MODULE 2

KNOWLEDGE REPRSENTATION ISSUES

Predicate logic

Representing Knowledge using Rules

Representation and mapping

- Two kind of entities
  - Facts
  - Representation of facts
- Structuring these entities are of two level
  - Knowledge level
  - Symbol level

Mapping between facts and Representations

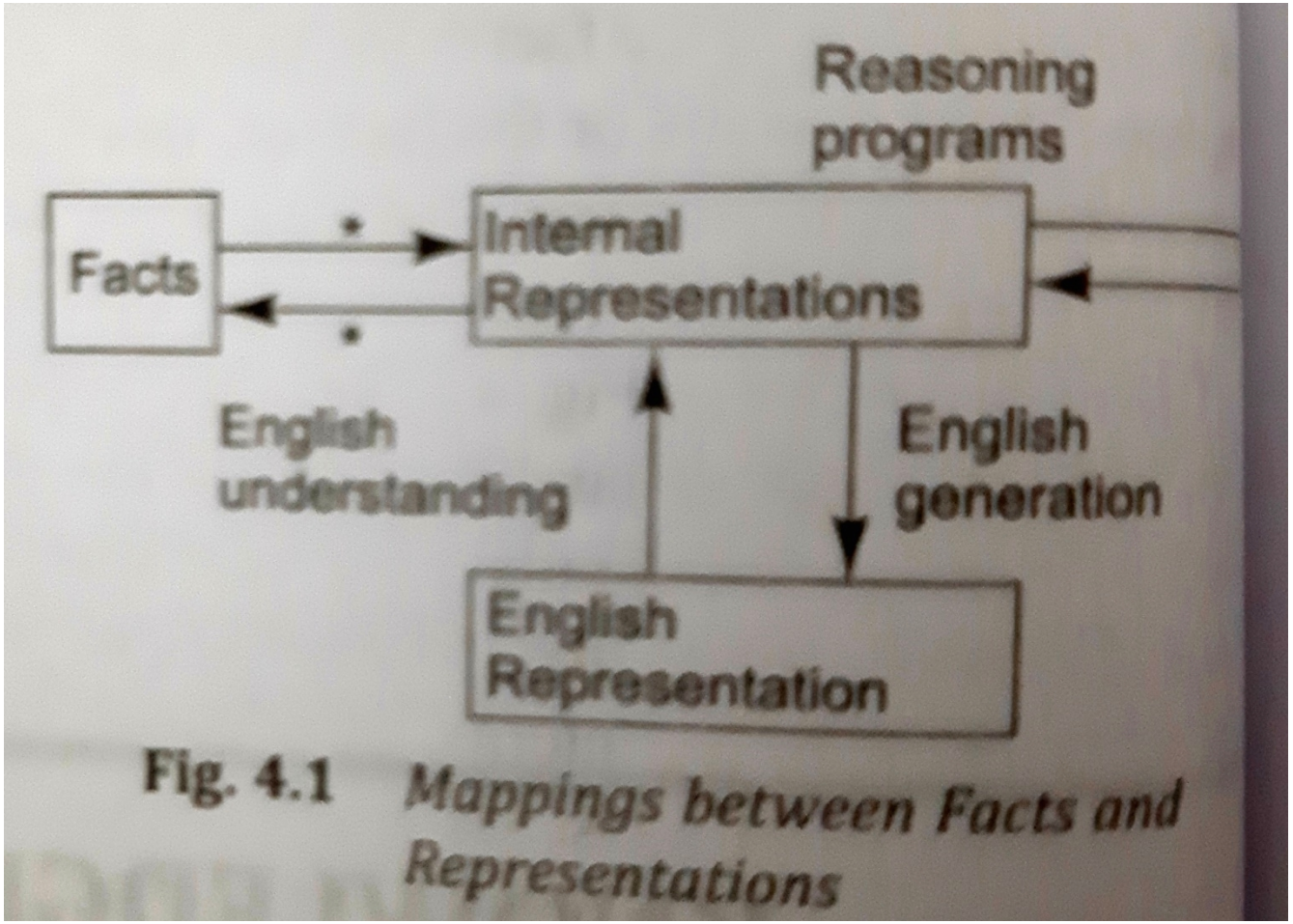


Fig. 4.1 *Mappings between Facts and Representations*

Spot is a dog

Represented in logic: dog(spot)

All dogs have tails

Logic: $\forall$x: dog(x)$\rightarrow$hastail(x)

Deductive mechanism: hastail(spot)

Use appropriate backward mapping function we could generate

Spot has a tail

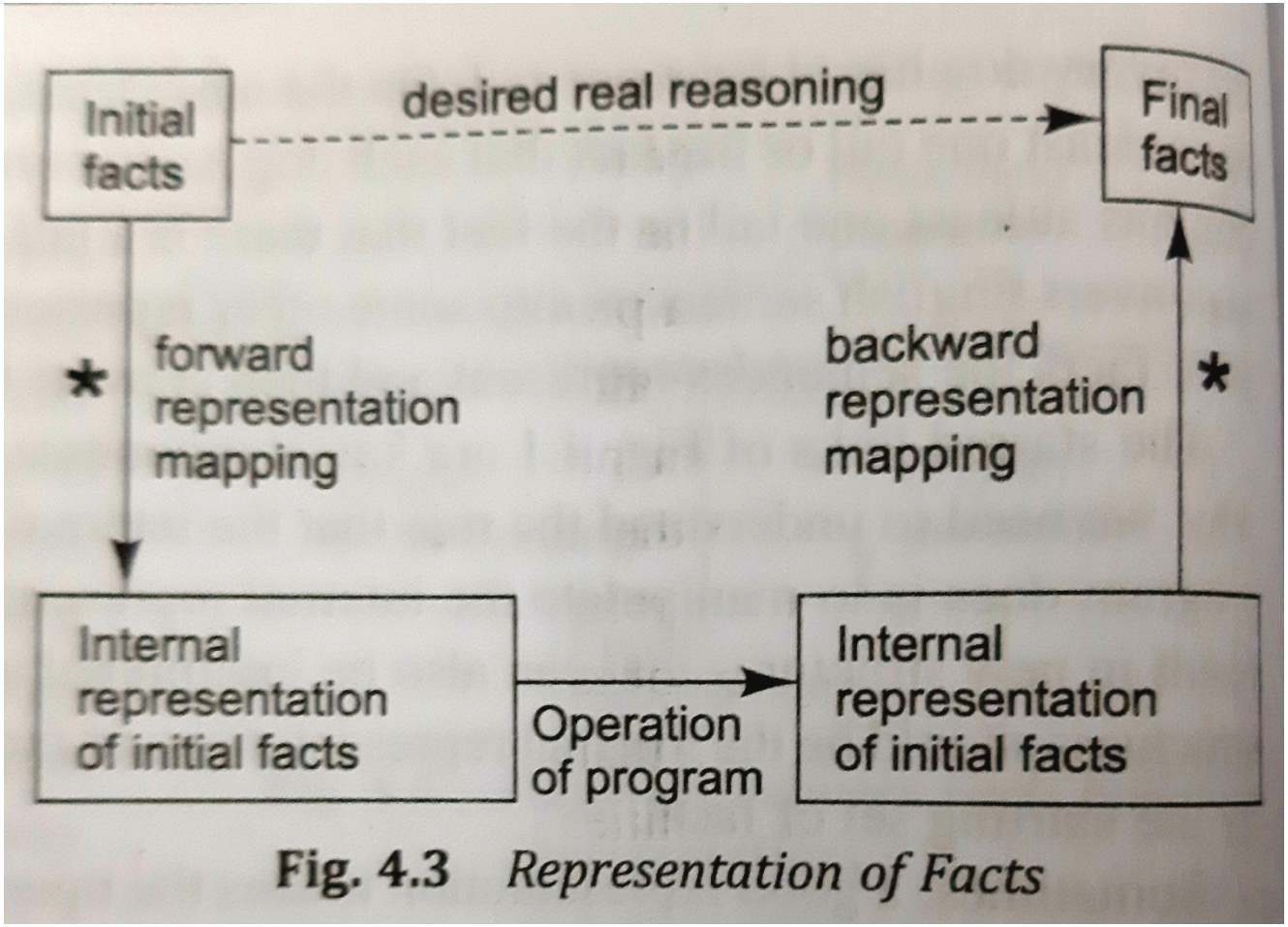


Fig. 4.3 *Representation of Facts*

- Representation of knowledge in particular domain should possess following properties:
  - Representational adequacy
  - Inferential adequacy
  - Inferential Efficiency
  - Acquisitional efficiency

Approaches to knowledge representation

- Simple relational knowledge:Set of relations

Player	Height	Weight	Bats-Throws
Hank Aaron	6-0	180	Right-Right
Willie Mays	5-10	170	Right-Right
Babe Ruth	6-2	215	Left-Left
Ted Williams	6-3	205	Left-Right

```
player_info('hank aaron', '6-0', 180,right-right).
```

Fig. 4.4 *Simple Relational Knowledge and a sample fact in Prolog*

- Inheritable knowledge: Useful form of inference is property inheritance. Object must ne arranged in form of classes and classes arranged in generalization hierarchy.
- Attribute isa(class inclusion) and instance(class memebership)

  - team(Pee wee Reese)=Brooklyn Dodgers
  - bating_average(Three_Finger_Brown)=.106
  - height(Pee Wee Reese)=6.1
  - bats(Three finger Brown)=Right

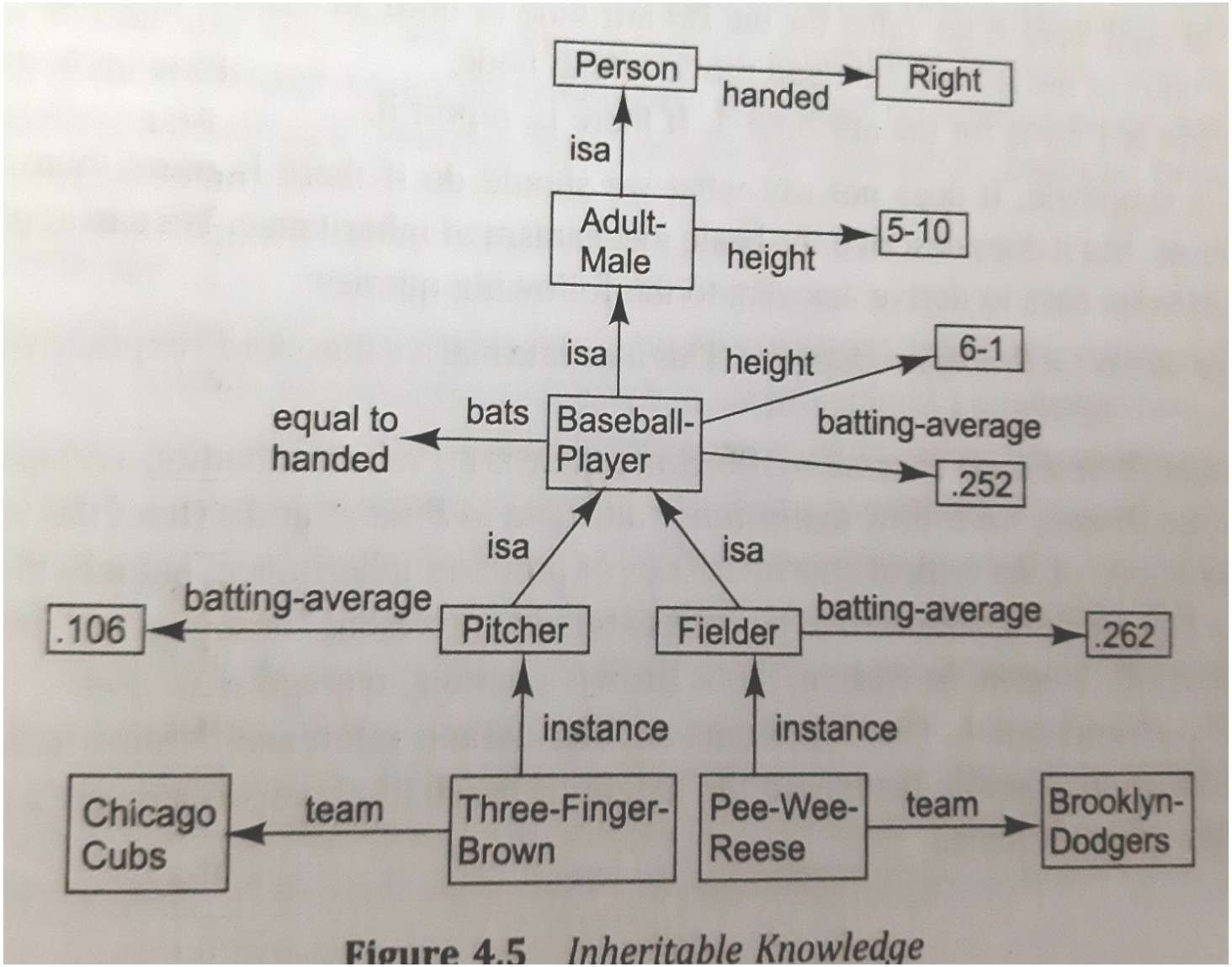


Figure 4.5 *Inheritable Knowledge*

- Inferential knowledge: There are many procedures which reason forward and some may backward. One of the most commonly used is resolution which exploits proof by contradiction

- Procedural knowledge: It specifies what to do when. It is represented in most common way as code(LISP)
- Most commonly used technique is production rules

Issues in knowledge representation

1. Important attributes: instance,isa
2. Relationship among attributes
   - Inverses
   - Existence in a hierarchy
   - Techniques for reasoning about values
   - Single valued attributes
3. Choosing granularity of representation
4. Representing set of objects
5. Finding right structures as needed

- Attributes that we use to describe objects are themselves entities that we represent.
- Inverses-represent both relationship in single representation that ignores focus

  Team(pee wee reeese, brooklyn dodgers)

  – Use attributes in that focus on single entity but to use them pairs, one the inverse of others
  – Team=brooklyn dodgers
  – Team-memebers=pee wee reese

- As there are classes as objects and specialized subset of classes, there are attributes and specialization of attributes(constraints on values)
- Techniques for reasoning about values
    - Information about type for a value
    - Constraints on a value
    - Rules for computing the value when it is needed(if needed/backward rules)
    - Rules that describe the action that should be taken if a value ever becomes known(forward rule/if added rules)

- Single valued attribute: which is guaranteed to take unique value
  - Introduce an explicit notation for a temporal interval
  - Assume only temporal interval that is of interest now. Sow new value is asserted discard the old value
  - Provide no explicit support.

Choosing granularity of representation

- What should be our primitives? Should there be smaller number of lower ones or should be larger number covering range of granularities
- John spotted Sue
- Spotted(agent(John),object(Sue))
- Who spotted Sue?
- Did john see Sue?

- We add other facts
- Spotted(x,y)→saw(x,y)
- Alternative to this is spotting is really type of seeing
  - Saw(agent(john),object(Sue),timespan(briefly))
  - We broke spotting into two primitives seeing and timespan

- High level facts may require lot of storage when broken down into primitives
- Substantial work must be done to reduce knowledge into primitive form
- It is not clear what primitives must be

Representing set of objects

- There are some property true for set but that are not true for individual member of set
- Represent set of objects ,if a property is true of all elements of set, then it is more efficient to associate with set rather to associate explicitly with every element of set

- 2 ways to state definition of set and its elements
- Extensional definition-list members
- Intensional definition-rule when a particular object evaluated ,return true or false
  - Used to describe infinite sets
  - Allow them to depend on parameters that change

Finding the right structure as needed

- John went to Steak and Ale last night.He ordered a large rare steak,paid his bill and left.

Did John eat dinner last night?

  - Selecting initial structure
  - Revising choice when necessary

Frame problem

- Problem of representing the facts that change as well as those that donot change is known as frame problem
- To support this kind of reasoning, we have explicit set of axioms called frame axioms, which describe all the things that donot change when a particular operator is applied

$color(x,y,s1) \wedge move(x,s1,S2) \rightarrow color(x,y,S2)$

State variable

Representing simple facts in logic

It is raining

RAINING

It is sunny

SUNNY

It is windy

WINDY

If it is raining,then it is not sunny

RAINING→¬SUNNY

Socrates is a man

SOCRATESMAN

Plato is a man

PLATOMAN

Predicates applied to argument

MAN(SOCRATES)

MAN(PLATO)

All men are mortal

MORTALMAN

Predicate logic is semidecidable

- Marcus was a man
- Marcus was a Pompeian
- All Pompeians were Romans
- Caesar was a ruler
- All Romans were either loyal to Caesar or hated him
- Everyone is loyal to someone
- People only try to assassinate rulers they are not loyal to
- Marcus tried to assassinate Caesar

1. man(Marcus)
2. pompeian(Marcus)
3. ∀x:Pompeian(x)→Roman(x)
4. ruler(Caesar)
5. ∀x:Roman(x)→loyalto(x,Caesar)∨ hate(x,Caesar)
6. ∀x:→y:loyalto(x,y)
7. ∀x:∀y:person(x)∧ruler(y)∧ tryassisassinate(x,y)→¬loyalto(x,y)
8. tryassassiante(Marcus,Caesar)

- Was Marcus loyal to Caesar?
- Formal proof reasoning backward from desired goal:¬ loyalto(Marcus,Caesar)

¬ loyalto(Marcus,Caesar)

↑(7)

Person(Marcus)∧ruler(Caesar)∧tryassassinate(Marcus,Caesar)

↑(4)

Person(Marcus)∧tryassassinate(Marcus,Caesar)

↑(8)

Person(Marcus)

9.All men are people

∀x:man(x)→person(x)

¬ loyalto(Marcus,Caesar)

↑(7)

Person(Marcus)∧ruler(Caesar)∧
tryassassinate(Marcus,Caesar)

↑(4)

Person(Marcus)∧tryassassinate(Marcus
,Caesar)

↑(8)

Person(Marcus)(9)

Representing instance and isa relationships

1. $man(Marcus)$
2. $Pompeian(Marcus)$
3. $\forall x : Pompeian(x) \rightarrow Roman(x)$
4. $ruler(Caesar)$
5. $\forall x : Roman(x) \rightarrow loyalto(x, Caesar) \vee hate(x, Caesar)$

1. $instance(Marcus, man)$
2. $instance(Marcus, Pompeian)$
3. $\forall x : instance(x, Pompeian) \rightarrow instance(x, Roman)$
4. $instance(Caesar, ruler)$
5. $\forall x : instance(x, Roman) \rightarrow loyalto(x, Caesar) \vee hate(x, Caesar)$

1. $instance(Marcus, man)$
2. $instance(Marcus, Pompeian)$
3. $isa(Pompeian, Roman)$
4. $instance(Caesar, ruler)$
5. $\forall x : instance(x, Roman) \rightarrow loyalto(x, Caesar) \vee hate(x, Caesar)$
6. $\forall x : \forall y : \forall z : instance(x, y) \wedge isa(y, z) \rightarrow instance(x, z)$

Computable functions and predicates

- Gt(1,0) lt(0,1)
- Gt(2,1) (lt(1,2)
- Gt(3,2) lt(2,3)
- … …
- … …

FACTS

Marcus was a man

Man(Marcus)

Marcus was Pompeian

Pompeian(Marcus)

Marcus was born in 40 A.D

Born(Marcus,40)

All men are mortal

$\forall x:man(x) \rightarrow mortal(x)$

All Pompeian died when volcano erupted in 79 A.D

$erupted(volacano,79) \wedge \forall x:[Pompeian(x) \text{—>} died(x,79)]$

No mortal lives longer than 150 years

$\forall x:\forall t1:\forall t2:mortal(x) \wedge born(x,t1) \wedge$

$gt(t2-t1,150) \rightarrow dead(x,t2)$

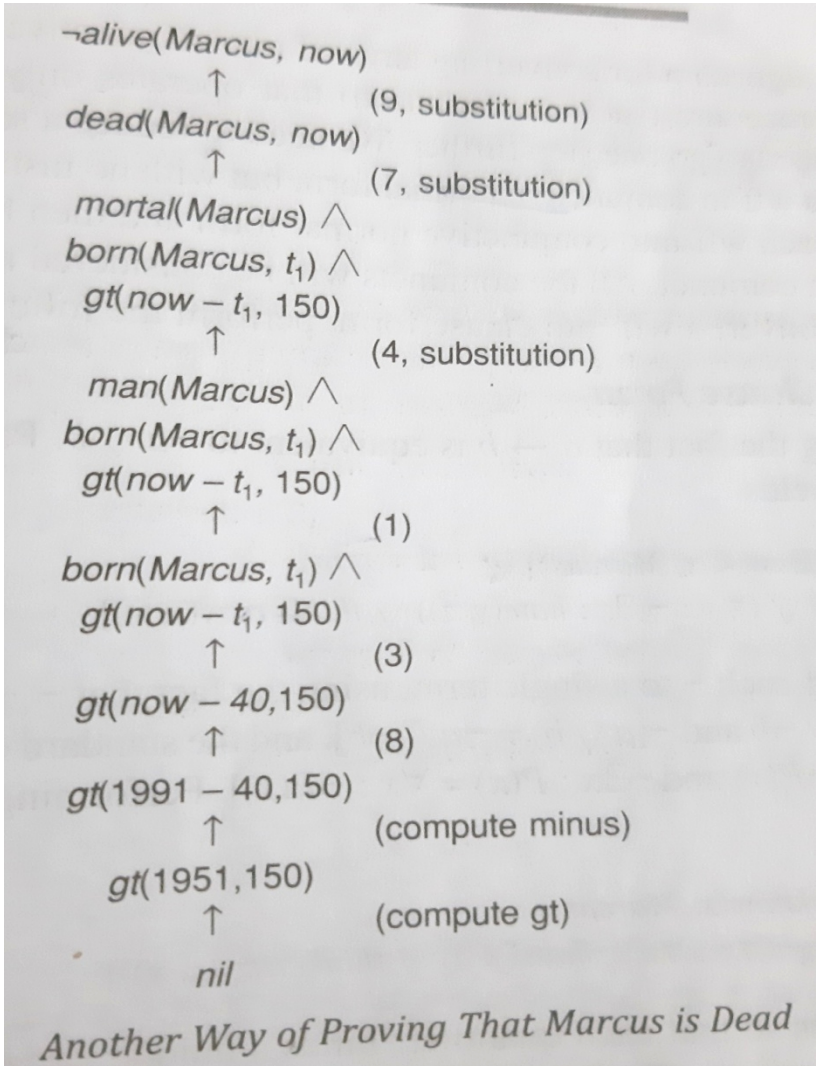
It is now 1991 A.D

now=1991

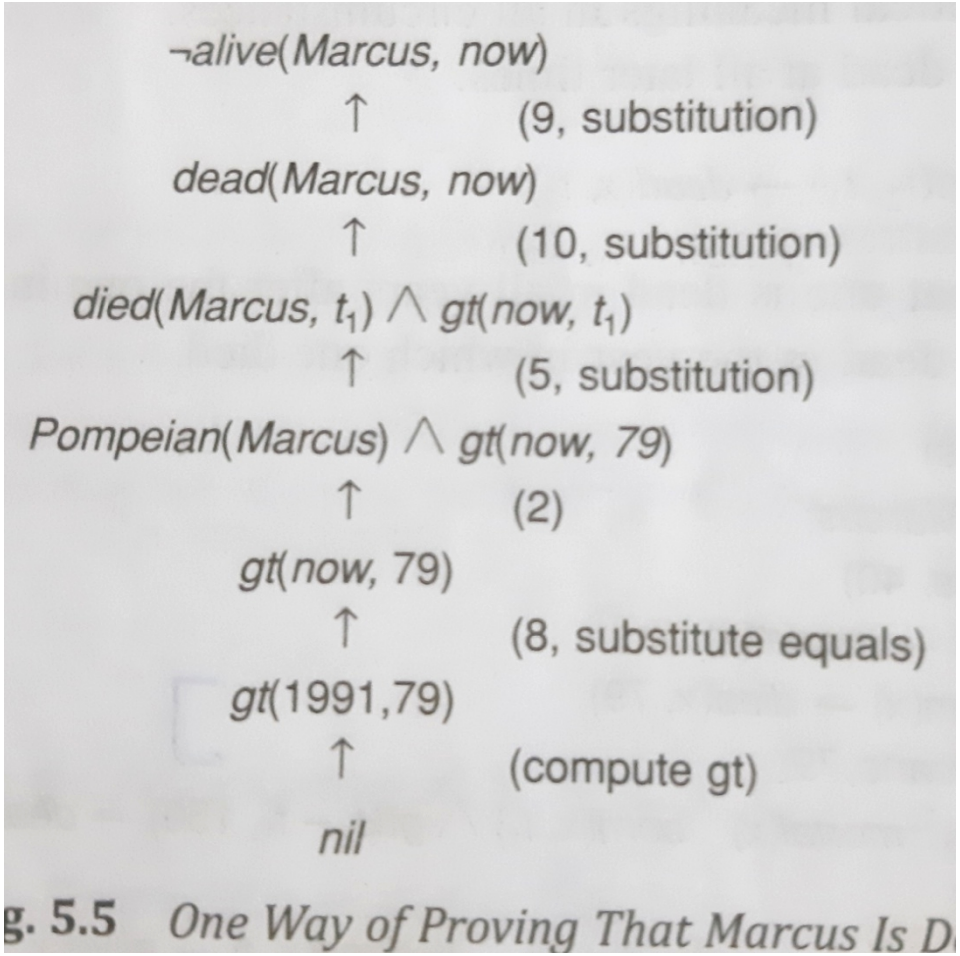
- Is Marcus alive?
  - Killed by volcano
  - More than 150 years
- Alive means not dead
  - ∀x:∀t:[alive(x,t)→¬dead(x,t)]^[¬dead(x,t)→ alive(x,t)
- If someone dies he is dead at all later times
  - ∀x:∀t1:∀t2:died(x,t1)∧gt(t2,t1)→ dead(x,t2)

1. $man(Marcus)$
2. $Pompeian(Marcus)$
3. $born(Marcus, 40)$
4. $\forall x : man(x) \rightarrow mortal(x)$
5. $\forall : Pompeian(x) \rightarrow died(x, 79)$
6. $erupted(volcano, 79)$
7. $\forall_x : \forall t_1 : \forall t_2 : mortal(x) \wedge born(x, t_1) \wedge gt(t_2 - t_1, 150) \rightarrow dead(x, t_2)$
8. $now = 1991$
9. $\forall x : \forall t : [alive(x, t) \rightarrow \neg dead(x, t)] \wedge [\neg dead(x, t) \rightarrow alive(x, t)]$
10. $\forall x : \forall t_1 : \forall t_2 : died(x, t_1) \wedge gt(t_2, t_1) \rightarrow dead(x, t_2)$

Fig. 5.4 *A Set of Facts about Marcus*



Another Way of Proving That Marcus is Dead



g. 5.5 *One Way of Proving That Marcus Is D*

Born(Marcus,t1)

Gt(now-t1,150)

- Even very simple conclusions can require many steps to prove
- Variety of processes such as matching, substitution are involved in the production of proof

Resolution

- Resolution produces proof by refutation. To prove a statement resolution attempts to show that the negation of the statement produces contradiction with known statements
- It operates on statement that have been converted to a very convenient standard form

Conjunctive normal form(CNF)

- Flatter(less embedding of components)
- Quantifiers were separated from rest of the formula

  All Roman who know Marcus either hate Caesar or think anyone who hates anyone is crazy

  ∀x:[Roman(x)∧ know(x,Marcus)]→[hate(x,Caesar)v(∀ y :∃z hate(y,z)→thinkcrazy(x,y))]
- CNF: ¬Roman(x) ∧¬know(x,Marcus)v hate(x,Caesar)v ¬hate(y,z)vthinkcrazy(x,z)

All Roman who know Marcus either hate Caesar or think anyone who hates anyone is crazy

∀x:[Roman(x)∧know(x,Marcus)]→[hate(x,Caesar) v(∀ y :∃z: hate(y,z)→thinkcrazy(x,y))]

- CNF:

¬Roman(x)∧¬know(x,Marcus)vhate(x,Caesar)v ¬hate(y,z)vthinkcrazy(x,z)

Algorithm:convert to clause form

1. Eliminate→,using the fact a→b is equivalent to ¬avb

∀x:¬[Roman(x)∧know(x,Marcus)]v [hate(x,Caesar)v(∀ y :¬(∃z hate(y,z))v thinkcrazy(x,y))]

2. Reduce the scope of each ¬to a single term using the fact ¬(¬p)=p,de Morgans law and standard correspondences between quantifier

∀x:[¬Roman(x)v¬know(x,Marcus)]v [hate(x,Caesar)v(∀ y :∀ z:¬ hate(y,z)v thinkcrazy(x,y))]

3.Standardize the variable so that each quantifier binds to unique variable

∀x:P(x)v∀x:Q(x)

Would be converted to

∀x:P(x)v∀y:Q(y)

4.Move all quantifiers to the left of the formula without changing their relative order

∀x: ∀y :∀z:

[¬Roman(x)v¬know(x,Marcus)]v[hate(x,Caesar)v (¬ hate(y,z)vthinkcrazy(x,y))]

Formula now is known as prenex normal form

5.Eliminate existential quantifier

Эy:President(y)

Can be transformed into formula

President(S1)

6.Drop the prefix,all remaining variables are universally quantified

[¬Roman(x)v¬know(x,Marcus)]v[hate(x,Caesar)v(¬ hate(y,z)vthinkcrazy(x,y))]

7.Convert the matrix into conjunction of disjuncts,but our example has no ands

¬Roman(x)v¬know(x,Marcus)v[hate(x,Caesar)v¬hate(y,z)vthinkcrazy(x,y)

8.Create a separate clause corresponding to each conjunct

9.Standardize apart the variables in set of clauses generated in step 8

Basis of Resolution

- Resolution operates by taking two clauses that each contain the same literal(ex-winter).
- The literal must occur in positive form in one clause and in negative form in the other.
- The resolvent is obtained by combining all of the literals of two parent clauses except the ones that cancel

Example→ winter v summer

¬winter v cold

We can deduce

summer v cold

If the clause that is produced is empty clause, then contradiction has been found

Example→ winter

¬winter

This example produce a empty clause

Predicate logic→ Herbrands theorem

Resolution in propositional logic

- Algorithm

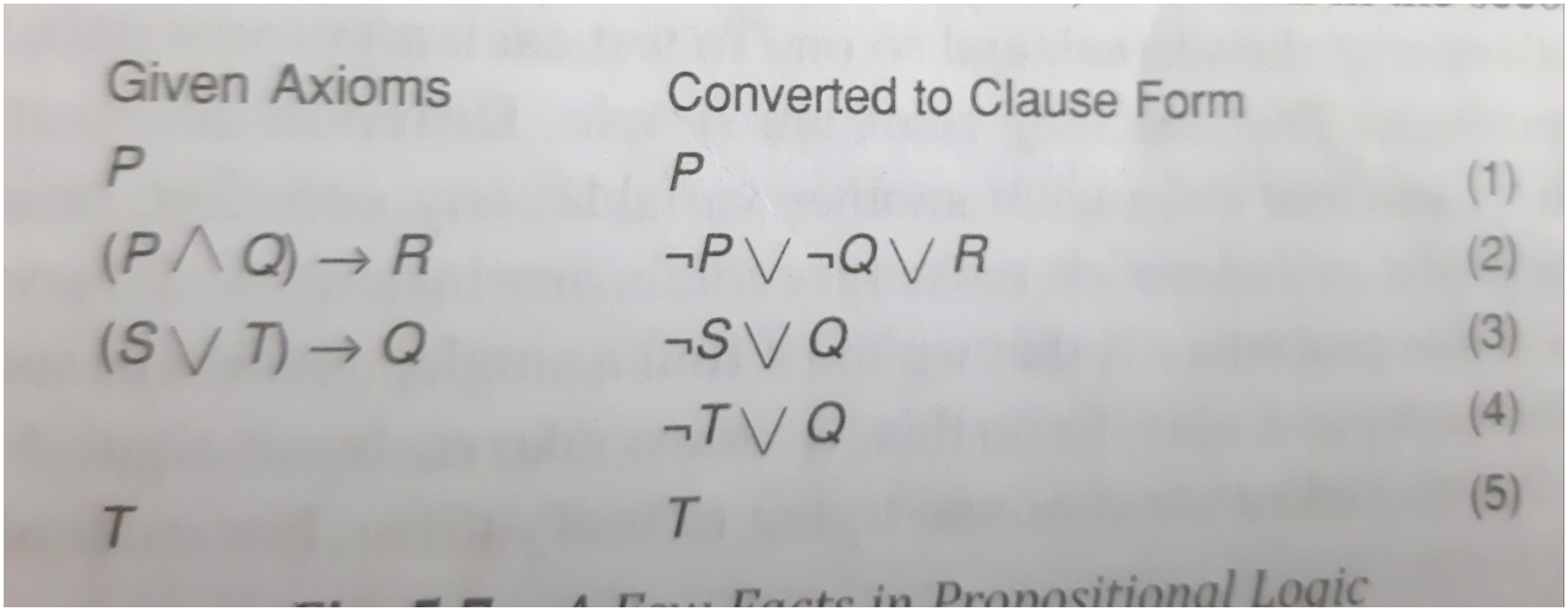
set of axioms F is the following.

producing a proof by resolution of proposition P with respect to a

Algorithm: Propositional Resolution

1. Convert all the propositions of F to clause form.
2. Negate P and convert the result to clause form. Add it to the set of clauses obtained in step 1.
3. Repeat until either a contradiction is found or no progress can be made:
   (a) Select two clauses. Call these the parent clauses.
   (b) Resolve them together. The resulting clause, called the *resolvent*, will be the disjunction of all of the literals of both of the parent clauses with the following exception: If there are any pairs of literals L and $\neg L$ such that one of the parent clauses contains L and the other contains $\neg L$, then select one such pair and eliminate both L and $\neg L$ from the resolvent.
   (c) If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

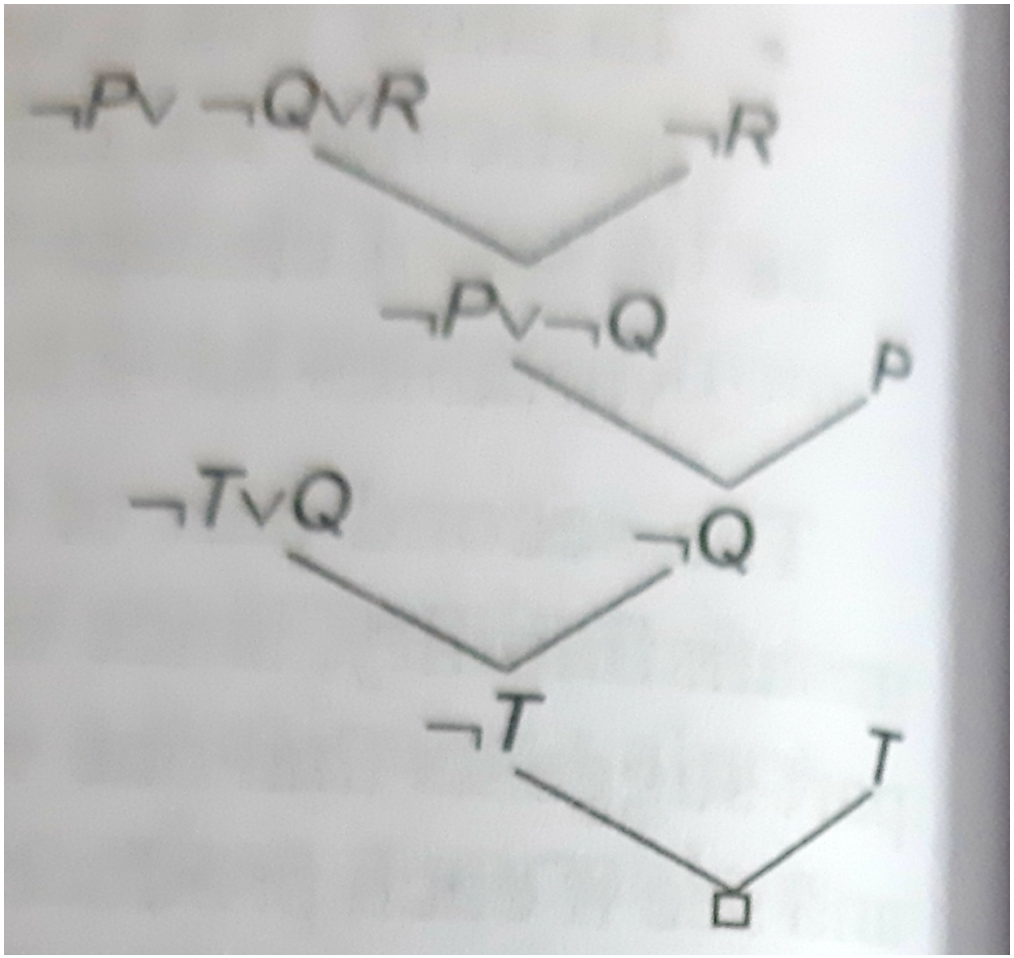
Facts in propositional logic



Given Axioms	Converted to Clause Form	
P	P	(1)
$(P \wedge Q) \rightarrow R$	$\neg P \vee \neg Q \vee R$	(2)
$(S \vee T) \rightarrow Q$	$\neg S \vee Q$	(3)
	$\neg T \vee Q$	(4)
T	T	(5)

Facts in Propositional Logic

Resolution in propositional logic



Unification Algorithm

- In predicate logic ,matching process is complicated since arguments of the predicates must be considered

Ex1: man(john)

¬man(john) is a contradiction

Ex2: man(john)

¬man(spot) is not a contradiction

P(x,x)

P(y,z)

Substitute y for x,i.e y/x

P(y,y)

P(y,z)

Composition of two substitutions (z/y)(y/x)

Example: hate(x,y)

hate(Marcus,z)

Can be unified with any of the following substitutions

(Marcus/x, z/y)

(Marcus/x, y/z)

(Marcus/x, Caeser/y, Caesar/z)

(Marcus/x, Polonius/y, Polonius/z)

Algorithm:Unify(L1,L2)

Algorithm: Unify(Ll, L2)

1. If $L1$ or $L2$ are both variables or constants, then:
   (a) If $L1$ and $L2$ are identical, then return NIL.
   (b) Else if $L1$ is a variable, then if $L1$ occurs in $L2$ then return {FAIL}, else return ($L2/L1$).
   (c) Else if $L2$ is a variable then if $L2$ occurs in $L1$ then return {FAIL}, else return ($L1/L2$).
   (d) Else return {FAIL}.
2. If the initial predicate symbols in $L1$ and $L2$ are not identical, then return {FAIL).
3. If LI and $L2$ have a different number of arguments, then return {FAIL}.
4. Set *SUBST* to NIL. (At the end of this procedure, *SUBST* will contain all the substitutions used to unify $L1$ and $L2$.)
5. For $i \leftarrow 1$ to number of arguments in $L1$:
   (a) Call Unify with the /th argument of $L1$ and the ith argument of $L2$, putting result in S.
   (b) If S contains FAIL then return {FAIL}.
   (c) If S is not equal to NIL then:
      (i) Apply S to the remainder of both $L1$ and $L2$.
      (ii) *SUBST* : = APPEND(S, *SUBST*).
6. Return *SUBST*.

have not yet discussed is the check in steps 1(*b*) and 1(*c*) t

Resolution in predicate logic

Algorithm: Resolution

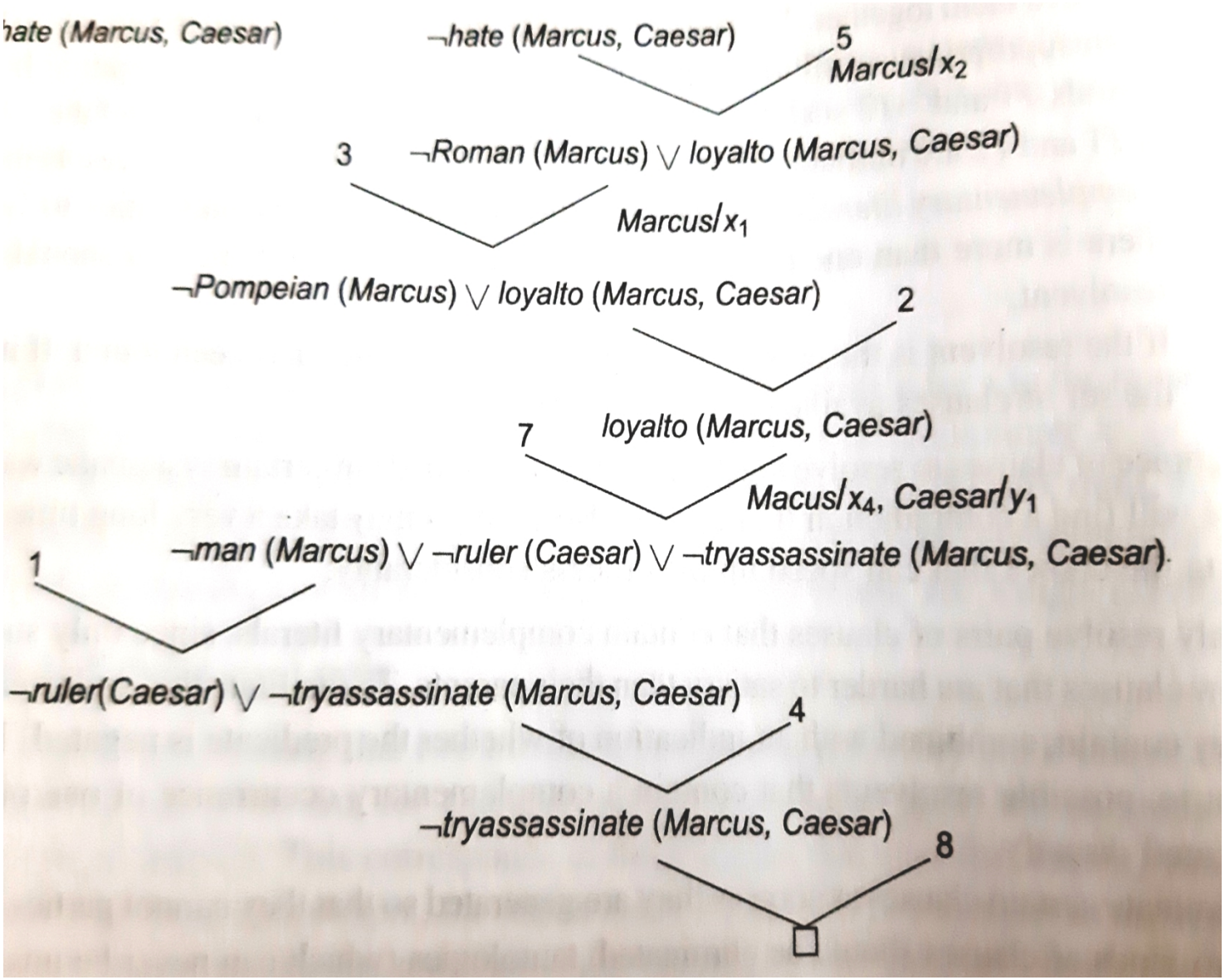
1. Convert all the statements of F to clause form.
2. Negate P and convert the result to clause form. Add it to the set of clauses obttfHied in 1.
3. Repeat until either a contradiction is found, no progress can be made, or a prede- termined amount of effort has been expended.
   (a) Select two clauses. Call these the parent clauses.
   (b) Resolve them together. The resolvent will be the disjunction of all the literals of both parent clauses with appropriate substitutions performed and with the following exception: If there is one pair of literals $T1$ and $\neg T2$ such that one of the parent clauses contains $T2$ and the other contains $T1$ and if $T1$ and $T2$ are unifiable, then neither $T1$ nor $T2$ should appear in the resolvent. We call $T1$ and $T2$ *Complementary literals.* Use the substitution produced by the unification to create the resolvent. If there is more than one pair of complementary literals, only one pair should be omitted from the resolvent.
   (c) If the resolvent is the empty clause, then a contradiction has been found. If it is not, then add it to the set of clauses available to the procedure.

1. Marcus was a man.
2. Marcus was a Pompeian.
3. All Pompeians were Romans.
4. Caesar was a ruler.
5. All Romans were either loyal to Caesar or hated him.
6. Everyone is loyal to someone.
7. People only try to assassinate rulers they are not loyal to.
8. Marcus tried to assassinate Caesar.

Axioms in clause form:

1. $man(Marcus)$
2. $Pompeian(Marcus)$
3. $\neg Pompeian(x_1) \vee Roman(x_1)$
4. $ruler(Caesar)$
5. $\neg Roman(x_2) \vee loyalto(x_2, Caesar) \vee hate(x_2, Caesar)$
6. $loyalto(x_3, fl(x_3))$
7. $\neg man(x_4) \vee \neg ruler(y_1) \vee \neg tryassassinate(x_4, y_1) \vee loyalto(x_4, y_1)$
8. $tryassassinate(Marcus, Caesar)$

(a)


hate (Marcus, Caesar)
¬hate (Marcus, Caesar)
5
Marcus/x2
3
¬Roman (Marcus) ∨ loyalto (Marcus, Caesar)
Marcus/x1
¬Pompeian (Marcus) ∨ loyalto (Marcus, Caesar)
2
7
loyalto (Marcus, Caesar)
Macus/x4, Caesar/y1
1
¬man (Marcus) ∨ ¬ruler (Caesar) ∨ ¬tryassassinate (Marcus, Caesar)
¬ruler(Caesar) ∨ ¬tryassassinate (Marcus, Caesar)
4
¬tryassassinate (Marcus, Caesar)
8

Strategies

1. Only resolve pairs of clauses that contain complementary literals
2. Eliminate certain clauses as soon as they are generated as they cannot participate in later resolutions.1→tauatologies 2→clauses that subsumed by other clauses(PvQ)
3. Whenever possible resolve either with one of clause that is part of statement we are trying to refute or clause generated by a resolution with such a clause
4. Whenever possible resolve the clauses that have single literal

- Two ways of generating new clause in addition to resolution rule

1. Substitution of one value to another to which it is equal
2. Reduction of computable predicates.

Need to try several substitutions

Hate(Marcus,Paulus)

Hate(Marcus,Julian)

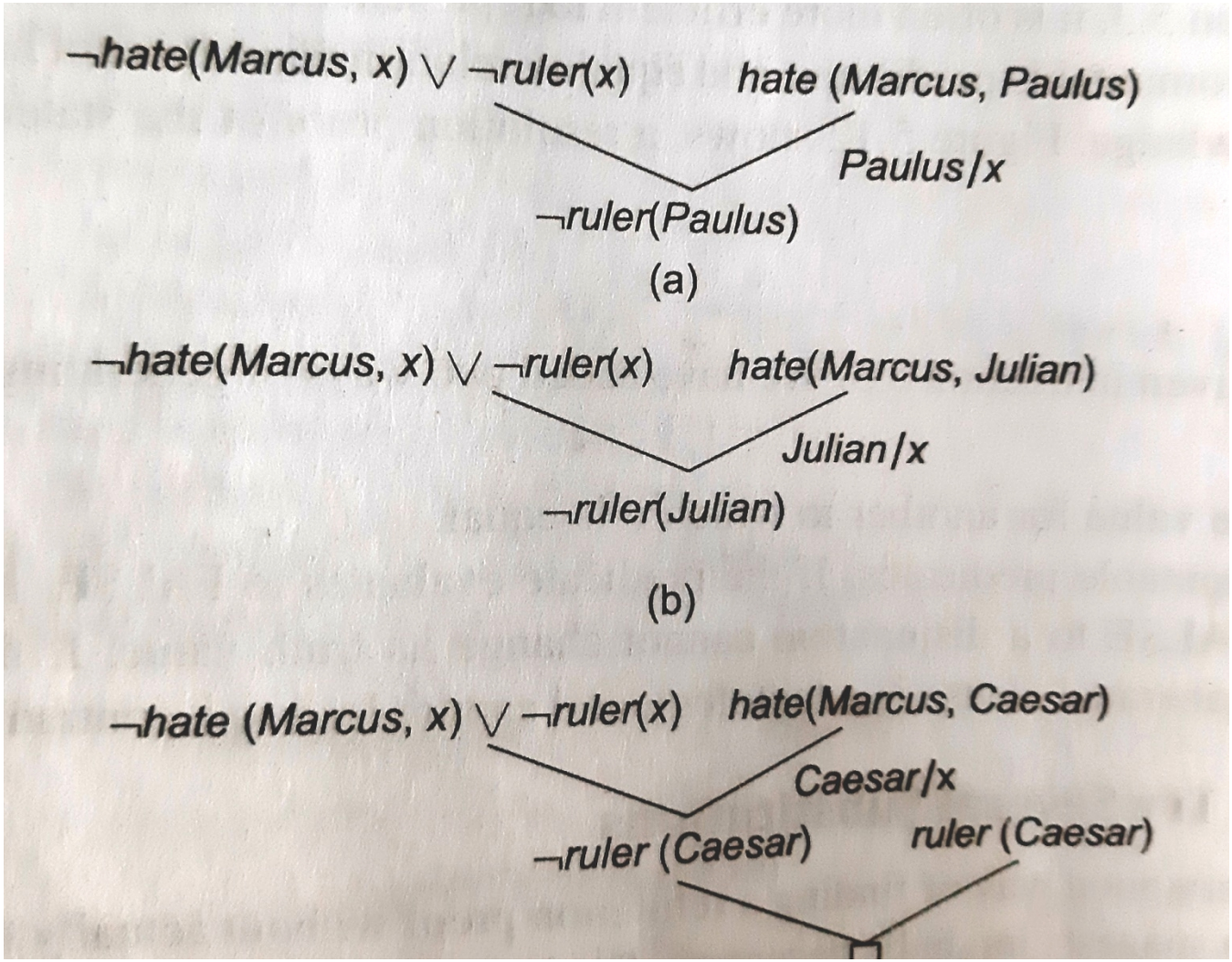
Hate(Marcus,Caesar)

Suppose we want prove “Marcus hates some ruler”

Prove:∃x:hate(Marcus,x) ∧ruler(x)

Negate:¬∃x:hate(Marcus,x) ∧ruler(x)

Clause:¬hate(Marcus,x) ∨¬ruler(x)


¬hate(Marcus, x) ∨ ¬ruler(x)
hate (Marcus, Paulus)
Paulus/x
¬ruler(Paulus)
(a)
¬hate(Marcus, x) ∨ ¬ruler(x)
hate(Marcus, Julian)
Julian/x
¬ruler(Julian)
(b)
¬hate (Marcus, x) ∨ ¬ruler(x)
hate(Marcus, Caesar)
Caesar/x
¬ruler (Caesar)
ruler (Caesar)

Question Answering

- Resolution can be used to answer fill in the blank questions such as “When did Marcus die?” or ”Who tried to assassinate a ruler?”
- Answering these questions involves finding a known statement that matches the term given in the question
- Ex:When did Marcus die?

  Died(Marcus,??) where?? is filled by some particular year.So we prove died(Marcus,79) and respond with the answer 79

Natuaral Deduction

Disadvantages of resolution

- Heuristic information contained in original statement would be lost

Ex:All the judges who are not crooked are well educated.

Logic form ∀x:judge(x)∧¬crooked(x)→educated(x)

Clause form¬judge(x)vcrooked(x)veducated(x)

- Another problem of the Use of resolution of theorem prover is that people donot think in resolution

- We are forced to look for a way of machine theorem proving that corresponds more closely to the process used in human theorem proving. We call this as Natural Deduction
- Natural Deduction describe mélange of techniques used in combination to solve problems that are not tractable by any method alone.
  - Arrange by objects involved in predicates
  - Rewrite the rules, where implication can be exploited in proof

Representing Knowledge using Rules

Representing knowledge using rules

- Procedural versus Declarative Knowledge

Declarative representation →knowledge is specified, but the use to which is not given

Procedural representation →control information necessary to use knowledge is embedded in knowledge itself

Example1

Man(Marcus)

Man(Caesar)

Person(Cleopatra)

∀x:man(x)→person(x)

Answer the question ∃y:person(y)

Example2

Man(Marcus)

Man(Caesar)

∀x:man(x)→person(x)

Person(Cleopatra)

Logic Programming

- PROLOG pgm is series of logical assertions ,each of which is horn clause
- Horn clause is a clause that has atmost one positive literal.p,¬pvq,p→q

Syntactic difference between Logic and PROLOG

- In logic variables are explicitly quantified. In PROLOG implicitly by having all variables as upper case and constants as lower case
- In Logic explicit symbols for (∧) and and (v) or.In PROLOG there is an explicit symbol for and(,) but there is none for or, it is represented as list of alternatives
- In Logic implication p→q.In PROLOG same is written as backward q:-p

- PROLOG P(X):-Q(X,Y)

- Suppose the problem given is to find value of X that satisfies the predicate apartmentpet(X).We state this goal in PROLG as

?-apartmentpet(X)

- Logical negation cannot be represented in pure PROLOG .Instead represented as lack of assertion.
- ∀x:dog(x)→¬cat(x)
- It uses as strategy called negation as failure

?-cat(fluffy)

- Closed world assumption

Forward versus Backward Chaining

- Forward from start states
- Backward from goal states

Example:8 puzzle

- Reason forward from the initial states
- Reason backward from goal states

Factors influence whether to reason forward or backward

- Are there more possible start states or goal states(move form smaller to larger set of states)
- In which direction is the branching factor?(lower branching factor)
- Will the program be asked to justify its reasoning process to a user?(way the user think)
- What kind of event is going to trigger a problem solving episode?(arrival of new fact is forward reasoning ,if it is query reason backward)

Same set of rules can be used for both forward and backward reasoning, but define two classes of rules each of which encodes a particular kind of knowledge

1. Forward rules which encode knowledge how to respond to certain input configuration
2. Backward rules which encode knowledge about how to achieve particular goals

- Backward chaining rule system→ goal directed problem solver. Ex:query system
- Forward chaining rule system->directed by incoming data.Ex:OPS5
- Combining Forward and Backward Reasoning

Matching

1. Indexing(Chess)
2. Matching with variables(RETE)
   - Temporal nature of data
   - Structural similarities in rules
   - Persistence of variable binding consistency
3. Complex and approximate matching
4. Conflict Resolution
   - Preference based on rule that matched
   - Preference based on objects that matched
   - Preference based on action that matched rule would perform

Control Knowledge

- Knowledge about which states are more preferable to others
- Knowledge about which rule to apply in given situation
- Knowledge about the order in which to pursue subgoals
- Knowledge about useful sequence of rules to apply

- Two AI system that represent their control knowledge with rules
  - SOAR
  - PRODIGY