادیر ویژگی‌های نمادهای گرامر با نسبت دادن قوانین معنایی به قوانین تولید گرامر محاسبه می‌شوند.

---

<sup>1</sup> syntax-directed translation

<sup>2</sup> attribute

- برای مثال، یک مترجم نحوی برای تبدیل عبارات میانوندی به عبارت پسوندی یک قانون معنایی به صورت زیر برای یکی از قوانین تولید دارد.

PRODUCTION

$$E \rightarrow E_1 + T$$

SEMANTIC RULE

$$E.code = E_1.code \parallel T.code \parallel '+'$$

- در این قانون دو متغیر به نام E و T وجود دارد. برای متمایز کردن متغیر E در سمت چپ و راست قانون، از  $E_1$  در بدنه قانون استفاده می‌کنیم.
- هر دو متغیر E و T یک ویژگی از نوع رشته به نام code دارند. در قانون معنایی مذکور، ویژگی code از متغیر E با الحاق  $E_1.code$  و T.code و کاراکتر + به دست می‌آید.
- پس از تشکیل دادن درخت تجزیه، می‌توان مقادیر ویژگی‌های رئوس درخت را با پیمایش درخت محاسبه کرد.

- فرض کنید برای قانون تولید  $E \rightarrow E_1 + T$  قانون معنایی  $E.code = E_1.code || T.code || '-'$  را تعریف می‌کنیم.
- همچنین برای قانون تولید  $E \rightarrow T$  قانون معنایی  $E.code = T.code$  و برای قوانین تولید  $T \rightarrow id$  قانون معنایی  $T.code = id$  را تعریف می‌کنیم.
- درخت تجزیه‌ای برای عبارت  $x - y + z$  رسم کنید و مقدار ویژگی `code` را برای ریشه درخت محاسبه کنید.

- گرامر نحوی-معنایی<sup>1</sup> ، یک گرامر مستقل از متن همراه با قوانین معنایی و ویژگی برای متغیرهاست.
- ویژگی‌ها به نمادهای گرامر نسبت داده می‌شوند و قوانین معنایی به قوانین تولید گرامر نسبت داده می‌شوند.
- اگر  $X$  یک نماد (متغیر یا ترمینال) و  $a$  یک ویژگی باشد، آنگاه در درخت تجزیه در رأسی که نماد  $X$  را شامل می‌شود، می‌نویسیم  $X.a$  که نشان دهنده ویژگی  $a$  از نماد  $X$  است.
- ویژگی‌ها می‌توانند هر نوعی از جمله اعداد، رشته‌ها، اشاره‌گر به جداول و غیره باشند.
- یک ویژگی می‌تواند از نوع رشته باشد و مقدار آن یک کد (برای مثال کد میانی) باشد.

---

<sup>1</sup> Syntax-directed definition (SDD)

## ویژگی‌های به ارث برده و ترکیب شده

- دو نوع ویژگی برای متغیرهای یک گرامر می‌توانند وجود داشته باشند : ویژگی‌های ترکیب شده<sup>1</sup> و ویژگی‌های به ارث برده شده<sup>2</sup>.

۱. یک ویژگی ترکیب شده (ساخته شده) برای متغیر  $A$  در رأس  $N$  از درخت تجزیه با استفاده از قوانین معنایی متناظر با قوانین تولید متغیر  $A$  به دست می‌آید. به عبارت دیگر یک ویژگی ترکیب شده در رأس  $N$  با استفاده از مقادیر ویژگی‌های فرزندان رأس  $N$  و خود رأس  $N$  محاسبه می‌شود.

۲. یک ویژگی به ارث برده شده برای متغیر  $B$  در رأس  $N$  از درخت تجزیه با استفاده از قوانین معنایی متناظر با قوانین تولیدی که  $B$  در بدنه آنهاست به دست می‌آید. به عبارت دیگر یک ویژگی به ارث برده شده در رأس  $N$  با استفاده از مقادیر ویژگی‌های پدر و همزادهای رأس  $N$  و خود رأس  $N$  محاسبه می‌شود.

---

<sup>1</sup> synthesized attribute

<sup>2</sup> inherited attribute

## ویژگی‌های به ارث برده و ترکیب شده

- ویژگی‌های ترمینال‌ها به طور مستقیم از تحلیل‌گر لغوی دریافت می‌شوند. به عبارت دیگر برای مقداردهی ویژگی‌های ترمینال‌ها قانون معنایی وجود ندارد. بنابراین ترمینال‌ها ویژگی ترکیب شده دارند و نه ویژگی به ارث برده شده.



## ویژگی‌های به ارث برده و ترکیب شده

- مثال : گرامر نحوی-معنایی (SDD) زیر را در نظر بگیرید.

PRODUCTION	SEMANTIC RULES
1) $L \rightarrow E \mathbf{n}$	$L.val = E.val$
2) $E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
3) $E \rightarrow T$	$E.val = T.val$
4) $T \rightarrow T_1 * F$	$T.val = T_1.val \times F.val$
5) $T \rightarrow F$	$T.val = F.val$
6) $F \rightarrow ( E )$	$F.val = E.val$
7) $F \rightarrow \mathbf{digit}$	$F.val = \mathbf{digit.lexval}$

## ویژگی‌های به ارث برده و ترکیب شده

- این گرامر عبارات محاسباتی با عملگرهای  $+$  و  $*$  و پرانتز تولید می‌کند. هر عبارت با کاراکتر  $n$  خاتمه پیدا می‌کند.
- هر یک از متغیرهای گرامر یک ویژگی ترکیب‌شده به نام `val` دارند. همچنین ترمینال `digit` یک ویژگی ترکیب‌شده به نام `lexval` دارد که عددی است که توسط تحلیل‌گر لغوی تولید شده است.
- برای هر یک از قوانین تولید، ویژگی متغیر قانون با استفاده از ویژگی‌های متغیرهای بدنه قانون به دست می‌آید و بنابراین ویژگی `val` یک ویژگی ترکیب‌شده است.

## ویژگی‌های به ارث برده و ترکیب شده

- اگر یک گرامر نحوی-معنایی (SDD) تنها شامل ویژگی‌های ترکیب شده باشد به آن ترجمه ترکیب شده یا گرامر نحوی-معنایی ترکیب شده یا گرامر نحوی-معنایی S<sup>1</sup> گفته می‌شود.
- در یک گرامر نحوی-معنایی ترکیب شده، در هر قانون تولید، ویژگی‌های متغیرهای قوانین از طریق ویژگی‌های متغیرهای بدنه قانون محاسبه می‌شوند.

---

<sup>1</sup> S-attributed (synthesized) SDD

## ویژگی‌های به ارث برده و ترکیب شده

- اگر یک قانون معنایی قادر باشد جدول علائم را دستکاری کند، می‌گوییم ترجمه نحوی-معنایی دارای اثرات جانبی<sup>1</sup> است. اگر یک گرامر نحوی-معنایی دارای اثرات جانبی نباشد به آن یک گرامر صفت<sup>2</sup> گفته می‌شود. در یک گرامر صفت مقادیر ویژگی‌ها تنها از طریق ویژگی‌های نمادهای دیگر محاسبه می‌شوند.

---

<sup>1</sup> side effect

<sup>2</sup> attribute grammar

- برای محاسبه ویژگی‌ها، ابتدا درخت تجزیه ساخته شده، سپس توسط قوانین معنایی مقادیر ویژگی‌ها محاسبه می‌شوند.
- درخت تجزیه‌ای که در آن ویژگی‌ها و مقادیر آنها نشان داده شده باشد درخت تجزیه حاشیه نویسی شده<sup>1</sup> نامیده می‌شود.
- قبل از اینکه بتوانیم مقدار یک ویژگی را در یک رأس محاسبه کنیم، باید مقدار همه ویژگی‌هایی را که آن ویژگی به آنها بستگی دارد محاسبه کرده باشیم. برای مثال اگر یک ویژگی ترکیبی باشد، باید همه ویژگی‌های فرزندان یک رأس را قبل از محاسبه ویژگی آن رأس محاسبه کنیم.

---

<sup>1</sup> annotated parse tree

## ارزیابی گرامر نحوی-معنایی

- در صورتی که ویژگی‌ها ترکیبی باشند می‌توانیم درخت تجزیه را از پایین به بالا ارزیابی کنیم و ویژگی‌ها را از پایین به بالا محاسبه کنیم.
- اگر در یک گرامر نحوی-معنایی ویژگی‌های به ارث برده شده و ترکیب شده وجود داشته باشد، هیچ تضمینی وجود ندارد که با پیمایش درخت تجزیه از پایین به بالا یا از بالا به پایین همه ویژگی‌ها محاسبه شوند.
- برای مثال فرض کنید دو متغیر  $A$  و  $B$  در یک گرامر دارای دو ویژگی ترکیب شده و به ارث برده شده باشند. گرامر نحوی-معنایی به صورت زیر خواهد بود.

PRODUCTION

$$A \rightarrow B$$

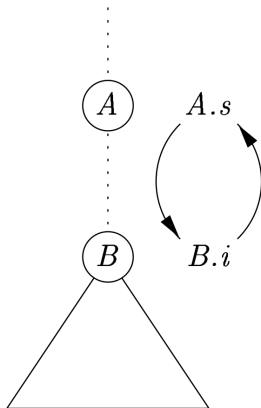
SEMANTIC RULES

$$A.s = B.i;$$

$$B.i = A.s + 1$$

## ارزیابی گرامر نحوی-معنایی

- این ویژگی‌ها دارای وابستگی در یک دور<sup>1</sup> هستند.
- این دور در شکل زیر نشان داده شده است.



---

<sup>1</sup> circular dependency

- مسئله پیدا کردن دور در یک درخت تجزیه حاشیه گذاری شده یک مسئله ان پی کامل است و بنابراین تحلیل درخت در زمان چند جمله‌ای امکان پذیر نیست. برای دسته‌ای از گرامرهای نحوی-معنایی می‌توان تضمین کرد که با ارزیابی آنها از بالا به پایین و یا از پایین به بالا همه ویژگی‌ها محاسبه خواهند شد که بعدها به آن می‌پردازیم.



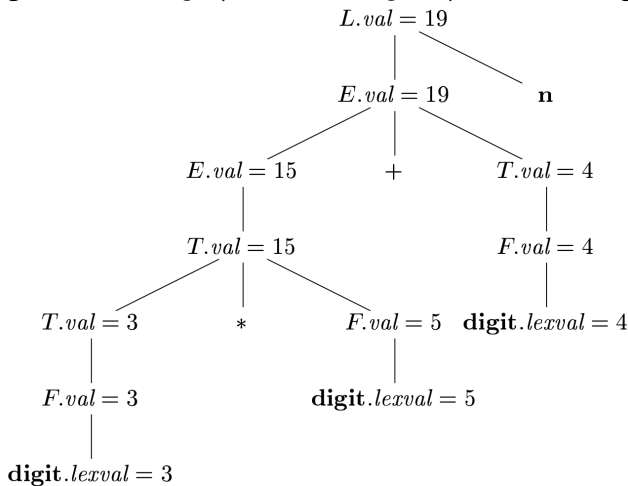
## ویژگی‌های به ارث برده و ترکیب شده

- مثال : درخت تجزیه حاشیه‌گذاری شده برای ورودی  $3 * 5 + 4 n$  را با استفاده از گرامر نحوی-معنایی زیر رسم کنید.

PRODUCTION	SEMANTIC RULES
1) $L \rightarrow E \mathbf{n}$	$L.val = E.val$
2) $E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
3) $E \rightarrow T$	$E.val = T.val$
4) $T \rightarrow T_1 * F$	$T.val = T_1.val \times F.val$
5) $T \rightarrow F$	$T.val = F.val$
6) $F \rightarrow ( E )$	$F.val = E.val$
7) $F \rightarrow \mathbf{digit}$	$F.val = \mathbf{digit.lexval}$

## ارزیابی گرامر نحوی-معنایی

- مثال : شکل زیر درخت تجزیه حاشیه گذاری شده برای ورودی  $3 * 5 + 4$  n را نشان می دهد. هریک از متغیرها دارای ویژگی val است که با پیمایش درخت تجزیه از پایین به بالا محاسبه می شوند.



- در مثال بعد نشان می‌دهیم چگونه ویژگی‌های به ارث برده شده استفاده می‌شوند.

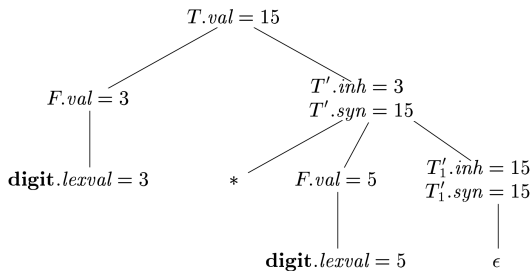
## ارزیابی گرامر نحوی-معنایی

- مثال : گرامر نحوی-معنایی (SDD) زیر عبارات محاسباتی مانند  $3 * 5 + 7$  و  $3 * 5$  را محاسبه می‌کند. تجزیه کننده بالا به پایین برای ورودی  $3 * 5$  با قانون تولید  $T \rightarrow FT'$  آغاز می‌کند. متغیر  $F$  عدد 3 و متغیر  $T'$  عملگر  $*$  و عدد 5 را تولید می‌کنند. متغیر  $F$  در زیر درخت سمت چپ و  $T'$  در زیر درخت سمت راست متغیر  $T$  قرار می‌گیرند و بنابراین از یک ویژگی به ارث برده شده برای انتقال عملوند سمت چپ به عملگر استفاده می‌شود.

PRODUCTION	SEMANTIC RULES
1) $T \rightarrow F T'$	$T'.inh = F.val$ $T.val = T'.syn$
2) $T' \rightarrow * F T'_1$	$T'_1.inh = T'.inh \times F.val$ $T'.syn = T'_1.syn$
3) $T' \rightarrow \epsilon$	$T'.syn = T'.inh$
4) $F \rightarrow \mathbf{digit}$	$F.val = \mathbf{digit.lexval}$

- هریک از متغیرهای  $T$  و  $F$  یک ویژگی ترکیب شده به نام  $val$  دارند و ترمینال  $digit$  یک ویژگی ترکیب شده به نام  $lexval$  دارد.
- متغیر  $T'$  دو ویژگی دارد. یک ویژگی به ارث برده شده به نام  $inh$  و یک ویژگی ترکیب شده به نام  $syn$ .
- متغیر  $T'$  از قانون تولید  $T' \rightarrow *FT'_1$  عملوند سمت چپ عملگر  $*$  را به ارث می‌برد.
- فرض کنید عبارت  $x * y * z$  داده شده است. ریشه زیر درخت حاوی  $*y * z$  عبارت  $x$  را به ارث می‌برد. ریشه زیر درخت حاوی  $*z$  مقدار  $x * y$  را به ارث می‌برد و این روند ادامه پیدا می‌کند تا جایی که متغیر دیگری وجود نداشته باشد.

- درخت تجزیه حاشیه‌گذاری شده برای عبارت  $3 * 5$  در زیر آمده است.



PRODUCTION	SEMANTIC RULES
1) $T \rightarrow F T'$	$T'.inh = F.val$ $T.val = T'.syn$
2) $T' \rightarrow * F T'_1$	$T'_1.inh = T'.inh \times F.val$ $T'.syn = T'_1.syn$
3) $T' \rightarrow \epsilon$	$T'.syn = T'.inh$
4) $F \rightarrow \text{digit}$	$F.val = \text{digit.lexval}$

## ارزیابی گرامر نحوی-معنایی

- توجه کنید که عملگر ضرب (همانند عملگر جمع و تفریق) وابستگی چپ دارد و بنابراین نمی‌توانیم درخت تجزیه را از پایین به بالا پیمایش کنیم و قوانین معنایی را تنها با ویژگی‌های ترکیب‌شده بنویسیم.
- اگر وابستگی راست داشتیم می‌توانستیم به جای قوانین معنایی زیر، برای قانون  $T \rightarrow FT'$  قانون معنایی  $T.val = F.val * T'.val$  و برای قانون  $T' \rightarrow *FT'_1$  قانون معنایی  $T'.val = F.val * T'_1.val$  و برای قانون  $T' \rightarrow \epsilon$  قانون معنایی  $T'.val = 1$  را تعریف کنیم.

PRODUCTION	SEMANTIC RULES
1) $T \rightarrow F T'$	$T'.inh = F.val$ $T.val = T'.syn$
2) $T' \rightarrow * F T'_1$	$T'_1.inh = T'.inh \times F.val$ $T'.syn = T'_1.syn$
3) $T' \rightarrow \epsilon$	$T'.syn = T'.inh$
4) $F \rightarrow \mathbf{digit}$	$F.val = \mathbf{digit.lexval}$

- گراف‌های وابستگی<sup>1</sup> ابزاری برای تعیین ترتیب ارزیابی ویژگی‌ها در درخت تجزیه هستند.
- گراف وابستگی نشان می‌دهد مقادیر ویژگی‌ها چگونه در یک درخت تجزیه انتشار پیدا می‌کنند. یک یال جهت‌دار از یک ویژگی به یک ویژگی دیگر بدین معنی است که مقدار ویژگی اول پیش از مقدار ویژگی دوم باید محاسبه شود.

---

<sup>1</sup> dependency graph



# گراف وابستگی

- یک گراف وابستگی به صورت زیر تشکیل می‌شود.

۱. به ازای هر رأس  $X$  در درخت تجزیه یک رأس  $X$  در گراف وابستگی ایجاد می‌شود.
۲. فرض کنید یک قانون معنایی به قانون تولید  $p$  نسبت داده شده باشد که در آن ویژگی ترکیب شده  $A.b$  با استفاده از مقدار  $X.c$  محاسبه می‌شود (البته مقدار  $A.b$  می‌تواند از مقدار ویژگی‌های دیگری غیر از  $X.c$  نیز استفاده کند). در این صورت در گراف وابستگی یالی از  $X.c$  به  $A.b$  وجود دارد. به عبارت دیگر در هر رأس  $N$  با برچسب  $A$  که در آن قانون  $p$  اعمال شده باشد، یک یال جهت‌دار از ویژگی  $c$  در رأس فرزند  $N$  متناظر با متغیر  $X$  به ویژگی  $b$  از رأس  $N$  رسم می‌شود.
۳. فرض کنید یک قانون معنایی به قانون  $p$  نسبت داده شده باشد که در آن ویژگی به ارث برده شده  $B.c$  با استفاده از مقدار  $X.a$  محاسبه می‌شود. در این صورت در گراف وابستگی یالی از  $X.a$  به  $B.c$  وجود دارد. به عبارت دیگر برای هر رأس  $N$  با برچسب  $B$  که متناظر با یک متغیر  $B$  در بدنه قانون  $p$  باشد، یالی از ویژگی  $a$  از رأس متناظر با  $X$  به ویژگی  $c$  از رأس  $N$  رسم می‌شود. توجه کنید که رأس متناظر با  $X$  می‌تواند پدر یا یکی از همزادهای  $N$  باشد.

# گراف وابستگی

- مثال : تعاریف نحوی-معنایی زیر را در نظر بگیرید.

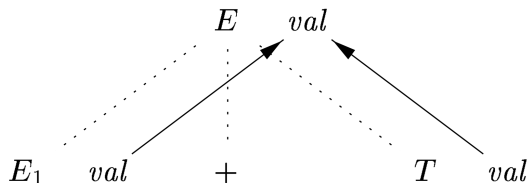
PRODUCTION

SEMANTIC RULE

$$E \rightarrow E_1 + T$$

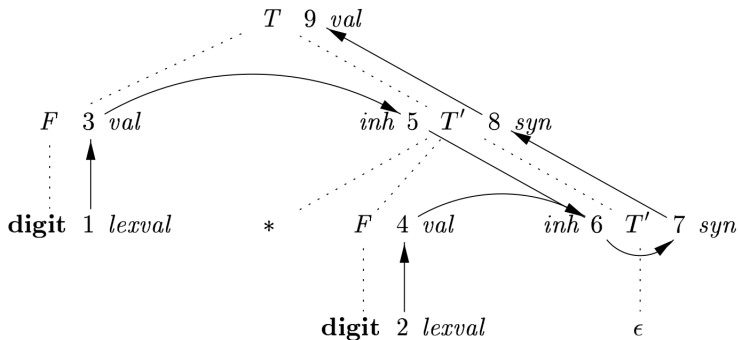
$$E.val = E_1.val + T.val$$

- در هر قسمتی از گراف تجزیه که از این قانون تولید استفاده شده باشد، گراف وابستگی به صورت زیر است.  
یال‌های درخت با نقطه چین نمایش داده شده‌اند.



# گراف وابستگی

- مثال : یک درخت تجزیه به همراه گراف وابستگی در شکل زیر نمایش داده شده است. برای محاسبه ویژگی در ریشه این درخت، ویژگی‌ها به ترتیب نشان داده شده از ۱ تا ۹ محاسبه شوند.



## ترتیب ارزیابی ویژگی‌ها

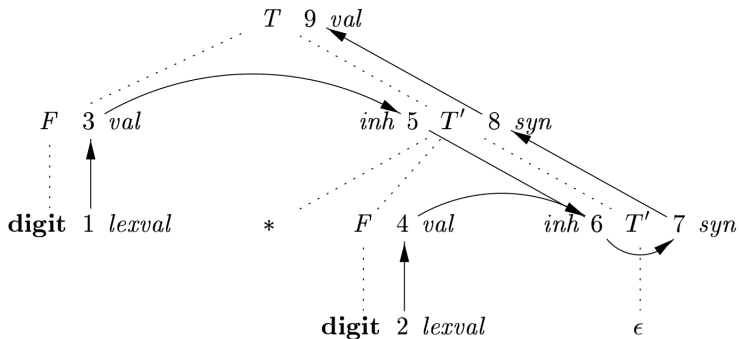
- گراف وابستگی ترتیب ارزیابی ویژگی‌ها در درخت تجزیه حاشیه‌گذاری شده را نشان می‌دهد.
- اگر گراف وابستگی یالی از رأس  $M$  به رأس  $N$  داشته باشد، آنگاه ویژگی متناظر با  $M$  باید قبل از ویژگی متناظر با  $N$  محاسبه شود.
- اگر در دنباله‌ای از رئوس به صورت  $N_1, N_2, \dots, N_k$  یال‌هایی از  $N_i$  به  $N_j$  وجود داشته باشد به طوری که  $j \in U_i$  آنگاه ویژگی‌های رئوس درخت باید به ترتیب عناصر دنباله محاسبه شوند. به ترتیبی که وابستگی‌ها در یک گراف جهت‌دار را حفظ کند، ترتیب توپولوژیکی<sup>1</sup> گراف گفته می‌شود.
- اگر در گراف دور وجود داشته باشد، هیچ ترتیب توپولوژیکی برای آن وجود ندارد و در این صورت امکان ارزیابی درخت تجزیه وجود ندارد.
- اگر در گراف وابستگی دور وجود نداشته باشد، حداقل یک ترتیب توپولوژیکی برای آن وجود دارد.

---

<sup>1</sup> topological sort

# ترتیب ارزیابی ویژگی‌ها

- درگراف وابستگی زیر دور وجود ندارد و یک ترتیب توپولوژیکی برای آن 1, 3, 5, 2, 4, 6, 7, 8, 9 است.



- به ازای یک گرامر نحوی-معنایی دلخواه، تعیین کردن اینکه وابستگی ویژگی‌ها دارای دور هستند یا خیر پیچیدگی زمانی بالایی دارد.
- معمولاً در عمل از زیر مجموعه‌ای از گرامرهای نحوی-معنایی استفاده می‌شود که تضمین می‌کنند گراف وابستگی ویژگی‌ها دارای دور نیست.
- یکی از مجموعه‌های گرامرهای نحوی-معنایی، گرامر صفت S<sup>1</sup> است.
- گرامر صفت S یک گرامر نحوی-معنایی است که در آن همه ویژگی‌ها ترکیب شده هستند.

---

<sup>1</sup> S-attributed grammar/definitions

- مثال : گرامر نحوی-معنایی زیر یک گرامر صفت S است زیرا همه ویژگی‌ها از نوع ترکیب شده هستند.

PRODUCTION	SEMANTIC RULES
1) $L \rightarrow E \mathbf{n}$	$L.val = E.val$
2) $E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
3) $E \rightarrow T$	$E.val = T.val$
4) $T \rightarrow T_1 * F$	$T.val = T_1.val \times F.val$
5) $T \rightarrow F$	$T.val = F.val$
6) $F \rightarrow ( E )$	$F.val = E.val$
7) $F \rightarrow \mathbf{digit}$	$F.val = \mathbf{digit.lexval}$

- وقتی یک گرامر نحوی-معنایی یک گرامر صفت باشد، می‌توانیم ویژگی‌های آن را در درخت تجزیه از پایین به بالا یعنی از برگ‌ها به سمت ریشه محاسبه کنیم.
- می‌توانیم از الگوریتم زیر برای ارزیابی ویژگی‌ها در درخت تجزیه استفاده کنیم.

```
postorder( $N$ ) {  
    for ( each child  $C$  of  $N$ , from the left )  $postorder(C)$ ;  
    evaluate the attributes associated with node  $N$ ;  
}
```



- گرامر صفت S می‌تواند در حین فرایند تجزیه پایین به بالا ارزیابی شود زیرا در تجزیه پایین به بالا تجزیه از برگ‌های درخت تجزیه آغاز و به ریشه منتهی می‌شود.
- بنابراین اگر از یک تجزیه‌کننده LR برای تجزیه استفاده شود، می‌توان در حین تجزیه ویژگی‌های گرامر صفت S را بدون ساختن درخت تجزیه محاسبه کرد.

- دسته دوم گرامرهای نحوی-معنایی، گرامر صفت  $L^1$  نام دارد.
- در یک گرامر صفت L وابستگی بین ویژگی‌ها در بدنه یک قانون تولید از چپ به راست است و نه از راست به چپ. بدین ترتیب این گرامر صفت تضمین می‌کند که در وابستگی‌ها دور وجود ندارد.
- در یک گرامر صفت L ویژگی‌ها می‌توانند ترکیب شده باشند.
- همچنین ویژگی‌ها می‌توانند با شروطی به شرح زیر به ارث برده باشند. فرض کنید یک قانون تولید به صورت  $A \rightarrow X_1 X_2 \dots X_n$  داشته باشیم و یک ویژگی به ارث برده مانند  $X_i.a$  وجود داشته باشد. آنگاه در قانون معنایی متناظر با این قانون تولید  $X_i.a$  می‌تواند یا از ویژگی‌های A استفاده کند و یا از ویژگی‌های  $X_1 X_2 \dots X_{i-1}$  که از قبل از آن قرار گرفته‌اند استفاده کند. همچنین برای محاسبه  $X_i.a$  می‌توان از ویژگی‌های  $X_i$  استفاده کرد به شرط آن که در محاسبه ویژگی‌های  $X_i$  دور به وجود نیاید.

---

<sup>1</sup> L-attributed grammar/definitions

- مثال : گرامر نحوی-معنایی زیر یک گرامر صفت L است.

PRODUCTION	SEMANTIC RULES
1) $T \rightarrow F T'$	$T'.inh = F.val$ $T.val = T'.syn$
2) $T' \rightarrow * F T'_1$	$T'_1.inh = T'.inh \times F.val$ $T'.syn = T'_1.syn$
3) $T' \rightarrow \epsilon$	$T'.syn = T'.inh$
4) $F \rightarrow \mathbf{digit}$	$F.val = \mathbf{digit.lexval}$

- قوانین نحوی-معنایی زیر را در این گرامر در نظر بگیرید.

PRODUCTION	SEMANTIC RULE
$T \rightarrow F T'$	$T'.inh = F.val$
$T' \rightarrow * F T'_1$	$T'_1.inh = T'.inh \times F.val$

- ویژگی به ارث برده  $T' \cdot inh$  فقط از  $F \cdot val$  استفاده می‌کند و  $F$  در سمت چپ  $T'$  در بدنه قانون تولید است.

- ویژگی  $T'_1.inh$  در قانون دوم از ویژگی‌های  $T'_1.inh$  استفاده می‌کند که در سمت چپ قانون تولید است و همچنین از  $F.val$  استفاده می‌کند و  $F$  در سمت چپ  $T'_1$  در بدنه قانون تولید قرار دارد.

- مثال : گرامر نحوی-معنایی شامل قوانین زیر نمی‌تواند گرامر صفت L باشد.

PRODUCTION	SEMANTIC RULES
$A \rightarrow B C$	$A.s = B.b;$ $B.i = f(C.c, A.s)$

- قانون معنایی اول ویژگی  $A.s$  را براساس  $B.b$  تعریف می‌کند و  $B$  در درخت تجزیه فرزند  $A$  است. پس تا اینجا گرامر می‌تواند گرامر صفت  $S$  یا  $L$  باشد.

- در قانون معنایی دوم یک ویژگی به ارث برده  $B.i$  وجود دارد پس گرامر نمی‌تواند گرامر صفت  $S$  باشد. به علاوه  $B.i$  از  $C.c$  استفاده می‌کند و  $C$  در سمت راست  $B$  قرار دارد پس گرامر نمی‌تواند یک گرامر صفت  $L$  باشد.

## قوانین معنایی با اثر جانبی محدود

- در یک گرامر نحوی-معنایی قوانین می‌توانند اثر جانبی داشته باشند. مثلاً قوانین معنایی می‌توانند جدول علائم را تغییر دهند.
- اگر یک گرامر نحوی-معنایی هیچ اثر جانبی نداشته باشد به آن گرامر صفت<sup>1</sup> گفته می‌شود.
- اگر قوانین معنایی اثر جانبی داشته باشند، این اثر جانبی نباید باعث شود که با عوض کردن ترتیب ارزیابی ویژگی‌ها نتایج متفاوت به دست آید. بنابراین اثرات جانبی باید به گونه‌ای محدود شوند که ابهام به وجود نیاورند، یعنی به ازای دو ترتیب ارزیابی دو نتیجه متفاوت به وجود نیاورند.

---

<sup>1</sup> attribute grammar

## قوانین معنایی با اثر جانبی محدود

- مثال : گرامر نحوی-معنایی زیر را در نظر بگیرید. این گرامر برای تعریف نوع متغیرها استفاده می‌شود.

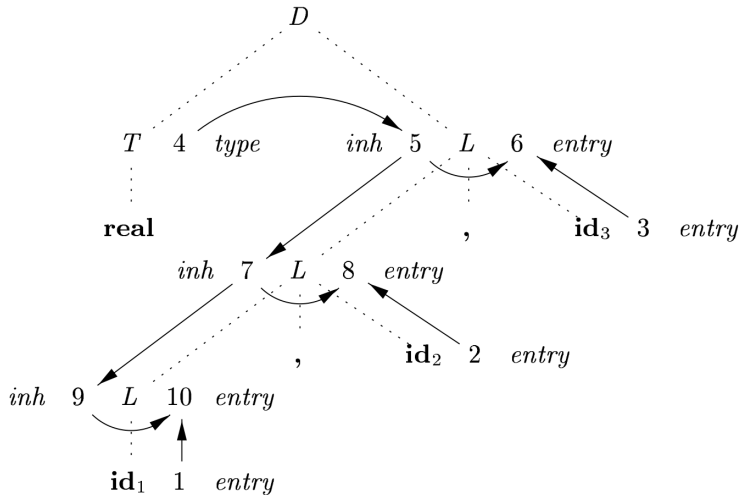
PRODUCTION	SEMANTIC RULES
1) $D \rightarrow T L$	$L.inh = T.type$
2) $T \rightarrow \mathbf{int}$	$T.type = \text{integer}$
3) $T \rightarrow \mathbf{float}$	$T.type = \text{float}$
4) $L \rightarrow L_1 , \mathbf{id}$	$L_1.inh = L.inh$ $addType(\mathbf{id}.entry, L.inh)$
5) $L \rightarrow \mathbf{id}$	$addType(\mathbf{id}.entry, L.inh)$

- در این گرامر به ازای هر متغیر، نوع متغیر در جدول علائم به روزرسانی می‌شود. اضافه کردن نوع یک متغیر بر نوع متغیرهای دیگر تأثیری ندارد، بنابراین ویژگی‌ها می‌توانند با هر ترتیبی ارزیابی و محاسبه شوند. در این گرامر بررسی نمی‌کنیم که نوع متغیر حتماً یک بار تعریف شده باشد، اما امکان تغییر گرامر برای ایجاد این بررسی وجود دارد.



# قوانین معنایی با اثر جانبی محدود

- گراف وابستگی برای عبارت `float id1, id2, id3` به صورت زیر است.



- ترجمه نحوی معمولاً برای تحلیل معنایی از جمله بررسی نوع در زبان‌های برنامه‌نویسی و همچنین تولید کدمیانی استفاده می‌شود.
- قبل از تولید کد میانی درخت نحوی توسط کامپایلر ساخته می‌شود.
- هر رأس در یک درخت نحوی یکی از ساختارهای زبان را نشان می‌دهد. برای مثال رأس درخت نحوی در عبارت  $E_1 + E_2$  با عملگر + برچسب زده می‌شود و دو فرزند آن دو عبارت  $E_1$  و  $E_2$  هستند.

- هر رأس در درخت نحوی یک شیء (با مفهوم شیء در برنامه نویسی شیءگرا) است.
- هر شیء دارای یک فیلد (ویژگی یا متغیر) به نام  $op$  است که برچسب آن رأس را نشان می دهد.
- اگر یک رأس، برگ باشد، آنگاه لغت متناظر با آن برگ را باید ذخیره کنیم. بنابراین تابع سازنده شیء به صورت  $Leaf(op, val)$  خواهد بود.
- اگر یک رأس، رأس میانی باشد، آنگاه می تواند تعدادی فرزند داشته باشد بنابراین باید اشاره گری به فرزندان آن را در شیء متناظر با آن رأس ذخیره کنیم. پس تابع سازنده به صورت  $Node(op, c_1, c_2, \dots, c_k)$  خواهد بود به طوری که  $k$  تعداد فرزندان و  $c_1$  تا  $c_k$  اشاره گرهایی به فرزندان هستند.

- مثال : گرامر صفت S زیر یک درخت نحوی برای یک گرامر ساده جهت توصیف عبارت حسابی با عملگرهای + و - می‌سازد.

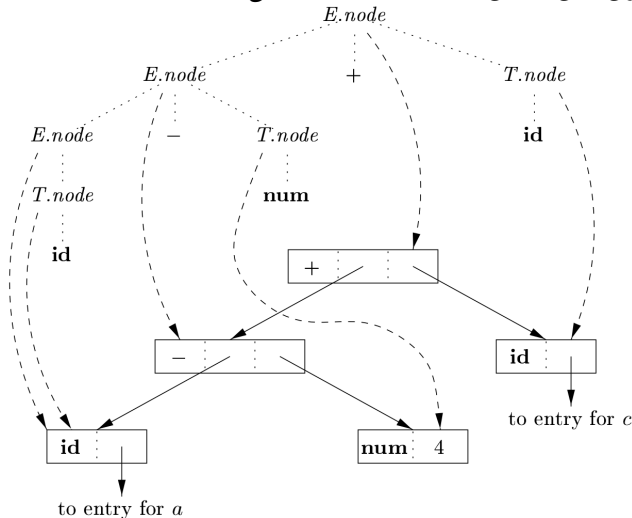
PRODUCTION	SEMANTIC RULES
1) $E \rightarrow E_1 + T$	$E.node = \mathbf{new} \text{ Node}('+', E_1.node, T.node)$
2) $E \rightarrow E_1 - T$	$E.node = \mathbf{new} \text{ Node}('-', E_1.node, T.node)$
3) $E \rightarrow T$	$E.node = T.node$
4) $T \rightarrow ( E )$	$T.node = E.node$
5) $T \rightarrow \mathbf{id}$	$T.node = \mathbf{new} \text{ Leaf}(\mathbf{id}, \mathbf{id.entry})$
6) $T \rightarrow \mathbf{num}$	$T.node = \mathbf{new} \text{ Leaf}(\mathbf{num}, \mathbf{num.val})$

- ویژگی node در این گرامر صفت اشاره‌گری به یک شیء است که رأسی را در درخت تجزیه ذخیره می‌کند.
- هر بار قانون تولید  $E \rightarrow E_1 + T$  برای تجزیه استفاده می‌شود، یک رأس با برچسب + و دو فرزند  $E_1.node$  و  $T.node$  ساخته می‌شود.

- برای قانون  $E \rightarrow T$  هیچ رأسی ساخته نمی‌شود، زیرا E.node همان T.node است. همچنین در قانون  $T \rightarrow (E)$  هیچ رأسی ساخته نمی‌شود زیرا پرانتز تنها برای دسته‌بندی عبارات استفاده می‌شود.
- قوانین ۵ و ۶ هرکدام یک ترمینال در بدنه قانون دارند، بنابراین از کلاس Leaf برای ساختن رئوس متناظر با برگ‌های درخت استفاده می‌شود.

# کاربردهای ترجمه نحوی

- شکل زیر درخت نحوی برای ورودی  $a - 4 + c$  را نشان می‌دهد.



- رئوس درخت نحوی به صورت رکوردهایی نمایش داده شده‌اند که فیلد اول مقدار `op` را نشان می‌دهد. درخت نحوی با خطوط پررنگ و درخت تجزیه با نقطه‌چین نمایش داده شده است.
- در برگ‌های درخت مقادیر `a` و `4` و `c` دیده می‌شوند که توسط کلاس `Leaf` یک رأس برای هر کدام از آنها ساخته شده است.
- فرض می‌کنیم `id.entry` به مکان شناسه در جدول علائم اشاره می‌کند و مقدار یک عدد در `num.val` ذخیره می‌شود. سه برگ درخت با قوانین معانی ۵ و ۶ ساخته می‌شوند. در این درخت نحوی از قوانین ۱ و ۲ برای تولید دو رأس برای عملگرهای جمع و تفریق استفاده شده است.



- اگر درخت تجزیه توسط یک الگوریتم عمق اول پس ترتیب<sup>1</sup> پیمایش شود، آنگاه ترتیب ساخته شدن رئوس به صورت زیر خواهد بود.

- 1)  $p_1 = \text{new Leaf}(\mathbf{id}, \text{entry-a});$
- 2)  $p_2 = \text{new Leaf}(\mathbf{num}, 4);$
- 3)  $p_3 = \text{new Node}('-', p_1, p_2);$
- 4)  $p_4 = \text{new Leaf}(\mathbf{id}, \text{entry-c});$
- 5)  $p_5 = \text{new Node}('+', p_3, p_4);$

---

<sup>1</sup> postorder depth first search traversal

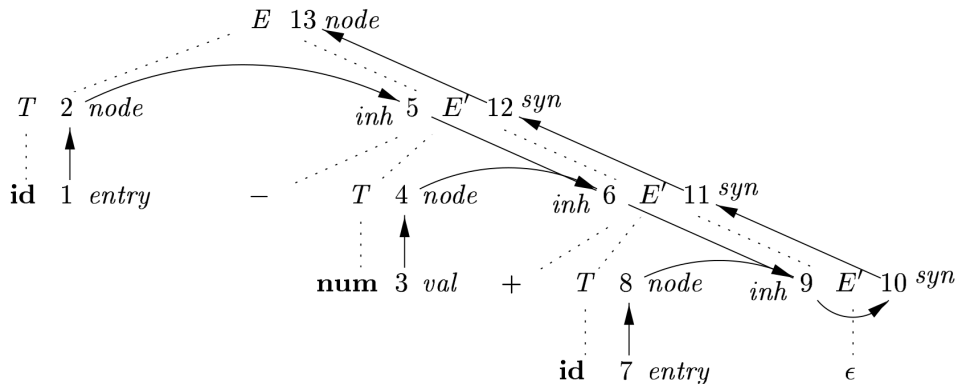
- مثال : گرامر صفت L و گرامر صفت S زیر هر دو ترجمه یکسان انجام می‌دهند.

PRODUCTION	SEMANTIC RULES
1) $E \rightarrow E_1 + T$	$E.node = \text{new Node}('+', E_1.node, T.node)$
2) $E \rightarrow E_1 - T$	$E.node = \text{new Node}('-', E_1.node, T.node)$
3) $E \rightarrow T$	$E.node = T.node$
4) $T \rightarrow ( E )$	$T.node = E.node$
5) $T \rightarrow \text{id}$	$T.node = \text{new Leaf}(\text{id}, \text{id.entry})$
6) $T \rightarrow \text{num}$	$T.node = \text{new Leaf}(\text{num}, \text{num.val})$

PRODUCTION	SEMANTIC RULES
1) $E \rightarrow T E'$	$E.node = E'.syn$ $E'.inh = T.node$
2) $E' \rightarrow + T E'_1$	$E'_1.inh = \text{new Node}('+', E'.inh, T.node)$ $E'.syn = E'_1.syn$
3) $E' \rightarrow - T E'_1$	$E'_1.inh = \text{new Node}('-', E'.inh, T.node)$ $E'.syn = E'_1.syn$
4) $E' \rightarrow \epsilon$	$E'.syn = E'.inh$
5) $T \rightarrow ( E )$	$T.node = E.node$
6) $T \rightarrow \text{id}$	$T.node = \text{new Leaf}(\text{id}, \text{id.entry})$
7) $T \rightarrow \text{num}$	$T.node = \text{new Leaf}(\text{num}, \text{num.val})$

- برای ساختن درخت نحوی  $x + y$  باید  $x$  به عنوان یک ویژگی به ارث برده شده به  $+y$  ارسال شود، زیرا  $x$  و  $+y$  در دو زیر درخت متفاوت هستند.

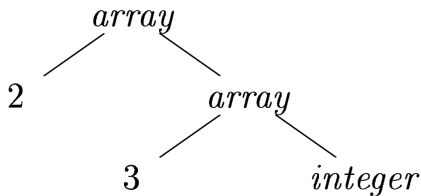
- گراف وابستگی برای عبارت  $a - 4 + c$  به صورت زیر است.



- متغیر  $E'$  یک ویژگی به ارث برده  $inh$  و یک ویژگی ترکیب شده  $syn$  دارد. ویژگی  $E'.inh$  در واقع ریشه درخت برای پیشوند رشته‌ای که سمت چپ زیر درخت  $E'$  است را نگهداری می‌کند.
- در رأس ۵ در درخت وابستگی  $E'.inh$  ریشه درخت نحوی برای شناسه  $a$  است. در رأس ۶، ویژگی  $E'.inh$  ریشه درخت نحوی برای عبارت  $a - 4$  است. در رأس ۹، ویژگی  $E'.inh$  درخت نحوی برای عبارت  $a - 4 + c$  است.
- ویژگی  $syn$  عبارت به دست آمده را منتقل می‌کند تا به  $E.node$  برسد.

- ویژگی‌های به ارث برده شده در مواقعی استفاده می‌شوند که می‌خواهیم اطلاعاتی را از یک قسمت درخت به یک قسمت دیگر درخت منتقل کنیم.

- مثال : در زبان سی، نوع `int[2][3]` آرایه‌ای است تشکیل شده از دو آرایه که هرکدام آن آرایه‌ها سه عدد صحیح را شامل می‌شوند.
- این آرایه را می‌توانیم به صورت `array(2, array(3, integer))` نمایش دهیم و یک درخت تجزیه به صورت زیر برای آن تولید کنیم.



- عملگر array دو پارامتر دریافت می‌کند، یک عدد که تعداد عناصر آن است و یک نوع که نوع عناصر آن است.

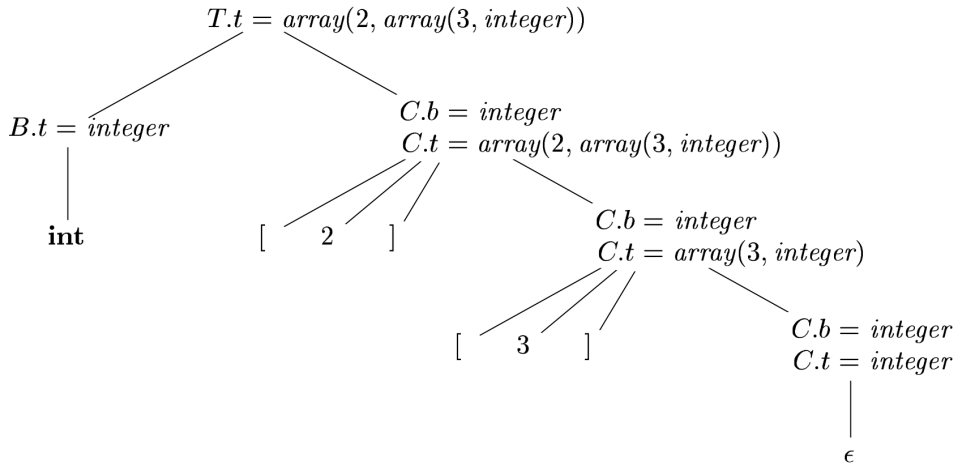


- گرامر نحوی-معنایی زیر برای ساختن درخت نحوی تعریف آرایه‌ها می‌تواند مورد استفاده قرار بگیرد.

PRODUCTION	SEMANTIC RULES
$T \rightarrow B C$	$T.t = C.t$ $C.b = B.t$
$B \rightarrow \text{int}$	$B.t = \text{integer}$
$B \rightarrow \text{float}$	$B.t = \text{float}$
$C \rightarrow [\text{num}] C_1$	$C.t = \text{array}(\text{num.val}, C_1.t)$ $C_1.b = C.b$
$C \rightarrow \epsilon$	$C.t = C.b$

- متغیر T یا یک نوع معمولی تولید می‌کند و یا یک نوع آرایه. متغیر B می‌تواند نوع‌های int یا float را تولید کند. یک نوع معمولی وقتی تولید می‌شود که T مشتق کند BC و C رشته تهی را مشتق کند. اگر C تهی مشتق نکند یک آرایه تولید می‌شود.
- متغیرهای B و T یک ویژگی ترکیب شده به نام t دارند که نوع را مشخص می‌کند.
- متغیر C دو ویژگی دارد : یک ویژگی به ارث برده شده b و یک ویژگی ترکیب شده t . ویژگی b نوع را در درخت تجزیه به پایین منتشر می‌کند و ویژگی t نتیجه را به دست می‌دهد.

- درخت تجزیه حاشیه‌گذاری شده برای ورودی  $\text{int}[2][3]$  در شکل زیر نمایش داده شده است.



- در ریشه درخت برای  $T \rightarrow BC$  متغیر  $C$  نوع را از  $B$  توسط ویژگی  $C.b$  به ارث می‌برد. در نود سمت راست برای  $C$ ، قانون تولید  $C \rightarrow \epsilon$  است پس  $C.t$  برابر با  $C.b$  است.
- قانون معنایی برای قانون تولید  $C \rightarrow [\text{num}]C_1$  مقدار  $C.t$  را تولید می‌کند.

- حال یک مثال دیگر را در نظر می‌گیریم. در این مثال می‌خواهیم در یک زبان برنامه نویسی، عبارت‌های تخصیص مقدار داشته باشیم. نام متغیرها می‌تواند A یا B یا C باشد و نوع متغیرها می‌تواند int یا real باشد. سمت راست یک عبارت تخصیص مقدار می‌تواند یک متغیر و یا جمع چندین مقدار باشد. وقتی دو متغیر سمت راست از نوع‌های متفاوت باشند، مقدار محاسبه شده real است. اما وقتی دو متغیر سمت راست از یک نوع باشند، مقدار محاسبه شده از نوع آن متغیرهاست. نوع محاسبه شده در سمت راست عملیات انتساب باید با نوع متغیر سمت چپ عملیات انتساب یکسان باشد.
- با استفاده از گرامر مستقل از متن، این گرامر را به صورت زیر می‌نویسیم :

$\langle \text{assign} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{expr} \rangle$

$\langle \text{expr} \rangle \rightarrow \langle \text{var} \rangle + \langle \text{var} \rangle \mid \langle \text{var} \rangle$

$\langle \text{var} \rangle \rightarrow A \mid B \mid C$

- حال برای متغیرهای این گرامر دو متغیر صفت در نظر می‌گیریم. نوع واقعی (actual-type) و نوع مورد انتظار (expected-type).
- نوع واقعی : هرکدام از متغیرهای `<var>` و `<expr>` در گرامر، یک صفت ساخته شده دارند که برای ذخیره نوع آنها (که `int` یا `real` است) به کار می‌رود. برای نماد غیر پایانی `<var>` صفت آن ذاتی است و برای نماد غیر پایانی `<expr>` صفت آن (که نوع داده‌ای آن است) از روی صفت فرزندان آن به دست می‌آید.
- نوع مورد انتظار : این نوع مورد انتظار در یک صفت به ارث برده شده برای نماد غیر پایانی `<expr>` است.

— گرامر صفت برای مثال قبلی را به صورت زیر می‌نویسیم.

- Syntax rule :     $\langle \text{assign} \rangle \rightarrow \langle \text{var} \rangle = \langle \text{expr} \rangle$

Semantic rule :     $\langle \text{expr} \rangle . \text{expected-type} \leftarrow \langle \text{var} \rangle . \text{actual-type}$
- Syntax rule :     $\langle \text{expr} \rangle \rightarrow \langle \text{var} \rangle [2]^+ \langle \text{var} \rangle [3]$

Semantic rule :     $\langle \text{expr} \rangle . \text{actual-type} \leftarrow \text{if } ( \langle \text{var} \rangle [2] . \text{actual-type} = \text{int} )$   
                                and (  $\langle \text{var} \rangle [3] . \text{actual-type} = \text{int}$  )  
                                then int  
                                else real  
                                and if

Predictor :     $\langle \text{expr} \rangle . \text{actual-type} = \langle \text{expr} \rangle . \text{expected-type}$
- Syntax rule :     $\langle \text{expr} \rangle \rightarrow \langle \text{var} \rangle$

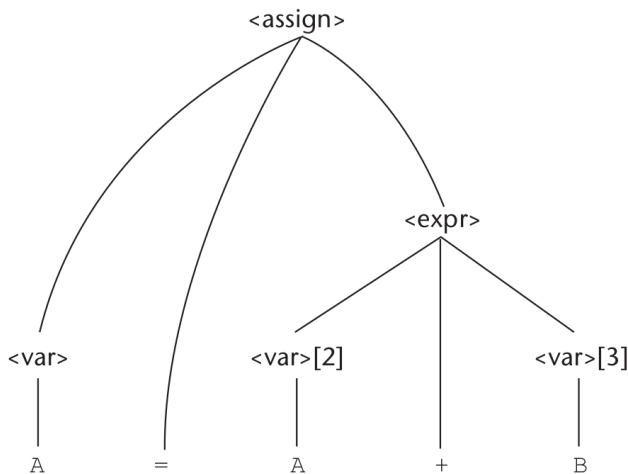
Semantic rule :     $\langle \text{expr} \rangle . \text{actual-type} \leftarrow \langle \text{var} \rangle . \text{actual-type}$

Predictor :     $\langle \text{expr} \rangle . \text{actual-type} = \langle \text{expr} \rangle . \text{expected-type}$
- Syntax rule :     $\langle \text{var} \rangle \rightarrow A \mid B \mid C$

Semantic rule :     $\langle \text{var} \rangle . \text{actual-type} \leftarrow \text{look-up } ( \langle \text{var} \rangle . \text{string} )$

- تابع look-up در واقع به ازای نام یک متغیر، نوع آن را باز می‌گرداند.

- درخت تجزیه برای عبارت  $A = A + B$  در شکل زیر نشان داده شده است.

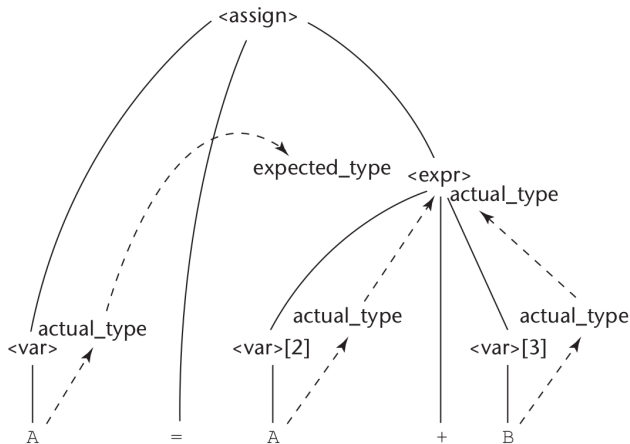




- حال فرایند محاسبه صفت‌ها در یک درخت تجزیه را در نظر بگیرید. اگر همهٔ صفت‌ها، صفت‌های به ارث برده شده بودند، با پیمایش درخت از بالا به پایین می‌توانیم صفت همه رئوس را بدست آوریم. اگر همه صفت‌ها، صفت‌های ساخته شده بودند با پیمایش درخت از پایین به بالا می‌توانیم همه صفت‌ها را محاسبه کنیم. اما در واقع همیشه ترکیبی از صفت‌های به ارث برده شده و ساخته شده داریم پس پیمایش از هر دو طرف صورت می‌گیرد.

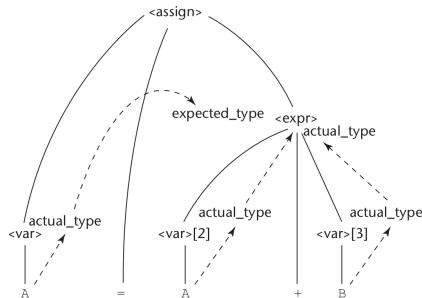
## کاربردهای ترجمه نحوی

- شکل زیر نحوه محاسبه صفت‌ها در درخت تجزیه را نشان می‌دهد. نوع واقعی (actual-type) یک صفت ساخته شده است، درحالی که نوع مورد انتظار (expected-type) یک صفت به ارث برده شده است.



- برای محاسبه صفت‌ها در درخت تجزیه برای عبارت  $A = A + B$  داریم :

1.  $\langle \text{var} \rangle$ . actual-type  $\leftarrow$  look-up (A) (Rule 4)
2.  $\langle \text{expr} \rangle$ . expected-type  $\leftarrow$   $\langle \text{var} \rangle$ . actual-type (Rule 1)
3.  $\langle \text{var} \rangle$  [2]. actual-type  $\leftarrow$  look-up (A) (Rule 4)
- $\langle \text{var} \rangle$  [3]. actual-type  $\leftarrow$  look-up (B) (Rule 4)
4.  $\langle \text{expr} \rangle$ . actual-type  $\leftarrow$  either int or real (Rule 2)
5.  $\langle \text{expr} \rangle$ . expected-type =  $\langle \text{expr} \rangle$ . actual-type is either true or false (Rule 2)



- شمای ترجمه نحوی<sup>1</sup> روشی مکمل برای نشانه‌گذاری گرامر نحوی-معنایی است.
- شمای ترجمه نحوی یک گرامر مستقل از متن است که قسمت‌هایی از کد در بدنه قوانین آن تعبیه شده است.
- قسمت‌هایی از کد که در شمای ترجمه نحوی تعبیه می‌شوند عملیات معنایی<sup>2</sup> نامیده می‌شوند و می‌توانند در هر قسمتی از بدنه قوانین تولید قرار بگیرند.
- به طور قراردادی اطراف عملیات معنایی اطراف عملیات معنایی کاراکتر آکولاد داشتیم، دو طرف آکولاد علامت آپوستروف قرار می‌دهیم.

---

<sup>1</sup> syntax-directed translation scheme

<sup>2</sup> semantic action

- برای پیاده‌سازی ترجمه نحوی، می‌توانیم از شمای ترجمه نحوی درخت تجزیه ساخته و در رئوسی از درخت تجزیه که عملیات معنایی قرار می‌گیرد، در هنگام پیمایش درخت قطعه کدها را اجرا کنیم.
- معمولاً در حین تجزیه، بدون ساختن درخت تجزیه، ترجمه نحوی پیاده‌سازی می‌شود.
- در اینجا دو نوع شمای ترجمه را بررسی می‌کنیم : (۱) گرامر شمای ترجمه LR و گرامر نحوی-معنایی گرامر صفت S باشد. (۲) گرامر شمای ترجمه LL و گرامر نحوی-معنایی گرامر صفت L باشد.

- ساده‌ترین پیاده‌سازی برای ترجمه نحوی-معنایی وقتی است که گرامر صفت  $S$  باشد و توسط یک تجزیه‌کننده پایین به بالا تجزیه شود.
- در این صورت، می‌توانیم یک شمای ترجمه نحوی بسازیم به گونه‌ای که عملیات معنایی در پایان قانون تولید اجرا می‌شوند وقتی که کاهش یک قانون به اتمام می‌رسد.
- اگر در یک شمای ترجمه نحوی همه عملیات معنایی در پایان قانون قرار بگیرد، به آن شمای ترجمه پسوندی<sup>1</sup> گفته می‌شود.

---

<sup>1</sup> postfix translation scheme

- مثال : شمای ترجمه پسوندی زیر یک ماشین حساب ساده را پیاده‌سازی می‌کند.

$L$	$\rightarrow$	$E \text{ n}$	$\{ \text{print}(E.val); \}$
$E$	$\rightarrow$	$E_1 + T$	$\{ E.val = E_1.val + T.val; \}$
$E$	$\rightarrow$	$T$	$\{ E.val = T.val; \}$
$T$	$\rightarrow$	$T_1 * F$	$\{ T.val = T_1.val \times F.val; \}$
$T$	$\rightarrow$	$F$	$\{ T.val = F.val; \}$
$F$	$\rightarrow$	$( E )$	$\{ F.val = E.val; \}$
$F$	$\rightarrow$	<b>digit</b>	$\{ F.val = \text{digit.lexval}; \}$

- گرامر نحوی-معنایی این ماشین حساب در زیر آمده است.

PRODUCTION	SEMANTIC RULES
1) $L \rightarrow E \mathbf{n}$	$L.val = E.val$
2) $E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
3) $E \rightarrow T$	$E.val = T.val$
4) $T \rightarrow T_1 * F$	$T.val = T_1.val \times F.val$
5) $T \rightarrow F$	$T.val = F.val$
6) $F \rightarrow ( E )$	$F.val = E.val$
7) $F \rightarrow \mathbf{digit}$	$F.val = \mathbf{digit.lexval}$

- از آنجایی که این گرامر LR است و گرامر نحوی-معنایی گرامر صفت S است، عملیات معنایی می‌توانند در حین تجزیه در زمان کاهش قوانین اجرا شوند.



- ترجمه نحوی پسوندی می‌تواند در حین تجزیه LR با اجرای عملیات معنایی وقتی عملیات کاهش انجام می‌شود، پیاده‌سازی شود.
- ویژگی‌های نمادهای گرامر می‌توانند بر روی پشته قرار بگیرند. می‌توان ویژگی‌ها را همراه با نمادهای گرامر در یک مکان بر روی پشته قرار داد.

## شمای ترجمه پسوندی

- در شکل زیر پشته تجزیه‌کننده نمادهای گرامر و ویژگی‌های آنها را شامل می‌شود.

	$X$	$Y$	$Z$
	$X.x$	$Y.y$	$Z.z$

State/grammar symbol

Synthesized attribute(s)

↑  
top

- نمادهای  $XYZ$  بر روی پشته قرار دارند و ممکن است با قانون  $A \rightarrow XYZ$  کاهش پیدا کنند. در اینجا یک ویژگی برای هر نماد استفاده شده اما می‌توان اشاره‌گری به لیستی از ویژگی‌ها برای هر نماد در نظر گرفت.
- اگر ویژگی‌ها همه ترکیب شده باشند و عملیات معنایی در انتهای قانون تولید باشند، می‌توانیم ویژگی‌ها برای متغیر یک قانون را وقتی بدنه قانون کاهش پیدا می‌کند محاسبه کنیم.
- وقتی می‌خواهیم  $XYZ$  را با  $A$  کاهش دهیم، همه ویژگی‌ها در دسترس هستند و پس از کاهش  $A$  با تمام ویژگی‌هایش بر روی پشته قرار می‌گیرد.

## شمای ترجمه پسوندی

- شمای ترجمه پسوندی زیر برای یک ماشین حساب طراحی شده است. در عملیات معنایی محتوای پشته مستقیماً دستکاری می‌شود.

PRODUCTION	ACTIONS
$L \rightarrow E \mathbf{n}$	{ $\text{print}(\text{stack}[\text{top} - 1].\text{val});$ $\text{top} = \text{top} - 1;$ }
$E \rightarrow E_1 + T$	{ $\text{stack}[\text{top} - 2].\text{val} = \text{stack}[\text{top} - 2].\text{val} + \text{stack}[\text{top}].\text{val};$ $\text{top} = \text{top} - 2;$ }
$E \rightarrow T$	
$T \rightarrow T_1 * F$	{ $\text{stack}[\text{top} - 2].\text{val} = \text{stack}[\text{top} - 2].\text{val} \times \text{stack}[\text{top}].\text{val};$ $\text{top} = \text{top} - 2;$ }
$T \rightarrow F$	
$F \rightarrow ( E )$	{ $\text{stack}[\text{top} - 2].\text{val} = \text{stack}[\text{top} - 1].\text{val};$ $\text{top} = \text{top} - 2;$ }
$F \rightarrow \mathbf{digit}$	

## شمای ترجمه با عملیات معنایی در بدنه قانون

- عملیات معنایی می‌تواند در بدنه قانون تولید قرار بگیرد. در این صورت هنگامی که همهٔ نمادهای سمت چپ عملیات معنایی در فرایند تجزیه پردازش شدند، عملیات انجام می‌شود.
- اگر قانونی به صورت  $B \rightarrow X\{a\}Y$  داشته باشیم، عملیات  $a$  وقتی  $X$  پردازش شد انجام می‌شود. اگر نماد  $X$  یک ترمینال باشد عملیات  $a$  پس از شناسایی ترمینال  $X$  انجام می‌شود و اگر  $X$  یک متغیر باشد  $a$  پس از شناسایی همه ترمینال‌هایی که از  $X$  مشتق می‌شوند انجام می‌شود.
- اگر تجزیه پایین به بالا باشد، عملیات  $a$  به محض اینکه  $X$  بر روی پشته قرار گرفت انجام می‌شود.
- اگر تجزیه بالا به پایین باشد، عملیات  $a$  قبل از اینکه  $Y$  پردازش شود، انجام می‌شود.
- توجه کنید همه شماهای ترجمه را نمی‌توان در حین تجزیه پیاده‌سازی کرد.

## شمای ترجمه با عملیات معنایی در بدنه قانون

- همه شمایای ترجمه را می‌توان به صورت زیر پیاده‌سازی کرد.

۱. با چشم‌پوشی از عملیات معنایی، ورودی را تجزیه کرده یک درخت تجزیه تولید می‌کنیم.
۲. همه رئوس میانی  $N$  متناظر با قانون تولید  $A \rightarrow \alpha$  را بررسی می‌کنیم. به ازای هریک از عملیات معنایی در  $\alpha$  یک فرزند برای رأس  $N$  اضافه می‌کنیم، به طوری که فرزند اضافه شده در مکان درست خود در  $\alpha$  در بین فرزندان  $N$  قرار بگیرد.
۳. یک پیمایش پیش‌ترتیبی<sup>۱</sup> بر روی درخت انجام می‌دهیم و هنگامی که یک رأس با یک عملیات معنایی در درخت پیمایش می‌شود، عملیات معنایی متناظر با آن اجرا می‌شود.

---

<sup>۱</sup> preorder

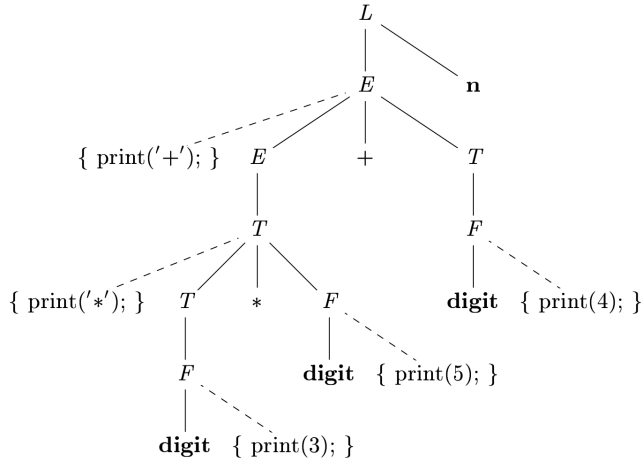
## شمای ترجمه با عملیات معنایی در بدنه قانون

- مثال : شمای ترجمه زیر را در نظر بگیرید.

- 1)  $L \rightarrow E \text{ n}$
- 2)  $E \rightarrow \{ \text{print}('+'); \} E_1 + T$
- 3)  $E \rightarrow T$
- 4)  $T \rightarrow \{ \text{print}('*'); \} T_1 * F$
- 5)  $T \rightarrow F$
- 6)  $F \rightarrow ( E )$
- 7)  $F \rightarrow \text{digit} \{ \text{print}(\text{digit.lexval}); \}$

## شمای ترجمه با عملیات معنایی در بدنه قانون

- درخت تجزیه برای عبارت  $3 * 5 + 4$  به صورت زیر خواهد بود. با پیمایش درخت تجزیه عبارت  $3 * 5 + 4$  چاپ می‌شود.



## حذف بازگشت چپ از شمای ترجمه

- از آنجایی که هیچ گرامری با بازگشت چپ نمی‌تواند از بالا به پایین تجزیه شود، باید بازگشت به چپ حذف شود. برای شمای ترجمه نیز باید بازگشت چپ بررسی شود.
- وقتی می‌خواهیم بازگشت چپ را از شمای ترجمه حذف کنیم باید با عملیات معنایی مانند ترمینال‌ها رفتار کنیم.
- قبلاً اشاره کردیم که برای حذف بازگشت چپ در قانون به شکل  $A \rightarrow A\alpha|\beta$  می‌توانیم آن را به شکل  $A \rightarrow \beta R$  و  $R \rightarrow \alpha R|\epsilon$  در آوریم، به طوری که گرامر جدید همان رشته‌های گرامر اول را با متغیر  $R$  تولید کند.



## حذف بازگشت چپ از شمای ترجمه

- مثال : شمای ترجمه زیر را در نظر بگیرید.

$$\begin{aligned} E &\rightarrow E_1 + T \quad \{ \text{print}(' + '); \} \\ E &\rightarrow T \end{aligned}$$

- می‌توانیم بازگشت چپ را از این گرامر حذف و آن را به صورت زیر درآوریم.

$$\begin{aligned} E &\rightarrow T R \\ R &\rightarrow + T \{ \text{print}(' + '); \} R \\ R &\rightarrow \epsilon \end{aligned}$$

## حذف بازگشت چپ از شمای ترجمه

- حال فرض کنید در شمای ترجمه می‌خواهیم ویژگی‌های متغیرها را محاسبه کنیم و شمای ترجمه بازگشت چپ دارد. با یک مثال حذف بازگشت چپ را بررسی می‌کنیم.
- فرض کنید دو قانون تولید به صورت زیر داریم.

$$\begin{aligned} A &\rightarrow A_1 Y \{A.a = g(A_1.a, Y.y)\} \\ A &\rightarrow X \{A.a = f(X.x)\} \end{aligned}$$

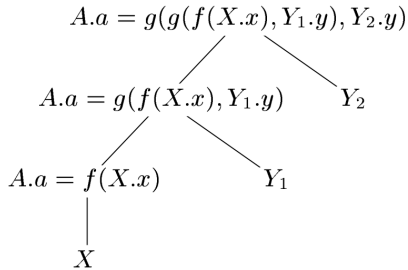
## حذف بازگشت چپ از شمای ترجمه

- در اینجا  $A.a$  یک ویژگی ترکیب شده است از متغیر  $A$  که بازگشت چپ دارد.
- می‌خواهیم این گرامر را با حفظ عملیات معنایی به صورت زیر درآوریم.

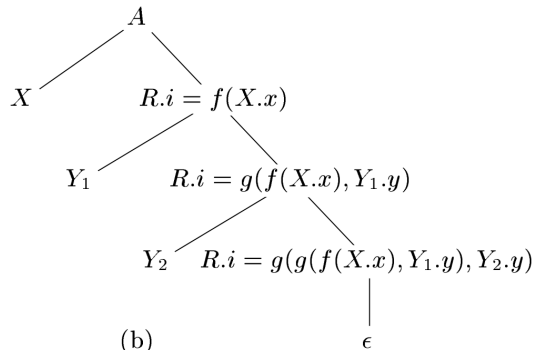
$$\begin{array}{lcl} A & \rightarrow & X R \\ R & \rightarrow & Y R \mid \epsilon \end{array}$$

# حذف بازگشت چپ از شمای ترجمه

- شکل زیر نشان می‌دهد شمای ترجمه غیربازگشتی چگونه باید باشد.



(a)



(b)

## حذف بازگشت چپ از شمای ترجمه

- تابع  $f$  یک بار اعمال می‌شود که مربوط به قانون تولید  $A \rightarrow X$  است و سپس  $g$  به تعداد باری که قانون  $A \rightarrow AY$  اعمال می‌شود باید اعمال شود.
- با هر بار اعمال قانون  $R \rightarrow YR$  تابع  $g$  یک بار اعمال می‌شود.
- برای  $R$  یک ویژگی به ارث برده شده  $R.i$  تعریف می‌کنیم که نتیجه اعمال تابع  $g$  را جمع‌آوری می‌کند.
- همچنین  $R$  باید یک ویژگی ترکیب شده  $R.s$  را داشته باشد. این ویژگی وقتی  $R$  تولید نماد  $Y$  را به اتمام می‌رساند محاسبه می‌شود. سپس  $R.s$  در درخت کپی می‌شود تا به  $A.a$  برسد.

## حذف بازگشت چپ از شمای ترجمه

- برای حذف بازگشت چپ در گرامر قبل آن را به صورت زیر در می آوریم.

$$\begin{aligned} A &\rightarrow X \{R.i = f(X.x)\} R \{A.a = R.s\} \\ R &\rightarrow Y \{R_1.i = g(R.i, Y.y)\} R_1 \{R.s = R_1.s\} \\ R &\rightarrow \epsilon \{R.s = R.i\} \end{aligned}$$

- دقت کنید که ویژگی به ارث برده شده  $R.i$  قبل از  $R$  در بدنه محاسبه می شود و ویژگی های ترکیب شده  $A.a$  و  $R.s$  در پایان قانون ارزیابی می شوند. بدین ترتیب ویژگی های مورد نیاز، قبل از ارزیابی در دسترس خواهند بود.

## ترجمه نحوی برای گرامر صفت L

- گفتیم در صورتی که یک گرامر LR باشد، ترجمه نحوی پسوندی می‌تواند در حین تجزیه پایین به بالا انجام شود.
  - حال فرض کنید یک گرامر صفت L داریم. فرض می‌کنیم گرامر می‌تواند توسط یک تجزیه کننده بالا به پایین تجزیه شود.
  - یک روش کلی برای محاسبه ویژگی‌ها و ارزیابی گرامر صفت L به صورت زیر است.
۱. عملیات معنایی برای محاسبه ویژگی‌های به ارث برده شده برای متغیر A را قبل از متغیر A در بدنه قانون تولید قرار می‌دهیم.
  ۲. عملیات معنایی برای ویژگی‌های ترکیب شده را در انتهای قوانین تولید قرار می‌دهیم.

- مثال : زبان Eqn زبانی است که قبل از زبان T<sub>E</sub>X برای حروف چینی مستندات استفاده می‌شد.
- در این زبان برای نوشتن عبارت  $a_{ij}$  می‌نویسیم `a sub i sub j`.
- می‌توانیم گرامری به صورت زیر برای توصیف این عبارات بنویسیم.  
$$B \rightarrow B_1 B_2 \mid B_1 \text{ sub } B_2 \mid ( B_1 ) \mid \text{text}$$
- این گرامر جعبه‌هایی<sup>1</sup> تولید می‌کند که هریک می‌توانند زیرنویس دیگری باشند و یا در کنار یکدیگر قرار بگیرند.

---

<sup>1</sup> box

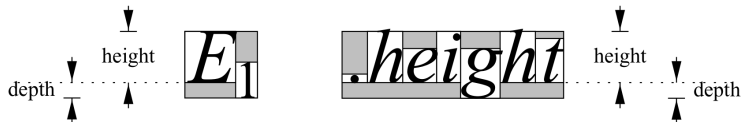


## ترجمه نحوی برای گرامر صفت L

- چهار قانون در این گرامر وجود دارد :

۱. یک جعبه می‌تواند تشکیل شود از دو جعبه که در کنار یکدیگر قرار گرفته‌اند.
۲. یک جعبه می‌تواند زیرنویس یک جعبه دیگر باشد. در این صورت جعبه دوم به صورت کوچک در سمت راست جعبه اول قرار می‌گیرد.
۳. برای دسته‌بندی جعبه‌ها از پرانتزگذاری استفاده می‌شود. در زبان Eqn و  $\text{TeX}$  از آکولاد برای دسته‌بندی استفاده می‌شود. در اینجا برای جلوگیری از ابهام و متمایز کردن عملیات معنایی از پرانتز به جای آکولاد استفاده کرده‌ایم.
۴. یک جعبه می‌تواند شامل یک رشته یا دنباله‌ای از کاراکترها باشد.

- در شکل زیر دو جعبه برای  $E_1$  و  $\text{height}$  نشان داده شده‌اند. دو جعبه  $E_1$  و  $\text{height}$  به یکدیگر متصل شده و جعبه  $\text{height} \cdot E_1$  را می‌سازند. همچنین جعبه عدد 1 در سمت راست جعبه E کمی پایین‌تر با اندازه ۳۰ درصد کوچکتر قرار گرفته است.



## ترجمه نحوی برای گرامر صفت L

- چند قانون برای تولید متن متناظر با قوانین گرامر وجود دارد.

۱. فرض می‌کنیم اندازه متن در یک جعبه اصلی  $10^\circ$  است و اگر اندازه یک جعبه  $^1$  باشد، اندازه جعبه زیرنویس  $p$  است. برای متغیر  $0.7p$  در گرامر ویژگی  $B$  اندازه آن را نشان می‌دهد. این ویژگی یک ویژگی به ارث برده شده است زیرا اندازه جعبه زیرنویس از اندازه جعبه اصلی به دست می‌آید.

۲. هر جعبه یک خط پایه  $^2$  دارد به طوری که حروف بر روی خط پایه قرار می‌گیرند. برخی از حروف مانند  $g$  پایین‌تر از خط پایه قرار می‌گیرند. هر جعبه یک ارتفاع  $^3$  دارد که فاصله بالای جعبه از خط پایه است. ویژگی  $B.ht$  ارتفاع جعبه  $B$  را به دست می‌دهد. هر جعبه یک عمق  $^4$  دارد که فاصله خط پایه از پایین جعبه را مشخص می‌کند. ویژگی  $B.dp$  عمق جعبه  $B$  را تعیین می‌کند.

---

$^1$  point size

$^2$  baseline

$^3$  height

$^4$  depth

## ترجمه نحوی برای گرامر صفت L

- گرامر نحوی-معنایی زیر قوانین معنایی برای محاسبه اندازه قلم، ارتفاع و عمق را مشخص می‌کند.

PRODUCTION	SEMANTIC RULES
1) $S \rightarrow B$	$B.ps = 10$
2) $B \rightarrow B_1 B_2$	$B_1.ps = B.ps$ $B_2.ps = B.ps$ $B.ht = \max(B_1.ht, B_2.ht)$ $B.dp = \max(B_1.dp, B_2.dp)$
3) $B \rightarrow B_1 \text{ sub } B_2$	$B_1.ps = B.ps$ $B_2.ps = 0.7 \times B.ps$ $B.ht = \max(B_1.ht, B_2.ht - 0.25 \times B.ps)$ $B.dp = \max(B_1.dp, B_2.dp + 0.25 \times B.ps)$
4) $B \rightarrow ( B_1 )$	$B_1.ps = B.ps$ $B.ht = B_1.ht$ $B.dp = B_1.dp$
5) $B \rightarrow \text{text}$	$B.ht = \text{getHt}(B.ps, \text{text.lexval})$ $B.dp = \text{getDp}(B.ps, \text{text.lexval})$

## ترجمه نحوی برای گرامر صفت L

- حال باید گرامر نحوی-معنایی را به شمای ترجمه تبدیل کنیم. در زیر این تبدیل انجام شده است. محاسبه ویژگی‌های به ارث برده شده از یک متغیر باید قبل از آن متغیر در شمای ترجمه قرار بگیرند.

PRODUCTION	ACTIONS
1) $S \rightarrow B$	$\{ B.ps = 10; \}$
2) $B \rightarrow B_1 B_2$	$\{ B_1.ps = B.ps; \}$ $\{ B_2.ps = B.ps; \}$ $\{ B.ht = \max(B_1.ht, B_2.ht);$ $B.dp = \max(B_1.dp, B_2.dp); \}$
3) $B \rightarrow B_1 \text{ sub } B_2$	$\{ B_1.ps = B.ps; \}$ $\{ B_2.ps = 0.7 \times B.ps; \}$ $\{ B.ht = \max(B_1.ht, B_2.ht - 0.25 \times B.ps);$ $B.dp = \max(B_1.dp, B_2.dp + 0.25 \times B.ps); \}$
4) $B \rightarrow ( B_1 )$	$\{ B_1.ps = B.ps; \}$ $\{ B.ht = B_1.ht;$ $B.dp = B_1.dp; \}$
5) $B \rightarrow \text{text}$	$\{ B.ht = \text{getHt}(B.ps, \text{text.lexval});$ $B.dp = \text{getDp}(B.ps, \text{text.lexval}); \}$

- در مثال بعدی می‌خواهیم توسط گرامر نحوی-معنایی برای دستور حلقه تکرار در یک گرامر کدمیانی تولید کنیم. کدمیانی در واقع یک رشته است که در فرایند تجزیه توسط ترجمه نحوی تولید می‌شود.

- مثال : گرامری با یک قانون تولید به صورت زیر را در نظر بگیرید.

$$S \rightarrow \mathbf{while} ( C ) S_1$$

- در اینجا S متغیری است که همه انواع عبارات را تولید می‌کند. متغیر C نیز عبارات منطقی تولید می‌کند.

- دستور while عبارت C را ارزیابی می‌کند. اگر مقدار آن درست بود کنترل برنامه به ابتدای دستورات  $S_1$  منتقل می‌شود. اگر مقدار آن نادرست بود، کنترل برنامه به بعد از دستور while منتقل می‌شود.

- برای تولید میانی از دستور `while` می‌توانیم گرامر نحوی معنایی زیر را بنویسیم.

$$S \rightarrow \mathbf{while} ( C ) S_1 \quad \begin{array}{l} L1 = new(); \\ L2 = new(); \\ S_1.next = L1; \\ C.false = S.next; \\ C.true = L2; \\ S.code = \mathbf{label} \parallel L1 \parallel C.code \parallel \mathbf{label} \parallel L2 \parallel S_1.code \end{array}$$



## ترجمه نحوی برای گرامر صفت L

- ویژگی‌های به کاربرده شده در این گرامر نحوی-معنایی به شرح زیراند.
- ۱. ویژگی به ارث برده شده `S.next` برچسب کدی که بعد از `S` باید اجرا شود را ذخیره می‌کند.
- ۲. ویژگی ترکیب شده `S.code` دنباله‌ای از کدهای میانی معادل `S` را ذخیره می‌کند.
- ۳. ویژگی به ارث برده شده `C.true` برچسب کدی است که باید اجرا شود اگر `C` درست باشد.
- ۴. ویژگی به ارث برده شده `C.false` برچسب کدی است که باید اجرا شود اگر `C` نادرست باشد.
- ۵. ویژگی ترکیب شده `C.code` دنباله‌ای از کدهای میانی که `C` را پیاده‌سازی می‌کند ذخیره می‌کند.

## ترجمه نحوی برای گرامر صفت L

- تابع new برچسب جدید تولید می‌کند.
- متغیرهای  $L_1$  و  $L_2$  برچسب‌هایی که در کد میانی نیاز داریم را ذخیره می‌کنند.
- متغیر  $L_1$  ابتدای کدی است که بعد از اتمام دستور while اجرا می‌شود. بنابراین  $S_1.next$  برابر است با  $L_1$ .  
متغیر  $L_2$  برچسب ابتدای کد  $S_1$  است. بنابراین مقدار  $C.true$  برابر است با  $L_2$ .
- مقدار  $C.false$  برابر است با  $S.next$  زیرا اگر شرط نادرست باشد باید کدی به دنبال  $S$  می‌آید اجرا شود.
- از نماد || برای الحاق قطعات کد استفاده می‌شود.

- شمای ترجمه معادل گرامر نحوی-معنایی قبل به صورت زیر است.

$$\begin{array}{lcl} S \rightarrow \text{while} ( & \{ L1 = \text{new}(); L2 = \text{new}(); C.\text{false} = S.\text{next}; C.\text{true} = L2; \} \\ C ) & \{ S_1.\text{next} = L1; \} \\ S_1 & \{ S.\text{code} = \text{label} \parallel L1 \parallel C.\text{code} \parallel \text{label} \parallel L2 \parallel S_1.\text{code}; \} \end{array}$$

## ترجمه نحوی برای گرامر صفت L

- برای پیاده‌سازی گرامرهای صفت L در حالت کلی ابتدا درخت تجزیه را ساخته و عملیات معنایی را در مکان مناسب اضافه می‌کنیم. سپس درخت به صورت پیش‌ترتیبی پیمایش می‌کنیم.