บทที่ 10

==========

การทำโปรแกรมเชิงตรรกะ
(Logic programming)

วัตถุประสงค์

(1) เพื่อให้ผู้เรียนเข้าใจแนวคิดพื้นฐานของการทำโปรแกรมเชิงตรรกะ

(2) เพื่อให้ผู้เรียนรู้สามารถแยกความแตกต่างระหว่างการทำโปรแกรมเชิงตรรกะ, การทำโปรแกรมเชิงหน้าที่, การทำโปรแกรมเชิงวัตถุ และการทำโปรแกรมเชิงคำสั่ง

(3) เพื่อทบทวนความรู้พื้นฐานเกี่ยวกับตรรกศาสตร์

(4) เพื่อแนะนำตรรกศาสตร์อันดับหนึ่งซึ่งใช้เป็นหลักการพื้นฐานของภาษาโปรล็อก

(5) เพื่ออธิบายวิธีการพิสูจน์ด้วยเทคนิคเรโซลูชันรวมถึงอธิบายขั้นตอนการทำยูนิฟิเคชัน

(6) เพื่อแนะนำผู้เรียนถึงแนวคิดของภาษาคอมพิวเตอร์ที่ใช้ในงานปัญญาประดิษฐ์

(7) เพื่ออธิบายโครงสร้างและวิธีการทำงานของภาษาโปรล็อก

(8) เพื่อแนะนำให้ผู้เรียนรู้จักวิธีการทำโปรแกรมเชิงตรรกะแบบมีเงื่อนไข

(9) เพื่อแสดงตัวอย่างการทำโปรแกรมเชิงตรรกะ

การทำโปรแกรมเป็นการเขียนขั้นตอนการทำงานเพื่อนำไปสู่การแก้ปัญหาบางอย่าง วิธีการทำโปรแกรมเชิงคำสั่งใช้วิธีสั่งการทำงานอย่างเป็นลำดับขั้นตอนโดยละเอียดเพื่อให้คอมพิวเตอร์แก้ปัญหาได้ และได้คำตอบที่ถูกต้อง วิธีการทำโปรแกรมเชิงวัตถุมีแนวทางแบบเดียวกับการทำโปรแกรมเชิงคำสั่งเพียงแต่เปลี่ยนจุดเน้นการทำงานจากคำสั่งควบคุมมาเป็นกลไกการทำงานของข้อมูลในลักษณะวัตถุ วิธีการทำโปรแกรมเชิงหน้าที่ ใช้การอธิบายแนวทางการแก้ปัญหาในรูปแบบของฟังก์ชันที่ทำหน้าที่เชื่อมโยงจากข้อมูลเข้าไปสู่คำตอบของปัญหา ซึ่งแนวทางนี้ช่วยให้โปรแกรมเมอร์สนใจกับปัญหาและคำตอบของปัญหาโดยไม่ต้องสนใจการสั่งงานคอมพิวเตอร์ในขั้นตอนโดยละเอียด วิธีการทำโปรแกรมเชิงตรรกะมุ่งเน้นที่ปัญหาและคำตอบของปัญหาเช่นเดียวกัน แต่เครื่องมือที่ใช้เปลี่ยนจากฟังก์ชันเป็นความสัมพันธ์ในเชิงตรรกศาสตร์นั่นคือความสัมพันธ์ที่ระบุค่าความจริงหรือเท็จได้ โปรแกรมเชิงตรรกะจึงประกอบด้วยข้อความที่เป็นจริงและความสัมพันธ์ที่เกี่ยวข้องกับปัญหา

10.1 หลักการทำโปรแกรมเชิงตรรกะ
(Principles of logic programming)

การทำโปรแกรมเชิงตรรกะถูกเรียกว่าการทำโปรแกรมเชิงประกาศ (declarative programming) เนื่องจากโปรแกรมเมอร์ใช้วิธีประกาศเป้าหมายของโปรแกรมและเงื่อนไขข้อบังคับต่างๆ ที่จะสอดคล้องก่อนการบรรลุเป้าหมาย โดยไม่ต้องระบุขั้นตอนที่จะไปสู่เป้าหมาย

ตัวอย่างเช่น ในโจทย์ปัญหาการจองเที่ยวบิน ข้อมูลที่เกี่ยวข้องประกอบด้วยหมายเลขเที่ยวบิน, ชื่อเมืองที่เครื่องบินขึ้น, ชื่อเมืองที่เครื่องบินลงจอด, เวลาเดินทาง, เวลาถึงที่หมาย ข้อมูลทั้งหลายที่เกี่ยวข้องกับการจองเที่ยวบิน เขียนอยู่ในรูปแบบของโปรแกรมเชิงตรรกะได้ดังนี้

```
flight (flight_number, from_city, to_city, departure_time,
        arrival_time)
```

การจองเที่ยวบินจากกรุงเทพฯ ไปยังลอสแองเจลิส ถ้าต้องการเที่ยวบินตรงสามารถระบุเป้าหมายได้ดังนี้

```
flight (flight_number, "Bangkok", "Los Angeles", departure_time,
        arrival_time)
```

การระบุชื่อเมือง Bangkok และ Los Angeles เป็นการระบุเงื่อนไขของเป้าหมายที่ต้องการ ตัวแปลภาษาจะใช้กลไกการทำงานภายในตรวจสอบข้อมูลการบินในฐานข้อมูลว่ามีเที่ยวบินใดตรงกับเงื่อนไขที่ระบุบ้าง หรือถ้าต้องการเที่ยวบินที่มีการพักเปลี่ยนเครื่องโดยช่วงเวลาเปลี่ยนเครื่องจะต้องนานกว่า 30 นาที เราสามารถระบุเงื่อนไขเพิ่มเติมได้ดังนี้

```
flight (flight1, "Bangkok", X, depart1, arrive1),
flight (flight2, X, "Los Angeles", depart2, arrive2),
        depart2 >= arrive1 + 30
```

จะสังเกตได้ว่าการทำโปรแกรมเชิงประกาศ โปรแกรมเมอร์ไม่ต้องระบุขั้นตอนการค้นหาข้อมูล การตรวจเช็คข้อมูล รวมถึงขั้นตอนในรายละเอียดอื่นๆ โปรแกรมเมอร์เพียงแต่ระบุจุดมุ่งหมายที่ต้องการ กลไกภายในของภาษาจะทำหน้าที่ตรวจสอบข้อมูลหาคำตอบที่ตรงกับเงื่อนไขที่ระบุอยู่ในโปรแกรม

การทำโปรแกรมเชิงประกาศถูกใช้งานอย่างมากในงานสองด้าน คือ ด้านฐานข้อมูลที่ใช้ภาษา SQL เป็นภาษาการทำโปรแกรม และงานด้านปัญญาประดิษฐ์ที่ใช้ภาษา Prolog เป็นภาษาการทำโปรแกรม

การทำโปรแกรมเชิงประกาศในงานด้านปัญญาประดิษฐ์ถูกเรียกว่า การทำโปรแกรมเชิงตรรกะ เนื่องจากใช้ตรรกศาตร์ซึ่งเป็นคณิตศาสตร์แขนงหนึ่ง เป็นพื้นฐานของการออกแบบภาษาและกลไกการทำงานของภาษา สาเหตุที่ตรรกศาสตร์กลายเป็นรากฐานของการทำโปรแกรมเชิงตรรกะก็เนื่องจากงานด้านปัญญาประดิษฐ์ ในยุคแรกเป็นการประมวลผลภาษาธรรมชาติ (natural language processing) และการพิสูจน์ทฤษฎีโดยอัตโนมัติ (automatic theorem proving) ซึ่งต้องใช้การประมวลผลกับค่าและสัญลักษณ์ต่างๆ และต้องมีกระบวนการพิสูจน์ความถูกต้องของทฤษฎีบท

ตรรกศาสตร์ที่ใช้ในการทำโปรแกรมเชิงตรรกะเป็นตรรกศาสตร์อันดับหนึ่ง (first-order logic) หรือเรียกได้อีกชื่อว่า ตรรกศาสตร์เพรดิเคต (predicate logic) ซึ่งขยายขอบเขตขึ้นมาจากตรรกศาสตร์พรอพโพซิชัน (propositional logic or zeroth-order logic) และกระบวนการพิสูจน์ความถูกต้องของทฤษฎี หรือ ข้อความต่างๆ จะใช้วิธีการพิสูจน์เชิงปฏิเสธที่เรียกว่าเรโซลูชัน (resolution) ซึ่งจะต้องอาศัยเทคนิคการทำให้ข้อความสอดคล้อง หรือเรียกว่า ยูนิฟิเคชัน (unification)

ภาษา Prolog (Programming in logic) เป็นภาษาคอมพิวเตอร์ที่ใช้ในการทำโปรแกรมเชิงตรรกะ พัฒนาขึ้นโดย Alaine Colmerauer และ Philippe Roussel ในช่วงปี ค.ศ. 1970 นอกจากยูนิฟิเคชันและเรโซลูชัน (คิดขึ้นโดย Robinson ในปี ค.ศ. 1965) แล้วการย้อนกลับ หรือ แบ็คแทร็คกิ้ง (backtracking) ยังเป็นกลไกที่ช่วยให้ภาษา Prolog ค้นหาคำตอบได้มากกว่าหนึ่งคำตอบ

10.2 พื้นฐานตรรกศาสตร์
(Fundamentals of logic)

การทำโปรแกรมเชิงตรรกะ เป็นการทำโปรแกรมที่ใช้วิธีประกาศเป้าหมายของโปรแกรมโดยไม่ต้องระบุขั้นตอนเพื่อไปสู่เป้าหมายนั้น คำประกาศเป้าหมายจะเขียนอยู่ในรูปแบบของข้อความทางตรรกะ ตรรกศาสตร์ที่ใช้เป็นตรรกศาสตร์อันดับหนึ่งหรือตรรกศาสตร์เพรดิเคต ซึ่งพัฒนาขึ้นมาจากตรรกศาสตร์พื้นฐานหรือตรรกศาสตร์พรอพโพซิชัน

<u>ตรรกศาสตร์พรอพโพซิชัน</u>

นิพจน์ตรรกะทั้งหลายในภาษาคอมพิวเตอร์จะใช้รูปแบบตรรกศาสตร์พรอพโพซิชัน ประโยคหรือข้อความทางตรรกศาสตร์ คือประโยคที่ให้ค่าความจริงหรือเท็จ และนิยมเรียกประโยคแบบนี้ว่าพรอพโพซิชัน (proposition) ประโยคที่ไม่สามารถหาค่าความเป็นจริงได้ (เช่น ประโยคคำถาม) จะไม่เรียกว่าพรอพโพซิชัน กฎเกณฑ์ที่ใช้ในการพิจารณาว่าประโยคเป็นพรอพโพซิชันหรือไม่ มี 3 ประการดังนี้

- ค่าคงที่ true และ false เป็นพรอพโพซิชัน
- ตัวแปร p, q, r, … ที่มีค่า true หรือ false เป็นพรอพโพซิชัน
- ถ้า P และ Q เป็นพรอพโพซิชันแล้ว ข้อความต่อไปนี้เป็นพรอพโพซิชัน

  $P \wedge Q$ (โอเปอเรเตอร์ $\wedge$ เรียกว่า conjunction)

  $P \vee Q$ (โอเปอเรเตอร์ $\vee$ เรียกว่า disjunction)

  $P \supset Q$ (โอเปอเรเตอร์ $\supset$ เรียกว่า implication)

  $P \equiv Q$ (โอเปอเรเตอร์ $\equiv$ เรียกว่า equivalence)

  $\neg P$ (โอเปอเรเตอร์ $\neg$ เรียกว่า negation)

ลำดับความสำคัญของโอเปอเรเตอร์ทั้ง 5 โอเปอเรเตอร์ เรียงลำดับจากสูงสุดไปต่ำสุดได้ดังนี้

$$\neg \gg \wedge \gg \vee \gg \supset \gg \equiv$$

ดังนั้นนิพจน์ต่อไปนี้ที่เขียนโดยไม่มีวงเล็บระบุลำดับการทำงาน

$$p \vee q \wedge r \supset \neg s \vee t$$

จะมีความหมายเทียบเท่ากับนิพจน์ต่อไปนี้

$$((p \vee (q \wedge r)) \supset ((\neg s) \vee t))$$

ตัวแปรในตรรกศาสตร์พรอพโพซิชัน จะใช้แทนประโยคที่ให้ค่าความจริงเป็นจริงหรือเท็จ เช่น กำหนดให้

p แทนประโยค “Mary speaks Russian”

q แทนประโยค “Bob speaks Russian”

r แทนประโยค “Mary and Bob can communicate with each other”

ดังนั้น

$p \wedge q$ แทนประโยค “Mary and Bob both speak Russian”

$p \vee q$ แทนประโยค “Either Mary or Bob speak Russian”

$p \wedge q \supset r$ แทนประโยค “If Mary and Bob both speak Russian, then they can communication with each other”

การประมวลผลที่เกิดขึ้นในตรรกศาสตร์พรอพโพซิชัน คือ การพิจารณาค่าความจริงของประโยคว่าให้ค่าเป็นจริงหรือเป็นเท็จ เครื่องมือที่ใช้ในการพิจารณาค่าความจริงคือตารางค่าความจริง (truth table) ซึ่งแสดงการหาค่าความจริงของทั้ง 5 โอเปอเรเตอร์ได้ดังรูปที่ 10.1

p	**q**	**p∧q**	**p∨q**	**p⊃q**	**p≡q**	**¬p**
T	T	T	T	T	T	F
T	F	F	T	F	F	F
F	T	F	T	T	F	T
F	F	F	F	T	T	T

รูปที่ 10.1 ตารางค่าความจริงของโอเปอเรเตอร์ที่ใช้ในนิพจน์พรอพโพซิชัน

ตัวอย่างเช่น ถ้าเรารู้ว่า p ให้เป็นค่าเป็นจริง q ให้ค่าเป็นจริง r ให้ค่าเป็นเท็จ เราสามารถหาค่าความจริงของประโยค $(\neg p \vee q) \supset r$ ได้ดังนี้

p	**q**	**r**	**¬p**	**(¬p ∨ q)**	**(¬p ∨ q) ⊃ r**
T	T	T	F	T	T
T	T	F	F	T	F
T	F	T	F	F	T
T	F	F	F	F	T
F	T	T	T	T	T
F	T	F	T	T	F
F	F	T	T	T	T
F	F	F	T	T	F

ดังนั้น ค่าความจริงของประโยค $(\neg p \vee q) \supset r$ ให้ค่าเป็นเท็จ

<u>ตรรกศาสตร์เพรดิเคต</u>

ตรรกศาสตร์เพรดิเคต หรือบางครั้งเรียกว่าแคลคูลัสเพรดิเคต (predicate calculus) ปรับปรุงเพิ่มเติมขึ้นจากตรรกศาสตร์พรอพโพซิชันโดยกำหนดให้พรอพโพซิชันนอกจากจะประกอบด้วยค่าคงที่ true/false, ตัวแปรตรรกะและนิพจน์เชิงประกอบที่ใช้โอปอเรเตอร์ทั้ง 5 ประเภท เป็นตัวเชื่อมแล้วยังสามารถประกอบด้วยตัวแปรประเภทอื่น (เช่น integer, real), ฟังก์ชันที่ให้ค่าตรรกะและตัวบ่งปริมาณ (quantifier) คำว่าเพรดิเคตหมายถึง พรอพโพซิชันที่ตัวแปรตรรกะสามารถถูกแทนที่ด้วยฟังก์ชันที่ให้ค่าตรรกะและนิพจน์ที่มีตัวบ่งปริมาณ (ตัวบ่งปริมาณที่ใช้มี 2 ตัวคือ $\forall$ หมายถึงทั้งหมดและ $\exists$ หมายถึงบางส่วน)

ฟังก์ชันที่ให้ค่าตรรกะ(Boolean-valued function) หมายถึง ฟังก์ชันที่ส่งค่ากลับเป็น true หรือ false ตัวอย่างเช่น

$0 \leq x + y$ ให้ค่าเป็นจริงเมื่อผลบวกของ x และ y ไม่ใช่ค่าจำนวนเต็มลบ มิฉะนั้นจะให้ค่าเป็นเท็จ

speak (x, y) ให้ค่าเป็นจริงเมื่อ x เป็นบุคคลที่สามารถพูดภาษา y มิฉะนั้นจะให้ค่าเป็นเท็จ

ตัวอย่างต่อไปนี้แสดงเพรดิเคต และ ค่าของเพรดิเคต

<u>เพรดิเคต</u>	<u>ค่าของเพรดิเคต</u>
$0 \leq x \wedge x \leq 1$	เป็นจริงเมื่อ x มีค่าอยู่ระหว่าง 0 ถึง 1 โดยรวมค่า 0 และค่า 1 มิฉะนั้นเป็นเท็จ
speaks (x, Russian) $\wedge$ speaks (y, Russian) $\supset$ talkswith (x, y)	เป็นจริงเมื่อ x สามารถพูดภาษารัสเซีย y สามารถพูดภาษารัสเซีย และ x สามารถสื่อสารกับ y มิฉะนั้นจะเป็นเท็จ
$\forall$x(speaks(x, Russian))	เป็นจริงถ้าทุกคนในโลกนี้สามารถพูดภาษารัสเซียมิฉะนั้นจะเป็นเท็จ
$\exists$x(speaks(x, Russian))	เป็นจริงถ้ามีคนจำนวนอย่างน้อย 1 คน สามารถพูดภาษารัสเซีย มิฉะนั้นจะเป็นเท็จ
$\forall$x$\exists$y(speaks(x, y))	เป็นจริงถ้าทุกคนพูดได้อย่างน้อย 1 ภาษา มิฉะนั้นจะเป็นเท็จ

รูปที่ 10.2 ต่อไปนี้แสดงโดยสรุปถึงนิพจน์ประเภทต่างๆ ที่ใช้ในตรรกศาสตร์เพรดิเคต พร้อมทั้งอธิบายโดยย่อถึงความหมายทางนิพจน์แต่ละประเภท

สัญลักษณ์	ความหมาย	หมายเหตุ/ตัวอย่าง
true, false	Boolean (truth) constants	
p, q, r	Boolean variables	
$\neg p$	Negation of p	True if p is false; otherwise false
$p \wedge q$	Conjunction of p and q	True if p and q are both true
$p \vee q$	Disjunction of p and q	True if either p or q (or both) is true
$p \supset q$	Implication: p implies q	Logically equivalent to $\neg p \wedge q$
$p \equiv q$	Logical equivalence of p and q	True if p and q are bot true or both false
$\forall xP(x)$	Universally quantified expression	For all values of x,P(x) is true
$\exists xP(x)$	Existentially quantified expression	There is an x for which P(x) is true
p is a tautology	Proposition p is always true	E.g., $q \vee \neg q$ is a tautology
p(X) is valid	Predicate p(X) is true for every value of x	E.g., even(x) $\vee$ odd(x) is valid

รูปที่ 10.2 นิพจน์และความหมายของนิพจน์ในตรรกศาสตร์เพรดิเคต

จากรูปที่ 10.2 ปรากฏนิพจน์สองประเภทคือ นิพจน์ที่เป็นทอตโทโลยี (tautology) และนิพจน์สมเหตุสมผล (valid) นิพจน์ทอตโทโลยีหมายถึงนิพจน์ที่เป็นจริงเสมอ เช่น $q \vee \neg q$ นิพจน์ที่มีการใช้ตัวแปรและมีค่าของตัวแปรบางค่าที่ทำให้นิพจน์เป็นจริง จะเรียกว่านิพจน์นั้นสามารถเป็นจริงได้ (satisfiable) ถ้าค่าทุกค่าที่เป็นไปได้ของตัวแปรส่งผลให้นิพจน์เป็นจริง จะเรียกว่านิพจน์นั้นสมเหตุสมผล (valid) ตัวอย่างเช่น นิพจน์ speaks(x, Russian) สามารถเป็นจริงได้เพราะจะต้องมีอย่างน้อยหนึ่งคนในโลกนี้ที่สามารถพูดภาษารัสเซีย แต่นิพจน์ไม่ใช่นิพจน์สมเหตุสมผลเพราะไม่เป็นความจริงที่ทุกคนในโลกนี้สามารถพูดภาษารัสเซียนิพจน์ $y \geq 0 \wedge n \geq 0 \wedge z = x(y-n)$ เป็นนิพจน์ที่สามารถเป็นจริงได้แต่ยังไม่ใช่นิพจน์สมเหตุสมผล เพราะมีบางค่าของ x,y,z,n ที่สามารถทำให้นิพจน์เป็นเท็จนิพจน์ที่สามารถเป็นจริงได้และสมเหตุสมผลได้แก่ even(y) $\vee$ odd(y)

นิพจน์ในตรรกศาสตร์เพรดิเคตมีคุณสมบัติเช่นเดียวกับนิพจน์พีชคณิต นั่นคือคุณสมบัติสมมูลหรือการเทียบเท่ากัน ซึ่งช่วยให้นิพจน์เพรดิเคตสามารถถูกแปลงรูปและถูกทำให้อยู่ในรูปที่สั้นลงได้ รูปที่ 10.3 แสดงคุณสมบัติความเทียบเท่ากันของนิพจน์เพรดิเคต

คุณสมบัติ	ความหมาย	
Commutativity	$p \vee q \equiv q \vee p$	$p \wedge q \equiv q \wedge p$
Associativity	$(p \vee q) \vee r \equiv p \vee (q \vee r)$	$(p \wedge q) \wedge r \equiv p \wedge (q \wedge r)$
Distributivity	$p \vee q \wedge r \equiv (p \vee q) \wedge (p \vee r)$	$p \wedge (q \vee r) \equiv p \wedge q \vee p \wedge r$
Idempotence	$p \vee p \equiv p$	$p \wedge p \equiv p$
Identity	$p \vee \neg p \equiv true$	$p \wedge \neg p \equiv false$
deMorgan	$\neg(p \vee q) \equiv \neg p \wedge \neg q$	$\neg(p \wedge q) \equiv \neg p \vee \neg q$
Implication	$p \supset q \equiv \neg p \wedge q$	
Quantification	$\neg \forall x P(x) \equiv \exists x \neg P(x)$	$\neg \exists x P(x) \equiv \forall x \neg P(x)$

รูปที่ 10.3 คุณสมบัติของนิพจน์ในตรรกศาสตร์เพรดิเคต

<u>ข้อความรูปแบบฮอร์น</u>

ข้อความรูปแบบฮอร์น (Horn clause) เป็นเซ็ตย่อยของข้อความเพรดิเคตถูกสร้างขึ้นมาเพื่อช่วยให้สามารถพิสูจน์คุณสมบัติสมเหตุสมผลของนิพจน์ได้ และเป็นรูปแบบข้อความที่ใช้ในภาษา Prolog

ข้อความรูปแบบฮอร์นเป็นข้อความตรรกะที่อยู่ในลักษณะ implication ดังนี้

$$h \leftarrow p_1, p_2, \ldots, p_n$$

h, p1, p2,….,p_n คือเพรดิเคต h เป็นเพรดิเคตที่อยู่ ตรงส่วนหัวของลูกศร จึงเรียกเพรดิเคตนี้ว่าเพรดิเคตส่วนหัว (head) และเรียก $p_1, p_2, \ldots, p_n$ ว่าเพรดิเคตส่วนบอดี้ (body) ข้อความรูปแบบฮอร์นนี้มีข้อกำหนดว่าเพรดิเคตส่วนหัวจะมีเพรดิเคตเดียว ส่วนเพรดิเคตในส่วนบอดี้ จะมีกี่เพรดิเคตก็ได้หรือไม่มีเพรดิเคตเลยก็ได้ (ในภาษา Prolog จะเรียกข้อความที่มีเฉพาะเพรดิเคตส่วนหัวโดยไม่มีเพรดิเคตส่วนบอดี้ว่า ข้อความที่เป็นจริงหรือ fact)

ข้อความ $h \leftarrow p_1, p_2, \ldots, p_n$,มีความหมายว่า n จะเป็นจริงก็ต่อเมื่อ p_1 และ p_2 และ … และ p_n เป็นจริง ดังนั้นเครื่องหมาย “,” จึงแทนความหมาย AND หรือ $\wedge$ (ในภาษาProlog จะเรียกข้อความแบบนี้ว่า กฎ หรือ rule)

ตัวอย่างต่อไปนี้แสดงข้อความรูปแบบฮอร์นที่มีความหมายว่า หิมะตกที่เมือง c จะเป็นจริงก็ต่อเมื่อมีฝนตกที่เมือง c และอุณหภูมิของเมือง c ในขณะนั้นต่ำกว่าจุดเยือกแข็ง

```
snowing (c) ← precipitation (c), freezing (c)
```

ข้อความรูปแบบฮอร์นข้างต้น สามารถเขียนอยู่ในรูปแบบเคลคูลัสเพรดิเคตได้ดังนี้

```
precipitation (c) ∧ freezing (c) ⊃ snowing (c)
```

ซึ่งมีความหมายของข้อความจะเทียบเท่ากับข้อความต่อไปนี้

```
¬precipitation (c) ∨ ¬freezing (c) ∨ snowing (c)
```

ข้อความทุกข้อความที่อยู่ในรูปแบบฮอร์นสามารถถูกแปลงให้อยู่ในรูปแบบแคลคูลัสเพรดิเคต แต่ในทางกลับกันอาจมีบางข้อความในรูปแบบแคลคูลัสเพรดิเคตที่ไม่สามารถแปลงให้เป็นรูปแบบฮอร์น ทั้งนี้เนื่องจากข้อความรูปแบบฮอร์นเป็นสับเซ็ตของแคลคูลัสเพรดิเคต แต่ถ้าข้อความใดสามารถแปลงเป็นรูปแบบฮอร์น ขั้นตอนในการแปลงจะประกอบด้วย

(1) กำจัดสัญลักษณ์ implication
เช่น $p \supset q$ จะถูกแปลงเป็น $\neg p \vee q$

(2) ใช้กฎเดอร์มอแกน (Demorgan's laws) เพื่อเลื่อนสัญลักษณ์ ¬ เข้าไปภายในวงเล็บ
เช่น $\neg(p \wedge q)$ จะถูกแปลงเป็น $\neg p \vee \neg q$

(3) กำจัดตัวบ่งปริมาณ ∃ ด้วยเทคนิคที่เรียกว่าการทำสโกเลม (skolemization) ซึ่งจะแทน
$\exists x\ p(x)$ ด้วยค่าคงที่ c และทำให้เทอมเปลี่ยนรูปเป็น p(c)

(4) ย้ายตัวบ่งปริมาณ ∀ ไปรวมกันที่ต้นข้อความ จากนั้นตัดสัญลักษณ์ ∀ ออกจากข้อความให้เหลือเพียงความหมายแฝงว่าตัวแปรทุกตัวที่ปรากฏในข้อความต้องพิจารณาทุกค่าของตัวแปร

(5) เปลี่ยนข้อความให้อยู่ในรูปแบบปกติแบบคอนจังทีฟ (conjunctive normal form)
หรือรูปแบบที่เชื่อมเทอมด้วย ∧ (เทอมชั้นในอาจเชื่อมด้วยเครื่องหมาย ∨ ได้)

(6) เปลี่ยนข้อความให้อยู่ในรูปแบบ implication

ตัวอย่าง เช่น ข้อความต่อไปนี้เขียนอยู่ในรูปแบบแคลคูลัส มีความหมายว่า สำหรับคนทุกคนถ้าคนๆ นั้นไม่ได้รับการศึกษาแล้วเขาจะไม่สามารถเขียน และไม่สามารถอ่านหนังสือได้แม้สักเล่มเดียว

```
∀x (¬literate(x) ∨ (¬writes(x) ∧ ¬∃y (reals(x, y) ∧ book(y))))
```

ข้อความนี้สามารถถูกแปลงให้อยู่ในรูปแบบฮอร์น (นั่นคือรูปแบบ $h \leftarrow p_1, p_2, \ldots, p_n$)ได้ดังนี้

(1) กำจัดสัญลักษณ์ implication

```
∀x (literate(x) ∨ (¬writes(x) ∧ ¬∃y (reads(x,y) ∨ book(y))))
```

(2) เลื่อน ¬ เข้าไปในวงเล็บ

```
∀x (literate(x) ∨ (¬writes(x) ∧ ∀y(¬reads(x,y) ∨ ¬book(y))))
```

(3) ไม่มี ∃ จึงไม่ต้องทำ skolemization

(4) ย้าย ∀ ไปไว้หน้าข้อความ

```
∀x∀y (literate(x) ∨ (¬writes(x) ∧ (¬read(x, y) ∨ ¬book(y))))
= literate(x) ∧ (¬writes(x) ∧ (¬reads(x, y) ∨ ¬book(y)))
```

(5) แปลงรูปแบบคอนจังทีฟ

```
(literate(x) ∨ ¬writes(x)) ∧ (literate(x) ∨ ¬reads(x, y) ∨
 ¬book(y))
= (¬writes(x) ∨ literate(x)) ∧ (¬reads(x, y) ∨ ¬book(y) ∨
   literate(x))
```

(6) แปลง $\neg p \vee q$ เป็น $p \supset q$

```
(writes(x) ⊃ literate(x)) ∧
((reads(x, y) ∧ book(y)) ⊃ literate(x))
```

จากนั้นแต่ละเทอมย่อยที่มีเครื่องหมาย ⊃ สามารถเขียนอยู่ในรูปแบบฮอร์นได้ดังนี้

```
literate(x) ← writes(x)
literate(x) ← reads(x, y), book(y)
```

ถ้าเปลี่ยนข้อความในตัวอย่างข้างต้นเป็น คนมีการศึกษาทุกคนสามารถเขียนหรืออ่านได้ ซึ่งเขียนเป็นแคลคูลัสเพรดิเคตได้ดังนี้

```
∀x (literate(x) ⊃ reads(x) ∨ writes(x))
```

ในขั้นตอนการแปลงเป็นรูปแบบฮอร์น จะมีการเปลี่ยนรูปข้อความไปดังนี้

```
= ∀x (¬literate(x) ∨ reads(x) ∨ writes(x))
= ¬literate(x) ∨ reads(x) ∨ writes(x)
=  literate(x) ⊃ reads(x) ∨ writes(x)
```

ถ้าเขียนอยู่ในรูปแบบ $h \leftarrow p_1, p_2, \ldots, p_n$ จะได้

```
reads(x) ∨ writes(x) ← literate(x)
```

ซึ่งไม่ใช่รูปแบบฮอร์นเพราะเพรดิเคตส่วนหัวมีมากกว่าหนึ่งเพรดิเคต ดังนั้นข้อความแคลคูลัสเพรดิเคตข้างต้นไม่สามารถแปลงเป็นรูปแบบฮอร์นได้

10.3 เรโซลูชันและยูนิฟิเคชัน

(Resolution and unification)

ในการทำโปรแกรมเชิงตรรกะโปรแกรมจะประกอบด้วยข้อความที่เป็นจริง 2 ประเภท คือข้อความที่เป็นจริงโดยไม่มีเงื่อนไข เรียกว่า ความจริง (fact) และข้อความที่เป็นจริงถ้าเงื่อนไขเป็นจริง เรียกว่ากฎ (rule) ตัวอย่างของความจริงและกฎ แสดงได้ดังนี้

```
speaks(Mary, English)
talkswith(X, Y) ← speaks(X, L), speaks(Y, L), X ≠ Y
```

ข้อความแรกคือ ความจริงที่ระบุว่า Mary พูดภาษาอังกฤษ และข้อความที่สองคือกฎหรือความจริงแบบมีเงื่อนไข ที่ระบุว่า บุคคลสองคนใดๆ (แทนด้วย X และ Y) X สามารถคุยกับ Y ได้ถ้า X พูดภาษา L และ Y พูดภาษา L (นั่นคือ X และ Y พูดภาษาเดียวกัน) และ X กับ Y ไม่ใช่บุคคลคนเดียวกัน

การประมวลผลโปรแกรมเป็นการสรุปความจริงใหม่ๆ จากความจริงเดิมที่มีอยู่ การสรุปความจริงใหม่นี้เรียกว่า *การทำนิรนัย* (deduction) เช่น จากข้อความสองข้อความข้างต้น เราสามารถทำนิรนัยเพื่อให้ได้ความจริงใหม่ที่เป็นกฎระบุว่า Mary สามารถคุยกับ Y ถ้า Y พูดภาษาอังกฤษได้ และ Y ไม่ใช่ตัว Mary เองซึ่งเขียนอยู่ในรูปแบบข้อความฮอร์นได้ดังนี้

```
talkswith(Mary, Y) ← speaks(Mary, English), speaks(Y, English),
                     Mary ≠ Y
```

การทำนิรนัยเพื่อให้ได้ความจริงใหม่ๆ จะใช้วิธีการที่เรียกว่า การทำเรโซลูชัน (resolution) ซึ่งเป็นการพิสูจน์โดยวิธีปฏิเสธ (refutation) หรือเรียกอีกอย่างได้ว่า เป็นการพิสูจน์แย้ง (contradiction)

การสร้างข้อสรุปที่เป็นความจริงใหม่จากความจริงเดิมที่มีอยู่ ความจริงใหม่ที่สรุปได้จะเรียกว่าเป็นความจริง (ไม่ว่าจะเป็นแบบมีเงื่อนไขหรือไม่มีเงื่อนไขก็ตาม) ได้ก็ต่อเมื่อได้รับการพิสูจน์แล้วว่าจริงในทุกกรณี (เรียกว่า valid) เช่น ในข้อความที่เป็นการสรุปความจริงใหม่

```
talkswith(Mary, Y) ← speaks(Mary, English), speaks(Y, English),
                     Mary ≠ Y
```

ข้อความนี้จะเรียกว่าความจริงได้ก็ต่อเมื่อ เราสามารถพิสูจน์ได้ว่าข้อความเป็นจริงกับทุกค่าของ Y เมื่อ Y หมายถึงบุคคลใดๆ ในโลก นั่นหมายถึงเราต้องพิสูจน์หาความจริงกับประชากรทุกคนในโลก (ที่อาจจะมีมากถึงห้าพันล้านคน) ซึ่งเป็นการพิสูจน์ที่ทำได้ยากหรือในบางครั้งอาจจะทำไม่ได้เลย วิธีที่ทำได้ง่ายกว่าคือการพิสูจน์แย้ง นั่นคือ สมมุติให้ความจริงใหม่ที่สรุปได้มีค่าเป็นเท็จ จากนั้นใช้การพิสูจน์ทางตรรกะซึ่งเป็นการให้เหตุผลของความจริงใหม่ (ที่สมมุติให้เป็นเท็จ) ร่วมกับความจริงเดิมที่มีอยู่ ถ้าได้ข้อสรุปว่าความจริงใหม่ (ที่

สมมุติให้เป็นเท็จ) เมื่อพิจารณาร่วมกับความจริงเดิมที่มีอยู่ให้ผลลัพธ์เป็นเท็จเสมอ นั่นแปลว่า ความจริงใหม่ที่สมมุติให้เป็นเท็จมีความเป็นเท็จเสมอ (ปฏิเสธซ้อนปฏิเสธ) หรือพูดได้อีกอย่างว่าความจริงใหม่เป็นจริงเสมอ

ตัวอย่างการพิสูจน์ความจริงใหม่ที่สรุปมาจากความจริงเดิมที่มีอยู่ด้วยวิธีเรโซลูชัน แสดงได้ดังนี้

ความจริงเดิม(1): p_1

ความจริงเดิม(2): $p_1 \supset q_1$

ความจริงเดิม(3): $q_1 \supset q_2$

ความจริงใหม่ที่สรุปได้ (4): q_2

<u>การพิสูจน์ด้วยวิธีเรโซลูชัน</u>

การสรุปความจริงใหม่ความจริงเดิม: $(1) \wedge (2) \wedge (3) \supset (4)$

วิธีพิสูจน์ขัดแย้ง: สมมุติให้การสรุปเป็นเท็จ นั่นคือ $\neg((1) \wedge (2) \wedge (3) \supset (4))$

ถ้าข้อสมมุติเป็นเท็จในทุกกรณี นั่นคือ $\neg((1) \wedge (2) \wedge (3) \supset (4)) = \text{FALSE}$

ดังนั้น $\neg(\neg(1) \wedge (2) \wedge (3) \supset (4))) = \neg\text{FALSE}$

$= \text{TRUE}$

นั่นคือ $(1) \wedge (2) \wedge (3) \supset (4)$ เป็นจริง

ดังนั้น (4) เป็นความรู้ใหม่ที่สรุปได้จากความรู้เดิม (1) และ (2) และ (3)

ขั้นตอนการพิสูจน์ขัดแย้ง:

$\neg((1) \wedge (2) \wedge (3) \supset (4))$

$= \neg(\neg(1) \vee \neg(2) \vee \neg(3) \vee (4))$

$= (1) \wedge (2) \wedge (3) \wedge \neg(4)$

(1) คือ p_1

(2) คือ $\neg p_1 \vee q_1$

(3) คือ $\neg q_1 \vee q_2$

(4) คือ q_2

$= \underline{p_1 \wedge (\neg p_1 \vee q_1)} \wedge (\neg q_1 \vee q_2) \wedge \neg q_2$

$= \underline{(p_1 \wedge \neg p_1)} \vee (p_1 \wedge q_1) \wedge (\neg q_1 \vee q_2) \wedge \neg q_2$

$= \underline{\text{FALSE} \vee (p_1 \wedge q_1)} \wedge (\neg q_1 \vee q_2) \wedge \neg q_2$

$= p_1 \wedge q_1 \wedge \underline{(\neg q_1 \vee q_2) \wedge \neg q_2}$

$= p_1 \wedge q_1 \wedge (\neg q_1 \wedge \neg q_2) \vee \underline{(q_2 \wedge \neg q_2)}$

$= p_1 \wedge q_1 \wedge \underline{(\neg q_1 \wedge \neg q_2) \vee \text{FALSE}}$

$= p_1 \wedge \underline{q_1 \wedge \neg q_1} \wedge \neg q_2$

$= \underline{p_1 \wedge \text{FALSE} \wedge \neg q_2}$

= FALSE

ดังนั้น $\neg\neg((1) \wedge (2) \wedge (3) \supset (4))$ เป็นจริง และแสดงว่า (4) เป็นความจริงใหม่ที่สรุปได้จากความจริงเดิม (1) และ (2) และ (3)

ขั้นตอนการพิสูจน์ขัดแย้งข้างต้นสามารถใช้ขั้นตอนที่สั้นลงด้วยเทคนิคเรโซลูชัน ดังนี้

$p_1 \wedge (\neg p_1 \vee q_1) \wedge (\neg q_1 \vee q_2) \wedge \neg q_2$

$= q_1 \wedge \underline{(\neg q_1 \vee q_2) \wedge \neg q_2}$

$= \underline{q_1 \wedge \neg q_1}$

= FALSE

เทคนิคเรโซลูชันใช้วิธีการยุบรวมเทอมสองเทอมที่พรอพโพซิชันภายในมีเครื่องหมายต่างกัน เช่น เทอม p_1 และเทอม $\neg p_1 \vee q_1$ พรอพโพซิชัน p_1 และ $\neg p_1$ มีเครื่องหมายต่างกันจึงสามารถยุบรวมเทอม แล้วตัด p_1, $\neg p_1$ ออกเหลือเพียง q_1 การทำเช่นนี้ได้เนื่องจาก

$$p_1 \wedge (\neg p_1 \vee q_1) \supset q_1$$

ซึ่งเป็น implication ที่เป็นจริงเสมอ สามารถพิสูจน์ด้วยตารางค่าความจริง ดังนี้

p_1	q_1	$\neg p_1 \wedge q_1$	$p_1 \vee (\neg p_1 \vee q_1)$	$p_1 \vee (\neg p_1 \vee q_1) \supset q_1$
T	T	T	T	T
T	F	F	F	T
F	T	T	F	T
F	F	T	F	T

การพิสูจน์ขัดแย้งด้วยเทคนิคเรโซลูชันสามารถเขียนอยู่ในรูปของแผนภาพได้ดังนี้

$p_1 \qquad \neg p_1 \vee q_1$

$q_1 \qquad \neg q_1 \vee q_2$

$q_2 \qquad \neg q_2$

$\square$

เครื่องหมาย $\square$ ใช้แทนความหมาย FALSE

ในกรณีของแคลคูลัสเพรดิเคต เทอมแต่ละเทอมหรือแต่ละเพรดิเคตสามารถมีตัวแปรภายในเพรดิเคต เช่น

ความจริงเดิม (1): speaks (Mary, English)

ความจริงเดิม (2): speaks (Bob, English)

ความจริงเดิม (3): talkswith (X, Y) ← speaks(X,L), speaks(Y,L), X ≠ Y

การใช้เทคนิคเรโซลูชัน เพื่อพิสูจน์ความรู้ที่สรุปได้ใหม่ talkswith (Mary, Bob) จะต้องมีขั้นตอนที่เพิ่มขึ้นคือ การแทนค่าตัวแปรเพื่อให้เพรดิเคตสองเพรดิเคตมีรูปแบบที่เหมือนกัน เรียกขั้นตอนนี้ว่า *การทำยูนิฟิเคชัน* (unification)

การใช้เทคนิคเรโซลูชันเพื่อพิสูจน์ talkswith (Mary, Bob) ว่าเป็นความจริงที่สืบเนื่องมาจากการที่ (1) และ (2) และ (3) เป็นจริงแสดงเป็นแผนภาพได้ดังนี้

¬ talkswith(Mary, Bob) ¬ speaks(X,L) ∨ ¬ spesks(Y,L) ∨ ¬ (x≠y) ∨ talkswith (X,Y)

X = Mary
Y = Bob

¬ speaks(Mary, L) ∨ ¬ speaks(Bob, L) ∨ ¬ (Mary≠Bob) speaks(Mary, English)

L = English

¬ speaks(Bob,English) ∨ ¬ (Mary≠Bob) speaks(Bob, English)

¬ (Mary≠Bob)

□

การที่ผลลัพธ์เป็น FALSE แสดงว่าความจริงที่สรุปได้ใหม่ talkswith(Mary, Bob) เป็นจริง ในขั้นตอนการพิสูจน์ด้วยเรโซลูชันจะต้องมีการทำยูนิฟิเคชัน หรือการทำให้เพรดิเคตมีรูปแบบเดียวกันด้วยการแทนตัวแปรด้วยค่าคงที่ เช่น ทำยูนิฟิเคชัน talkswith(X, Y) กับ ¬ talkswith(Mary, Bob) ด้วยการแทนค่า X ด้วย Mary และแทน Y ด้วย Bob

การทำยูนิฟิเคชันของเพรดิเคตสองเพรดิเคตจะต้องอาศัยวิธี *การแทนค่าตัวแปร* (substitution) การระบุการแทนค่าอาจเขียนได้ว่า X = Mary, Y = Bob หรืออาจจะเขียนด้วยสัญลักษณ์ {Mary/X, Bob/Y}ค่าที่จะแทนลงในตัวแปร เป็นไปได้ 3 กรณี คือ ค่าคงที่, ตัวแปรอื่น, และฟังก์ชัน การทำยูนิฟิเคชันจัดเป็นกระบวนการจับคู่แพทเทิร์น (pattern matching) ในการทำโปรแกรมเชิงตรรกะยูนิฟิเคชันเป็นกลไกสำคัญที่ภาษา Prolog ใช้ช่วยในการประมวลผล

10.4 หลักการพื้นฐานของภาษาโปรล็อก

(Fundamentals of Prolog)

ข้อความต่างๆ ในภาษาโปรล็อกประกอบขึ้นจากเทอม เทอมสามารถเป็นค่าคงที่, ตัวแปร, หรือโครงสร้าง ค่าคงที่อาจจะเป็นอะตอมหรือเป็นจำนวนเลข อะตอม คือ ข้อความที่เขียนเริ่มต้นด้วยตัวพิมพ์เล็ก เช่น tiger, zebra, a, big_bear หรือเป็นข้อความที่เขียนอยู่ในเครื่องหมายคำพูดและอาจเริ่มต้นด้วยตัวพิมพ์ใหญ่ เช่น ‘Bob’,’Mary’,’My Tiger’ ค่าคงที่ประเภทจำนวนเลขอาจเป็นเลขจำนวนเต็มหรือเลขจำนวนจริง ตัวแปรประกอบขึ้นจากตัวอักษร (A-Z, a-z, _) โดยตัวอักษรตัวแรกจะต้องเป็นตัวพิมพ์ใหญ่ เช่น X, Y, Bob โครงสร้าง (structure) หรือ บางครั้งเรียกว่า ความสัมพันธ์ (relation) คือ เพรดิเคตที่มีจำนวนอาร์กิวเมนต์ 0, 1, หรือมากกว่า 1 ตัวอย่างของความสัมพันธ์ ได้แก่

```
n(zebra)
speaks(Who, russian)
np(X,Y)
```

การเขียนโปรแกรมในภาษาโปรล็อก คือ การประกาศข้อความที่เป็นจริง ซึ่งมีอยู่ 2 ประเภท คือ ข้อความที่เป็นจริงแบบไม่มีเงื่อนไข เรียกว่า **ความจริง** (fact) และข้อความที่เป็นจริงเมื่อเงื่อนไขทั้งหมดที่ระบุเป็นจริง เรียกว่า **กฎ** (rule) ตัวอย่างต่อไปนี้แสดงความจริงและกฎพร้อมทั้งหมายเหตุ (ภาษาโปรล็อกใช้เครื่องหมาย % เป็นข้อความหมายเหตุ) อธิบายความหมายของความจริงและกฎแต่ละข้อ

```
% Example of Prolog program
% Every fact and rule must end with a period
% Facts: eats(x, y).  means x eats y

eats (fred, meat).
eats (wilma, meat).
eats (betty, vegetables).
eats (wilma, vegetables).
eats (barney, meat).
eats (barney, vegetables).
```

```
% Rules:
% Symbol:- in rule is equivalent to ← in Horn clause.
carnivore(Individual):- eats(Individual, meat).
carnivore(Individual):- eats(Individual, meat),
                        eats(Individual, vegetables).
food(Thing):- eats(Individual, Thing).
% First rule means: If someone eats meat then they are a carnivore.
% Second rule means: If someone eats meat and vegetables then they are an omnivore.
% Third rule means: If someone eats X then X is food.
```

การเขียนความจริงและกฎ จะใช้รูปแบบของฮอร์น รูปแบบของกฎจะเป็น

$$term :- term_1, term_2, \ldots, term_n .$$

ซึ่งตรงกับรูปแบบฮอร์น คือ

$$term \leftarrow term_1, term_2, \ldots, term_n .$$

ความจริง (fact) อาจพิจารณาได้ว่าคือกฎ ที่มีเฉพาะส่วนหัวโดยไม่มีส่วนบอดี้

การรันโปรแกรมโปรล็อก คือ การถามคำถามเกี่ยวกับความจริงและกฎที่ประกาศไว้ในตัวโปรแกรม เช่น การถามคำถาม

Is meat a food?
Is Fred a carnivore?
Is dirt a food?
Is are the omnivore?

การประมวลผลโปรแกรมจะเป็นการค้นหาจากความจริงและกฎทั้งทั้งหลายในโปรแกรมที่เกี่ยวข้องกับคำถาม เพื่อพยายามพิสูจน์ด้วยวิธีเรโซลูชัน (ร่วมกับเทคนิคยูนิฟิเคชัน) ว่าข้อความที่ปรากฏในคำถามเป็นความจริงใหม่ที่สามารถสรุปจากความจริงและกฎในโปรแกรม หรือไม่ถ้าใช่จะได้คำตอบ Yes (พร้อมค่าตัวแปรที่เมื่อแทนด้วยค่านี้แล้วทำให้ได้คำตอบ Yes) แต่ถ้า คำถามซึ่งถือเป็นความจริงใหม่ที่ระบุโดยผู้ใช้ไม่สามารถสรุปได้จากความจริงและกฎในโปรแกรม จะได้คำตอบ No

ก่อนที่จะสั่งรันโปรแกรมได้จะต้องพิมพ์โปรแกรมด้วยเอดิเตอร์ แล้วบันทึกโปรแกรมเป็นไฟล์ที่มีนามสกุล .pl เช่น flint.pl จากนั้นใช้คอมไพเลอร์หรืออินเตอร์พรีตเตอร์ ภาษาโปรล็อกสั่งรันโปรแกรมด้วยคำสั่ง consult(flint.pl) หรือใช้รูปแบบ [‘flint.pl’] ตัวอย่างต่อไปนี้แสดงการสั่งรันโปรแกรม และการตั้งคำถามเพื่อให้โปรแกรมค้นหาคำตอบ (เครื่องหมายพร้อมรับคำสั่ง จะเป็นเครื่องหมายคำถาม ?-)

```
?-['flint.pl'].
flint.pl  compiled, 0.00 sec, 1,512 bytes.
Yes

?-food(meet).
Yes

?-carnivore(fred)
Yes


?-food(dirt).
No

?-omnivore(Indivivual).
Individual = wilma
Yes
```

การจบการทำงาน กับโปรแกรมโปรล็อกจะใช้คำสั่ง halt.(แล้วกดแป้น return)

```
?-halt.
```

<u>เรโซลูชันและยูนิฟิเคชัน</u>

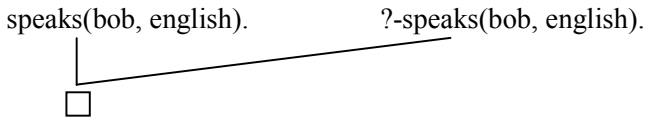
เมื่อสร้างโปรแกรมโปรล็อกและคอมไพล์โปรแกรมแล้ว การกระตุ้นให้โปรแกรมทำงานทำโดยการตั้งคำถาม การทำงานของโปรแกรมจะเป็นการพยายามพิสูจน์ด้วยวิธีเรโซลูชันว่าคำถามเป็นสิ่งที่สรุปความจากความจริงทั้งหลายที่มีอยู่ในโปรแกรมหรือไม่ กระบวนการทำงานของภาษาโปรล็อกจึงเป็นการค้นหาเทอมที่ปรากฏใน facts และ เทอมที่ปรากฏที่ส่วนหัวของ rules เพื่อตรวจสอบว่ามีเทอมใดที่สามารถนำมาใช้ในการทำเรโซลูชันเพื่อตอบข้อคำถามของผู้ใช้ ตัวอย่างโปรแกรมต่อไปนี้เป็นโปรแกรมที่จะใช้ประกอบการแสดงขั้นตอนการทำงานของภาษาโปรล็อก

```
speaks(allen, russian).
speaks(bob, english).
speaks(mary, english).
talkswith(Person1, Person2):-speaks(Person1, L),
                             speaks(Person2, L),
                             Person1 =\= Person2.
```

ถ้าผู้ใช้เริ่มสั่งให้โปรแกรมทำงานด้วยการตั้งคำถาม “Bob สามารถพูดภาษาอังกฤษได้ใช่หรือไม่?” รูปแบบคำถามนี้ในภาษาโปรล็อกจะเป็น

```
?-speaks(bob, english).
```

เมื่อตัวประมวลผลโปรล็อกรับคำถามจากผู้ใช้ กระบวนการพิสูจน์ด้วยเรโซลูชัน จะเริ่มต้นทำงานด้วยการค้นหาเทอมที่ตรงกับคำถาม ซึ่งจะได้เทอมในบรรทัดที่สองของโปรแกรมที่มีทั้งชื่อเพรดิเคต และอาร์กิวเมนต์ของเพรดิเคตที่ตรงกัน ทำให้สามรถจบการพิสูจน์ได้ (รูปที่ 10.4)



รูปที่ 10.4 แผนภาพแสดงการพิสูจน์คำถามด้วยวิธีเรโซลูชัน

เมื่อผลการพิสูจน์เป็น FALSE (แทนด้วย) แสดงว่า ด้วยความจริงเดิมทั้งหมดที่ปรากฏในโปรแกรม ข้อความที่ระบุในคำถาม (?-speaks(bob, english).) เป็นความจริงใหม่ที่สามารถสรุปได้จากคว้ามจริงเดิมที่มีอยู่

รูปแบบของข้อความแบบฮอร์นที่ใช้ในโปรแกรมภาษาโปรล็อก ใช้รูปแบบ

$$\text{heal} :- \text{term}_1, \text{term}_2$$

ซึ่งจะตรงกับรูปแบบเพรดิเคตแคลคูลัสดังนี้

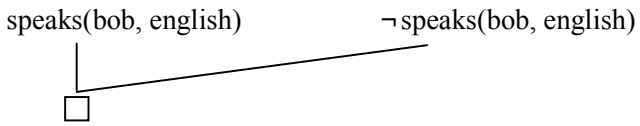
$$\text{term}_1 \wedge \text{term}_2 \supset \text{head}$$

และสามารถแปลงรูปแบบได้เป็น

$$\neg(\text{term}_1 \wedge \text{term}_2) \vee \text{head}$$

$$= \neg\text{term}_1 \wedge \neg\text{term}_2 \vee \text{head}$$

ดังนั้นเทอมที่ปรากฏที่ส่วนหัวของกฎ จะเป็นเทอมที่เป็นบวก (เรียกว่า positive literal) และเทอมนี้ส่วนบอดี้ของกฎ จะเป็นเทอมที่เป็นนิเสธ (เรียกว่า negative literal) ในโปรแกรมภาษาโปรล็อกจะปรากฏข้อความอยู่ 3 ประเภท คือ **ความจริง** (fact), **กฎ**(rule) และ **คำถาม**(query) ความจริงคือข้อความแบบฮอร์นที่ไม่มีส่วนบอดี้ (แสดงว่ามีเฉพาะ positive literal) กฎคือข้อความแบบฮอร์นที่มีครบทั้งส่วนหัวและส่วนบอดี้ และคำถาม คือข้อความแบบฮอร์นที่มีเฉพาะส่วนบอดี้ (แสดงว่ามีเฉพาะ negative literal) ดังนั้นการพิสูจน์ดังแผนภาพในรูปที่ 10.4 ข้างต้นจึงเป็นการทำเรโซลูชันระหว่าง positive literal (ความจริง) และ negative literal (คำถาม) ดังนี้



ข้อความแต่ละบรรทัดในโปรแกรมโปรล็อกจะเชื่อมกันด้วย ความหมาย AND ดังนั้นโปรแกรมข้างต้นจะมีความหมายดังนี้

```
speaks(allen, russian) ∧ speaks(bob, russian) ∧…..
```

การทำเรโซลูชันข้างต้นจึงต้องมีความหมายเทียบเท่ากับ

```
speaks(bob, english) ∧ ¬speaks(bob, english)
```

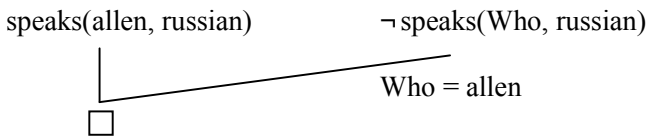
ซึ่งจะได้ผลลัพธ์เป็น FALSE ดังนั้นการประมวลผลคำถามจึงได้คำตอบเป็น Yes

```
?-speaks(bob, english).
Yes
```

ในกรณีที่คำถามปรากฏตัวแปร เช่น

```
?-speaks(Who,russian).
```

ในการประมวลผลด้วยการพยายามพิสูจน์ขัดแย้งหาผลสรุปเป็น FALSE (เพื่อให้คำตอบ Yes) จะต้องมีการแทนค่าตัวแปร Who เพื่อพยายามหาข้อความในโปรแกรมที่ตรงกันกับข้อความในคำถาม การค้นหาจะกระทำเรียงลำดับตั้งแต่บรรทัดที่ 1 ในโปรแกรมลงมาจนถึงบรรทัดสุดท้าย ดังนั้นข้อความหรือเทอมที่ถูกนำมาใช้ในการพิสูจน์ด้วยเรโซลูชันจะเป็นดังนี้



ดังนั้นตัวประมวลผลโปรล็อก จะแสดงคำตอบที่จอภาพให้ผู้ใช้ทราบดังนี้

```
?-speaks(Who, russian)
Who = allen
```

ผู้ใช้สามารถพิมพ์เครื่องหมาย ";" เพื่อบอกตัวประมวลผลให้ค้นหาคำตอบอื่นอีกที่เป็นไปได้ เมื่อได้รับคำสั่ง ";" ตัวประมวลผลจะยกเลิกการแทนค่าตัวแปร Who ด้วยค่าคงที่ allen และค้นหาข้อความต่อไปในโปรแกรมที่มีแพทเทิร์นเป็น speaks(…, russian) ซึ่งจะได้ข้อความในบรรทัดที่ 3 และนำมาพิสูจน์ด้วยเรโซลูชันดังนี้

speaks(mary, russian) ¬speaks(Who, russian)

Who = mary

□

ดังนั้นผลลัพธ์ที่ปรากฏที่จอภาพจะเป็นดังนี้

```
?-speaks(Who,russian)
Who = allen;
Who = mary;
No
```

การใช้คำสั่ง ";" ครั้งที่สองได้ผลลัพธ์เป็น No เนื่องจากในโปรแกรมไม่มีความจริงอื่นอีกที่ระบุความสัมพันธ์ว่ามีใครที่สามารถพูดภาษารัสเซีย

ในกรณีที่คำถามเป็นการถามที่ต้องใช้กฎในโปรแกรมเพื่อทำเรโซลูชัน ดังตัวอย่างคำถามต่อไปนี้ที่เป็นการถามว่า "Bob สามารถพูดสื่อสารกับ Mary ได้หรือไม่?"

```
?-talkswith(bob, mary).
```

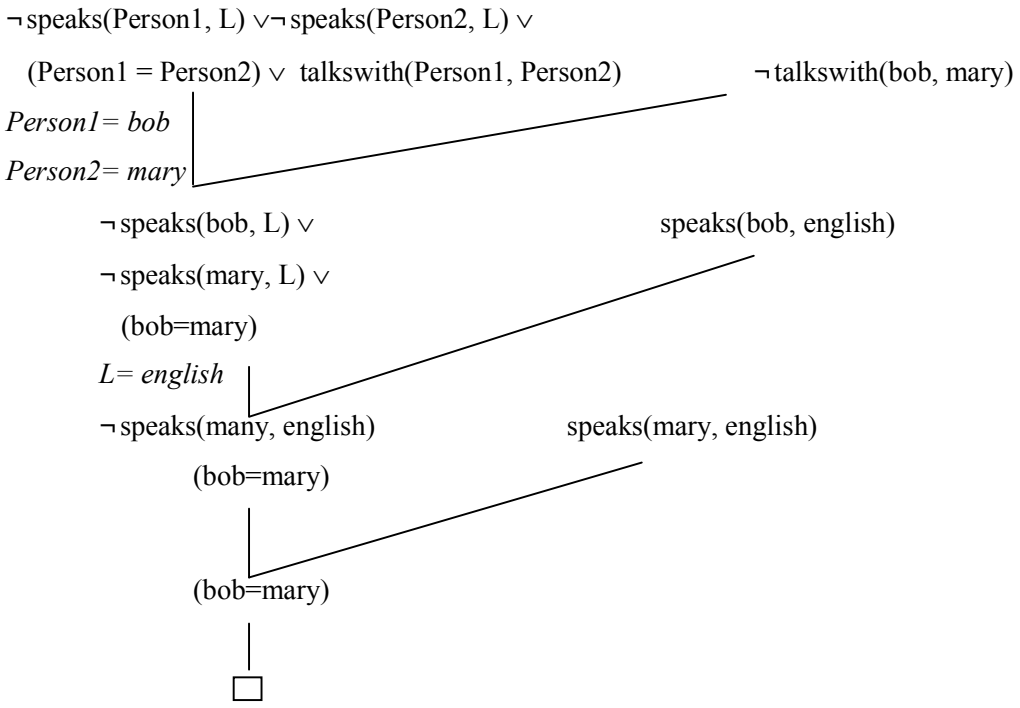
การทำงานของตัวประมวลผลจะเริ่มด้วยการค้นหาเทอม หรือ ชื่อเพรดิเคต ที่ตรงกับคำถาม (คือ ชื่อเพรดิเคต talkswith) ซึ่งในโปรแกรมมีข้อความเดียว คือ ข้อความที่เป็นกฎ

```
talkswith(Person1, Person2):-speaks(Person1, L),
                             speaks(Person2,L),
                            Person1 =\= Person2
```

ซึ่งถ้าเขียนอยู่ในรูปแบบแคลคูลัสเพรดิเคตจะเป็น

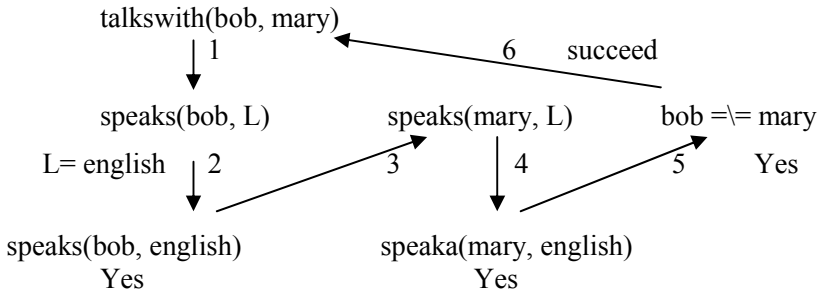
$$\neg speaks(Person1, L) \vee \neg speaks(Person2, L) \vee \neg(Person1 \neq Person2) \vee talkswith(Person1, Person2)$$

เมื่อนำข้อความนี้มาพิสูจน์เรโซลูชันร่วมกับคำถามที่เป็นข้อความ ¬talkswith(bob, mary) จะได้ผลการพิสูจน์ดังแสดงในรูปที่ 10.5



รูปที่ 10.5 ขั้นตอนการพิสูจน์ด้วยวิธีเรโซลูชันเมื่อคำถามเกี่ยวข้องกับกฎ

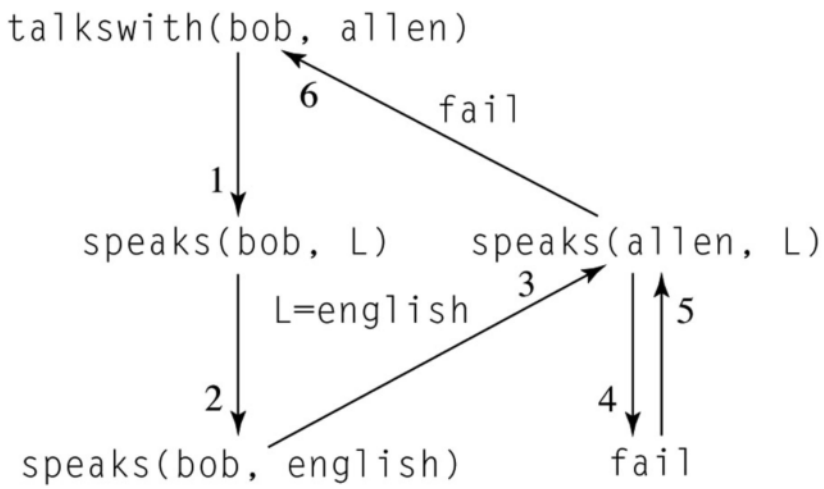
จากรูปจะสังเกตได้ว่าเมื่อคำถามใช้เพรดิเคต `talkswith` ตัวประมวลผลจะไปค้นหาเพรดิเคตที่ตรงกันซึ่งจะเป็นเพรดิเคตส่วนหัวของกฎ `talkswith(Person1, Person2):-...`จากนั้นในขั้นตอนแรกของการพิสูจน์จะเป็นการทำยูนิฟิเคชัน คือทำให้ `talkswith(Person1,Person2)` มีรูปแบบตรงกันกับ `talkswith(bob, mary)` จากนั้นทั้งสองเพรดิเคตนี้จะถูกตัดออกไป (เนื่องจาก `talkswith(bob, mary)` ∧ ¬`talkswith(bob, mary)` ให้ค่าFALSE) จะเหลือเฉพาะส่วนบอดี้ของกฎที่ตัวประมวลผลจะต้องไปค้นหาในโปรแกรมเพื่อหาข้อความที่เพรดิเคตตรงกันมาทำเรโซลูชันต่อไป การทำยูนิฟิเคชันและเรโซลูชันจะกระทำทีละเทอมและทำเรียงลำดับจากเทอมที่1ในส่วนบอดี้ แล้วจึงไปทำยูนิฟิเคชันและเรโซลูชันกับเทอมที่สองไปเป็นลำดับจนจบ ดังนั้นการแสดงขั้นตอนการทำเรโซลูชันจึงมักนิยมแสดงด้วยขั้นตอนที่สั้นลงและใช้ข้อความในรูปแบบฮอร์นแทนที่จะเป็นรูปแบบแคลคูลัสเพรดิเคต ดังรูปที่ 10.6 (ความหมายในรูปที่ 10.6 จะไม่ต่างจากรูปที่ 10.5 เพียงแต่การแสดงขั้นตอนจะกระชับขึ้น)



รูปที่ 10.6 ลำดับขั้นตอนการทำงานของโปรล็อกเพื่อพยายามพิสูจน์ความจริงของคำถาม *?-talkswith(bob, mary)*

ถ้าคำถามเปลี่ยนเป็น ?-talkswith(bob, allen) ในครั้งนี้คำถามจะไม่เป็นจริงเพราะ Bob ไม่สามารถพูดสื่อสารกับ Allen ขั้นตอนการทำงานของโปรล็อกแสดงได้ดังรูปที่ 10.7 แต่ในครั้งนี้เมื่อทำยูนิฟิเคชันกับบอดี้เทอมที่สอง speaks(allen, english) จะได้ผลลัพธ์เป็น fail คือไม่สามารถหาความจริงในโปรแกรมที่ตรงกับเทอมนี้ได้ การประมวลผลจะหยุดและให้คำตอบเป็น No โดยไม่ต้องทำงานกับเทอมที่เหลือในบอดี้ (คือ เทอม bob\= allen) ข้อความที่ปรากฏบนจอภาพจะเป็นดังนี้

```
?-talkswith(bob, allen)
No
```



รูปที่ 10.7 ขั้นตอนการทำงานของโปรล็อกเพื่อพิสูจน์ความจริงของคำถาม *?-talkswith(bob, allen)*

<u>การย้อนกลับ</u>

การย้อนกลับ (backtracking) จะเกิดขึ้นเมื่อในขั้นตอนพิสูจน์ความจริงของคำถามมีกฎที่เกี่ยวข้องมากกว่าหนึ่งกฎ โปรล็อกจะใช้กฎแรกก่อนในกระบวนการพิสูจน์ด้วยเรโซลูชัน ถ้ากฎแรกไม่ได้ผล (เรียกว่า fail เพราะไม่สามารถทำเรโซลูชันจนกระทั่งได้ค่า FALSE) โปรล็อกจะย้อนกลับเพื่อพยายามพิสูจน์ใหม่ด้วยกฎที่สอง และในช่วงของการย้อนกลับตัวแปรที่เคยถูกแทนค่าจะยกเลิกการแทนค่านั้น (ภาษาโปรล็อกไม่อนุญาตให้มีการเปลี่ยนค่าตัวแปร เช่น จาก x ที่มีค่า 10 จะเปลี่ยนเป็น x+1 เพื่อให้มีค่า 11 ไม่ได้ แต่สามารถยกเลิกค่าเดิมแล้วให้ x มีค่าใหม่ได้ ซึ่งกระบวนการยกเลิกค่าตัวแปรนี้เป็นกลไกภายในของภาษาโปรล็อกโปรแกรมเมอร์ไม่สามารถระบุคำสั่งควบคุมโดยตรงได้)

ตัวอย่างโปรแกรมต่อไปนี้ แสดงข้อมูลความสัมพันธ์ของบุคคลในครอบครัว มีการใช้กฎเพื่อนนิยามความสัมพันธ์ parent(A, B) ว่า A เป็น parent ของ B ได้ก็ต่อเมื่อ A เป็นพ่อของ B หรือ A เป็นแม่ของ B

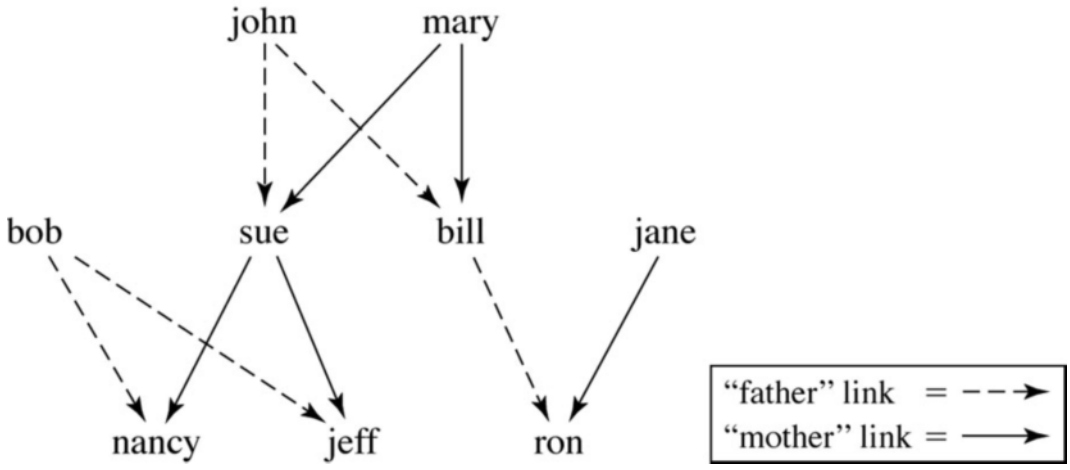
ความสัมพันธ์ parent นี้นิยามด้วยกฎสองข้อ นอกจากนี้มีการนิยามความสัมพันธ์ grandparent(A, C) ซึ่งเป็นการนิยามความสัมพันธ์ ปู่, ย่า, ตา หรือ ยาย โดย A จะเป็นปู่/ย่า/ตา/ยาย ของ C ได้ ก็ต่อเมื่อ A เป็นพ่อหรือแม่ของ B แล้ว B ไปเป็นพ่อหรือแม่ของ C ความสัมพันธ์ของบุคคลในครอบครัวนี้แสดงได้เป็นแผนภาพได้ดังรูปที่ 10.8

```
mother(mary, bill).
mother(mary, sue).
mother(sue, nancy).
mother(sue, jiff).
mother(jane, ron).

father(john, sue).
father(john, bill).
father(bob, nancy).
father(bob, jiff).
father(bill, ron).

parent(A, B) :- father(A, B).
parent(A, B) :- mother(A, B).

grandparent(A, C) :- parent(A, B), parent(B, C)
```



รูปที่ 10.8 แผนภาพแสดงโครงสร้างความสัมพันธ์ของบุคคลในครอบครัว

ถ้ามีการถามว่า Mary เป็น grandparent ของ Ron ใช่หรือไม่?

```
?-grandparent(mary, ron)
```

กระบวนการทำงานของโปรล็อกจะเริ่มต้นจากเพรดิเคตที่เป็นคำถาม แล้วพยายามค้นหาเพรดิเคตในโปรแกรมที่ตรงกัน ถ้าชื่อเพรดิเคตตรงกันแต่อาร์กิวเมนต์ภายในไม่ตรงกันจะทำยูนิฟิเคชันเพื่อแทนค่าตัวแปรให้ได้อาร์กิวเมนต์ที่ตรงกัน ต่อจากนั้นจะพิสูจน์ด้วยเรโซลูชันถ้ากระบวนการพิสูจน์ได้ FALSE จะตอบ Yes มิฉะนั้นจะตอบ No การพยายามพิสูจน์ด้วยการยึดเอาคำถามเป็นเป้าหมายของกระบวนการพิสูจน์ เรียกว่า **การพิสูจน์แบบย้อนกลับ** (backward chaining proof) ซึ่งเป็นวิธีที่ใช้ในภาษาโปรล็อก แต่ถ้ากระบวนการพิสูจน์เอาข้อมูลทั้งหมดในโปรแกรมเป็นจุดเริ่มต้นแล้วพยายามให้เหตุผลมาเป็นลำดับจนกระทั่งได้ความจริงใหม่ที่ตรงกับคำถาม กระบวนการพิสูจน์แบบนี้จะเรียกว่า **การพิสูจน์ไปข้างหน้า** (forward chaining proof)

ขั้นตอนการพิสูจน์แบบย้อนกลับเพื่อหาคำตอบให้กับคำถามข้างต้นแสดงเป็นลำดับได้ดังนี้

?-grand parent(mary, ron)

A=mary | 1

B=ron

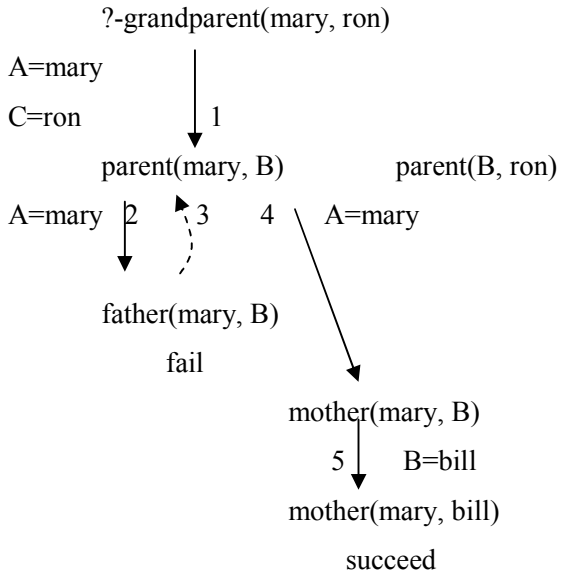
<u>parent(mary, B)</u> parent(B, ron)

A=mary | 2

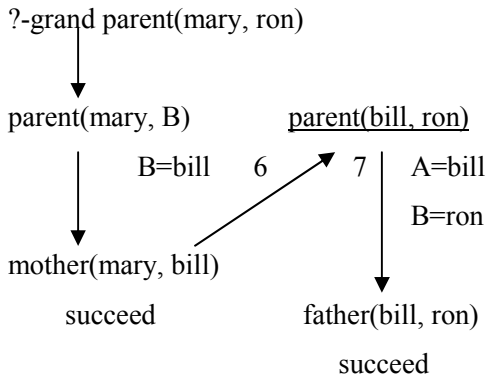
father(mary, B)

fail

ในขั้นตอนของการพิสูจน์ parent(mary, B) มีกฎที่นิยามเกี่ยวกับ parent สองข้อโปรล็อกจะทดลองใช้กฎแรกก่อนในการพิสูจน์ จากกระบวนการเรโซลูชันทำให้ได้เทอม father(mary, B) ซึ่งไม่สามารถทำยูนิฟิเคชันต่อไปได้ นั่นคือไม่สามารถหาเพรดิเคตจากในโปรแกรมที่ตรงกับเทอม father(mary,…) ทำให้การทำเรโซลูชันในขั้นตอนนี้ล้มเหลว แต่เนื่องจากกฎเกี่ยวกับ parent ยังมีอีกข้อที่ยังไม่ได้นำมาพิจารณา โปรล็อกจะมีการย้อนกลับพร้อมทั้งยกเลิกการแทนค่า A=mary ในช่วงที่ 2 ของการพิสูจน์(การย้อนกลับจะแทนด้วยเส้นประ) แล้วทำเรโซลูชันด้วยกฎ parent ข้อที่ 2 ดังนี้



การพิสูจน์ในช่วงแรกนี้สำเร็จเพราะสามารถหาความสัมพันธ์จากในโปรแกรมที่ตรงกับ mother(mary, B) โดย B จะต้องถูกแทนที่ด้วยค่าคงที่ bill ทำให้ในเทอมที่สองของบอดี้ของกฎ grand parent นั่นคือ เทอม parent(B, ron) จะต้องถูกแทนค่าไปพร้อมกันทำให้เทอมที่สองมีอาร์กิวเมนต์เป็น parent(mary, bill) และกระบวนการพิสูจน์ดำเนินต่อไปดังนี้



เมื่อกระบวนการพิสูจน์จบลงด้วยความสำเร็จ (succeed) คำตอบที่ได้จึงเป็น

```
?-grand parent(mary, ron)
Yes
```

<u>การใช้ฟังก์ชัน assert และ retract</u>

ในระหว่างการรันโปรแกรมโปรล็อกผู้ใช้สามารถเพิ่มข้อมูลใหม่ด้วยฟังก์ชัน assert ดังนี้

```
?-assert(sibling(x, y):-parent(w, x), parent(w, y),x =/= y)).
```

ความสัมพันธ์ sibling เป็นการนิยามความสัมพันธ์พี่น้องโดยระบุว่า x จะเป็นพี่น้องกับ y เมื่อ x และ y มี parent คนเดียวกัน และ x และ y ต้องไม่ใช่บุคคลคนเดียวกัน

เมื่อมีการเพิ่มข้อมูลและความสัมพันธ์ใหม่ให้กับโปรแกรม เราสามารถถามคำถามเกี่ยวกับ sibling ได้ดังนี้

```
?-sibling(ron, joe).
Yes
```

แต่ถ้าต้องการยกเลิกข้อมูลหรือความสัมพันธ์ สามารถใช้ฟังก์ชัน retract ได้ดังนี้

```
?-retract(mother(jane, ron)).
```

<u>การควบคุมการย้อนกลับด้วยคำสั่ง CUT</u>

การนิยามความสัมพันธ์ใดด้วยกฎมากกว่าหนึ่งข้อ ทำให้ขั้นตอนการประมวลผลของโปรล็อกจะต้องมีการย้อนกลับเพื่อไปทดลองพิสูจน์กับกฎข้ออื่นๆที่เหลือ โปรแกรมเมอร์สามารถควบคุมการทำงานของโปรล็อกให้ทำงานได้เร็วขึ้นโดยลดการย้อนกลับที่ไม่จำเป็น การควบคุมการย้อนกลับทำได้ด้วยฟังก์ชัน CUT ซึ่งแทนด้วยเครื่องหมาย “!”

ตัวอย่างโปรแกรมโปรล็อกต่อไปนี้แสดงการนิยามฟังก์ชัน f ว่า f จะให้ผลผลลัพธ์เป็น 0 ถ้าอินพุตมีค่าน้อยกว่า 3 ให้ผลลัพธ์เป็น 2 ถ้าอินพุตมีค่าตั้งแต่ 3 จนถึงน้อยกว่า 6 และกรณีสุดท้ายให้ผลลัพธ์เป็น 4 ถ้าอินพุตมีค่ามากกว่าหรือเท่ากับ 6

% Example of a binary relation f(x, y)

% Rule1: if x < 3 then y = 0

% Rule2: if 3 ≤ x < 6 then y = 2

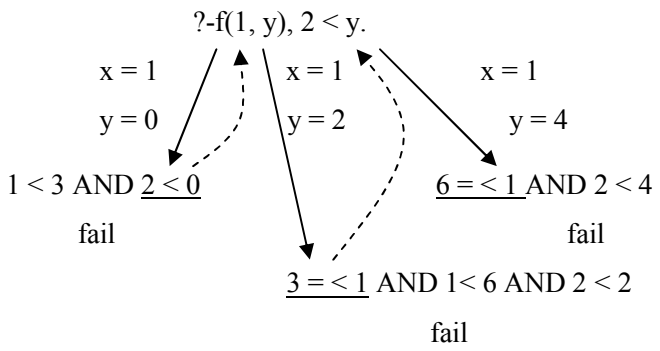
% Rule3: if 6 ≤ x then y = 4

```
f(x, 0) :- x < 3.
f(x, 2) :- 3 =< x, x < 6.
f(x, 4) :- 6 =< x.
```

ถ้ามีคำถามให้พิสูจน์เพื่อหาค่าของ y โดยระบุค่าอินพุตเป็น 1 และมีเงื่อนไขเพิ่มเติมว่าผลลัพธ์จะต้องมีค่ามากกว่า 2 (เครื่องหมาย “,” แทนความหมาย AND)

```
?-f(1, y), 2 < y.
```

กระบวนการประมวลผลจะพยายามพิสูจน์คำถามด้วยการใช้กฎข้อที่ 2 และถ้าไม่ประสบผลสำเร็จอีกจะย้อนกลับมาทดลองกฎข้อที่ 3 ซึ่งเป็นข้อสุดท้าย แสดงแผนภาพการค้นหาคำตอบได้ดังนี้

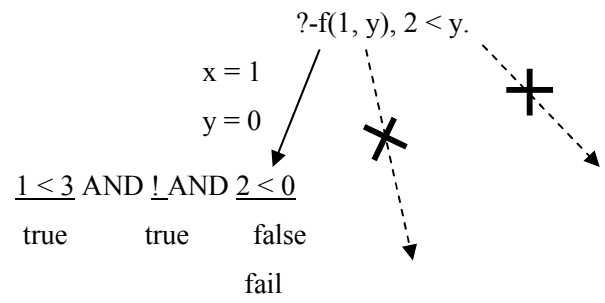


ผลการทดลองใช้กฎทั้ง 3 ข้อได้เป็น fail ทุกข้อทำให้คำตอบที่ได้เป็น No เพราะไม่มีความสัมพันธ์ f ข้อใดที่รับอินพุตเป็น 1 และให้ผลลัพธ์ที่มีค่ามากกว่า 2 (จากกฎทั้ง 3 ข้อ ถ้าอินพุตเป็น 1 ผลลัพธ์ก็ต้องเป็น 0)

การปรับปรุงโปรแกรมเพื่อให้ได้คำตอบที่ถูกต้องเหมือนเดิมแต่การประมวลผลเร็วขึ้นเพราะตัดขั้นตอนย้อนกลับไปทดลองกฎข้ออื่นๆ โดยไม่จำเป็น ทำได้โดยใช้เครื่องหมาย ! ดังนี้

```
f(x, 0) :- x < 3, !.
f(x, 2) :- 3 =< x, x < 6,!.
f(x, 4) :- 6 =< x.
```

การใช้เครื่องหมาย ! มีผลต่อการประมวลผลคำถาม ดังนี้



เครื่องหมาย ! จะถูกประมวลผลให้มีค่าเป็นจริงเสมอโดยผลข้างเคียงของ CUT จะทำให้การย้อนกลับไปทางเลือกอื่นๆ ของ f ถูกตัดทิ้ง

นิยามของฟังก์ชัน CUT คือถ้า ! ปรากฏในข้อความ เช่น

```
head :- term1, term2, ! , term3, term4.
```

ทางเลือกอื่นๆ ของ head, term1, term2 ที่ยังไม่ได้นำมาพิจารณาจะถูกตัดทางเลือกเหล่านั้นทิ้ง แต่ทางเลือกอื่น (ถ้ามี) ของ term3 และ term4 ไม่ถูกตัดทิ้ง

<u>การคำนวณทางคณิตศาสตร์</u>

ภาษาโปรล็อกมีฟังก์ชันพื้นฐานเพื่อใช้ในงานคำนวณทางคณิตศาสตร์ ดังนี้

- abs(X), sign(X), float(X), floor(X), ceiling(X), truncate(X), round(X), float-fractional-part(X), float-integer-part(X)
- max(X, Y), min(X, Y)
- random(X) ค่าสุ่มอยู่ระหว่าง 0 ถึง x-1
- sqrt(X), log(X), log10(X), exp(X)
- sin(X), asin(X), cos(X), acos(X), tan(X), atan(X)
- succ(X, Xplus), pred(X, Xminus) ฟังก์ชันเพื่อการเพิ่มค่าและลดค่า x

การคำนวณพื้นฐานและการเปรียบเทียบค่าจะเขียนด้วยรูปแบบนิพจน์อินฟิกซ์ดังนี้

- การคำนวณ : (X*Y), (X+Y), (X-Y), (X/Y), (X**Y)
- การหารเลขจำนวนเต็ม : (X//Y), (X mod Y), (X rem Y)

- การเปรียบเทียบค่า : (X < Y), (X > Y), (X =< Y), (X >= Y), (X =:= Y), (X =\= Y)
- การทำ bitwise : (X << Y), (X >> Y), (X ∨ Y), (X ∧ Y), (X xor Y), (\X)
- การใช้ is เพื่อคำนวณค่า : X is Y เช่น X is 4*2 ทำให้ตัวแปร X มีค่า 8 แต่ถ้าเขียน X = 4*2 จะเป็นการแทนตัวแปร X ด้วยเทอม 4*2 โดยไม่มีการคูณเกิดขึ้น

การรับค่าใช้คำสั่ง read(Term) และการแสดงผลใช้คำสั่ง write(Term) และ write_ln(Term) เช่น

```
?-read(X), Y is sqrt(X).
| 4
X = 4
Y = 2
Yes

?-write_ln('foo').
foo
Yes
```

<u>การทำงานกับโครงสร้างข้อมูลลิสต์</u>

โครงสร้างลิสต์จะเขียนอยู่ภายในวงเล็บ [ ] และสมาชิกแต่ละตัวภายในลิสต์จะแยกจากกันด้วยเครื่องหมาย "," เช่น [red, green, blue] โอเปอเรเตอร์ที่ทำงานกับลิสต์ คือ เครื่องหมาย "|" ใช้ในการแยกสมาชิกส่วนหัวของลิสต์ออกจากส่วนท้าย ซึ่งเป็นลิสต์ที่เหลือ เช่น

```
?-[red, green, blue] = [Head|Tail].
Head = red
Tail = [green, blue]
Yes

?-[red, green, blue] = [A, B|Tail].
A = red
B = green
Tail = [blue]
```

ลิสต์ว่างแทนด้วยสัญลักษณ์ [ ] และเครื่องหมายขีดล่าง (underscore) จะแทนความหมายว่าเป็นค่าอะไรก็ได้ เช่น [_, _, blue] เป็นการระบุแพทเทิร์นว่าเราต้องการลิสต์ที่มีสมาชิก 3 ตัว โดยกำหนดว่าสมาชิกตัวสุดท้ายต้องเป็น blue และสมาชิกสองตัวแรกเป็นอะไรก็ได้

ตัวอย่างโปรแกรมที่ทำงานกับลิสต์ ได้แก่โปรแกรม member ที่รับอินพุตเป็นค่าสมาชิกและลิสต์ จากนั้นโปรแกรมจะตรวจสอบว่าค่าสมาชิกที่ระบุเป็นสมาชิกอยู่ในลิสต์จริงหรือไม่

```
member(Element,  [Element | _ ]).
member(Element, [ _ | Tail ]) :- member(Element, Tail).
```

โปรแกรม member เป็นโปรแกรมที่เรียกตัวเองซ้ำ จึงต้องเริ่มต้นคำสั่งด้วยกรณีพื้นฐานซึ่งเป็นกรณีง่ายที่สุด นั่นคือ Element ที่รับอินพุตปรากฏเป็นสมาชิกตัวแรกในลิสต์ การเขียนลิสต์ในลักษณะ [Element | _] เป็นการระบุแพทเทิร์นของลิสต์ว่าตัวแรกคือ Element ส่วนที่เหลือเป็นอะไรก็ได้ และส่วนที่เหลือจะเป็นลิสต์ยาวเท่าไรก็ได้ ข้อความในบรรทัดที่สองของโปรแกรมเป็นขั้นตอนเรียกตัวเองซ้ำ นั่นคือ ถ้าสมาชิกตัวแรกในลิสต์ไม่ใช่ Element ให้ตัดสมาชิกตัวแรกออกจากการพิจารณา แล้วเรียกลิสต์ส่วนที่เหลือว่า Tail (ระบุความหมายนี้ด้วยแพทเทิร์น [_ | Tail]) จากนั้นเรียกโปรแกรม member ซ้ำด้วยคำสั่ง member(Element, Tail) สังเกตว่าการเรียกตัวเองซ้ำจะมีจุดสิ้นสุดเพราะแต่ละรอบของการเรียกตัวเองซ้ำ ลิสต์จะถูกตัดสมาชิกออกหนึ่งตัว จนในที่สุดถ้าถึงกรณีเลวร้ายสุดคือ ค่า Element ไม่ปรากฏในลิสต์เลย ลิสต์จะถูกทำให้สั้นลงจนเหลือเป็นลิสต์ว่าง โปรแกรมจะส่งคำตอบกลับเป็น false

ถ้าต้องการติดตามการเรียกตัวเองซ้ำ สามารถใช้คำสั่ง trace ได้ดังตัวอย่างต่อไปนี้

```
?-trace.
?-member(a, [b, c]).
call: member(a, [b, c]).
call: member(a, [c]).
call: member(a,[]).
fail: member(a,[]).
fail: member(a,[c]).
fail: member(a, [b, c]).
No
```

10.5 การทำโปรแกรมเชิงตรรกะในภาษาโปรล็อก

(Logic programming in Prolog)

ตัวอย่างที่ 1 แสดงการทำโปรแกรมเชิงตรรกะเพื่อบวกค่าจำนวนเลขในอาร์เรย์ (ในภาษาโปรล็อกใช้โครงสร้างลิสต์แทนอาร์เรย์)

```
%editor data.prolog
/*Read in data as a Prolog Relation*/
datavals(4,[1,2,3,4]).
%editor pgm.prolog
        go :- reconsult('data,prolog'),
                datavals(A,B),
                INSUM is 0,
                for(A,B,INSUM,OUTSUM),nl.
                write('sum ='),write(OUTSUM),nl.
        /*for loop executes 'I'times*/
        for(I,B,INSUM,OUTSUM):- not(I=0),
                B=[HEAD|TALL],
                write(HEAD),
                NEWVAL is INSUM+HEAD,
                for(I-1,TALL,NEWVAL,OUTSUM).
 /*If I is 0, return INSUM computed valued*/
        for(_,_,INSUM,OUTSUM):- OUTSUM = INSUM.
        not(X) :- X,!,fail.
        not(_).
%prolog
|?-consult('pgm.prolog').
{consulting/aaron/mvz/book/pgm.prolog…}
{/aaron/mvz/book/pgm.prolog consulted,30 msec 1456 bytes}
yes
|?-go
{consulting/aaron/mvz/book/data.prolog…}
{/aaron/mvz/book/data.prolog consulted, 10 msec 384 bytes}
1234
SUM =10
Yes
```

ตัวอย่างที่ 2 แสดงการทำโปรแกรมเชิงตรรกะเพื่อคำนวณค่าดิฟเฟอเรนเชียล(ตามสูตรคำนวณในรูปที่ 10.9)

$$\frac{dc}{dx} = 0 \qquad c \text{ is a constant}$$

$$\frac{dx}{dx} = 1$$

$$\frac{d}{dx}(u+v) = \frac{du}{dx} + \frac{dv}{dx} \qquad u \text{ and } v \text{ are functions of } x$$

$$\frac{d}{dx}(u-v) = \frac{du}{dx} - \frac{dv}{dx}$$

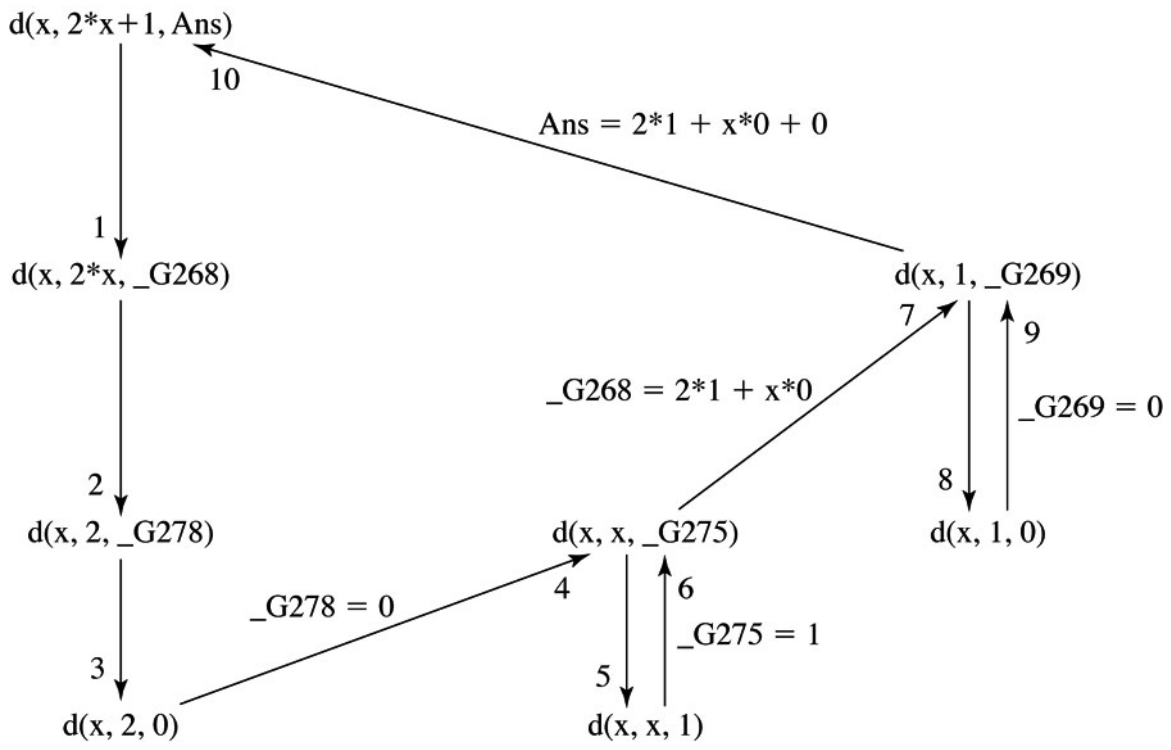
$$\frac{d}{dx}(uv) = u\frac{dv}{dx} + v\frac{du}{dx}$$

$$\frac{d}{dx}\left(\frac{u}{v}\right) = \left(v\frac{du}{dx} - u\frac{dv}{dx}\right)\Big/v^2$$

รูปที่ 10.9 กฎที่ใช้ในการคำนวณค่าดิฟเฟอเรนเชียล

การสั่งคำนวณค่าดิฟเฟอเรนเชียลใช้การเรียกเพรดิเคต d ที่อาร์กิวเมนต์ 3 ตัวอาร์กิวเมนต์ตัวแรกเป็น x ตัวที่สองเป็นนิพจน์ที่ต้องการทำดิฟเฟอเรนเชียลเทียบกับ x และอาร์กิวเมนต์ตัวที่ 3 เป็นตัวแปรที่ใช้แสดงผลลัพธ์สุดท้าย รูปที่ 10.10 แสดงขั้นตอนการทำงานของโปรล็อก เมื่อสั่งทำดิฟเฟอเรนเชียลกับนิพจน์ 2*x+1

```
d(X, U+V, DU+DV) :- d(X, U, DU), d(X, V, DV).
d(X, U-V, DU-DV) :- d(X, U, DU), d(X, V, DV).
d(X, U*V, U*DV + V*DU) :- d(X, U, DU), d(X, V, DV).
d(X, U/V, (V*DU - U*DV)/(V*V)) :- d(X, U, DU), d(X, V, DV).
d(X, C, 0) :- atomic(C), C\=X.
d(X, X, 1).
```



รูปที่ 10.10 ขั้นตอนการทำงานของโปรแกรมคำนวณค่าดิฟเฟอเรนเชียล

ตัวอย่างที่ 3 การทำโปรแกรมเชิงตรรกะเพื่อแก้ปัญหา word puzzle โดยปัญหาระบุว่า

"Baker, Cooper, Fletcher, Miller และ smith อาศัยอยู่ในตึก 5 ชั้น
Baker ไม่ได้พักอยู่ที่ชั้นห้า
Cooperไม่ได้พักอยู่ที่ชั้นหนึ่ง
Fletcherไม่ได้พักที่ชั้นบนสุดหรือล่างสุด และไม่ได้อยู่ชั้นติดกับ Smith หรือ Cooper
Millerพักอยู่ชั้นเหนือชั้นที่ Cooperพัก
คำถามคือใครพักอยู่ชั้นใดบ้าง?"

การแก้ปัญหาลักษณะเช่นนี้ สามารถทำได้ง่ายด้วยโปรแกรมเชิงตรรกะ ซึ่งมีรายละเอียดคำสั่งในโปรแกรมดังนี้

```
floors([floor(_,5),floor(_,4),floor(_,3),floor(_,2),floor(_,1)]).
building(Floor):- floors(Floors),
          member(floor(baker,B), Floors), B =\= 5,
          member(floor(cooper, C), Floors), C =\=1,
          member(floor(fletcher, F), Floors), F ='=1, F =\= 5,
          member(floor(miller, M), Floors), M > C,
          member(floor(smith, S), Floors),
          not(adjacent(S,F)), not(adjacent(F, C)),
          print_floors(Floors).
     member(X,[X|_]).
     member(X,[_|Y):- member(X,Y).

     adjacent(X,Y):- X =:= Y+1.
     adjacent(X,Y):- X =:= Y-1.

     print_floors([]).
     print_floors([A|B]):- write(A), nl, print_floors(B).
```

การสั่งให้โปรแกรมทำงานใช้การตั้งคำถาม

```
?-building(X).
```

10.6 การทำโปรแกรมเชิงตรรกะแบบมีเงื่อนไขบังคับ

(Constraint logic programming)

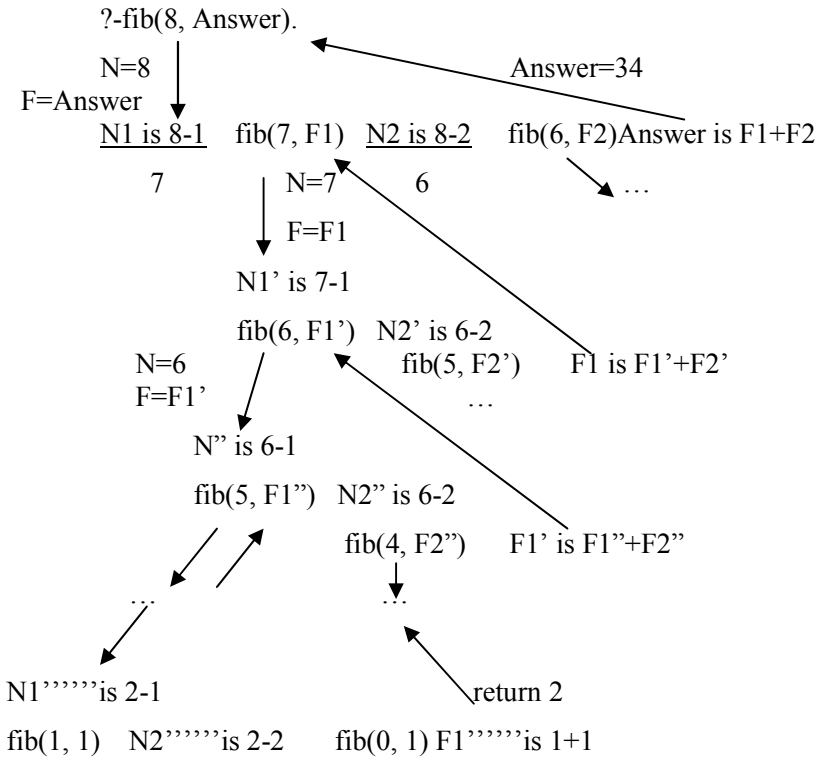
ตัวอย่างโปรแกรมโปรล็อกต่อไปนี้ใช้ในการพิจารณาเลขไฟโบนาชี จำนวนเลขไฟโบนาชีเป็นจำนวนเลขที่มีการเรียงลำดับเป็น 1,1,2,3,5,8,13,21,… ค่าในแต่ละตำแหน่งเกิดจากผลบวกของค่าในตำแหน่งสองตำแหน่งก่อนหน้านั้น เช่น 5 เกิดจาก 2+3, 8 เกิดจาก 5+3 ถ้ากำหนดให้ตำแหน่งแรกเป็นตำแหน่งที่ศูนย์ โปรแกรม fib(N, F) ทำหน้าที่คำนวณหาค่าไฟโบนาชี F ของตำแหน่งที่ N

```
fib(0,1).
fib(1,1).
fib(N,F):-N1 is N-1, fib(N1, F1),
          N2 is N-2, fib(N2,F2),
          F    is F1 + F2.
```

การตั้งคำถามเพื่อหาค่าไฟโบนาชีตำแหน่งที่ 8 จะเป็นดังนี้

```
?-fib(8, Answer).
Answer = 34
Yes
```

โดยมีเงื่อนไขการค้นหาคำตอบดังนี้



แต่ในการถามคำถามว่าค่า 34 เป็นเลขไฟโบนาชชีลำดับที่เท่าไร?

```
?-fib(Position, 34).
```

ขั้นตอนการหาคำตอบจะเป็น

?-fib(Position, 34).

↓ N = Position
F = 34

<u>N1 is Position-1</u> fib(N1, F1) N2 is Position-2 34 is F1 + F2

Fail

โปรล็อกจะให้คำตอบเป็น No เพราะในขั้นตอนแรกของการทำงานคือ N1 is Position-1 ไม่สามารถหาค่าได้ เพราะไม่สามารถกระทำการลบเพื่อหาค่าให้ N1

การทำโปรแกรมเชิงตรรกะแบบมีเงื่อนไขบังคับแก้ปัญหานี้ด้วยการปรับปรุงโปรแกรมเป็น

```
fib(0, 1).
fib(1, 1).
fib(N, F1 + F2) :- N >= 2, fib(N-1, F1), fib(N-2, F2).
```

ดังนั้นคำถาม `?-fib(Position, 34)` จะมีขั้นตอนการทำงานดังนี้

?-fib(Position, 34)
↓ N = Position
F1+F2 = 34

Position >= 2 fib(Position-1, F1) fib(Position-2, F2)
↓ N' = Position-2 …
F1''+F2'' = F1'

Position-1 >= 2 fib(Position-2, F1') fib(Position-3,F2')
↓ N" = Position-2 …
F1"+F2" = F1'

…

การหาคำตอบให้กับคำถามข้างต้นจึงเป็นการหาคำตอบให้กับเงื่อนไข

F1+F2 = 34	Position >= 2
(F1'+F2')+F2 = 34	Position >= 3
((F1"+F2")+F2')+F2 = 34	Position >= 4
…..	

เมื่อใดที่หาผลบวก F ที่เท่ากับ 34 ได้โปรแกรมจะหยุดการทำงานแล้วส่งคำตอบกลับเป็นค่าตำแหน่ง

10.7 สรุป

การทำโปรแกรมเชิงตรรกะมีความแตกต่างอย่างมากจากการทำโปรแกรมเชิงคำสั่งในการทำโปรแกรมเชิงคำสั่งโปรแกรมเมอร์จะกำหนดปัญหาและกำหนดขั้นตอนวิธีอย่างละเอียดเพื่อนำไปสู่คำตอบของปัญหา แต่ในการทำโปรแกรมเชิงตรรกะโปรแกรมเมอร์เพียงแต่ประกาศความสัมพันธ์เชิงตรรกะที่เกี่ยวข้องกับปัญหา การค้นหาคำตอบเป็นหน้าที่ของโปรแกรมประมวลผลที่จะสรุปคำตอบ (infer a solution) จากความสัมพันธ์ต่างๆ ที่ถูกประกาศไว้

รูปแบบของคำประกาศความสัมพันธ์ จะใช้รูปแบบของแคลคูลัสเพรดิเคตที่มีข้อกำหนดว่าในแต่ละข้อความ(หรือ คลอส--claus) จะมีเทอมที่เป็นบวก (position literal) เพียงเทอมเดียว ข้อความในรูปแบบนี้จะเรียกว่า ข้อความรูปแบบฮอร์น (Horn clause) และขั้นตอนการสรุปคำตอบจะใช้เทคนิคการพิสูจน์ที่เรียกว่า เรโซลูชัน ซึ่งเป็นการพิสูจน์ว่าคำตอบที่สรุปได้เป็นจริงจะใช้วิธีสมมุติว่าคำตอบเป็นเท็จแล้วเมื่อผลการพิสูจน์เป็นเท็จ จึงเป็นการพิสูจน์ประเภทพิสูจน์ขัดแย้ง หรือเป็นการทำปฏิเสธซ้อนปฏิเสธ นั่นคือ ถ้าต้องการพิสูจน์ว่าคำตอบที่สรุปได้เป็นจริง จะใช้วิธีสมมุติว่าคำตอบเป็นเท็จแล้วเมื่อผลการพิสูจน์เป็นเท็จ จึงเป็นการยืนยันว่าคำตอบที่ได้เป็นจริง ภาษาโปรล็อกซึ่งเป็นภาษาที่ใช้ในการทำโปรแกรมเชิงตรรกะจะใช้เทคนิคเรโซลูชันร่วมกับการทำยูนิฟิเคชัน เพื่อทำให้เทอมที่มีรูปแบบตรงกันก่อนที่จะลดรูปข้อความไปตามลำดับจนกระทั่งได้ค่าเท็จ นอกจากนี้ในกรณีที่ความสัมพันธ์ได้รับการนิยามไว้หลายลักษณะโปรล็อกจะใช้เทคนิคย้อนกลับเพื่อการพิสูจน์ความสัมพันธ์ให้ครบทุกลักษณะ การทำโปรแกรมเชิงตรรกะจึงใช้เวลาและเนื้อที่หน่วยความจำจำนวนมากเพื่อการประมวลผล แต่ข้อเด่นของการทำโปรแกรมในแนวทางนี้ คือ เป็นวิธีการทำโปรแกรมในระดับนามธรรม หรือ ในระดับสูงมาก ทำให้เหมาะสมที่จะใช้งานปัญญาประดิษฐ์ และงานพัฒนาระบบผู้เชี่ยวชาญ

แบบฝึกหัดท้ายบทที่ 10

คำถามอัตนัย

1. จงแสดงตารางค่าความจริง (truth table) ของนิพจน์ต่อไปนี้

   1.1 $p \rightarrow q$

   1.2 $q \rightarrow p$ (converse ของ $p \rightarrow q$)

   1.3 $\neg p \rightarrow \neg q$ (inverse ของ $p \rightarrow q$)

   1.4 $\neg q \rightarrow \neg p$ (contrapositive ของ $p \rightarrow q$)

2. กำหนดข้อความ

   "ถ้าเหตุการณ์ที่ 1 ไม่เกิดขึ้น แล้วเหตุการณ์ที่ 2 จะเกิดขึ้น"

   จงตอบคำถามต่อไปนี้

   2.1 จงเขียนข้อความนี้ในรูปแบบของ p และ q

   2.2 จงเขียนตารางค่าความเป็นจริงของ 4.1

   2.3 จงเขียนข้อความนี้ใหม่โดยมีค่าความจริงเหมือนเดิม

3. ต่อไปนี้ข้อใดถูกต้อง

   (a) $(p \rightarrow q) = ((\neg q) \rightarrow (\neg p))$

   (b) $(p \wedge q) = p$

   (c) $(pq) = p$

   (d) $\neg(p \wedge q) = (\neg q) \vee (\neg p)$

4. จงแสดงให้เห็นชัดเจนว่า

   $(p \rightarrow q) \wedge (q \rightarrow p)$

   มีค่าความจริงเท่ากัน

5. ต่อไปนี้เป็นพื้นฐานของหลักการ resolution

   $a \cup b$ มีค่าจริง

   $\underline{\neg a \vee c}$ <u>มีค่าจริง</u>

   จะได้ $a \vee c$ มีค่าจริง

   จงแสดงตารางค่าความจริง

6. เมื่อพิจารณาการกระทำต่อไปนี้

   $(\neg q) \rightarrow (\neg p)$ มีค่าจริง

   $\underline{p}$ <u>มีค่าจริง</u>

   จะได้ q เป็นจริงหรือไม่

   (a) จงอธิบายว่าข้อสรุปนี้เป็นจริง (ทุกกรณีหรือไม่)

(b) จงอธิบายว่าเราสามารใช้หลักการ chaining-based inference ด้วยลักษณะนี้ได้หรือไม่

7. จงใช้ unification ของนิพจน์ต่อไปนี้

(this what Is what)
(What Class is thin)

โดยที่คำที่ขึ้นต้นด้วยตัวอักษรตัวใหญ่ คือ ตัวแปร

โดยที่คำที่ขึ้นต้นด้วยตัวอักษรตัวเล็ก คือ ค่าคงที่

8. ลำดับของตัวบอกปริมาณ (quantification) มีความสำคัญต่อการพิจารณาค่าความจริงจงอธิบายความแตกต่าง ระหว่าง 2 statement ต่อไปนี้

$\forall x(\exists y \text{ hates } (x, y))$
$\exists x(\forall y \text{ hates } (x, y))$

เมื่อ hates (x, y) มีความหมายว่า “x เกลียด y”

9. กำหนดให้นิพจน์ที่จะทำการ unification คือ

a) (x a y)
(y z b)

b) (A B C D)
(a b c d)

จงอธิบายว่า ต่อไปนี้เป็นการแทนค่าของ unification ข้อใด

9.1 {y/x, a/z, y/z, a/x}

9.2 {y/z, a/z, a/x, x/y}

9.3 {a/z, z/y, a/y}

9.4 {x/y, a/z, y/z, a/y}

10. จงใช้ resolution เพื่อพิสูจน์ว่า การสรุปที่ 1 และ 2 มีความถูกต้อง

สมมุติฐานที่ 1 : $P \rightarrow$

สมมุติฐานที่ 2 : $P \rightarrow Q$

<u>สมมุติฐานที่ 3 : $(R \wedge Q) \rightarrow S$</u>

ข้อสรุปที่ 1 : Q

ข้อสรุปที่ 2 : $R \rightarrow S$

คำถามปรนัย: ให้เลือกคำตอบที่ถูกต้องที่สุด

1. กำหนดโปรแกรม Prolog ให้ดังต่อไปนี้

```
father(bob).
man(X) :- father(X).
```

ถ้าพิมพ์คำสั่ง ?man(X). โปรแกรมจะแสดงผลลัพธ์ใด ?

ก. yes
ข. no
ค. X = bob
yes
ง. father(bob)
yes

2. ถ้า parent(john, mary) หมายถึง john เป็นพ่อ/แม่ ของ mary ข้อใดต่อไปนี้นิยามบรรพบุรุษ (ancestor) ได้ถูกต้องที่สุด ?

ก. parent(john, mary).
ancestor(john, mary).

ข. parent(john, mary).
parent(X, Y) :- ancestor(X, Y).

ค. ancestor(john,mary).
ancestor(X, Y) :- parent(X, Y).

ง. ancestor(X, Y) :- parent(X, Y).
ancestor(X, Y) :- parent(X, Z), ancestor(Z, Y).

3. ภาษาเชิงตรรกะ (logic programming language) มีชื่อเรียกอีกอย่างว่าอย่างไร ?

ก. declarative language
ข. calculus language
ค. non-functional language
ง. logical language

4. ข้อใดกล่าวได้ถูกต้องเกี่ยวกับ query ในภาษา Prolog ?

ก. เป็นข้อความที่ประกอบด้วย 1 predicate และสามารถใช้ตัวแปร
ข. เป็นข้อความที่มีได้มากกว่า 1 predicate และเชื่อมด้วยเครื่องหมาย ;
ค. เป็นข้อความที่มีได้มากกว่า 1 predicate และเชื่อมด้วยเครื่องหมาย ,
ง. สามารถใช้เครื่องหมาย !- กำหนดว่าให้หาคำตอบแบบ forward chaining

5. ในภาษา Prolog การกำหนดค่าให้กับตัวแปร เรียกว่าอะไร ?

ก. declaration ข. instantiation

ค. resolution ง. chaining

6. ภาษา Prolog ใช้กระบวนการใดในการตอบ query ?

ก. resolution ข. forward chaining

ค. instantiation ง. binding

7. กำหนดข้อความในภาษา Prolog ให้ดังต่อไปนี้

Sum is Sum + Number.

คำกล่าวในข้อใดถูกต้องที่สุด ?

ก. ข้อความนี้เป็นการสั่งบวกค่า Sum และ Number

ข. ข้อความนี้ให้ผลเช่นเดียวกับ Sum = Sum + Number.

ค. ข้อความนี้ผิด เพราะ Sum ไม่สามารถเปลี่ยนค่า

ง. ข้อความนี้ผิด เพราะไม่กำหนดค่าเริ่มต้นให้ Sum

8. กำหนดข้อความในภาษา Prolog ให้ดังต่อไปนี้

parent(X, Y) :- mother(X, Y).

คำกล่าวในข้อใดถูกต้องที่สุด ?

ก. เป็นการกำหนด rule ในลักษณะที่มีเงื่อนไข

ข. เป็นการนิยามความหมายว่า “parent ทุกคนจะต้องมี mother”

ค. เป็นการนิยามฟังก์ชัน mother(X,Y)

ง. เป็นการเขียนที่ผิดเนื่องจาก mother ต้องมีพารามิเตอร์เดียว