# 공간 파서 생성기 SPG의 구현

정 석 태† · 정 성 태†


## 요 약

본 논문에서는 GUI(Graphical User Interface)를 사용하여 사용자가 상호 작용적으로 도형 언어(visual language)의 CMG(Constraint Multiset Grammars) 문법을 기술함으로서 자동으로 공간 파서를 생성하는 공간 파서 생성기 SPG(Spatial Parser Generator)의 구현에 대하여 논한다. 본 시스템의 장점은 다음과 같다. (1) 사용자가 도형 언어의 문법을 정의하고 실제로 파싱하고 싶은 도형 언어를 입력하는데 사용되는 도형 에디터를 가지고 있다. (2) 사용자가 도형을 이용하여 대략적인 CMG 문법을 자동으로 생성한 뒤, 수정하여 최종적인 CMG 문법을 정의하도록 한다. (3) 제약 해소기(Constraint solver)를 가지고 있어서 파싱된 도형 언어들이 그 생성 규칙에 쓰여져 있는 제약을 유지한다.


# Implementation of a Spatial Parser Generator SPG

Sucktae Joung† · Sungtae Jung†


## ABSTRACT

We developed a spatial parser generator, SPG, which can automatically create a spatial parser if CMG(Constraint Multiset Grammars) grammars for a visual language are provided by the user with GUI(Graphical User Interface). SPG has the following features. (1) The user uses a visual editor to define the grammars of a visual language and draw the visual language which should be parsed. (2) The user roughly defines CMG grammars in a visual way at first. Then the user modifies them and defines final grammars. (3) Because SPG has a constraint solver, it maintains constraints in the parsed visual language according to the grammars.


**키워드 : 도형 언어(visual language), 공간 파서 생성기(spatial parser generator), CMG(Constraint Multiset Grammars)**

## 1. 서 론

현재, 도형은 공정도, 조직 구성도, 시스템 순서도, 회로도, 데이터베이스 분야에서 실세계의 데이터 구조를 기술하는데 이용되는 ER(Entity-Relationship) 다이어그램[1] 등 정보를 이해하기 쉽게 시각적으로 표현함으로 여러 분야에서 사용되어 지고 있다. 이러한 도형 중에 도형 요소나 도형의 구성 요소의 배치 규칙이 확실히 정해져 있고, 문장이나 수식과 같이 도형의 구성 요소의 조합에 대한 의미를 알아낼 수 있는 도형을 특히 도형 언어(visual language)라고 한다[2].

회로도나 ER 다이어그램과 같은 도형 언어는 전용 에디터를 이용하여 도형 언어의 구성 요소를 그리는 경우가 많다. 범용의 에디터를 이용하여 도형 언어의 구성 요소를 그리게 되면, 에디터가 그 도형 언어의 의미를 알 수 없기 때문에 사용자가 많은 조작을 해야한다. 예를 들면, 원의 중심에 라벨로서 문자가 쓰여져 있는 도형을 노드(node), 그 노드들을 연결하는 직선을 에지(edge)라 하고 이를 이용하여 그래프를 표현한다고 가정하자. 여기서 노드의 구성 요소는 원과 문자, 에지의 구성 요소는 직선, 그래프의 구성 요소는 노드와 에지이다. 노드를 이동시키기 위하여 원을 이동시키면 문자나 직선은 그 원의 움직임에 따르지 않아 그래프 구조를 상실하게 된다. 이것은 도형 언어의 구성 요소의 조합에 대한 의미 구조를 갖지 않은 도형 언어로서 에디터가 간주하기 때문에 생기는 문제이다. 그러기 때문에 전용 에디터는 도형 언어를 처리하기 위한 시스템이라고 볼 수 있다. 도형 언어는 매우 다양하므로 모든 도형 언어를 처리할 수 있는 시스템이 존재한다고는 볼 수 없다. 기존의 도형 언어를 처리할 수 있는 시스템은 특정한 도형 언어의 사양에 고정되어 있어서, 새로운 도형 언어를 정의하기 위해서는 기존의 시스템을 변경시키거나 새롭게 시스템을 구축해야 한다. 이러한 작업은 어려운 과정을 필요로 하며 상당히 많은 시간을 필요로 한다. 그러므로 사용자가 도형 언어를 정의하고, 그 처리기를 손쉽게 작성할 수 있도록 하는 것이 중요하다.

텍스트 언어로 프로그래밍하는 경우를 생각해 보자. 텍스트 언어로 프로그램을 작성할 경우에 먼저 어떤 언어를 사용하여 정해진 문제를 프로그래밍하면 처리기가 그 프로그램을 번역 또는 해석하여 실행 가능하도록 만든다. 언어는 일반적으로 처리기에 의존하지 않고, 하나의 언어는 여러 개의 처

※ 이 논문은 2001년도 원광대학교의 교비 지원에 의해서 연구됨.
† 종신회원 : 원광대학교 컴퓨터 및 정보통신공학부 교수
논문접수 : 2001년 8월 2일, 심사완료 : 2002년 5월 16일

리기를 가질 수 있다. 이렇게 텍스트 언어에서는 언어와 처리기를 분리함으로써 구문 분석기를 자동으로 생성해주는 파서 생성기 YACC(Yet Another Compiler Compiler)[3]를 만드는 것이 가능하게 됐다. 도형 언어도 언어와 처리기를 분리하여 생각하면, 처리기에 의존하지 않은 도형 언어를 정의할 수 있으며, 도형 언어의 문법(도형의 형태나 배치 등에 관한 사양)을 기술함으로써 자동으로 공간 파서(Spatial Parser)를 생성하는 공간 파서 생성기를 구현할 수 있다.

본 논문에서는 GUI(Graphical User Interface)를 사용하여 사용자가 손쉽고 상호 작용적으로 도형 언어의 문법을 기술함으로써 자동으로 공간 파서를 생성하는 공간 파서 생성기 SPG(Spatial Parser Generator)의 구현에 대하여 논한다.

## 2. 도형 언어의 정의 문법

도형 언어의 문법을 기술하는 방법에는 PG(Positional Grammars)[4], RG(Relational Grammars)[5], PLG(Picture Layout Grammars)[6], CMG(Constraint Multiset Grammars)[7, 8]등이 있다.

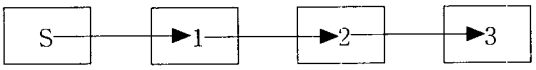
PG에서는 도형 언어의 구성 요소에 대한 속성으로 하나의 좌표만을 가지는 것에 반해 CMG에서는 도형 언어의 구성 요소에 대한 속성으로 여러 종류의 좌표를 정의할 수 있으므로 PG보다 도형 언어의 구성 요소간의 관계를 충분히 기술할 수 있다. RG에서는 두 개의 도형 언어의 구성 요소에 대한 속성간의 관계 밖에 기술하지 못하지만 CMG에서는 임의의 수의 도형 언어의 구성 요소에 대한 속성간의 관계를 기술할 수 있다. PLG는 CMG와 도형 언어를 표현하는 능력은 같으나 도형 언어의 구성 요소에 대한 속성의 네거티브(negative) 관계(속성간에 성립하지 않아야 할 관계)를 기술하지 못하므로 CMG보다 백 트랙킹하지 않고 결정적으로 파싱하는 능력이 부족하다. 그러므로 본 논문에서는 CMG를 사용하여 도형 언어의 문법을 기술한다. 또 도형 언어의 문법을 기술하는 방법으로 CMG를 사용하는 이유 중의 하나는 CMG 이외의 문법은 도형 언어의 구성 요소의 속성, 구성 요소의 종류와 수, 구성 요소의 속성간의 제약(constraint) 등을 모두 함께 기술하기 때문에 문법의 의미를 이해하는데 어려움이 있지만 CMG는 이것들을 따로 따로 기술하므로 이해하기 쉽다는데 있다. 여기서 제약이란, 도형 언어의 구성 요소의 속성간에 성립되어져 있는 관계를 말한다.

구체적으로 CMG는 터미널 심볼(terminal symbol)의 집합, 넌 터미널 심볼(nonterminal symbol)의 집합, 시작 심볼(start symbol), 생성 규칙(production rule)의 집합으로 구성된다. 모든 각 심볼은 색깔, 크기, 위치 등과 같은 속성을 갖는다. 생성 규칙은 토큰(token, 터미널 심볼 또는 넌 터미널 심볼의 인스턴스)의 멀티 세트(multiset)가 이것들의 속성간의 제약을 만족할 경우 새로운 심볼을 생성하는 규칙이다. 여기서 사용되어지는 토큰의 멀티 세트를 CMG에서는 normal의 구성 요소라고 한다. 그 밖에 exist, not_exist, all의 구성 요소가 존재한다. exist의 구성 요소는 새로운 심볼을 생성하는데 존재할 필요가 있는 토큰의 멀티 세트를 의미한다. 나머지 구성 요소의 상세한 것은 참고 문헌 [7]과 [8]을 참조하기 바란다.

(그림 1)과 같은 리스트 구조를 생각해 보자. 리스트 구조의 문법을 정의하기 위해서 다음과 같은 두 가지 생성 규칙이 필요하다.

**생성 규칙1** : 사각형의 중심에 라벨로서 S가 쓰여져 있는 도형을 리스트라고 한다.

**생성 규칙2** : 하나의 리스트가 화살표에 의해서 사각형에 이어져 있고, 그 사각형의 중심 에 라벨로서 숫자가 쓰여져 있는 도형을 리스트라고 한다.




(그림 1) 리스트 구조

이 생성 규칙1과 2를 CMG로 기술하면 다음과 같다.

```
 1 : list(point mid) :: = R : rectangle, T : text
 2 :     where (
 3 :         R.mid == T.mid &&
 4 :         T.text == "S"
 5 :     ) {
 6 :     mid = R.mid ;
 7 : }
 8 :
 9 : list(point mid) :: = R : rectangle, T : text, L : line, LL : list
10 :     where (
11 :         R.mid == T.mid &&
12 :         R.mid == L.end &&
13 :         LL.mid == L.start
14 :     ) {
15 : mid = R.mid ;
16 : }
```

생성 규칙1은 (그림 1)의 라벨 S의 사각형을 리스트(list)로 파싱시키기 위한 생성 규칙으로서 1행부터 7행까지가 CMG의 정의이다. 1행은 구성 요소가 사각형(R)과 텍스트(T)인 넌 터미널 심볼 list를 정의하고 있으며, 그 list의 속성으로서 중심(mid)을 갖고 있음을 나타내고 있다. 2, 3, 4, 5행은 list의 구성 요소간의 제약을 정의하고 있다. 3행은 사각형의 중심(R.mid)과 텍스트의 중심(T.mid)이 같다는 제약을 나타내고 있다. 4행은 텍스트의 문자열(T.text)이 S이여야 함을 나타낸다. 3, 4행의 제약을 만족했을 경우에 사각형(R)과 텍스트(T)를 list로 파싱하고 6행을 수행한다. 6행은 list의 속성인 중심(mid)에 사각형(R)의 중심 값(R.mid)을 대입함을 의미한다.

생성 규칙2는 라벨이 쓰여져 있는 사각형에 하나의 list가 연결되어 있는 도형을 list로 파싱시키기 위한 생성 규칙으로서 9행부터 16행까지가 CMG의 정의이다. 12행은 사각형의 중심과 직선의 종점(L.end)이 일치하는 제약, 13행

은 list의 중심(LL.mid)과 직선의 시작점(L.start)이 일치하는 제약을 의미하고 있다.

## 3. 공간 파서 생성기 SPG의 구현

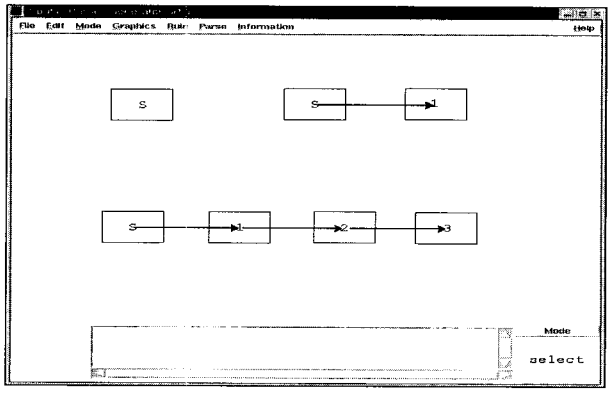
공간 파서 생성기 SPG는 스크립트 언어인 Tcl/Tk와 C언어를 사용하여 구현했으며, 다음과 같은 특징을 가지고 있다.

(1) 도형 에디터
(2) 도형에 의한 CMG의 정의
(3) CMG 문법에 의한 도형 언어의 파싱
(4) 도형 언어의 구성 요소간의 의미적 관계를 보전

### 3.1 도형 에디터

공간 파서 생성기 SPG는 (그림 2)와 같은 도형 에디터를 가지고 있다. 이 도형 에디터는 사용자가 도형 언어의 문법을 정의하고 실제로 파싱하고 싶은 도형 언어를 입력하는데 사용된다. 이러한 두 가지 작업을 하나의 화면상에서 직접 조작에 의해 수행할 수 있도록 한 이유는 보통 사용자는 도형 언어의 문법을 정의하면서 도형 언어를 입력하여 문법이 정확한가를 검사하는 일을 반복하므로 문법의 정의 모드와 파싱 모드를 한 화면에서 수행하는 것이 편리하기 때문이다.

도형 에디터에서는 도형의 복사, 삭제, 저장, 불러오기 등의 조작이 가능하다. 또 도형 에디터에서 다룰 수 있는 도형의 종류로는 타원, 사각형, 직선, 문자열 등이 있으며 이것들은 선의 굵기, 색깔, 폰트 등의 속성을 지정할 수 있도록 되어 있다.




(그림 2) SPG의 도형 에디터

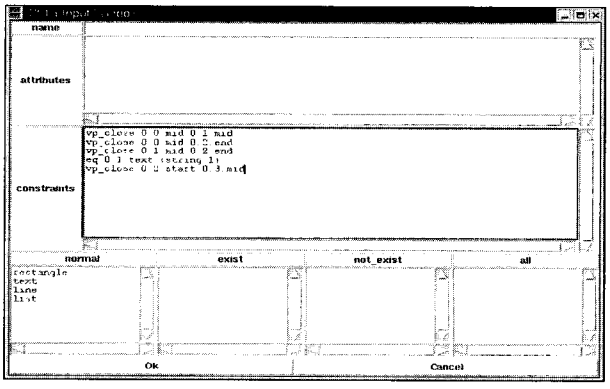
### 3.2 도형에 의한 CMG의 정의

CMG로 도형 언어의 문법을 정의하는 것은 생성 규칙을 정의한다는 것이며, 2차원 상에 존재하는 도형 언어의 구성 요소간의 관계를 다루는 것이다. 그런데 CMG는 다른 도형 언어의 문법을 기술하는 방법과 비교하면 이해하기 쉽다고 볼 수 있지만, 사용자의 입장에서 보면 1차원적인 텍스트로 문법을 정의하므로 감각적이지 못해서 기술하기가 어렵다. 또 문법을 처음부터 텍스트로 기술하기 때문에 CMG를 충분히 이해하지 못하면 도형 언어의 문법 정의가 불가능하게 된다.

공간 파서 생성기 SPG는 먼저 도형을 이용하여 대략적인 CMG 문법을 자동으로 생성한 뒤, 사용자로 하여금 수정하여 최종적인 CMG 문법을 결정하도록 한다. 예로서 2절에서 설명한 리스트 구조의 생성 규칙2를 정의하는 과정을 설명하겠다. 생성 규칙1은 이미 정의되어 있다고 가정하자.

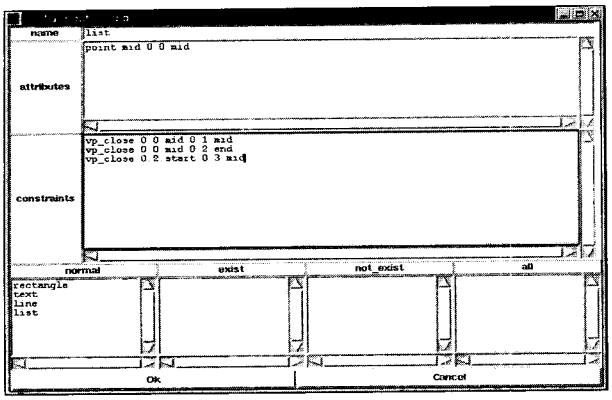
사용자는 하나의 넌 터미널 심볼로 정의하고 싶은 도형을 도형 에디터에 입력한다. (그림 2)의 상단의 왼쪽 도형은 생성 규칙1을 정의하기 위해 입력한 도형이고, 상단의 오른쪽 도형은 생성 규칙2를 정의하기 위해 입력한 도형이다. 다음에 사용자는 상단의 오른쪽 도형을 모두 선택하여 CMG 입력 윈도우((그림 3))를 연다. 그러면 SPG는 선택한 도형으로부터 구성 요소와 구성 요소간에 성립되어 있는 제약을 CMG 입력 윈도우에 자동으로 생성시킨다. CMG 입력 윈도우는 위에서부터 순서적으로 넌 터미널 심볼의 이름(name), 속성(attributes), 제약(constraints), 구성 요소(normal, exist, not_exist, all)를 정의하는 부분으로 구성되어 있다. 여기서 사용자는 제약을 수정하고 넌 터미널 심볼의 이름, 속성을 추가시킴으로써 생성 규칙을 정의해 나간다((그림 4)). (그림 4)의 윈도우의 각 부분에 쓰여져 있는 내용은 2절에서 CMG로 기술한 생성 규칙2(9행부터 16행까지)의 내용과 동일하다.

CMG 입력 윈도우에서 제약을 정의하는 부분에 기술되는 제약으로는 eq(equal), neq(not equal), gt(greater than), ge(greater or equal), lt(less than), le(less or equal), vp_close가 있다. 또 제약을 정의하는 방법은 "제약명 변수1 변수2"와 같이 기술한다. 제약명은 eq, neq, gt, ge, lt, le이고 변수1과 변수2는 정의하려고 하는 넌 터미널 심볼의 구성 요소가 되는 터미널 심볼 혹은 넌 터미널 심볼의 속성을 나타내고 있다. 여기서 제약 중 eq는 심볼의 속성을 나타내는 변수1과 변수2의 값이 같음을 표시하는 제약이고, vp_close는 변수1과 변수2의 값이 어느 정도 가까운 경우임을 표시하는 제약이다. 속성 값간의 가까운 정도는 SPG 시스템에 의해 결정된다.




(그림 3) CMG 입력 윈도우(1)

심볼에 대한 속성의 참조는 "구성 요소의 종류. 구성 요소의 순서. 속성명"의 형태이다. 구성 요소의 종류는 구성




(그림 4) CMG 입력 윈도우(2)

요소가 되는 심볼이 normal의 구성 요소이라면 0이된다. 만약 exist, not_exist, all의 구성 요소라면 각각 1, 2, 3이 된다. 구성 요소의 순서는 구성 요소의 종류 중에서 몇 번째의 구성 요소인가를 나타낸다 (0부터 시작한다). 예를 들면 (그림 4)의 CMG 입력 윈도우에서 제약을 정의하는 부분에 기술되는 첫 번째 제약은 "vp_close 0.0.mid 0.1.mid"라고 되어 있다. normal의 구성 요소 중 0번째의 구성 요소(rectangle)의 속성인 중심(0.0.mid)과 normal의 구성 요소 중 1번째의 구성 요소(text)의 속성인 중심(0.1.mid)이 어느 정도 가까운 값을 갖는다는 제약을 나타내고 있다.

### 3.3 CMG 문법에 의한 도형 언어의 파싱

이 절에서는 CMG 문법에 의한 도형 언어의 파싱 알고리즘에 관하여 설명하겠다.

먼저, 사용자가 CMG로 기술한 도형 언어의 생성 규칙들은 생성 규칙 데이터베이스 PDB(Production rule DataBase)에 보존된다. 생성 규칙의 내부 표현은 <표 1>과 같다. 단 p-id는 각각의 생성 규칙에 부여되는 식별자이다. 또 파싱하고 싶은 도형 언어(리스트 구조의 예로는 (그림 2)의 제일 밑에 있는 도형)를 도형 에디터에 입력하면 도형 언어의 모든 터미널 심볼의 토큰은 토큰 데이터베이스 TDB(Token DataBase)에 저장된다. 그 내부 표현은 TDB(t-id.속성명)의 형태이다. 여기서 t-id는 각각의 토큰에 부여되는 식별자이다. 예를 들어 2라는 t-id를 갖는 토큰의 속성인 중심(mid)은 TDB(2.mid)로 표시된다.

〈표 1〉 생성 규칙의 내부 구조

| 생성 규칙의 요소 | 내 용 |
|---|---|
| PDB(p-id.name) | 심볼의 이름 |
| PDB(p-id.attributes) | 심볼의 속성의 리스트 |
| PDB(p-id.constraints) | 심볼의 구성 요소간의 제약의 리스트 |
| PDB(p-id.normal) | 심볼의 normal의 구성 요소의 리스트 |
| PDB(p-id.exist) | 심볼의 exist의 구성 요소의 리스트 |
| PDB(p-id.not_exist) | 심볼의 not_exist의 구성 요소의 리스트 |
| PDB(p-id.all) | 심볼의 all의 구성 요소의 리스트 |

이렇게 생성 규칙 데이터베이스와 토큰 데이터베이스가 생성되어 지면 실제적으로 도형 언어의 파싱은 다음과 같은 알고리즘에 의해서 실행된다.

```
repeat
  foreach PDB의 생성 규칙
    구성 요소의 후보가 될 수 있는 토큰의 조합 리스트를 구한다
    foreach 토큰의 조합 리스트
      if 생성 규칙의 제약을 만족하는가 then
        새롭게 생성되는 토큰을 TDB에 추가
        생성에 사용된 토큰을 TDB에서 삭제
      end if
    end foreach
  end foreach
until TDB가 변경되지 않음
```

### 3.4 도형 언어의 구성 요소간의 의미적 관계를 보전

도형 언어의 구성 요소간의 의미적 관계를 보전하기 위해서는 제약 해소기(constraint solver)가 필요하다. 쉬운 예를 들면, 3개의 변수 a, b, c간에 a + b = c라는 제약이 성립되어 있다고 하자. 지금 a, b값이 각각 3, 4라고 한다면 c값은 7이다. 이때 a값이 5로 변했다면 그 제약이 성립하게 하기 위해서 c값은 9가 되던가 아니면 b값이 2가 되어야 한다. 또 다른 예로는 3개의 변수 a, b, c간에 (1) a = b, (2) b = c라는 2개의 제약이 성립되어 있다고 하자. 지금 a값이 5라고 한다면 b, c의 값은 각각 5가 된다. 이때 c값이 8로 변했다면 (2)에 의해서 b값은 8이 되고, 그 영향을 받아서 (1)에 의해 a값이 8이 된다. 이와 같이 여러 변수간에 성립되어 있는 제약을 어떤 변수 값이 변경되더라도 제약을 유지시키려는 기능을 제약 해소기라고 한다.

공간 파서 생성기 SPG는 도형 언어가 3.3절에서 설명한 알고리즘에 의해서 파싱되면 그 도형 언어의 생성 규칙에 쓰여져 있는 제약이 제약 해소기에 저장됨으로써 그 구성 요소간에 의미적 관계를 유지한다. 그러기 때문에 사용자가 도형 언어를 편집하는데 있어서 넌 터미널 심볼로 파싱된 토큰은 자동으로 하나의 독립적인 구성 요소로 그룹화되어 이동될 수 있다. (즉 보통 에디터에서 그룹화하여 사용하는 것과 같은 효과를 가지는 것이다). SPG는 제약 해소기로서 SkyBlue[9]를 사용한다. SkyBlue는 C언어로 구현되어 있는데, Tcl에서 SkyBlue를 사용하게 하기 위해서 인터페이스를 작성했다. 또 SPG는 토큰의 속성에 대한 값으로 SkyBlue의 변수를 갖도록 함으로써 각 속성 값이 변할 경우에 제약 해소를 SkyBlue가 수행하도록 했다.

예를 들어 토큰A가 토큰B, C를 구성 요소로 가지고 있다고 가정하고, 토큰B와 C의 t-id는 각각 1과 2라고 하자. 토큰B와 C의 속성인 중심(mid)은 각각 TDB(1.mid)와 TDB(2.mid)로 표시되는데, 이 값들은 SkyBlue의 변수를 갖고, SkyBlue의 변수가 실제 토큰의 중심 값을 갖게된다. 토큰B, C의 중심이 일치한다는 제약이 토큰A를 파싱하는 생성

규칙 중에 존재한다면, 그 제약이 토큰B, C의 속성간에 성립되어 SkyBlue에 저장된다. 사용자가 토큰A를 편집하는데 있어서 토큰B를 이동시키면 TDB(1.mid)의 값인 SkyBlue의 변수를 찾아 그 변수 값을 변경하고 TDB(2.mid)의 값인 SkyBlue의 변수를 찾아 그 변수 값을 변경함으로써 제약 해소를 SkyBlue가 수행하도록 하여 그 결과 토큰A가 이동되는 것과 같이 된다.

## 4. 공간 파서 생성기 SPG을 이용한 공간 파서 생성의 예

이 절에서는 오토마타의 표현이 어떤 구조의 스트링을 인식하는지를 쉽게 알 수 있도록 나타낸 일종의 흐름도인 상태 전이도의 공간 파서를 공간 파서 생성기 SPG을 이용하여 생성하는 방법을 설명하겠다. 상태 전이도의 문법을 정의하기 위해 다음과 같은 생성 규칙이 필요하다.

**생성 규칙 1** : 원의 중심에 S가 쓰여져 있는 도형을 start-Node라고 한다.

**생성 규칙 2** : 이중 원의 중심에 상태명이 쓰여져 있는 도형을 finalNode라고 한다.

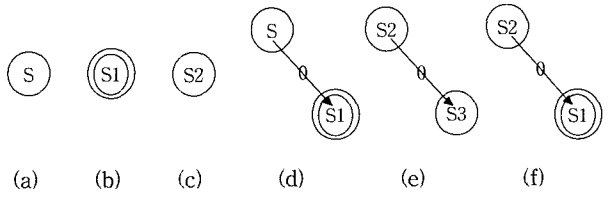
**생성 규칙 3** : 원의 중심에 상태명이 쓰여져 있는 도형을 normalNode라고 한다.

**생성 규칙 4** : startNode와 normalNode간의 상태 천이를 나타내는 직선과 그 직선의 중심에 입력 심볼이 쓰여져 있는 도형을 transition이라고 한다.

**생성 규칙 5** : normalNode간의 상태 천이를 나타내는 직선과 그 직선의 중심에 입력 심볼이 쓰여져 있는 도형을 transition이라고 한다.

**생성 규칙 6** : normalNode와 finalNode간의 상태 천이를 나타내는 직선과 그 직선의 중심에 입력 심볼이 쓰여져 있는 도형을 transition이라고 한다.

SPG에서 생성 규칙1, 2, 3, 4, 5, 6을 기술하기 위해 각각 (그림 5)의 (a), (b), (c), (d), (e), (f)와 같은 도형을 도형 에디터에 입력한다.




(그림 5) 상태 전이도의 문법 정의를 위한 도형

그리고 각각 도형들을 선택하여 CMG 입력 윈도우를 열어 다음과 같이 기술한다. 여기서는 이해하기 쉽게 하기 위해 2절의 CMG 기술 형태를 취하겠다.

**생성 규칙 1**

```
1 : startNode(point mid) :: = C : circle, T : text
2 : where (
3 : C.mid == T.mid &&
4 : T.text == "S"
5 : ) {
6 : mid = C.mid ;
7 : }
```

**생성 규칙 2**

```
1 : finalNode(point mid) :: = C1 : circle, C2 : circle, T : text
2 : where (
3 : C1.mid == C2.mid &&
4 : C1.mid == T.mid
5 : ) {
6 : mid = C1.mid ;
7 : }
```

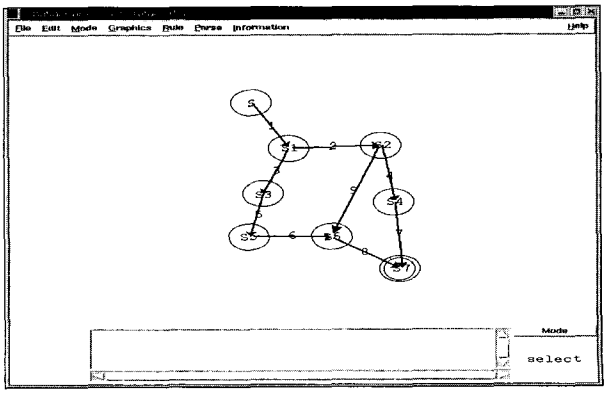
**생성 규칙 3**

```
1 : normalNode(point mid) :: = C : circle, T : text
2 : where (
3 : C.mid == T.mid
4 : ) {
5 : mid = C.mid ;
6 : }
```

**생성 규칙 4**

```
1 : transition(point mid) :: = S : startNode, L : line, T : text
2 : where ( exist N : normalNode
3 : where (
4 : S.mid == L.start &&
5 : N.mid == L.end &&
6 : L.mid == T.mid
7 : )
8 : ) {
9 : mid = L.mid ;
10 : }
```

**생성 규칙 4**에서 normalNode를 exist의 구성 요소로 구분한 것은 normalNode로부터 여러 개의 상태 천이가 있을 수 있기 때문이다. **생성 규칙 5**와 **생성 규칙 6**은 **생성 규칙 4**와 유사함으로 생략한다. 이렇게 생성 규칙을 정의한 다음, 상태 천이도를 (그림 6)과 같이 도형 에디터에 입력함으로써 파싱하는 것이 가능하게 된다.




(그림 6) 파싱 하고 싶은 상태 천이도

## 5. 관련 연구

공간 파서 생성기에 대한 관련 연구로서 GREEN(GRaphical Editing ENvironment)[10]와 SPARGEN(Spatial PARser GENerator)[11] 등이 있다. GREEN은 PLG를 사용하여 도형 언어의 문법을 정의함으로써 공간 파서를 생성하는 시스템이다. SPARGEN은 PLG를 확장한 OOPLG(Object-Oriented Picture Layout Grammars)를 사용하여 도형 언어의 문법을 정의함으로써 공간 파서를 생성하는 시스템이다.

이 시스템들과 본 논문의 시스템 SPG와의 차이점은 먼저 사용하는 문법이 다르다는 것이다. PLG나 OOPLG는 도형 언어의 구성 요소에 대한 속성의 네거티브 관계를 기술하지 못하여 파싱하는 능력이 부족하므로 폭 넓은 공간 파서의 생성이 불가능하다. 또한 이 시스템들은 문법을 정의할 때 문법을 처음부터 텍스트로 기술하기 때문에 PLG나 OOPLG의 충분한 이해가 없으면 문법 기술이 어렵다. SPG는 도형을 이용하여 대략적인 CMG 문법을 자동으로 생성한 뒤, 사용자로 하여금 수정하여 최종적인 CMG 문법을 결정하도록 한다. 마지막으로 이 시스템들은 파싱된 도형 언어들이 그 생성 규칙에 쓰여져 있는 제약을 유지하지 못한다는 것이다. 즉 제약 해소기를 가지고 있지 않다. 따라서 도형 언어의 구성 요소간의 의미적 관계를 보전하지 못하므로 사용자가 도형 언어를 편집하는데 어려움이 있다.

## 6. 결 론

본 논문에서는 GUI를 사용하여 사용자가 손쉽고 상호작용적으로 도형 언어의 CMG 문법을 기술함으로써 자동으로 공간 파서를 생성하는 공간 파서 생성기 SPG의 구현에 대하여 논했다. 본 시스템의 장점은 다음과 같다.

(1) 사용자가 도형 언어의 문법을 정의하고 실제로 파싱하고 싶은 도형 언어를 입력하는데 사용되는 도형 에디터를 가지고 있다.
(2) 사용자가 도형을 이용하여 대략적인 CMG 문법을 자동으로 생성한 뒤, 수정하여 최종적인 CMG 문법을 정의하도록 한다.
(3) 제약 해소기를 가지고 있어서 파싱된 도형 언어들이 그 생성 규칙에 쓰여져 있는 제약을 유지한다.

실제로 도형 언어를 파싱하는 공간 파서를 생성하는 것만으로는 응용 시스템을 작성하는데 부족함이 있다. 현재 도형 언어를 처리하는 시스템(비주얼 시스템)은 도형 언어를 파싱한 다음 그 결과를 이용하여 어떤 도형 언어의 구성 요소를 생성, 삭제, 수정을 수행하는 경우가 많다. 앞으로 CMG 문법에 액션(action)의 개념을 도입함으로써 이를 개선할 예정이다.

## 참 고 문 헌

[1] Chen. P, "The Entity-Relationship Model : Towards a Unified View of Data," *ACM Trans. Database System*, Vol.1, pp.9-36, 1976.
[2] 杉山 公造, "グラフ自動描畫法とその應用", コロナ社, 1993.
[3] 田中 正弘, "yaccとlexの使い方", HBJ出版局, 1992.
[4] Costagliola. G., Orefice. S., Polese. G., Tortora. G. and Tucci. M., "Automatic Parser Generation for Pictorial Languages," *Proc. IEEE Symposium on Visual Languages*, pp.306-313, 1993.
[5] Ferrucci. F., Tortora. G., Tucci. M. and Vitiello. G., "A Predictive Parser for Visual Languages Specified by relation Grammars," *Proc. IEEE Symposium on Visual Languages*, pp.245-252, 1994.
[6] Golin. E. J., "Parsing Visual Languages with Picture Layout Grammars," *Journal of Visual Languages and Computing*, No.2, pp.371-393. 1991.
[7] Marriott. K., "Constraint Multiset Grammars," *Proc. IEEE Symposium on Visual Languages*, pp.118-125, 1994.
[8] Marriott. K. and Meyer. B., "Towards a Hierarchy of Visual Languages," *Proc. IEEE Symposium on Visual Languages*, pp.196-203, 1996.
[9] Sannella. M., "Constraint Satisfaction and Debugging for Interactive User Interface," *Technical report*, University of Wasington, 1994.
[10] Golin. E. J. and Reiss. S. P., "The Specification of Visual Language Syntax," *Journal of Visual Languages and Computing*, No.1, pp.141-157, 1990.
[11] Golin. E. J. and Magliery. T., "A Compiler Generator for Visual Languages," *Proc. IEEE Symposium on Visual Languages*, pp.314-321, 1993.

**정 석 태**

e-mail : stjoung@wonkwang.ac.kr
1989년 전남대학교 전산학과 졸업
1996년 스꾸바대학 이공학연구과 석사 학위 취득
2000년 스꾸바대학 공학연구과 박사학위 취득
2001년~현재 원광대학교 컴퓨터 및 정보통신공학부 전임강사
관심분야 : 공간 파서 생성기, 비주얼 시스템, 오감 정보통신

**정 성 태**

e-mail : stjung@wonkwang.ac.kr
1987년 서울대학교 컴퓨터공학과 졸업
1989년 서울대학교 컴퓨터공학과 석사 학위 취득
1994년 서울대학교 컴퓨터공학과 박사 학위 취득
1994년~1995년 한국전자통신연구소 박사 후연수연구원
1999년~1999년 미국 Univ. of Utah 과학재단지원 해외 Post-Doc.
1995년~현재 원광대학교 컴퓨터 및 정보통신공학부 교수
관심분야 : VLSI / CAD, 영상 인식, 영상 기반 렌더링, 컴퓨터 그래픽스