1- مقدمه

پرولوگ یک زبان برنامه نویسی است که بر اساس مجموعه ای از مکانیسم ها همانند یکسان سازی الگوها[1]، ساختارهای داده درختی[2]، و عقب گرد اتوماتیک[3]، کار میکند. این مجموعه کوچک از مکانیسم ها، خوشبختانه یک محیط برنامه نویسی قدرتمندی را فراهم می نمایند. پرولوگ در سیستمهایی که اشیاء (و مخصوصا اشیاء ساختاری) و روابط بین آنها مطرح هستند کارآیی مناسبی دارد. مثلا بیان قوانینی همانند " اگر شیء X بالاتراز شیء Y و شیء Y بالاتر از شیء Z باشد، آنگاه شیء X بالاتر از شیء Z است" کار بسیار راحتی در پرولوگ است. چنین قابلیتهایی پرولوگ را به یک زبان برنامه نویسی قدرتمند برای سیستمهای هوش مصنوعی (و غیر عددی) تبدیل کرده است. مثالهای مختلفی وجود دارند که پیاده سازی آنها در زبانهای استاندارد دیگر دهها صفحه کد دارد در حالیکه در پرولوگ فقط در یک صفحه پیاده سازی می شوند.

پرولوگ بر گرفته از اصطلاح *برنامه نویسی منطقی*[4] است که در سال 1970 مطرح شده و هدف از آن استفاده از*منطق* بعنوان یک زبان برنامه نویسی بود. جدیدترین گسترش های پرولوگ شامل *برنامه نویسی منطقی محدودیتها*[5] *(CLP)* است. در روش CLP امکان بیان و پردازش محدودیتها وجود دارد. نشان داده شده است که این متد برنامه نویسی دارای قدرت زیادی در مسائل زمانبندی و برنامه ریزی منطقی دارد.

از آنجائیکه پرولوگ ریشه در منطق ریاضی دارد، معمولا آموزش آن توأم با آموزش منطق صورت میگیرد. اگر چه برای آموزش برنامه نویسی پرولوگ نیاز زیادی به مباحث گسترده منطق ریاضی نیست. لذا در این متن اشاره چندانی به منطق ریاضی نشده و درعوض تکنیکهایی که در حل مسائل توسط پرولوگ استفاده می شود، توضیح داده خواهد شد. در حالیکه اکثر زبانهای برنامه نویسی مرسوم مبتنی بر روش رویه ای[6] (امری) هستند، پرولوگ مبتنی بر روش توصیفی[7] است. در این روش نوع جدیدی از تفکر برنامه نویسی مطرح میشود که متفاوت از روش امری است و باعث میشود برنامه نویسی در پرولوگ جالب و جاذب باشد.

2- تعریف روابط توسط واقعیتها

پرولوگ یک زبان برنامه نویسی برای حل مسائل غیر عددی و سمبولیک است و مناسب مسائلی است که با اشیاء سر و کار دارند.به عنوان مثال این را که tom پدرbob است با استفاده از رابطه خانوادگی parent به صورت مقابل می توان نشان داد:

```
parent (tom, bob)
```

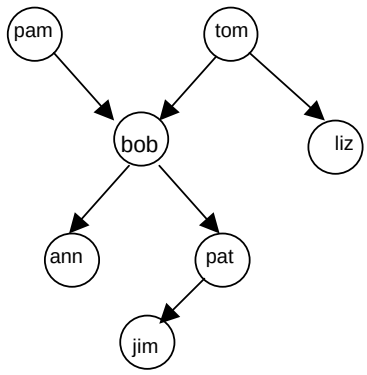
در اینجا parent نام رابطه است و tom و bob آرگومانهای آن هستند. درخت شکل 1 که روابط خانوادگی والد[8] را مابین برخی افراد نشان می دهد در پرولوگ به صورت زیر تعریف می شود:

```
parent (pam, bob).
```

---

[1] Pattern matching
[2] Tree based data structures
[3] Automatic backtracking
[4] Programming in Logic
[5] Constraint Logic Programming
[6] Procedural
[7] Declarative
[8] Parent

```
parent (tom, bob).
parent (tom, liz).
parent (bob, ann).
parent (bob, pat).
parent (pat, jim).
```



شکل 1- درخت فامیلی

درخت شکل 1 با شش ***کلاوز***[9] نشان داده شده است. کلاوزها دارای انواع دیگری نیز هستند که بعدا توضیح داده خواهند شد. کلاوزهای بالا از نوع ***واقعیت***[10] است که هر کدام یک نمونه از رابطة والد (parent) را نشان میدهند. یک راه کلی برای تعریف یک رابطه در پرولوگ تعریف نمونه های آن رابطه (کلاوز هایی از نوع واقعیت) می باشد.
برای پرسیدن سوال در پرولوگ از کلاوزهای نوع ***سوالی***[11] استفاده می شود. در پرولوگ می توان از رابطه بالا سوال به صورت زیرکرد:

```
?- parent (tom, bob).
```

این سوال یعنی آیا tom پدر bob است؟ جواب پرولوگ به سوال با توجه به حقیقت های تعریف شده در بالا، yes خواهد بود و اگر سوال جزو حقیقت هایمان نباشد، جواب no خواهد بود. می توان با استفاده از پرولوگ از این حقیقت ها سوال دیگری به صورت زیر پرسید که در واقع می گوید پدر jim چه کسی است که جواب pat خواهد بود:

```
?- parent(X, jim)
```

جواب پرولوگ: X= pat

بعضی اوقات ممکن است یک سوال بیش از یک جواب داشته باشد: (چه کسانی فرزند bob هستند؟)

```
?- parent (bob,X).
```

اولین جواب پرولوگ: X= ann

ما ممکن است جوابهای دیگر را نیز بخواهیم (با زدن **;**) که در اینصورت پرولوگ جواب میدهد:

دومین جواب پرولوگ: X= pat

---

[9] Clause
[10] Fact
[11] Question

البته سوال فوق می تواند دو مجهول داشته باشد که در این صورت تمام واقعیتها چاپ خواهد شد. سوال زیراز پرولوگ می پرسد که " تمام X ها و Y ها را پیدا کن که X والدY است.

```
?- parent(X,Y).
X=pam          جواب اول
Y=bob ;
X=tom          جواب دوم
Y=bob;
X=tom          جواب سوم
Y=liz;
...            ...
```

پرولوگ در جواب سوال فقط یک جواب را نشان میدهد. برای چاپ جواب بعدی از علامت ; استفاده می کنیم. برای توقف جوابها به جای RETURN را وارد می کنیم. سوالات می توانند مرکب نیز باشند. بعنوان مثال برای پرسیدن اینکه والد بزرگ[12] Jim چه کسی است میتوان نوشت:(اگروالد Jim، Y باشد آنگاه چه کسی والد Y است؟)

```
? – Parent (Y, jim), parent (X, Y).
X=bob
Y=pat
```

چند نکته مهم:

1- در پرولوگ تعریف یک رابطه نظیر رابطه parent به سادگی و با مشخص کردن n شی که اعضای رابطه هستند صورت می گیرد.

2- کاربر به سادگی می تواند از سیستم پرولوگ درباره رابطه تعریف شده در برنامه سوال نماید.

3- یک برنامه پرولوگ از چند عبارت تشکیل شده است. هر عبارت با یک نقطه پایان می یابد.

4- آرگومانهای روابط می توانند اشیاء ثابتها (نظیر ann , tom) و یا اشیاء عمومی نظیر X , Y باشند. اشیاء نوع اول اتم و اشیاء نوع دوم متغیر نامیده می شوند.

5- سوالاتی که از سیستم پرسیده می شود از یک یا چند هدف تشکیل شده است.

6- یک پاسخ به سوال می تواند بسته به اینکه هدف مورد نظر قابل دستیابی است یا خیر ، مثبت یا منفی باشد. اگر پاسخ مثبت باشد یعنی هدف مورد نظر قابل دسترسی بوده است.

7- اگر چندین جواب وجودداشته باشد پرولوگ به درخواست کاربر آنها را پیدا خواهد کرد.

3- تعریف روابط بوسیله قواعد

مثال بالا از جهات متعددی می تواند گسترش یابد. به عنوان مثال برای رابطه Parent رابطه عکس آن یعنی offspring را اگر با واقعیت ها بیان کنیم باید نمونه های مختلف مثل offspring(bob, tom) را تعریف کرد. در صورتی که میتوان آن را به صورت فرمول نویسی پرولوگ به صورت زیر تعریف کرد:

```
offspring(Y , X): - parent (X , Y)
```

که در واقع این دستور بیان میکند که برای تمام X ها وYها، Y فرزند X [13]است اگر X والد Y باشد.

[12] Grandparent
[13] Offspring

در پرولوگ به فرمول بالا قاعده گفته می شود. قاعده دارای دو قسمت، سر[14] و بدنه[15] است. سر و بدنه قاعده در زیر نشان داده شده اند.

$$\underbrace{\text{offspring(Y , X)}}_{\textbf{سر}}\text{: - }\underbrace{\text{parent (X , Y)}}_{\textbf{بدنه}}$$

در واقع درعبارت بالا اگر parent (X , Y) صحیح باشد آنگاه نتیجه منطقی این است که offspring (Y , X) صحیح است. بعداز تعریف قانون بالا می توان از برنامه سوال کرد که offspring(Liz, tom) درست است یا نه؟ در این صورت پرولوگ می تواند بررسی کند که آیا parent(tom, liz) درست است یا نه. در واقع *بدنهٔ* یک قاعده نشان دهنده شرط (هایی) است که باعث استنتاج قسمت *سر* می شوند. *بدنه* می تواند خالی بوده یا بیش از یک عبارت داشته باشد. در واقع اگر بدنه یک قاعده خالی باشد آن قاعده به *واقعیت (fact)* تبدیل میشود. در ضمن ممکن است چند قاعده داشته باشیم که دارای *سر*های همانند باشند. چنین حالتی نشان میدهد که برای رسیدن به آن *سر* راههای مختلفی وجود دارد. مثال زیر نشان می دهد که یک شیء تیره است اگر سیاه یا قهوه ای باشد.

```
dark(X):- black(X).
dark(X):- brown(X).
```

نکات مهم:

1. پرولوگ به ما آزادی کامل را برای انتخاب ساختار برنامه می دهد.
2. برنامه های پرولوگ با اضافه کردن عبارتهای جدید گسترش می یابند.
3. *کلاوزهای* پرولوگ سه نوع می باشند: *واقعیت ها* ، *قواعد* ، *سوالات*.
4. *واقعیت ها* عباراتی هستند که همواره صحیح می باشند.
5. *قواعد* عباراتی هستند که بسته به شروط خاصی می توانند صحیح باشند.
6. *قواعد* پرولوگ از دو قسمت *سر* و *بدنه* تشکیل شده اند. بدنه از لیست هدف ها تشکیل شده که با کاما از هم جدا شده اند. کاما به معنی اتصال می باشد.
7. در هنگام محاسبات، یک متغیر می تواند با یک شیء جایگزین شود و می گوییم که متغیر مقدارگرفته است.
8. توضیحات در پرولوگ در یک خط با قرار گرفتن % در اولش یا در چند خط و بین /* ... */ قابل نوشتن است.

4 قوانین بازگشتی

اگر بخواهیم در یک درخت خانوادگی رابطه جد[16] را تعریف کنیم، می توان آن را بر اساس رابطه والد تعریف نمود. در ساده ترین حالت X جد Y است اگر X والد Y باشد. در این حالت میتوان قانونی به شکل زیر تعریف کرد:

```
predecessor(X, Z):- parent (X, Z).
```

این تعریف فقط به یک جد (که همان والد باشد) دسترسی دارد و به جدهای بالاتر دسترسی ندارد. یک روش این است که قواعدی به صورتهای زیر تعریف کرد که شامل جدهای مرتبه بالاتر باشد:

```
predecessor(X, Z):- parent (X, Y), parent(Y, Z).
predecessor(X, Z):- parent (X, Y), parent(Y, U), parent(U, Z).
```

[14] Head
[15] Body
[16] Predecessor

...

در اینجا، در بدنه قواعد به تعدادی که می خواهیم به اجداد بالا دسترسی داشته باشیم عبارت parent را می نویسیم ولی در این حالت رابطه جد تا مرتبه محدودی تعریف می شود.

یک راه صحیح فرموله کردن وجود داردکه تا هر عمقی جواب می دهد . برای تمام X ها وZ ها, X جد Z است اگر X والد Z باشد یا یک Y وجود داشته باشد که

1 – X والد Y باشد،
2- Y جد Z باشد.

```
predecessor(X, Z) :- parent (X, Z).
predecessor(X, Z) :- parent (X, Y), predecessor(Y, Z).
```

در حقیقت برنامه نویسی بازگشتی یکی از مفاهیم اساسی در برنامه نویسی پرولوگ است.

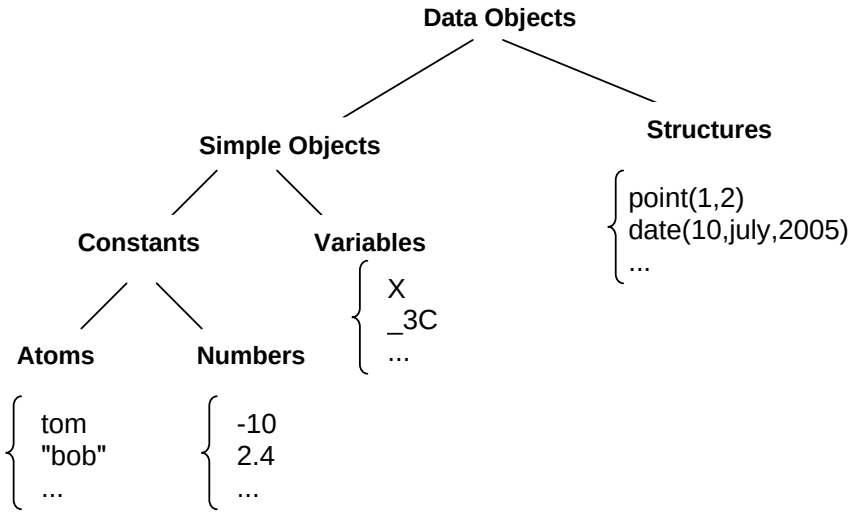
1-4 چگونه پرولوگ به سوالات پاسخ می دهد

یک سوال برای پرولوگ همیشه مجموعه ای از یک یا چند هدف است. پرولوگ برای جواب دادن به یک سوال سعی می کند اهداف را برآورده سازد. یعنی اینکه فرض می کند روابط برنامه صحیح است و از آنها به اهداف برسد. همچنین برای متغیرها نمونه سازی می کند اگر نمونه ای پیدا نشود جواب پرولوگ " خیر " خواهد بود.

پرولوگ از هدف برنامه شروع می کند و با استفاده از قوانین ، اهداف جاری را با اهداف جدید جایگزین کرده تا به جایی برسد که یک هدف یک واقعیت ساده باشد. پس سعی می کند که این هدف را برآورده سازد. برای این کار سعی می کند در برنامه عبارتی را پیدا کند که از آن عبارت هدف فوق را بتوان نتیجه گرفت. در واقع مثل اینکه در یک درخت حرکت می کند.

5 اشیاء داده ای در پرولوگ

طبقه بندی اشیاء داده در پرولوگ به صورت روبرو است :



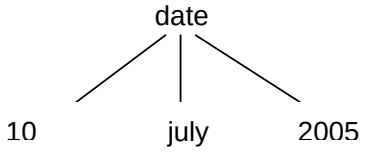
1-5 اتمها و اعداد

یک اتم می تواند رشته ای از حروف، ارقام و کاراکتر '_' باشد که با حرف کوچک شروع شود (مثل tom). همچنین یک اتم میتواند رشته ای از کاراکتر های خاص مثل +، -، *، /، >، < و غیره باشد. یک اتم همچنین می تواند رشته ای از کاراکترهایی که در داخل علائم " " قرار می گیرند باشد مثل "bob".

اعداد در پرولوگ میتوانند صحیح، طبیعی و اعشاری باشند. به دلیل اینکه پرولوگ در درجه اول زبان علائمی است و نه زبان عددی، اعداد چندان در برنامه نویسی عادی پرولوگ مورد استفاده قرار نمی گیرند. البته در محاسبه علائمی مثل شمارش آیتمهای یک لیست اغلب اعداد صحیح مورد استفاده قرار می گیرند.

متغیر، رشته ای از حروف، اعداد و کاراکتر '_' است که یا با یک حرف بزرگ و یا با کاراکتر _ شروع می شود مثل X یا *2xM یا* M_4. بعضی اوقات می توان از علامت _ بعنوان متغیری استفاده کرد که نشانگر یک متغیر بی نام جدید است. در یک عبارت سوالی اگر از کاراکتر خط زیر(_) استفاده بکنیم مقدار آن جزو خروجی نخواهد بود. حوزه یک متغیر، به محدوده ای گفته میشود که مقدار آن متغیر در آن محدوده قابل استفاده است. در پرولوگ معمولاً حوزه یک متغیر از ابتدا تا انتهای کلاوزی است که از ان متغیر استفاده می کند.

ساختارها[17]، اشیایی هستند که چندین جزء دارند که هر جزء به خودی خود می تواند یک ساختار باشد. مثل date(10,july, 2005).

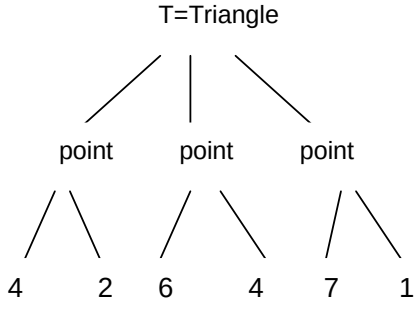


اگر یک نام، در دو نقش مختلف در یک برنامه ظاهر شود، مثلاً point (X,Y,Z), point (X1,Y1) سیستم پرولوگ آن را با تعداد آرگومانهای متفاوت تشخیص می دهد، و این نام را به عنوان دو اجرا کننده تفسیر می کند. هر اجرا کننده با دو چیز تعریف می شود:

1- نام، که ساختار دستوری آن همانند اتمهاست.

2- اریتی[18] ، یعنی تعداد آرگومانها

مثال از نمایش درختی یک شی T=triangle(point (4,2),point(6,4) , point(7,1))



[17] Structures
[18] Arity

6 تطابق

پرولوگ برای جواب دادن به سوالات، از تکنیکی به نام تطابق[19] استفاده می کند. دو عبارت در پرولوگ با هم تطابق دارند اگر با هم برابر باشند، یا متغیرهای هر دو عبارت را بتوان به گونه ای مقدار دهی کرد که پس از جایگزینی متغیرها توسط اشیاء، عبارات حاصل با هم مساوی باشند. به عنوان مثال دو عبارت date (D, M, 2001) و date(D1, may, Y1) با هم در صورتی که جایگذاری های D=D1 , M=may , Y1=2001 انجام شود، تطابق دارند. قوانین عمومی برای تصمیم گیری اینکه آیا دو عبارت S , T با هم تطابق دارند به صورت زیر است:

1- اگرS و T ثابت باشند، در صورتی با هم تطابق دارند که هر دو یک شیء باشند.

2- اگر S متغیر و T هر چیزی باشد، پس با هم تطابق دارند و S مقدارش برابرT می شود.

3- اگر S و T ساختار باشند، تنها در حالتی تطابق خواهند داشت که:

الف) S , T اجرا کننده اصلی همگونی داشته باشند.

ب) تمام اجزاء آنها نظیر به نظیر با هم تطابق داشته باشند.

به مثال زیر توجه کنید. فرض کنید کلاوز زیر به پرولوگ تعریف شده باشد.

vertical (seg(point(X,Y),point(X,Y1))) (1)

حال می توان از پرولوگ سوال زیر را پرسید:

?-vertical (seg (point (1, 1), point (1, 2))). (2)

Yes

پرولوگ به این سوال جواب 'yes' خواهد داد. این جواب با استفاده ازتطابق ما بین سوال پرسیده شده در (2)، با واقعیت بیان شده در(1)، بدست می آید. این تطابق بدین صورت حاصل میشود که متغیر X مقدار 1 و Y مقدار 1 و Y1 مقدار 2 میگیرد.

مثالهایی از تطابق مابین دو عبارت دلخواه در جدول زیر دیده می شوند.

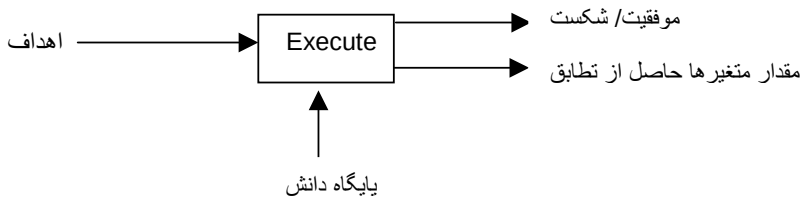
عبارت اول	عبارت دوم	مقدار دهی متغیرها
parent(tom, pat)	parent(X,Y)	X=tom, Y=pat
parent(X,pat)	parent(jim,Y)	X=jim, Y=pat
parent(tom,pat)	parent(tom,pat)	تطابق دارند
parent(X,jim)	parent(Y,Y)	X=Y=jim
parent(X,jim)	parent(tom,pat)	تطابق ندارند

1-6 معنای توصیفی و رویه ای برنامه های پرولوگ

برنامه های پرولوگ می توانند به دو راه فهمیده شوند.به صورت توصیفی و رویه ای. برای مثال عبارت P: - Q, R را در نظر می گیریم. در روش توصیفی میگوئیم P درست است اگر Q و R درست باشند.درروش رویه ای میگوئیم برای حل مسئله P، ابتدا زیر مسئله Q و سپس زیر مسئله R حل می شود. مفهوم رویهای تعیین می کند که پرولوگ

[19] Matching

چگونه به سوالات جواب می دهد. در حقیقت پرولوگ برنامه ورودی را که شامل واقعیت ها و قواعد است و *پایگاه دانش* [20] نیز نامیده می شود، دریافت میکند و به کمک آن به سوالات پرسیده شده که اهداف نامیده میشوند پاسخ می دهد.



در شکل بالا، ورودی یک پایگاه دانش و تعدادی هدف است و خروجی یک نشانگر موفقیت/عدم موفقیت و معرفی مقدار متغیرها است (برای موفقیت "yes" و عدم موفقیت "no" چاپ می شود). مقدار متغیرها فقط در حالت موفقیت آمیز تولید می شود.

قبلا گفتیم که می توان از راههای مختلف به یک هدف رسید. برای بیان این راهها میتوان از علامت **;** استفاده نمود. مثلاً عبارت P: - Q; R. بیان میکند که: "P درست است اگر Q **یا** R درست باشد". این عبارت می تواند به صورت زیر نیز بیان شود:

```
P :- Q.
P :- R.
```

خطر حلقه نا محدود

عبارت P :- P. از لحاظ توصیفی درست است ولی از نظر رویه ای کاملاً بی استفاده است. اگر برای جواب سوال ?- P از قاعده بالا استفاده شود، هدف P با هدف مشابه (یعنی P) جایگزین میشود. به همین صورت در این حالت پرولوگ وارد یک حلقه نا مشخص می شود. فلذا باید دقت شود که از نوشتن قواعدی که باعث ایجاد حلقه نا محدود می شوند پرهیز گردد. بعنوان مثال اگر رابطه predecessor را که قبلا تعریف کرده بودیم، به صورت زیر بنویسیم، در یک حلقه نامحدود خواهیم افتاد.

```
predecessor(X, Z) :- predecessor(Y, Z), parent (X, Y).
predecessor(X, Z) :- parent (X, Z).
```

7 لیستها، عملگرها و محاسبات

یک لیست دنباله ای از تعدادی آیتم است [ann, tennis, tom , skiing]. یک لیست می تواند ترکیبی از دو چیز در نظر گرفته می شود: عنصراول لیست که *سر* [21] لیست نامیده می شود و بخش باقیمانده لیست که *دنباله* [22] نامیده می شود. برای مثال در لیست بالا ann سر لیست و [tenis, tom, skiing] دنباله لیست است. سر یک لیست هر شیء میتواند باشد و دنباله باید یک لیست باشد. در حالت کلی یک لیست می تواند به صورت [H| T] نوشته شود که H همان سر و T همان دنباله است. مثلاً از پرولوگ سوال زیر را می پرسیم:

[20] Knowledge base
[21] Head
[22] Tail

?- [H| T] = [a, b, c].

H = a, T=[b,c] جواب پرولوگ

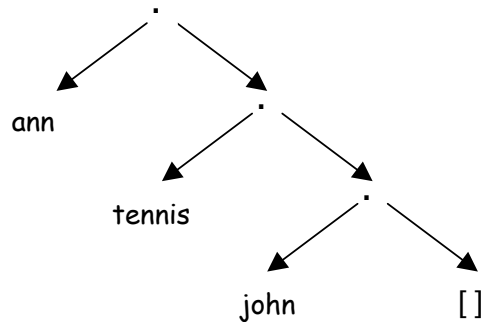
برای نمایش درختی یک لیست میتوان از علامت '.' نیز استفاده کرد:

[ann, tennis, john] = . (ann, [tennis, john])

= . (ann, . (tennis, [john]))

= . (ann, . (tennis, . (john, [])))

لیست بالا به صورت درختی زیر می تواند نشان داده شود:



معمولاً از نماد گذاری اول (با استفاده از نوارعمودی |) برای نمایش لیست ها استفاده می شود:

[a,b,c] = [a | [b,c]]= [a, b | [c]] = [a, b, c | []]

برخی عملیات روی لسیتها

از لیست ها می توان برای نمایش مجموعه ها استفاده کرد. اگر چه یک تفاوت وجود دارد: ترتیب عناصر در یک مجموعه مهم نیست در حالی که ترتیب آیتمها در یک لیست مهم است ، همچنین در یک لیست یک شی می تواند تکرار شود.

1- **عمل عضویت:**

X عضوی از لیست L است اگر:

الف- X سر L باشد

ب- یا X عضوی از دنبالهٔ L باشد.

member(X,L) : عضویت X در لیست L

member(X, [X|Tail]). شرط الف

member(X, [Head|Tail]) :- member(x,Tail) شرط ب

2- **عمل الحاق:**

الحاق دو لیست L1 و L2 لیست L3 است که بصورت زیر بدست می آید:

الف- الحاق لیست خالی با هر لیست L، برابرلیست L است

ب- الحاق لیست [X|T] با لیست L2 برابر با لیستی است که سر آن X و دنباله آن L3 است که L3 خود برابر الحاق T با L2 است.

conc (L1, L2, L3)

```
conc ([], L, L).
conc ([X|T], L2, [X|L3]) :- conc (T, L2, L3).
```

بعنوان نمونه داریم: conc ([a, b], [c, d], Y) که جواب پرولوگ برابر با Y=[a,b,c,d] است.

3- اضافه کردن یک آیتم

این روال به صورت یک واقعیت نوشته می شود: add(X, L, [X |L]).

4- حذف یک آیتم

اگر فرض کنیم با حذف X از لیست L، لیست L1 حاصل می شود و آن را به صورت del(X, L, L1) نشان دهیم، آنگاه L1 به صورت زیر بدست می آید:

الف- اگر X سرلیست باشد؛ آن را برداشته و دنباله را به عنوان جواب در نظر می گیریم.

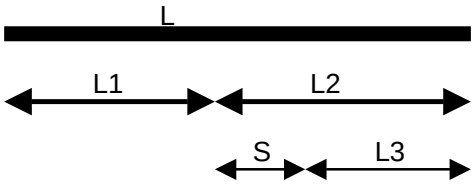
ب- اگر X در دنباله باشد، آنگاه آنرا از دنباله حذف می کنیم.

del (X, L, L1)

```
del (X, [X |Tail], Tail).
del (X, [Y |Tail], [Y|Tail1]) :- del (X, Tail, Tail1).
```

5- زیر لیست

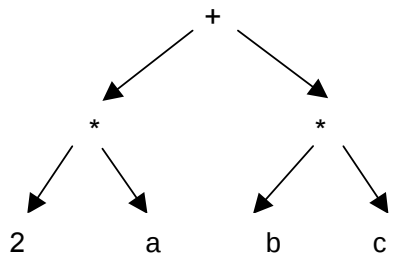
عمل زیر لیست را بدین صورت تعریف میکنیم که S زیر لیست L است اگر درون آن موجود باشد. مثلاً [a,c] زیر لیست [b,a,c,d,e] است ولی زیر لیست [a,b,c,d,e] نیست. یک راه برای تشخیص اینکه S زیر لیست L است این است که L از الحاق L1 و L2 درست شده باشد و L2 خود از الحاق S و لیستی به نام L3 تشکیل شده است. این حالت در شکل زیر به صورت شماتیک نشان داده شده است.




```
sublist (S, L) :- conc (L1, L2, L), conc (S, L3, L2).
```

نمادگزاری ریاضی

عبارت ریاضی 2 * a + b * c در پرولوگ به صورت پیشوندی + (* (2, a), *(b, c)) ذخیره میشود که نمایش درختی آن به صورت زیر است:



به خاطر اینکه عبارت ریاضی را در پرولوگ به صورت اصلی آن بنویسیم، عملگرها ی میانوندی را بصورت X f Y و عملگرهای پیشوندی را بصورت f X و عملگرهای پسوندی را به صورت X f با توجه به نوع عملگر و تقدم نسبت به دیگران تعریف می کنیم. البته برخی از عملگرها در سیستم پرولوگ از قبل تعریف شده اند. دستور زیر برای تعریف عملگر has از نوع میانوندی و با تقدم 600 می باشد.

:-op (600, XfX, has).

به عنوان مثال دیگر عملگرهای V، &، ⇔ و ~ را در زیر تعریف میکنیم:

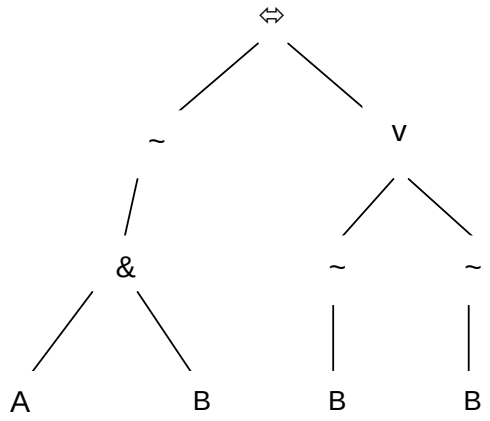
:- op(800, Xf, ⇔).

:- op(700, XfY, V).

:- op(600, XfY, &).

:- op(500, fY, ~).

اگرپس ازتعریف عملگر های بالا قضیه دمورگان را به صورت زیر بنویسیم: ~(A & B) ⇔ ~A v ~B ، پرولوگ آنرا به صورت درخت زیر تفسیر می کند.



حساب

برخی ازعملگرهای از پیش تعریف شده پرولوگ عبارتند از: جمع (+)، تفریق (-)، ضرب (*)، تقسیم (/)، توان (**)، تقسیم صحیح (//) و باقیمانده تقسیم صحیح (mod). برای انتساب[23] به جای مساوی ازis استفاده میشود. مثالهایی از عملیات ریاضی عبارتند از:

[23] Assignment

? – X is 5/2, Y is 5//2, Z is 5 mod 2

پاسخ پرولوگ Z = 1 , Y = 2 , X = 2.5

عملگرهای مقایسه ای عبارتند از:

- X > Y X بزرگتر از Y است
- X < Y X کوچکتر از Y است
- X>=Y X بزرگتر یا مساوی با Y است.
- X=<Y X کوچکتر یا مساوی با Y است.
- X =:= Y مقادیر X , Y برابرند.
- X =\= Y مقادیر X , Y برابر نیستند

مثالهای زیر عملگرهای ریاضی را بیشتر توضیح می دهند:

? – 1+2 =:= 2+1

پاسخ پرولوگ Yes

? -1+2 = 2+1

پاسخ پرولوگ No

? – 1+A = B+2

پاسخ پرولوگ B = 1 , A = 2

مثال زیر پیدا کردن بزرگترین مقسوم علیه مشترک را نشان میدهد.

اگر X > Y باشد دراین صورت بزرگترین مقسوم علیه مشترک آنها (D)برابر با بزرگترین مقسوم علیه مشترک X و X mod Y برابراست:

gcd(X, Y, D)

```
gcd(X, 0, X).
gcd(X, Y, D) :- X < Y, gcd(Y, X, D).
gcd(X, Y, D) : -  X >= Y  , X1 IS  X mod Y,  gcd (Y, X1 ,D).
```

مثال برای استفاده از ساختارها

هر پایگاه داده می تواند بطور طبیعی به صورت مجموعه ای از واقعیت ها در پرولوگ نشان داده شود. مثلاً یک پایگاه داده درباره خانواده می تواند ارائه شود بطوریکه هر خانواده با یک عبارت بیان می شود. هر خانواده دارای 3 جزء شوهر، زن و فرزند است. اطلاعات ساختاری خانواده به صورت زیر است و میتوان آن را در یک ساختار درختی نمایش داد:

```
family (person (ann, fox, date (7, may, 1969), works (bbc, 1520) ),
        person (ann, fox, date (9, may, 1961), unemployed),
        [person (pat, fox, date (5, may, 1983), unemployed),
         person (jim, fox, date (5, may, 1983), unemployed) ] ).
```

در این مثال اولین person در یک family، شوهر است. دومین person همسر و مابقی person ها درون لیست فرزندان هستند.

یک ویژگی خوب پرولوگ این است که می توانیم بدون تعیین تمام اجزاء اشیاء به آنها مراجعه کنیم. مثلاً برای پیدا کردن تمام زنان متأهل که حداقل سه فرزند دارند سوال به صورت زیر خواهد بود.

```
?-family (- , person (Name, Surname,-,-,-,), [-,-,-|-])
```

بعنوان مثالی دیگر یافتن نام پدر خانواده هایی که فامیل آنها sterling است و تولد آنها قبل از 1973 است:

```
?- family (person (Name, sterling, date(- , - , Year), -), -,-) ,
    Year < 1973.
```

انجام تجرد داده ها

تجرد داده ها را می توان به عنوان یک فرایند سازماندهی قطعات گوناگون اطلاعات درواحدهای طبیعی در نظر گرفت. بنابراین اطلاعات به شکلی ساخته می شوند که با معنی و مفهوم باشد. نکته این فرایند در اینجاست که برای امکان پذیر ساختن استفاده از اطلاعات بدون برنامه نویسی مجبوریم به جزئیات چگونگی نمایش واقعی اطلاعات توجه کنیم. مثالی از روش انجام این اصل نمایش خانواده به عنوان یک شی ساخت یافته در بالا دیده می شود. برای پیاده سازی تجرد داده ای می توان روابطی را تعریف کرد که به کمک آنها به اجزاء یک خانواده دسترسی داشت. در اینصورت میتوانیم بدون اینکه جزئیات شکل صفحه قبل را بدانیم , به اجزاء خاص خانواده دسترسی داشته باشیم. چنین روابطی را انتخاب کننده (Selector) مینامند. معمولاً چنین روابطی دو آرگومانی هستند که آرگومان اول شیئ است که حاوی جزء می باشد و آرگومان دوم خود جزء.

```
selector–relation (Object, Component–selector)
```

مثالهایی از انتخاب کننده ها برای ساختار خانوادگی عبارتند از:

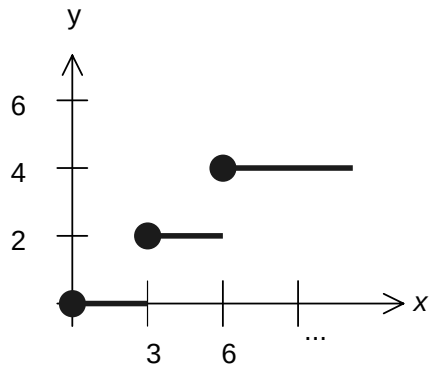
```
husband (family (Husband, - , -), Husband)
wife (family (- , wife, -), Wife).
children (family (- , - , Chil_list), Chil_list).
firstchild (Family, First): - children (Family, [First | -])
```

برای ایجاد و دستکاری این روش اطلاعات ، بایستی نام روابط انتخاب کننده را بدانیم و از آنها در بقیه برنامه استفاده کنیم. به طور ایده آل ، تمام جزئیات پیاده سازی چنین ساختاری باید برای کاربر نا مشهود باشد. مزیت دیگر استفاده از روابط انتخاب کننده , آسانتر شدن تغییر برنامه ها می باشد.

کنترل بازگشت به عقب

پرولوگ در صورت لزوم برای برآورده کردن هدفی، عمل بازگشت به عقب[24] را انجام می دهد. بازگشت به عقب ممکن است برخی مواقع باعث عدم کارایی در یک برنامه شود. بنابراین، گاهی اوقات می خواهیم بازگشت به عقب را کنترل کنیم و یا از آن جلوگیری کنیم. این کار با استفاده از امکان ***برش***[25] در پرولوگ قابل پیاده سازی است. به عنوان مثال فرض کنید که می خواهیم تابع ***f*** را که با نمودار زیر تعریف شده است در پرولوگ پیاده سازی کنیم.

[24] Backtracking
[25] Cut



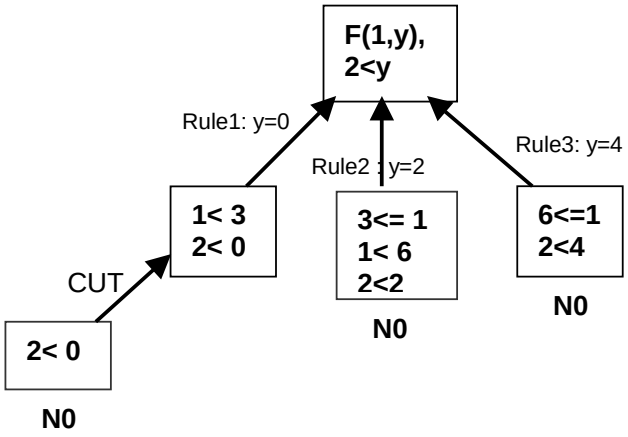
یک رابطه دودویی f (X , Y) را به صورت زیر تعریف می کنیم.

```
f(X, 0) :- X < 3.
f(X, 2) :- 3 =< X , X < 6.
f(X, 4) :- 6 =< X.
```

حال بررسی می کنیم که پس از مطرح شدن این سوال چه اتفاقی می افتد.

```
?-f (1, Y), 2 <Y.
```

هنگامیکه اولین هدف **f (1 , Y)** اجرا شود ، **Y** با صفر نمونه سازی می شود. بنابراین دومین هدف **2 <0** می شود که مردود است، بنابراین کل لیست هدف (یا همان سوال) انجام می شود. اما پیش از اینکه قبول کنیم که لیست هدف قابل ارضاء نیست، پرولوگ ردیابی دیگری را انجام می دهد.



عمل cut

این عملگر با علامت **!** نشان داده شده و به عنوان یک شبه هدف بین اهداف اضافه می شود. با توجه به اینکه Y مقدار0 را از قاعده اول می گیرد، و این به این سبب رخ می دهد که X کوچکتر از 3 است فلذا Y مقدار دیگری نمی تواند بگیرد و نیازی به بررسی مابقی قواعد نمی باشد. برای اینکه از بررسی قوانین دیگر جلوگیری کنیم برنامه را به صورت زیر بازنویسی می کنیم.

```
f(X, 0): - X < 3,!.
f(X, 2): - X <6,!.
f(X, 4).
```

حال اگر دوباره این سوال را بپرسیم ردیابی بعد از رسیدن به 0> 2 و no شدن جواب, شاخه های دیگر را بررسی نمی کند بنابراین کارایی برنامه بهبود می یابد. حال سوال **?- f (7, Y)** را می پرسیم باز هم برنامه شاخه ها را کنترل می کند ولی به **cut** نمی رسد. در اینجا برای حل این مشکل قانون سوم را که در واقع اگر 2 قانون اول برقرار نباشد برقرار است بدون شرط نوشته ایم، بنابراین جواب این سوال Y=4 خواهد بود. در شرط اول وقتی 3> X نباشد در شرط دوم دیگر نیاز به کنترل اینکه 3 =< X باشد نیست و در شرط سوم وقتی Xکمتر از 3 و بین 3 و 6 نباشد بنابراین حالت 3 است بنابراین نیاز به کنترل هیچ شرطی نیست.

مثال زیراز برش استفاده می کند و ماکزیمم دو عدد را محاسبه می کند. عدد بزرگ بین دو عدد X و Y را با استفاده از عبارتی به صورت max (X , Y , Max) نشان می دهیم.

```
Max(X, Y, X): -X > = Y.
Max (X, Y, Y): -X < Y.
```

با توجه به مطالب گفته شده در برش, معادله بالا به صورت زیرمیتواند نوشته شود.

```
Max (X, Y, Max):- X > = Y, ! , Max = X;
                  Max = Y.
```

بررسی مخالف بودن دو چیز

می خواهیم قاعده ای بنویسیم که متفاوت بودن X و Y (مساوی نبودن آنها) را نشان دهد. برای اینکار می توان از دستور fail استفاده کرد. این دستور نشان دهنده عدم موفقیت است و بیان می کند که درشاخهٔ فعلی جستجو نباید ادامه پیدا کند.

```
Different(X, Y) : - X = Y , ! , fail ;
                    true.
```

مثال) مریم تمام حیوانات را دوست دارد به جز مارها؟

```
likes (maryam , X): - snake (X) , ! , fail ;
                      animal (X)
```

علاوه بر این در پرولوگ یک گزاره با نام 'not' وجود دارد که می تواند در این مواقع استفاده شود، تا چنین حالتهایی راحت توصیف شود. در واقع not این را می گوید که اگر هدفی با عنوان goal پذیرفته شود در آن صورت not(goal) رد می شود، در غیر این صورت not(goal) پذیرفته می شود. بنابراین not به صورت زیر در پرولوگ تعریف خواهد شد.

```
Not (P): -P , ! , fail ;
         true.
```

not یک روال تعبیه شده در پرولوگ است که به عنوان یک عملگر پیشوندی تعریف شده است. در ضمن می توانیم هدف(not(goal را به صورت not goal بنویسیم. مثال بالا با استفاده از not به این صورت نوشته خواهد شد.

Likes (mary , X): - Animal (X) , not snake(X).

مزایای برش

1- با برش اغلب می توانیم کارایی برنامه را بهبود ببخشیم. این ایده دقیقاً به پرولوگ می گوید: سعی نکن راههای دیگر را امتحان کنی زیرا مقید به عدم پذیرش شده اند.

2- با استفاده از برش می توان قوانین انحصار متقابل را تعیین کرد، بنابراین می توانیم قوانینی به شکل زیر را بیان کنیم.

اگر شرط p درست باشد، آنگاه نتیجه Q درست است

در غیر این صورت نتیجه R درست است

اگر هیچ برش در برنامه نباشد می توانیم ترتیب اهداف و عبارات را تغییر دهیم این کار هیچ تاثیری بر مفهوم اعلانی نخواهد داشت. در حالی که در برنامه های همراه با برش, یک تغییر در ترتیب عبارات ممکن است بر مفهوم اعلانی تاثیر گذارد. مثال زیر را ببینید:

P: -a , ! , b.

P: -c

مفهوم اعلانی : P ⇔ (a&b) V (~a & c)

اگر عبارت جابه جا شود:

P: -c.

P: -a , ! , b.

مفهوم اعلانی : p⇔ (C V (a & b)

استدلال not فرض می کند که اطلاعات ما (یعنی پایگاه دانش) یک دنیای بسته را نشان می دهند. این بدین معنی است که هر چیزی که در پایگاه دانش وجود دارد، درست محسوب می شود و اگر چیزی در پایگاه دانش نباشد درست نیست و در نتیجه نفی[26] آن صحیح است.

مثال زیر را در نظر بگیرید:

```
good_standard (jeanluis).
expensive (jean Luis).
good_standard (francesco).
reasonable (Restaurant):- not expensive(Restaurant).
```

حال سؤال زیر را می پرسیم:

```
?- good_standard(X), reasonable(X).
   X=francesco                    جواب پرولوگ
```

حال سؤال فوق رابه صورت زیر می نویسیم، که از نظر توصیفی با سؤال بالا هم ارز است:

```
?-reasonable(X), good_standard(X)
   no                             جواب پرولوگ
```

در اینجا ابتدا باید یک X پیدا شود که reasonable است. این معادل زیر است:

[26] Not

Exists X : reasonable(X) ?

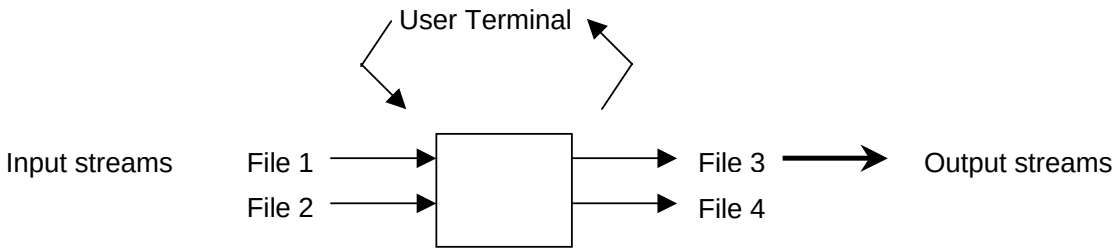
پرولوگ بهتر بود که این را بدین صورت تفسیر کند که آیا X ای وجود دارد که not expensive(X) صحیح باشد؟ که در این صورت می توانست به جواب برسد. در حالی که براي پاسخ به این سوال پرولوگ سوال زیر را بررسی می کند:

Not (Exists X: expensive (X)) ?

که این معادل است با اینکه " بأ زاء همهٔ X ها،"not expensive(X)" که جواب این سؤال برابر no است، زیرا وجود دارد رستورانی که گران است. و این یک مشکل برای استفاده از not در برنامه های پرولوگ می باشد. فلذا توصیه می شود که وقتی از not استفاده می شود بهتر است همهٔ متغیرها قبلاً مقدار گرفته باشند.

8 ورودی و خروجی

موقعیت کلی ارتباط با برنامه پرولوگ به صورت شکل زیر است



در هر زمانی در طی اجرای یک برنامه پرولوگ تنها دو فایل " فعال " هستند: یکی برای ورودی و دیگری برای خروجی. این دو فایل ورودی جاری و جریان خروجی جاری نامیده می شوند. در ابتدای اجرا، این جریان متناظر با ترمینال کاربر است.

جریان ورودی جاری می تواند به فایل دیگری **filename** با دستور**see(filename)** تغییر کند. چنین هدفی و عبارتی به عنوان یک تاثیر جانبی پذیرفته می شود. برای تغییر جریان خروجی از دستور **tell(filename)** استفاده می کنیم. مثلاً اگر فایلی با نام file1 داشته باشیم، آنگاه دستور **see(file1)** باعث می شود دستورات خواندن از ورودی بعدی از این فایل استفاده کنند. فایلهای ترتیبی به روشی همانند ترمینال رفتار می کنند. هر چند بیشتر پیاده سازیهای پرولوگ فایلهایی را به صورت دستیابی تصادفی نیز مدیریت می کنند، در حالت ترتیبی نوشتن در فایل به انتهای آن اضافه می کند و نمی توان در بخش های قبلی فایل بازنویسی کرد. مشاهدهٔ فایلها در پرولوگ دو روش عمده دارد یکی اینکه کاراکتر به کارکتر و دیگری به صورت بلوکی از کاراکتر ها و بلوک به بلوک.

پردازش فایلهایی از عبارات

گزاره تعبیه شده **read** برای خواندن عبارات از جریان ورودی جاری استفاده می کند. هدف **read(X)** باعث میشود عبارت بعدی خوانده شود و این عبارت با X تطبیق خواهد شد (در X قرار می گیرد). اگر ورودی وجود نداشته باشد هدف **read(x)** پذیرفته نمی شود.

گزاره **read** قطعی است، بنابراین در صورت عدم پذیرش، هیچ بازگشت به عقب برای ورود عبارت دیگر وجود ندارد. هرعبارت در فایل ورودی بایستی با یک توقف کامل (نقطه) و یک جای خالی و یا با کارکتر خط بعد (enter) خاتمه یابد. اگر **read(x)** در انتهای فایل ورودی جاری اجرا شود X با اتم **end-of-file** نمونه سازی می شود.

گزاره تعبیه شده **write** عبارتی را به خروجی می فرستد. بنابراین هدف **write(X)** عبارت X را به فایل خروجی جاری می فرستد. X به شکل ساختار دستوری استاندارد مشابهی که پرولوگ به طور طبیعی مقدار متغیر ها را نمایش می دهد ، در خروجی نوشته خواهد شد.

گزاره های دیگری نیز برای فضاهای خالی و خطوط جدید برای جریان خروجی وجود دارد. هدف**tab(N)** باعث می شود تا **N** فضای خالی د رخروجی قرار گیرد و گزاره بدون آرگومان **nl** باعث شروع خط جدید در خروجی می شود.

مثال: برنامه cube (N, C) :- C is N*N*N را به برنامه ای تبدیل کنید که از فایل بخواند.

```
cube: - read (X), process(X).
process (stop): - ! .
process(N): - C is N * N * N , write ( C ) , cube.
```

حال این برنامه از فایل تازمانیکه به stop برسد می خواند و در خروجی چاپ می کند. در عبارت process(N) حالت بازگشت به cube را داخل دستورات قرار داده ایم.

مثال: نوشتن اعضای لیست در خروجی در حالت دلخواه

```
write_list ([ ]).
write_list ([X | L]): - write (X), n1, write_list (L).
```

مثال: لیست اعداد صحیح را به صورت یک گراف نواری نشان دهد.

```
bars ([ ]).
bars ([N|L]) : - stars (N), nL, bars (L).
stars (N) : - N > 0, write (*), N1 is N – 1, stars(N1).
stars (N): - N =< 0.
```

ساخت و تجزیه اتمها

برنامه زیر رویه ای برای انتقال یک جمله به لیستی از اتمها است.

```
getsentence (Wordlist): -
    get0 (Char),
    getrest (Char, Wordlist).
getrest (46, []): -! .          % end of sentence: 46 = ASCII for '.'
```

```
getrest (32, Wordlist): - ! ,        % 32 = ASCII for black
    getsentence (Word List).            % skip the black
getrest (Letter, [Word |Word list]):-
    getletters (Letter, Letters, Nextchar),    % read letters of current word
    name (Word, Letters),
    getrest (Nextchar, Wordlist).
getletters (46, [ ], 46): -! .                % End of word: 46= full stop.
getletters (32, [ ], 32): -! .                % End of word: 32 = blank
getletters (Let, [Let | Letters], Nextchar): -
    get0 (Char),
    getletters (Char, Letters, Nextchar).
```

در واقع رویه ی بالا یک جمله راکه به نقطه ختم شده رابه عنوان ورودی برای لیستی از اتمها قبول می کند و هر حرف را یک اتم در نظر می گیرد و از روی کاراکتر فاصله عبور می کند تا به کلمه ی بعدی برسد. برای مثال عبارت getsentence(Wordlist) از روی ورودی Mary was pleased to see the robot fail خروجی زیر را می سازد: [“Mary”, was, pleased, to, see, the, robot, fail]

گزاره های تعبیه شده

برخی از گزاره های تعبیه شده عبارتند از:

number(X) : درست است اگر X یک عدد باشد.

var (X) پذیرفته است اگر x یک متغیر نمونه سازی نشده باشد.

novar (X) پذیرفته است اگر x یک متغیر نباشد ، یا متغیر نمونه سازی شده باشد.

atom (X) درست است اگر x هم اکنون برای یک اتم در نظر گرفته شده باشد.

integer(X) درست است اگر x هم اکنون برای یک عدد صحیح در نظر گرفته شده باشد.

float (X) درست است اگر x هم اکنون برای یک عدد اعشاری در نظر گرفته شده باشد.

number(X) درست است اگر x هم اکنون برای یک عدد در نظر گرفته شده باشد.

atomic(X) درست است اگر x هم اکنون برای یک عدد یا یک اتم در نظر گرفته شده باشد.

compound(X) درست است اگر x هم اکنون برای یک عبارت مرکب (یک ساختار) در نظر گرفته شده باشد.

سوالات زیر کاربرد این گزاره ها را برای پرولوگ تشریح می کنند.

?- var (Z) , Z = 2.	**?-Z = 2 , var (Z)**
Z = 2	**no**

? – integer (Z) , Z = 2
No
? – atom (3.14)
No
? – atom (= =>)
Yes
? – compound (2+X)
Yes

?-Z=2 , integer(Z),
Z = 2
?-atomic (3.14)
yes
? – atom (p(1)).
no

9 اعمال روی ساختمان داده ها

مرتب سازی لیستها

ابتدا رابطهٔ ترتیب gt(X,Y) را طوری تعریف می کنیم که X بزرگتر از Y است را تعریف می کند بنابراین

```
g t (X , Y): - X > Y.
```

حال برای مرتب کردن یک لیست با استفاده از رابطه gt زوج عنصر کنار هم را مقایسه کرده و جا به جا می کنیم تا لیستمان مرتب شود. به این روش مرتب سازی bubblesort یا مرتب سازی حبابی می گویند.

```
bubblesort (List, Sorted): -
    swap (List, List1), ! ,                    % A useful swap in List?
    bubble sort (List, Sorted).
bubblesort (Sorted, Sorted).                   % otherwise List is already sorted
swap( [ X, Y | Rost] , [ Y , X | Rest] ): -    % swap first two elements
    gt (X,Y).
swap ( [Z | Rest ] , [Z | Rest1] ): -                % swap elements in dial
    swap (Rest , Rest1).
```

الگوریتم سادهٔ دیگر برای مرتب سازی (insertion sort) است که بر اساس ایده زیر می باشد.

برای مرتب سازی یک لیست غیر تهی L = [X | T] :

1- دنبالهT از L را مرتب کنید.

2- سرX از L را در محلی که لیست نتیجه مرتب شده به دنباله اضافه کنید.

```
insertsort ([ ] , [ ] ).
insertsort ( [X | Tail] , Sorted ) : - insertsort (Tail , Sortedtail ) ,      % sort the Tail
    insert (X , Sortedtail, Sorted).                          % insert X at proper place
insert (X , [Y | Sorted ] , [Y | Sorted1] ) : -
    gt (X, Y), ! ,
    insert (X, Sorted, Sorted1).
insert (S, Sorted, [ S | Sorted ] ).
```

یکی دیگر از مرتب سازیها روش مرتب سازی Quick Sort است که به صورت زیر است:

1- عنصرX را از L حذف می کنیم و بقیه L ها را به دو لیست به نامهای Big و Small به این صورت تقسیم می کنیم که تمام عناصر L که بزرگتر از X هستند متعلق به Big و بقیه متعلق به Small میباشند.

2- Small را مرتب کن تا Sortedsmall بدست آید.

3- Big را مرتب کن تا Sortedbig بدست آید.

4- کل لیست مرتب شده الحاق Sortedsmall و [X | Sortedbig] است.

```
%quick sort (list, sortedlist): sort list by the quick sort algorithm
    quicksort ([ ], [ ]),
    quicksort([X|Tail], Sorted):-
        split(X, Tail, Small, Big),
        quicksort (Small, Sortedsmall),
        quicksort(Big, Sortedbig),
        conc (Sortedsmall, [X|Sortedbig], Sorted).
    split(X, [ ], [ ], [ ]).
    split(X, [Y|Tail], [Y|Small], Big):-
        gt(X, Y), ! ,
        split(X, Tail, Small, Big).
    split(X, [ Y| Tail ], Small, [Y|Big] ) :-
        split(X, Tail, Small, Big),
```