به نام خدا

# طراحی کامپایلر

آرش شفیعی

# تحلیل معنایی

# ترجمه نحوی

- 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 [1] 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 [2] (صفت) در نظر گرفته می‌شود.
- در ترجمه نحوی مقادیر ویژگی‌های نمادهای گرامر با نسبت دادن قوانین معنایی به قوانین تولید گرامر محاسبه می‌شوند.

[1] syntax-directed translation
[2] attribute

# ترجمه نحوی

- 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 | SEMANTIC RULE |
|---|---|
| $E \to E_1 + T$ | $E.code = E_1.code \parallel T.code \parallel '+'$ |

- در این قانون دو متغیر به نام E و T وجود دارد. برای متمایزکردن 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.

# ترجمه نحوی

- فرض کنید برای قانون تولید $E \to E_1 - T$ قانون معنایی $E.code = E_1.code || T.code || '-'$ را تعریف می‌کنیم.
- همچنین برای قانون تولید $E \to T$ قانون معنایی $E.code = T.code$ و برای قوانین تولید $T \to id$ 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

- گرامر نحوی-معنایی [1] ، یک گرامر مستقل از متن همراه با قوانین معنایی و ویژگی برای متغیرهاست.
- ویژگی‌ها به نمادهای گرامر 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 از هر نوعی از جمله اعداد، رشته‌ها، اشاره‌گر به جداول وغیره باشند.
- یک ویژگی می‌تواند از نوع رشته باشد و مقدار آن یک کد (برای مثال کدمیانی) باشد.

[1] Syntax-directed definition (SDD)

# ویژگی‌های موروثی و ترکیبی

- دو نوع ویژگی برای متغیرهای یک گرامر می‌توانند وجود داشته باشند : ویژگی‌های ترکیبی [1] و ویژگی‌های موروثی [2].

۱. یک ویژگی ترکیبی (ویژگی ساختگی) 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 ترکیبی در رأس N با استفاده از مقادیر ویژگی‌های فرزندان رأس N و خود رأس N محاسبه می‌شود.

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

[1] synthesized attribute
[2] inherited attribute

# ویژگی‌های موروثی و ترکیبی

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

# ویژگی‌های موروثی و ترکیبی

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

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

# ویژگی‌های موروثی و ترکیبی

- 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 ترکیبی به نام `val` دارند. همچنین ترمینال **`digit`** یک ویژگی ترکیبی 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 ترکیبی است.

# ویژگی‌های موروثی و ترکیبی

- اگر یک گرامر نحوی–معنایی (SDD) تنها شامل ویژگی‌های ترکیبی باشد به آن گرامر نحوی–معنایی ترکیبی یا گرامر نحوی–معنایی S [1] گفته می‌شود.
- در یک گرامر نحوی–معنایی ترکیبی، در هر قانون تولید، ویژگی‌های متغیرهای قوانین از طریق ویژگی‌های متغیرهای بدنه قانون محاسبه می‌شوند.

[1] S–attributed (synthesized) SDD

# ویژگی‌های موروثی و ترکیبی

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

[1] side effect
[2] attribute grammar

# ارزیابی گرامر نحوی–معنایی

- 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 [1] نامیده می‌شود.
- قبل از اینکه بتوانیم مقدار یک ویژگی را در یک رأس محاسبه کنیم، باید مقدار همهٔ ویژگی‌هایی را که آن ویژگی به آنها بستگی دارد محاسبه کرده باشیم. برای مثال اگر یک ویژگی ترکیبی باشد، باید همهٔ ویژگی‌های فرزندان یک رأس را قبل از محاسبه ویژگی آن رأس محاسبه کنیم.

[1] annotated parse tree

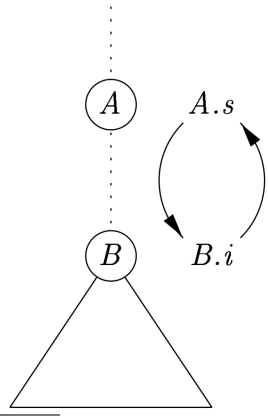
# ارزیابی گرامر نحوی–معنایی

- 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 موروثی و ترکیبی 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 ترکیبی و موروثی باشند. گرامر نحوی–معنایی به صورت زیر خواهد بود.

| PRODUCTION | SEMANTIC RULES |
|---|---|
| $A \to B$ | $A.s = B.i;$ <br> $B.i = A.s + 1$ |

# ارزیابی گرامر نحوی–معنایی

- این ویژگی‌ها دارای وابستگی مُدَوَّر (دارای دور) [1] هستند.
- این دور در شکل زیر نشان داده شده است.




[1] circular dependency

# ارزیابی گرامر نحوی–معنایی

- مسئله تعیین وجود دور در یکی از درخت‌های تجزیه حاشیه‌گذاری شده برای یک گرامر نحوی–معنایی دلخواه، یک مسئلهٔ ان پی کامل است و بنابراین تحلیل گرامر نحوی–معنایی 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 همهٔ ویژگی‌ها محاسبه خواهند شد که بعدها به آن می‌پردازیم.

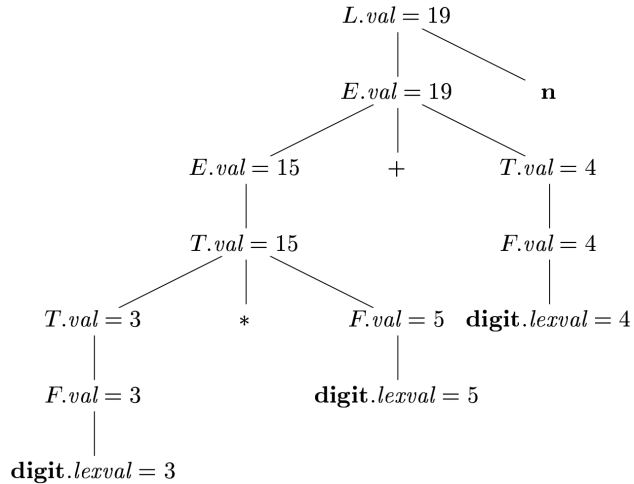
# ارزیابی گرامر نحوی–معنایی

- 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) | $L \to E$ **n** | $L.val = E.val$ |
| 2) | $E \to E_1 + T$ | $E.val = E_1.val + T.val$ |
| 3) | $E \to T$ | $E.val = T.val$ |
| 4) | $T \to T_1 * F$ | $T.val = T_1.val \times F.val$ |
| 5) | $T \to F$ | $T.val = F.val$ |
| 6) | $F \to ( E )$ | $F.val = E.val$ |
| 7) | $F \to$ **digit** | $F.val =$ **digit**.lexval |

# ارزیابی گرامر نحوی–معنایی

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



# ارزیابی گرامر نحوی–معنایی

- در مثال بعد نشان می‌دهیم چگونه ویژگی‌های موروثی استفاده می‌شوند.

# ارزیابی گرامر نحوی–معنایی

- مثال : گرامر نحوی–معنایی (SDD) زیر عبارات محاسباتی دارای عملگر ضرب مانند $3 * 5$ و $7 * 5 * 3$ 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 موروثی برای انتقال عملوند سمت چپ به عملگر استفاده می‌شود.

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

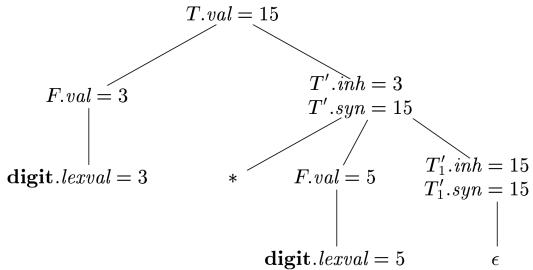
# ارزیابی گرامر نحوی–معنایی

- هریک از متغیرهای T و F یک ویژگی ترکیبی به نام `val` دارند و ترمینال **`digit`** یک ویژگی ترکیبی به نام `lexval` دارد.
- متغیر $T'$ دو ویژگی دارد. یک ویژگی موروثی به نام `inh` و یک ویژگی ترکیبی به نام `syn`.
- متغیر $T'$ ابتدا مقدار متغیر F (ویژگی $F.val$) را در قانون $T \to FT'$ به ارث می‌برد. سپس متغیر $T'_1$ در قانون تولید $T' \to *FT'_1$ از ویژگی موروثی $T'$ استفاده می‌کند.

| | Production | Semantic Rules |
|---|---|---|
| 1) | $T \to F\ T'$ | $T'.inh = F.val$<br>$T.val = T'.syn$ |
| 2) | $T' \to *\ F\ T'_1$ | $T'_1.inh = T'.inh \times F.val$<br>$T'.syn = T'_1.syn$ |
| 3) | $T' \to \epsilon$ | $T'.syn = T'.inh$ |
| 4) | $F \to \mathbf{digit}$ | $F.val = \mathbf{digit}.lexval$ |

# ارزیابی گرامر نحوی–معنایی

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




| | PRODUCTION | SEMANTIC RULES |
|---|---|---|
| 1) | $T \to F\ T'$ | $T'.inh = F.val$<br>$T.val = T'.syn$ |
| 2) | $T' \to *\ F\ T'_1$ | $T'_1.inh = T'.inh \times F.val$<br>$T'.syn = T'_1.syn$ |
| 3) | $T' \to \epsilon$ | $T'.syn = T'.inh$ |
| 4) | $F \to$ **digit** | $F.val =$ **digit**$.lexval$ |

- فرض کنید عبارت $7 * 5 * 3$ داده شده است. ریشه زیر درختی که عبارت $7 * 5*$ را تجزیه می‌کند، عبارت 3 را به ارث می‌برد و ریشه زیر درختی که $7*$ را تجزیه می کند، مقدار $5 * 3$ را به ارث می‌برد و در صورتی که عملوندهای بیشتری وجود داشته باشند، این روند ادامه پیدا می‌کند تا جایی که عملوند دیگری وجود نداشته باشد.

# 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 ترکیبی 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) | $T \rightarrow F\ T'$ | $T'.inh = F.val$<br>$T.val = T'.syn$ |
| 2) | $T' \rightarrow *\ F\ T'_1$ | $T'_1.inh = T'.inh \times F.val$<br>$T'.syn = T'_1.syn$ |
| 3) | $T' \rightarrow \epsilon$ | $T'.syn = T'.inh$ |
| 4) | $F \rightarrow \mathbf{digit}$ | $F.val = \mathbf{digit}.lexval$ |

# گراف وابستگی

- گراف‌های وابستگی [1] 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.

[1] 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 ترکیبی 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.

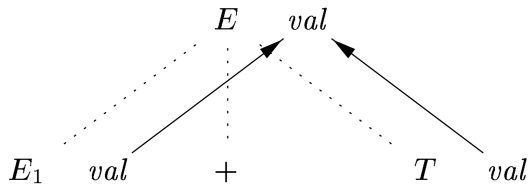
۳. فرض کنید یک قانون معنایی به قانون p نسبت داده شده باشد که در آن ویژگی موروثی B.c با استفاده از مقدار X.a محاسبه می‌شود. در این صورت در گراف وابستگی یالی از X.a به B.c وجود دارد. به عبارت دیگر برای هر رأس N با برچسب B که متناظر با یک متغیر B در بدنه قانون p باشد، یالی از ویژگی a از رأس متناظر با X 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 : گرامر نحوی–معنایی زیر را در نظر بگیرید.

| PRODUCTION | SEMANTIC RULE |
|---|---|
| $E \to E_1 \ + \ T$ | $E.val = E_1.val + T.val$ |

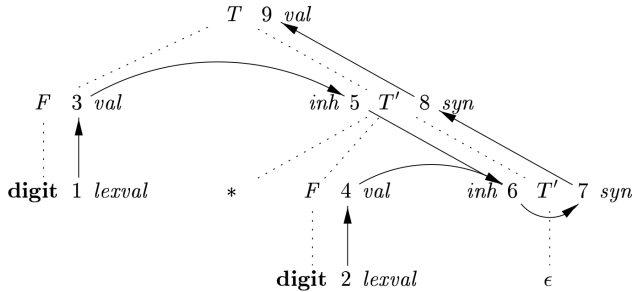
- در هر قسمتی از گراف تجزیه که از این قانون تولید استفاده شده باشد، گراف وابستگی به صورت زیر است. یال‌های درخت با نقطه چین نمایش داده شده‌اند.



# 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.

| | Production | Semantic Rules |
|---|---|---|
| 1) | $T \to F\ T'$ | $T'.inh = F.val$<br>$T.val = T'.syn$ |
| 2) | $T' \to *\ F\ T'_1$ | $T'_1.inh = T'.inh \times F.val$<br>$T'.syn = T'_1.syn$ |
| 3) | $T' \to \epsilon$ | $T'.syn = T'.inh$ |
| 4) | $F \to \mathbf{digit}$ | $F.val = \mathbf{digit}.lexval$ |



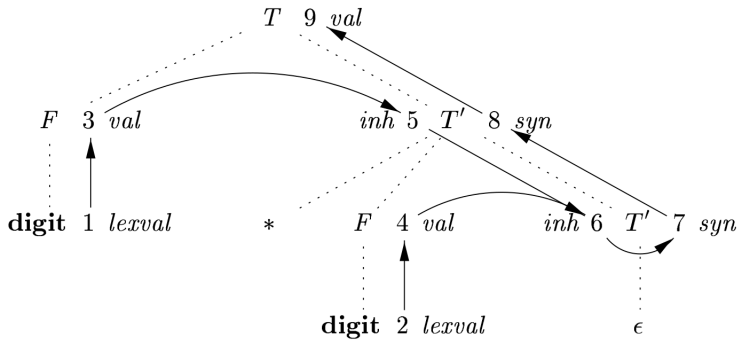
# ترتیب ارزیابی ویژگی‌ها

- 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 $N_1, N_2, \cdots, N_k$ یال‌هایی از $N_i$ به $N_j$ وجود داشته باشد به طوری که $i < j$ آنگاه ویژگی‌های رئوس درخت باید به ترتیب عناصر دنباله محاسبه شوند. به ترتیبی که وابستگی‌ها در یک گراف جهت‌دار را حفظ کند، ترتیب توپولوژیکی [1] 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. مرتب‌سازی توپولوژیکی گراف می‌تواند توسط الگوریتم جستجوی عمق‌اول انجام شود.

[1] topological sort

# ترتیب ارزیابی ویژگی‌ها

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



# گرامر صفت S

- به ازای یک گرامر نحوی–معنایی دلخواه، مسئله تعیین کردن اینکه وابستگی ویژگی‌ها دارای دور هستند یا خیر یک مسئله انپی‌کامل است.
- معمولاً در عمل از زیر مجموعه‌ای از گرامرهای نحوی–معنایی استفاده می‌شود که تضمین می‌کنند گراف وابستگی ویژگی‌ها دارای دور نیست.
- اگر یک گرامر نحوی–معنایی هیچ اثر جانبی نداشته باشد به آن گرامر صفت [1] گفته می‌شود.
- یکی از زیرمجموعه‌های گرامرهای نحوی–معنایی، گرامر صفت S [2] است.
- گرامر صفت S یک گرامر صفت است که در آن همهٔ ویژگی‌ها ترکیبی هستند.

[1] attribute grammar
[2] S-attributed grammar/definitions

# گرامر صفت S

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

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

# گرامر صفت S

- وقتی یک گرامر نحوی–معنایی یک گرامر صفت S 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

- 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.

# گرامر صفت L

- یکی دیگر از زیرمجموعه‌های مهم گرامرهای نحوی–معنایی، گرامر صفت L [1] نام دارد.
- در یک گرامر صفت L وابستگی بین ویژگی‌ها در بدنهٔ 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 ترکیبی باشند.
- همچنین ویژگی‌ها می‌توانند با شروطی به شرح زیر موروثی باشند. فرض کنید یک قانون تولید به صورت $A \rightarrow X_1\ X_2\ \cdots\ X_n$ داشته باشیم و یک ویژگی موروثی 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 $X_1\ X_2\ \cdots\ X_{i-1}$ که قبل از آن (در سمت چپ آن) قرار گرفته‌اند استفاده کند.

[1] L-attributed grammar/definitions

# گرامر صفت L

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

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

# گرامر صفت L

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

| PRODUCTION | SEMANTIC RULE |
|---|---|
| $T \to F\ T'$ | $T'.inh = F.val$ |
| $T' \to *\ F\ T'_1$ | $T'_1.inh = T'.inh \times F.val$ |

- ویژگی موروثی $\mathrm{T}'.\mathrm{inh}$ فقط از $\mathrm{F}.val$ استفاده می‌کند و F در سمت چپ $\mathrm{T}'$ در بدنه قانون تولید است.
- ویژگی $\mathrm{T}'_1.\mathrm{inh}$ در قانون دوم از ویژگی $\mathrm{T}'.\mathrm{inh}$ استفاده می‌کند که در سمت چپ قانون تولید است و همچنین از $\mathrm{F}.val$ استفاده می‌کند و F در سمت چپ $\mathrm{T}'_1$ در بدنه قانون تولید قرار دارد.

# گرامر صفت L

- مثال : گرامر نحوی–معنایی شامل قوانین زیر گرامر صفت L نیست.

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

- 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 موروثی 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 [1] 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.

[1] 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) | $L \rightarrow L_1\ ,\ \mathbf{id}$ | $L_1.inh = L.inh$<br>$addType(\mathbf{id}.entry, L.inh)$ |
| 5) | $L \rightarrow \mathbf{id}$ | $addType(\mathbf{id}.entry, L.inh)$ |

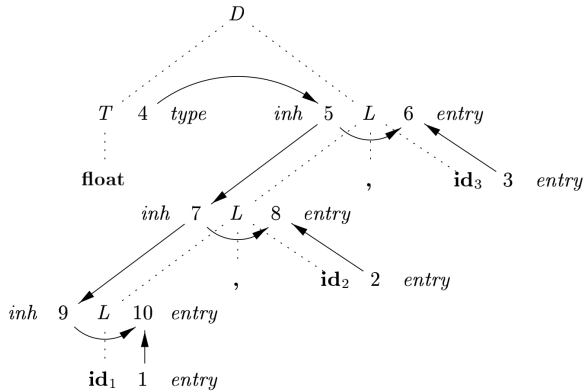
# قوانین معنایی با اثر جانبی محدود

- 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 حتما یک بار تعریف شده باشد، اما امکان تغییر گرامر برای ایجاد این بررسی وجود دارد.

| | PRODUCTION | SEMANTIC RULES |
|---|---|---|
| 1) | $D \to T\ L$ | $L.inh = T.type$ |
| 2) | $T \to \textbf{int}$ | $T.type = \text{integer}$ |
| 3) | $T \to \textbf{float}$ | $T.type = \text{float}$ |
| 4) | $L \to L_1\ ,\ \textbf{id}$ | $L_1.inh = L.inh$<br>$addType(\textbf{id}.entry, L.inh)$ |
| 5) | $L \to \textbf{id}$ | $addType(\textbf{id}.entry, L.inh)$ |

# قوانین معنایی با اثر جانبی محدود

- گراف وابستگی برای عبارت $\texttt{float}\ \texttt{id}_1, \texttt{id}_2, \texttt{id}_3$ به صورت زیر است. در رئوس ۱ و ۲ و ۳ ویژگی entry از شناسه‌ها که اشاره‌گری به جدول علائم است نشان داده شده است. در رئوس ۶ و ۸ و ۱۰ اعمال تابع addType برای به روز رسانی نوع متغیرها نشان داده شده است. نوع متغیر از T دریافت می‌شود و توسط رئوس ۵ و ۷ و ۹ و ویژگی موروثی inh به شناسه‌ها منتقل می‌شود.




| | Production | Semantic Rules |
|---|---|---|
| 1) | $D \to T\ L$ | $L.inh = T.type$ |
| 2) | $T \to \textbf{int}$ | $T.type = \text{integer}$ |
| 3) | $T \to \textbf{float}$ | $T.type = \text{float}$ |
| 4) | $L \to L_1\ ,\ \textbf{id}$ | $L_1.inh = L.inh$<br>$addType(\textbf{id}.entry, L.inh)$ |
| 5) | $L \to \textbf{id}$ | $addType(\textbf{id}.entry, L.inh)$ |

# کاربردهای ترجمه نحوی

- 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.

# کاربردهای ترجمه نحوی

- هر رأس در درخت نحوی یک شیء (با مفهوم شیء در برنامه‌نویسی شیءگرا) است.
- هر شیء دارای یک فیلد (ویژگی یا متغیر) به نام op است که برچسب آن رأس را نشان می‌دهد.
- اگر یک رأس، برگ باشد، آنگاه توکن متناظر با آن برگ را باید ذخیره کنیم. بنابراین تابع سازنده شی به صورت $Leaf(op, val)$ خواهد بود. فیلد $val$ مقدار توکن را تعیین می‌کند. مثلا اگر توکن مورد نظر یک عدد است، مقدار آن ذخیره می‌شود و اگر یک شناسه است اشاره‌گری به جدول علائم ذخیره می‌شود.
- اگر یک رأس، رأس میانی باشد، آنگاه می‌تواند تعدادی فرزند داشته باشد بنابراین باید اشاره‌گری به فرزندان آن را در شیء متناظر با آن رأس ذخیره کنیم. پس تابع سازنده به صورت $Node(op, c_1, c_2, \cdots, c_k)$ خواهد بود به طوری که $k$ تعداد فرزندان و $c_1$ تا $c_k$ اشاره‌گرهایی به فرزندان هستند.

# کاربردهای ترجمه نحوی

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

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

# کاربردهای ترجمه نحوی

- ویژگی node در این گرامر صفت اشاره‌گری به یک شیء است که رأسی را در درخت نحوی 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.

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

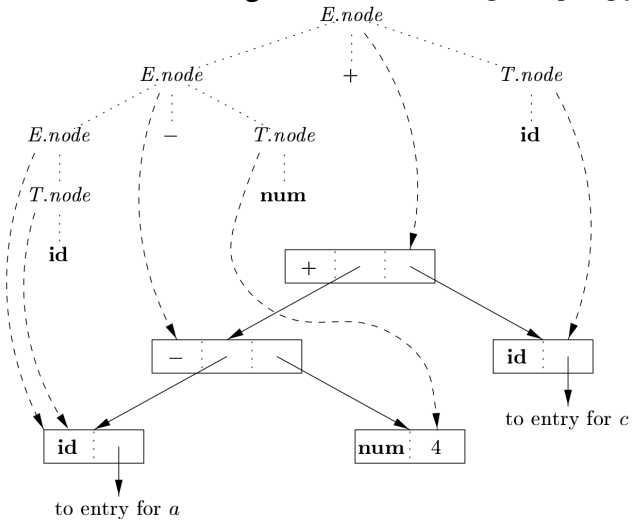
# کاربردهای ترجمه نحوی

- برای قانون $E \to T$ هیچ رأسی ساخته نمی‌شود، زیرا $E.node$ همان $T.node$ است. همچنین در قانون $T \to (E)$ 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.

| | Production | Semantic Rules |
|---|---|---|
| 1) | $E \to E_1 + T$ | $E.node = \textbf{new } Node('+', E_1.node, T.node)$ |
| 2) | $E \to E_1 - T$ | $E.node = \textbf{new } Node('-', E_1.node, T.node)$ |
| 3) | $E \to T$ | $E.node = T.node$ |
| 4) | $T \to (\ E\ )$ | $T.node = E.node$ |
| 5) | $T \to \textbf{id}$ | $T.node = \textbf{new } Leaf(\textbf{id}, \textbf{id}.entry)$ |
| 6) | $T \to \textbf{num}$ | $T.node = \textbf{new } Leaf(\textbf{num}, \textbf{num}.val)$ |

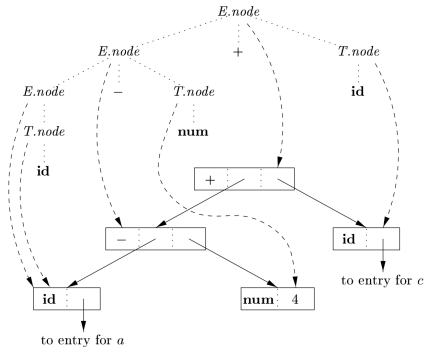
# کاربردهای ترجمه نحوی

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



# کاربردهای ترجمه نحوی

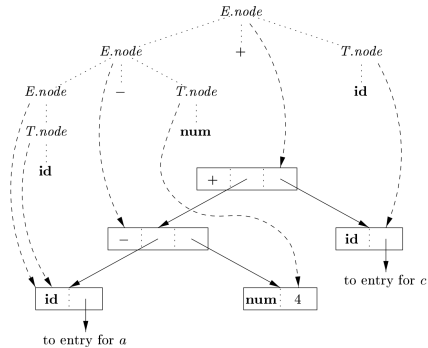
- رئوس درخت نحوی به صورت رکوردهایی نمایش داده شده‌اند که فیلد اول مقدار op را نشان می‌دهد. درخت نحوی با خطوط پررنگ و درخت تجزیه با نقطه‌چین نمایش داده شده است.
- در برگ‌های درخت مقادیر a و 4 و c دیده می‌شوند که توسط کلاس Leaf یک رأس برای هرکدام از آنها ساخته شده است.



# کاربردهای ترجمه نحوی

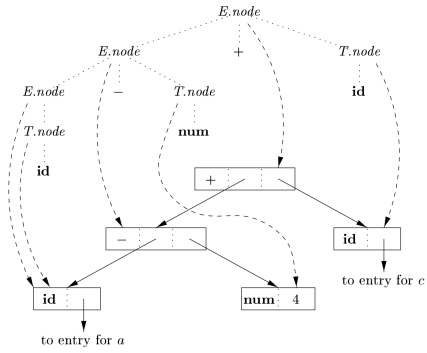
- فرض می‌کنیم `id.entry` به مکان شناسه در جدول علائم اشاره می‌کند و مقدار یک عدد در `num.val` ذخیره می‌شود. سه برگ درخت با قوانین معانی ۵ و ۶ ساخته می‌شوند. در این درخت نحوی از قوانین ۱ و ۲ برای تولید دو رأس برای عملگرهای جمع و تفریق استفاده شده است.

| | Production | Semantic Rules |
|---|---|---|
| 1) | $E \to E_1 + T$ | $E.node =$ **new** $Node('+', E_1.node, T.node)$ |
| 2) | $E \to E_1 - T$ | $E.node =$ **new** $Node('-', E_1.node, T.node)$ |
| 3) | $E \to T$ | $E.node = T.node$ |
| 4) | $T \to (\ E\ )$ | $T.node = E.node$ |
| 5) | $T \to$ **id** | $T.node =$ **new** $Leaf($**id**$,$ **id**$.entry)$ |
| 6) | $T \to$ **num** | $T.node =$ **new** $Leaf($**num**$,$ **num**$.val)$ |



# کاربردهای ترجمه نحوی

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




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

[1] postorder depth first search traversal

# کاربردهای ترجمه نحوی

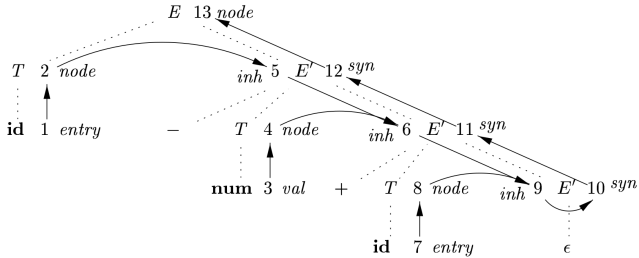
- مثال : گرامر صفت L و گرامر صفت S زیر هر دو ترجمه یکسان انجام می‌دهند.

| | Production | Semantic Rules |
|---|---|---|
| 1) | $E \to E_1 + T$ | $E.node = \textbf{new}\ Node('+', E_1.node, T.node)$ |
| 2) | $E \to E_1 - T$ | $E.node = \textbf{new}\ Node('-', E_1.node, T.node)$ |
| 3) | $E \to T$ | $E.node = T.node$ |
| 4) | $T \to (\ E\ )$ | $T.node = E.node$ |
| 5) | $T \to \textbf{id}$ | $T.node = \textbf{new}\ Leaf(\textbf{id}, \textbf{id}.entry)$ |
| 6) | $T \to \textbf{num}$ | $T.node = \textbf{new}\ Leaf(\textbf{num}, \textbf{num}.val)$ |

| | Production | Semantic Rules |
|---|---|---|
| 1) | $E \to T\ E'$ | $E.node = E'.syn$<br>$E'.inh = T.node$ |
| 2) | $E' \to +\ T\ E'_1$ | $E'_1.inh = \textbf{new}\ Node('+', E'.inh, T.node)$<br>$E'.syn = E'_1.syn$ |
| 3) | $E' \to -\ T\ E'_1$ | $E'_1.inh = \textbf{new}\ Node('-', E'.inh, T.node)$<br>$E'.syn = E'_1.syn$ |
| 4) | $E' \to \epsilon$ | $E'.syn = E'.inh$ |
| 5) | $T \to (\ E\ )$ | $T.node = E.node$ |
| 6) | $T \to \textbf{id}$ | $T.node = \textbf{new}\ Leaf(\textbf{id}, \textbf{id}.entry)$ |
| 7) | $T \to \textbf{num}$ | $T.node = \textbf{new}\ Leaf(\textbf{num}, \textbf{num}.val)$ |

# کاربردهای ترجمه نحوی

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

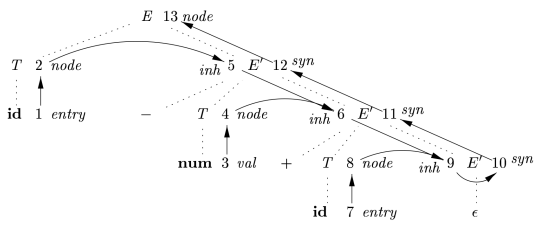



| | Production | Semantic Rules |
|---|---|---|
| 1) | $E \to T\ E'$ | $E.node = E'.syn$<br>$E'.inh = T.node$ |
| 2) | $E' \to +\ T\ E'_1$ | $E'_1.inh = \textbf{new}\ Node('+', E'.inh, T.node)$<br>$E'.syn = E'_1.syn$ |
| 3) | $E' \to -\ T\ E'_1$ | $E'_1.inh = \textbf{new}\ Node('-', E'.inh, T.node)$<br>$E'.syn = E'_1.syn$ |
| 4) | $E' \to \epsilon$ | $E'.syn = E'.inh$ |
| 5) | $T \to (\ E\ )$ | $T.node = E.node$ |
| 6) | $T \to \textbf{id}$ | $T.node = \textbf{new}\ Leaf(\textbf{id}, \textbf{id}.entry)$ |
| 7) | $T \to \textbf{num}$ | $T.node = \textbf{new}\ Leaf(\textbf{num}, \textbf{num}.val)$ |

# کاربردهای ترجمه نحوی

- متغیر $E'$ یک ویژگی موروثی $inh$ و یک ویژگی ترکیبی $syn$ دارد. ویژگی $E'.inh$ در واقع ریشه درخت برای پیشوند رشته‌ای که سمت چپ زیر درخت $E'$ است را نگهداری می‌کند.

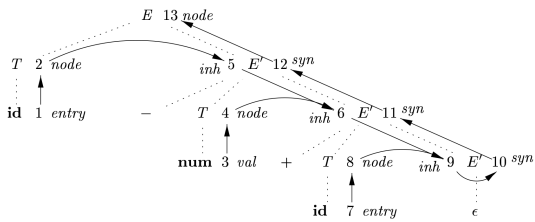
| | Production | Semantic Rules |
|---|---|---|
| 1) | $E \to T\ E'$ | $E.node = E'.syn$<br>$E'.inh = T.node$ |
| 2) | $E' \to +\ T\ E'_1$ | $E'_1.inh =$ **new** $Node('+', E'.inh, T.node)$<br>$E'.syn = E'_1.syn$ |
| 3) | $E' \to -\ T\ E'_1$ | $E'_1.inh =$ **new** $Node('-', E'.inh, T.node)$<br>$E'.syn = E'_1.syn$ |
| 4) | $E' \to \epsilon$ | $E'.syn = E'.inh$ |
| 5) | $T \to (\ E\ )$ | $T.node = E.node$ |
| 6) | $T \to$ **id** | $T.node =$ **new** $Leaf($**id**, **id**$.entry)$ |
| 7) | $T \to$ **num** | $T.node =$ **new** $Leaf($**num**, **num**$.val)$ |



# کاربردهای ترجمه نحوی

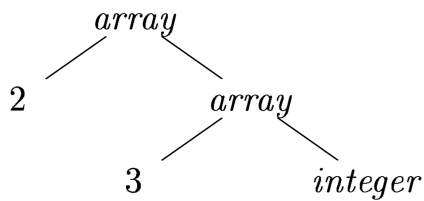
- در رأس ۵ در درخت وابستگی، $E'.inh$ ریشه درخت نحوی برای شناسه $a$ است. در رأس ۶، ویژگی $E'.inh$ ریشه درخت نحوی برای عبارت $a-4$ است. در رأس ۹ ، ویژگی $E'.inh$ درخت نحوی برای عبارت $a-4+c$ است.
- ویژگی $syn$ عبارت به دست آمده را منتقل می‌کند تا به $E.node$ برسد.

| | Production | Semantic Rules |
|---|---|---|
| 1) | $E \to T\ E'$ | $E.node = E'.syn$<br>$E'.inh = T.node$ |
| 2) | $E' \to +\ T\ E'_1$ | $E'_1.inh =$ **new** $Node('+', E'.inh, T.node)$<br>$E'.syn = E'_1.syn$ |
| 3) | $E' \to -\ T\ E'_1$ | $E'_1.inh =$ **new** $Node('-', E'.inh, T.node)$<br>$E'.syn = E'_1.syn$ |
| 4) | $E' \to \epsilon$ | $E'.syn = E'.inh$ |
| 5) | $T \to (\ E\ )$ | $T.node = E.node$ |
| 6) | $T \to$ **id** | $T.node =$ **new** $Leaf($**id**, **id**$.entry)$ |
| 7) | $T \to$ **num** | $T.node =$ **new** $Leaf($**num**, **num**$.val)$ |



# کاربردهای ترجمه نحوی

- مثال : فرض کنید نوع $\mathrm{int}[2][3]$ آرایه‌ای است تشکیل شده از دو آرایه به طوری که هرکدام آن آرایه‌ها سه عدد صحیح را شامل می‌شوند.
- این آرایه را می‌توانیم به صورت $\mathrm{array}(2, \mathrm{array}(3, \mathrm{integer}))$ نمایش دهیم و یک درخت تجزیه به صورت زیر برای آن تولید کنیم.




- عملگر $\mathrm{array}$ دو پارامتر دریافت می‌کند. پارامتر اول تعداد عناصر آرایه و پارامتر دوم نوع عناصر آرایه را تعیین می‌کند.

# کاربردهای ترجمه نحوی

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

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

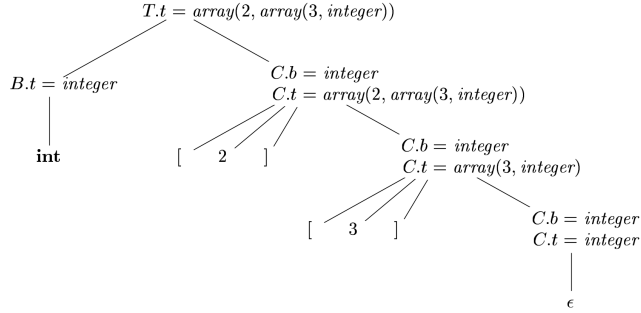
# کاربردهای ترجمه نحوی

- متغیر گرامر T می‌تواند یک متغیر برنامه از نوع اصلی (غیر آرایه) تولید کند، یا یک متغیر برنامه از نوع آرایه. متغیر B می‌تواند نوع‌های اصلی `int` یا `float` را تولید کند. یک متغیر برنامه از نوع اصلی وقتی تولید می‌شود که T مشتق کند BC و C رشته تهی را مشتق کند. اگر C تهی مشتق نکند یک متغیر برنامه از نوع آرایه تولید می‌شود.
- متغیرهای B و T یک ویژگی ترکیبی به نام t دارند که نوع را مشخص می‌کند.
- متغیر C دو ویژگی دارد : یک ویژگی موروثی b و یک ویژگی ترکیبی t . ویژگی b نوع اصلی را در درخت تجزیه به پایین منتشر می‌کند و ویژگی t نتیجه را به دست می‌دهد.

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

# کاربردهای ترجمه نحوی

- درخت تجزیه حاشیه‌گذاری شده برای ورودی `int [2] [3]` در شکل زیر نمایش داده شده است.




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

# کاربردهای ترجمه نحوی

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

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

# کاربردهای ترجمه نحوی

- حال یک مثال دیگر را در نظر می‌گیریم. در این مثال می‌خواهیم در یک زبان برنامه نویسی، عبارت‌های تخصیص مقدار داشته باشیم. 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 :

$assign \rightarrow var = expr$
$expr \rightarrow var + var \mid var$
$var \rightarrow \texttt{id}$

# کاربردهای ترجمه نحوی

- حال برای متغیرهای این گرامر، دو ویژگی در نظر می‌گیریم. ویژگی نوع واقعی (actual-type) و ویژگی نوع مورد انتظار (expected-type).
- نوع واقعی : هرکدام از متغیرهای $var$ و $expr$ در گرامر، یک ویژگی ترکیبی دارند که برای ذخیره نوع آنها (که int یا real است) به کار می‌رود. ویژگی متغیر $var$ از ویژگی ترمینال `id` از جدول علائم محاسبه می‌شود و ویژگی متغیر $expr$ بر اساس ویژگی فرزندان آن به دست می‌آید.
- نوع مورد انتظار : نوع مورد انتظار، یک ویژگی موروثی برای متغیر $expr$ است.

# کاربردهای ترجمه نحوی

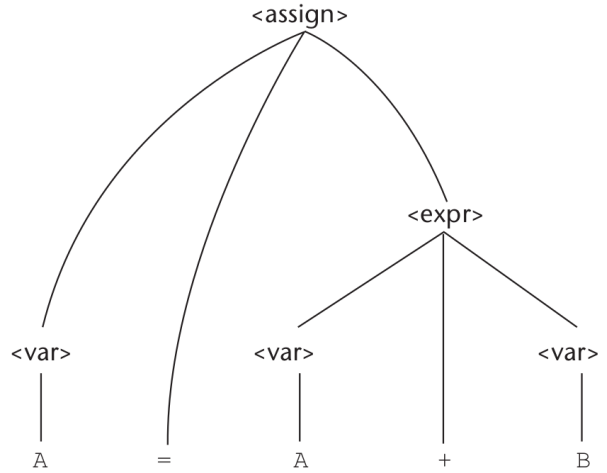
- گرامر صفت برای مثال قبلی را به صورت زیر می‌نویسیم.

1. Syntax rule : $assign \rightarrow var = expr$
Semantic rule : $expr$.expected-type = $var$.actual-type
2. Syntax rule : $expr \rightarrow var1 + var2$
Semantic rule : $expr$.actual-type = if ( $var1$.actual-type == int )
and ( $var2$.actual-type == int )
then int
else real
and if
if ( $expr$.actual-type != $expr$.expected-type ) error()
3. Syntax rule : $expr \rightarrow var$
Semantic rule : $expr$.actual-type = $var$.actual-type
if ( $expr$.actual-type != $expr$.expected-type ) error()
4. Syntax rule : $var \rightarrow$ **id**
Semantic rule : $var$.actual-type = type-lookup ( `id` )

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

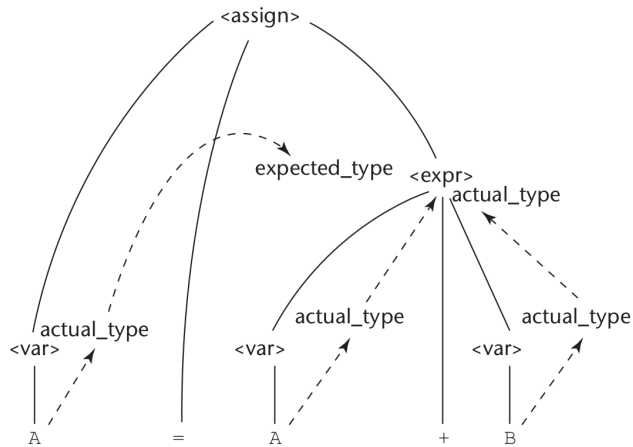
# کاربردهای ترجمه نحوی

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



# کاربردهای ترجمه نحوی

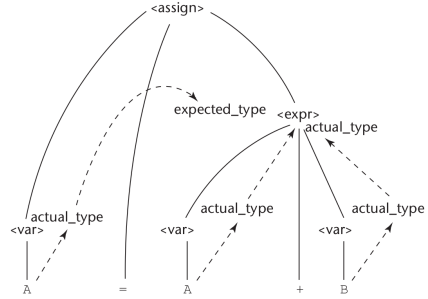
- شکل زیر نحوهٔ محاسبه ویژگی‌ها در درخت تجزیه را نشان می‌دهد. نوع واقعی (actual-type) یک ویژگی ترکیبی است، درحالی‌که نوع مورد انتظار (expected-type) یک ویژگی موروثی است.



# کاربردهای ترجمه نحوی

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

1. *var*.actual-type = type-lookup (A) (Rule 4)
2. *expr*.expected-type = *var*.actual-type (Rule 1)
3. *var1*.actual-type = type-lookup (A) (Rule 4)
4. *var2*.actual-type = type-lookup (B) (Rule 4)
5. *expr*.actual-type = either int or real (Rule 2)
6. if ( *expr*.expected-type != *expr*.actual-type ) error() (Rule 2)



# شمای ترجمه نحوی

- شمای ترجمه نحوی [1] روشی برای نشانه‌گذاری گرامر نحوی–معنایی است.
- شمای ترجمه نحوی، یک گرامر مستقل از متن است که قوانین معنایی در بدنه قوانین تولید آن تعبیه شده اند.
- قوانین معنایی که در شمای ترجمه نحوی تعبیه می‌شوند، عملیات معنایی [2] نامیده می‌شوند و می‌توانند در هر قسمتی از بدنه قوانین تولید قرار بگیرند.
- به طور قراردادی قبل و بعد از عملیات معنایی از کاراکتر آکولاد باز و بسته استفاده می‌شود و اگر در قوانین نحوی کاراکتر آکولاد داشتیم، دو طرف آکولاد علامت آپوستروف قرار می‌دهیم.
- 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 عملیات معنایی را اجرا کنیم.

[1] syntax-directed translation scheme
[2] semantic action

# شمای ترجمه پسوندی

- 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 بگیرید، به آن شمای ترجمه پسوندی [1] گفته می‌شود.

[1] postfix translation scheme

# شمای ترجمه پسوندی

- مثال : شمای ترجمه پسوندی زیر، برای عبارات محاسباتی ساده ریاضی به کار می‌رود.

$$
\begin{array}{lcll}
L & \rightarrow & E\ \mathbf{n} & \{\ \mathrm{print}(E.val);\ \} \\
E & \rightarrow & E_1 + T & \{\ E.val = E_1.val + T.val;\ \} \\
E & \rightarrow & T & \{\ E.val = T.val;\ \} \\
T & \rightarrow & T_1 * F & \{\ T.val = T_1.val \times F.val;\ \} \\
T & \rightarrow & F & \{\ T.val = F.val;\ \} \\
F & \rightarrow & (\ E\ ) & \{\ F.val = E.val;\ \} \\
F & \rightarrow & \mathbf{digit} & \{\ F.val = \mathbf{digit}.lexval;\ \}
\end{array}
$$

# شمای ترجمه پسوندی

- گرامر نحوی–معنایی معادل این شمای ترجمه نحوی در زیر آمده است.

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

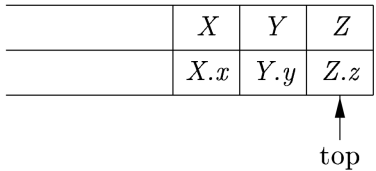
- از آنجایی که این گرامر LR است و گرامر نحوی–معنایی گرامر صفت S است، عملیات معنایی می‌توانند در حین تجزیه در زمان کاهش قوانین اجرا شوند.

# شمای ترجمه پسوندی

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

# شمای ترجمه پسوندی

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




- نمادهای XYZ برروی 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 ترکیبی 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، متغیر A با تمام ویژگی‌هایش برروی پشته قرار می‌گیرد.

# شمای ترجمه پسوندی

- شمای ترجمه پسوندی زیر برای عبارات محاسباتی طراحی شده است. در عملیات معنایی، محتوای پشته مستقیماً دستکاری می‌شود.

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

# 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 $X$ برروی پشته قرارگرفت انجام می‌شود.
- اگر تجزیه بالا به پایین باشد، عملیات $a$ قبل از اینکه $Y$ پردازش شود، انجام می‌شود.

# شمای ترجمه با عملیات معنایی در بدنه قانون

- روشی که به آن اشاره کردیم برای پیاده‌سازی شمای ترجمه پسوندی در حین فرایند تجزیه پایین به بالا بود. توجه کنید که برخی از انواع شماهای ترجمه را نمی‌توان در حین تجزیه پیاده‌سازی کرد.
- با این حال، همهٔ شماهای ترجمه را می‌توان به صورت زیر پیاده‌سازی کرد.

۱. با چشم‌پوشی از عملیات معنایی، ورودی را تجزیه کرده یک درخت تجزیه تولید می‌کنیم.

۲. همهٔ 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 عمق‌اول برروی درخت انجام می‌دهیم و هنگامی که یک رأس با یک عملیات معنایی در درخت پیمایش می‌شود، عملیات معنایی متناظر با آن اجرا می‌شود.

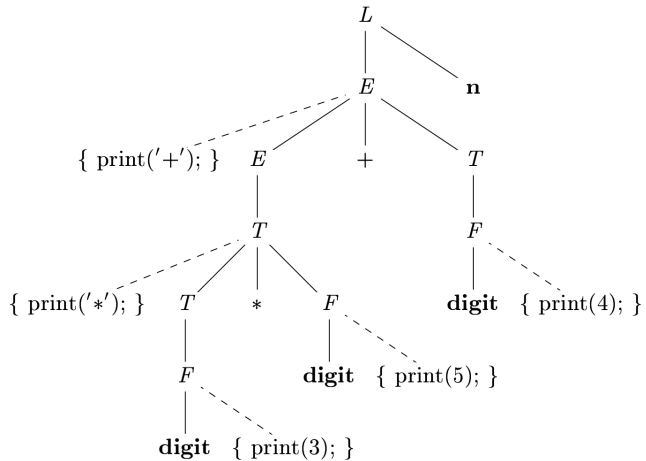
# شمای ترجمه با عملیات معنایی در بدنه قانون

- مثال : شمای ترجمه زیر را در نظر بگیرید. این شمای ترجمه عبارات میانوندی را به پیشوندی تبدیل می‌کند.

$$
\begin{array}{lllll}
1) & L & \rightarrow & E\ \mathbf{n} \\
2) & E & \rightarrow & \{\ \mathrm{print}('+');\ \}\ \ E_1 + T \\
3) & E & \rightarrow & T \\
4) & T & \rightarrow & \{\ \mathrm{print}('*');\ \}\ \ T_1 * F \\
5) & T & \rightarrow & F \\
6) & F & \rightarrow & (\ E\ ) \\
7) & F & \rightarrow & \mathbf{digit}\ \ \{\ \mathrm{print}(\mathbf{digit}.\mathit{lexval});\ \}
\end{array}
$$

# شمای ترجمه با عملیات معنایی در بدنه قانون

- درخت تجزیه برای عبارت $4 + 5 * 3$ به صورت زیر خواهد بود. با پیمایش درخت تجزیه عبارت $4\ 5\ 3\ * +$ 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.
- قبلا اشاره کردیم که برای حذف بازگشت چپ در قوانینی به شکل $A \rightarrow A\alpha|\beta$ می‌توانیم آنها 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.

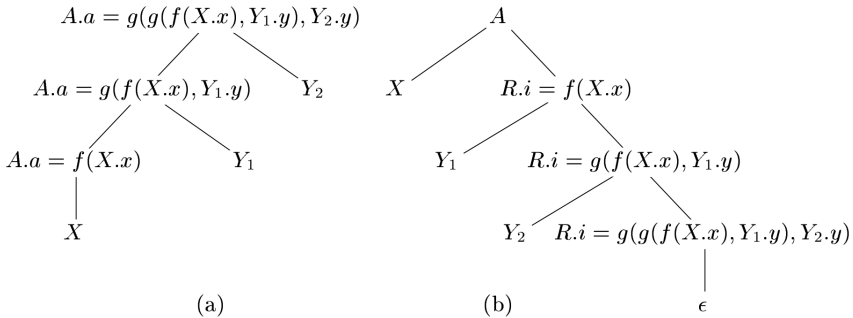
$$\begin{array}{lcl} A & \rightarrow & A_1\ Y\ \{A.a = g(A_1.a, Y.y)\} \\ A & \rightarrow & X\ \{A.a = f(X.x)\} \end{array}$$

- در اینجا A.a یک ویژگی ترکیبی است و A 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 پس از حذف بازگشت چپ چگونه باید باشد.




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

$$A \rightarrow X\ R$$
$$R \rightarrow Y\ R \mid \epsilon$$

# حذف بازگشت چپ از شمای ترجمه

- تابع $f$ یک بار اعمال می‌شود که مربوط به قانون تولید $A \to X$ است و سپس $g$ به تعداد باری که قانون $A \to AY$ اعمال می‌شود باید اعمال شود.
- با هربار اعمال قانون $R \to YR$ تابع $g$ یک بار اعمال می‌شود.
- برای $R$ یک ویژگی موروثی $R.i$ تعریف می‌کنیم که نتیجه اعمال تابع $g$ را جمع‌آوری می‌کند.
- همچنین $R$ باید یک ویژگی ترکیبی $R.s$ را داشته باشد. این ویژگی وقتی $R$ تولید نماد $Y$ را به اتمام می‌رساند محاسبه می‌شود. سپس $R.s$ توسط ویژگی‌های ترکیبی در درخت به سمت بالا منتقل می‌شود تا به $A.a$ برسد.

# حذف بازگشت چپ از شمای ترجمه

- بنابراین اگر دو قانون تولید به صورت زیر داشته باشیم

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

- پس از حذف بازگشت چپ، گرامر به صورت زیر در می‌آید.

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

- دقت کنید که ویژگی موروثی R.i قبل از R در بدنه محاسبه می‌شود و ویژگی‌های ترکیبی A.a و R.s در پایان قانون ارزیابی می‌شوند. بدین ترتیب ویژگی‌های 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 موروثی برای متغیر $A$ را قبل از متغیر $A$ در بدنه قانون تولید قرار می‌دهیم.

۲. عملیات معنایی برای ویژگی‌های ترکیبی را در انتهای قوانین تولید قرار می‌دهیم.

# ترجمه نحوی برای گرامر صفت L

- مثال : زبان Eqn زبانی است که قبل از زبان TEX برای حروف چینی مستندات استفاده می‌شد.
- در این زبان برای نوشتن عبارت $a_{i_j}$ می‌نویسیم (a **sub** (i **sub** j).
- می‌توانیم گرامری به صورت زیر برای توصیف این عبارات بنویسیم.

$$B \rightarrow B_1\ B_2 \mid B_1\ \textbf{sub}\ B_2 \mid (\ B_1\ ) \mid \textbf{text}$$

- این گرامر جعبه‌هایی [1] تولید می‌کند که هریک می‌توانند زیرنویس دیگری باشند و یا در کنار یکدیگر قرار بگیرند.

[1] 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 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.

# ترجمه نحوی برای گرامر صفت L

- در شکل زیر دو جعبه برای $E_1$ و height. نشان داده شده‌اند. دو جعبه $E_1$ و height. به یکدیگر متصل شده و جعبه $E_1$.height 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 ۱۰ است و اگر اندازه یک جعبه p [1] باشد، اندازه جعبه زیرنویس 0.7p است. برای متغیر B در گرامر، ویژگی B.ps اندازه آن را نشان می‌دهد. این ویژگی یک ویژگی موروثی است زیرا اندازه جعبه زیرنویس از اندازه جعبه اصلی به دست می‌آید.

۲. هر جعبه یک خط پایه [2] دارد به طوری که حروف برروی 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) | $S \to B$ | $B.ps = 10$ |
| 2) | $B \to B_1\ B_2$ | $B_1.ps = B.ps$<br>$B_2.ps = B.ps$<br>$B.ht = \max(B_1.ht, B_2.ht)$<br>$B.dp = \max(B_1.dp, B_2.dp)$ |
| 3) | $B \to B_1\ \textbf{sub}\ B_2$ | $B_1.ps = B.ps$<br>$B_2.ps = 0.7 \times B.ps$<br>$B.ht = \max(B_1.ht, B_2.ht - 0.25 \times B.ps)$<br>$B.dp = \max(B_1.dp, B_2.dp + 0.25 \times B.ps)$ |
| 4) | $B \to (\ B_1\ )$ | $B_1.ps = B.ps$<br>$B.ht = B_1.ht$<br>$B.dp = B_1.dp$ |
| 5) | $B \to \textbf{text}$ | $B.ht = getHt(B.ps, \textbf{text}.lexval)$<br>$B.dp = getDp(B.ps, \textbf{text}.lexval)$ |

# 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 موروثی از یک متغیر باید قبل از آن متغیر در شمای ترجمه قرار بگیرند.

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

# ترجمه نحوی برای گرامر صفت L

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

# ترجمه نحوی برای گرامر صفت L

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

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

- 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.

# ترجمه نحوی برای گرامر صفت L

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

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

# ترجمه نحوی برای گرامر صفت L

- ویژگی‌های به کاربرده شده در این گرامر نحوی–معنایی به شرح زیراند.

۱. ویژگی $S.next$ برچسب کدی است که بعد از S باید اجرا شود و ویژگی $C.code$ کدهای میانی متعلق به عبارت C است.

۲. ویژگی $C.true$ برچسب کدی است که باید اجرا شود اگر C درست باشد و ویژگی $C.false$ برچسب کدی است که باید اجرا شود اگر C نادرست باشد.

۳. ویژگی ترکیبی $S.code$ دنباله‌ای از کدهای میانی معادل S را ذخیره می‌کند.

# ترجمه نحوی برای گرامر صفت L

- تابع `new` برچسب جدید تولید می‌کند.
- مقادیر L1 و L2 برچسب‌هایی که در کدمیانی نیاز داریم را ذخیره می‌کنند. مقدار L1 برچسب ابتدای کدی است که بعد از اتمام دستور `while` اجرا می‌شود. بنابراین $S_1.next$ برابراست با L1 . مقدار L2 برچسب ابتدای کد $S_1$ است. بنابراین مقدار $C.true$ برابراست با L2.
- مقدار $C.false$ برابراست با $S.next$ زیرا اگر شرط نادرست باشد، باید کدی که به دنبال S می‌آید اجرا شود.
- از نماد || برای الحاق قطعات کد استفاده می‌شود.

# ترجمه نحوی برای گرامر صفت L

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

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

# 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 را پیمایش می‌کنیم.