대한수학회 회보
제11권 (1974), pp. 39-43

컴파일러 構成時 最適造織
—ALGOL 60의 立體造織概念의 直線化研究를 中心으로—

申 鉉 千　　金 榮 澤

序　論

프로그램잉 言語의 컴파일링은 言語의 構成形態에 따라 方法을 다르게 하고 있다. 이들은 대개 有限한 構造를 이루고 있기 때문에 言語에 따라서 目的을 中心으로 그 경제성의 向上에 置重하게되면 컴파일하는 양식은 言語마다 컴파일링의 方法을 다르게 하는 경우가 많다.

Algol 60에서는 言語의 構成을 Syntax 中心으로 組織하였기 때문에 이 組織의 컴파일링이 매우 중요하지 않을 수 없다. 特히 syntax 構成中에서 recursion 개념의 컴파일링은 다른 言語에 比하여 복잡하고 recursion 개념을 完全히 消化하는데는 여러가지 검토가 이루어져야 하겠다.

본 논문에서는 言語의 全體的 構造를 컴파일하는데 치중하였으며 각 statement 中에서 單純한 型은 post fix notation으로 解決되는 가정을 하였다.

特히 Cheathum의 개념도 本 컴파일링에서는 관련하지 않으며 可及的 statement 컴파일링은 간단한 分析으로 解決코저한다. 이러한 간단한 statement를 單位 statement로 分類하며 本論에서 言語의 全體的 컴파일링을 다루고저 한다.

本　論

Algol 60의 構成에서 單位 statement를 定義하고 또한 全體言語를 部分別로 나누어 보았다. 單位 statement는 〈basic statement〉나 이와 비슷한 形態를 갖춘 것들을 말하는데 컴파일링 하는데 있어서 이들 statement는 recursion 개념을 構成하지 않는다. 이 單位 statement는 컴파일時에 여러가지 方法을 이용할 수 있으며 全體的인 경향을 보아서 그중에서 어떤 方法을 택하는 것은 가능하다.

이 單位 statement들을 群으로 構成하여 次上의 單位를 形成하는 statement를 살펴보면 몇 개의 群으로 나누어 생각할 수 있다. 이들은 單位 statement가 될 수도 있고 많은 單位 statement를 合하여 自己 statement를 構成하기도

한다. 이들 複合 statement는 nesting 개념이나 recursion 개념을 포함하고 있기 때문에 복잡한 수차重複이 일어나고 있다. 그러기 때문에 이러한 statement는 수개의 單位 statement로 단순화 될때까지 分析하여 그 構成을 直線化시켜야 한다. 本論文에서는 이 直線化의 조직을 重點的으로 다루므로써 全體言語構造를 살펴 보고 새로운 컴파일링 기술을 硏究코저 한다.

① 〈simple boolean〉
② 〈simple arith expression〉
③ 〈arithmetic expression〉
④ 〈boolean expression〉
⑤ 〈if clause〉
⑥ 〈block〉
⑦ 〈compound statement〉
⑧ 〈assignment statement〉
⑨ 〈go to statement〉
⑩ 〈dummy statement〉
⑪ 〈conditional statement〉
⑫ 〈for statement〉
⑬ 〈procedure statement〉
⑭ 〈type declaration〉
⑮ 〈array declaration〉
⑯ 〈switch declaration〉
⑰ 〈procedure declaration〉

(도표 1)

A gol 60의 statement.

Algol 60의 構成을 單位 statement나 或은 複合 statement로 區分하여 보면 위와 같이 나열할 수가 있다. 各 statement는 單位 statement나 이들 單位 statement를 nesting 型이나 或은 recursion 型으로 조직한 복합 statement로 組織된다.

〈array declaration〉
〈type declaration〉
〈procedure statement〉
〈for statement〉

〈conditional statement〉
〈go to statement〉
〈assignment statement〉
〈block〉

〈block〉의 구조

〈procedure statement〉
〈for statement〉
〈conditional statement〉
〈dummy statement〉
〈go to statement〉
〈assignment statement〉
〈block〉

(도표 2)

〈for statement〉의 구조

이 statement 들은 相互間에 얽혀서 構成되어 있기 때문에 보기의 block statement 와 for statement 에서와 같이 statement 구성시에 相互 recursion 의

(도표 3)

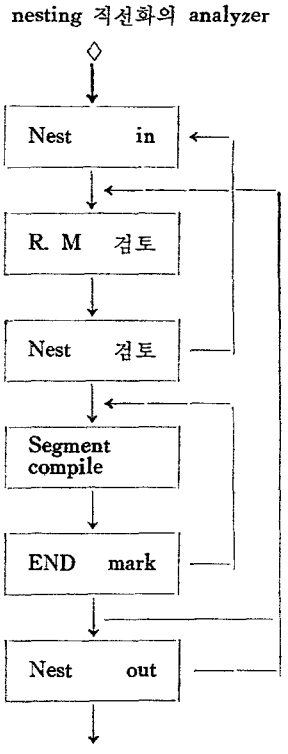
relation matrix

		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
1	s·B																	
2	s·a·e																	
3	a·e		/			/												
4	B·e																	
5	if clause				/													
6	b							/	/	/	/	/	/	/	/	/	/	
7	c·m·s							/	/	/	/	/	/	/				
8	a·s			/	/													
9	q·t																	
10	d·s																	
11	c·d·s				/		/		/	/	/	/	/	/				
12	f·s						/		/	/	/	/	/	/				
13	p·s																	
14	t·d																	
15	a·d																	
16	s·d																	
17	p·d						/	/	/	/	/	/	/	/		/	/	
			s a e	a e	B e	if	b	cm s	a s	g t	d s	cd s	f s	p s	t d	q d	s d	p d

관계를 갖고 있다. 이들의 相互構成 관계를 matrix로 表現해 보면 relation matrix와 같이 그 상호관계를 간단히 規定할 수 있다. 그러므로 이 상호관계의 relation matrix는 syntax error의 發見에도 使用될 수 있다. 복합 statement의 컴파일을 위해서는 그 構成上의 검사는 relation matrix에 의하여 이루어지므로 모든 recursion 은 parsing에 의하여 간소화 될 수가 있다. recursion이나 nesting 개념의 直線化를 이 parsing 개념으로 檢討코저 한다.

全體文章의 直線化를 위하여 部分的인 컴파일과 이들의 assembling을 생각할 수 있다. 컴파일이 진행되고 있는 statement를 current statement라고 하면 이 current statement는 relation matrix에 의하여 그 構造가 검토되고 nesting 양식으로 segmenting이 이루어 지면서 control의 tagging이 이루어져야 한다. 이것의 과정을 block diagram으로 살펴보면 다음과 같다.

(도표 4)



이들 nesting을 예문을 들어서 간단히 說明하면 begin에서 nesting이 시작되어 if statement의 begin에서 nesting이 또 한번 시작되며 속의 nesting이

compile되면 밖의 nesting이 compile되어 전체 프로그램은 직선화가 이루어진다. 이러한 개념은 nesting의 깊은 입체개념에도 상관없이 적용이 될 수 있다.

```
begin
  s;
  if a then begin s; s end;
  s;
end
```

위의 program은 analyzer로 이해와 분석이 되는 간단한 예가 되겠다.

結　論

全體프로그램이 nesting의 構造로 分析하는데 control tag의 적절한 연결로써 입체화된 프로그램은 직선화 되어도 그 내용 변화는 없게된다. 단지 입체의 개념과 직선의 개념의 차이가 object프로그램에서 어떠한 효과를 가져오느냐 하는 것은 object프로그램의 최적화에서 檢討되어야 하겠다. 이 연구에서는 입체개념의 직선화로 tree개념의 복잡한 algol 구조는 해석이 되고 있다.

참 고 문 헌

[1] E.W. Dijkstra, *Primer of algol* 60, Academic Press 1964.

[2] B. Randall, *Algol* 60 *implementation*, Academic Press 1964.

[3] S. Rosen, *Programming language*, McGrawhill 1967.