บทที่ 6

แบบชนิดข้อมูล (Data Types)

6.1 แบบชนิดข้อมูล และสารสนเทศของชนิด

(Data Types and Type Information)

6.2 ชนิดอย่างง่าย (Simple Types)

6.3 ตัวสร้างชนิด *(Type Constructors)*

6.4 ชนิด Nomenclature aนภาษาเหมือน Pascal

(Type Nomenclature in Pascal-llke Languages)

6.5 ความสมมูลของชนิด (Type Equivalence)

6.6 การตรวจสอบชนิด (Type Checking)

6.7 การแปลงผันชนิด (Type Conversion)

แบบฝึกหัด (Exercises)

บทที่ 6

แบบชนิดข้อมูล (Data Types)

โปรแกรมทุกโปรแกรมใช้ข้อมูล อาจจะชัดแจ้ง หรือ โดยนัย เพื่อให้ได้ผลลัพธ์ ข้อมูลในโปรแกรมรวบรวม ขึ้นเป็นโครงสร้างข้อมูล ซึ่งจะถูกคุมแต่งโดย โครงสร้างควบคุม ซึ่งแทนอัลกอริทึม สิ่งนี้ แสดงให้เห็นชัดเจนด้วย สมการเทียม ต่อไปนี้

algorithms + data structures = programs

รูปแบบดั้งเดิมมากที่สุด ของ ข้อมูล ภายในคอมพิวเตอร์ เป็นเพียง กลุ่มของบิต

(Data in its most primitive form inside a computer is just a collection of bits.)

ภาษาโปรแกรม นำสิ่งนี้ เป็นมูลฐานของมัน และสร้างข้อมูลทั้งหมดจากบิต สิ่งนี้สำคัญซึ่งเป็นส่วนของบทนิยามของภาษา แต่มันซับซ้อน และไม่สะดวก และไม่เป็นการจัดหาการนิยามนามธรรม และไม่เป็นอิสระจากเครื่องคอมพิวเตอร์ ซึ่งภาษาโปรแกรมพยายามที่จะจัดหาให้

ภาษาโปรแกรมส่วนใหญ่ จัดหา เซต ของเอนทิตีข้อมูล อย่างง่าย เช่น integers reals และ Booleans เช่นเดียวกับ กลไกต่างๆ สำหรับการสร้าง เอนทิตีข้อมูลใหม่ จากสิ่งเหล่านี้

การนิยามนามธรรม เป็นกลไกที่สำคัญในภาษาโปรแกรม และมีส่วนช่วยเหลือ (contribute) เป้าหมายการออกแบบเกือบทุกอย่าง: การอ่านง่าย (readability) การเขียนง่าย (writeability) ความเชื่อถือได้ (reliability) และความเป็นอิสระจากเครื่อง (machine independence)

การใช้การนิยามนามธรรม เพื่ออธิบายข้อมูล ทำให้เกิดกำลัง (power) และประโยชน์มากมาย กับภาษาโปรแกรม จุดมองของข้อมูล ในภาษาโปรแกรม จะเริ่มด้วย แนวคิด ของ **แบบชนิดข้อมูล** (data types) ซึ่งเป็นกลไกการนิยามนามธรรมพื้นฐาน ชนิดที่สำคัญ (principal types) และตัวสร้างแบบชนิดข้อมูล (type constructors) ที่มีให้ใช้ ในภาษาโปรแกรมต่างๆ

6.1 แบบชนิดข้อมูล และสารสนเทศของชนิดข้อมูล

(Data Types amd Type Information)

ข้อมูลของโปรแกรม ถูกจำแนกอออกตาม**ชนิด** (types) ของมัน ตัวอย่างเช่น -1 เป็นชนิด integer และ 3.14159 เป็นข้อมูลชนิด real การรวมชนิดข้อมูล เข้าไปใน บทนิยาม ของภาษาโปรแกรม มีความสำคัญ สำหรับ ความเชื่อถือได้, การอ่านง่าย และการบำรุงรักษาได้ (maintainability) : มันทำให้ตัวแปลภาษา บอกได้ว่า ชนิดของค่า หรือวัตถุนั้น ถูกต้องหรือไม่ (type checking) และยอมให้ โปรแกรมเมอร์ และตัวแปลภาษา ได้ให้ข้อสรุปเกี่ยวกับวิธีซึ่งจะสามารถนำค่ามาใช้ และการดำเนินงาน ซึ่งจะสามารถประยุกต์ใช้ได้กับค่าเหล่านั้น

ในบทที่ 5 เราได้เห็นแล้วว่า ชนิดของตัวแปร บ่อยครั้งเกี่ยวข้องกับตัวแปร ด้วยการประกาศ เช่น

```
var x : ineteger;
```

ซึ่งกำหนด แบบชนิดข้อมูล integer ให้กับ ตัวแปร x ในการประกาศ คล้ายกับตัวอย่างนี้ ชนิดข้อมูลเป็นเพียง ชื่อ (name) ซึ่ง มีคุณสมบัติบางอย่าง เช่น ชนิดของ ค่าซึ่งสามารถเก็บได้ และวิธีซึ่งค่าเหล่านี้ จะถูกแทนที่ภายในเครื่องคอมพิวเตอร์

เนื่องจาก การแทนที่ภายใน เป็น คุณสมบัติ ซึ่งขึ้นอยู่กับระบบ จากจุดการมองแบบนามธรรม

(Since the internal representation is a system-dependent feature, from the abstract pointof of view.)

เราสามารถพิจารณาว่า ชื่อของชนิด (type name) แทน ค่าที่เป็นไปได้ ซึ่งตัวแปร ของชนิดนั้น สามารถเก็บได้ ถ้าเป็นนามธรรมมากขึ้น คือ ชื่อชนิด เป็นอย่างเดียวกับ เซตของค่าต่างๆ ที่มันแทน โดยให้นิยามดังนี้

> **แบบชนิดข้อมูล** หมายถึง เซตของค่าต่างๆ
>
> (A **data type** is a set of values.)

แบบชนิดข้อมูล เป็นเซต ซึ่งกำหนดได้ หลายวิธี :

- เขียนรายการ หรือ แจงนับ อย่างชัดแจ้ง

  (it can be cxplicity listed or enumerated)

- กำหนดให้เป็นพิสัยย่อย ของ ค่าอื่นๆ ซึ่งทราบแล้ว

  (it can be given as a subrange of otherwise known values)

- ขอยืมมาจากวิชาคณิตศาสตร์ ในกรณีซึ่งไม่สนใจ การจำกัดของเซต ในการทำให้เกิดผลจริง

  (it can be borrowed from mathematics, in which case the finiteness of the set in an actual implementtion my be left vague or ignored.)

การดำเนินการบนเซต สามารถนำมาประยุกต์ใช้เพื่อให้ได้ เซตใหม่ๆ นอกเหนือจากของเก่า (ดูหัวข้อ **6.3**)

ตัวอย่าง

ภาษา Pascal มี ตัวคงที่นิยามมาแล้ว (predefined constant) ชื่อ **maxint** ซึ่งขึ้นอยู่กับระบบ

และหมายถึง จำนวนเต็ม ใหญ่ที่สุด ซึ่งสามารถแทนได้บนระบบ ดังนั้น ชนิด integer ใน Pascal ซึ่งได้แก่ พิสัย 0 .. maxint และปกติคือ (- maxint - 1) .. maxint ถ้าภายใต้การแทนที่ เป็นรูปแบบ ส่วนเติมเต็ม ของสอง (two's complement form)

บนคอมพิวเตอร์ขนาดเล็ก ในการเก็บ integers ใช้เนื้อที่ สองไบต์ (two bytes) ดังนั้น maxint = 32767 และพิสัย integer คือ จาก -32768 ถึง 32767

ภาษา Modula-2 มี predefined functions ชื่อ MAX และ MIN ให้ค่าใหญ่ที่สุด และค่าเล็กที่สุด ของ แบบชนิดข้อมูล ดังนั้น ในกรณีนี้ MAX(INTEGER) = 32767 และ MIN(INTEGER) = -32768

เซตของค่าต่างๆ โดยทั่วไป จะมีกลุ่มของการดำเนินการ ซึ่งสามารถประยุกต์ใช้ กับค่าเหล่านั้น การดำเนินการเหล่านี้ บ่อยครั้ง ไม่ได้กล่าวอย่างชัดแจ้ง กับ ชนิด แต่อยู่ในส่วนของบทนิยามของมัน ตัวอย่างเช่น การดำเนินการ คำนวณบน integers หรือ reals การดำเนินการ successor และ predecessor บน enumerations และการดำเนินการแบบ field dereferencing บนชนิดระเบียน

การดำเนินการเหล่านี้ อาจมีคุณสมบัติเฉพาะด้าน หรือไม่มีคุณสมบัติเฉพาะด้าน กล่าวไว้อย่างชัดแจ้ง

ตัวอย่าง

succ(pred(x)) = x

หรือ

x + y = y + x

ดังนั้น เราอาจต้องการแก้ไข (revise) บทนิยามชุดแรก ให้รวมการดำเนินการ อย่างชัดแจ้ง ดังนี้

> **แบบชนิดข้อมูล** หมายถึง เซตของค่าต่างๆ รวมกับ เซตของการดำเนินการ บนค่าเหล่านั้น ซึ่งมีคุณสมบัติแน่นอน (A **data type** is a set of values, together with a set of operations on those values having certain properties.)

ในแง่นี้ แบบชนิดข้อมูล จึงเป็น พีชคณิตเชิงคณิตศาสตร์ อย่างแท้จริง ตัวแปลภาษาโปรแกรม สามารถใช้สารสนเทศของชนิดข้อมูล ใน หลายวิธี สามารถตรวจสอบความถูกต้อง

(validity) ของการดำเนินการ และการกระทำ ด้วยเหตุนี้ จึงเป็นการตรวจจับข้อผิดพลาดที่ดีขึ้น (thus providing improved error detection.) ตัวอย่างเช่น ตัวดำเนินการ หาร "/" อาจมีข้อจำกัดกับค่า real เท่านั้น ในขณะที่ truncated division (div) ใช้ได้เฉพาะกับ integers เท่านั้น รวมทั้ง การกำหนดค่าของ **values** จากตัวแปร ไปยัง ตัวแปร เช่น

x := y

ใช้ได้เฉพาะ เมื่อ ชนิดของ x และ y เป็นชนิดเดียวกัน หรือ เกี่ยวข้องกันอย่างใกล้ชิด ในบางแง่

ตัวแปลภาษา สามารถใช้สารสนเทศของชนิด (type information) เพื่อจัดสรรเนื้อที่ ให้กับตัวแปร การจัดสรรเนื้อที่อย่างมีประสิทธิภาพ เป็น หนึ่ง ใน เหตุผลที่สำคัญ ซึ่งจะต้องมีสารสนเทศของชนิด ให้ใช้ ณ เวลาแปลโปรแกรม โครงร่างของการจัดสรรปกติจะกล่าวถึงโดยย่อกับข้อมูลแต่ละชนิดและตัวสร้างชนิดข้อมูล สารสนเทศของชนิดอาจจะอยู่ในโปรแกรมโดยนัย หรือชัดแจ้ง

สารสนเทศของชนิดโดยนัย (Implicit type information) ได้แก่ ชนิดของ constants และ values ชนิดซึ่ง สามารถอนุมาน (inferred) ได้จากข้อตกลงของชื่อ และชนิดซึ่งอาจจะอนุมานได้จากบริบท (context) ตัวอย่างเช่น ในภาษาส่วนใหญ่ เลข 2 เป็น integer โดยนัย TRUE เป็น Boolean และ ตัวแปร I ในภาษา FORTRAN เมื่อไม่มีสารสนเทศอื่นๆ จะหมายถึง ตัวแปรชนิด integer

สารสนเทศชนิดชัดแจ้ง (Explicit type information) เริ่มตั้งแต่อยู่ใน**การประกาศ** (declarations)

ตัวอย่าง ตัวแปร ซึ่งประกาศเป็นชนิดเฉพาะด้าน

```
var  x : array [1 .. 10] of integer;
     b : boolean;
```

แต่ในหลายภาษา มันเป็นเป็นไปได้ ที่จะให้ชื่อกับ ชนิดใหม่ (new types) ใน**การประกาศชนิด** (type declaration)

ตัวอย่าง

```
type intarray = array [1 .. 10] of integer;
```

บางภาษาเรียกการประกาศเหล่านี้ว่า **บทนิยามของชนิด** (type declarations) เพราะว่ายังไม่มีการจัดสรรเนื้อที่จริง เช่นที่ทำกับการประกาศตัวแปร เราจะใช้คำว่า **declaration** สำหรับการประกาศชนิดและการประกาศตัวแปรทั้งคู่

กระบวนการ ของ ตัวแปลภาษา ในการบอกว่า สารสนเทศของชนิด ใน โปรแกรม **ต้อง**

กัน (consistent)) หรือไม่ เรียกว่า **การตรวจสอบชนิด**

(The process a translator goes through to determine whether the type information in a program is consistent is called **type checking**.)

การตรวจสอบชนิด เกี่ยวข้องกับ กฎต่างๆ สำหรับการกำหนด เมื่อสองชนิดเหมือนกัน : สิ่งนี้ เรียกว่า **ความสมมูลของชนิด**

(Type checking involves rules for detrmining when two types are the same : this is **type equivalence**.)

อัลกอริทึม สำหรับ ความสมมูลของชนิด จะศึกษา ในหัวข้อ 6.5

การตรวจสอบ ยังใช้กฎต่างๆ สำหรับการอนุมาน ชนิดของตัวสร้างภาษา จาก สารสนเทศของชนิดที่มีให้ใช้

กลุ่มของกฎ **การอนุมานชนิด** เหล่านี้ อัลกอริทึมความสมมูลของชนิด และวิธีต่างๆ ซึ่งใช้สำหรับสร้างชนิด ทั้งหมดนี้ เรียกว่า **ระบบของชนิด**

(The collection of these **type inference** rules, the type equivalence algorithm, and the methods used for constructing types are collectively referred to as a **type system**.)

ถ้าบทนิยามของภาษาโปรแกรม กำหนดว่า วัตถุ (objects) ทั้งหมดของภาษา ต้องเป็นชนิด well-defined ซึ่งกำหนดไว้อย่างคงที่ และกำหนด เซตบริบูรณ์ของกฎ สำหรับ ความสมมูล ของชนิด และการอนุมานชนิด ซึ่งจะประยุกต์ใช้อย่างคงที่ เราเรียกภาษานั้นว่า **ภาษาชนิดแข็งแรง** (strongly typed language) พูดอีกอย่างหนึ่งคือ จากบทนิยามนี้ หมายความว่า ข้อผิดพลาดทั้งหมดของชนิด ในภาษาชนิดแข็งแรง บอกได้ ณ เวลาแปลโปรแกรม

(this definition means that all type errors in a strongly typed language can be determined at translation time.)

อย่างไรก็ตาม ในภาษาชนิด strongly typed ส่วนใหญ่ ข้อยกเว้น (exceptions) ซึ่งกระทำกับเงื่อนไขนั้น ตรวจสอบยาก หรือ เป็นไปไม่ได้ที่จะตรวจสอบ ณ เวลาแปลโปรแกรม เช่น ขอบเขตของพิสัยย่อย (subrange bounds) หรือ ระเบียนแปรผัน (record variants)

ภาษา Ada และ Algol68 เป็นภาษาชนิด strongly typed

Modula-2 และ Pascal ปกติถือว่าเป็นภาษาชนิด strongly typed เช่นกัน ถึงแม้ว่าจะมี loopholes เล็กน้อย

ภาษา C มี loopholes มากกว่า และไม่ใช่ภาษาชนิด strongly typed

ภาษา ซึ่งไม่มี **ระบบชนิดคงที่สมบูรณ์** (languages without complete static type systems) เรียกว่า **ภาษาชนิดอ่อน** (weakly typed languages) **หรือ** untyped languages **ได้แก่ภาษา** LISP, APL, SNOBOL, และ BLISS

กฎการตรวจสอบชนิด จะดูในหัวข้อ 6.6 วิธีต่างๆ ของการตรวจสอบชนิด relaxing ใน ภาษา strongly typed จะดูในหัวข้อ 6.7 แต่สิ่งแรก เราต้องการศึกษา **ชนิดพื้นฐาน ที่มีให้ใช้** ใน ภาษาต่างๆ และตัวสร้างชนิดร่วม หรือ วิธีต่างๆ ของการสร้างชนิดใหม่ จาก **ชนิดที่มีอยู่แล้ว**

6.2 <u>ชนิดอย่างง่าย</u> (Simple Types)

ภาษาเหมือน Algol (เช่น Pascal, Algol68, C, Modula-2, Ada) **ทั้งหมดนี้ แบ่งชนิดตาม** โครงร่างพื้นฐาน กับ minor variations โชคไม่ดี ชื่อซึ่งใช้ในบทนิยาม ของภาษาซึ่งแตกต่างกัน บ่อยครั้ง ใช้ชื่อแตกต่างกัน ทั้งๆ ที่มีแนวคิดเหมือนกัน เราจะพยายามใช้ โครงร่างของชื่อที่ใช้ทั่วไป (generic name scheme) และหลังจากนั้น จะชี้ให้เห็นความแตกต่างในภาษาต่างๆ

ภาษาทุกภาษามี เซต ของ **<u>predefined types</u>** (ชนิดนิยามมาแล้ว) **ซึ่งชนิดอื่นๆ ทั้งหมด** จะนำเอามาประกอบเข้าด้วยกัน ชนิดเหล่านี้ โดยทั่วไปกำหนดโดยใช้ ไอเดนติไฟเออร์นิยามมาแล้ว เช่น **<u>integer</u>**, **<u>real</u>**, **<u>boolean</u>** และ **<u>char</u>**

บางครั้ง **ชนิดเลขนิยามมาแล้ว** (predefined numeric types) กับ **ความแม่นตรง ซึ่งกำหนด**ไว้ มีรวมอยู่ด้วย เช่น **<u>longreal</u>**, **<u>double</u>**, **<u>longint</u>**, **<u>short</u>**, **<u>shortint</u>** และอื่นๆ

บางภาษา มี รูปแบบแตกต่างกัน สำหรับ **ชนิด** real **ที่มีให้ใช้** เช่น **<u>fixed</u>** และ **<u>float</u>**

บางครั้ง **พิสัย** integer **พิเศษ มีให้ใช้ด้วยเช่นกัน เช่น** unsigned integer (**ในภาษา** C) และ cardinal (**ในภาษา** Modula-2)

Predefined types ชุดแรก ได้แก่ **<u>ชนิดอย่างง่าย</u>** (simple types) :

ชนิดนี้ ไม่มีโครงสร้างอื่นใด ยกเว้น **คณิตศาสตร์ดั้งเดิม หรือ โครงสร้างแบบลำดับ**

(types that have no other structure than their inherent arithmetic or sequential structure.)

ชนิดทั้งหมดที่กล่าวมานี้ เป็น **ชนิดอย่างง่าย** อย่างไรก็ตาม ยังมี **ชนิดอย่างง่าย ซึ่งไม่ได้**เป็น predefined ได้แก่ **<u>ชนิดแจงนับ</u>** (enumerated types) และ **<u>ชนิดพิสัยย่อย</u>** (subrange types)

ชนิดแจงนับ หมายถึง เซต ซึ่ง สมาชิก ของมัน ทุกตัวมีชื่อและเขียนรายชื่อไว้ ชัดแจ้ง

(Enumerated types are set whose elements are named and listed explicitly.)

ตัวอย่าง ภาษา Pascal

type colors = (red, blue, green);

ชนิดแจงนับ ไม่ใช่เป็นแค่เซตเท่านั้น แต่มันเป็นเซตแบบอันดับ (ordered sets) **หมายถึง ส มาชิกของมัน เรียงตามลำดับ ในอันดับซึ่งกำหนดให้ในการประกาศ ดังนั้นจึงมีการดำเนินการ แบบหลัง** (successor) **และ แบบก่อน** (predecessor) **ตัวอย่างข้างต้น succ(red) =** blue

แต่ pred(red) ไม่ทราบค่า (undefined)

(ภาษา Modula-2 **ใช้** INC **และ** DEC **เพื่อให้ได้ ผลลัพธ์เช่นเดียวกับ succ และ** pred)

ชนิดพิสัยย่อย หมายถึง เซตย่อยติดกัน ของ ชนิดอย่างง่าย กำหนดโดย สมาชิกตัวที่มีค่าเล็กที่สุด และสมาชิกตัวที่มีค่าใหญ่ที่สุด

(Subrauge types are **contiguous** subsets of simple types specified by giving a least and greatest element.)

ตัวอย่าง การประกาศ ของ Pascal

type **digit = 0 .. 9;**
byte = 0 . . 255;

หรือ การประกาศ ของ Modula-2 **(โปรดสังเกตความแตกต่างเรื่องวงเล็บ)**

TYPE digit = [0 . . 93;
byte = [0 . . 255];

ในการประกาศ ข้างต้นนี้ simple type **ซึ่ง พิสัยย่อย นำเอามา แต่ไม่ได้แจ้งไว้ : ชนิดฐาน** (base type) **ของ พิสัยย่อย จึงเป็นโดยนัย ในบางภาษานั้น** base type **ต้องกล่าวไว้ชัดแจ้ง เช่น การประกาศของ** Ada

ตัวอย่าง a

subtype digit is INTEGER range 0 . . 9;

หรือ เหมือนกับการประกาศของ Modula-2 ข้างล่างนี้

TYPE digit = INTEGER [0 . . 9];

ในกรณีที่ ไม่มี ชนิดฐาน ชัดแจ้ง ให้สมมติว่า ถูกกระทำโดยอัตโนมัติเกี่ยวกับ base type **และสิ่งนี้ อาจนำไปสู่ ความกำกวมและความเข้ากันไม่ได้ ให้ดูการอภิปราย ในหัวข้อ 6.6 เรื่อง ชนิดคาบเกี่ยวกันและโดยนัย** (on implicit and overlapping types)

ชนิดพิสัยย่อย โดยทั่วไป มีข้อจำกัด กับ ชนิดอย่างง่าย ซึ่งมีการดำเนินการแบบ successor

และแบบ predecessor อยู่ ชนิดเหล่านี้ เรียกว่า **ชนิดเชิงเลข (ordinal types)** เพราะว่ามี อันดับไม่ต่อเนื่อง อยู่บนเซต ทุกชนิดที่กล่าวมาแล้วนั้น เป็น ordinal ยกเว้น สิ่งที่เกี่ยวข้องกับ จำนวนจริง (real numbers)

จำนวนจริง เป็น การเรียงลำดับ (เช่น 3.98 < 3.99) แต่ไม่มีการดำเนินการ แบบ successor และ predecessor ดังนั้น การประกาศพิสัยย่อย ข้างล่างนี้

TYPE Unitinterval = [0.0 .. 1.0];

จึงผิด (illegal) ในภาษาส่วนใหญ่

ภาษาโปรแกรมโดยปกติ implement ชนิดอย่างง่าย ทำโดยใช้ชนิดที่มี ให้ใช้ในฮาร์ดแวร์นั้น ได้แก่ integers, unsigned integers และ reals

การ implement ในทางปฏิบัติ จัดสรรเนื้อที่ สองไบต์ หรือ สี่ไบต์ สำหรับ integers ในรูปแบบส่วนเติมเต็มของสอง จัดสรรสี่ไบต์ หรือแปดไบต์ สำหรับ real ในรูปแบบต่างๆ และหนึ่งไบต์ สำหรับ Boolean และ Character (Booleans ใช้เฉพาะบิต อันดับต่ำเท่านั้น : 0 = FALSE, 1 = TRUE) ชนิดแจงนับ บ่อยครั้ง ถูกแปลภายใน ให้เป็น unsigned integer และเก็บใน จำนวนต่ำสุดของไบต์ที่จำเป็น พิสัยย่อย อาจถูกจัดสรรให้เต็มเนื้อที่ ของ base type ของมัน หรือ ตัวแปลภาษา สามารถจัดสรร จำนวนเลขต่ำสุด ของไบต์ ที่จำเป็นสำหรับเก็บค่าทั้งหมด

6.3 ตัวสร้างชนิด (Type Constructors)

เนื่องจาก แบบชนิดข้อมูล เป็น เซต การดำเนินการบนเซต สามารถนำมาใช้ เพื่อสร้างชนิดใหม่ นอกเหนือไปจาก ชนิดที่มีอยู่แล้ว

การดำเนินการ เช่นนี้ ได้แก่ ผลคูณคาร์ทีเซียน (Cartesian product) ผลผนวก (union) เซตกำลัง (power set) เซตของฟังก์ชัน (functions) และเซตย่อย (subset)

เมื่อประยุกต์ใช้กับชนิด การดำเนินการบนเซตเหล่านี้ เรียกว่า **ตัวสร้างชนิด** (type constructs) ในภาษาโปรแกรม แบบชนิดข้อมูลทั้งหมด ถูกสร้างขึ้นจาก ชนิดอย่างง่ายโดยใช้ตัวสร้างชนิด ในหัวข้อที่แล้ว เราได้เห็น รูปแบบ จำกัด ของ หนึ่ง ใน ตัวสร้างเหล่านี้ ได้แก่ การสร้างเซตย่อย (subset construction) ใน ชนิดพิสัยย่อย (subrange types)

ยังมี ตัวสร้างชนิดอื่นๆ อีก ซึ่ง ไม่สมนัยกับ การสร้างเซตเชิงคณิตศาสตร์ สิ่งเหล่านี้ ได้แก่ ตัวชี้ (pointer) และชนิดแฟ้ม (file types) ในหัวข้อนี้ เราจะจำแนก และให้ตัวอย่างของตัวสร้างชนิดร่วม

6.3.1 **ผลคูณคาร์ทีเชียน** (Cartesian Product)

กำหนดเซตให้สองชุด ชื่อ U และ V เราสามารถสร้างผลคูณคาร์ทีเชียน หรือ **เซตผลคูณ** (cross product) ซึ่งประกอบด้วย คู่อันดับทั้งหมด ของสมาชิก จาก U และ V ดังนี้

U x V = {(u,v) I u is in U and v is in V}

ผลคูณคาร์ทีเชียน มาพร้อมกับ ฟังก์ชันการฉาย (projection functions)

p_1 : ux v → u

และ p_2 : u x v → v

เมื่อ $p_1((u,v))$ = u และ $p_2((u,v))$ = v

การสร้างนี้ สามารถขยายไปมากกว่าสองเซตได้

ดังนั้น

u x v x w = {(u,v,w) I u in u, v in v, w in W}

มีฟังก์ชันการฉายจำนวนมากมาย เทียบเท่ากับ จำนวนสมาชิก

ในหลายๆ ภาษา มี ตัวสร้างชนิด ผลคูณคาร์ทีเชียน ให้ใช้ได้ เช่น การสร้าง<u>ระเบียน</u> (record construction)

<u>**ตัวอย่าง**</u> การประกาศในภาษา Modula-2

```
TYPE IntBoolReal = RECORD
                     i : INTEGER;
                     b : BOOLEAN;
                     r : REAL;
                   END;
```

สร้าง ชนิดผลคูณคาร์ทีเชียน INTEGER x BOOLEAN x REAL

อย่างไรก็ตาม มีข้อแตกต่างระหว่าง ผลคูณคาร์ทีเชียน และระเบียนดังนี้ เขตข้อมูลมีชื่อในระเบียน ในขณะที่ในผลคูณ เขตข้อมูลถูกอ้างถึงโดยตำแหน่ง (the fields have names in a record, while in a product they are referred to by position.)

การฉาย (projections) ในระเบียน ถูกกำหนดโดย <u>field selector operation</u> : ถ้า x เป็นตัวแปร ชนิด IntBoolReal แล้ว x.i คือการฉายของ x ไปยัง integers

ผู้เขียนหนังสือบางคน พิจารณาว่า ชนิดระเบียน แตกต่างจากชนิดผลคูณคาร์ทีเชียน จริงๆ แล้ว ภาษาส่วนใหญ่ ถือว่า ชื่อเขตข้อมูล (field names) เป็นส่วนหนึ่งของชนิดซึ่งนิยามโดย

ระเบียน

ดังนั้น

```
RECORD
    i : INTEGER;
    c : BOOLEAN;
    s : REAL;
END;
```

ถือว่าแตกต่าง จากระเบียนเพียงแค่การนิยามเท่านั้น ทั้งๆ ที่มันแทนเซตของผลคูณคาร์ทีเซียนเหมือนกัน

โครงร่างการจัดสรรเนื้อที่ โดยปกติ ของ ชนิดผลคูณ คือ การจัดสรรแบบลำดับ ขึ้นอยู่กับเนื้อที่ซึ่งสมาชิกแต่ละตัวจำเป็นต้องใช้ ดังนั้น ตัวแปรชนิด IntBoolReal จึงต้องจัดสรรให้เท่ากับเจ็ดไบต์ : สองไบต์แรก สำหับ INTEGER ไบต์ที่สาม สำหรับ BOOLEAN และสี่ไบต์สุดท้าย สำหรับ REAL

6.3.2 <u>ผลผนวก</u> (Union)

การสร้างชนิดข้อมูลแบบที่สองคือ ผลผนวกของสองชนิด : ประกอบขึ้นจากการนำผลผนวกเชิงทฤษฎีเซต ของ เซตของค่าต่างๆ ชนิดของผลผนวก แบ่งออกเป็นสองประเภท คือ discriminated unions และ undiscriminated unions

ผลผนวก จะเป็น แบบ **[illegible]d** ถ้ามีการใส่ **<u>tag</u>** หรือ **di[illegible]** ให้กับเขตข้อมูลของสมาชิกแต่ละตัว เพื่อแยกว่าสมาชิกตัวนั้นเป็นชนิดใด นั่นคือ มาจากเซตใด

ผลผนวกแบบ **[illegible]l** คล้ายกับ ผลผนวกของเซตต่างสมาชิก (disjoint unions) ในวิชาคณิตศาสตร์

ส่วนผลผนวกแบบ undiscriminated ไม่มี tag และต้องมีข้อสมมติเกี่ยวกับชนิด ของสมาชิกโดยเฉพาะ

ผลผนวกของภาษาเหมือน Algol บ่อยครั้ง สร้างขึ้น โดยใช้ **<u>variant records</u>**

ในภาษา Modula-2 ผลผนวกแบบ discriminated ของชนิด INTEGER และ REAL กำหนดโดยการประกาศ ดังนี้

```
TYPE IntOrReal = RECORD
                     CASE IsInt : BOOLEAN OF
                          TRUE : i : INTEGER I
                          FALSE : r : REAL
                     END;
                 END;
```

ในที่นี้ tag field ชื่อ IsInt จะบอกว่า สมาชิกตัวนั้น เป็น INTEGER หรือ เป็น REAL กำหนดให้ตัวแปร x ชนิด IntOrReal เราสามารถกำหนดค่า จำนวนเต็ม 0 ให้กับ x ดังนี้

```
x.IsInt := TRUE;
   x.i := 0;
```

ผลผนวกแบบ undiscriminated นิยามดังนี้

```
TYPE IntOrReal = RECORD
                     CASE BOOLEAN OF
                       TRUE : i : INTEGER I
                       FALSE : r : REAL
                     END;
                 END;
```

โปรดสังเกตว่า การประกาศชนิด BOOLEAN ในข้อความสั่ง CASE ยังคงต้องเขียนไว้ แต่ไม่ต้องมี tag หรือ ชื่อเขตข้อมูล

ผลผนวกในภาษา C กำหนดให้โดยตรง และเป็น undiscriminated เขียนดังนี้

```
typedef union
   {int i;
    float r;} utype;
```

นิยามว่า utype เป็นผลผนวกของ reals และ integers ซึ่งอาจถูกเข้าถึง โดยใช้ สัญกรณ์ dot (.) เหมือนกับ ภาษา Modula-2 หรือ Pascal

Algol68 มีการประกาศคล้ายกันคือ :

```
ir = union(int, real)
```

แต่ ชนิดของ object ถูกทดสอบระหว่าง การกระทำการเพื่อหลีกเลี่ยง การอ้างถึงที่ผิด (illegal references) (สิ่งนี้ หมายความว่า ลักษณะประจำของชนิด ต้องถูกเก็บไว้ระหว่างการดำเนินงาน และเป็นเรื่องสำคัญ)

ผลผนวก โดยทั่วไป จัดสรรเนื้อที่เท่ากับ เนื้อที่มากที่สุด ซึ่งจำเป็นสำหรับ แต่ละ variants และ variants ต่างๆ ถูกเก็บในพื้นที่ ของหน่วยความจำที่คาบเกี่ยวกัน ดังนั้น ตัวแปรชนิด IntOrReal (ไม่มี discriminant) ต้องใช้เนื้อที่จัดสรรสี่ไบต์ : สองไบต์แรก ใช้สำหรับ INTEGER variants และทั้งหมด สี่ไบต์ ใช้สำหรับ REAL variants ถ้า ใส่เขตข้อมูล tag ชนิด BOOLEAN เพิ่ม จะใช้เนื้อที่ทั้งหมด ห้าไบต์

6.3.3 <u>เซตย่อย</u> (Subset)

ในวิชาคณิตศาสตร์ เซตย่อย สามารถกำหนดได้ โดยการให้กฎเพื่อแยกความแตกต่าง สมาชิกของมัน เช่น

posint = {x | x an integer and x > 0}

กฎ ในทำนองเดียวกัน สามารถกำหนดให้ในภาษาโปรแกรม เพื่อสร้างชนิดใหม่ ซึ่งเป็นเซตย่อยของชนิดที่รู้จักแล้ว (known types)

โดยการกำหนดขอบเขตล่าง และขอบเขตบน พิสัยย่อยของชนิดเชิงอันดับที่ (ordinal type) สามารถประกาศได้ ในภาษา Pascal และ Modula-2 ภาษา Ada มีการสร้าง เซตย่อยของชนิดแถวลำดับ โดยระบุ พิสัยย่อย สำหรับ เซตของดรรชนี (index set)

<u>ตัวอย่าง</u> รหัสภาษา Ada

```
type  digit is range  0 .. 9;
type  ar1 is array(digit) of INTEGER;
subtype ar2 is ar1 (2 .. 5);
```

หมายถึง แถวลำดับของชนิด ar2 มีดรรชนี จำกัดที่พิสัย 2 ถึง 5 ชนิดย่อยของชนิดแถวลำดับ เช่นนี้ เรียกว่า **<u>slices</u>**

ส่วนที่เป็น variant ของระเบียน สามารถกำหนดได้ในวิธีนี้เช่นกัน ตัวอย่างเช่น การประกาศของ IntOrReal ของภาษา Ada สามารถเขียนได้ดังนี้ (โปรดสังเกต การแทนที่ของ discriminant ใน ชื่อของชนิด)

```
type IntOrReal (IsInt BOOLAN) is record
                CASE ISInt is
                    when TRUE => i : INTEGER
                    when FALSE => r : REAL;
                end case;
        end record;
```

ขณะนี้ ชนิดเซตย่อย สามารถ ประกาศ ซึ่ง คงที่ส่วนของ variant (จากนั้น ต้อง มี ค่า ซึ่ง กำหนดไว้) **ดังนี้**

```
subtype IRInt is IntOrReal (IsInt => TRUE);
subtype IRReal is IntOrReal (IsInt => FALSE);
```

ชนิดของเซตย่อยเช่นนี้ สืบทอด (inherit) การดำเนินการ จาก **ชนิดของ** parent **ของมัน ภาษาโปรแกรมส่วนใหญ่ ไม่มีวิธีใดๆ ซึ่ง ผู้ใช้สามารถกำหนด** การดำเนินการ **ซึ่งถูกสืบทอด และ การดำเนินการ ซึ่งไม่ถูกสืบทอดได้** แต่การดำเนินการ จะถูกสืบทอด **อย่างอัตโนมัติ หรือ ถูกสืบทอดโดยนัย ตัวอย่างเช่น** ภาษา Ada เซตย่อย สืบทอด การดำเนินการ**ทั้งหมด ของ** parent type **มันจะเป็นการดี ถ้าสามารถ ตัด** (exclude) การดำเนินการ **ซึ่งไม่มีสาระออกไปจาก ชนิดเซตย่อยได้ ตัวอย่างเช่น** การลบแบบเอกภาพ (unary minus) มีสาระเล็กน้อย (little sense) **สำหรับ ค่าของชนิด** digit **ในภาษาเชิงวัตถุ** (object-oriented languages) **นั้น** ยอมให้มีการควบคุมมากกว่า **บน** การดำเนินการเช่นนี้ **และ ชนิดย่อย** (subtype) และกลไกของการสืบทอด **ของ** ภาษาเชิงวัตถุ **มี ความซับซ้อน**มากกว่า และใช้ได้หลายอย่าง (versatile) มากกว่า กลไก **ซึ่งเรากำลังอภิปรายในที่นี้**

6.3.4 <u>**เซตกำลัง**</u> (Powerset)

เซตกำลัง หรือ เซตของเซตย่อยทั้งหมด หมายถึง ตัวสร้างชนิดร่วม อีกประเภทหนึ่ง

(The **powerset** or set of all subsets is another common type constructor.)

<u>**ตัวอย่าง**</u> **ภาษา** Modula-2

```
TYPE digit = [0 . . 9];
        digitSet = SET OF digit;
```

ในการสร้าง SET OF t **มีข้อจำกัด** บ่อยมาก **บน** t **ตัวอย่างเช่น ในภาษา** Modula-2 **และ** Pascal **กำหนดว่า** t **ต้องเป็นชนิด เชิงอันดับที่** (ordinal type) **เท่านั้น และ ตัวแปลภาษาบาง**

ตัว มีข้อจำกัดเพิ่ม บน ขนาดของ t ตัวอย่างเช่น SET OF INTEGER บ่อยครั้ง ไม่ถูกยอมรับ เพราะว่า ขนาดของ ตัวแปรของ ชนิดนี้ ค่อนข้างใหญ่มาก

ปกติ เซต ถูกทำให้เกิดผล ในลักษณะ เวกเตอร์ของบิต (bit vectors) : สมาชิกที่เป็นไปได้ แต่ละตัว อาจจะมีหรือไม่มี ขึ้นอยู่กับว่า บิตที่สมนัยของมัน เป็น 1 หรือ 0 ดังนั้น จำนวนไบต์ ซึ่งต้องการ เพื่อจัดสรรเนื้อที่ ให้กับ ตัวแปรของชนิด SET OF t หมายถึง จำนวนนับ ของ t หาร ด้วย 8 ปัดเศษ ให้ ไบต์สูงกว่า หรือ ขอบเขตของคำ

(Thus the number of bytes required to allocate to a variable of type SET OF t is the cadinality of t divided by 8, rounded off to the nearest higher byte or word boundary.)

ตัวอย่างเช่น ตัวแปรชนิด SET OF CHAR ในภาษาหนึ่ง ใช้ ชุดอักขระ ASCII จำเป็นต้องใช้ 16 ไบต์ ของ หน่วยเก็บ เพราะว่า ชุดอักขระ ASCII มีอักขระ 128 ตัว

6.3.5 ฟังก์ชัน (Functions)

เซตของฟังก์ชันทั้งหมด $f : U \rightarrow V$ สามารถทำให้เกิด ชนิดใหม่ ในสองวิธีดังนี้

- ชนิดฟังก์ชัน (function type)
- ชนิดแถวลำดับ (array type)

เมื่อ U เป็น ชนิดเชิงอันดับที่ (ordinal type) ฟังก์ชัน f อาจคิดว่าเป็นแถวลำดับ ด้วย **index type** U และ **component type** V : ถ้า i อยู่ใน U แล้ว f(i) หมายถึง สมาชิกของแถวลำดับ และฟังก์ชัน ทั้งหมด อาจแทนด้วย ลำดับ หรือ ทูเปิล (tuple) ของค่าของมัน (f(low), ..., f(high)) เมื่อ low คือ สมาชิกตัวเล็กที่สุด ใน U และ high เป็นสมาชิกตัวใหญ่ที่สุด (ด้วยเหตุผลนี้ ชนิดแถวลำดับ บางครั้ง อ้างถึง **ชนิดลำดับ** (sequence types)

ตัวอย่าง การประกาศของภาษา Modula-2

```
TYPE  digit = [0 .. 9];
      digitToInt = ARRAY digit OF INTEGER;
```

แทน ฟังก์ชัน จาก digit ไปยัง INTEGER หรือ ทูเปิลแบบอันดับ ของ จำนวนเต็มสิบตัว

บ่อยครั้งที่ข้อจำกัด จะอยู่บน ขนาด หรือ description ของดรรชนีชนิดเชิงอันดับที่ (index ordinal type) ตัวอย่างเช่น ARRAY INTEGER OF INTEGER บางครั้งทำให้เกิด ข้อผิดพลาด ของการจัดสรร เนื้อที่ (allocation error) โดยเฉพาะบนคอมพิวเตอร์เครื่องเล็ก

ในบางภาษา พิสัยดรรชนี (index range) หรือ แม้กระทั่ง เซตดรรชนี (index set) อาจจะ

ไม่ต้องกำหนดได้ ในภาษา Modula-2 นั้น open-index array type สามารถนำมาใช้ ในการประกาศ พารามิเตอร์ ของ โปรซีเดอร์ และ พารามิเตอร์ของฟังก์ชัน ได้ดังนี้

ตัวอย่าง

```
PROCEDURE FindLargest (a : ARRAY OF INTEGER) : INTEGER;
VAR i : CARDINAL;
    max : INTEGER
BEGIN
    max := a[O]
    FOR i := 1 TO HIGH(a) DO
        IF a[i] > max THEN
            max := a[i];
        END; (* IF *)
    END;        (* FOR *)
    RETURN max;
END FindLarges;
```

ในที่นี้ หมายถึง การประกาศฟังก์ชัน ซึ่ง เอา แถวลำดับของ integers เป็นพารามิเตอร์ โดยไม่กำหนด เซตของครรชนี

ฟังก์ชันนี้ ใช้ predefined ฟังก์ชัน HIGH เพื่อหา ขอบเขตบนของแถวลำดับ ส่วนขอบเขตล่าง ปกติ จะเป็น 0

ถ้า ชนิดของครรชนีจริง (actual index type) ไม่ได้อยู่ใน พิสัยนี้ มันจะถูกแปลงส่ง (mapped) ไปยังตัวมัน (any ordinal type)

ตัวอย่าง การประกาศ ใน Ada

```
type IntToInt is array (INTEGER range <>) of INTEGER;
```

ในที่นี้ หมายถึง สร้าง ชนิดแถวลำดับ (array type) จากพิสัยย่อย ของ integers ไปยัง integers เครื่องหมาย "<>" แสดงว่า ไม่มีการกำหนด พิสัยย่อย ชนิดเช่นนี้ สามารถนำมาใช้ สำหรับพารามิเตอร์ ให้กับ โปรซีเดอร์ เหมือนกับ ใน Modula-2 อย่างไรก็ตาม เมื่อประกาศ ตัวแปร จะต้องกำหนดพิสัย ไว้ดังนี้

```
table : IntToInt (-10 . . 10);
```

แถวลำดับหลายมิติ (Multidimensional arrays) อาจเป็นไปได้เช่นกัน เช่นใน รหัสภาษา Modula-2 ข้างล่างนี้

TYPE IntMatrix = ARRAY digit, digit of INTEGER;

สิ่งนี้ หมายถึง ฟังก์ชันจาก (digit x digit) ไปยัง INTEGER

แถวลำดับอาจจะเป็น ตัวสร้างชนิด ซึ่งใช้กันแพร่หลายมากที่สุด (Arrays are perhaps the most widely used type constructor.) เพราะว่าการทำให้เกิดผลของมัน กระทำได้ อย่างมีประสิทธิภาพมาก : เนื้อที่ถูกจัดสรร แบบลำดับ ในหน่วยความจำ และ indexing กระทำโดย offset calculation จาก เลขที่อยู่เริ่มต้น (starting address) ของแถวลำดับ

ในกรณีของ แถวลำดับหลายมิติ การจัดสรรเนื้อที่ ยังคงเป็นเชิงเส้น และต้องตัดสินใจว่าจะใช้ ดรรชนีอันไหน เป็นอันดับแรก ในโครงร่างการจัดสรร (allocation scheme) :

ถ้า x เป็นชนิด IntMatrix แล้ว x ถูกเก็บไว้ดังนี้

x[0,0], x[0,1], x[0,2], . . ., x[0,9],

x[1,0], x[1,1], x[1,2], . . ., x[1,9],

. . .

x[9,0], x[9,1], x[9,2], . . ., x[9,9],

x[10,0], x[10,1], x[10,2], . . ., x[10,9]

. . .

เรียกว่า **row-major form**

หรือ เก็บดังนี้

x[0,0], x[1,0], x[2,0], . . ., x[9,0]

x[0,1], x[1,1], x[2,1], . . ., x[9,1]

เรียกว่า **column-major form**

ชนิดฟังก์ชัน และ ชนิดโปรซีเดอร์ โดยทั่วไป สร้างขึ้นได้เช่นกัน ในบางภาษา ตัวอย่างเช่น ในบทนิยาม ของภาษา Modula-2

TYPE intFunction = PROCEDURE (INTEGER) : INTEGER;

นิยาม ชนิดของฟังก์ชัน (function type) จาก integers ไป integers

(ภาษา Modula-2 คล้ายกับ Algol60 กล่าวคือ คำสงวน PROCEDURE ใช้ทั้ง ฟังก์ชัน และโปรซีเดอร์)

ภาษา Pascal นั้น ชนิดฟังก์ชันและชนิดโปรซีเดอร์ มีอยู่เฉพาะในพารามิเตอร์ ไป ฟังก์ชัน

อื่น หรือ โปรซีเดอร์อื่น และในคอมไพเลอร์บางตัว สิ่งเหล่านี้ ทำให้เกิดผลไม่ได้

ตัวอย่าง การประกาศ

```
procedure p(function f(integer) : integer, i : integer);
begin
    if f(i+l) = 0 then
    ...
end;
```

มีพารามิเตอร์ ตัวแรก เป็น ฟังก์ชัน จาก integers ไปยัง integers

รูปแบบ และ การจัดสรรเนื้อที่ ของ ตัวแปรชนิดฟังก์ชัน (function variables) ขึ้นอยู่กับ ขนาด ของ เลขที่อยู่ ซึ่งจำเป็น เพื่อชี้ ไปยัง รหัส ซึ่งแทน ฟังก์ชัน และบนสิ่งแวดล้อมเวลาดำเนินงาน ซึ่งต้องใช้โดย ภาษานั้น รายละเอียดเพิ่มเติม ให้ดูบทที่ 7

6.3.6 ชนิดตัวชี้และชนิดการเรียกซ้ำ

(Pointers and Recursive Types)

ตัวสร้างชนิด ซึ่ง ไม่สมนัยกับ การดำเนินการบนเซต ได้แก่ ตัวสร้าง **reference** หรือ **pointer** ซึ่งสร้างเซตของเลขที่อยู่ทั้งหมด ของชนิดที่กำหนดให้

ตัวอย่างเช่น การประกาศ ในภาษา Modula-2

```
TYPE InPtr = POINTER TO INTEGER;
```

สร้าง ชนิดของเลขที่อยู่ทั้งหมด ของ integers ถ้า x เป็นตัวแปร ชนิด InPtr ดังนั้น การ **dereferenced** x จะได้ค่าของชนิด integer :

```
x^ := 10
```

คือการกำหนดค่าจำนวนเต็ม 10 ให้กับตำแหน่ง ซึ่งกำหนดโดย x (assigns the integer value 10 to the location given by x.)

ทั้งนี้ x ต้องมีการกำหนดเลขที่อยู่ถูกต้อง มาก่อนแล้ว สิ่งนี้ ปกติทำให้สำเร็จอย่างพลวัต โดยเรียก ฟังก์ชันการจัดสรรเนื้อที่ NEW(x) (ตัวแปรชนิดตัวชี้ ได้อภิปรายแล้ว ในบทที่ 5)

การประกาศ เช่นเดียวกัน ในภาษา Pascal เขียนดังนี้

```
type IntPtr = ^integer;
```

เมื่อใช้สัญลักษณ์ "^" ทั้งการ reference และการ dereference อาจทำให้ สับสนได้เล็ก

น้อย (ฝ่าฝืน หลักของ การเป็นรูปแบบเดียวกัน (uniformity))

ภาษา C ปัญหานี้มีอยู่เช่นกัน เมื่อ เครื่องหมาย "*" ถูกนำมาใช้ แทนเครื่องหมาย "^" ของ Pascal ดังนั้น การประกาศตัวชี้ ไป integer ในภาษา C เขียนดังนี้

```
int *x;
```

และการกำหนดค่า integer ให้กับตำแหน่งซึ่งชี้โดย x เขียนดังนี้

```
*x = 10;
```

(โปรดสังเกตว่า การใช้เครื่องหมาย "*" อยู่บนด้านเดียวกัน ของ x ทั้งในการประกาศ และในการกำหนดค่า)

ตัวชี้ มีประโยชน์มากที่สุด ในการสร้าง **ชนิดการเรียกซ้ำ** (recursive types) : **ชนิดซึ่งใช้**ตัวมันเองในการประกาศของมัน (a type that uses itself in its declaration.)

ชนิดเรียกซ้ำ มีความสำคัญอย่างมากใน โครงสร้างข้อมูล และอัลกอริทึม เพราะว่า มันสมนัยกันอย่างธรรมชาติ กับ อัลกอริทึมการเรียกซ้ำ และแทนข้อมูลซึ่ง ขนาดและโครงสร้าง ไม่เป็นที่รู้จักในขั้นสูง แต่อาจจะมีการเปลี่ยนแปลง ขณะทำการคำนวณ ตัวอย่าง สองชนิด ได้แก่ รายการ (lists) และต้นไม้แบบทวิภาค (binary tree)

ตัวอย่าง จงพิจารณา การประกาศรายการของตัวอักขระ ใน ภาษาเหมือน Modula-2 ข้างล่างนี้

```
TYPE  charlist =`RECORD
                    data : CHAR;
                    next : charlist;
                 END;
```

ไม่มีเหตุผลในหลักเกณฑ์ว่าทำไม บทนิยามการเรียกซ้ำเช่นนี้ควรจะผิด ฟังก์ชันการเรียกซ้ำ มี โครงสร้างคล้ายกัน อย่างไรก็ตาม เมื่อพิจารณาอย่างใกล้ชิด จะเห็นว่า บทนิยามนี้ มีปัญหา กล่าวคือ ข้อมูลเช่นนี้ ต้องประกอบด้วย จำนวนของตัวอักขระ ไม่จำกัด ในทำนองเดียวกันกับฟังก์ชันการเรียกซ้ำ สิ่งนี้เหมือนกับ ฟังก์ชันการเรียกซ้ำ ซึ่งไม่มี "base case" นั่นคือ การทดสอบการหยุด ของการเรียกซ้ำ

ตัวอย่าง บทนิยามฟังก์ชัน

```
PROCEDURE  fact(n : INTEGER) : INTEGER;
BEGIN
    RETURN  fact(n-1) * n;
END;
```

ไม่มีการทดสอบ สำหรับ small n และผลลัพธ์ในจำนวนไม่จำกัดของการเรียก fact (อย่างน้อยที่สุด จนกระทั่ง หน่วยความจำหมด)

B.S.7 **ตัวสร้างชนิดอื่นๆ** (Others Type Constructors)

บางภาษามีตัวสร้างชนิดพิเศษเพิ่มขึ้นอีก ตัวอย่างเช่น *(**file** type) ในภาษา Pascal :

type **recfile** = file of employeerec;

ภาษา Pascal มี predeclared file type คือ text = file of char

ภาษา Modula-2 ไม่มีชนิดแฟ้มใดๆ files ถูกสมมติให้ขึ้นอยู่กับระบบ และชนิดของมันจะนำเข้า (imported) จาก **library** module

ภาษา Ada มีวิธีการเข้าถึง คล้ายกัน ยกเว้น รูปแบบ ของ file library ซึ่งกำหนดไว้แน่นอน

ชนิดซึ่งเพิ่มเติมอีกอย่างหนึ่ง ซึ่ง predefined ในบางภาษา ได้แก่ **string type** นั่นคือ ลำดับของตัวอักขระ ความยาวใดๆ ก็ได้ มีวิธีการเข้าถึงที่เป็นไปได้สองวิธีคือ : อาจจะมี single string type ไม่กำหนดความยาวสูงสุด หรือ สำหรับจำนวนเต็มบวกแต่ละตัว ในพิสัยบางอย่าง ซึ่ง string type สามารถถูกประกาศด้วย ความยาวสูงสุด

ตัวอย่างเช่น การประกาศของ PL/1

DCL A CHAR(80);

ในที่นี้ สร้าง (creates) ตัวแปร A ซึ่งเป็น สายอักขระ (string) ความยาว 80

ภาษา Pascal และ Modula-2 ทั้งคู่ หลีกเลี่ยง ข้อแนะนำ ของชนิดสายอักขระพิเศษ โดยกำหนดว่า strings หมายถึง แถวลำดับของตัวอักขระ ด้วย ชนิดของดรรชนี อย่างหนึ่ง

ชนิด string ของ Pascal ปกติจะมี ขอบเขตล่าง เป็น 1 ตัวอย่างเช่น array [1 .. n] of char

ส่วน สายอักขระ ของ Modula-2 ปกติจะมีขอบเขตล่างเป็น 0 ตัวอย่างเช่น ARRAY [0 .. n] OF CHAR

สิ่งนี้ แสดงทันทีให้ implementors เห็น เพื่อให้ เสนอแนะ ชนิด string ของตนเอง ทำให้เกิดความสับสน และการใช้แทนกันไม่ได้

Modula-2 มี กฎเฉพาะ สำหรับ จัดกระทำ (handling) ชนิดแถวลำดับ ของตัวอักขระ ทำ

ให้ภาพออกมาชัดเจนกว่า อย่างไรก็ตาม สายอักขระเช่นนี้ ยังคงมีข้อจำกัดที่ว่า ไม่มี การดำเนินการสายอักขระ โดยเฉพาะ เช่น การต่อกัน (concatenation) การแบ่งสายอักขระ (substring extraction) หรือ ความสามารถที่จะขยาย หรือ ลดอย่างพลวัต (to grow or shrink dynamically)

บางภาษา รวม การจัดสรรเนื้อที่ directive เป็น รูปแบบของ ชนิดตัวสร้าง ตัวอย่างเช่น <u>packed</u> directive ใน Pascal และ <u>align</u> directive ใน C

คำถามมีอยู่ว่า ในบทนิยามของภาษา ควรจะรวม directives เช่นนี้ ไว้ด้วยหรือไม่ เหตุผลหนึ่งคือ มันเป็นเหตุให้ ขึ้นอยู่กับการ implement อีกเหตุผลหนึ่งคือ ตัวแปลภาษาที่ดี ควรจะสามารถ optimize การจัดสรรเนื้อที่ โดยไม่จำเป็นต้องมี directives เช่นนั้น (Modula-2 ไม่มี *packed directive* เหมือน Pascal)

สุดท้าย มันจะเป็นประโยชน์ ถ้าสามารถกำหนด ชนิด "notype" : ชนิด ซึ่งแตกต่างจากชนิดอื่นๆ ทั้งหมด (a type that is distinct from other types.) ตัวอย่างเช่น ชนิด เช่น เซต ซึ่งประกอบด้วย ค่าหนึ่งค่า แตกต่างจากค่าอื่นๆ ทั้งหมด เช่นชนิด <u>void</u> ใน C หรือ Algol68 และชนิด <u>unit</u> ของ ML

6.4 ชนิด Nomenclature ในภาษาเหมือน Pascal

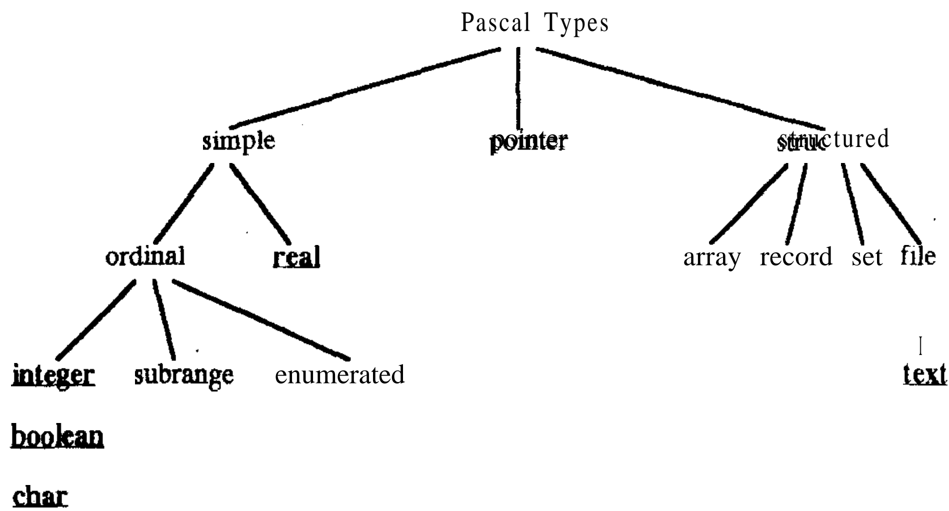
(Type Nomenclature in Pascal-like languages)

ถึงแม้ว่า เราจะได้นำเสนอ โครงร่างทั่วไปของกลไกชนิดต่างๆ ในหัวข้อ 6.2 และ 6.3 ไปแล้ว บทนิยามของภาษาหลากหลาย ใช้สัญลักษณ์ที่สับสน และแตกต่างกัน เพื่อนิยาม สิ่งที่เหมือนกัน

ในหัวข้อนี้ จะได้ให้ การอธิบายโดยย่อ ของ ความแตกต่าง ระหว่าง ภาษาเหมือน Algol หลายภาษา

6.4.1 Pascal/Modula-2

ภาพทั่วไปของชนิด Pascal กำหนดให้แล้ว ในรูป 6.1 ชนิดอย่างง่าย สมนัย อย่างใกล้ชิดกับ การอธิบายของเรา



รูป 6.1 โครงสร้างชนิด ของ Pascal (The Type Structure of Pascal)

ชนิด ซึ่งถูกสร้าง โดยใช้ตัวสร้าง แถวลำดับ ระเบียน เซต และ แฟ้ม เรียกว่า **ชนิดเชิงโครงสร้าง** (structured types) ชนิดเหล่านี้ แตกต่างจาก ชนิดอย่างง่าย และ ชนิดตัวชี้

ภาษา Modula-2 มีโครงสร้างชนิด คล้ายกัน ด้วยข้อยกเว้นเล็กน้อย มี CARDINAL ซึ่งเป็น predefined ordinal type แตกต่างจาก INTEGER

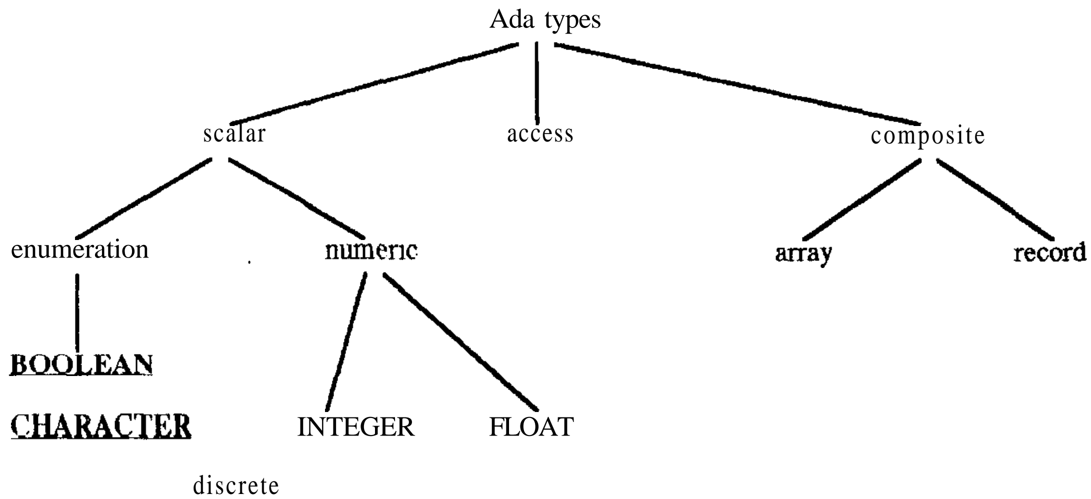
CARDINAL เริ่มต้นที่ 0 ไปจนถึง ค่าสูงสุด ไม่นิยามบางอย่างซึ่ง อาจมากกว่า INTEGER ค่ามากที่สุด (บนระบบจำนวนมาก INTEGER = [-32768 .. 32767] และ CARDINAL = [0 .. 65535])

Modula-2 มีชนิดโปรซีเดอร์ ซึ่งในภาษา Pascal ไม่มี แต่ ไม่มี ชนิดแฟ้ม กำหนดใน บทนิยาม ของ Modula-2 (files เป็นคุณสมบัติ ซึ่งขึ้นอยู่กับระบบ จัดหา แยกต่างหาก โดยการ implement แต่ละตัว)

ในทางตรงกันข้าม Pascal มี predefined type คือ text ซึ่งเป็น ชนิดโครงสร้าง และเหมือนกับ file of char

6.4.2 Ada

ภาษา Ada มีเซตของ ชนิด จำนวนมาก ภาพซึ่งย่อไว้ ในรูป 6-2 เป็นการย่อ



รูป 6-2 โครงสร้างชนิดของ Ada

ใน Ada ชนิดอย่างง่าย เรียกว่าชนิด **scalar** และแบ่งย่อยเป็น ประเภทต่างๆ ซึ่งคาบเกี่ยวกัน ชนิดเชิงอันดับที่ (ordinal types) เรียกว่า ชนิด **discrete** ส่วนชนิดเชิงเลข (numeric type) ประกอบด้วย ชนิด real และ integer ชนิดตัวชี้ เรียกว่า **access** types ส่วน ชนิดแถวลำดับและระเบียน เรียกว่า ชนิด **composite**

มี ชนิด files เช่นเดียวกับ Modula-2 ยกเว้น รูปแบบของ สิ่งอำนวยความสะดวก แฟ้มมาตรฐาน กำหนดให้ แน่นอนมากกว่า ใน Ada ไม่มี ชนิดโปรซีเดอร์ หรือ ชนิดเซต

6.5 ความสมมูลของชนิด (Type Equivalence)

คำถามสำคัญ เกี่ยวข้องในการประยุกต์ใช้ ของชนิด กับ การตรวจสอบชนิด คือ **ความสมมูลของชนิด** : เมื่อใดที่สองชนิดเหมือนกัน วิธีหนึ่งของความพยายามที่จะตอบคำถามนี้ คือ การเปรียบเทียบ เซตของค่าต่างๆ เช่นเดียวกับเรื่องเซต ในวิชาคณิตศาสตร์ เซตสองชุดจะเท่ากัน ถ้ามันประกอบด้วยค่าต่างๆ เหมือนกัน ตัวอย่างเช่น ชนิดใดๆ ก็ตาม ซึ่งนิยามเป็นผลคูณคาร์ทีเซียน A x B หมายถึง เหมือนกับ ชนิดอื่นๆ ซึ่งนิยาม ในวิธีเดียวกัน

ในทางตรงกันข้าม สมมติว่า ถ้า B ไม่ใช่ชนิดเดียวกับ A ดังนั้น ชนิดซึ่งนิยามเป็น A x B จะไม่เหมือนกับ ชนิดซึ่งนิยามเป็น B x A เพราะว่า A x B ประกอบด้วย คู่ (a, b) แต่ B x A ประกอบด้วยคู่ (b, a)

มองด้านนี้ ความสมมูลของชนิด คือชนิดข้อมูลสองอย่าง จะเหมือนกัน ถ้ามันมีโครงสร้างเหมือนกัน : มันสร้างขึ้น ในวิธีเดียวกัน โดยใช้ตัวสร้างชนิดเหมือนกัน จาก ชนิดอย่างง่ายเหมือนกัน รูปแบบนี้ ของความสมมูลของชนิด เรียกว่า **<u>ความสมมูลเชิงโครงสร้าง</u>** (structural equivalence) และ เป็นหนึ่ง ในรูปแบบหลัก ของความสมมูลของชนิด ในภาษาโปรแกรม

<u>ตัวอย่าง</u>

ในวากยสัมพันธ์ของภาษา เหมือน Pascal ชนิด rec1 และ rec2 ซึ่ง นิยามข้างล่างนี้ เท่ากันเชิงโครงสร้าง แต่ rec1 และ rec3 ไม่เท่ากัน (เขต boolean และ integers ถูกสงวนไว้ ในบทนิยาม ของ red) :

```
type
    range = 1 . . 10;
      ar = array [range] of boolean:
     rec1 = record
              x : boolean;
              y : integer;
              z : ar;
           end;
     rec2 = record
              x : boolean;
              y : integer;
              z : array [ 1 .. 10] of boolean:
           end:
      rec3 = record
              x : integer;
              y : boolean;
              z : ar
           end;
```

ความสมมูลเชิงโครงสร้าง ค่อนข้างง่าย ในการ implement (อย่างน้อยที่สุด ในเมื่อไม่มีชนิดเรียกซ้ำ ดูหัวข้อ 6.3.6) และจัดหาสารสนเทศทั้งหมด ซึ่งจำเป็น เพื่อกระทำการตรวจสอบ

ข้อผิดพลาด และ การจัดสรรหน่วยเก็บ ซึ่งใช้ในภาษา Algol60, Algol68, FORTRAN และ COBOL

ในการตรวจสอบ ความสมมูลเชิงโครงสร้าง ตัวแปลภาษา อาจแทนชนิดข้อมูล ด้วยรูปต้นไม้ และตรวจสอบความสมมูล อย่างเรียกซ้ำ บน ต้นไม้ส่วนย่อย

คำถามยังคงมีอยู่ ในการคำนวณว่า สารสนเทศมากเท่าไหร่ ซึ่งต้องรวมอยู่ในชนิด ภายใต้การประยุกต์ใช้ ของ ตัวสร้างชนิด

ตัวอย่าง สองชนิดข้างล่างนี้

t_1 = array [-1 .. 9] of integer;

t_2 = array [0 .. 10] of integer;

เป็นความสมมูลเชิงโครงสร้างหรือไม่

คำตอบ ใช่ ถ้าเราพิจารณาเฉพาะ ขนาด ของ เซตดรรชนี (index set) เท่านั้น

คำตอบ ไม่ใช่ ถ้าพิจารณาว่า เซตดรรชนี ต้อง **จับคู่กัน** (match)

คำถามที่คล้ายกัน เกี่ยวกับ ชื่อเขตข้อมูล ของ ระเบียน ถ้าระเบียนนั้น เป็นเพียง ผลคูณคาร์ทีเชียน ตัวอย่างเช่น สองระเบียนข้างล่างนี้

```
reca = record
          x : boolean:
          y : mteger;
       and:
```

และ

```
recb = record
          a : boolean:
          b : integer;
       end;
```

ควรจะเป็นความสมมูลเชิงโครงสร้าง อย่างไรก็ตาม ใน Algol68 ทั้งสองระเบียนนี้ไม่ใช่ เพราะว่า ระเบียนจะเท่ากันเชิงโครงสร้าง ต้องมีชื่อเขตข้อมูลเหมือนกัน

ในกรณีของ แถวลำดับแบบพลวัต ยังมีปัญหาต่อไปอีกว่า ขอบเขตของ เซตดรรชนี ยังไม่ทราบค่า (are not known) ดังนั้น จะเห็นชัดเจนว่า ยังรวมชนิดไม่ได้ จริงๆ แล้ว เราอาจต้องหลีกเลี่ยง การนับรวม ของ ชนิดดรรชนี เข้าไปใน ชนิดแถวลำดับ เพื่อให้ แถวลำดับของ

จำนวนเต็มทั้งหมด เท่ากัน
ดังนั้น

array [colors] of integer

และ

array [0 .. maxint] of integer

ควรจะเท่ากัน ใน Algol68 สิ่งนี้สำคัญ : มิติ และ ชนิดของสมาชิก ของแถวลำดับ เป็นส่วนหนึ่งของชนิดของมัน แต่ไม่รวมเซตของดรรชนี อย่างไรก็ตาม เซตของดรรชนี ถูกจำกัดให้เป็นจำนวนเต็ม

ภาษา Ada นั้น base type ของเซตของดรรชนี เป็นส่วนหนึ่งของ ชนิดแถวลำดับด้วย แต่ไม่รวมขอบเขตของดรรชนี
ดังนั้น ชนิด

array (INTEGER range <>) of INTEGER

หมายถึง แถวลำดับ ของ จำนวนเต็ม และมีดรรชนี เป็นจำนวนเต็ม

<u>ข้อที่สอง</u> และ อัลกอริทึมความสมมูลของชนิด ซึ่งเข้มงวดมาก เป็นสิ่งที่เป็นไปได้ เมื่อชนิด สามารถมีชื่อ ในการประกาศชนิด : ชนิดของชื่อ สองชื่อ จะเท่ากัน ก็ต่อเมื่อ มันมีชื่อเหมือนกัน เพราะว่า ชื่อหนึ่งชื่อ หมายถึง การประกาศชนิดหนึ่ง หนึ่งชนิดเท่านั้น สิ่งนี้เป็นเงื่อนไขที่แข็งกว่า ความสมมูลเชิงโครงสร้าง อัลกอริทึมความสมมูลของชนิดนี้ เรียกว่า **<u>ความสมมูลของชื่อ</u>** (name equivalence)

ความสมมูลของชื่อ มีให้ใช้ใน Ada แต่มีให้ใช้เป็นส่วนน้อยในภาษาโปรแกรมอื่นๆ

<u>ตัวอย่าง</u>

การประกาศของภาษาเหมือน Pascal ข้างล่างนี้

```
type  ar1 = array [1 .. 10) of  INTEGER;
      ar2 = ar1;
      age = integer;
```

ในที่นี้ ar1 และ ar2 เป็นความสมมูล เชิงโครงสร้าง แต่ ไม่ใช่ความสมมูล ของชื่อ ในทำนองเดียวกัน age และ integer เป็นความสมมูลเชิงโครงสร้าง แต่ไม่ใช่ความสมมูล ของชื่อ ภาษา Ada การประกาศเหล่านี้ จะเขียนดังนี้ (การกำหนด "new" บังคับให้เป็นความสมมูลของชื่อ)

```
type  ar1  is  array (1 .. 10) of  INTEGER ;
```

```
type ar2 is array (1 .. 10) of integer;
type age is new INTEGER;
```

ความสมมูลของชื่อ มีความกำกวม ของมัน เช่นกัน ตัวอย่างเช่น ในภาษา ซึ่งมีการประกาศ ชนิด ปกติยังคงเป็นไปได้ ที่จะใช้ชนิด ในการประกาศตัวแปร โดยไม่ต้องให้ชื่อ ตัวอย่างเช่น ในการประกาศตัวแปร ของ Pascal ข้างล่างนี้

```
var x : array [1 .. 10] of integer;
    y : array [1 .. 10] of integer;
```

(การประกาศในทำนองเดียวกันนี้ มีอยู่ใน Ada เช่นกัน) ชนิดแถวลำดับสองชุด ของ x และ y ถูกสร้างขึ้นโดยตรง และไม่มีการกำหนดชื่อให้ ทั้งสองชนิดนี้ มีความสมมูลของชื่อ หรือไม่ คำตอบมาตรฐานคือ **ไม่ใช่** ให้พิจารณาว่า ชนิดเช่นนี้ มีชื่อภายใน ซึ่งปกติแตกต่างกัน แต่การประกาศข้างล่างนี้

```
var x, y : array [1 .. 10] of integer;
```

ขณะนี้ x และ y อาจจะมี ชนิดเหมือนกัน หรือ ชนิดไม่เหมือนกัน ภายใต้ความสมมูลของชื่อ (ภาษา Ada แก้ไขสถานการณ์นี้ โดยกำหนดว่า การประกาศร่วมใดๆ ก็ตาม ของ ตัวแปร จะเท่ากันกับ การประกาศซึ่งแยกต่างหากจากกัน ดังนั้น x และ y จะไม่ใช่ ความสมมูล ของชนิดใน Ada)

ความสมมูลของชื่อ และ ความสมมูลเชิงโครงสร้าง เป็นสองเรื่องสำคัญ ของ อัลกอริทึมความสมมูลของชนิด อย่างไรก็ตาม เรามักจะไม่ค่อยเห็นในรูปแบบบริสุทธิ : โปรแกรมเมอร์และนักออกแบบภาษา พบว่า มันจะเป็นประโยชน์ที่จะนิยาม จำนวนของสถานการณ์ เมื่อ ข้อยกเว้น ถูกกระทำขึ้น สิ่งเหล่านี้ จะศึกษา ในรายละเอียดมากขึ้นในหัวข้อถัดไป ซึ่งอธิบายเรื่อง การตรวจสอบชนิด และเรามีข้อสังเกตว่า ถึงแม้ว่า ปกติบทนิยามจะ ไม่ชัดเจน (clear-cut) ใน ทุกสถานการณ์ และ การละเว้นได้ หลากหลาย ยังมีให้ใช้ได้ กับ นักออกแบบภาษา

อัลกอริทึมความสมมูลของชนิด ที่สำคัญ ซึ่งอยู่ระหว่าง ความสมมูลของชื่อ และความสมมูลเชิงโครงสร้าง ได้แก่ **ความสมมูลของการประกาศ** (declaration equivalence) ซึ่งใช้ในภาษา Pascal และ Modula-2 ใน อัลกอริทึมนี้ ชื่อของชนิด ซึ่งนำกลับไปยัง การประกาศโครงสร้างดังเดิมเหมือนกัน โดยชุดของการประกาศใหม่ (redeclarations) ถูกพิจารณาว่า เป็นชนิดของความเท่ากัน

ตัวอย่าง การประกาศ

```
type
    t1 = array [1 .. 10] of integer;
    t2 = t1;
    t3 = t2;
```

ในที่นี้ t_1, t_2 และ t_3 ทั้งหมดนี้ เป็นความสมมูลของการประกาศ แต่ไม่ใช่ความสมมูลของชื่อ ในทำนองเดียวกัน ในตัวอย่างก่อนหน้านี้ ar1 และ ar2 เท่ากันในการประกาศ เช่นเดียวกับ age และ integer ซึ่งเท่ากันในการประกาศ

ตัวอย่าง การประกาศ

```
type  ar3 = array [1 .. 10] of integer;
      ar4 = array [1 .. 10] of integer;
```

ในที่นี้ ar3 และ ar4 ไม่ใช่ความสมมูลของการประกาศ เพราะว่า แต่ละชุด สร้างแยกต่างหากจากกัน ในการประกาศของมันเอง

วิธีง่ายๆ ของการมอง ความสมมูลของการประกาศ คือ ทุกครั้งที่มีการประยุกต์ใช้ ตัวสร้างชนิด ชื่อชนิดภายในตัวใหม่ จะถูกสร้างขึ้น ซึ่งโปรแกรมเมอร์ อาจจะให้ หรือ อาจจะไม่ หรือ อาจจะไม่ให้ ชื่อชัดแจ้ง ก็ได้

ตัวอย่าง การประกาศของ Modula-2

```
TYPE  t1 = ARRAY [1 .. 10] OF INTEGER;
      t2 = t1;
      t3 = ARRAY [1 . . 10] OF INTEGER;
VAR  x : t1;
      y : t2;
      z : t3;
      W : ARRAY [1 .. 10] OF INTEGER;
```

ในที่นี้ มี ชนิดแตกต่างกัน 3 อย่าง มีโครงสร้างเป็น ARRAY [1 .. 10] OF INTEGER; **ชื่อ** t1, t3 และ **ชนิดไม่มีชื่อ** ของ w (ชนิด t2 เหมือนกับ t1 ภายใต้ ความสมมูลของการประกาศ) จริงๆ แล้ว w อาจจะไม่เป็นการสมมูลของชนิด กับ ตัวแปรใดๆ เพราะว่า ชนิดของมัน ไม่ได้กำหนดชื่อไว้ และ ไม่สามารถอ้างถึงได้อีก ในทำนองเดียวกัน การประกาศ ข้างล่างนี้

VAR x, y : ARRAY [1 .. 10] OF INTEGER;

x และ y เป็นตัวแปรที่มีความสมมูล ของการประกาศ แต่ ไม่ใช่การสมมูล กับ ตัวแปรอื่นใด

ภาษา C ใช้การผสมกันของการสมมูลเชิงโครงสร้าง และการสมมูลของการประกาศ กล่าวคือ ความสมมูลของการประกาศ สำหรับโครงสร้าง และผลผนวก ส่วนความสมมูลเชิงโครงสร้าง สำหรับตัวชี้ และแถวลำดับ

6.6 <u>การตรวจสอบชนิด</u> (Type checking)

การตรวจสอบชนิด หมายถึง กระบวนการซึ่ง ตัวแปลภาษา ตรวจสอบว่า ตัวสร้างทั้งหมด ใน โปรแกรม มีความหมายในเทอม ของ ชนิด ของ ตัวคงที่ของมัน ตัวแปร กระบวนงาน และเอนทิตีอื่นๆ

(Type checking is the process a translator goes through to verify that all constructs in a program make sense in terms of the types of its constants, variables, procedures, and other entities.)

ซึ่งเกี่ยวกับ การประยุกต์ ของ อัลกอริทึมการสมมูล ของชนิด กับ นิพจน์และข้อความสั่ง ซึ่งแปรผันจาก ความเข้มงวดไปยัง การประยุกต์ชนิด ใช้งานได้ (permissive)

การตรวจสอบชนิด แบ่งออกเป็น สองประเภท คือ **<u>การตรวจสอบแบบพลวัต</u>** (dynamic checking) และ**<u>การตรวจสอบแบบคงที่</u>** (static checking) ถ้า ชนิดของสารสนเทศ เก็บไว้และตรวจสอบ ณ เวลาดำเนินการ การตรวจสอบนี้ เป็นแบบพลวัต ตัวแปลคำสั่ง (interpreters) โดยบทนิยาม กระทำการตรวจสอบ ชนิดแบบพลวัต แต่ คอมไพเลอร์ (compilers) สร้างรหัส ซึ่งเก็บลักษณะประจำของชนิด ระหว่างเวลาดำเนินงาน ในตาราง หรือ เป็น tags ชนิด ในสิ่งแวดล้อม ตัวอย่างเช่น คอมไพเลอร์ ภาษา LISP การตรวจสอบชนิดแบบพลวัต จำเป็นต้องใช้ เมื่อชนิดของวัตถุ หาได้เฉพาะ ณ เวลาดำเนินงานเท่านั้น

อีกทางเลือกหนึ่ง คือ ชนิดคงที่ (static typing) : วิธีนี้ ชนิดของนิพจน์ละวัตถุ หาได้จากบริบท (text) ของโปรแกรม และการตรวจสอบชนิด กระทำโดยตัวแปลภาษา ก่อนการกระทำการ

ภาษาชนิด strongly typed ข้อผิดพลาดของชนิดทั้งหมด ต้องถูกจับได้ก่อนเวลาดำเนินงาน ดังนั้นภาษาเหล่านี้ ต้องเป็นชนิดคงที่ อย่างไรก็ตาม บทนิยามของภาษา อาจไม่ได้กำหนดว่าให้ใช้ชนิดพลวัต หรือ ชนิดคงที่

ตัวอย่าง 1

ใน Pascal มาตรฐาน มีความแตกต่างระหว่างข้อผิดพลาด ซึ่งตัวแปลภาษาต้องตรวจพบ (เรียกว่า **violations**) และข้อผิดพลาด ซึ่งอาจจะตรวจพบได้ เฉพาะบนการกระทำการ (เรียกว่า **errors**) ตัวอย่างของ type error ซึ่งเป็น "errors" ไม่ใช่ "violation" ได้แก่ การเข้าถึง เขตระเบียนของ variant ซึ่งยังไม่ได้ใช้งาน (not active) และ การครรชนี แถวลำดับ ซึ่ง ค่านั้นไม่ได้อยู่ในพิสัยของครรชนี

เนื่องจาก ข้อผิดพลาดของชนิด ส่วนใหญ่ เป็น "violations" ไม่ใช่ "errors" ดังนั้นการตรวจสอบชนิดแบบคงที่ จึงจำเป็นต้องใช้โดยมาตรฐาน

ตัวอย่าง 2

คอมไพเลอร์ ของภาษา C ใช้การตรวจสอบชนิดแบบคงที่ ระหว่างการแปลโปรแกรม แต่ภาษา C ไม่ใช่ strongly typed ที่แท้จริง เพราะว่าความไม่พ้องกันของชนิด จำนวนมาก ไม่ได้เกิดข้อผิดพลาด ณ เวลาแปลโปรแกรม แต่ ถูกตัดออกอย่างอัตโนมัติ โดยคอมไพเลอร์ คอมไพ-เลอร์ สมัยใหม่ส่วนใหญ่ มีการตั้ง ระดับของข้อผิดพลาด ซึ่ง จัดให้เป็นชนิดเข้มงวดได้ ถ้าต้องการ

ตัวอย่าง 3 ข้อปลีกย่อย โครงร่างของภาษา LISP เป็นภาษาชนิด weakly, dynamically typed ไม่มีชนิดใดๆ ในการประกาศ ตัวแปรและสัญลักษณ์อื่นๆ ไม่มี predeclared type แต่ อยู่บนชนิดของค่า ซึ่งมันครอบครอง ณ แต่ละขณะ ของการกระทำการ ดังนั้น ชนิด ในโครงร่าง ต้องเก็บเป็นลักษณะประจำชัดแจ้ง ของค่า

การตรวจสอบชนิดภายใน คือการจำกัดให้ generating errors สำหรับฟังก์ชัน ต้องการค่าที่แน่นอน เพื่อ กระทำการดำเนินการของมัน ตัวอย่างเช่น car และ cdr ต้องมี ตัวถูกดำเนินการของมัน ให้กับ lists : (car2) จะมีข้อผิดพลาด

อย่างไรก็ตาม ชนิดสามารถถูกตรวจสอบอย่างชัดแจ้ง โดย โปรแกรมเมอร์ โดยใช้ predefined test functions ชนิดในโครงร่าง ได้แก่ รายการ สัญลักษณ์ อะตอม และ เลข (Types in Scheme include lists, symbols, atoms and numbers.)

Predefined test functions ได้แก่ atom? number'? และ symbol? (ฟังก์ชันทดสอบเช่นนี้เรียกว่า predicates และปกติ จบด้วยเครื่องหมายคำถาม)

ส่วนสำคัญ ของการตรวจสอบชนิด ได้แก่ **การอนุมานของชนิด** (type inference) **ซึ่ง** ชนิด ของนิพจน์ อนุมานจาก ชนิดของนิพจน์ย่อยของมัน กฎการตรวจสอบของชนิด (นั่นคือ เมื่อสร้างเป็นชนิดถูกต้อง) และกฎการอนุมาน ชนิด บ่อยครั้ง เกี่ยวข้องกัน (intermingled) ตัวอย่างเช่น นิพจน์ e1 + e2 ควรจะเป็น การประกาศชนิดถูกต้อง ถ้า e1 และ e2 มีชนิดเหมือนกัน และชนิดนั้น มี การดำเนินการ "+" (การตรวจสอบชนิด) และชนิดผลลัพธ์ ของนิพจน์ คือชนิดของ e1 และ e2 (การอนุมานชนิด)

ใน Pascal สิ่งนี้หมายความว่า e1 และ e2 อาจจะเป็น เซตของชนิดเดียวกัน หรือ เป็นชนิด integer หรือ real หรือ พิสัยย่อย ของ integer ดังนั้น ชนิดผลลัพธ์ เป็นชนิด set (ถ้าทั้งคู่เป็น sets) ชนิดผลลัพธ์ จะเป็น real ถ้าอันใดอันหนึ่งเป็น real และอีกอันหนึ่งเป็น integer

อีกตัวอย่างหนึ่ง ใน function call ชนิดของ พารามิเตอร์จริง หรือ อาร์กิวเมนต์ ต้องจับคู่แบบรูป (match) กับชนิด ของ พารามิเตอร์ทางการ (การตรวจสอบชนิด) และชนิดผลลัพธ์ของการเรียก คือ ชนิดผลลัพธ์ ของ ฟังก์ชัน (การอนุมานชนิด)

กฎการตรวจสอบชนิด และการอนุมานชนิด มีความกระทำได้ตอบ ใกล้ชิด กับ อัลกอริทึมความสมมูลของชนิด ตัวอย่างเช่น การประกาศของภาษา Modula-2 ข้างล่างนี้

```
PROCEDURE  p(ar : ARRAY [1 .. max] OF INTEGER);
```

เป็น ข้อผิดพลาด ภายใต้ ความสมมูลของการประกาศ เพราะว่า ไม่มีพารามิเตอร์จริง ที่สามารถมี ชนิด ของ พารามิเตอร์ทางการ ar ดังนั้น ชนิดการจับคู่ ไม่เข้ากัน จะต้องประกาศ บนการเรียก p ผลก็คือ วากยสัมพันธ์ ของ Modula-2 มีข้อจำกัดการประกาศ พารามิเตอร์ ให้กับ ชื่อชนิด (type names) ไม่ใช่ ชนิดทั่วไป ซึ่งกำหนดว่า รวมอยู่ในตัวสร้างชนิด ดังนั้น ต้องเขียน ดังนี้

```
TYPE  artype = ARRAY [1 .. max] OF INTEGER);
        ...
PROCEDURE  p(ar : artype);
```

สถานการณ์ ซึ่งเกิดขึ้น ใน Pascal และ Ada ก็เป็นเช่นเดียวกัน

กระบวนการ ของ การอนุมานชนิด และการตรวจสอบชนิด ใน ภาษาชนิดคงที่ (statically typed languages) ถูกช่วยเหลือโดย การประกาศชัดแจ้ง ของ ชนิดของตัวแปร ฟังก์ชัน และวัตถุอื่นๆ ตัวอย่างเช่น ถ้า x และ y เป็นตัวแปร ความถูกต้องและชนิดของนิพจน์ x + y เป็นการยากที่จะบอก ก่อน การกระทำการ ยกเว้น ชนิดของ x และ y ซึ่งได้มีการกล่าวไว้ชัดแจ้ง ในการประกาศ

อย่างไรก็ตาม การประกาศชนิดชัดแจ้ง ไม่ใช่สิ่งต้องการอย่างสมบูรณ์ สำหรับ static typing : ภาษา ML และ Miranda กระทำการตรวจสอบชนิดแบบคงที่ แต่ชนิดนั้น ไม่จำเป็นต้องถูกประกาศ ทั้งนี้ เพราะว่า ชนิด ถูกอนุมาน ได้จาก บริบท (context) โดยใช้กลไกการอนุมาน ซึ่ง powerful มากกว่า สิ่งที่เราได้อธิบายมาแล้ว

กฎการอนุมานเชิงชนิด และความถูกต้อง บ่อยครั้ง เป็นหนึ่งใน ส่วนซับซ้อน มากที่สุด ของ อรรถศาสตร์ของภาษา Nonorthogonalities เป็นสิ่งยากที่จะหลีกเลี่ยง ใน **ภาษาเชิงคำสั่ง** (imperative languages) เช่น Modula-2 และ Ada ในส่วนที่เหลือ ของ หัวข้อนี้ เราจะอภิปรายเรื่องสำคัญ และปัญหา ใน กฎ ของ type system

6.6.1 <u>ความเข้ากันได้ของชนิด</u> (Type Compatibility)

บางครั้ง มันจะเป็นประโยชน์ ในการ relax กฎความถูกต้องของชนิด เพื่อให้ ชนิดของส่วนประกอบ (components) ไม่จำเป็นต้อง เหมือนกันทีเดียว ตาม อัลกอริทึมความสมมูลของชนิด ตัวอย่างเช่น นิพจน์ e1 + e2 ยังคงมีความหมาย ถึงแม้ว่า ชนิด ของ e1 และ e2 เป็น พิสัยย่อย ของ integer ที่แตกต่างกัน ในสถานการณ์เช่นนี้ สองชนิดที่แตกต่างกัน ซึ่งยังคงถูกต้อง เมื่อรวมเข้าด้วยกัน ด้วย วิธีต่างๆ บ่อยครั้ง เรียกว่า **<u>ชนิดเข้ากันได้</u>** (compatible types) ในภาษา Modula-2 และ Pascal นั้น พิสัยย่อย สองชุดใดๆ ที่มี base type เหมือนกัน จะ สามารถใช้แทนกันได้

เกี่ยวข้องกับเทอม **<u>ความเข้ากันได้ของการกำหนดค่า</u>** (assignment compatibility) บ่อยครั้ง ใช้สำหรับ ความถูกต้อง ชนิด ของ ข้อความสั่ง x := e เริ่มต้น ข้อความนี้ อาจถูกตัดสินว่า ชนิด ถูกต้อง เมื่อ x และ e มีชนิดเหมือนกัน แต่สิ่งนี้ ละทิ้ง (ignores) ความแตกต่างที่สำคัญ : ทางด้านซ้ายมือ ต้องเป็น l-value หรือเลขที่อยู่ (ดูบทที่ 5) ในขณะที่ ทางด้านขวามือ ต้องเป็น r-value ภาษาจำนวนมาก แก้ปัญหานี้ โดยกำหนดว่า ทางด้านซ้ายมือ ต้องเป็นชื่อตัวแปร ซึ่งเลขที่อยู่ เป็น l-value และโดยอัตโนมัติ **<u>dereferencing</u>** ชื่อตัวแปรทางด้านขวามือ เพื่อให้ได้ r-value ของมัน ในภาษา Algol68 สิ่งนี้กระทำชัดแจ้งมากกว่า โดยพูดว่า การกำหนดค่าจะเป็นชนิดถูกต้องถ้า ชนิดของทางด้านซ้ายมือ (ซึ่งอาจจะเป็นนิพจน์ใดๆ) คือ ref t (เลขที่อยู่ ของค่าของ ชนิด t) และ ชนิดของด้านขวามือ คือ t

Algol68 อนุญาตให้ dereferencing เกิดขึ้นอัตโนมัติ ถ้าทางด้านขวามือ เป็นชนิด ref t

BLISS ต้องเป็น dereferencing ชัดแจ้ง :

ถ้า y เป็นตัวแปร เราต้องเขียน x := .y เครื่องหมาย "." ใช้เป็น dereference operator

ความเข้ากันได้กำหนดค่า อาจถูกขยายต่อไปได้ เพื่อรวม กรณีอื่นๆ เมื่อ ทั้งสองด้าน มีชนิด ไม่เหมือนกัน ตัวอย่างเช่น ใน Pascal แต่ไม่มี ใน Modula-2 การกำหนดค่า x := e ถูกต้องเมื่อ e เป็นชนิด integer และ x เป็นชนิด real : ค่า integer ของ e จะถูกแปลงผัน ให้เป็นค่า real จากนั้นเก็บไว้ใน x

ในทางตรงกันข้าม Modula-2 นิยามชนิด INTEGER และ CARDINAL ให้เป็นการกำหนดค่าซึ่งเข้ากันได้ ดังนั้น x := e เป็นชนิดถูกต้อง ถ้า x มี ชนิด CARDINAL และ e มีชนิดเป็น INTEGER หรือ ในทางย้อนกลับ ถูกต้องเช่นกัน ความเข้ากันได้ของการกำหนดค่านี้ เป็นการแปลงผันชนิดจริง จะอภิปรายในหัวข้อ 6.7

6.6.2 <u>ชนิดคาบเกี่ยวกัน</u> (Overlapping Types)

ชนิดอาจคาบเกี่ยวกันได้ เมื่อ สองชนิด ประกอบด้วย ค่าที่เหมือนกัน ตัวอย่างเช่น พิสัยย่อยสองชุด อาจคาบเกี่ยวกันดังนี้

พิสัยย่อยชนิด integer [-5 .. 5] และ [0 .. 10] คาบเกี่ยวกัน ในค่า 0 .. 5

เมื่อ ความถูกต้องชนิด ของ นิพจน์ และ ข้อความสั่ง ขยายถึงความเข้ากันได้ เพื่อรวมพิสัยย่อย ของชนิดเดียวกัน สิ่งนี้ทำให้เกิดข้อผิดพลาด ตัวอย่างเช่น x มีชนิด [-5 .. 5] และ y มีชนิด [0 .. 10] ดังนั้น โดยความเข้ากันได้ x := y เป็นชนิดถูกต้อง แต่ถ้า y มีค่าเท่ากับ 6 เมื่อข้อความสั่งนี้ ถูกกระทำการ จะเกิดข้อผิดพลาดการกระทำการ (execution error) แทนที่จะเป็น (static type error)

ใน Pascal และ Modula-2 *พิสัยย่อยของชนิดเดียวกัน เป็นการกำหนดค่าที่ใช้แทนกันได้* ถึงแม้ว่า พิสัยย่อย จะไม่มี ค่าจริง ร่วมกัน การตรวจสอบข้อผิดพลาด จะถูกเลื่อนไป ตรวจสอบในพิสัยเวลาการดำเนินงาน สิ่งนี้มองดูว่า เป็นการประนีประนอมชนิดที่ แข็ง หรือ อาจมองว่าพิสัยย่อย ไม่ใช่ ชนิดถูกแยกต่างหาก ออกไปจากตัวมันเอง ดังนั้น พิสัยย่อยไม่ใช่ชนิดใหม่ ถึงแม้ว่า จะถูกกำหนดชื่อ แตกต่าง ในการประกาศชนิด

กลยุทธอีกทางเลือกหนึ่ง ซึ่งใช้ใน Ada คือทำความแตกต่างระหว่าง ชนิดใหม่ จริง กับข้อกำหนด สำหรับการตรวจสอบพิสัยแบบพลวัต ใน Ada การประกาศพิสัยย่อย ซึ่ง โดยนัยแล้วคือ การตรวจสอบพิสัย เรียกว่า **<u>ชนิดย่อย</u>** (subtype) ในขณะที่พิสัยย่อย กลายเป็นชนิดใหม่ของมันเอง เรียกว่า <u>derived type</u>

ตัวอย่าง

```
subtype digit is INTEGER range 0 . . 9:
type newdigit is new INTEGER range 0 . . 9;
```

ในที่ นี้ ชนิดย่อย digit ไม่ใช่ ชนิดใหม่ : สมาชิกของชนิด digit จะมีค่าของมัน ซึ่ง ถูกตรวจสอบ ณ เวลากระทำการ อย่างไรก็ตาม newdigit เป็น derived type แตกต่างจาก digit และ integer คำถามของการตรวจสอบชนิดที่คล้ายกันเกิดขึ้น กับ ตัวสร้างชนิดเซตย่อย (any subset type constructor)

6.3.3 <u>ชนิดโดยนัย</u> (Implicit Types)

ชนิดของ เอนทิตีพื้นฐาน เช่น ตัวคงที่ และตัวแปร ไม่มีการกำหนด อย่างชัดแจ้งในการประกาศ ในกรณีนี้ ชนิด ต้องถูกอนุมาน จาก ตัวแปลภาษา อาจจะเป็น จาก สารสนเทศของบริบท หรือ จากกฎมาตรฐาน เราอาจกล่าวได้ว่า เป็น ชนิด<u>โดยนัย</u> (implicit) เพราะว่ามันไม่ได้กล่าวอย่างชัดแจ้งในโปรแกรม โดยที่ กฎซึ่งกำหนดชนิด ของมัน ต้องมีไว้ชัดแจ้งในบทนิยามของภาษา ตัวอย่างเช่น ใน Modula-2 ตัวคงที่ ไม่มีชนิดชัดแจ้ง ดังนั้น ต้องมีกฎกำหนดไว้ ในกรณีกำกวม โดยเฉพาะปัญหาที่เป็น จำนวนเต็มบวก เซต และ สายอักขระ Modula-2 มีทั้งชนิด INTEGER และ CARDINAL ซึ่งคาบเกี่ยวกัน สำหรับจำนวนเต็มบวก ตัวอย่างตัวคงที่ เช่น 3 หรือ 42 เป็น integer หรือเป็น cardinal กฎพูดว่า มันเป็นชนิด <u>ใช้แทนกันได้</u> (compatible) และมีคุณสมบัติทั้งสองชนิด สิ่งนี้คือ เราไม่ต้องวิตกเกี่ยวกับ การผสมกัน (mixing) ของเลข และ ตัวแปรของ ชนิด integer หรือชนิด cardinal ใน นิพจน์ใดๆ ยกเว้นเฉพาะ ตัวแปรชนิด cardinal และตัวแปร ชนิด integer ผสมกันไม่ได้

(except that variables of cardinal and integer type cannot be mixed.)

ในทางตรงกันข้าม ตัวคงที่ชนิดสายอักขระ (string constant) ถือว่ามีชนิดโดยนัย เป็น ARRAY [0 .. length] OF CHAR (ความสมมูลเชิงโครงสร้าง ให้กับ ชนิดทั้งหมด) ดังนั้น การประกาศข้างล่างนี้

```
CONST  blanks = '     '; (* five blanks *)
VAR  str1 : ARRAY [1 .. 5] OF CHAR;
     str2 : ARRAY [0 .. 4] OF CHAR;
```

การกำหนดค่า

str1 := blanks;

เกิดข้อผิดพลาดของชนิด (generate a type error) แต่การกำหนดค่า

str2 := blanks;

ไม่เกิด error

ใน Modula-2 อนุญาตให้ มี **ตัวคงที่เซต** (set constants) แต่เกิดปัญหา : **ไม่มี** explicit casting กับ ชนิด ซึ่งนิยามก่อนหน้านี้ (ดูหัวข้อ 6.7) มันเป็นชนิด predefined BITSET **ซึ่งขึ้น**อยู่กับการ implement

ปัญหาที่คล้ายกัน เกิดขึ้นได้ กับ ชนิดเซตย่อย ถ้า base type ไม่ได้กล่าวไว้อย่างชัดแจ้ง อาจเกิดความกำกวมได้ ตัวอย่างเช่น ใน Modula-2 คือ พิสัยย่อย [0 .. 9] เป็นเซตย่อยของ INTEGER หรือ CARDINAL มันอาจจะเป็นอันใดก็ได้ แต่บทนิยามของภาษา พูดว่า มันเป็น CARDINAL ปัญหานี้ หลีกเลี่ยงได้ โดย กำหนด base type อย่างชัดแจ้ง ดังนี้

TYPE digit = INTEGER [0 .. 9];

การแก้ปัญหาเช่นเดียวกันนี้ มีอยู่ใน Ada

6.6.4 <u>การดำเนินการร่วมกัน</u> (Shared Operations)

ชนิดต่างๆ มี เซต ของ การดำเนินการ ซึ่งเกี่ยวกับ มัน ปกติ เป็นโดยนัย บ่อยครั้งซึ่งการดำเนินการเหล่านี้ มีการ shared กัน ระหว่างหลายชนิด หรือ มีชื่อเหมือนกันกับ การดำเนินการอื่นๆ ซึ่งอาจจะแตกต่างกัน ตัวอย่างเช่น การดำเนินการ "+" อาจเป็น การบวก real หรือ การบวก integer หรือ ผลผนวกของเซต ตัวดำเนินการเช่นนี้ เรียกว่า **<u>overload</u>** เพราะว่า ชื่อเหมือนกัน ใช้กับ การดำเนินการต่างๆ ที่แตกต่างกัน ที่สำคัญ (ตัวดำเนินการชนิด overloaded เกี่ยวข้องกับ การดำเนินการ แบบ **<u>polymorphic</u>** ซึ่งหมายถึง การดำเนินการ ซึ่งสามารถประยุกต์ใช้กับ ชนิดที่แตกต่างกันได้)

ในกรณีของ ตัวดำเนินการแบบ overloaded ตัวแปลภาษา ต้องตัดสินใจว่า หมายถึง การดำเนินการอันไหน จากชนิดของตัวถูกดำเนินการของมัน ถ้า ชนิดของอาร์กิวเมนต์ ของ ตัวดำเนินการแบบ overloaded ต่างสมาชิกกัน (disjoint) ตัวแปลภาษา สามารถทำให้ การเลือก ของ ความหมาย การดำเนินการ ซึ่งต้องการ มีความกำกวมได้

เกิดอะไรขึ้น เมื่อชนิดของอาร์กิวเมนต์ ของ ตัวดำเนินการแบบ overloaded ผสมกัน หรือ คาบเกี่ยวกัน ตัวอย่างเช่น สำหรับนิพจน์ 3.14 + 1 ควรจะใช้การดำเนินการอันไหน ในบาง

ภาษา สิ่งนี้ จะเป็นสาเหตุให้ จำนวนเต็ม 1 ถูกแปลงผัน ให้เป็น real จากนั้น จึงประยุกต์ใช้ การบวกแบบ real นี่คือ ชนิด **การบังคับ** (coercion) ซึ่งจะอภิปรายรายละเอียดในหัวข้อ 6.7 ภาษาชนิด strongly typed ยืนยันว่า ชนิดของอาร์กิวเมนต์ทั้งหมด ของ ตัวดำเนินการ ต้องเหมือนกัน ดังนั้น จึงไม่มีการแปลงผัน เกี่ยวข้องด้วย แต่สิ่งนี้เกิดปัญหา เมื่อชนิด คาบเกี่ยวกัน ตัวอย่างเช่น ใน Modula-2 ถ้า i เป็น integer และ c เป็น cardinal นิพจน์ i + c ทำให้เกิด type error ถึงแม้ว่า ผลลัพธ์จะนิยามอย่างดี ไม่สนใจการตีความ นอกจากนี้แล้ว นิพจน์เช่น 2 + 3 อาจจะเป็นชนิด CARDINAL หรือชนิด INTEGER ก็ได้ (โปรดสังเกตว่า ตัวแปร ชนิด CARDINAL และตัวแปร ชนิด INTEGER เป็น ความเข้ากันได้แบบกำหนดค่า ใน Modula-2 แต่มัน ใช้แทนกันไม่ได้ในนิพจน์คำนวณ)

6.6.5 วัตถุซึ่งมีชนิดมากกว่าหนึ่ง (Multiply-Typed Objects)

บางครั้ง มันจะเป็นประโยชน์ สำหรับ การใช้แทนกันได้ เพื่อยอมให้ วัตถุมี ชนิดมากกว่าหนึ่งสิ่ง เราได้เห็นแล้วว่า จำนวนเต็ม ไม่ใช่ค่าลบ ใน Modula-2 มีชนิดเป็น INTEGER และชนิด CARDINAL ตัวอย่างที่คล้ายกัน เช่น ตัวชี้ NIL ซึ่งเป็นชนิดตัวชี้ใดๆ (any pointer type) สำหรับการเขียนโปรแกรมระบบ มันจะเป็นประโยชน์ ที่จะมี ตัวชี้อื่นๆ ซึ่งเป็น ชนิดตัวชี้ใดๆ ตัวอย่างเช่น ใน Modula-2 มีชนิด ADDRESS ซึ่ง จัดให้โดย มอดูลของ SYSTEM ซึ่งใช้แทนกันได้กับตัวชี้อื่นๆ ทั้งหมด ในทำนองเดียวกัน ตัวอักขระ หนึ่งตัว "a" อาจเป็นชนิด character หรือชนิด string ในบางภาษา

6.7 การแปลงผันของชนิด(Type Conversion)

ถ้าตัวแปร I และ J มีชนิดเป็น integer ข้อความสั่ง

I := J + 2.718

จะมีการตีความ (could be interpreted) หลายอย่าง ดังนี้

ภาษา C ยอมให้ การบวก สามารถกระทำได้ สำหรับ mixed types : J ถูกแปลงผันให้เป็น real และชนิดของผลลัพธ์ ของ J + 2.718 เป็น real จากนั้น ค่า real ถูกตัดปลาย (truncated) ให้เป็น integer และกำหนดให้เป็นค่าของ I

ในภาษา ซึ่งเป็น strongly type มากกว่า Modula-2 นั้น ใน นิพจน์คำนวณ reals และ integers ผสมกันไม่ได้ และ real กำหนดค่าให้กับ integers ไม่ได้ ดังนั้น ข้อความสั่งข้างต้น

เกิด (generate) type error

ใน Modula-2 จะต้องเขียนดังนี้

```
I := TRUNC(FLOAT(J) + 2.718)
```

เมื่อ FLOAT และ TRUNC เป็น built-in ฟังก์ชัน

ในภาษาโปรแกรม แต่ละภาษา มีความจำเป็นที่จะต้องแปลงผัน **ชนิดหนึ่ง ให้เป็นอีกชนิดหนึ่ง** *ภายใต้สถานะการณ์ที่แน่นอน* **การแปลงผันของชนิด** (type conversion) ถูกสร้างขึ้น *ภายใน* ระบบชนิด เพื่อให้การแปลงผันถูกกระทำอย่างอัตโนมัติ ตัวอย่างเช่น ใน ภาษา C เราเรียก การแปลงผันชนิดอัตโนมัติ ว่า **การบังคับชนิด** (type coercion)
บางครั้งอาจเรียกว่า การแปลงผันชนิดโดยนัย (implicit type conversion)

การบังคับ (coercion) มีประโยชน์ตรงที่ โปรแกรมเมอร์ ไม่ต้องเขียน รหัส เพิ่มขึ้น เพื่อให้กระทำการแปลงผันชัดเจน แต่เป็นความช่วยเหลือ ของเป้าหมาย ในการออกแบบ ให้ง่าย อย่างไรก็ตาม การถ่ายโอนอัตโนมัติ มี ผลกระทบ ไม่น่าพอใจเช่นกัน กล่าวคือ การตรวจสอบชนิดจะอ่อนลง เพื่อให้ข้อผิดพลาดนั้น ถูกจับไม่ได้ สิ่งนี้ ประนีประนอม ของการตรวจสอบอย่างแข็ง และความเชื่อถือได้ของภาษาโปรแกรม การแปลงผันอัตโนมัติ ทำให้เกิดพฤติกรรม ซึ่งคาดการณ์ไม่ได้ ที่ว่า โปรแกรมเมอร์ อาจคาดว่าการแปลงผันกระทำแล้วใน วิธีหนึ่ง ในขณะที่จริงๆ แล้ว ตัวแปลภาษากระทำการแปลงผันที่แตกต่าง

ตัวอย่างเช่น นิพจน์ ภาษา PL/I

1/3 + 15

ซึ่งแปลงผัน ให้เป็นค่า 5.333333333333333 (บนเครื่องชนิดความถูกต้อง 15 หลัก (15-digit precision)) เลข 1 นำหน้าหายไป เพราะ overflow โดย กฎของภาษา กำหนดว่า ความถูกต้องของค่าเศษส่วน ต้องเก็บไว้ (the language rules state that the precision of the functional value must be maintained.)

อีกทางเลือกหนึ่งให้กับ coercion คือ ใช้ **ฟังก์ชันการแปลงผันของชนิด** (type conversion functions) : ฟังก์ชันนั้น ถูกสร้าง ให้กับภาษา หรือ มีให้ใช้จาก library ซึ่งจะแปลงผัน ค่าของชนิดหนึ่ง ให้เป็น อีกชนิดหนึ่ง สิ่งนี้คือ รูปแบบ ของ **การแปลงผันชนิดชัดแจ้ง** (explicit type conversion) เพราะว่า การแปลงผัน กล่าวไว้อย่างชัดแจ้ง ในรหัส ตัวอย่างเช่น ฟังก์ชัน trunc และ round ใน Pascal และ TRUNC และ FLOAT ใน Modula-2 รูปแบบสำเร็จของ string มีรวมอยู่ในฟังก์ชันเช่นกัน เช่น StringToInt และ IntToString เป็นการแปลงผัน string ของ digit ให้เป็น value และในทางตรงกันข้าม จาก value ให้เป็น string ของ digit

ตัวอย่าง

StringToInt ('012') = 12

และ

IntToString (100) = '100'

ภาษา Modula-2 มีฟังก์ชันการแปลงผัน โดยทั่วไปสำหรับ ordinal type เรียกว่า ฟังก์ชัน VAL : มันเอาอาร์กิวเมนต์ตัวแรกของมันเป็น type และอาร์กิวเมนต์ตัวที่สองเป็น value ส่งกลับค่าใหม่ ของชนิดซึ่งกำหนดไว้

ตัวอย่าง

TYPE colors = (red, blue, green)

ดังนั้น

VAL (integer, red) = 0

และ

VAL (colors, 1) = blue

ฟังก์ชัน ซึ่งเอา อาร์กิวเมนต์เป็นชนิด ทำให้เป็นทั่วไปของระบบชนิด ในอีกวิธีอื่นๆ ได้เป็นเพียงการถ่ายโอนชนิดเท่านั้น

วิธีที่สาม เพื่อกระทำการถ่ายโอนชนิด ได้แก่ **cast** : ค่าหรือวัตถุ ของ ชนิดหนึ่ง ถูกนำหน้าโดย ชื่อของชนิด สิ่งนี้ ให้ผลลัพธ์ในการแปลงผัน ให้กับชนิดของชื่อ

(: a value or object of one type is preceded by a type name. This results in conversion to the named type.)

ในภาษา Modula-2 casts นั้นเขียนเหมือนกับฟังก์ชัน : INTEGER(red) และ colors(1) ให้การแปลงผันเหมือนกับ ฟังก์ชัน VAL ซึ่งได้กล่าวมาแล้ว จริงๆ แล้ว ในบทนิยามของ Modula-2 สิ่งเหล่านี้ เรียกว่า **ฟังก์ชันการถ่ายโอนชนิด** (type transfer functions) อย่างไรก็ตาม สิ่งเหล่านี้ ไม่ใช่ ฟังก์ชัน เช่นในความหมายปกติ ในรูปแบบพื้นฐานของมัน cast แตกต่างจาก ฟังก์ชันการถ่ายโอนชนิด ตรงที่ มันไม่ได้ทำให้เกิด การแปลงผันใดๆ กับการถูกกระทำนั้น : การแทนที่ภายใน คือ การตีความใหม่ ให้เป็นค่าของชนิดใหม่ ดังนั้น ตัวอย่างเช่น CARDINAL.(-1) = 65535 ในการ implement ของภาษา Modula-2 ส่วนมาก เพราะว่า -1 ถูกเก็บในรูปแบบของส่วนเติมเต็มของสอง (two's complement form) เป็น two-byte value

ภาษา C นั้น cast ผลลัพธ์ คือ ทำให้เกิดการแปลงผัน โดยนัย

(int) 3.14

ตัดปลาย ของ real ให้เป็น 3 ดังนั้น เป็นการกระทำ การแปลงผันจริง ใน Modula-2 เขียน INTEGER(3.14) ถือว่า ผิด (illegal)

ใน Ada, cast เขียนเหมือนฟังก์ชัน เช่นเดียวกับ Modula-2 แต่กระทำการแปลงผัน เหมือนใน C

Casts เป็นการแปลงผันชนิดอย่างชัดแจ้ง อีกประเภทหนึ่ง ทั้งมีความสำคัญ และเป็นประโยชน์ ขึ้นอยู่กับระบบชนิดของภาษานั้น ตัวอย่างเช่น ใน Modula-2 ตัวคงที่ set มี predefined type โดยนัยเป็น BITSET ยกเว้น cast เป็นอีกชนิดหนึ่ง ดังนั้น จึงเป็นไปไม่ได้ที่จะกำหนดค่า set constant ให้กับ set variable โดยไม่ใช้ cast :

<u>ตัวอย่าง</u>

```
TYPE  colorset = SET OF colors;
VAR   x : colorset;
...
x := colorset{real}
```

เป็นความพยายามที่จะทำให้เป็นการกำหนดค่า x := {red} จะเกิด type error ใน Modula-2 เพราะว่า set constant {red} ไม่มีชนิด เป็น SET OF colors (**<u>หมายเหตุ</u>** พูดอีกอย่างหนึ่งคือ สิ่งนี้เป็น type specifier ไม่ใช่ cast เพราะว่า ไม่ได้นำวงเล็บมาใช้ อย่างไรก็ตาม ความคิดพื้นฐานเหมือนกัน)

Algol68, casts นำมาใช้ ในการ dereferencing ได้ ตัวอย่างเช่น ถ้า x เป็นชนิด ref ref int (นั่นคือ ตัวชี้ ซึ่งค่าของมัน เป็น ตัวชี้ ไปยัง integer) และ y เป็นชนิด ref int, ดังนั้น x := y ทำให้ x ชี้ไปยัง y แต่ (ref int) x := (int) y หมายถึง การทำสำเนา (copies) ค่าของ y ไว้ในตำแหน่งซึ่ง x ชี้ (copies the value of y into the location that x points to.)

<u>วิธีสุดท้าย</u> สำหรับ การแปลงผันค่า จากชนิดหนึ่ง ไปเป็นอีกชนิดหนึ่ง คือ การจัดให้ โดย loophole ใน ชนิดแข็ง ของบางภาษา : ผลผนวกไม่แบ่งแยก สามารถเก็บค่าของชนิดต่างๆ และถ้าไม่มีการแบ่งแยก หรือ tag ตัวแปลภาษา จะไม่สามารถ แยกค่าต่างๆ ของชนิดหนึ่ง ออกจากอีกชนิดหนึ่งได้ ดังนั้น จึงยอม ให้ ไม่แบ่งแยก การตีความใหม่ของค่าต่างๆ

<u>ตัวอย่าง</u> การประกาศของ Modula-2

```
VAR x : RECORD CASE! BOOLEAN OF
                        TRUE : c : CHAR I
                        FALSE: b : BOOLEAN
                       END; (* case *)
     END (* record *)
```

และข้อความสั่ง

```
     x.b := TRUE;
     WritelnInt (ORD(x.c),1);
```

จะทำให้เราเห็น ค่าภายใน ซึ่งใช้แทนค่าแบบบูล TRUE การ implement ส่วนใหญ่ โปรแกรมนี้จะพิมพ์ 1

19-2

ในกรณีของ ผลผนวก แบบแบ่งแยกชนิด (discriminated union)

```
VAR x : RECORD CASE kind : BOOLEAN OF
                        TRUE : c : CHAR |
                        FALSE: b : BOOLEAN
                       END; (* case *)
     END. (* record *);
```

เป็นการยากที่จะสร้าง การตรวจสอบแบบพลวัต ซึ่งอาจจะเป็นเหตุให้รหัสต่อไปนี้ ล้มเหลว (fail) :

```
     x . kimd := TRUE
     x . b    := TRUE
     x . kind := TRUE
     WriteInt (ORD(x.c), 1);
```

จริงๆ แล้ว มีคอมไพเลอร์ จำนวนมาก ซึ่ง ไม่สามารถสร้างรหัสเพื่อตรวจสอบ discriminator value เมื่อเขตทางเลือกถูกเข้าถึง ดังนั้น รหัสเหมือนกัน แต่สำหรับ undiscriminated union จะทำงานสำหรับ discriminated ได้ดี

ใน Ada ใช้ ผลผนวก เพื่อกำจัด การตรวจสอบชนิด หมายถึง การป้องกัน โดยกำหนด discriminators และห้าม การ กำหนดค่าใหม่ (reassignment) ของ discriminator fields โดยตัวมันเอง

ตัวอย่าง ภาษา A d a

```
type CharBool (kind : BOOLEAN) is
    record case kind is;
        when hue => c : CHARACTER;
        when false => b : BOOLEAN;
        end case;
    end record;
```

แล้วกำหนด การประกาศ

```
x : CharBool (TRUE);
y : CharBool;
```

ในที่นี้ discriminant x.kind เปลี่ยนแปลงไม่ได้ เพราะว่า x ถูกประกาศ เป็นค่าคงที่สำหรับ discriminant และ y.kind กำหนดค่าโดยตัวมันเอง ไม่ได้ แต่ทุก เขตข้อมูล ของ y ต้องถูกกำหนดค่า หนึ่งครั้ง

```
y := (kind => FALSE, b => FALSE);
y := (kind => TRUE, c => 'a');
```

แบบฝึกหัด

1. สมมติว่า การแทนที่ของเซต เป็น บิต-เวกเตอร์ (Assume a bit-vector representation for sets.)

   (a) ถ้ากำหนดให้ INTEGER ใช้เนื้อที่ สองไบต์ การจัดสรรเนื้อที่ ให้กับตัวแปร ชนิด SET OF INTEGER จะต้องใช้ กี่ไบต์?

   (b) ถ้ากำหนดให้ INTEGER ใช้เนื้อที่ 4 ไบต์ การจัดสรรเนื้อที่ ให้กับ ตัวแปร ชนิด SET OF INTEGER จะต้องใช้กี่ไบต์?

   (c) จะต้องใช้เนื้อที่ กี่ไบต์ เพื่อแทน ตัวแปรชนิด SET OF SET OF CHAR

2. ข้อความสั่ง ใน text แสดงว่า ภาษา Pascal, Modula-2 และ Ada นั้น predefined type แบบบฎู หมายถึง ชนิด ordinal จะมีการเรียงลำดับอย่างไร? สิ่งนี้คือวิธีที่โปรแกรมเมอร์ใช้ ใช่หรือไม่? และมีเหตุผลอะไร ที่ว่า ชนิดแบบ Boolean ควรจะเป็น ordinal type

3. กำหนด การประกาศของ Pascal เป็นดังนี้

```
type    rc = record
                data : integer;
                next : ^rc;
             end;
var     x  :  ^rc;
```

   ข้อความสั่ง x := x^.next เกิด type error ให้อธิบาย และสิ่งเดียวกันนี้ จะเกิดขึ้นใน ภาษา Modula-2, C และ Ada หรือไม่

4. รหัสข้างล่างนี้ เป็นการประกาศชนิด ในวากยสัมพันธ์ ของ Pascal

```
type ptrl  =  ^rec1;
     ptr2 = ^rec2;
     rec1 = record
               data : integer;
               next : ptr2;
            end;
     rec2 = record
               data : integer;
               next : ptr1;
            end;
```

สิ่งเหล่านี้ถูกต้องหรือไม่ ให้อธิบาย

5. (a) จงอธิบาย ข้อแตกต่างระหว่าง ชนิด subtype กับชนิด derived ใน ภาษา Ada

   (b) จงเขียนการประกาศชนิดของ Pascal ซึ่ง equivalent กับการประกาศของ Ada ข้างล่างนี้

   ```
   subtype New_Int is INTEGER;
   ```

   (c) จงเขียนการประกาศชนิดของ Pascal ซึ่ง equivalent กับการประกาศของ Ada ข้างล่างนี้

   ```
   type New_Int is new INTEGER;
   ```

6. สมมติว่า เราต้องการให้นิยาม นิพจน์ if-then-else ซึ่งกระทำการทดสอบ จากนั้น ประเมินผล ค่า ของ then-part หรือ ประเมินผล ค่าของ else-part ขึ้นอยู่กับ ค่าของการทดสอบ

   <u>**ตัวอย่าง**</u> นิพจน์

   ```
   if 1 = 2 then 'a' else 'b'
   ```

   ควรจะประเมินผล ให้กับตัวอักขระ 'b' จงอธิบาย กฎของความถูกต้องของชนิดและกฎการอนุมานชนิด อะไร ซึ่งควรจะนำมาประยุกต์ให้กับ นิพจน์ if-then-else นี้ และเป็นไปได้หรือไม่ ที่จะนิยามนิพจน์เช่นนี้ กับ else-part ซึ่งละเว้นได้

7. ภาษา C มี นิพจน์ if-then-else คล้ายกับแบบฝึกหัดข้อ 6 ยกเว้น "If e1 then e2 else e3" เขียนดังนี้

   ```
   e1 ? e2 : e3
   ```

   จงอธิบาย กฎความถูกต้องของชนิด และกฎการอนุมาน สำหรับนิพจน์ชุดนี้ ใน C

8. รหัสข้างล่างนี้ เป็นการประกาศชนิดและตัวแปรในวากยสัมพันธ์ของ Pascal :

```
type
    range = -5 . . 5;
    table1 = array [range] of char;
    tabke2 = table1;
var
    x, y : array [-5 . . 5] of char,
      z : tablel;
      w : table2;
      i : range;
      j : -5 . . 5;
```

   จงอธิบายว่า ตัวแปรตัวไหนบ้าง ซึ่งเป็นความสมมูลของชนิด ภายใต้

(a) ความสมมูลเชิงโครงสร้าง

(b) ความสมมูลของชื่อ

และ (c) ความสมมูลของการประกาศ

แยกประเภท กรณีต่างๆ นี้ ซึ่งอาจจะกำกวม จากสารสนเทศเท่าที่มีอยู่

9. จงอธิบายถึง การดำเนินการ ชุดใดบ้าง ซึ่งสร้างโดยนัยด้วย ตัวสร้างชนิดแฟ้ม ใน Pascal

10. ในภาษาโปรแกรม ซึ่ง "/" อาจจะหมายถึง การหาร integer หรือ การหาร real และยอมให้มี coercion ระหว่าง integers กับ reals นิพจน์ I + J/K อาจให้ผลลัพธ์แตกต่างกัน จงอธิบายว่า สิ่งนี้เกิดขึ้นได้อย่างไร ภาษา FORTRAN ใช้การตีความ (interpretation) อันไหน? ภาษา C ใช้การตีความอันไหน การตีความชุดไหนดีกว่า

11. การดำเนินการอะไร ซึ่ง จำเป็น สำหรับ data type ชนิด string และกรณีของแถวลำดับ ของตัวอักขระ สนับสนุน การดำเนินการ เหล่านี้ คืออย่างไร?

12. กำหนด การประกาศ ของ Modula-2 ข้างล่างนี้

```
VAR  i : INTEGER;   c : CARDINAL;
```

สมมติว่า data type ชนิด INTEGER จาก -32768 ถึง 32767 และ ชนิด CARDINAL จาก 0 ถึง 65535 จงอธิบายว่า ข้อความสั่งต่อไปนี้ ชุดใดบ้าง ทำให้เกิด runtime error และชุดใดบ้างไม่เกิด

```
c := 50000;
i := c;
i := INTEGER(c);
i := VAL(INTEGER, c);
```