mud lecture

[Rock Lecture] 12. Language Concept

why do you ask
2014. 2. 20. 12:34

I'm not going to post this lecture on edac
This article is written in order to leave a record of making tools.

First of all, as you saw until the 11th
Trigger programming in Python or anything else is a bit mental.
So, go out for a bit.

I could make a library that allows trigger programming in Python.
The concept of trigger programming itself can be applied in so many ways.
I think it would be okay to make a tool dedicated to trigger programming.

The trigger programming syntax I will create is as follows.

```
 Trigger:
Conditions:
Deaths("Player 1", AtLeast, 1, "Terran Marine") @cond1
Actions:
SetDeaths("Player 1", Subtract, 1, "Terran Marine")
SetDeaths(&cond1, Add, 2, 0)
PreserveTrigger()
```

It is a combination of Starforge and SCMDraft2, and the following expansions are made.
 1. @cond1 is called Label. The address of the expression on the same line as @cond1 is assi gned to cond1.

The corresponding EBNF rules for this format are:

```
< identifier> ::= string | '&'? (number | name ('.' name)*)
<term> ::= <vchunk> (['*' '/'] <vchunk>)*
<poly> ::= ('+' | '-' | '') <term> (['+' '-'] <term>)*
<vchunk> ::= '(' <poly> ')' | <identifier>
```

```
<addresser> ::= '@' name

<trgheader> ::= 'trigger' ('preserved')? ':'
<condheader> ::= 'condition' ':'
<actheader> ::= 'action' ':'
<prev> ::= 'prev' <expr>
<next> ::= 'next' <expr>
<normalline> ::= name '\(' (<expr> (',' <expr>)*)? '\)'

<spacer> ::= 'space' number
<filestring> ::= 'file' path

<line> ::= (<trgheader> | <condheader> | <actheader> | <prev> | <next> | <normalline> | <spacer> | <filestrin
g> | '') <addresser>? <EOF>
```

It's a simple format.

This is my first time making a proper script parser. So, first, the parser was simply written as a recursive des cent parser, and to simplify error handling, the program was parsed up to the <line> unit. (Parse line by line.)

If you want to parse a program in <program> units, it will be a bit cumbersome when an error message is dis played. There seems to be something called a predictive parser, but I don't know the details, I need to learn more. First, I made a parser by hand for practice.
The script I made will probably only print which lines are correct and which are incorrect, and will not provide any further information.
Find out where I went wrong.

Interpreting the above source according to this format is as follows. (I printed it with the pprint m odule)

```
[ None,
  [['triggerheader', None], None],
  [['condheader', None], None],
  [['normalline',
       'deaths',
       [['expr', 1, ['vchunk', 1, ['identifier', 0, '"player 1"']]],
          ['expr',
             One,
             ['vchunk', 1, ['identifier', 2, None, ['atleast']]]],
          ['expr', 1, ['vchunk', 1, ['identifier', 1, None, 1]]],
          ['expr',
             One,
             ['vchunk', 1, ['identifier', 0, '"terran marine"']]]]],
    ['addresser', None]],
  [['actheader', None], None],
  [['normalline',
       'setdeaths',
       [['expr', 1, ['vchunk', 1, ['identifier', 0, '"player 1"']]],
```

whyask37's blog

```
            ['expr', 1, ['vchunk', 1, ['identifier', 1, None, 1]]],
            ['expr',
               One,
               ['vchunk', 1, ['identifier', 0, '"terran marine"']]]]],
      None],
   [['normalline',
          'setdeaths',
          [['expr',
               One,
               ['vchunk', 1, ['identifier', 2, '&', ['cond1']]]],
            ['expr',
               One,
               ['vchunk', 1, ['identifier', 2, None, ['add']]]],
            ['expr', 1, ['vchunk', 1, ['identifier', 1, None, 2]]],
            ['expr', 1, ['vchunk', 1, ['identifier', 1, None, 0]]]]],
      None],
   [['normalline', 'preservetrigger', None], None],
   None]
```

The source code itself is a common recursive descent parser, so you can analyze it with a little search.
Here, rather than analyzing the source code itself, let's talk a little more about why the parser ru les are set like this.

---

1. SCMDraft2 Rules

The rules of SCMDraft2 are like C. An example trigger is:

```
Trigger("Avoider"){
Conditions:
Elapsed Time(At least, 8);

Actions:
Set Countdown Timer(Set To, 36000);
Set Score("Current Player", Set To, 30000, Custom);
Leader Board Points("\x004 Survival Points", Custom);
Leaderboard Computer Players (disabled);
Set Switch("GameStart", set);
Set Switch("StageStart", set);
Set Resources("Current Player", Set To, 500, ore);
Set Deaths("Player 8", "Terran Physics Lab", Set To, 1);
Set Deaths("Player 1", "Terran Physics Lab", Set To, 1);
Comment("초기화 트리거");
}
```

그냥 자작맵에서 하나 따왔습니다. 코드를 구분하자면 다음과 같이 6가지로 나눌 수 있겠죠.

1. 트리거 헤더 : Trigger("Avoider")
2. 트리거 시작 : {
3. 조건부 시작 : Conditions:
4. 액션부 시작 : Actions:
5. 트리거 끝 : }
6. 조건과 액션 : Elapsed time(At least, 8)

주석은 무시하도록 하겠습니다.

모든 트리거는 다음과 같이 쓸 수 있을겁니다.
- 트리거 헤더가 하나 있고
- 트리거 시작하는 괄호가 하나 있고
- 조건부 시작이 있고
- 조건이 여러개 있고
- 액션부 시작이 있고
- 액션이 여러개 있고
- 트리거를 끝내는 괄호가 있고.

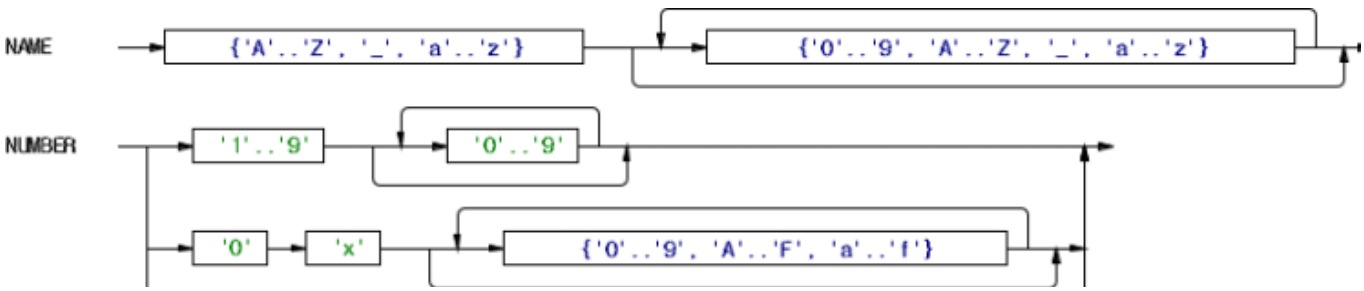
위의 6가지를 각각 trigger_header, trigger_start, condition_start, action_start, trigger_end, statement 라고 부르도록 합시다.
그러면 다음과 같이 trigger를 쓸 수 있습니다.

trigger ::= trigger_header trigger_start condition_start statement* action_start statement* trigger_end

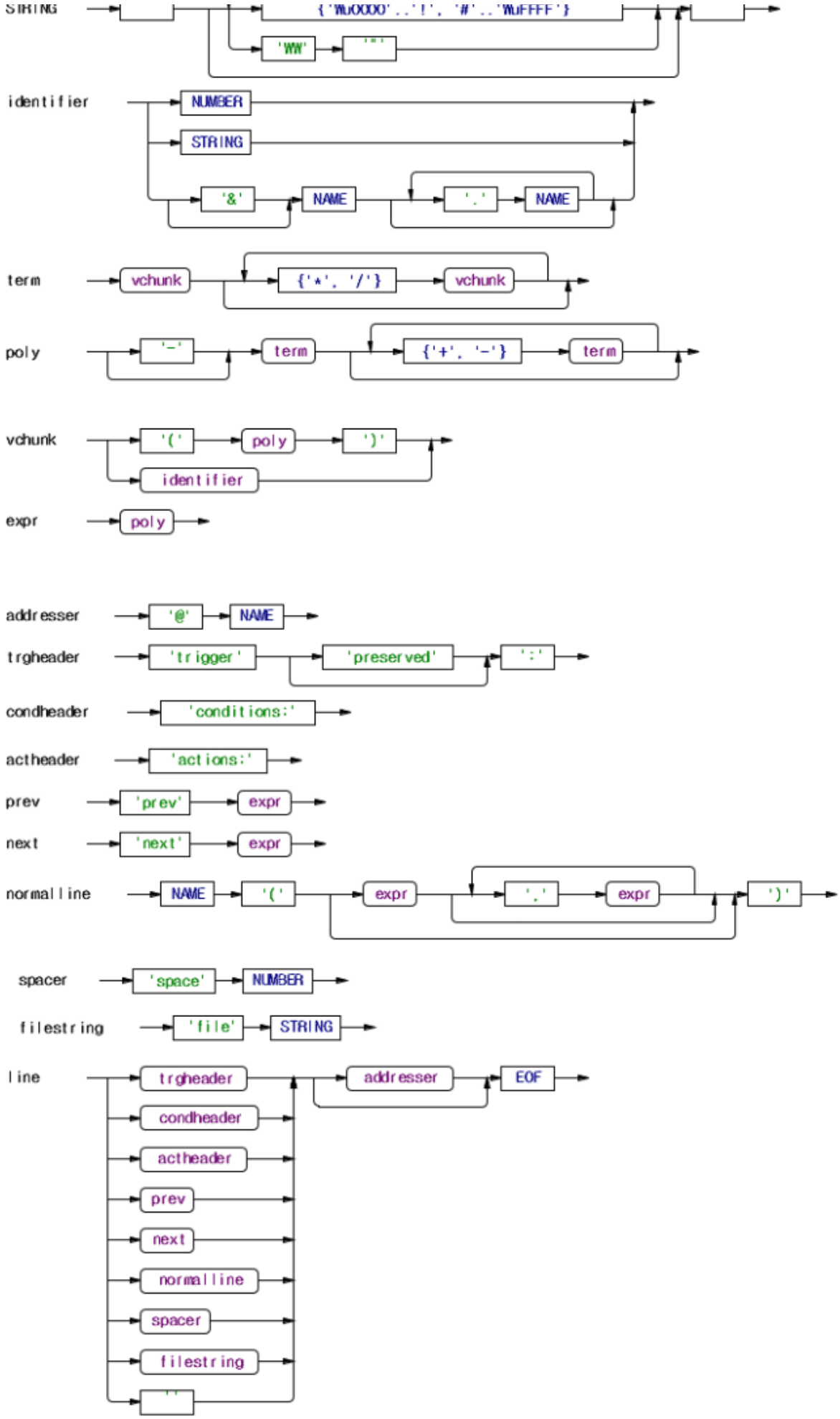
여기에서
- statement*에서 *는 statement를 여러번 (0번 포함) 반복하라는 의미입니다.

위에서 봤던 EBNF랑 비슷한 형태죠. 대충 이런 형식으로 EBNF를 쓰면 됩니다.
더 하기 귀찮아서 생략.

위에 있던 EBNF 포맷을 Syntax diagram으로 표현하면 다음과 같습니다.



whyask37's blog



whyask37's blog

syntax diagram은 antlr3works의 내장 프로그램으로 만들었습니다.
원래 antlr3으로 파서까지 만드려 했는데 사용법이 ㅈ같아서 연습도 할 겸 파서를 자작으로 했는데
이거 사람이 할만한 일이 아니에요.

그렇게 EBNF로 줄 단위 파싱을 했으면
이제 프로그램 단위로 다시 파싱을 해야죠.
프로그램은 무려 LL-0이라서 굉장히 파싱이 쉽습니다.
걍 유한상태기계 하나 가지고 잘 파싱하면 되요.

파서는 2-pass로 움직이는데
일단 레이블들의 주소를 계산하고
그 다음에 실제 바이트코드를 만듭니다.

레이블 주소에서는 각 줄들이 어떤 의미를 가지고 있는지 대략 파악만 하고, (실제 파싱은 안함)
바이트코드 만들 때에는 각 줄들을 정확하게 파싱합니다.

쓰기귀찮아...

#IT·컴퓨터

[Lesson Lecture] Extra: Trigger Programming - Practice

2014. 2. 22.

2

[Rock Lecture] 12. Language Concept

2014. 2. 20.

0

[Rock Lecture] 11. Put a trigger on the STR section

2014. 2. 18.

0

[Rude Lecture] 10. Relocation table

2014. 2. 18.

0

이 블로그 인기글

MPQ 가지고놀기 (1) - 간단한 MPQ 파일 분석

2013. 10. 19.

11

5. SFmpq (ShadowFlare's MPQ Library) 와 예제

2013. 9. 11.

1

[뻘강의] 13. 트리거 프로그래밍 - TRIG-MRGN 루프

2014. 2. 24.

0

4. scenario.chk

2013. 9. 10.

whyask37's blog

[뻘강의] 2. 데스 사이의 대입, 더하기

2014. 1. 19.

1

back to top

View in PC version

0