บทที่ 2

การออกแบบภาษา : วากยสัมพันธ์
(Language Design : Syntax)

2.1 ชุดอักขระ
2.2 ประมวลศัพท์
2.3 โครงสร้างวากยสัมพันธ์
2.4 การอธิบายวากยสัมพันธ์ของ COBOL
2.5 แผนภาพวากยสัมพันธ์
2.6 หัวข้ออื่น ๆ ในวากยสัมพันธ์
2.7 วากยสัมพันธ์ และการเขียนโปรแกรม
2.8 วากยสัมพันธ์ และความหมาย
2.9 วากยสัมพันธ์ ความหมาย และการออกแบบคอมไพเลอร์
แบบฝึกหัด

บทที่ 2
การออกแบบภาษา : วากยสัมพันธ์
Language Design : Syntax

ในบทนี้ เราศึกษาการพัฒนาหลักซึ่งนำไปสู่วิธีปัจจุบัน ในการให้นิยาม "วากยสัมพันธ์" ของภาษาชุดคำสั่ง

"วากยสัมพันธ์" ของภาษาชุดคำสั่ง หมายถึงเซตของกฎต่าง ๆ และข้อตกลงการเขียน ซึ่งเป็นรูปแบบของโปรแกรมถูกต้อง ในหนึ่งภาษา โดยมองที่การแทนที่เท่านั้น

(The "syntax" of a programming language, broadly speaking, is that set of rules and writing conventions that allow the formation of correct programs in a language, from the point of view of "representation" only.) นั่นคือ วากยสัมพันธ์ ไม่ได้กระทำกับ "ความหมาย" หรือ พฤติกรรม ณ เวลาดำเนินการ ของ โปรแกรม ตัวอย่างเช่นในรูป 2-1 เป็นโปรแกรมภาษา Pascal และ Fortran ถูกต้องตามวากยสัมพันธ์ แต่ไม่มีความหมายใด ๆ

แต่วากยสัมพันธ์ เป็นสิ่งต้องการล่วงหน้า ก่อนความหมายของนิพจน์ เช่นเดียวกับในภาษาอังกฤษ ดังนั้น ภาษาชุดคำสั่งต้องนิยามวากยสัมพันธ์ดี ก่อนการสนับสนุนอย่างถูกต้อง เพื่อพัฒนาของโปรแกรมที่มีความหมาย

รูป 2-1 ซ้ายมือ เป็นโปรแกรมภาษา Pascal และขวามือ เป็นโปรแกรมภาษา Fortran ถูกต้องตามวากยสัมพันธ์

Pascal	Fortran
program p; begin end.	END

การอธิบายวากยสัมพันธ์ของภาษาชุดคำสั่ง(the syntactic description of a programming language) มีการเข้าถึงอย่างมีเหตุผลหลายวิธีแต่ละวิธีมีทั้งความ แข็งและความอ่อน หลาย ๆ อย่าง (and along with each come various strengths and weakness) การเข้าถึง เหล่านี้ มอง 2 ด้าน คือ : ทางด้านนักออกแบบภาษา และ

ทางด้านนักเขียนโปรแกรม

นักออกแบบภาษา มีเป้าหมายที่จะ ทำให้การเกิดผล ของภาษาโปรแกรมชัดเจน เป็นผลสำเร็จ ในขณะที่ นักเขียนโปรแกรมต้องการ การแสดงออก และความสะดวกมากที่สุด ในโดเมน การเขียนโปรแกรม (The language designer aims to achieve a clean implementation of the language, while the programmer wants maximum expressivity and convenience in particular programming domain.) ดังนั้น ภาษาที่ดี จึงต้อง รวมเป้าหมายสองสิ่งนี้ เข้าด้วยกันเมื่อมันเป็นส่วนเติมเต็มของกัน และมีการประนีประนอมเมื่อมันไม่ตรงกัน

2.1 ชุดอักขระ (Character set)

ชุดอักขระของภาษา หมายถึงเซตของสัญลักษณ์ต่าง ๆ ซึ่งเอามาประกอบเข้าด้วยกันเป็นโปรแกรมทั้งหมด (The "character set" of a language is simply that set of symbols from which all programs are composed.) การเข้าถึงที่แตกต่างกันหลายอย่าง กับการเลือกชุดอักขระ เกิดขึ้นโดยภาษาที่แตกต่างกัน พูดอีกอย่างหนึ่ง คือ ภาษาต่างกัน จะเลือกชุดอักขระต่างกัน

ในช่วงปี ค.ศ. 1950s และ 1960s ภาษา ALGOL และ FORTRAN เป็นภาษาที่มองเห็นชัดเจนมากที่สุด ชุดอักขระของสองภาษานี้ แสดงให้เห็นในรูป 2-2 นักออกแบบภาษา Fortran ใช้ ชุดอักขระเชิงหน้าที่น้อยที่สุด คือเฉพาะอักษรตัวใหญ่ เลข และสัญลักษณ์พิเศษอีก 13 ตัวเท่านั้น สำหรับเขียนโปรแกรม เหตุผลสำคัญ สำหรับการตัดสินใจ ณ เวลานั้น คือ ขึ้นอยู่กับการพิมพ์ที่จำกัด ของ เครื่องมือเขียนโปรแกรม มาตรฐาน ได้แก่ เครื่องเจาะบัตร IBM 026 นอกจากนี้แล้ว ภาษา Fortran ยังได้รับการสนับสนุน อย่างแข็งขันมากที่สุดโดย IBM ดังนั้น การเกี่ยวข้องใกล้ชิด ระหว่าง ชุดอักขระ และรหัสเครื่องเจาะบัตร 026 จึงไม่น่าเป็นเรื่องประหลาดใจ

ตรงกันข้ามกับ นักออกแบบภาษา ALGOL ชุดอักขระนั้น สนับสนุน การพิมพ์ (publication) โปรแกรม ใน วารสารและหนังสือ ดังนั้น ชุดอักขระของ ALGOL จึงมีทั้ง อักษรตัวใหญ่ และอักษรตัวเล็ก และส่วนขยาย ของสัญลักษณ์พิเศษ เพิ่มเติม สำหรับเครื่องหมายวรรคตอนและนิพจน์คณิตศาสตร์ บางภาษาในช่วงต้นของปี ค.ศ.1960s ชุดอักขระที่เพิ่มเติมนั้น สนับสนุนโดยฮาร์ดแวร์ของคอมพิวเตอร์ เช่นเดียวกัน ดังนั้น ชุดอักขระของ ALGOL จึงบริการ การออกแบบฮาร์ดแวร์ได้ดีกว่า ขีดจำกัดของฮาร์ดแวร์ปัจจุบัน

รูป 2-2 a ชุดอักขระของภาษา ALGOL

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

a b c d e f g h i j k l m n o p q r s t u v w x y z

0 1 2 3 4 5 6 7 8 9

< ≤ = > ≥ ≠ ~ ∨ ∧ ⊃ ≡

+ - x ÷ ≠ , . ; : ' " ( ) [1 ƀ (blank)

รูป 2-2 b ชุดอักขระของภาษา Fortran

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

0 1 2 3 4 5 6 7 8 9

= + - * / ( ) , . $ ' : 6 (blank)

เนื่องจากชุดอักขระ เป็นสมาชิกพื้นฐาน ของ การออกแบบภาษาชุดคำสั่ง และการแทนที่ข้อมูล ในเครื่องคอมพิวเตอร์ จึงมีความพยายามอย่างมากที่จะพัฒนาชุดอักขระมาตรฐาน และการแทนที่ภายในเครื่องให้เป็นสากล (international) ผลลัพธ์ที่ได้ คือ มีชุดอักษร สองชุด เกิดขึ้น และกลายเป็นมาตรฐานสำหรับ ในทางอุตสาหกรรม ชุดหนึ่งเรียกว่า ASCII (American Standard Code for Information Interchange) และอีกชุดหนึ่ง เรียกว่า EBCDIC (Extended Binary Coded Decimal Interchange Code) โดยแรงผลักดันของสถาบัน มาตรฐานแห่งชาติอเมริกัน (American National Standards Institute) และโดย IBM ตามลำดับ ชุดอักขระเหล่านี้ แสดงให้เห็นในรูป 2-3

การออกภาษาชุดคำสั่งปัจจุบัน ส่วนใหญ่ สอดคล้อง (conform) กับชุดอักขระมาตรฐานเหล่านี้ ชุดใดชุดหนึ่ง หรือ ทั้งสองชุด ตัวอย่างเช่น มาตรฐาน Ada กำหนดว่า รหัส ASCII เป็นมูลฐานของมัน สำหรับการแทนที่ ค่าของสายอักขระ ทั้งหมด และโปรแกรม Ada

แต่ก็มีบางภาษา ซึ่งไม่สอดคล้องกับ มาตรฐานใด ๆ ในสองชุดนี้ เช่น ชุดอักขระ ALGOL ประกอบด้วย สัญลักษณ์หลายตัว ซึ่งไม่ใช่เซตของ ASCII หรือ EBCDIC เป็นข้อยกเว้น

รูป 2-3 a ชุดอักขระ ASCII

ASCII CHARACTER SET

bits 1,2,3								bits 4,5,6,7
000	001	010	011	100	101	110	111	
NUL	DLE	SP	0	@	P	`	p	0000
SOH	DC1	!	1	A	0	a	a	0001
STX	DC2	''	2	B	R	b	r	0010
ETX	DC3	#	3	C	S	c	s	0011
EOT	DC4	$	4	D	T	d	t	0100
ENQ	NAK	%	5	E	U	e	u	0101
ACK	SYN	&	6	F	V	f	v	0110
BEL	ETB	'	7	G	W	g	w	0111
BS	CAN	(	6	H	X	h	x	1000
HT	EM	)	9	I	Y	i	v	1001
LF	SUB	*	:	J	Z	j	z	1010
VT	ESC	+	;	K	[	k	{	1011
FF	FS	<		L	\	l			1100
CR	GS	•	=	M	1	m	}	1101
SO	RS	>		N	•	n	~	1110
SI	us	/	?	O		__0	DEL	1111

รูป 2-3 b ชุดอักขระ EBCDIC

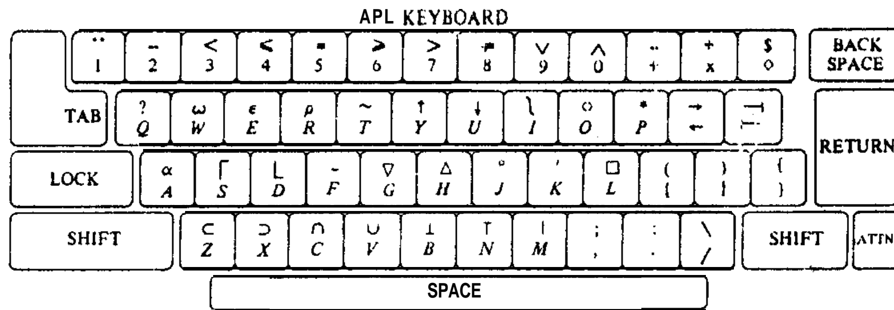
EBCDIC CHARACTER SET

bits 0.1.2.3

0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111	bits 4.5.6.7
NUL	DLE	DS		SP	&	-						{	}	\	0	0000
SOH	DC1	SOS				/		a	j	~		A	J		1	0001
STX	DC2	FS	SYN					b	k	s		B	K	S	2	0010
ETX	TM							c	l	t		C	L	T	3	0011
PF	RES	BYP	PN					d	m	u		D	M	U	4	0100
HT	NL	LF	RS					e	n	v		E	N	V	5	0101
LC	BS	ETB	UC					f	o	w		F	O	W	6	0110
DEL	IL	ESC	EOT					g	p	x		G	P	X	7	0111
GE	CAN							h	q	y		H	Q	Y	8	1000
RLF	EM							i	r	z		I	R	Z	9	1001
SMM	CC	SM		¢	!	:	:								|	1010
VT	CU1	CU2	CU3		$	,	#									1011
FF	IFS		DC4	<	*	%	@									1100
CR	IGS	ENQ	NAK	(	)	_	'									1101
SO	IRS	ACK		+		>	=									1110
SI	IUS	BEL	SUB	|	¬	?	"								EO	1111

ที่ประหลาดมาก ชุดอักขระของ APL มีจำนวน สัญลักษณ์มาก ซึ่ง มีความหมายเพียงอย่างเดียวกับตัวมันเท่านั้น สัญลักษณ์ส่วนใหญ่หมายถึง ฟังก์ชัน APL พิเศษ การทำให้เกิดผลใด ๆ ของ APL อยู่ที่ความหลากหลายของ เซตของตัวอักขระนี้ โดย ใช้แป้นพิมพ์ที่ออกแบบพิเศษ (ดูรูป 2-4) หรือ an alternative transliteration scheme สำหรับตัวอักขระพิเศษ ใน APL

รูป 2.4 แป้นพิมพ์ของภาษา APL



The APL keyboard.

ความจำเป็นของชุดอักขระพิเศษ ทำให้ภาษาชุดคำสั่งเหล่านี้ เข้าถึงได้น้อยลง โดยเฉพาะในการใช้งานทั่วไป ค่าใช้จ่าย ของ การออกแบบ หรือเพื่อซื้อ แป้นพิมพ์พิเศษ และเทอร์มินัล เป็นตัวยับยั้ง (deterrent) ที่สำคัญ คำถามมีว่าได้ประโยชน์หรือไม่ จากการมีสัญลักษณ์พิเศษ สำหรับฟังก์ชันพิเศษ โดยชั่งน้ำหนักกับค่าใช้จ่ายนี้ ยังคงเป็นคำถามเปิดอยู่

(The question of whether the extra benefit of having special symbols for special functions outweights this cost remains open.)

2.2 ประมวลศัพท์ (Vocabulary)

"ประมวลศัพท์" ของภาษาชุดคำสั่ง หมายถึง เซตของตัวอักขระ และคำที่ประกอบเข้าเป็นโปรแกรม (The "vocabulary" of a programming language is that set of characters and words from which programs are constructed.) ตัวอย่างเช่น ในรูป 6-5 โปรแกรม Pascal อยู่ทางซ้ายมือ และคำศัพท์ต่าง ๆ ที่ประกอบขึ้น คือ รายการ

ทางขวามือ

รูป 2-5 โปรแกรม Pascal (ซ้ายมือ) และคำศัพท์ของมัน (ขวามือ)

```
program  P;                      program   P        ;
   var x, y : integer;           var       x        :
   begin                         integer   y        (
      read(x);                   begin     read     )
      Y := x t 2.5;              end       write    :=
      write(y)                   2.5       t        .
   end.
```

สมาชิกของประมวลศัพท์ของภาษา เรียกว่า **โทเค็น** (The elements of a language's vocabulary are called its "tokens") โทเค็นเหล่านี้ ปกติ จะแบ่งออกเป็นประเภท ขึ้นอยู่กับบทบาทของมัน ใน ภาษา ตัวอย่างเช่น เรามี โทเค็น ซึ่งเป็น ไอเดนติไฟเออร์ (identifiers) ได้แก่ P, x, y, read, และ write

ค่าคงที่ ได้แก่ 2.5

ตัวปฏิบัติการ ได้แก่ :=, +

อักขระคั่น (delimeter) ได้แก่ **program, var, :, ;, (,), ., begin,** และ **end**

เซตของ โทเค็น สำหรับภาษาชุดคำสั่ง ถูกนิยามขึ้นในวิธี ซึ่งอนุญาต ให้รู้จำได้อย่าง**มีประสิทธิภาพ โดยตัวแปลชุดคำสั่ง** (กระบวนการนี้ เรียกว่า การวิเคราะห์ศัพท์) และแสดงออก**ชัดเจนโดย นักเขียนโปรแกรม**

(The set of tokens for a programming language is defined in such a way that permits efficient recognition by a compiler (a process called "lexical analysis") and clear expression by a programmer.)

ตัวรู้จำ ทีมีประสิทธิภาพ (an efficient recognizer) **ควรจะสามารถแยก**โปรแกรม ออกเป็น โทเค็นประกอบ ของมัน และแบ่งประเภท โทเค็น ได้ **ในการผ่านตัวโปรแกรมหนึ่งครั้ง** (in a single pass over the program text)

3-2

การรู้จำของ **โทเค็น หนึ่งสัญลักษณ์** (single-symbol tokens) เช่น + เป็นไปอย่างตรงไปตรงมา สำหรับไอเดนติไฟเออร์ และค่าคงที่ ต้องการบิตเพิ่มขึ้น อีก 1 บิต เพื่อที่จะได้ นิยามและแบ่งประเภทมันได้อย่างไม่กำกวม วิธีหนึ่งของการทำสิ่งนี้ คือใช้อุปกรณ์อภิภาษา (metalinguistic device)* เช่น **ไวยากรณ์ ไม่พึ่งบริบท** (context free grammar) ในการนิยามประเภทของ โทเค็นเหล่านี้ สิ่งนี้คือรูปแบบร่วมสำหรับการนิยาม วากยสัมพันธ์ของภาษาชุดคำสั่ง ยุคแรกนั้น พัฒนาโดย Noam Chomsky ในช่วงปี ค.ศ. 1950s สำหรับการนิยาม โครงสร้าง วากยสัมพันธ์ของภาษาอังกฤษ ต่อมาไวยากรณ์ ไม่พึ่งบริบท ได้รับการดัดแปลงโดยแบกคัส (Backus) และเนาร์ (Naur) และนำไปใช้ อธิบาย วากยสัมพันธ์ของภาษา ALGOL ในปีค.ศ. 1962 การดัดแปลงนี้ จึงได้ชื่อว่า "Backus-Naur form" หรือเรียกว่า BNF ตั้งแต่นั้นเป็นต้นมา BNF หรือ หนึ่งในหลาย ๆ รูปแบบของมัน ได้กลายเป็นเครื่องมือมาตรฐาน สำหรับ การนิยาม วากยสัมพันธ์ ของภาษาชุดคำสั่งต่าง ๆ

ตัวอย่าง สมมติว่าเราต้องการนิยาม ประเภท โทเค็น ซึ่งเป็น ไอเดนติไฟเออร์ตามความเข้าใจ ต่อไปนี้

ไอเดนติไฟเออร์ หมายถึง ตัวอักษรใด ๆ อาจจะตามด้วยลำดับของ ตัวอักษร และ/หรือ ตัวเลข

(An identifier is defined as any letter, followed optionally by any sequence of letters and/or digits.)

ในภาษาอังกฤษ คำจำกัดความชนิดนี้ ค่อนข้างงุ่มง่าม แต่อะไรที่เรากำลังนิยามอยู่นี้ คือ ประเภทหนึ่ง ของโทเค็น เหมือนกับสิ่งต่อไปนี้

X

Y

Alpha

---

*Metalinguistic device ได้แก่

(1) รูปแบบแบกคัส-เนาร์(บีเอ็นเอฟ)(Backus Naur form (BNF)) - อธิบาย syntax ของภาษา Pascal

(2) Meta language - อธิบาย syntax ของ COBOL

(3) Syntax diagram - อธิบาย syntax ของ Pascal

(4) Descriptive definition - อธิบาย syntax ของภาษาอังกฤษ, FORTRAN

x1

x2

SUM

ใน BNF เราใช้สัญลักษณ์อภิภาษา (**metalinguistic symbols**) ต่อไปนี้ ซึ่งเป็นสัญลักษณ์ที่เขียนสั้นกว่า

::= หมายถึงถูกนิยามว่าเป็น ("is defined as")

{ } หมายถึง items ภายในวงเล็บนี้ ให้เลือกitems ได้ตั้งแต่ 0 อย่าง หรือมากกว่าขึ้นไป ("any sequence of 0 or more of the items enclosed.")

[ ] หมายถึง "either 0 or one concurrence of the items enclosed."

| หมายถึง หรือ ("or") ในความรู้สึกไม่รวม (in the exclusive sense)

Items ซึ่ง ตามกฎ BNF อาจจะเป็น ชื่อประเภทของ โทเค็น เช่น ไอเดนติไฟเออร์ หรือ สัญลักษณ์หนึ่งตัว ของ ชุดอักษรของภาษา

กฎ BNF จะมีชื่อ ประเภท โทเค็นหนึ่งชื่อ อยู่ทางซ้ายมือ แล้วตามด้วย ::= ตามด้วย ลำดับของชื่อประเภท และ/หรือ สัญลักษณ์ อาจจะคั่นด้วย | และอาจจะจัดกลุ่มโดย { }

(A BNF rule always has a single token class name on its left, followed by ::=, followed by a sequence of token class names and/or symbols, separated possibly by | and grouped possibly by { }.)

<u>ตัวอย่าง</u> กฎ BNF ข้างล่างนี้ นิยามอย่างถูกต้องว่า อะไร หมายถึง โทเค็นประเภท "letter"

letter ::=a|b|c|...|z|A|B|C|...|Z

ในภาษาอังกฤษ เราตีความบทนิยามนี้ว่า "ตัวอักษร หมายถึง "a" หรือ "b" หรือ "c" หรือ ... หรือ "z" หรือ "A" หรือ "B" หรือ "C" หรือ ... หรือ "Z" (ในที่นี้เราใช้คำย่อของบทนิยาม อย่างอิสระด้วย ... เพราะว่า ลำดับนั้น โดยนัยชัดเจน โดยไม่ต้องเขียนรายละเอียดทั้งหมด อย่างไรก็ตาม สิ่งนี้ ไม่ใช่ส่วนที่ถูกต้องของกลไก บทนิยาม BNF) ในทำนองเดียวกัน โทเค็น ประเภท "digit" นิยามดังนี้

CT 314 3-1

digit ::= **0|1|2|3|4|5|6|7|8|9**

จากบทนิยาม พื้นฐานเหล่านี้ เราสามารถสร้างกฎใหม่ ซึ่งวางอยู่บนกฎเดิม เช่นตัวอย่างต่อไปนี้

identifier ::= letter **{letter | digit}**

นั่นคือ identifier หมายถึง ตัวอักษร ตามด้วย การเกิดของ ตัวอักษร หรือ ตัวเลข ตั้งแต่ 0 ครั้ง หรือมากกว่าขึ้นไป การเข้าถึงอย่างเดียวกัน นำมาใช้นิยาม โทเค็น ประเภท "number" ขึ้นอยู่กับ ประเภท พื้นฐาน "natural", "integer" และ "digit" ดังนี้

natural ::= digit **{digit}**

integer ::= [+ |-] natural

number ::= integer | [+|-] . natural | integer . natural

ในที่นี้ โทเค็น ประเภท "natural" (จำนวนธรรมชาติ) รวม จำนวนเต็ม ไม่มีเครื่องหมายอยู่ด้วย เช่น 0, 7, 32 และ 4435

ประเภท "integer" หมายถึง จำนวนธรรมชาติ อาจจะมีเครื่องหมายกำกับได้ เช่น -7 และ +32

ประเภท "number" รวม integer และบวกด้วยส่วนของทศนิยม (decimal fraction) (เช่น .7 และ -.32) และเลขมีจุดทศนิยม (decimal number) (เช่น -7.32 และ 32.7)

ดังนั้น บทนิยาม จึงสมบูรณ์และรวบรัดทั้งคู่ และทำให้**ตัววิเคราะห์ศัพท์** (lexical analyser) สามารถรู้จำ โทเค็น เช่นนี้ได้อย่างมีประสิทธิภาพ

อีกหัวข้อหนึ่ง ซึ่งเผชิญหน้า (confront) นักออกแบบภาษา คือการปฏิบัติของ อักขระคั่น (delimiters) ในประมวลศัพท์ ซึ่งเป็นคำสมบูรณ์ (เช่น **begin**, **end** และ **program**) สิ่งนี้สำคัญเพราะว่า ภาษาชุดคำสั่ง ปกติ ยอมให้ ชื่อตัวแปร, ชื่อ procedure และเลเบลคำสั่ง นิยามเป็น ไอเดนติไฟเออร์ ได้ และเป็นไปได้ที่ นักเขียนโปรแกรม จะเลือกไอเดนติไฟเออร์ เช่น "begin" และ "end" เป็นชื่อตัวแปร ดังนั้น ทำอย่างไร จึงจะหลีกเลี่ยงความยุ่งยาก เมื่อตัวแปลชุดคำสั่ง พบคำ "begin" ในโปรแกรม

ผลเฉลยวิธีที่หนึ่ง คือ กำหนด (prescribe) ใน บทนิยามภาษาว่า โทเค็นเช่นนี้ทั้งหมดเป็น คำสงวน (reserved words) นั่นคือ แยกคำเหล่านี้ ออกจาก โทเค็น ประเภท ไอเดนติไฟเออร์ ซึ่ง โปรแกรมสามารถนิยามสำหรับการใช้ของตนเอง การเข้าถึงวิธีนี้ ยกตัวอย่างเช่นในภาษา Pascal, Ada และ COBOL รายการคำสงวนของภาษา Pascal มีน้อยมาก (ดูบทที่ 2) เมื่อเปรียบเทียบกับ คำสงวนในภาษา Ada (ดูบทที่ 13) และ ภาษา COBOL (ดูบทที่ 4) ข้อดีของการเข้าถึง คำสงวน คือ ตัวแปลชุดคำสั่ง วิเคราะห์ศัพท์ได้ง่าย ส่วนข้อไม่ดีคือ รายการคำสงวนที่มี จำนวนมาก เป็น**ภาระติดพัน** (encumbers) กับนักเขียนโปรแกรม เพราะว่า จะต้องเอาออก จากคำทั่ว ๆ ไป ที่ปกติจะเลือกเป็น ไอเดนติไฟเออร์ ในโปรแกรม ตัวอย่างเช่น SUM เป็นคำสงวน ในภาษา COBOL

ผลเฉลยวิธีที่สอง คือ กำหนดว่าคำเหล่านี้เป็น คำหลัก (keywords) โดยมีเครื่องหมายชัดเจนกำกับ ในตัวโปรแกรม ด้วย อักขระคั่นพิเศษ หรือ พิมพ์พิเศษ การเข้าถึงวิธีนี้ ได้นำไปใช้ ในการทำให้เกิดผล ในภาษา ALGOL และ BASIC บางเวอร์ชัน ตัวอย่างเช่น การทำให้เกิดผล วิธีหนึ่ง คำหลักแต่ละตัว ซึ่งอยู่ในตัวโปรแกรม ให้คีย์ เครื่องหมายคำพูด (') นำหน้า โปรแกรม Pascal ข้างต้น จะปรากฏข้างล่าง ถ้าเราใช้ข้อตกลงนี้

```
'PROGRAM P;
   'VAR   X, Y : 'INTEGER;
   'BEGIN
      READ(X);
      Y := X + 25;
      WRITE(Y)
   'END.
```

ขณะนี้ โทเค็น PROGRAM, BEGIN, END และอื่น ๆ เป็นอิสระสำหรับการนำไปใช้เป็นชื่อตัวแปร ภายใน โปรแกรม อย่างไรก็ตาม ข้อตกลงนี้ ทำให้การเขียนโปรแกรม น่าเบื่อมากขึ้น ในขณะที่ มันทำให้ กระบวนการ การวิเคราะห์ศัพท์ของ ตัวแปลชุดคำสั่ง ง่าย

ผลเฉลยวิธีที่สาม คือยอมให้ โทเค็นเหล่านี้ ใช้เป็น ไอเดนติไฟเออร์ ในโปรแกรมได้ และตำแหน่งที่อยู่ของมัน แยกความแตกต่างระหว่าง การใช้สองชนิด ของ ไอเดนติไฟเออร์ บน เนื้อหา ซึ่งมันปรากฏ ในโปรแกรม ตัวอย่างเช่น บทนิยามของภาษา PL/1 เข้าถึงโดยวิธีนี้ ดังตัวอย่างข้างล่างนี้

```
IF IF = 1 THEN IF = 0;
```

DO DO = IF TO THEN;

ตัวอย่างแรก ตัวแปร "IF" ถูกทดสอบ ถ้าเงื่อนไขเป็นจริง กำหนดค่าให้ใหม่ เป็น "0" เป็นการใช้คำสั่ง IF

ตัวอย่างที่สอง ตัวแปรลูป "DO" ถูกกำหนดค่าเริ่มต้น ให้เป็นค่าของ ตัวแปร "IF" และลูปจะถูกทำซ้ำ จนกระทั่งค่าของมันถึงค่าของตัวแปร "THEN" เป็นการใช้คำสั่ง DO

แน่นอน นักเขียนโปรแกรม ซึ่งเขียนคำสั่งเหล่านี้ ไม่ได้รับอะไรเลย นอกจากนี้ ภาระที่สำคัญ วางอยู่บนความพยายามของ ตัวแปลชุดคำสั่งที่จะสนับสนุน "อิสระของนิพจน์" (freedom of expression) และรวมไปถึงความไม่ชัดเจนที่ว่า ผลลัพธ์คุ้มค่ากับความพยายามนั้น

2.3 โครงสร้างวากยสัมพันธ์ (Syntactic Structure)

บทนิยามวากยสัมพันธ์ที่สมบูรณ์ ของภาษาชุดคำสั่ง คือ ดูที่การนิยามอย่างพร้อมมูล เซต ของสายอักขระทั้งหมดของสัญลักษณ์ ซึ่งประกอบเป็นโปรแกรมถูกต้อง จากการมองทางด้านไวยากรณ์ (The complete syntactic definition of a programming language seeks to fully define the set of all strings of symbols that form correct programs, from a grammatical point of view.) บางภาษา เช่น FORTRAN, COBOL, BASIC และ SNOBOL นิยามวากยสัมพันธ์ บน หนึ่งบรรทัด โปรแกรม (a program line) และจบแต่ละบรรทัด โดย อักขระคั่นซ่อน (a hidden delimiter) ใน ตัวของโปรแกรม

ตัวอย่างเช่น ในหนึ่งคำสั่งของภาษา FORTRAN ปกติจะเริ่มต้นบนบรรทัดใหม่ โดย ไม่จำเป็นต้อง มี ตัวคั่นคำสั่งชัดแจ้ง (an explicit statement separator) ในชุดอักขระ นอกจากนี้แล้ว เมื่อ หนึ่งคำสั่ง เขียนเกิน จนถึงบรรทัดที่ สอง จะต้อง ทำข้อสังเกต (ไม่ใช่เครื่องหมาย blank ในตำแหน่งที่ 6) เพื่อแสดงให้เห็นว่า เป็นการต่อเนื่องกัน

ภาษา COBOL มีรูปแบบอิสระมากกว่า เพราะว่า จุด (.) เป็นอักขระคั่นประโยค (sentence delimiter) และประโยคต่าง ๆ สามารถต่อได้อย่างอิสระ จากบรรทัดหนึ่ง ไปยังบรรทัดถัดไป โดยไม่ต้องมี เครื่องหมายชัดแจ้ง แสดงการต่อเนื่องกัน แต่ มีข้อยกเว้น สำหรับกฎนี้ เช่นการต่อ สายอักขระ (literal string) และ การสำรองพิเศษ (special reservation) ที่ ตำแหน่ง 8-11 ตอนหน้าของแต่ละบรรทัด สำหรับชื่อพารากราฟ

ภาษาสมัยใหม่ ส่วนใหญ่ เช่น Pascal, Ada และ C เป็นภาษาที่มีรูปแบบอิสระเชิงวากยสัมพันธ์ (are syntactically free-form) โครงสร้างของ โปรแกรมเป็นอิสระจาก

บรรทัดและการกำหนดขอบ (**tab boundaries**) คำสั่งถูกคั่น ด้วย โทเค็น ชัดแจ้ง (ปกติใช้ ;) และเลเบล ถูกแยกโดยเครื่องหมายวรรคตอน (ปกติใช้ :) รูปแบบ BNF นำมาใช้นิยามวากยสัมพันธ์ของภาษาเหล่านี้ เป็นส่วนใหญ่ ดังตัวอย่างข้างล่างนี้ เป็นรูปแบบหนึ่ง แสดงการอธิบาย วากยสัมพันธ์ ของ การสร้าง Pascal : "statement" และ "expression" และประเภทของ โทเค็นอื่น ๆ ที่เกี่ยวข้อง

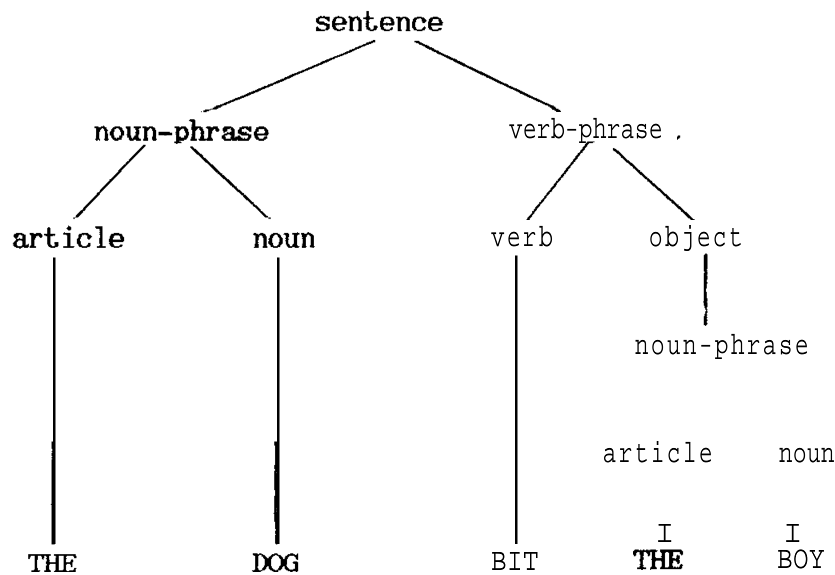
```
statement ::= unlabeled-stmt I
          label : unlabeled-stmt
unlabeled-stmt ::= simple statement I
          structured-stmt
simple stmt ::= assignment-smt I
          procedure-stmt I
          goto-stmt
structured-stmt ::= compound-stmt I
          conditional-stmt I
          repetitive-stmt I
          with-stmt
compound-stmt ::= begin statement f; statement) end
conditional-stmt ::= if-stmt I case-stmt
if-stmt : : = if expression then statement I
            if expression then statement else statement
assignment-stmt ::= identifier := expression
expression ::= simple exp I simple exp relop simple exp
relop ::= = I <> I < I <= I >= I > I in
simple exp ::= [+ I -1 term I simple exp addop term
addop ::= t I - I or
term ::= factor I term mulop factor
mulop ::= * I / I div I nod I and
factor ::= identifier I number I (expression) I
           function designator I set I not factor
```

กฎข้างต้นนี้ เป็น เซตย่อยของ กฎ BNF ทั้งหมด 108ข้อ ซึ่งรวมเข้าด้วยกัน ประกอบเป็น วากยสัมพันธ์ของ Pascal

เซตของกฎ BNF นิยาม วากยสัมพันธ์ ของภาษาชุดคำสั่งในสาระดังนี้ สายอักขระของโทเค็น (จากชุดอักขระของภาษา) หมายถึง โปรแกรมถูกต้องเชิงวากยสัมพนธ์ (หรือ ประโยค หรือ นิพจน์ เป็นต้น) ในภาษา ถ้ามันสามารถถูกได้มา (can be derived) โดยใช้กฎ BNF ซึ่งเหมาะสมกับประเภท (โปรแกรม คำสั่ง นิพจน์) ในคำถาม

(A string of tokens (from the language character's set) is a syntactically correct program (or statement or expression, etc.) in the language if it can be "derived" using the BNF rules that are appropriate to the class (program, statement, expression) in the question.)

การได้มา (derivation) ของ โปรแกรม, ประโยค หรือ นิพจน์ โดยใช้กฎ BNF คล้ายกับ การกระจายคำ (parsing) ในประโยคภาษาอังกฤษ เช่น ถ้าเราจะหา คำตอบของคำถามที่ว่า สายอักขระของคำต่าง ๆ ที่กำหนดให้ เป็นประโยค หรือไม่? สิ่งแรก หาประธาน (subject) และเพรดิเคต (predicate) หรือ กริยาและกรรม วากยสัมพันธ์ของภาษาอังกฤษ บอกว่า subject คือ noun phrase และ "predicate" คือ คำกริยา และอาจจะตามด้วยกรรม ดังนั้น เราสามารถ กระจาย หรือวาดรูปต้นไม้ ประโยคภาษาอังกฤษ จะเป็นดังนี้



ในที่นี้ ภายใต้กฎไวยากรณ์ สามารถแสดงให้เห็นโดย BNF ดังนี้

sentence ::= noun-phrase verb-phrase

noun-phrase ::= noun I article noun

verb-phrase ::= verb I verb object

object ::= noun-phrase

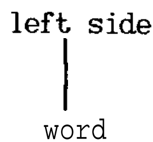
noun ::= BOY I DOG I GIRL

article ::= A I AN I THE

verb ::= BIT I SAW I WROTE

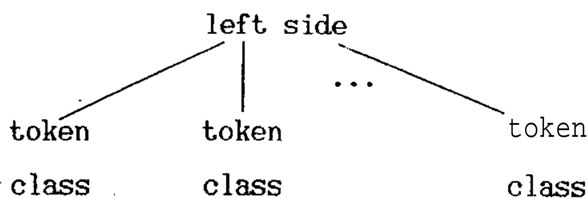
กำหนด ประโยค "The dog bit the boy" เราพยายามที่จะสร้าง **ต้นไม้วิภัช หรือ ต้นไม้วิเคราะห์กระจาย** (a parse tree) โดยใช้กฎ ต่อไปนี้

1. สำหรับแต่ละคำในประโยค ซึ่งเกิดขึ้น คือ ทางด้านขวามือของกฎ สร้างส่วนหนึ่งของต้นไม้วิภัช ดังนี้



ในที่นี้ "left side" หมายถึง ทางซ้ายมือของกฎที่สมนัยกัน

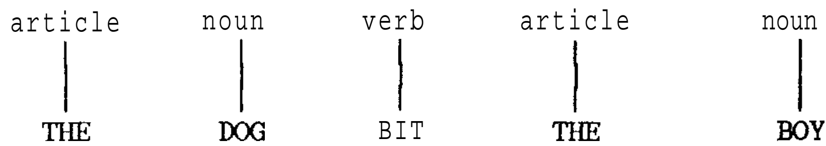
2. ทำซ้ำขั้นตอนที่ 1 สำหรับแต่ละ (กลุ่มของ) ประเภท โทเค็น ซึ่งเกิดขึ้น ระหว่างรากของ ต้นไม้ที่จะสร้าง เป็นส่วน ๆ และยังคงรักษาอันดับจากซ้ายไปขวา



ในที่นี้ ผลลัพธ์ เป็นต้นไม้ย่อย ซึ่งรากอยู่ทางซ้ายมือของ กฎที่ใช้ และกิ่งต่าง ๆ ของมัน นำไปสู่ประเภทโทเค็น ของต้นไม้ย่อย ตามลำดับ

กรรมวิธีนี้ จะให้ผลลัพธ์ เป็น (ต้นไม้) การกระจายสมบูรณ์ สำหรับ สายอักขระ (string) ถ้ามันเป็นประโยคถูกต้อง ตามวากยสัมพันธ์ หรือ จะถูกปิดกั้น จากความสำเร็จ ซึ่งเป็นเป้าหมาย เพราะว่า สายอักขระนั้น ไม่ถูกต้อง ในความรู้สึกนี้

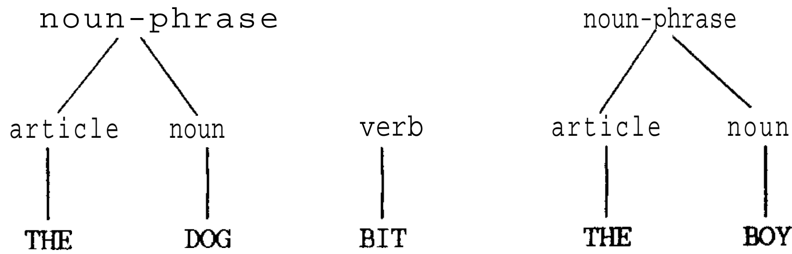
ตัวอย่าง สมมติว่า ต้องการกระจาย สายอักขระ "THE DOG BIT THE BOY" โดยใช้ ไวยากรณ์ที่กำหนดให้ จากขั้นตอนที่ 1 เรามีส่วนต่าง ๆ ดังนี้



จากขั้นที่ 2 เราจับกลุ่ม ราก "article **noun**" เพราะว่า มันเกิด ทางขวามือ ของกฎ

noun-phrase **::=** noun I article noun

และทำให้ละเอียดยิ่งขึ้น



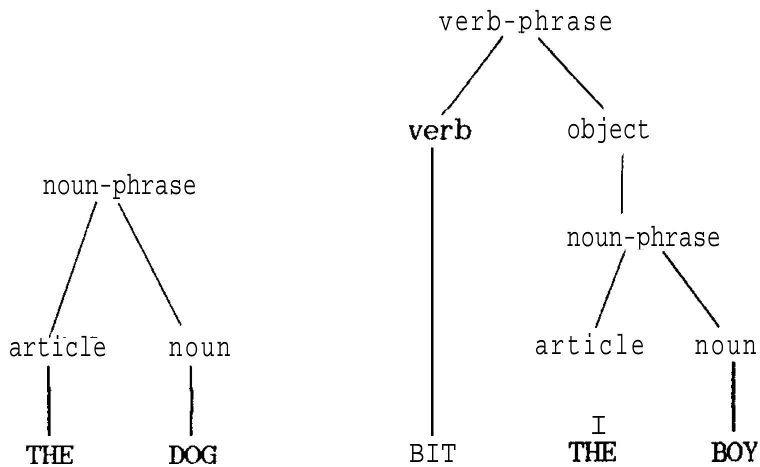
จากขั้นที่ 2 อีกครั้งหนึ่ง เราใช้กฎ

object ::= noun-phrase

และจากกฎ

verb-phrase ::= verb object

ทำให้ละเอียดยิ่งขึ้น



ทำการกระจายให้สมบูรณ์ โดยใช้กฎ

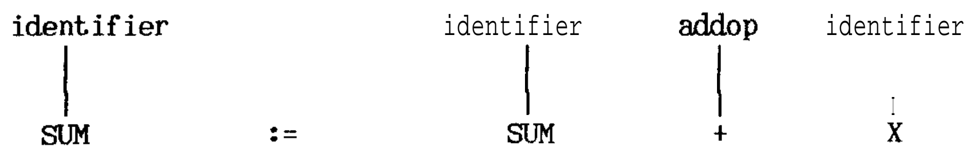
```
sentence  ::=  noun-phrase  verb-phrase
```

ซึ่งจะให้ผลลัพธ์เป็น โครงสร้างต้นไม้ ดังแสดงไว้ตอนต้นของการอภิปรายนี้

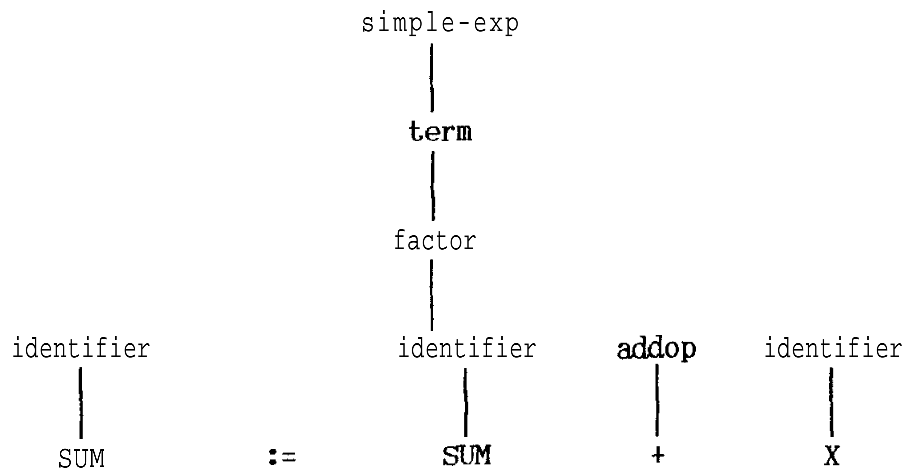
กลับมาที่กฎ BNF สำหรับภาษา Pascal เราสามารถใช้ กฎเหล่านี้ กระจาย โครงสร้างต่าง ๆ ของโปรแกรม Pascal และค้นหา ข้อผิดพลาดวากยสัมพันธ์ (syntax error) ใด ๆ ในวิธีนั้น

ตัวอย่างเช่น ต้องการกระจาย สายอักขระ SUM := SUM + X เป็น assignment-stmt จากนั้นเราต้องหา ชุดของกฎที่ใช้ สร้างต้นไม้วิภัช กับโทเค็น SUM, :=, SUM, + และ X หมายถึงใบของมัน และ assignment-stmt หมายถึงรากของมัน

<u>ขั้นที่ 1</u> จะเป็นดังนี้



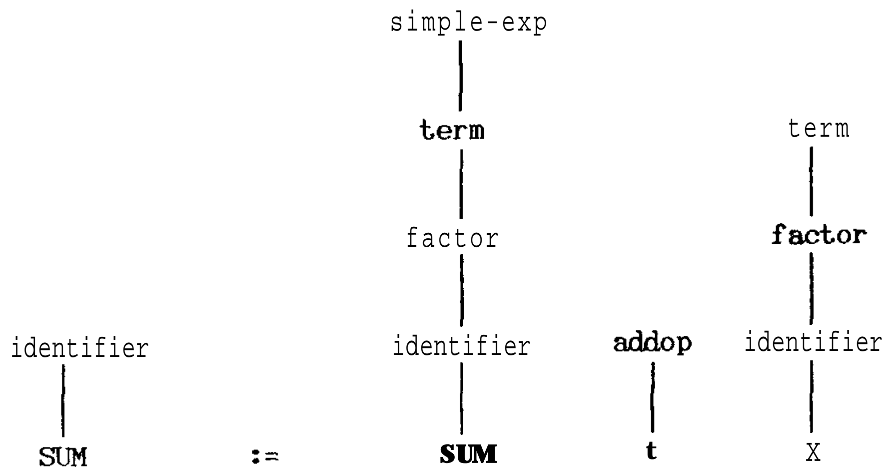
หลังจากทำขั้นที่ 2 อีกสามครั้ง จะได้ดังนี้



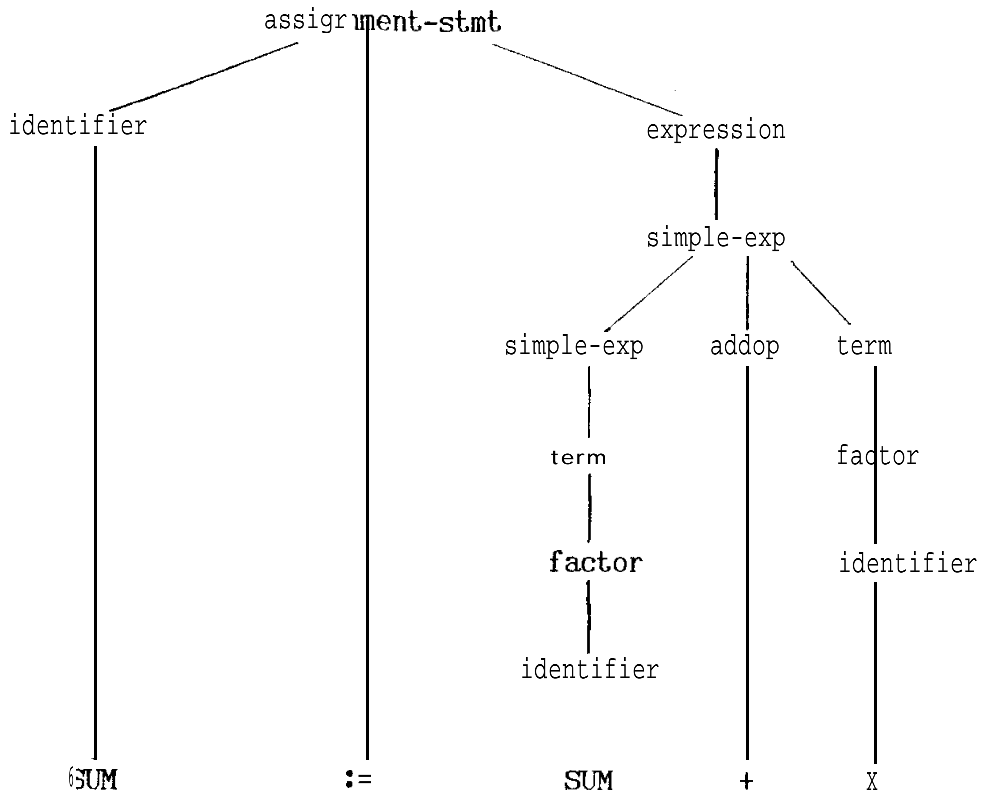
กฎต่าง ๆ ที่ใช้ คือ

```
factor ::= identifier
term   ::= factor
simple-exp ::= term
```

ถ้าเราทำซ้ำ กรรมวิธีนี้ โดยใช้เฉพาะ สองกฎแรกเท่านั้น สามารถทำสำเร็จ และเปลี่ยนแปลง เป็นดังนี้

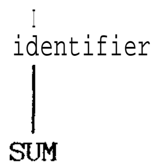


ณ จุดนี้ เราใช้กฎต่าง ๆ สำหรับ **simple-exp, expression** และ **assignment-stmt** ในลำดับนั้น เพื่อทำให้ การกระจายสมบูรณ์ ดังนี้



ผู้อ่านอาจจะสังเกตได้ว่า ในการมาถึงการกระจายนี้ เราทำเฉพาะ การเลือก "ถูกต้อง" ทีละขั้นตอน นั่นคือ การเลือกต่าง ๆ เพื่อที่สุดท้ายนำไปสู่ ต้นไม้สมบูรณ์ ของ **"assignment-stmt"** ที่รากของมัน

<u>ตัวอย่าง</u> เราอาจทำทางเดินผิด



สำหรับการเกิดของ SUM ทางซ้ายมือของสัญลักษณ์ กำหนดค่า (:=) แต่สิ่งนี้ ปิดกั้นความสำเร็จ ของการกระจายที่สมบูรณ์ สำหรับ ประโยคทั้งหมด

วิธีการกระจาย ซึ่งเราได้อธิบายอย่างคร่าวๆนี้ เป็นที่รู้จักกันทั่วไปว่าเป็น**การกระจายแบบล่างขึ้นบน (bottom-up parsing)** และเป็นยุทธวิธีหนึ่งในยุทธวิธีต่างๆ ซึ่งใช้โดย ตัวแปล

ชุดคำสั่ง สำหรับรายละเอียด ของ การกระจายชั้นสูง ปกติจะอยู่ในวิชา ตัวแปลชุดคำสั่ง (compiler) และปรากฏอยู่ในตำราเกี่ยวกับ compilers เป็นส่วนใหญ่

2.4 การอธิบายวากยสัมพันธ์ของ COBOL
(COBOL syntax description)

อภิภาษา ใช้อธิบายวากยสัมพันธ์ของ COBOL มีข้อจำกัดของ ขอบเขต มากกว่า BNF (The metalanguage used to describe the COBOL syntax is more limited in scope than BNF.) คือใช้ เฉพาะวากยสัมพันธ์ ของแต่ละคำสั่ง มากกว่า โครงสร้าง ของโปรแกรมทั้งหมด นอกจากนี้แล้ว มันถูกออกแบบ และนำมาใช้ ในวิธีที่จะเป็นเครื่องมือ สำหรับคู่มือผู้ใช้ (reference manuals) อธิบาย วากยสัมพันธ์ของคำสั่ง ให้กับนักเขียนโปรแกรม เช่นเดียวกับ นักเขียนตัวแปลชุดคำสั่ง ตรงกันข้ามกับ BNF ซึ่งตั้งใจแต่แรกแล้วว่าจะให้ใช้ได้โดย นักออกแบบภาษา และนักทำภาษาให้เกิดผล ประโยชน์ในทางการเรียนการสอนภาษาสำหรับ นักเขียนโปรแกรม ปรากฏว่า BNF มีข้อจำกัด มากกว่า

ตัวอย่างของภาษาที่ใช้อธิบายวากยสัมพันธ์ ของ COBOL อยู่ในบทที่ 4 โดยพื้นฐานข้อตกลงของมัน สรุปดังนี้

{ } ปิด รายการของ items ที่ซึ่ง item หนึ่งตัวจะต้องถูกเลือก

[ ] ปิด ลำดับของ items ซึ่งอาจจะละเว้นได้

<u>ตัวอย่าง</u> วากยสัมพันธ์ สำหรับคำสั่ง SELECT

```
SELECT file-name ASSIGN TO device-name
       [REVERSE integer AREAS]
       [ORGANIZATION IS {SEQUENTIAL}]
       [                {INDEXED   }]
       [ACCESS MODE IS {SEQUENTIAL}]
       [               {RANDOM    }]
       [RECORD KEY IS data-name]
```

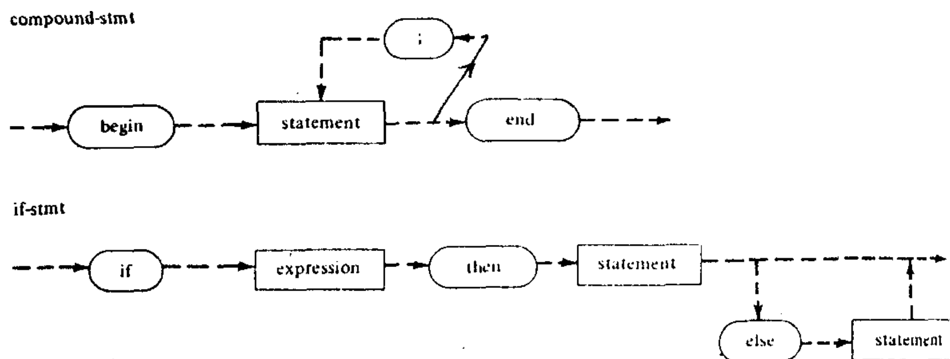
จากรูปแบบข้างต้น จะเห็นได้ทันทีว่า อะไรคือ options และอะไรคือ requirements ของคำสั่ง SELECT โดยดูจาก items ในรายการ และอักขระคั่นที่ปิด (วงเล็บปีกกาหรือวงเล็บใหญ่)

ตัวอย่าง สามแบบ จากหลาย ๆ ทางเลือก ข้างล่างนี้

```
SELECT MYFILE ASSIGN TO SYSIN
SELECT MYFILE ASSIGN TO SYSIN
       REVERSE 4 AREARS
SELECT MYFILE ASSIGN TO SYSIN
       ORGANIZATION IS INDEXED
       ACCESS MODE IS RANDOM
       RECORD KEY IS SOCIAL-SECURITY-NO
```

2.5 แผนภาพวากยสัมพันธ์ (Syntax Diagrams)

ทางเลือกต่างๆ และหลายรูปแบบ (variants) ของ BNF ซึ่งใช้อธิบายวากยสัมพันธ์ของภาษาต่าง ๆ ซึ่งแบบหนึ่ง ได้อภิปรายแล้ว ในหัวข้อ 2.3 น่าจะเป็น แบบที่นิยมมากที่สุด ทางเลือกอีกอย่างหนึ่ง นอกจาก BNF และที่ใช้ใน COBOL คือ **แผนภาพวากยสัมพันธ์** (syntax diagram) ซึ่งนิยมมากใน การอธิบาย Pascal อย่างเป็นทางการ ตัวอย่างข้างล่างนี้ เป็นแผนภาพวากยสัมพันธ์ ของ "compound-stmt" และ "if-stmt" ซึ่งมีความหมายอย่างเดียวกับ การอธิบาย BNF ในหัวข้อ 2.3

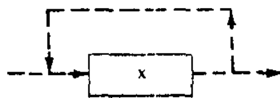


ในที่นี้ โครงสร้างนี้คือ กราฟมีทิศทาง มีทางเข้าหนึ่งทางด้านซ้ายมือ และทางออกหนึ่งทางด้านขวามือ สมาชิกซึ่งอยู่ในวงกลม (เช่น **begin** และ **end** ข้างต้น) หมายถึง โทเค็น (tokens) ในภาษา ขณะที่สมาชิกซึ่งอยู่ในสี่เหลี่ยมผืนผ้า (เช่น **"statement"** และ "expression" เป็นประเภทของ โทเค็น (token classes)

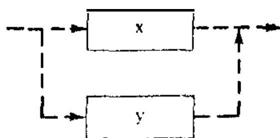
แผนภาพวากยสัมพันธ์ เป็นการอธิบาย กฎรูปแบบต่าง ๆ ของภาษา ด้วยรูปภาพ ลูป หมายถึง การทำซ้ำของการสร้าง และกิ่งหมายถึงทางเลือก

(The syntax diagram thus gives a pictorial description of the language's formation rules; loops denote repetition of constructs, and branches denote alternatives.)

ตัวอย่างเช่น รูปข้างล่างนี้ หมายถึง x{x} in BNF



รูปข้างล่างนี้ หมายถึง x|y ใน BNF



ดังนั้นแผนภาพวากยสัมพันธ์ จึงไม่ได้เป็นอุปกรณ์การอธิบายที่มี **powerful** มากกว่า BNF มันเป็นทางเลือกหนึ่ง ซึ่งหลายคนชอบเพราะ คุณภาพในการอธิบาย และความง่ายของมัน (Thus, the syntax diagram is no more powerful as description device than BNF; it is an alternative which many prefer because of its descriptive quality and its simplicity.)

2.6 หัวข้ออื่น ๆ ในวากยสัมพันธ์ (Other issues in syntax)

หัวข้อสำคัญ ในการอธิบายวากยสัมพันธ์ คือ คำถามของ ความพอเพียง (adequacy) นั่นคือ BNF ในการอธิบายข้อกำหนดเชิงวากยสัมพันธ์ ทั้งหมดของภาษาชุดคำสั่ง ทำให้สมบูรณ์ได้อย่างไร? โชคไม่ดี คือ ไม่ครบถ้วนทั้งหมด (not entirely) การจำกัดที่มากเกินไปของ

BNF **เรียกว่า ไวยากรณ์ไม่พึ่งบริบท** (context-free grammar) นั่นคือ ถ้าภาษาหนึ่งมี **ข้อกำหนดวากยสัมพันธ์**บางอย่างซึ่งไวต่อเนื้อความ ซึ่งเป็น ตัวสร้างเขียนโปรแกรม แล้วข้อ-**กำหนดนั้น** ไม่สามารถแสดงออกด้วย **BNF** (that is, if a **langauge** has some syntactic requirement that is sensitive to the context in which a construct is written in the program, then that requirement cannot be **expressed in BNF.**)

ตัวอย่าง ข้อกำหนด ซึ่งไวต่อเนื้อหา ภายในภาษาร่วมสมัย เช่น ข้อกำหนดที่ว่า ตัวแปรทั้งหมด ซึ่งใช้ใน โปรแกรมต้องมีการประกาศ ข้อกำหนดนี้ มีอยู่ในภาษา **Pascal, COBOL** และ **Ada** แต่ไม่มีใน ภาษา **FORTRAN, PL/1** และ **LISP**

มีอยู่สองวิธีในการจัดการ ข้อจำกัดเชิงวากยสัมพันธ์นี้ ในบทนิยามของภาษา คือ วิธีที่หนึ่ง อธิบายด้วยภาษาอังกฤษ หรืออีกวิธีหนึ่ง เลือกรูปแบบของวากยสัมพันธ์ ซึ่งดีกว่า (**more powerful**) **BNF** รูปแบบเช่นนี้มีอยู่แล้ว รูปแบบหนึ่งเรียกว่า ไวยากรณ์พึ่งบริบท (**context-sensitive grammar**) แต่โดยทั่วไปแล้วมันซับซ้อนมาก และในทางปฏิบัติ ไม่สะดวก ที่จะเก็บไว้ภายในตัวแปลชุดคำสั่ง

วิธี **Ad hoc** โดยทั่วไปเป็นไปอย่างตรงไปตรงมา และดีกว่า เป็นการเข้าถึงเพื่อบังคับ ข้อกำหนดวากยสัมพันธ์ ซึ่งไวต่อเนื้อหา จบด้วย ตารางสัญลักษณ์ (**symbol table**) ซึ่งเป็นโครงสร้างข้อมูลที่สำคัญมากที่สุด อย่างหนึ่ง จัดการโดยตัวแปลชุดคำสั่ง ตารางสัญลักษณ์ หมายถึง รายการของสัญลักษณ์ทั้งหมดที่ใช้ในโปรแกรม เช่น ชื่อตัวแปร, เลเบลคำสั่ง ชื่อโปร-ซีเดอร์ และ อื่น ๆ รวมทั้งลักษณะเฉพาะของมันด้วย (**The symbol table is a list of all symbols used in program - variable names, statement labels,** procedure names, and so forth - together with their attributes.)

ตัวอย่าง ตารางสัญลักษณ์ขวามือ เป็นผลลัพธ์จากโปรแกรม Pascal ทางซ้ายมือ

```
program P;
  var  x : integer;
   begin
     read(x);
     y := x + 2.5;
     write(y)
   end.
```

Symbol	Attributes	Declared?
P	proc-name	Y
X	integer-var	Y
y		N

ในที่นี้ตัวแปลชุดคำสั่ง มีการระบุ สัญลักษณ์ (y) ซึ่งไม่ปรากฏ ในการประกาศ ที่ตอนบนของโปรแกรม

สิ่งที่ข้อกำหนดคือ ตัวแปรทั้งหมด ต้องมีการประกาศ ทำไมจึงต้องทำการประกาศ ที่ตอนบน (at the top) ของโปรแกรม มากกว่า ตอนล่าง (at the bottom) หรือที่อื่น ๆ การพบสัญลักษณ์ y ในโปรแกรมข้างต้น และเห็นว่า y ยังไม่ได้มีการประกาศ ตัวแปลชุดคำสั่งสามารถยุติได้ว่า y จะไม่มีการประกาศที่ใดๆ อีกในโปรแกรม ดังนั้น สามารถทำเครื่องหมายที่ข้อผิดพลาดวากยสัมพันธ์นี้ได้ทันที การหละหลวมของ ข้อกำหนดนี้ จะย้ายภาระไปให้ตัวแปลชุดคำสั่งในรูปฟอร์มของ การส่งผ่าน เหนือโปรแกรม (additional pass over the program text) เพื่อที่ว่า การประกาศทั้งหมด สามารถประมวลผล ก่อนการอ้างถึงใด ๆ ที่เป็นไปได้ ที่จะพบตัวแปรที่ไม่มีการประกาศ

ตารางวากยสัมพันธ์ เป็นอุปกรณ์ ad hoc ที่พอเพียง สำหรับการบังคับข้อกำหนดพึ่งบริบท ซึ่งตัวแปรทั้งหมดในการประกาศ หนึ่งครั้ง ต้องมีเพียงชื่อเดียวเท่านั้น (The syntax table is also an adequate ad hoc device for enforcing the context-sensitive requirement that all variables in a declaration have mutually unique names.1

<u>ตัวอย่าง</u> การประกาศวากยสัมพันธ์ ทำให้เกิดข้อผิดพลาดวากยสัมพันธ์ (syntax error)

```
var   x : integer;
      x : real;
```

การค้นตารางสัญลักษณ์ ในการประกาศ ตัวแปรแต่ละตัว จะตรวจพบ ข้อผิดพลาดชนิดนี้

การประกาศเพียงหนึ่งครั้งเท่านั้น เป็นประเด็นซับซ้อนมากขึ้น เมื่อคำถามของโครงสร้างบล็อก และ scope ได้ถูกแนะนำ นั่นคือ ชื่อตัวแปรอาจมีการประกาศหลายครั้ง ทำให้ scope ของแต่ละกรณี (each instance) ของชื่อ ไม่คาบเกี่ยวกัน กับกรณีอื่น (any other instance)

ตัวอย่าง ภาษา Pascal

```
program P;
   var x, y : integer;
   procedure Q;
      var x : real:
```

```
    begin
        :
        :
    end Eof Q};
begin
    :
    :
end {of P}.
```

ในที่นี้ scope ของ **integer** x รวมคำสั่งต่างๆ ของบล็อก P แต่ไม่รวมคำสั่งของบล็อก Q ในขณะที่ scope ของ **real** x รวมคำสั่งของ Q แต่ไม่รวมคำสั่งของ P ในทางตรงกันข้าม scope ของ y รวมทั้ง P และ Q หัวข้อเหล่านี้ ทั้งหมดเป็นวากยสัมพันธ์ที่สำคัญ และจัดการโดยวิธี ad hoc เกี่ยวข้องกับ ตารางสัญลักษณ์ มากกว่า ภายในรูปแบบวากยสัมพันธ์ของมันเอง

หัวข้อสำคัญอีกหนึ่งเรื่องในวากยสัมพันธ์ คือ ปัญหาของการขจัดความกำกวมเชิงวากยสัมพันธ์ (Another important issue in syntax is the problem of eliminating syntactic ambiguity.) โดยทั่วไป ภาษาจะมีความกำกวมเชิงวากยสัมพันธ์ ถ้ามันประกอบด้วยการสร้างซึ่งสามารถกระจาย ได้ตั้งแต่สองวิธีขึ้นไป (a language is syntactically ambiguous if it contains a construct which can be parsed in two or more different ways.)

ตัวอย่าง คำสั่งภาษาFORTRAN

```
      DO 10 I = 1.5
         :
   10 CONTINUE
```

ถ้าในโปรแกรม เครื่องหมาย space ไม่สำคัญ คำสั่ง DO นี้ อาจถูกตีความหมายอีกหนึ่งทางเลือก คือ คำสั่งกำหนดค่า ดังนี้

```
      DO10I = 1.5
```

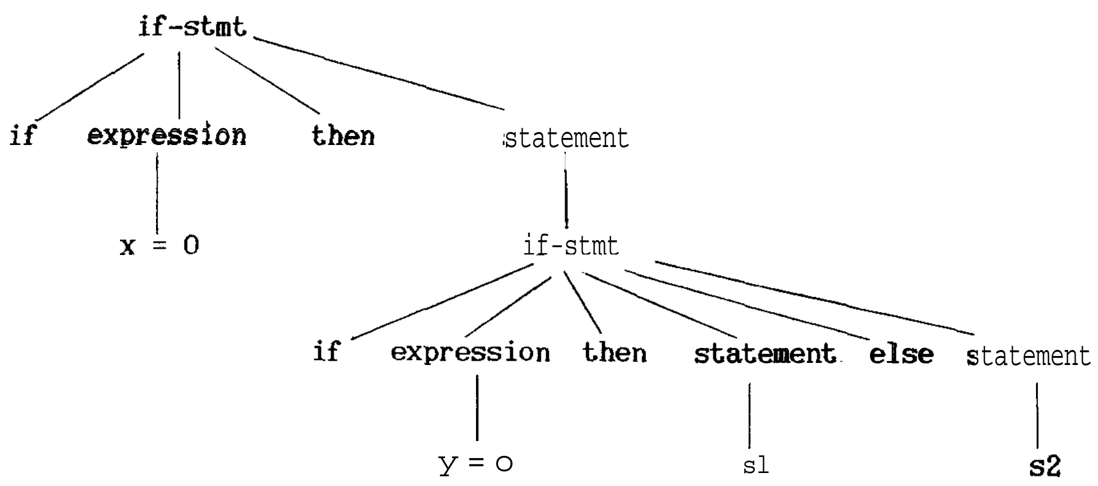
ตัวแปรชื่อ DO10I มีค่าเท่ากับ 1.5

วากยสัมพันธ์ BNF สำหรับ "if-stmt" ที่กำหนดให้ในหัวข้อก่อนหน้านี้ ทำให้เห็นความกำกวม ได้เป็นอย่างดี

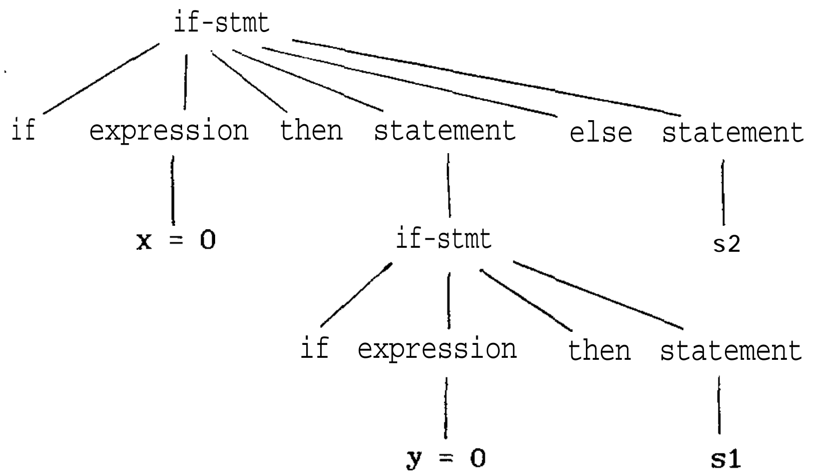
<u>**ตัวอย่าง**</u> ให้ s1 และ s2 เป็นคำสั่งใด ๆ จงพิจารณาการกระจายถูกต้อง สองชุด ข้างล่างนี้

if x = 0 then if y = 0 then s1 else s2

รูปที่ 1



รูปที่ 2



ตัวอย่างของความกำกวมนี้ เป็นที่รู้จักกันว่า <u>ปัญหาการแขวน else</u> ("dangling else problem") และถูกจัดการในภาษาต่าง ๆ ด้วยความหมายแตกต่างกัน ในกรณีของ Pascal มันจัดการ โดย กฎ ad hoc ที่ว่า กำหนดอย่างอัตโนมัติ ให้ <u>else</u> แต่ละอัน กับ <u>if</u>

ใกล้ที่สุด (อันในสุด) ซึ่งอยู่ข้างหน้ามัน ดังนั้น การกระจายชุดแรกข้างต้น (รูปที่ 1) จึงเป็นชุดถูกต้อง ซึ่งการตีความเช่นนี้ เหมือนกับภาษา PL/1

ภาษา ALGOL วากยสัมพันธ์ BNF ไม่อนุญาตให้ คำสั่ง **if** ตามหลังคำสั่ง **if** อีกชุดหนึ่งทันที ถ้าจำเป็นต้องทำแล้ว **if** ชุดที่สอง ต้องอยู่ใน compound statement โดยการปิดประตู บังคับโดยตัวคั่น **begin** และ **end** เพื่อชี้ขาดการติดกันของ dangling else clause ดังนั้น ใน ALGOL โครงสร้างการกระจายสองชุดข้างต้น จึงเขียนด้วยคำสั่ง **if** ให้เลือก สองชุด ตามลำดับดังนี้

```
if x = 0 then begin if y = 0 tben s1 else s2 end
if x = 0 then begin if y = 0 then s1 end else s2
```

ภาษา Ada มียุทธวิธีที่แตกต่างออกไปเล็กน้อยคือ คำสั่ง <u>if</u> ทั้งหมด ต้องปิดท้ายด้วย สัญลักษณ์ **endif** ไม่ว่ามันจะมีส่วน else หรือไม่มีก็ตาม ดังนั้น ปัญหา การแขวน else จึงไม่เกิดขึ้นใน Ada สิ่งนี้ทำให้สำเร็จได้ โดยการเขียนเพิ่มของนักเขียนโปรแกรม การกระจายสองชุด ข้างต้น จึงเขียนเป็นคำสั่ง Ada ตามลำดับดังนี้

```
if x = 0 then if y = 0 then s1 else s2 endif endif
if x = 0 then if y = 0 then s1 endif else s2 endif
```

2.7 <u>วากยสัมพันธ์ และการเขียนโปรแกรม</u>

(Syntax and Programming)

เกณฑ์ 9 ข้อสำหรับประเมินค่าภาษา เกณฑ์บางข้อเกี่ยวข้องโดยตรง กับวากยสัมพันธ์ คุณภาพของการอธิบายวากยสัมพันธ์ ของภาษา มีอิทธิพลโดยตรงต่อแนวโน้ม สำหรับนักเขียนโปรแกรม ที่จะทำให้เขียนโปรแกรมง่ายขึ้น และ ยอมรับ ข้อผิดพลาดเชิงแนวคิด วากยสัมพันธ์ส่วนใหญ่ ที่เป็นสาเหตุ ให้เกิดข้อผิดพลาดในการเขียนโปรแกรม มีดังนี้ (Some of the most common syntactic causes of programming errors are the fllowing :)

A. <u>ลืมเขียนหรือ ใช้เครื่องหมายวรรคตอนผิด หรือตัวคั่นผิด</u>

(Missing or Incorrect Punctuation or Other Delimiter)

เมื่อเขียนโปรแกรม Pascal เครื่องหมาย semi colon คือตัวคั่นหนึ่งคำสั่ง

(a statement separator) เมื่อเขียนโปรแกรม PL/1 หรือ Ada เครื่องหมาย semi colon เป็นตัวจบหนึ่งคำสั่ง (a statement terminator) เมื่อเขียนโปรแกรมภาษา BASIC ใช้เครื่องหมาย colon คั่นคำสั่งต่าง ๆ แต่ใช้เฉพาะสองคำสั่งขึ้นไปบนบรรทัดเดียวกัน ในภาษา COBOL เครื่องหมาย period เป็นตัวจบ หนึ่ง statement (a sentence terminator) แต่ statement ตั้งแต่หนึ่งคำสั่งหรือมากกว่า มาประกอบเข้าด้วยกันเป็น sentence ไม่จำเป็นต้องมี เครื่องหมายวรรคตอน ระหว่าง statements นอกจากนี้แล้ว ภาษา COBOL บางครั้ง ยอมให้ comma space คั่น ระหว่าง items ใน list แทนที่ จะเป็น a single space

ภาษา FORTRAN จบหนึ่งบรรทัดโดยนัย คือ จบหนึ่งคำสั่ง (In FORTRAN, the end of a line is implicitly a statement terminator.)

การรวมกลุ่ม ของกฎการใช้เครื่องหมายวรรคตอนเช่นนี้ แม้แต่ นักเขียนโปรแกรมที่มีประสบการณ์ ยังไม่เขียน (leaves out) เครื่องหมายวรรคตอน ซึ่งเป็นเครื่องหมายอาจจะไม่เขียนก็ได้ เนื้อหาของการออกแบบภาษาชุดคำสั่งนี้ ถูกร้องขอเพื่อทำให้องศาของการเป็นมาตรฐานมีมากขึ้น เนื่องจากมันเป็นแหล่งของข้อผิดพลาด วากยสัมพันธ์ ในการเขียนโปรแกรมที่เกิดบ่อยที่สุดหนึ่งอย่าง

B. <u>คำสั่งและการรวมกลุ่มรายการ</u>

(Statement and List Bracketing)

ข้อตกลงสำหรับ การจัดกลุ่ม หรือการใส่วงเล็บ กลุ่มของคำสั่งไม่เหมือนกันระหว่างภาษาต่าง ๆ ภาษา Pascal วงเล็บใหญ่ [ ] สำรองไว้สำหรับ ดรรชนีล่างของแถวลำดับ (array subscripts), วงเล็บเล็ก ( ) ใช้สำหรับอาร์กิวเม้นต์ ในโปรแกรมย่อย **begin** และ **end** ใช้สำหรับ compound statements และวงเล็บปีกกา { } ใช้สำหรับปิดคำอธิบาย

ภาษา FORTRAN เรามีวงเล็บเล็กสำหรับแถวลำดับ และ อาร์กิวเม้นต์ ของโปรแกรมย่อย IF ... ENDIF สำหรับ เงื่อนไขต่าง ๆ และ DO n ... n CONTINUE สำหรับ ลูป ส่วนภาษา PL/1 และ Ada ใช้วงเล็บเล็ก สำหรับ ดรรชนีล่างของแถวลำดับและอาร์กิวเม้นต์ต่าง ๆ ของโปรแกรมย่อย ความหลากหลายของสัญลักษณ์ต่าง ๆ สำหรับจัดกลุ่มคำสั่ง ได้ทำสรุปไว้ข้างล่างนี้ โดยเปรียบเทียบกับภาษา Pascal

Statement grouping	Pascal	PL/1	Ada
complete program	program P; begin : end	P: PROC; : END P;	procedure P is begin : end P

Looping	while e do
	begin
	
	end
		DO WHILE(e);
		:
		END;
			while e loop
			:
			end loop;

Selection	case i of : end	SELECT; : END;	select .: end select,;
Compounds,	begin : end	DO; : END;	begin : end

Conditionals	ifethen
	begin
	
	end
		IF e THEN
		DO;
		:
		END;
			if e then
			:
			end if;

ในที่นี้ จะเห็นว่า ภาษา Pascal และ Ada มีข้อตกลงที่พ้องกัน สำหรับจัดกลุ่ม มากกว่า ภาษา PL/1 ใน PL/1 คำสั่ง END ใช้ปิด การจัดกลุ่ม ที่แตกต่างกัน หลายชนิดและเป็นสาเหตุ ของ

ความยุ่งยาก เมื่ออ่านโปรแกรมที่ซับซ้อน ภาษา Ada ธรรมชาติของการปิด กำหนดให้อย่างชัดแจ้ง ดังนั้นจึงเป็น วากยสัมพันธ์ที่ชัดเจน สำหรับทั้งผู้อ่าน และโปรแกรมแปลชุดคำสั่ง โดยที่โปรแกรมแปลชุดคำสั่ง สามารถใช้สารสนเทศ พิเศษนี้ กระทำการตรวจสอบวากยสัมพันธ์ ที่เชื่อถือได้มากขึ้น

<u>ตัวอย่าง</u> จงพิจารณา ข้อผิดพลาดในส่วนของโปรแกรม PL/1 และ Ada ข้างล่างนี้

PL/1	Ada
DO WHILE (e);	**while** e **loop**
:	:
IF e THEN DO;	**if** e **then**
:	:
END;	**end loop**

ทั้งสองโปรแกรม ขาด หนึ่ง END (หรือ **end**) แต่ส่วนของ Ada เท่านั้น มองเห็นจุดที่ผิดพลาด ซึ่งขาด <u>end</u> ไป หนึ่งตัว และตัวไหนที่มีอยู่ ในตัวอย่าง PL/1 เห็นไม่ชัดว่า END ตัวไหนที่ขาดหายไป จนกระทั่งจบโปรแกรม ทั้งหมดแล้ว โปรแกรมแปลชุดคำสั่ง จึงจะตรวจพบว่ามีการขาดหายไป ของ END ณ ที่ใดที่หนึ่ง สิ่งนี้ ไม่ใช่ สารสนเทศ การวินิจฉัยที่ดีมากสำหรับโปรแกรมขนาดใหญ่ และซับซ้อน

C. <u>ความหลายหลายของอภิภาษา</u>

(Diversity of Metalanguage)

เมื่อเราศึกษาภาษาต่าง ๆ จะสังเกตได้ว่า แต่ละภาษา มีเพียงหนึ่งเดียวเท่านั้น ในวิธี ให้ชื่อประเภทโทเค็น และโครงสร้างการเขียนโปรแกรมอื่น ๆ สิ่งนี้น่ารำคาญ โดยเฉพาะเมื่อความแตกต่างนี้ เกิดขึ้น ให้เป็นความแตกต่างอย่างสำคัญ หรือพูดง่ายคือ การให้ชื่อแตกต่างกัน กับ ความคิดที่เป็นสิ่งเดียวกัน

ตัวอย่างเช่น "identifier" ในภาษา Pascal เป็นสิ่งเดียวกับ "data-name" ในภาษา COBOL ส่วนแถวลำดับ เรียกว่า "table" ในภาษา COBOL ในขณะที่คำสั่งกำหนดค่าเรียกว่า COMPUTE statements ส่วนโครงสร้างในภาษา PL/1 เรียกอีกอย่างหนึ่งว่า "records" ในภาษา Pascal และเรียกว่า "record description entries" ในภาษา COBOL ความคิดที่เข้าใจได้อย่างดีมีเหตุผล ในภาษา ALGOL เรื่อง "type" (the resonably well-understood notion of "type" in Algol-like languages) หมายถึงเซตของค่าต่าง ๆ ซึ่ง ตัวแปรหนึ่งตัว อาจจะมี สิ่งนี้เหมือนกับ เซตของ attributes ของตัวแปร PL/1 ชนิดต่าง ๆ จะมีชื่อต่างกันด้วยในภาษาที่ไม่เหมือนกัน เช่น

Pascal type	PL/1 equivalent	COBOL equivalent	FORTRAN equivalent
real	FLOAT	COMP-1	REAL
integer	FIXED BIN	COMP-2	INTEGER
Boolean	BIT(1)	PIC '9'B	LOGICAL
character	CHARACTER(1)	PIC 'X'	CHARACTER *1

นี่เป็นเพียง ตัวอย่างเล็กน้อยของความแตกต่าง ความแตกต่างยังมีอีกมากมาย จะปรากฏให้เห็น เมื่อ เราศึกษาภาษาต่าง ๆ เช่น APL, LISP และ PROLOG ในบทต่อ ๆ ไป

2.8 วากยสัมพันธ์ และความหมาย

(Syntax and Semantics)

ถึงแม้ว่า วากยสัมพันธ์ จะเกี่ยวข้องเฉพาะรูปแบบของโปรแกรมเท่านั้น มันยัง ผูกติดกับ "semantics" ซึ่งหมายถึง ความหมายของโปรแกรม ปกติ ความหมาย นิยามในเทอมของพฤติกรรม ณ เวลาดำเนินงาน ของโปรแกรม (Usually, semantics is defined in terms of the program's run-time behavior:) ได้แก่

- เกิดอะไรขึ้น เมื่อ โปรแกรมถูกปฏิบัติการด้วย เซตของอินพุท (what happens when the program is executed with a certain set of inputs)
- คำสั่งอะไรถูกปฏิบัติการ (what statements are executed)
- ค่าอะไร ถูกกำหนดให้กับ ตัวแปรต่าง ๆ (what values are assigned to the variables)
- และผลิตเอ้าพุทอะไร (what output is produced)

เพราะว่าเป้าหมายพื้นฐานของการออกแบบภาษาชุดคำสั่ง คือ เพื่อนิยามความหมายสำหรับอธิบายกรรมวิธีต่าง ๆ ของการคำนวณ วากยสัมพันธ์ เกิดขึ้นโดยหลักการเพื่อบริการ (serve) การจบความหมายเหล่านี้ ดังนั้น จุดประสงค์ของความหมาย คือ แรงจูงใจเริ่มแรกสำหรับ การออกแบบวากยสัมพันธ์ (Thus, semantic goals are the original motivation for syntax design.)

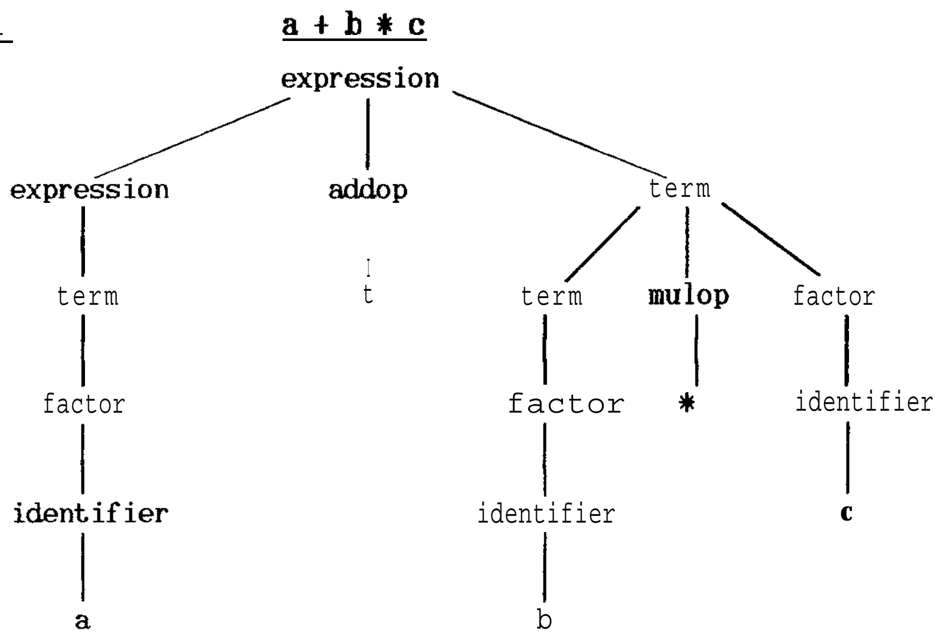
การต่อเนื่องระหว่าง วากยสัมพันธ์ และความหมาย จะเห็นได้อย่างชัดเจน ใน บทนิยาม BNF ของนิพจน์คำนวณ และผลลัพธ์การกระจาย ซึ่งได้มาโดยบทนิยามนั้น

จงพิจารณาการผลิต (productions) โดยย่อ สำหรับนิพจน์ Pascal ข้างล่างนี้

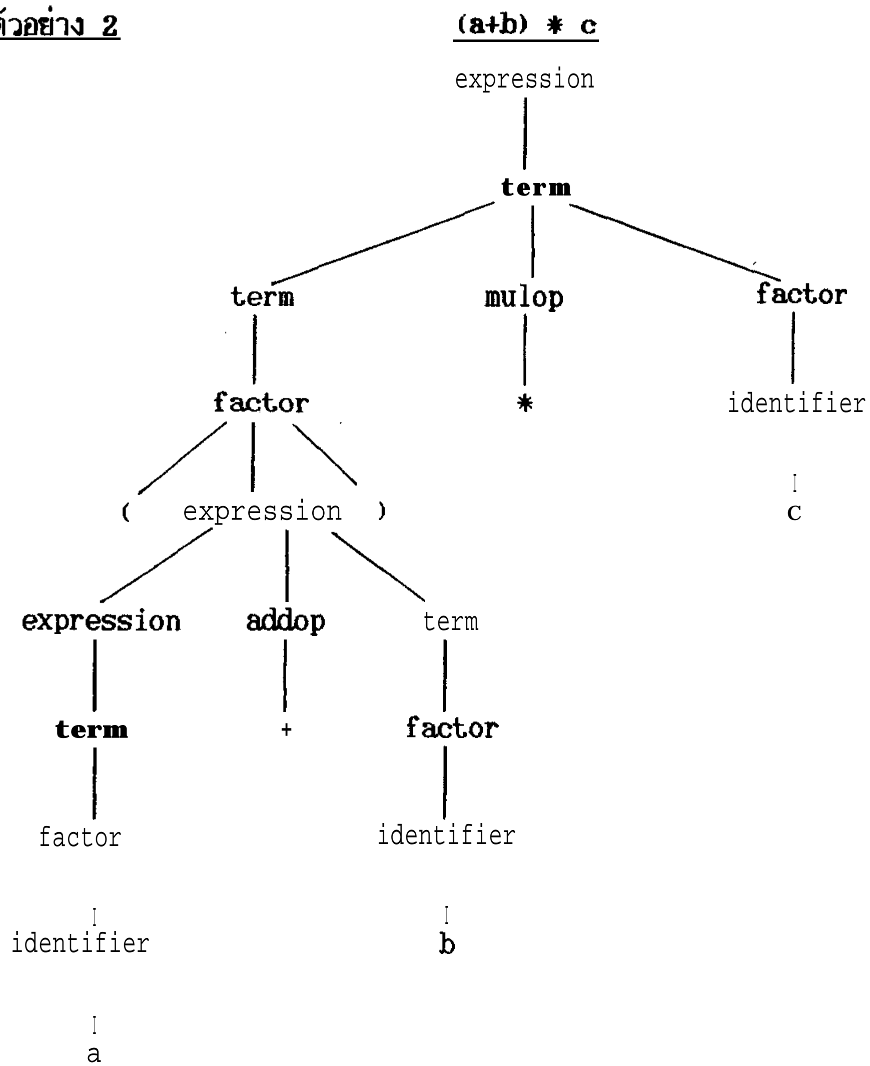
```
expression  ::= [+ | -] term |
                expression addop term
addop       ::= + | - | or
term        ::= factor | term mulop factor
mulop       ::= * | / | div | mod | and
factor      ::= identifier | number | (expression)
```

การผลิตเหล่านี้ ควบคุม การกระจาย นิพจน์ ต่อไปนี้

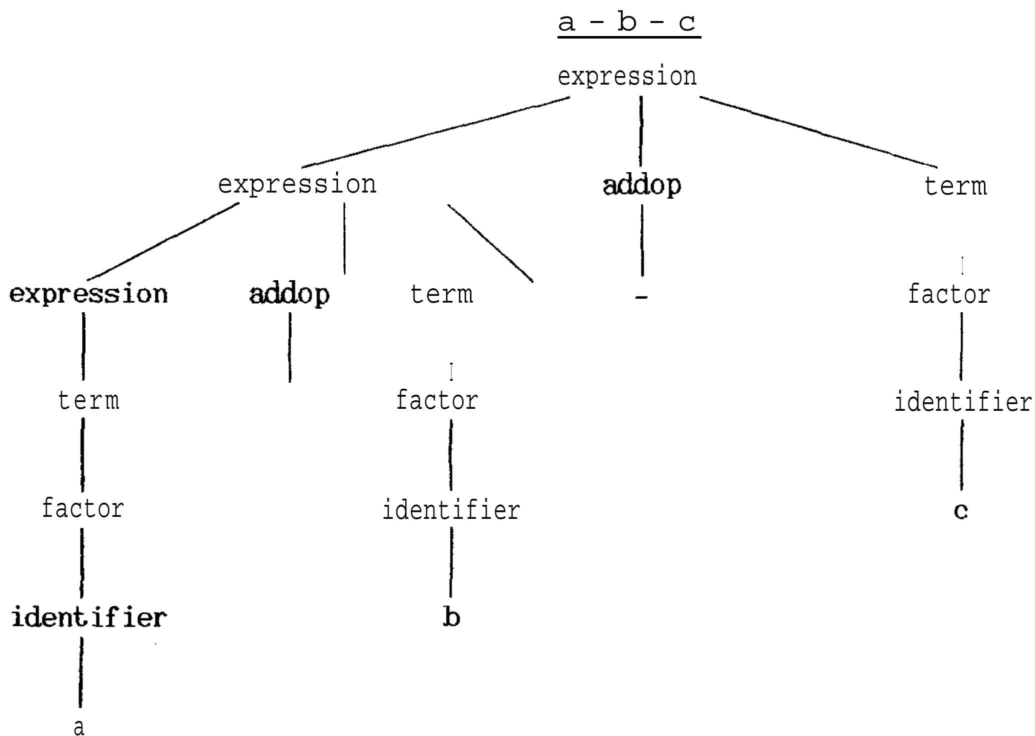
<u>ตัวอย่าง 1</u> <u>**a + b * c**</u>



ตัวอย่าง 2



ตัวอย่าง 3



จะเห็นว่า ต้นไม้กระจาย เป็น บทนิยาม ของลำดับ (order) ของการปฏิบัติงาน ใน นิพจน์คำนวณ ในลักษณะแบบ จากล่างขึ้นบน ซ้ายไปขวา (in a bottom-up, left-right sense) ในการกระจายตัวอย่างแรกข้างต้น สิ่งนี้ หมายความว่า นิพจน์ a + b * c หมายถึง การคูณกันของ b * c แล้วตามด้วย การบวกกัน ของผลลัพธ์ กับ a นั่นคือ <u>ลำดับความสำคัญ</u> (priority) ของการคูณ สูงกว่า การบวก ถูกบังคับในวากยสัมพันธ์

การกระจาย ตัวอย่างที่สอง จะเห็นว่า เครื่องหมายวงเล็บ นำมาใช้เพื่อแสดงลำดับความสำคัญที่สูงกว่า ในที่นี้ การบวกของ a + b กระทำเป็นอันดับแรก และจากนั้น ผลลัพธ์ที่ได้คูณกับ c

การกระจายชุดที่สาม วากยสัมพันธ์ แสดงให้เห็นว่า เพื่อบังคับการประเมินผลของตัวปฏิบัติการที่มีความสำคัญเท่ากัน จากซ้ายไปขวา (left-to-right) ในที่นี้ คำนวณ a - b เป็นสิ่งแรก จากนั้น จึงเอา c ไปลบออกจากผลลัพธ์

นอกจากนี้แล้ว ผู้อ่านควรจะมั่นใจว่า กฎ BNF สำหรับนิพจน์ ไม่ยอมให้มี การกระจายอื่นใดเลย สำหรับนิพจน์เหล่านี้ นั่นคือ วากยสัมพันธ์ ไม่กำกวม

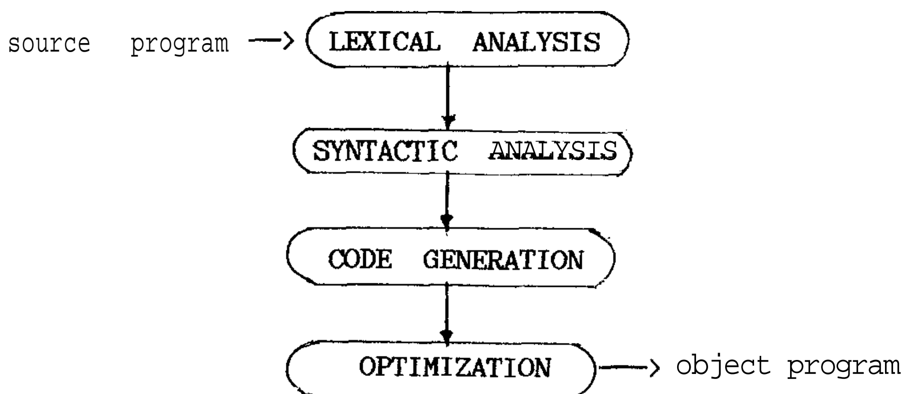
การศึกษาในเนื้อหา ความหมาย ของภาษาชุดคำสั่ง ยังมีอีกมากมาย นี่เป็นเพียงการแสดงให้เห็นโดยย่อเฉพาะ การโยงกันที่สำคัญ ระหว่าง วากยสัมพันธ์ และความหมาย หัวข้อความหมายอื่น ๆ เช่น การบังคับ (coercion), การจัดสรรหน่วยความจำ (storage allocation), การเชื่อมโปรซีเดอร์ (procedure linkage) และการอ้างถึงแถวลำดับ (array referrencing) จะอภิปรายรายละเอียด ในบทที่ 3

2.9 วากยสัมพันธ์ ความหมาย และการออกแบบโปรแกรมแปลชุดคำสั่ง

(Syntax, Semantics, and Compiler Design)

หัวใจของการทำภาษาให้เกิดผล คือ การออกแบบตัวแปลชุดคำสั่ง ที่มีประสิทธิภาพสำหรับภาษานั้น การออกแบบตัวแปลชุดคำสั่ง เป็นเรื่องซับซ้อน และสมควรเป็นหนึ่งกระบวนวิชาเต็ม (At the heart of language implementation is, of course, the design of effective compilers for the languages. Compiler design is, of course, a complex subject and deserves a full course in itself.) อย่างไรก็ตาม ในที่นี้ เราจะร่าง การออกแบบพื้นฐานของตัวแปลชุดคำสั่งอย่างคร่าว ๆ เพื่อว่าจะได้มีความเข้าใจอย่างดีถึง การโยงระหว่างวากยสัมพันธ์ของภาษา ความหมาย และการทำให้เกิดผล

สมาชิกพื้นฐานของตัวแปลชุดคำสั่ง แสดงให้เห็นดังนี้



ในที่นี้ ชุดคำสั่ง ภาษาต้นฉบับ (source program) ถูกจัดให้ทำการวิเคราะห์ศัพท์ (lexical analysis) ซึ่งเป็นงาน เพื่อรู้จำ โทเค็นพื้นฐาน ซึ่งเกิดขึ้นในโปรแกรม และแยกประเภทโทเค็นเหล่านี้ว่า ตัวไหนเป็น ค่าคงที่ ไอเดนติไฟเออร์ คำสงวน และอื่น ๆ ดังนั้นขั้นตอนนี้

แปลงผัน (converts) ตัวโปรแกรมจากรูปแบบสายอักขระ ให้เป็นชุดรายการโทเค็นของภาษา การสร้างขั้นแรกของตารางสัญลักษณ์ เกิดขึ้นในขั้นตอนนี้เช่นกัน

การวิเคราะห์วากยสัมพันธ์ (Syntactic analysis) แปลงผันรายการนี้ ให้เป็นต้นไม้กระจาย โดยใช้การแทนที่ภายในของ ไวยากรณ์ของภาษาเป็นแนวทางของมัน ยุทธวิธีหลายอย่างที่มีอยู่ สำหรับการวิเคราะห์วากยสัมพันธ์ เพราะว่ามันเป็นกรรมวิธีที่ซับซ้อน และต้องทำให้มีประสิทธิภาพเท่าที่เป็นไปได้

การก่อกำเนิดรหัส (Code generation) เป็นการโยงอย่างพื้นฐาน ระหว่างวากยสัมพันธ์ของภาษา กับ ความหมายของมัน หรือการแทนที่ภายในเครื่อง นั่นคือ มันแปลงผันต้นไม้กระจาย ให้เป็น รายการของคำสั่งเครื่อง (list of assembly (machine) instructions) ที่มีความหมายเหมือนกัน สำหรับโปรแกรม การอภิปราย ในบทที่ 10 จะให้รายละเอียดที่มองเห็นมากขึ้นในการก่อกำเนิดรหัส แสดงให้เห็น การก่อกำเนิดรหัสแอสเซมบลีสำหรับการสร้างภาษาระดับสูง

สุดท้าย การเล็งผลเลิศ (Optimization) คือความพยายามที่จะทำให้รหัสก่อกำเนิดชัดเจนยิ่งขึ้น เพื่อว่าการกระทำขณะเวลาดำเนินงาน (run-time) ของมัน จะได้ปรับให้ดีขึ้น สิ่งนี้เป็นกรรมวิธีซับซ้อนเช่นกัน เป็นความพยายามที่จะหาตำแหน่งการสร้างซ้ำเชิงความหมาย (as it attempts to locate semantically redundant constructs) การใช้เรจิสเตอร์อย่างไม่มีประสิทธิภาพ (inefficient use of registers) และอื่นๆ บ่อยครั้งที่ขั้นตอนนี้ เกิดขึ้นทั้งก่อนหน้าและภายหลัง การก่อกำเนิดรหัส การปฏิบัติการกรณีแรก กระทำโดยตรงบนต้นไม้กระจาย

ชุดคำสั่งภาษาจุดหมาย (object program) ซึ่งได้มาจากการแปลชุดคำสั่ง (compilation) อาจเป็นภาษาเครื่อง หรือเป็นภาษากลาง (intermediate language) บางภาษา ภาษาหลังนั้น เป็นที่ชื่นชอบในหลายกรณี เมื่อต้องการความสามารถในการเคลื่อนย้ายได้ (where portability is desired)

จากการผ่านขั้นตอนทั้งหมด จะเห็นชัดเจนว่า ตารางสัญลักษณ์ กระทำบทบาทส่วนกลาง นอกจากนี้แล้ว ยุทธวิธีทั้งหมด สำหรับการรายงาน ข้อผิดพลาดวากยสัมพันธ์ หรือ "diagnostics" ต้องรวมเข้าไว้กัน ภายในการออกแบบนี้เช่นกัน

space ไม่ได้รับอนุญาต ให้ได้รับการปฏิบัติ ของการออกแบบโปรแกรมแปลชุดคำสั่ง ในตำราเล่มนี้ ตำราที่ดีเยี่ยมหลายเล่ม ได้เขียนเกี่ยวกับหัวข้อนี้ และสมควรจะได้ศึกษาด้วยตนเอง นอกเนื้อหา การศึกษา ภาษาชุดคำสั่ง

แบบฝึกหัด

1. จงออกแบบ BNF syntax สำหรับคำสั่ง if ในภาษา Ada เช่นที่ได้อธิบายในบทนี้ แล้ว กระจายคำสั่งข้างล่างนี้ โดยใช้วากยสัมพันธ์ ของท่าน

   if x = 0 then if y = 0 then **s1** else **s2** **endif endif**

2. จงกระจาย นิพจน์ต่อไปนี้ โดยใช้ วากยสัมพันธ์ที่กำหนดให้ในบทนี้

   (a) 1 + b * c + d

   (b) 1

   (c) (1 + b) * (c + d)

3. จงแสดงให้เห็นว่า BNF syntax ข้างล่างนี้ กำกวม โดยการหา นิพจน์ ซึ่ง มีการกระจาย ได้ตั้งแต่ 2 วิธีที่แตกต่างกันขึ้นไป

```
expression  ::= term I expression t expression
      term  ::= factor I term * term
    factor  ::= identifier I number I (expression)
```

4. จงเขียน syntax charts ซึ่งมีความหมายเหมือนกับ BNF productions สำหรับ "expression" ซึ่งกำหนดให้ในบทนี้

5. จงเขียน Pascal procedure ซึ่งรู้จำได้ (recognizes) ว่าสายอักขระที่กำหนดให้มีส่วนแรกเป็น "identifier" ถูกต้อง หรือไม่ ถ้ามี ให้ส่งคืน ส่วนนั้น ถ้าไม่มี, ให้ procedure นี้ ส่งคืน สายอักขระว่าง (empty string) แล้ว เขียน อีกหนึ่ง procedure ทำสิ่งเดียวกัน สำหรับตัวเลข (number)

6. จงใช้ BNF syntax ซึ่งกำหนดให้ ในบทนี้ กระจายคำสั่งต่อไปนี้

   (a) if a < b t c then **a** := 0 as an if-stmt

   (b) a := b - c t d as an assignment-stmt

   (c) begin x := 0; y := 0 **end** as a compound-stmt

7. บทนิยามวากยสัมพันธ์ที่ดี ไม่ได้รับประกันว่า บทนิยาม ความหมายดีด้วย จงกระจาย "sentence" ต่อไปนี้ โดยใช้กฎ BNF ที่กำหดให้ในบทนี้

   (a) THE GIRL BIT THE DoG

   (B) THE DOG **WROTE**

8. จงเปรียบเทียบ ข้อตกลงที่ใช้แยกคำสั่งต่าง ๆ ในโปรแกรม Pascal, FORTRAN, COBOL และ PL/1 และ ให้บอกข้อดี ข้อไม่ดีของแต่ละชนิด

9. syntax ต่อไปนี้ บางส่วนใช้นิยาม โครงสร้าง ของ ลูปต่าง ๆ ใน Ada

<loop> ::= [<iterator>]<basic loop>

<iterator> ::= while <exp> |

for <var> in <subrange>

<basic loop> ::= loop <stmts> end loop

<stmts> ::= <stmt>{<stmr>}

ในที่นี้ <exp>, <var>, <stmt> และ <subrange> หมายถึง expressions, variables, statements และ subscript ranges ใน Pascal สำหรับ syntax นี้ จงแสดง ให้เห็น ลูปของ Ada ที่มีความหมาย เหมือนกับ ลูปของ Pascal ข้างล่างนี้ จากนั้น จงแสดงต้นไม้กระจายของมัน

```
for i := 1 to 10 do
   begin
      <stmt>;
      <stmt>
   end
```

10. จงเขียน แผนภาพวากยสัมพันธ์ (syntax diagrams) ซึ่งมีความหมายเหมือนกับ BNF productions ในคำถามที่ผ่านมา

11. ภาษา FORTRAN ไม่สนใจ (ignores) blanks ซึ่งปรากฏในโปรแกรม แต่บ่อยครั้งที่ ปฏิบัติว่ามันเป็นศูนย์ (0) เมื่อมันเป็น อินพุท ตัวอย่างเช่น ลำดับ "1 000" แทน เลข 1000 ทั้งสองแห่ง ในคำสั่ง "IF (X.EQ.1 000) GO TO 1 000" อย่างไรก็ตาม มันแทน เลข 10000 ถ้าอ่าน ภายใต้รูปแบบ I5 จากอินพุท จงตรวจสอบการปฏิบัติ (treatment) ของ blanks ในภาษาอื่น ๆ ซึ่งท่านได้ศึกษาไปแล้ว ความไม่สอดคล้องนี้ ปรากฏที่ใดหรือไม่?