زبان برنامه نویسی Dyna

استاد: دکتر فرشی

تهیه کننده: محمد رضا بشردوست

آذر 1390

معرفی:

Dyna یک زبان برنامه نویسی کوچک و سطح بالا است که برای ساده کردن پیاده سازی و پیمایش وزن برنامه های پویا طراحی شده است.

Dyna جزو زبان های اعلانی (declarative) و خالص (pure) و منطقی وزن-دار(weighted logic) است، بطوری که یک برنامه ی آن معمولا از مجموعه ای از روابط برابری (equation) تشکیل شده است که الگوریتم اصلی حل مساله را مشخص می کنند. در واقع این روابط قواعد استنتاج (inference rule) هستند که طبق آن ها نتایج (consequent) از مقدمات (antecedent) استنتاج می شوند (infer). البته عبارت های دیگری نیز برای معرفی و مشخص کردن نوع اشیا، یا عبارت هایی برای مقداردهی و موارد دیگری نیز ممکن است در برنامه وجود داشته باشند.

کامپایلر Dyna (dynac) بنا به برنامه ی نوشته شده، کلاس هایی به زبان C++ تولید می کند که این کلاس ها هسته ی برنامه ی نهایی را تشکیل می دهند و توسط آن ها به اضافه ی دستورات دیگری بنا به نیاز، برنامه ی کاملی ساخته می شود و سپس این برنامه توسط کامپایلر C++ (معمولا gcc) به کد اجرایی تبدیل می شود.

از آنجا که Dyna، turing-complete است هر الگوریتمی را می توان در آن پیاده سازی کرد، ولی این زبان برای مواردی طراحی شده است که راه حل بهینه ی آن ها از طریق برنامه ریزی پویا بدست می آید. مانند اثبات قضایا در دستگاه های اصول

موضوعی (axiomatic system) و مسایلی در حوزه ی پردازش زبان های طبیعی (NLP) و هوش مصنوعی (AI).

این زبان هنوز به اندازه ی کافی برای استفاده ی گسترده توسعه نیافته و بیشتر در دانشگاه ها استفاده ی تحقیقاتی دارد. ویرایش جدیدی از آن از پایه در حال توسعه است که هنوز به بهره برداری نرسیده است. (برای اطلاعات بیشتر به سایت www.dyna.org یا www.jhu.edu مراجعه کنید.)

نوشتن برنامه با Dyna:

به طور سنتی اولین برنامه ای که هر کس در اولین برخورد با یک زبان جدید می نویسد، برنامه ی Hello world است، که معمولا برنامه ی ساده ایست که ورودی نمی گیرد و رشته ی "Hello world" را در خروجی چاپ می کند. ولی در مورد Dyna قضیه کمی متفاوت است و از آنجا که Dyna یک زبان خالص است و ارتباطی با سخت افزار و ورودی/خروجی ندارد و فقط محاسبات را انجام می دهد، این برنامه را باید به شکل دیگری با حاصل عددی پیاده سازی کنیم.

مثال 1:

```
% My first Dyna program (helloworld.dyna)
:- item(item, double, 0).   % values are double-precision real numbers
goal += hello*world.        % an inference rule for deriving values
hello := 6.                 % some initial values
world := 7.
```

برنامه ی بالا را می توانید در یک ویرایشگر متن نوشته و در فایلی متنی مثلا به نام `helloworld.dyna` ذخیره کنید سپس با اجرای فرمان

```
dynac helloworld.dyna --driver=goal
```

آن را کامپایل کنید (کامپایلر قبلا باید بر روی سیستم شما موجود باشد.). فرمان بالا علاوه بر کامپایل برنامه ی نوشته شده و تولید کلاس های C++ برنامه ی راه انداز به زبان C++ را هم تولید کرده و آن را با کامپایلر مربوط کامپایل می کند و در نتیجه کد اجرایی هم تولید می شود که در اینجا به نام `helloworld` است،

و با اجرای فرمان

```
./helloworld
```

در کنسول (در سیستم های Unix و Unix-like) خروجی برنامه که "42" است بر روی خروجی استاندارد چاپ می شود.

حال به جزییات این برنامه و فرمان های بالا می پردازیم تا معنی هر خط از برنامه و پارامتر استفاده شده در فراخوانی کامپایلر روشن شود.

```
% My first Dyna program (helloworld.dyna)
```

خط بالا فقط توضیح (comment) است و در عملکرد برنامه تاثیری ندارد. به طور کلی عملکرد علامت % در Dyna مانند علامت // در C++ است و هر چیزی که بعد از آن در همان خط بیاید، به عنوان توضیح در نظر گرفته شده و کامپایل نمی شود.

```
:- item(item, double, 0).   % values are double-precision real numbers
```

این خط مشخص می کند که تمام آیتم های (item) این برنامه از نوع عدد حقیقی با دقت مضاعف (double) با مقدار اولیه 0 هستند. آیتم ها در Dyna همانند متغیرها (variable) در C++ هستند و در واقع ساده ترین عبارت (expression) هستند که مقدار مشخصی دارند. اعلان ها با :- شروع می شوند و در انتهای همه ی قواعد (rule) و اعلان ها (declaration) باید . (نقطه) گذاشته شود. (double، int و string از انواع اولیه (primitive) هستند.)

```
goal += hello*world.        % an inference rule for deriving values
```

این یک قاعده ی استنتاج است و مشخص می کند که چگونه متغیر سمت چپ += از مقادیر سمت راست آن محاسبه می شود. += به معنای جمع مقادیر است (مانند سیگما در حساب) ولی در اینجا طرف راست فقط یک مقدار دارد. نکته اینکه goal همواره وابسته به مقادیر سمت راست است و اگر در طول برنامه مقدار hello یا world تغییر کند، مقدار goal هم تغییر می کند. ترتیب قاعده های استنتاج و اعلان ها مهم نیست. فاصله های خالی (whitespace) هم تا جایی که تاثیری بر تشخیص علامت ها (tokenization) نداشته باشد، مهم نیستند.

```
hello := 6.                 % some initial values
world := 7.
```

دو خط بالا به آیتم های hello و world مقدار اولیه می دهد. این مقادیر در برنامه ی درایور هم می توانند داده شوند. به این آیتم ها به صورت منطقی اصل موضوع (axiom) گفته می شود و قضیه ی (theorem) goal از آن ها استنتاج می شود. := عملگر انتصاب است.

-پایان متن برنامه-

```
dynac helloworld.dyna --driver=goal
```

حال با اجرای دستور بالا در خط فرمان کامپایلر (dynac) علاوه بر کامپایل کردن برنامه ی helloworld.dyna ، برنامه ی درایور (که به زبان C++ است) را هم تولید و به گونه ای تنظیم می کند که در زمان اجرا مقدار goal را به در خروجی چاپ کند و برنامه ی C++ را هم توسط کامپایلر مربوطه کامپایل می کند و فایل اجرایی تولید می

کند. اگر `--driver=goal` را حذف کنیم، برنامه ی درایور تولید و کامپایل نمی شود و این در صورتی است که برنامه نویس خود مایل به نوشتن برنامه باشد. یک راه دیگر هم ویرایش درایور تولید شده است (این راه ساده تر و مطمئن تر است).

یک مثال پیچیده تر برای آشنایی بیشتر با Dyna، یافتن کوتاهترین مسیر در گراف است که به صورت زیر است:

مثال 2:

```
:- item(item, double, inf).
pathto(V) min= edge("Start", V).
pathto(V) min= pathto(U) + edge(U,V).
goal      min= pathto("End").
```

در برنامه ی بالا راس ها و یال ها و مقدار یال ها داده نشده است و فقط روش بدست آوردن کوتاه ترین مسیر مشخص شده است، در نتیجه این داده ها باید به برنامه داده شوند. برای این کار می توان داده ها را در برنامه درایور مشخص کرد (در داخل کد یا از طریق گرفتن داده ها از ورودی). البته به طور پیش فرض کامپایلر برنامه را به گونه ای تنظیم می کند که کد اجرایی، این داده ها را از ورودی (یک پارامتر که فایلی حاوی داده ها است) دریافت کند. توجه کنید که تعداد یال ها محدودیتی ندارند و اسم یال ها هم به صورت رشته (string) تعریف شده است. `Start` و `End` کلمات کلیدی Dyna نیستند و فقط اسم دو راس هستند که باید در ورودی وجود داشته باشند تا برنامه بتواند کوتاهترین مسیر از راسی به نام `Start` به راسی به نام `End` را (در صورت وجود) پیدا کند.

فایل ورودی را می توان به صورت زیر تهیه کرد و در موقع اجرا به عنوان پارامتر به برنامه داد.

```
% driving to and from the airport
edge("Start","BOS") := 10.
edge("ORD","End") := 20.

% available flights
edge("BOS","JFK") := 187.
edge("BOS","MIA") := 1258.
edge("JFK","DFW") := 1391.
edge("JFK","SFO") := 2582.
edge("JFK","MIA") := 1090.
edge("MIA","DFW") := 1121.
edge("MIA","LAX") := 2342.
edge("DFW","ORD") := 802.
edge("DFW","LAX") := 1235.
edge("ORD","DFW") := 802.
edge("LAX","ORD") := 1749.
```

این فایل را مثلا با نام `flights.par` ذخیره کرده و دو فرمان زیر را در کنسول اجرا می کنیم.

```
dynac path.dyna --driver=goal
./path flights.par
```

و در نتیجه خروجی برنامه که "2410" است، در خروجی استاندارد چاپ می شود.

اکنون به جزییات برنامه می پردازیم تا معنی و عملکرد فرمان ها مشخص شود:

```
:- item(item, double, inf).
```

خط بالا مشخص می کند که همه ی مقادیر، اعداد حقیقی با دقت مضاعف با مقدار اولیه بی نهایت هستند.

```
pathto(V) min= edge("Start", V).
pathto(V) min= pathto(U) + edge(U,V).
```

دو خط بالا مشخص می کند که کوتاه ترین مسیر از راس `Start` به هر راس `V` چگونه محاسبه می شود. `V` و `U` متغیر هستند و بطور کلی متغیرها با حروف بزرگ مشخص می شوند. عملگر `min=` از بین تمام مقدار های موجود کمترین مقدار را به `pathto(V)` اختصاص می دهد. دو عبارت بالا با هم هم ارز هستند، به این صورت که کوتاه ترین مسیر از `start` به هر راس `V` برابر است با مینیمم "کوتاه ترین یال از `Start` به `V`" و "مینیمم کوتاه ترین مسیر تا `U` به اضافه ی یالی از `U` به `V`".

```
goal      min= pathto("End").
```

این فرمان مقدار کوتاه ترین مسیر از `Start` به `End` به `goal` اختصاص می دهد. از آن جا که `pathto("End")` کوتاه ترین مسیر است، استفاده از `min=` بی معنی به نظر می رسد، ولی با توجه به محدودیت های ویرایش فعلی Dyna که باید از عملگرهای یکسان در یک برنامه استفاده شود، مجبوریم از `min=` استفاده کنیم، که در اینجا همان کاری را انجام می دهد که برای ما مناسب است.

حال که تا اندازه ای با زبانِ Dyna و فرم کلی برنامه های آن آشنا شدیم به تعریف بعضی از اجزای آن می پردازیم.

انواع اولیه (primitive type):

عبارتند از عدد صحیح (int)، عدد حقیقی با دقت مضاعف (double)، رشته (string) و بولی (bool). شما همچنین می توانید کلاس های C++ را به عنوان انواع اولیه وارد کنید (هنوز کاملا پیاده سازی نشده و باید ملاحضاتی را در نظر گرفت.).

اتم (atom):

نام ها و همچنین هر ترتیب از کاراکترها در بین کوتیشن (علامت نقل قول تکی که به شکل ' است)، اتم هستند. اتم اگر فقط شامل حروف الفبا و عدد و زیرخط (_) باشد، و با حرف کوچک شروع شود، احتیاج به کوتیشن ندارد. در غیر این صورت باید در بین دو کوتیشن تکی قرار گیرد.

ترم (term):

به هر شی داده ای ترم گفته می شود. هر چیز در Dyna یک ترم است تقریبا به همان طریق که هر چیز در C یک شی است. هر ترم داری نوع است.

عبارت (expression):

به ترم هایی که مقدار اختصاص داده شده است، عبارت (expression) گفته می شود.

آیتم (item):

به ساده ترین نوع عبارت آیتم می گویند. بنابر این، آیتم ها در Dyna شبیه متغیرها در زبان های رویه ای مانند C++ هستند.

متغیر (variable):

در قواعد استنتاج، یک متغیر، مشخص کننده ی مکانی است که همه ترم ها می توانند به جای آن بنشینند.

قاعده ی استنتاج (inference rule):

یک قاعده ی استنتاج مشخص می کند که چگونه مقدار یک آیتم از روی مقدار آیتم های دیگر محاسبه می شود.

الگو (pattern):

به ترم هایی که دارای متغیر هستند، الگو گفته می شود.

مثال 3:

```
:- structure(isUpperCase(string word)).
:- item(isUpperCase, bool, false).
:- structure(word(string w, int from, int to)).
:- item(word, double, 0).
```

در سطر اول یک ساختار (structure) به نام `isUpperCase` اعلان (declare) شده است. که آن شامل یک رشته به نام `word` است. و همچنین طبق اعلان خط سوم `word` نیز یک ساختار است.

در سطر دوم نوع آیتم `isUpperCase` و مقدار پیش فرضش اعلان شده است.

سطر سوم و چهارم هم به همین ترتیب بالا.

طبق تعریف های بالا:

تمامی اسامی مانند `word` اتم هستند.

تمامی اسامی تعریف شده، آیتم هستند، چون دارای مقدار هستند. پس به همین ترتیب همه ی آن ها یک عبارت ساده هستند.

همچنین شی های اعلان شده، ترم هستند که در این مثال تمام آن ها مقدار و نوع دارند، پس همانطور که گفته شده عبارت هستند (این عبارت های ساده آیتم نامیده شدند.).

در مثال بالا قاعده ی استنتاج، الگو و متغیر وجود نداشت.

مثال 4:

```
maxout(U) max= edge_weight(U,V).
maxin(V) max= edge_weight(U,V).
```

در این مثال همه ی عبارت های ظاهر شده مانند `edge_weight(U,V)` یک الگو هستند، چون در آن ها متغیر (مانند `V`)به کار رفته است، و این متغیرها بر روی تمام ترم ها تغییر می کنند.

هر خط یک قاعده ی استنتاج را بیان می کند و مشخص می کند که چگونه مقدار عبارت سمت چپ از مقدار عبارت سمت راست محاسبه می شود. فرم کلی این قواعد به شکل زیر است:

```
consequent accumulation_operator antecedent
```

مثلا در خط اول مقدار `maxout(this_vertix)` (برآیند: consequent) برابر با بزرگترین مقدار `edge_weight(this_vertix,U)` (مقدم: antecedent) است، که `U` به ازای تمام راس های موجود در برنامه تغییر می کند. `max=` نیز یک عملگر جمع آوری (`accumulation_operator`) است.

محدودیت ها:

در حال حاضر برنامه نویسی با Dyna دارای محدودیت های زیادی است که در ویرایش آینده کمتر خواهد شد. بعضر از این محدودیت ها از قرار زیر است:

- اعلان آیتم ها استفاده ی آن ها در برنامه فقط به صورت های زیر امکان پذیر است:

```
:- item(item,double,0) with += and *
:- item(item,bool,false) with |= and &
:- item(item,double,-inf) with log+= and +
:- item(item,double,-inf) with max= and +
:- item(item,double,inf) with min= and +
```

- انواع ابتدایی را نمی توان در سمت راست قواعد استفاده کرد.
- همه ی آیتم ها در یک برنامه باید نوع یکسان و مقدار اولیه یکسان داشته باشند.
- همه ی عملگرهای جمع آوری (مانند min= یا +=) که در برنامه استفاده شده اند باید یکسان باشند.
- همه ی عملگرهای محاسباتی که در عبارت ها استفاده شده اند، باید یکسان باشند.
- در یک قاعده ی استنتاج، همه ی متغیرهای ظاهر شده در برآیند (سمت چپ رابطه)، باید در عبارت های مقدم (سمت راست رابطه) نیز وجود داشته باشند.

- و محدودیت های دیگری که ممکن است در موقع کامپایل کردن به آن بر بخورید.

پایان