04장 의미 분석

4.8 연습문제

4.23 임시 속성 공간 관리를 사용해서 연습문제 4.14를 반복하라. 그러나 변환을 자료 구조에 축적하지 말고 바로 파일에 출력하라.

4.24 속성 흐름이 L 속성을 가져야 한다는 요구 조건 없이 ©(심화학습)에 있는 예 4.25의 선언을 위한 문법을 다시 작성하라. 문법을 최대한 간단하고 간명하게 만들어 보자(식별자 리스트를 모을 필요는 없다).

4.25 ©(심화학습)에 있는 그림 4.16에서 생략한 행들을 채워라.

4.26 다음과 같은 동작 루틴을 보자.

```
params → mode ID par_tail
    { params.list := insert(mode.val, ID.name, par_tail.list) }
par_tail → , params { par_tail.list := params.list }
         → { par_tail.list := nil }
mode → IN { mode.val := IN }
     → OUT { mode.val := OUT }
     → IN OUT { mode.val := IN OUT }
```

입력 `IN a, OUT b`를 구문 분석하고 있으며, 컴파일러는 구문 트리의 활성화된 부분을 유지하기 위해 자동으로 유지되는 속성 스택을 사용한다고 가정하자. 구문 분석기가 생산 *par_tail* → ϵ를 예측하기 직전의 속성 스택 내용을 보여

라. 속성 스택에서 lhs와 rhs가 가리키는 곳을 반드시 나타내자. 또 저장된 lhs와 rhs 값을 보이고 속성 스택에서 각기 어디를 가리키는지도 보여라. ssf (지금까지 본, seen so far) 포인터는 무시해도 좋다.

4.27 하향식 구문 분석기에서 속성을 위한 자동 공간 관리의 문제 중 하나는 리스트와 나열에서 발생한다. 예로 다음과 같은 문법을 생각해보자.

block → begin *stmt_list* end
stmt_list → stmt *stmt_list_tail*
stmt_list_tail → *stmt_list* | ϵ
stmt → ...

n개의 문장으로 구성된 블록의 마지막 문장을 예측한 후 속성 스택은 다음을 포함하게 된다(행 바꿈과 들여쓰기는 명확성을 위한 것뿐이다).

```
block begin stmt list end
stmt stmt list tail ; stmt list
stmt stmt list tail ; stmt list
stmt stmt list tail ; stmt list
{ n times }
```

속성 스택의 크기가 유한하다면 일직선으로 배열된 코드에서 길지만 유효한 블록에 대해 오버플로우는 반드시 발생한다. 축적된 출력 코드는 제외하고 일단 속성 스택의 하위 구조가 구문 분석되면 속성 스택에서 반복되는 기호들은 어떤 유용한 속성도 포함하지 않기 때문에 문제는 더욱 심각해진다.

속성 스택에서 불필요한 기호들을 동적으로 "제거하는" 기술을 제안하라. 이상적으로 제안하는 기술은 컴파일러 작성자에게 부담을 주지 않기 위해 자동 구현이 가능해야 한다.

또 자동으로 관리되는 속성 스택을 이용하지만 불필요한 기호를 제거하지는 않는 하향식 구문 분석기를 가지는 컴파일러를 사용한다고 가정하자. 어떤 프로그램이 컴파일러로 하여금 스택 공간을 모두 소진하게 한다면 어떤 조치를 취할 수 있는가? 이 문제를 "해결하기 위해" 프로그램을 어떻게 수정할 수 있는가?