공간 파서의 자동 생성 시스템에 대한 구현

정석태[0] 정성태
원광대학교 컴퓨터 및 정보통신공학부
{stjoung, stjung}@wonkwang.ac.kr

Implementation of a Spatial Parser generator

Suck-Tae Joung[0] Sung-Tae Jung
Dept. of Computer & Communication Engineering, Wonkwang University

요 약

본 논문에서는 GUI(Graphical User Interface)를 사용하여 사용자가 상호 작용적으로 도형 언어(visual language)의 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 문법을 자동으로 생성한 뒤, 수정하여 최종적인 문법을 정의하도록 한다. (3) 제약 해소기(Constraint solver)를 가지고 있어서 파싱된 도형 언어들이 그 생성 규칙에 쓰여져 있는 제약을 유지한다.

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]. 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.

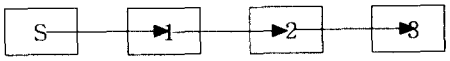
본 논문에서는 GUI(Graphical User Interface)를 사용하여 사용자가 손쉽고 상호 작용적으로 도형 언어의 문법을 기술함으로써 자동으로 공간 파서를 생성하는 공간 파서 생성기 SPG(Spatial Parser Generator)의 구현에 대하여 논한다.

2. CMG

본 논문에서는 CMG[3,4] (Constraint Multiset Grammars)를 사용하여 도형 언어의 문법을 기술한다. 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. 그밖에 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 [3]과 [4]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 중심에 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, 13행은 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.

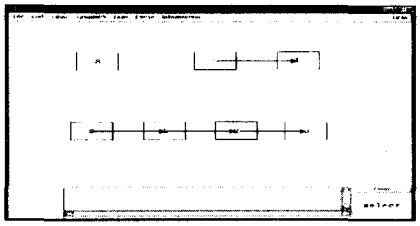
(1) 도형 에디터
(2) 도형에 의한 CMG의 정의
(3) CMG 문법에 의한 도형 언어의 파싱
(4) 도형 언어의 구성 요소간의 의미적 관계를 보전

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.

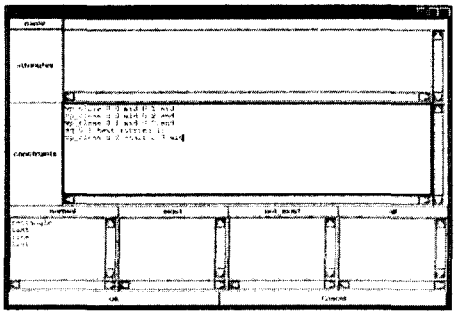
3.2 도형에 의한 CMG의 정의

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.



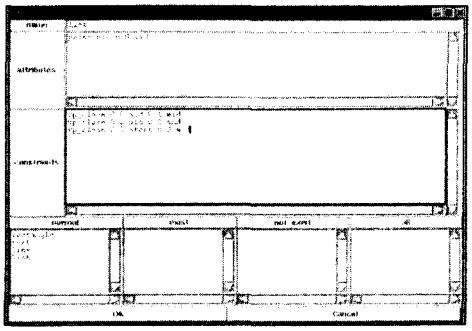
<그림 2> SPG의 도형 에디터

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.



<그림 3> CMG 입력 윈도우(1)

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



<그림 4> CMG 입력 윈도우(2)

값간의 가까운 정도는 SPG 시스템에 의해 결정된다.

심볼에 대한 속성의 참조는 "구성 요소의 종류. 구성 요소의 순서. 속성명"의 형태이다. 구성 요소의 종류는 구성 요소가 되는 심볼이 normal의 구성 요소이라면 0이 된다. 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. 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. 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)[5]가 필요하다.

SPG는 제약 해소기로서 SkyBlue[5]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.

4. 관련 연구

공간 파서 생성기에 대한 관련 연구로서 GREEN (GRaphical Editing ENvironment)[6]와 SPARGEN (Spatial PARser GENerator)[7] 등이 있다.

이 시스템들과 본 논문의 시스템 SPG와의 차이점은 SPG는 도형을 이용하여 대략적인 CMG 문법을 자동으로 생성한 뒤, 사용자로 하여금 수정하여 최종적인 CMG 문법을 결정하도록 한다. 또한 이 시스템들은 파싱된 도형 언어들이 그 생성 규칙에 쓰여져 있는 제약을 유지하지 못한다는 것이다. 즉 제약 해소기를 가지고 있지 않다.

5. 결론

본 논문에서는 GUI를 사용하여 사용자가 손쉽고 상호 작용적으로 도형 언어의 CMG 문법을 기술함으로써 자동으로 공간 파서를 생성하는 공간 파서 생성기 SPG의 구현에 대하여 논했다.

참고 문헌

[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] Marriott. K., "Constraint Multiset Grammars", *Proc. IEEE Symposium on Visual Languages*, pp. 118-125, 1994.

[4] Marriott. K. and Meyer. B., "Towards a Hierarchy of Visual Languages", *Proc. IEEE Symposium on Visual Languages*, pp. 196-203, 1996.

[5] Sannella. M., "Constraint Satisfaction and Debugging for Interactive User Interface", *Technical report*, University of Wasington, 1994.

[6] 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.

[7] Golin. E. J. and Magliery. T., "A Compiler Generator for Visual Languages", *Proc. IEEE Symposium on Visual Languages*, pp. 314-321, 1993.