LOGIC,
A SURVEY

T. Hürlimann

Working Paper No. 215

September 1993

INSTITUT D'INFORMATIQUE, UNIVERSITE DE FRIBOURG

INSTITUT FÜR INFORMATIK DER UNIVERSITÄT FREIBURG

Institute of Informatics, University of Fribourg, site Regina Mundi

rue de Faucigny 2

CH-1700 Fribourg / Switzerland

Bitnet: HURLIMANN@CFRUNI51

phone: (41) 37 21 95 60 fax: 37 21 96 70

Logic,
a Survey

Tony Hürlimann, Dr. rer. pol.

Key-words: logic, first-order logic, logic programming

Abstract: This paper gives a concise survey of classical logic from the point of view of computer science. The reader should already be familiar with logic. The emphasis is on first-order logic and on mechanical theorem proving which is the main component of logic programming. The resolution procedure is exposed from a logical point of view. Several logic fragments such as Presburger arithmetic and probabilistic logic are also briefly exposed.

Stichworte: klassische Logik, logische Programmierung.

Zusammenfassung: Dieser Artikel gibt eine konzise Zusammenfassung der klassischen Logik aus Sicht der Informatik. Der Leser sollte bereits mit der Logik und dem Grundvokabular der Informatik vertraut sein. Das Gewicht liegt auf der mechanischen Ableitung. Das Resulotionsprinzip wird von logischer Seite her angegangen. An Schluss sind lose, unsystematische Notizen über Logikfragmente und nicht-klassische Logik angefügt.

INTRODUCTION

Until the 19 century, logic consisted of a couple of simple syllogisms which had to take the rap to prove anything. Boole was the first who formalized systematically the propositional calculus (Boolean logic). This formalisation was extended to a predicate logic by Frege. Since then a intensive research has taken off with the attempt to put the foundation of mathematics on to a logical basis (Russell, Whitehead), especially since the discovery of paradoxes in the set theory (Russells Paradox). There was another research input coming from the Hilbert program to find a finite deductive system that should allow to prove or disprove any formula in logic and mathematics. Unfortunately, it was proved by Gödel 1930 that such a program is impossible.
With the beginning of the Artificial Intelligence (AI) the need for mechanical reasoning and deduction has grown. It was proved by Robinson 1965 that a single deduction principle – the resolution principle – is needed only to prove any valid formula (in first order logic). The resolution principle was subject to many refinements, some off them are easy to implement. It is still the most powerful method for automatic theorem proving.
In the following sections, the syntax and semantic of logic is introduced. Afterwards, a systematic overview of the (basic) resolution method is given. This is the basic know-how to understand logic programming which is summarized in a second paper [Hürlimann 93b].

Classical logic can be classified into a hierachy of logics. The most simplest logic is *propositional logic* (Boolean logic). The basic elements are propositions and connectors. An example is: "if some proposition p is true OR another proposition q is true then proposition r is true too". Symbolically, one may write: $p \vee q \rightarrow r$. Propositional logic is not very powerful. It is a subset of *first order logic* which allows variables and quantifiers. An example is: "For any x the following holds: if x has property P then x has also property Q". Symbolically: $\forall(x)(P(x) \rightarrow Q(x))$. Still a more powerful logic is *second order logic*, where predicates may be variables too: "For any property P the following is true: if Fred has property P then Bill has the property too", symbolically: $\forall(P)(P(Fred) \rightarrow P(Bill))$. The more powerful a logic gets, the more difficult it is to manipulate it. Second order logic is already so powerful that it is (provably) impossible to solve problems formulated in second order logic in the general case. First order logic is powerful enough to state a very large number of (relevant) problems. Therefore, most practical applications are limited to

first order logic. It is difficult enough to solve problems within this framework!

Since some times non-classical logics has emerged. Why do we need non-classical logics? There are a variety of reasons. There are constructs that cannot be handled or cannot be handled conveniently by classical logic: “it is possible that...” or “it is necessary that...” are such constructs. To approach them *modal logic* was invented. Another reason is that propositions may be considered neither true nor false, but something between or something else. This leads us to *multi-valued logics*. Depending on how we look on them, *fuzzy logic* and *probability logic* can be considered as special cases, there are many others. It is needless to justify such logics today. Everybody who knows that most Japanese photo camera uses fuzzy logic to adjust the zoom sees the practical use of such highly abstract constructs. Another direction is *non-monotonic logic*, that was invented to overcome the inconsistencies of a growing knowledge base. Knowledge is accumulated over time in a knowledge base. Very quickly the knowledge becomes inconsistent. For the classical point of view, such a knowledge base becomes useless, since anything can be deduced from an inconsistent statement. It would be necessary to redesign the entire knowledge base. Non-monotonic logic tries to avoid the complete redesign.

In the next sections, we will concentrate on classical logic.

SYNTAX

The alphabet consists of the following classes of symbols:

1) T, F (truth symbols)
2) □ □ Δ ∅ × (not and or implication equivalence)
3) ∀ ∃ (all- and existence-quantifiers)
4) Constants: a) n-ary functions *f, g, h,...,* 0-ary: *a, b,...*
   b) n-ary predicates *p,q,r,...*
5) Variables: a) n-ary functions *F,G,H,...,* 0-ary: *X,Y,...*
   b) n-ary predicates *P,Q,R,...*
6) punctuation symbols: , ( )

The syntax of a logical expression is defined as following:

A *term* is defined as

1) each constant or each variable is a term
2) if $t_1, t_2,...,t_n$ ($n \geq 1$) are terms, then so are $f(t_1, t_2,...,t_n)$ and $F(t_1, t_2,...,t_n)$

An *atom* is defined as

1) T and F are atoms

2) each 0-ary constant or variable predicate is an atom

3) if t_1, t_2,...,t_n ($n \geq 1$) are terms, then $p(t_1, t_2,...,t_n)$ and $P(t_1, t_2,...,t_n)$ are atoms

A *well formed formula* (wff) is defined as

1) each atom is a wff

2) If A and B are wff, then so are $\neg A$, $A \square B$, $A \Delta B$, $A \varnothing B$, $A \times B$

3) if x is a variable and A is a wff, then $\forall xA$ and $\exists xA$ is a wff.

$\neg A$ is called a *negated wff* (sometimes we use also the notation $\bar{A}$ for a negated wff), $A \square B$ is called a *conjunction*, and $A \Delta B$ is called a *disjunction*. A wff will also be called *formula*. Normally, we use uppercase letters for variables and lowercase letters for constants, except for x, y, and z which are used as variables only. Since the roman alphabet is very limited (26 letters), we may use multiletter characters or indexed letters to designate the constants or the variables. We use the uppercase letters A, B, and C for a wff. If A is a wff and x is any variable, we use also the syntax $A[x]$ for saying that within A the variable x occurs. The construct $B \blacklozenge A$ which is more common in programming logic is the same as $A \varnothing B$. The parentheses () are used for precedence evaluation. To avoid having formulas cluttered with brackets, the following precedence hierarchy is adopted:

$\neg \ \forall \ \exists$

$\square$

Δ

$\varnothing \ \times$

If we have the wff $\forall xA[xy]$, then we say that the variable x is *bound*, and the variable y is *free* (since it is not bounded by a quantifier). A formula is called *closed* if all variables are bounded. We may also shorten $\forall x \forall y \forall z$ into $\forall xyz$, and $\exists x \exists y \exists z$ into $\exists xyz$.

A *clause* is a – possibly empty – disjunction of literals. (An empty clause is equivalent to F). A *literal* is a unnegated or a negated atom. A *Horn-clause* is a clause, where at most one literal is unnegated. A *definite clause* is a clause, where exactly one literal is unnegated. A *conjunctive normal form* (*CNF*) is a formula which is a conjunction of a set of clauses. A *disjunctive normal form* (*DNF*) is just a CNF where the two connectives $\square$ and Δ are exchanged.

The class of wff described here is called second-order predicate calculus. There are several subclasses of wffs which are obtained by restricting the set of

constant and variable symbols

1) *propositional calculus*: only the 0-ary (constant) predicates (called propositions) are allowed.
2) *quantified propositional calculus*: only 0-ary (constant and variable) predicates (called propositions) are allowed.
3) *first-order predicate calculus* (= *first-order logic*): any constants (functions and predicates) and 0-ary variable functions (individual variables) are allowed.

In the rest of this paper, we will be concerned mainly with the first-order logic, (therefore, we will also use the letters *P*, *Q*, ... for constant predicates to make the expressions more readable).

SEMANTICS

The semantics of a wff is given by assigning a meaning to each symbol within the formula. We say that we give the formula an *interpretation*.

Example: the formula (Manna p.77)

$$\exists F((F(a)=b)\wedge\forall x(p(x)\rightarrow(F(x)=g(x,F(f(x)))))), \qquad (1)$$

may be interpreted as following:

$x,a,b\in\mathbf{N}$ (x, a, and b are elements of the natural number set).
a=0, b=1, and x is a variable
$f(x)$ means x-1 (f is a function)
$g(x,y)$ means xy (g is the multiplication function)
$p(x)$ means x>0 (p is a predicate)
$F(x)$ means the faculty function x!

Under this interpretation formula (1) becomes true, since there exists a function F (namely the faculty) over **N**, such that $F(0)=1$ and for every $x\in\mathbf{N}$, if $x>0$, then $F(x)=xF(x1)$.

However, if we choose the interpretation

$x,a,b\in\mathbf{N}$ (x, a, and b are elements of the natural number set).
a=0, b=1, and x is a variable
$f(x)$ means x (f is a function)
$g(x,y)$ means y+1 (g is a function)
$p(x)$ means x>0 (p is a predicate)
$F(x)$ means ?

the formula (1) becomes false, since there is no (total) function F over **N**, such that $F(0)=1$ and for every $x\in\mathbf{N}$, if $x>0$, then $F(x)=F(x)+1$.

Generally, an *interpretation* I of a formula *A* consists of the following:

1) a non-empty set *D*, called the *domain*
2) for each constant *c* in *A*, the assignment *c*□*D*
3) for each n-ary function *f* in *A*, the assignment of a mapping $D^n \varnothing D$
4) for each n-ary predicate *p* in *A*, the assignment of a mapping $D^n \varnothing \{T,F\}$.

The meaning of ¬, □, Δ, ∅, and × is as following: if *A* is true then ¬*A* is false. *A*□*B* is true only if both *A* and *B* are true otherwise it is false. *A*Δ*B* is false only if both *A* and *B* are false otherwise it is true. *A*∅*B* is false only if *A* is true and *B* is false otherwise it is true. *A*×*B* is true only if *A* and *B* have the same value otherwise it is false.
∀*x*(*A*[*x*]) is true only if *A*[*x*] is true for every element in the domain. ∃*x*(*A*[*x*]) is true only if *A*[*x*] is true for at least one element in the domain.

An interpretation for which the wff expresses a true statement is called a *model* of the formula. A *theory* is a set of formula with an intended interpretation, naturally the intended interpretation should be a model.

A wff is *valid*, if it is true under all possible interpretations. A valid wff is also called a *tautology*. A wff is *inconsistent* (or *unsatisfiable*) if and only if its negation is valid. A wff is *satisfiable*, if it is true under at least one interpretation. A wff is *nonvalid* if it is false under at least one interpretation. Any formula belongs to one of the family as shown by Figure 1.

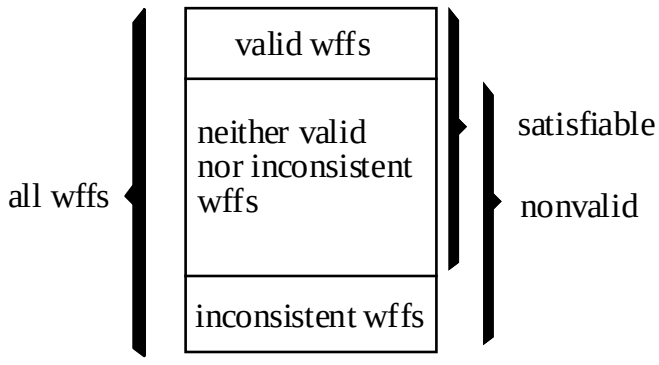


Figure 1

Example: given the formula: $(\forall x)(P(x) \rightarrow Q(f(x), a))$ with domain *D*={1,2 }. An interpretation is:

$a := 1$

$f(1) := 2,\ f(2) := 2$

$P(1) := \mathrm{T},\ P(2) := \mathrm{F}$

$Q(1,1) := Q(1,2) := Q(2,2) := \mathrm{T},\ \ Q(2,1) := \mathrm{F}$

substitute x=1 and x=2, then check the validity:

$x = 1$: $\quad P(1) \to Q(f(1),1)$ gives $\mathrm{T} \to Q(2,1)$ gives F

$x = 2$: $\quad P(2) \to Q(f(2),1)$ gives $\mathrm{F} \to Q(2,1)$ gives T

Hence, the formula is neither valid nor inconsistent, since we have found an interpretation which makes the formula true and another which makes it false.

In the following list (1-14), the expression A □ B means, "B can be substituted be A or vice-versa". We say that the two wffs A and B are *equivalent*, which means that if I is a model of A then I is also a model of B and vice-versa. If the □ symbol is replaced by the equivalence operator $\times$, then the list is made of tautologies. (Q, Q_1, and Q_2 are either $\forall$ or $\exists$, y is a variable which does not appear in $A[x]$.)

1) $A \leftrightarrow B \Leftrightarrow (A \to B) \wedge (B \to A)$

2) $A \to B \Leftrightarrow \neg A \vee B$

3*a*) $A \vee B \Leftrightarrow B \vee A$

3*b*) $A \wedge B \Leftrightarrow B \wedge A$

4*a*) $(A \vee B) \vee C \Leftrightarrow A \vee (B \vee C)$

4*b*) $(A \wedge B) \wedge C \Leftrightarrow A \wedge (B \wedge C)$

5*a*) $A \vee (B \wedge C) \Leftrightarrow (A \vee B) \wedge (A \vee C)$

5*b*) $A \wedge (B \vee C) \Leftrightarrow (A \wedge B) \vee (A \wedge C)$

6*a*) $A \vee \mathrm{F} \Leftrightarrow A$

6*b*) $A \wedge \mathrm{F} \Leftrightarrow \mathrm{F}$

7*a*) $A \vee \mathrm{T} \Leftrightarrow \mathrm{T}$

7*b*) $A \wedge \mathrm{T} \Leftrightarrow A$

8*a*) $A \vee \neg A \Leftrightarrow \mathrm{T}$

8*b*) $A \wedge \neg A \Leftrightarrow \mathrm{F}$

9) $\neg(\neg A) \Leftrightarrow A$

10*a*) $\neg(A \vee B) \Leftrightarrow \neg A \wedge \neg B$

10*b*) $\neg(A \wedge B) \Leftrightarrow \neg A \vee \neg B$

11*a*) $QxA[x] \vee B \Leftrightarrow Qx(A[x] \vee B)$

11*b*) $QxA[x] \wedge B \Leftrightarrow Qx(A[x] \wedge B)$

12*a*) $\neg(\forall xA[x]) \Leftrightarrow \exists x(\neg A[x])$

12*b*) $\neg(\exists xA[x]) \Leftrightarrow \forall x(\neg A[x])$

13*a*) $\forall xA[x] \wedge \forall xB[x] \Leftrightarrow \forall x(A[x] \wedge B[x])$

13*a*) $\exists xA[x] \vee \exists xB[x] \Leftrightarrow \exists x(A[x] \vee B[x])$

14*a*) $Q_1xA[x] \vee Q_2xB[x] \Leftrightarrow Q_1xQ_2y(A[x] \vee B[y])$

14*b*) $Q_1xA[x] \wedge Q_2xB[x] \Leftrightarrow Q_1xQ_2y(A[x] \wedge B[y])$

(Note that $\exists x\,(P(x) \wedge Q(x))$ is not equivalent to $\exists x\,(P(x)) \wedge (\exists x)\,(Q(x))$, and $\forall x\,(P(x) \vee Q(x))$ is not equivalent to $\forall x\,(P(x)) \vee (\forall x)\,(Q(x))$.)

A *prenex normal form (PNF)* is a wff of the form

$$Q_1x_1Q_2x_2\mathrm{K}\ Q_nx_n(M)$$

where M is a wff which does not contain any quantifiers. M is called the *matrix* and $Q_1x_1Q_2x_2\mathrm{K}\ Q_nx_n$ is called the *prefix.* Every wff can be transformed into a PNF using the substitutions 1-14 (above). The procedure goes as following:

1) eliminate all connectives except ¬, □, and Δ.
2) push the ¬ inwards as much as possible (and eliminate double negations (□□)
3) rename bounded variable if necessary
4) push the quantifiers to the right.

If M is a CNF, then the wff is called a *prenex-conjunctive normal form* (cPNF), if M is a DNF then the wff is called a *prenex-disjunctive normal form* (cPNF). Since M is quantifier-free, it is easy to translate M into a CNF (or a DNF) by applying the rules 1-14. The procedure goes as following

1) eliminate all connectives except ¬, □, and Δ.
2) push the ¬ inwards as much as possible
3) distribute the Δ connectives over the connectives □ (or the □ connectives over the Δ connectives for a DNF)

The procedure is easy, but it may take exponential time in the number of literals to produce the CNF. The prenex normal form of a wff is, in general, not unique.

There is a important *duality theorem* between the two prenex forms which says: If A is a prenex-conjunctive normal form and B is a prenex-disjunctive normal form which was obtained from A by

1) replacing all □ by Δ and vice-versa
2) replacing all ∀ by ∃ and vice-versa
3) negating all literals in the matrix of *A*

then *A* and ¬*B* are equivalent.

A *Skolem standard form* is a wff of the form

$$\forall x_1 x_2 \mathrm{K}\ x_n (M[x_1 x_2 \mathrm{K}\ x_n])$$

Normally, we cannot produce – by a sequence of substitutions – a Skolem standard form from any wff. But we can produce a Skolem standard form *S* from every wff *A*, such that *S* is inconsistent if and only if *A* is inconsistent (Theorem of Skolem). This is a very important theorem (for a proof see Chang/Lee p.48). The transformation goes as following, given a wff *A*

1) find a prenex-conjunctive normal form of *A*
2) Now we have a wff of the form $Q_1 x_1 Q_2 x_2 \mathrm{K}\ Q_n x_n (M)$. suppose Q_r ($1 \le r \le n$) is a ∃-quantifier. If no quantifier occurs on the left (r=1), then we choose a new constant *c* and replace every x_r in *M* by *c* and we delete $\exists x_r$ from the prefix. If the (all-)quantifiers $Q_1 x_1 Q_2 x_2 \mathrm{K}\ Q_m x_m$ (m<r) occurs on the left, then we choose a new m-ary function $f(x_1,x_2,...,x_m)$ and replace every x_r in *M* by $f(x_1,x_2,...,x_m)$ and we delete $\exists x_r$ from the prefix. This step is repeated until all ∃-quantifiers are eliminated.

A very nice example for this transformation is given by Chang/Lee p.49: Proof that if $x \cdot x = e$ for all *x* in group G, then G is commutative (see example 7 below).

A Skolem standard form of a wff (where the matrix is in CNF) can be represented just by the set of clauses within the matrix: We write this fact as following: $\{C_1,C_2,...,C_n\}$ (all variables are supposed to be bounded by all-quantifiers, of course). We will use the term "*set of clauses*" interchangeably with the term "*Skolem standard form*".

THE VALIDITY PROBLEM

A formula *A* is said *provable* or *derivable*) from a set of clauses $\{C_1,C_2,...,C_n\}$, if the wff $C_1 \wedge C_2 \wedge \mathrm{K} \wedge C_n \rightarrow A$ is valid. This is the same as to say that $C_1 \wedge C_2 \wedge \mathrm{K} \wedge C_n \wedge \neg A$ is inconsistent or that the set of clauses $\{C_1,C_2,...,C_n, \neg A\}$ is unsatisfiable. The *validity problem* of a wff is to determinate whether the formula is valid or if a formula *A* is provable from a set of clause $\{C_1,C_2,...,C_n\}$. Normally, we – in computer science – are more interested in the second case where a wff is provable from a set of clauses. In this context, the set of clauses

is also called a *knowledge base*.

We say that a problem is *solvable*, if we can build a Turing machine which will take a wff as input and determinates whether the wff is valid or not (that is, the Turing machine reaches an accept- or a reject-halt-state in a finite time), otherwise the problem is called *unsolvable*. An (unsolvable) problem is *partially solvable* (or *semi-solvable*), if we can build a Turing machine that takes a wff as input and reaches an accept-halt-state if the wff is indeed valid, and reaches a reject-halt-state or loops forever if the wff is nonvalid. Without proof, we state (see Manna, p.105ff):

1) The validity problem of a wff of second-order predicate calculus is unsolvable (not even partially solvable) (Gödel).
2) The validity problem of a wff of first-order predicate calculus is unsolvable, but partially solvable (Church).
3) The validity problem of a wff of second-order predicate calculus, where all constants and variables have 0-arity or the predicates have at most 1-arity, is solvable (see Manna p.107). (Note that the equality function is also allowed). (Note that this implies also that the validity problem of the propositional calculus is solvable).
4) The validity problem of a wff of first-order predicate calculus, where the wff is in one of the following three prenex normal form
   $\forall x_1 x_2 \mathrm{K}\ x_m \exists x_{m+1} \mathrm{K}\ x_n (M)$
   $\forall x_1 x_2 \mathrm{K}\ x_m \exists x_{m+1} \forall x_{m+2} \mathrm{K}\ x_n (M)$
   $\forall x_1 x_2 \mathrm{K}\ x_m \exists x_{m+1} x_{m+2} \forall x_{m+3} \mathrm{K}\ x_n (M)$
   is solvable.

The whole business in logic is the solve the validity problem of some wff. Many methods have been proposed to solve the validity problem. One method is *natural deduction*. Gentzen has proposed such a system (1934) (see Manna p. 108ff for a overview) which is convenient for solving the validity problem 'manually'. (Note that Fitting (p.82) calls this method sequent calculus). Another – for mechanical deduction more interesting method – is *resolution*. Resolution was invented by Robinson (1965) and is, to some extent, a variant of the Davis-Putnam method (1960) which is much less efficient. Since we are mostly interested in mechanical (computer-based) deduction, only resolution will be treated here. The Robinson article was a very important, influential paper for the whole development of modern logic in computer science. For a fascinating historical account of this discovery see Robinson [1992].

THE HERBRAND THEOREM

Using resolution, the validity problem of a wff is solved by a refutation procedure (if it exists). That means to proof the validity of a wff A, we proof that the wff $\neg A$ is inconsistent together with the knowledge base.
By definition, a set of clauses is unsatisfiable if and only it is false under *all* interpretations over *all* domains. Of course, it is impossible to consider all interpretation over all domains. But – luckily – we can fix a domain H – called the *Herbrand universe* – such that a set of clauses C is unsatisfiable if and only if C is false under all interpretation of domain H. This is a very important theorem which was proved by Herbrand in 1930. The Herbrand universe H of a set of clauses C can be constructed using the following procedure:

1) let H_o be the set of all constants within the set of clauses A. If there is no constant then choose a arbitrary constant a as the single element of H_o.
2) For i=0,1,2,... let H_{i+1} be the union of H_i and the set of all functions f of the form $f(t_1,t_2,...t_n)$ occurring in A, where t_i are members of H_i.
3) H (the Herbrand universe) is defined as $\lim_{i\varnothing\square} H_i$.

Let S be a set of clauses. The set of atoms of S which are obtained by replacing all variables within the atoms by members of the Herbrand universe is called *Herbrand base* of S or *atom set* of S. Let $\{A_1, A_2, A_3, ...\}$ be an atom set of a set of clauses. then a *H-interpretation* is a set $\{m_1, m_2, m_3, ...\}$ where every m_i with i=1,2,3,... is either A_i or $\neg A_i$. Every interpretation of a set of clauses can be mapped into a H-interpretation, that's what the mentioned theorem says. If the set of clauses, for example, is: $\{P(x) \vee Q(x), R(f(x))\}$, the Herbrand universe is $\{a, f(a), f(f(a)), ... \}$. The atom set is $\{P(a), Q(a), R(a), P(f(a)), Q(f(a)), R(f(a)), ...\}$. Consider the (arbitrary) interpretation I with domain D={1,2}, and $f(1)=2$, $f(2)=1$, $P(1)$=F, $P(2)$=F, $Q(1)$=T, $Q(2)$=F, $R(1)$=F, $R(2)$=T. If we map 1 into a, we have $P(a)$=F, $Q(a)$=T, $R(a)$=F, $P(f(a))$=F, $Q(f(a))$=F, $R(f(a))$=T, ... Therefore, a H-interpretation of I is $\{\neg P(a), Q(a), \neg R(a), P(f(a)), \neg Q(f(a)), R(f(a)), ...\}$. Now we have the first part of the important Herbrand theorem: *A set S of clauses is unsatisfiable if and only if S is false under all the H-interpretations of S* (for a proof see Chang/Lee p 55).

Let S be a set of clauses and $A = \{A_1, A_2, A_3, ...\}$ the corresponding atom set. *A semantic tree* of S is a tree T, where each link is attached with a finite set of atoms or there negations of A such that

1) each node N contains finitely many links $L_1, L_2, ... , L_n$. Let Q_i be the

conjunction of all atoms attached to a link L_i where i={1,2,...,*n*}. Then $Q_1 \Delta Q_2 \Delta ... \Delta Q_n$ is a valid formula.

2) For each node N, let I(N) be the union of all the sets attached to the links on the path from the root of the tree down to N, then I(N) does not contain a set which contains an atom and its negation at the same time.

The semantic tree is called *complete*, if for every leaf N_L, $I(N_L)$ contains either A_i or $\neg A_i$ with i={1,2,3,...}. If the atom set *A* is infinite, the corresponding semantic tree *T* is also infinite. One can easy verify that the complete semantic tree corresponds to a exhaustive survey of all interpretations; more precisely, for every leaf N_L, $I(N_L)$ is an interpretation. If *S* is unsatisfiable, then *S* fails to be true for each interpretation $I(N_L)$.

A node N is a *failure node* if I(N) falsifies some ground instance of a clause in *S*, but I(N') does not falsify some ground instance of a clause in *S*, where N' is the ancestor node of N. (A *ground instance* of a clause *C* is a clause obtained by replacing variables in *C* by members of the Herbrand universe). A semantic tree T is *closed* if and only if every branch of T terminates at a failure node.

Example: Suppose the set *S* of clauses is: $\{P(x), \neg P(x) \vee Q(f(x)), \neg Q(f(a))\}$. The corresponding atom set is $\{P(a), Q(a), P(f(a)), Q(f(a)), ...\}$. A closed semantic tree of S is shown in Figure 2. (The leaves are all failure nodes).

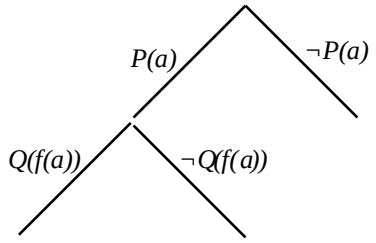


Figure 2

Now we are ready to state the Herbrand theorem:

Version I: *A set S of clauses is unsatisfiable if and only if there exists a finite closed semantic tree*.

Version II: *A set S of clauses is unsatisfiable if and only if there exists a finite unsatisfiable set S' of ground instances of clauses of S*.

The proof follows easily from the definition of a semantic tree. (see Chang/Lee p.61).

Example: The set *S* of clause is $\{P(x), \neg P(f(a))\}$. A closed semantic tree (in fact

the smallest one) is shown in Figure 3.

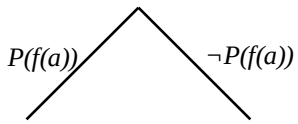


Figure 3

A finite set of ground instances replacing x by $f(a)$, which makes S unsatisfiable is $S'=\{P(f(a)), \neg P(f(a))\}$. One can easy see that S' is itself unsatisfiable.

Let's summarize what we have done so far: To prove a formula A we transform it into the prenex-conjunctive normal form (cPNF). That can always be done using the given substitutions. Next we find a Skolem standard form S which is not necessarily equivalent to the original formula A, but S is unsatisfiable if and only if A is unsatisfiable. The Skolem standard form can be represented as a set of clauses. Next we map the interpretation of S into the Herbrand universe of S, and finally, we look for a finite set of ground instances of the clauses of S which is unsatisfiable. Therefore, we can determinate in a finite length of time, whether the original formula A is unsatisfiable or not. If, however, A were satisfiable, we would have no general procedure to prove it.

RESOLUTION

Normally, it is not so easy to find an unsatisfiable finite set of ground instances from a given set of clauses. Gilmore (1960) was the first to implement a procedure which finds such a set if it exists. He systematically searched through the Herbrand universe. This can be very inefficient (for an impressive example see Chang/Lee p.70).

A much more efficient method is resolution: Instead of finding an unsatisfiable finite set of ground instances, resolution is directly applied to the set S of clauses. The essential idea is to check whether S contains the empty clause. If S contains the empty clause, then S is unsatisfiable. If S does not contain the empty clause, then we have to check whether the empty clause can be derived (= proved) from S.

As an example, suppose S is $\{p, \neg p\}$. There is no empty clause F within S, but the empty clause can be proved, since $p \square\square p$ is unsatisfiable (remember that the empty clause is equivalent to F (false); we may simply identify an empty clause with a contradiction such as the clause set $\{p, \neg p\}$).

To take another example from propositional logic, suppose we have the set S of

clauses: $\{p \vee q \vee r, s \vee \neg q \vee t\}$. Which clauses can be derived (proved) from S? We may derive $p \vee r \vee s \vee t$, since $((p \vee q \vee r) \wedge (s \vee \neg q \vee t) \rightarrow (p \vee r \vee s \vee t)$ is a valid formula.

More generally, let S be a set of clauses. For any two clauses C_1 and C_2 within S, if there is a literal L_1 in C_1 that is complementary to a literal L_2 in C_2, then delete L_1 from C_1 and L_2 from C_2, and construct the disjunction of the remaining clauses. The constructed clause is the *resolvent* of C_1 and C_2. Add the resolvent to the original set of clauses. (One may note that the resolvent is nothing else than a logical consequence of two clauses). Repeat this procedure until the empty clause shows up or, loosely speaking, all combinations have been tried. (What 'trying all combinations' means exactly will be clear shortly).

Example: $\{ p \Delta r , \neg p \Delta q \}$. the first clause contains p and the second contains $\neg p$. We get the resolvent $r \Delta q$. The new set of clauses is $\{ p \Delta r , \neg p \Delta q , r \Delta q \}$. We have 'tried all combinations', therefore, we are done. The empty clause is not in the set. Therefore, the original set of clause is *not* a contradiction.

To find the resolvent of two clauses in first order logic, an additional step must taken place. Suppose the set S of clauses contains two clauses $\{C_1, C_2\} = \{P(x) \vee Q(x), \neg P(f(x)) \vee R(x)\}$. There is no complementary literal in both clauses. However, if we substitute $f(a)$ for x in the first clause and a for x in the second clause, we get the set of clauses $\{P(f(a)) \vee Q(f(a), \neg P(f(a)) \vee R(a)\}$. Now we have a complementary pair and we get the resolvent C_3: $Q(f(a) \vee R(a)$. If, however, we substitute $f(x)$ for x in C_1 we obtain a different resolvent C_3^*: $Q(f(x) \vee R(x)$ which is more general than the first one(C_3). C_3^* is the most general resolvent which can be obtained from the two clauses C_1 and C_2. There exists an procedure – called *unification* – which produces the most general substitution term, if it exists, in order to obtain the most general resolvent.

To state the unification algorithm, we have to introduce some definitions: A *substitution* is a finite (possibly empty) set of the form $\{t_1/v_1,...,t_n/v_n\}$ where every v_i is a variable and every t_i is a term different from v_i, and all v_i are different from each other. If no t_i contains a variable, the substitution is called a *ground substitution*. If $\theta=\{t_1/v_1,...,t_n/v_n\}$ is a substitution and A is a wff, then $A\theta$ is the expression obtained by replacing all variables $v_1,...,v_n$ by the terms $t_1,...,t_n$. If $\theta=\{t_1/v_1,...,t_n/v_n\}$ and $\lambda=\{u_1/y_1,...,u_m/y_m\}$ are two substitution then the *composition* of θ and λ, denoted $\theta\lambda$, is defined as $\{t_1\lambda/v_1,...,t_n\lambda/v_n,$

$u_1/y_1,...,u_m/y_m\}$. A substitution θ is called *unifier* for a set of wff $\{A_1, ... , A_k\}$ if and only if $A_1\theta = ... = A_k\theta$, and the set $\{A_1, ... , A_k\}$ is called *unifiable*. The *disagreement set* of a set $\{A_1, ... , A_k\}$ of wffs is a non-empty set D of wffs which is obtained by locating the first symbol (counting from the left) at which not all wffs A_i have exactly the same symbol, and then extracting from each wff A_i the subexpressions that begin with the symbol occupying that position.
Example: Given the set of wff $\{A_1, A_2\} = \{P(a,x,f(g(y))) , P(z,f(z),f(u))\}$. The substitution $\{a/z\}$ applied to A_2 gives $A_2\{a/z\} = P(a,f(a),f(u))\}$. The disagreement set of $\{A_1, A_2\}$ is $\{a,z\}$. A unifier of $\{A_1, A_2\}$ is $\{a/z, f(a)/x, g(y)/u\}$.

Now we can formulated the **unification algorithm** which finds the most general unifier of a set A of wffs:

Step 1 Let k=0, A_k=A, and θ_k={}

Step 2 if A_k is a singleton, stop and θ_k is the most general unifier for A, otherwise find the disagreement set D_k for A_k.

Step 3 If there exist a variable v_k and a term t_k (such that v_k does not occur within t_k) in D_k, go to Step 4 otherwise stop; A is not unifiable.

Step 4 Now let $\theta_{k+1}=\theta_\kappa\{t_k/v_k\}$ and $A_{k+1}=A_k\{t_k/v_k\}$. (note that $A_{k+1}=A_k\theta_{k+1}$).

Step 5 set k=k+1 and go to Step 2

One can prove that the algorithm terminates and returns the most general unifier if it exists (for a proof see Chang/Lee p.79).

To state the resolution principle in first order logic, some more definitions are needed. If two or more literals in a clause C have a most general unifier θ, then Cθ is called a *factor* of C. Let C_1 and C_2 be two clauses with no variables in common (if common variables show up then we must rename them). Let L_1 and L_2 be two literals in C_1 and in C_2, respectively. If L_1 and $\neg L_2$ have a most general unifier θ, then the clause $(C_1\theta - L_1\theta) \approx (C_2\theta - L_2\theta)$ is called a *(binary) resolvent* of C_1 and C_2. A clause C_1 *subsumes* another clause C_2 if and only if there exists a substitution θ such that $C_1\theta \prod C_2$. C_2 is called a *subsumed* clause.
Example: Let C be the clause $P(x)\Delta P(f(y))\Delta\neg Q(x)$. Then the first and second literals have the most general unifier $\theta=\{f(y)/x\}$. Hence, $C\theta = P(f(y)\Delta\neg Q(f(y))$ is a factor of C. Let $C_1 = P(x)\Delta Q(x)$ and $C_2 = \neg P(a)\Delta R(y)$ be two clauses: choosing the unifier $\theta=\{a/x\}$ and the two literals L_1=$P(x)$ from C_1 and L_2=$\neg P(a)$ from C_2, then $Q(a)\Delta R(y)$ is a resolvent. Let $P(x)$ and $P(a)\Delta Q(a)$ be two clauses.

If θ is the substitution $\{a/x\}$ then $P(a)\theta$ is $P(a)$. Since $P(a) \prod P(a)\Delta Q(a)$, $P(a)$ subsumes $P(a)\Delta Q(a)$.

The **resolution principle** in first order logic can now be stated as following: Let *S* be a set of clauses. Choose two clauses with complementary literals, find their most general unifier and add the corresponding resolvent to the set *S* of clauses. If the resolvent is a tautology or is subsumed by another clause within *S*, then drop it. Repeat this procedure until the empty clause shows up or no resolvent different from any clause in the (augmented by resolvents) *S* can be produced.
The resolution is complete which means that given a unsatisfiable set *S* of clauses, there exists a deduction by resolution (by repeatedly producing resolvents) which yields the empty clause. This does not mean that every such deduction terminates. Somethime it depends in which order the resolvents are produced. Consider the example with the set *S* of the three clauses:

$1) \neg P(x) \vee P(f(x))$

$2) P(a)$

$3) \neg P(f(a))$

using 1) and 2) we get

$4) P(f(a))$

using 3 and 4, we get the empty clause

If we apply, however, the rules in a different order - as Prolog does - we have:
using 1) and 2), we get as before

$4) P(f(a))$

using 1) and 4), we get

$5) P(f(f(a)))$

using 1) and 5), we get

$6) P(f(f(f(a))))$

.... and so on ....

The procedure does not terminate, although the set *S* is unsatisfiable. If the set *S* of clauses is satisfiable, we are even worse off: the procedure may run forever since the validity problem of first order logic is semi-decidable as we know already.

There are many heuristics which try to define a order of the clauses (or the literals within the clauses):

- **Semantic resolution** groups the clauses into two disjoints sets. Two clauses which are in different set are only candidates for a resolution. The problem is how to build the groups. Hyperresolution and set-of-support strategy are two special cases of semantic resolution.
- **Lock resolution** is based on indexing all literals in the set of clauses. Resolution is only allowed between the two literals of two clauses with the lowest index.
- **Linear resolution** starts with a clause (the top clause), resolves it against a clause to obtain a resolvent, and resolves this resolvent against some clause until the empty clause is obtained. This strategy is especially easy to implement (most Prologs are based upon). Input resolution and unit resolution – both not complete – are special cases of linear resolution.

APPLICATIONS AND EXAMPLES

This section gives some examples, build to exercise the resolution 'by hand'.

EXAMPLE 1: CHEMICAL SYNTHESIS PROBLEM (CHANG/LEE P.21):

The following chemical reactions are given:

$MgO + H_2 \varnothing Mg + H_2O$

$C + O_2 \varnothing CO_2$

$CO_2 + H_2O \varnothing H_2CO_3$

Suppose we have some quantities of MgO, H_2, O_2, and C. Show that we can make H_2CO_3. Suppose that the molecules *are* some propositions then we can build the knowledge base as following:

$MgO \square H_2 \varnothing Mg \square H_2O$

$C \square O_2 \varnothing CO_2$

$CO_2 \square H_2O \varnothing H_2CO_3$

together with the four facts: MgO, H_2, O_2, and C.

Formulated in CNF, this gives:

1a) $\neg MgO \Delta \neg H_2 \Delta Mg$

1b) $\neg MgO \Delta \neg H_2 \Delta H_2O$

2a) $\neg C \Delta CO_2$

2b) $\neg O_2 \Delta CO_2$

3a) $\neg CO_2 \Delta H_2CO_3$

3b) $\neg H_2O \Delta H_2CO_3$

4) MgO (four facts)

5) H_2

6) O_2

7) C

8) $\neg H_2CO_3$ (add the negation of the goal)

Applying resolution we get from (8) and (3a)

9) $\neg CO_2$

from (9) and (2a) get

10) $\neg C$

and from (10) and (7) we get the empty clause. Therefore H_2CO_3 has been proved.

EXAMPLE 2:

Consider the propositions: p = "Fred likes beer", q = "Bill likes beer", r = "Fred likes Bill", and s = "Bill likes Fred". Let's the knowledge base be $(p \rightarrow q) \rightarrow (r \rightarrow s)$, p, q. Now we wanted to prove r!

Translate the first axiom into a CNF:

$\neg(\neg p \vee q) \vee (\neg r \vee s)$ (using $p \rightarrow q \leftrightarrow \neg p \vee q$)

$(p \wedge \neg q) \vee (\neg r \vee s)$ (using the law of Morgan $\neg(p \vee q) \leftrightarrow \neg p \wedge \neg q$, $\neg(p \wedge q) \leftrightarrow \neg p \vee \neg q$)

$(\neg p \vee \neg q \vee r) \wedge (\neg p \vee \neg q \vee s)$ (using $p \vee (q \wedge r) \leftrightarrow (p \vee q) \wedge (p \vee r)$, $p \wedge (q \vee r) \leftrightarrow (p \wedge q) \vee (p \wedge r)$)

So we have got the following clauses:

1a. $\neg p \vee \neg q \vee r$

1b. $\neg p \vee \neg q \vee s$

2. p

3. q

To prove r, add its negation to the knowledge base:

4. $\neg r$

5. $\neg q \vee r$ (resolution between 1a and 2)

6. r (resolution between 3 and 5)

7. > (resolution between 4 and 6 produces the empty clause)

Hence the negation of r together with the knowledge base produces a contradiction, therefore, r is true and provable from the knowledge base.

EXAMPLE 3: (CHENG/LEE, P.40)

Problem: "No used-car dealer buys a used car for his family. Some people who buy used cars for their families are absolutely dishonest. Is there any absolutely dishonest people which is not a used-car dealer?"

Using the three predicates $U(x)$, $B(x)$, and $D(x)$ for "x is a used-car dealer", "x

buys a used car for his family", and "x is absolutely dishonest", we can formulate this knowledge in first order logic as following (note that, although the predicate are not variables, we use uppercase letters for them. This is to make the wff more readable):

1) $\forall x(U(x) \rightarrow \neg B(x))$

2) $\exists x(B(x) \wedge D(x))$

3) $\neg\exists x(D(x) \wedge \neg U(x))$ (the negation of the goal)

Translated into a cPNF this is

1) $\forall x(\neg U(x) \vee \neg B(x))$

2a) $B(a)$

2b) $D(a)$

3) $\forall x(\neg D(x) \vee U(x))$

Dropping the quantifiers we get the set of clauses

1) $\neg U(x) \vee \neg B(x)$

2a) $B(a)$

2b) $D(a)$

3) $\neg D(x) \vee U(x)$

Resolving (1) and (2a) with unification {a/x} we get

4) $\neg U(a)$

Resolving (2b) and (3) with unification {a/x} we get

5) $U(a)$

Resolving (4) and (5) produce the empty clause, therefore (3) is proved.

EXAMPLE 4: (DAVIS/WEYUKER P.269):

Problem: "Every shark eats a tadpole; all large white fish are sharks; some large white fish live in deep water; any tadpole eaten by a deep water fish is miserable; prove that: some tadpoles are miserable." Using the definitions $T(y)$, $E(x,y)$, $S(x)$, $B(x)$, $R(x)$, and $M(x)$ means "y is a tadpole", "x eats y", "x is a shark", "x is a large white fish", "x lives in deep water", and "x is miserable", we may formulate the knowledge base as:

$\forall x\,(S(x) \rightarrow \exists y\,(T(y) \wedge E(x,y)))$

$\forall x\,(B(x) \rightarrow S(x))$

$\exists x\,(B(x) \wedge R(x))$

$\forall xy\,(R(x) \wedge T(y) \wedge E(x,y) \rightarrow M(y))$

PROVE: $\exists y\,(T(y) \wedge M(y))$

So we need to prove the unsatisfiability of

$$\forall x(S(x) \rightarrow \exists y(T(y) \wedge E(x,y)))$$
$$\wedge \forall x(B(x) \rightarrow S(x))$$
$$\wedge \exists x(B(x) \wedge R(x))$$
$$\wedge \forall xy(R(x) \wedge T(y) \wedge E(x,y) \rightarrow M(y))$$
$$\wedge \neg(\exists y(T(y) \wedge M(y)))$$

Eliminate $\varnothing$ and moving $\neg$ inwards gives

$$\forall x(\neg S(x) \vee \exists y(T(y) \wedge E(x,y)))$$
$$\wedge \forall x(\neg B(x) \vee S(x))$$
$$\wedge \exists x(B(x) \wedge R(x))$$
$$\wedge \forall xy(\neg R(x) \vee \neg T(y) \vee \neg E(x,y) \vee M(y))$$
$$\wedge (\forall y(\neg T(y) \vee \neg M(y)))$$

Renaming variables gives

$$\forall x(\neg S(x) \vee \exists t(T(t) \wedge E(x,t)))$$
$$\wedge \forall z(\neg B(z) \vee S(z))$$
$$\wedge \exists u(B(u) \wedge R(u))$$
$$\wedge \forall vw(\neg R(v) \vee \neg T(w) \vee \neg E(v,w) \vee M(w))$$
$$\wedge (\forall y(\neg T(y) \vee \neg M(y)))$$

pulling quantifiers to the right (first the $\exists$) gives the PNF

$$\exists u \forall x \exists t \forall z v w y\ ((\neg S(x) \vee (T(t) \wedge E(x,t)))$$
$$\wedge (\neg B(z) \vee S(z))$$
$$\wedge (B(u) \wedge R(u))$$
$$\wedge (\neg R(v) \vee \neg T(w) \vee \neg E(v,w) \vee M(w))$$
$$\wedge (\neg T(y) \vee \neg M(y)))$$

eliminating all $\exists$-quantifiers gives the Skolem standard form (which is not equivalent any more with the last PNF)

$$\forall x \forall z v w y\ ((\neg S(x) \vee (T(g(x)) \wedge E(x, g(x))))$$
$$\wedge (\neg B(z) \vee S(z))$$
$$\wedge (B(c) \wedge R(c))$$
$$\wedge (\neg R(v) \vee \neg T(w) \vee \neg E(v,w) \vee M(w))$$
$$\wedge (\neg T(y) \vee \neg M(y)))$$

The set of clauses is therefore

$$\{(\neg S(x) \vee T(g(x)),$$
$$\neg S(x) \vee E(x, g(x)),$$
$$\neg B(z) \vee S(z),$$
$$B(c),$$
$$R(c),$$
$$\neg R(v) \vee \neg T(w) \vee \neg E(v, w) \vee M(w),$$
$$\neg T(y) \vee \neg M(y)\}$$

The Herbrand universe H is $\{c, g(c), g(g(c)), ...\}$.

Using resolution, we may unify $S(z)$ and (both) $\neg S(x)$ using the substitution $\{x/z\}$ to get

$\{(\neg B(x) \vee T(g(x)),$
$\neg B(x) \vee E(x, g(x)),$
$B(c),$
$R(c),$
$\neg R(v) \vee \neg T(w) \vee \neg E(v, w) \vee M(w),$
$\neg T(y) \vee \neg M(y)\}$

We may unify now $\neg E(v,w)$ and $E(x,g(x))$ which leads to the substitution $\{v/x, g(x)/w\}$, so we get

$\{(\neg B(x) \vee T(g(x)),$
$B(c),$
$R(c),$
$\neg R(x) \vee \neg T(g(x)) \vee \neg B(x) \vee M(g(x)),$
$\neg T(y) \vee \neg M(y)\}$

Unifying $B(c)$ with (both) $\neg(B(x)$ using the substitution $\{c/x\}$ yields

$\{T(g(c)),$
$R(c),$
$\neg R(c) \vee \neg T(g(c)) \vee M(g(c)),$
$\neg T(y) \vee \neg M(y)\}$

Resolving $T(g(c))$ with $\neg T(g(c))$ and $\neg T(y)$ using the unifier $\{g(c)/y\}$ yields

$\{ R(c),$
$\neg R(c) \vee M(g(c)),$
$\neg M(g(c))\}$

Finally, unifying $M(g(c))$ with $M(g(y))$ yields

$\{ R(c),$
$\neg R(c)\}$

This completes the proof, since we get the empty clause.

EXAMPLE 5

Problem: “If you are rich and good-looking you are happy. Fred is rich but not happy. Prove that Fred is not good-looking.”

The following is a formulation of the knowledge base together with the negation of the conclusion and the meanings: $R(x)$, $H(y)$, and $G(y)$ means “x is rich”, “y is happy”, and “x is good-looking”.

1. $(\forall x)(R(x) \wedge G(x) \rightarrow H(x))$

   $(\forall x)(\neg R(x) \vee \neg G(x) \vee H(x))$ (the same as the last line)

2. $R(Fred) \wedge \neg H(Fred)$

3. $G(Fred)$

The clauses are

1. $\neg R(x) \vee \neg G(x) \vee H(x))$
2. $R(Fred)$
3. $\neg H(Fred)$
4. $G(Fred)$

Unify {Fred/x} using clauses 1 and 2. We get

5. $\neg G(Fred) \vee H(Fred)$

Using 3 and 5 we get

6. $\neg G(Fred)$

Using 4 and 6 we get the empty clause. Therefore, the conclusion is correct.

EXAMPLE 6

Problem: "Some Canadians like every beer. No Canadian likes sweet beers. Prove that there is no sweet beer." $C(x)$, $B(y)$, and $L(x,y)$ means "x is a Canadian", "y is a beer", and "x likes y".

1. $(\exists x)(C(x) \wedge (\forall y)(B(y) \rightarrow L(x,y)))$

   $(\exists x)(\forall y)(C(x) \wedge (\neg B(y) \vee L(x,y)))$

   $(\forall y)(C(Fred) \wedge (\neg B(y) \vee L(Fred,y)))$

2. $(\forall x)(\forall y)(C(x) \rightarrow (S(y) \rightarrow \neg L(x,y))$

   $(\forall x)(\forall y)(\neg C(x) \vee \neg S(y) \vee \neg L(x,y)$

3. $\neg(\forall x)(B(x) \rightarrow \neg S(x))$

   $(\exists x)\neg(\neg B(x) \vee \neg S(x))$

   $(\exists x)(B(x) \wedge S(x))$

   $B(Labatt) \wedge S(Labatt)$

The clauses are:

1. $C(Fred)$
2. $\neg B(x) \vee L(Fred,y)$
3. $\neg C(x) \vee \neg S(y) \vee \neg L(x,y)$
4. $B(Labatt)$
5. $S(Labatt)$

1 and 3 gives:

6. $\neg S(y) \vee \neg L(Fred,y)$

2 and 4 gives:

7. $L(Fred, Labatt)$

6 and 7 gives:

8. $\neg S(Labatt)$

5 and 8 gives the empty clause. Therefore, we have proved the conclusion.

EXAMPLE 7: (CHANG/LEE P.49) A PROOF IN GROUP THEORY

Prove the following theorem: "If $x \cdot x = e$ for all x in group G, where $\cdot$ is a binary operator and e is the identity in G, then G is commutative". We must first formulate the axioms of a group definitions. G is a *group* if and only if

1) $x,y \in$ G implies that $x \cdot y \in$ G (closure property)
2) $x,y,z \in$ G implies that $x \cdot (y \cdot z) = (x \cdot y) \cdot z$ (associativity property)
3) $x \cdot e = e \cdot x = x$ for all $x \in$ G (identity property)
4) for every $x \in$ G there exists an element $x^{-1} \in$ G such that $x \cdot x^{-1} = x^{-1} \cdot x = e$ (inverse property).

Let $P(x,y,z)$ stand for $x \cdot y = z$ and $i(x)$ for x^{-1}. Then the four axioms can be formulated in first order logic as following:

1) $\forall xy \exists z\, P(x,y,z)$
2) $\forall xyzuvw\,(P(x,y,u) \wedge P(y,z,v) \wedge P(u,z,w) \rightarrow P(x,v,w))$
   $\wedge\, \forall xyzuvw\,(P(x,y,u) \wedge P(y,z,v) \wedge P(x,v,w) \rightarrow P(u,z,w))$
3) $\forall x\,(P(x,e,x) \wedge \forall x\, P(e,x,x))$
4) $\forall x\, P(x,i(x),e) \wedge \forall x\, P(i(x),x,e)$

The theorem can be formulated as

5) $\forall x\, P(x,x,e) \rightarrow \forall uvw\,(P(u,v,w) \rightarrow P(v,u,w))$

Adding the negation of the theorem to the four axioms and transforming them to a Skolem standard form, we get the set of clauses:

1) $P(x,y,f(x,y))$
2a) $\neg P(x,y,u) \vee \neg P(y,z,v) \vee \neg P(u,z,w) \vee P(x,v,w)$
2b) $\neg P(x,y,u) \vee \neg P(y,z,v) \vee \neg P(x,v,w) \vee P(u,z,w)$
3a) $P(x,e,x)$
3b) $P(e,x,x)$
4a) $P(x,i(x),e)$
4b) $P(i(x),x,e)$
5a) $P(x,x,e)$
5b) $P(a,b,c)$
5c) $\neg P(b,a,c)$

Adding the clauses which define the properties of the equality predicate (see next section), we can use resolution to prove the theorem.

EXAMPLE 8: (CHANG/LEE P.91)

Problem: “Students are citizens. Prove that students' votes are citizens' votes”. Using the definition $S(x)$, $C(x)$, and $V(x,y)$ which mean “x is a student”, “x is a citizen”, and “x is a vote of y” a first order formulation is:

$\forall y\,(S(y) \rightarrow C(y))$ (the premise)

$\forall x\,(\exists y\,(S(y) \wedge V(x,y)) \rightarrow \exists z\,(C(z) \wedge V(x,z)))$ (the conclusion)

Negating the conclusion gives:

$$\exists xy \forall z\,(S(y) \wedge V(x,y) \wedge (\neg C(z) \vee \neg V(x,z)))$$

The corresponding set of clauses is

$1)\ \neg S(y) \vee C(y)$

$2a)\ S(b)$

$2b)\ V(a,b)$

$2c)\ \neg C(z) \vee \neg V(a,z)$

From 1) and 2) we get

$3)\ C(b)$

From 2c) and 3) we get

$4)\ \neg V(a,b)$

And from 4) and 2b) we get the empty clause.

The rest of the paper is a somewhat unordered collection of notes about different logic fragments or non-classical logics.

THE LOGIC WITH THE EQUALITY RELATION

It is easy to add the very important predicate of equality to first order logic. Adding that predicate, the successor function, and the constant 0, one can formalize number theory within first order logic [Chang/Lee, p.163ff]. To add the property of equality, we need only add the following axioms to our knowledge base. Note that $x \neq y$ is the same as $\neg(x=y)$.

$$
\begin{aligned}
&1)\ \forall x\,(x = x)\\
&2)\ \forall xy\,((x \neq y) \vee (y = x))\\
&3)\ \forall xyz\,((x \neq y) \vee (y \neq z) \vee (x = z))\\
&4)\ \forall xyu\,((x \neq y) \vee (x \neq u) \vee (y = u))\\
&5)\ \forall xyu\,((x \neq y) \vee (u \neq x) \vee (u = y))\\
&6)\ \forall xyz\,((x \neq y) \vee (f(x,z) = f(y,z))\\
&7)\ \forall xyz\,((x \neq y) \vee (f(z,x) = f(z,y))\\
&8)\ \forall xy\,((x \neq y) \vee (i(x) = i(y))
\end{aligned}
\qquad (K)
$$

In the example 7 of the last section, we may formulate the four axioms of commutativity as following

$$
\begin{aligned}
&1)\ \forall xy \exists z\,(x \cdot y = z)\\
&2)\ \forall xyz\,((x \cdot (y \cdot z)) = ((x \cdot y) \cdot z))\\
&3)\ \forall x\,((x \cdot e = x) \wedge (e \cdot x = x))\\
&4)\ \forall x\,((x \cdot i(x) = e) \wedge (i(x) \cdot x = e))
\end{aligned}
\qquad (A)
$$

The theorem to prove is formulated as

$$5)\ \forall x\,((x \cdot x = e) \rightarrow \forall uv\,(u \cdot v = v \cdot u)) \qquad (t)$$

Applying resolution the set of clauses K≈A≈{¬t} produces the empty clause. Therefore, the theorem t is true.

THE LOGIC OF ARITHMETIC (CHANDRU, HOOKER 1988)

Presburger arithmetic (also called *the Additive Arithmetic*, see Williams 1987) is built up from natural numbers, integer variables, addition, the arithmetic relations ($<$, $\leq$, $\geq$, $>$, $=$), the quantifiers ($\forall$, $\exists$) and the usual propositional connectives. The formulas must be closed. Multiplication as syntactical sugar may be present (one may write $2x$ instead of $x + x$, for example). Viewed from the angle of first order logic, one can build Presburger arithmetic from the three functions "0", "successor", and "+", and the predicate "<" (see Duffy 1991, p 106).

The Presburger arithmetic is decidable (see Presburger 1929), but the most efficient known decision procedure has complexity of the order of $2^{2^{2^n}}$ (Cooper 1972, Oppen 1975).

The $\forall$-Fragment of Presburger arithmetic is more tractable, its worst bound is 2^n. Bledsoe (1975) developed a sup-inf-method which was improved by Shostack (1977).

If natural numbers are replaced by real numbers then we get *the Presburger real arithmetic*. The $\forall$-fragment is called *the Bledsoe real arithmetic* (also called *the Theory of Dense Linear Order,* see Williams 1987). The sub-inf-method can also be applied to the Bledsoe real arithmetic.

Linear Programming is useful for solving decision problems in the Presburger real arithmetic (Williams 1987).

A formula of the ∃-fragment can be translated into a set of disjoint linear models and if one of them is feasible then the formula is true.
Example: the formula

$$\neg\forall x\forall y\forall z((2x < y \vee x + y \geq 5 \vee x > 6) \wedge (y + z < 2 \vee y + z \geq 1)) \quad (1)$$

is translated into

$$\exists x\exists y\exists z((2x \geq y \wedge \neg(x + y = 5) \wedge x \leq 6) \vee (y + z \geq 2 \wedge y + z < 1)) \quad (2)$$

which is the same as

$$\exists x\exists y(2x \geq y \wedge \neg(x + y = 5) \wedge x \leq 6) \vee \exists y\exists z(y + z \geq 2 \wedge y + z < 1) \quad (3)$$

since

$$\neg(x + y = 5) \text{ is the same as } (x + y \leq 5 - \varepsilon) \vee (x + y \geq 5 + \varepsilon) \quad (4)$$

where ε is a small number, we need only check if one of the following systems of inequalities is feasible

$$\begin{array}{lll} 2x - y \geq 0 & 2x - y \geq 0 & \\ x + y \leq 5 - \varepsilon & x + y \geq 5 + \varepsilon & y + z \geq 2 \\ x \leq 6 & x \leq 6 & y + z \leq 1 - \varepsilon \end{array} \quad (5)$$

To check a formula of the ∀-fragment (Bledsoe real arithmetic), one may simply negate the formula and check again the LPs for feasibility (like in the ∃-fragment.

If the ∀ quantifiers precede the ∃ quantifiers in the prenex form (we called this the ∀∃ fragment), we can combine linear programming with variable elimination method (Fourier-Motzkin elimination, see Williams 1976, 1986, and 1987). An example (from Chandru al.) shows this procedure. Suppose we want to check the validity of

$$\forall x\exists y\exists z((x \geq 2y + 2z \wedge 2x \geq -3y - 6 \wedge x \geq y - z) \vee \neg(x \geq 0)) \quad (6)$$

which is equivalent to

$$\forall x(\exists y\exists z(x \geq 2y + 2z \wedge 2x \geq -3y - 6 \wedge x \geq y - z) \vee x < 0) \quad (7)$$

To eliminate the existentially quantified variables, we use Fourier-Motzkin elimination as following: To eliminate y we solve each inequality for y

$$\begin{array}{l} y \leq \frac{1}{2}x - z \\ -\frac{2}{3}x - 2 \leq y \\ y \leq x + z \end{array} \quad (8)$$

This system of inequalities is satisfied if we pair the left with the right expressions as following

$$-\tfrac{2}{3}x - 2 \le \tfrac{1}{2}x - z$$
$$-\tfrac{2}{3}x - 2 \le x + z \qquad (9)$$

Eliminating z in the same manner gives

$$x \ge -\tfrac{24}{17}$$

(10)

which reduces the whole expression (6) to the Bledsoe real arithmetic formula

$$\forall x(x \ge \tfrac{24}{17} \vee x \le -\varepsilon)$$

(11)

The $\exists\forall$-fragment can be negated to produce a $\forall\exists$ fragment. If the latter fragment is false then the original formula is valid.

For Presburger (integer) arithmetic the variables and constants are restricted to be integers. In this case we need a IP-solver instead of a LP-solver procedure to handle the $\forall$ and the $\exists$ fragment. For the $\forall\exists$ fragment Williams (1976) proposes a modification of the Fourier elimination procedure. If y is integer the expression (9) is no longer equivalent to (8) . Rather it is equivalent to

$$-4x - 12 \le 6y \le 3x - 6z$$
$$-2x - 6 \le 3y \le 3x + 3z$$

(12)

(12) can be expressed with the following disjunction

$$\vee_{i=0}^{5}(-4x - 12 + 1 \le 3x - 6z \wedge -4y - 12 + i \equiv 0(\mathrm{mod}\,6))$$

Note that any congruence $A \equiv a(\mathrm{mod}\,b)$ can be removed by adding the constraint $A = bw + a$, where w is a new integer variable (Chandru p.356).

PROBABILISTIC LOGIC (NILSSON 1986)

In logic all propositions are either true or false. In probabilistic logic, a probability is assigned to all proposition and we ask for the probability that another statement can be inferred. Example: Suppose the statements p and $p \varnothing q$ are true with probabilities p_1 and p_2, what is the probability of the proposition q? In propositional logic we can assign any atomic proposition a probability and then calculate the probabilities of every interpretation. With n propositions we get 2^n interpretations. The sum of the probabilities of all interpretations must be one. In our example we get

interpretations	probability
(T,T)	0.1
(T,F)	0.2
(F,T)	0.3
(F,F)	0.4

p is true in the first two interpretations, therefore its probability is 0.3, q has probability 0.7 and p∅q has probability of 0.8.

In general, we have n atomic propositions with probabilities p_i with i={1..2^n} of the i-th interpretation and m statements with probabilities of π_j with j={1..m}. Furthermore, let a_{ij} be 1 if statement i is true in interpretation j and 0 otherwise. Then we have

$\pi = Ap$

and pe' = 1

In the small example we have therefore

$$\begin{pmatrix} 0.6 \\ 0.8 \\ \pi_3 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{pmatrix}$$

$$p_1 + p_2 + p_3 + p_4 = 1$$

The probability values we can assign to π_3 are those which satisfy the linear system of equations. Since this system forms a polyhedron, all probabilities satisfy the system which are in the interior of the convex hull. Therefore, the probabilities π may take a range. To find them for a specific t, we just solve the two systems

max/min π_t

s.t. $\pi = Ap$, pe' = 1

The trouble with this approach is that the matrix A contains 2^n columns which becomes quickly impractical for larger problems. A solution to this problem is to generate only those columns of A that are likely to receive positive coefficients in the convex combination using a heuristic (Nilsson 1986). Another method is described in Hooker (1988), and Chandru 1988). A third possibility might be to choose at random some columns of A and to add columns by solving a 0-1 Knapsack problem (Gomory cutting stock problem).

MODAL LOGIC

Lewis 1917: He didn't like the theorems about the implications which he called material implication (denoted by $\rightarrow$).

$$(p \rightarrow q) \vee (q \rightarrow p)$$

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

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

Lewis wanted a 'stronger' implication (denoted $\Rightarrow$). He made a distinction between 'true' and 'necessarily true'

true : it rains today

necessarily true: if p is a prime, then p is divisible by 1 and p only

(philosopher still disputed about the difference!)
To make the distinction he invented the operator L ([])
Various interpretation are possible

1. Lp: p is necessarily true ¬L¬p (Mp): p is possible
2. Lp: I know that p is true Mp: I agree that p is possibly true
3. Lp: p is obligatory Mp: p is admissible (deontic!)
4. Lp: from now on p is true Mp: an instance in the future exist such that p (temporal logic).

Definition: $p \Rightarrow q$ is the same as $L(p \to q)$. We have five condition to define this operator:

1) ¬M¬ = L and ¬L¬ = M
2) we want: $(p \Rightarrow q) \to \neg M(p \wedge \neg q)$, the inverse is not universally accepted, although it simplifies the 'life'. Therefore we have: $(p \Rightarrow q) \leftrightarrow \neg M(p \wedge \neg q)$ which is the same as $(p \Rightarrow q) \leftrightarrow L(p \to q)$.
3) Neither Lp=p, nor Lp=¬p need to be theorems (otherwise L would be a simply unary null or negation operator!).
4) A theorem is not only true but necessarily true, hence $p \to Lp$ if p is a theorem.
   $Lp \wedge (p \Rightarrow q) \to Lq$
5) $Lp \wedge L(p \to q)) \to Lq$
   $L(p \to q) \to (Lp \to Lq)$
   is accepted as axiom (all three formulations are equivalent).

System K: Take the five conditions, the propositional calculus and the three inference rules (substitution, modus ponens, and necessitation (=condition 4))
K is not very useful, since $Lp \wedge L\neg p$ is satisfiable. But at least in K, one can prove $L(p \wedge q) \leftrightarrow Lp \wedge Lq$ and also $M(p \vee q) \leftrightarrow Mp \vee Mq$. The proof of the first is as following

1. $L(p \wedge q)$ given
2. $(p \wedge q) \to p$
3. $L((p \wedge q) \to p)$ condition 4
4. $L((p \wedge q) \to p) \to (L(p \wedge q) \to Lq)$ axiom and subst.
5. $L(p \wedge q) \to Lp$ from 3. and 4.
6. Lp from 1. and 5. by modus ponens

One may prove Lq at the same way.

System D: Add the axiom $Lp \to Mp$ to K. (if one believes p, then p is possible). The following formulations are equivalent:

$\neg Lp \vee Mp$

$\neg Lp \vee \neg L\neg p$

$\neg(Lp \wedge L\neg p)$

$\neg L(p \wedge \neg p)$

$\neg L \perp$

(The axiom 'means': I to not believe p and ¬p at the same time, or something false) (axiom of necessity)

System T: Add an axiom to D: $Lp \rightarrow p$ (if I know p then p is true). Equivalent formulations are

$\neg Lp \vee p$

$\neg L\neg q \vee \neg q$

$Mq \vee \neg q$

$q \rightarrow Mq$

(if p is true, then I agree that p is possible (axiom de possibility)).

In T $(p \rightarrow q) \vee (q \rightarrow p)$ is a theorem but $(p \Rightarrow q) \vee (q \Rightarrow p)$ is not, also $(p \wedge q) \rightarrow (p \rightarrow q)$ is a theorem but $(p \wedge q) \rightarrow (p \Rightarrow q)$ is not. This is exactly what Lewis wanted!

T (as well as D and K) is consistent (if F is a theorem then ¬F is not)

T is also weakly complete (any valid formula is provable).

$Lp \rightarrow LLp$ is not a theorem in T. Therefore we have no mean to reduce repeated modal operator (as in LMLL¬Mp). There are four possibilities to reduce such formulas:

1) Mp = LMp, 2) Lp = MLp, 3) Mp = MMp, and 4) Lp = LLp.

We have the following relations between the formulas (Figure 4). (A arc means: if the source is a axiom then the destination is a theorem). (therefore from 1 or 2 any follows).

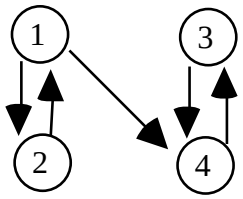


Figure 4

System S4: Add the axiom of 'introspection positive' to T: $Lp \rightarrow LLp$ (if I know p, then I know that I know p). That adds the 'knowledge about knowledge' to the system (whatever this means).

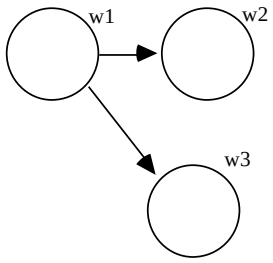
There are 14 modalities: none, L, M, LM, ML, LML, MLM and there negations.

System S5: Add the axiom of 'introspection negative' to the system T: $Mp \rightarrow LMp$ and $\neg Lp \rightarrow L\neg Lp$ (If I do not know p, then I know that I do not know p). (The axiom of S4 is a theorem in S5). There are only six modalities: none, L, M and there negations (all repeated formulas can be reduced!).

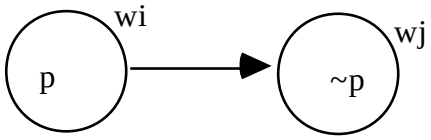
We have the following summary:

K	$(Lp \wedge L(p \rightarrow q)) \rightarrow Lq$ $L(p \rightarrow q) \rightarrow (Lp \rightarrow Lq)$	(both wff are equivalent) (I believe anything)
D	$Lp \rightarrow Mp$	(all I believe is possible)
T	$Lp \rightarrow p$	(all I believe is true (all modalities)
S4	$Lp \rightarrow LLp$	(I know what I know) (16 modalities)
S5	$\neg Lp \rightarrow \neg L\neg p$	(I know what I do not know) (6 modalities)
B	$p \rightarrow LMp$	

Semantics: (Hintikka, Kripke) 'The possible world' (feasible space?): We have a set of {w1,...wn} world and a accessibility relation wi --> wj. Each proposition can have a different value in the different world.
Lp is true in world wi if and only if p is true in all accessible world from wi.
Mp is true in world wi, if and only if there is at least one p which is true in a accessible world.

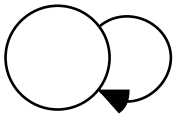


K: no constraint on the relation of accessibility

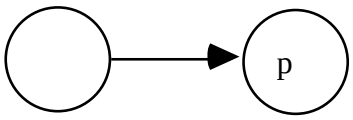


In wi $p \wedge L\neg p$ is true
In wj Lp is true and L¬p is true too

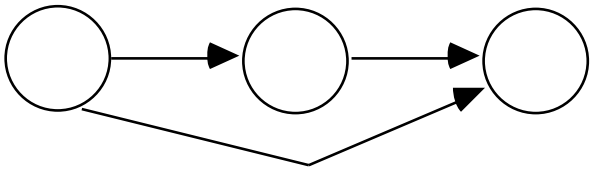
T: a method to make the axiom true is a accessibility reflexive



D: each world has access to at least one other world (relation serial)



S4: accessibility transitive:



S5: accessibility transitive, reflexive and symmetric (= equivalence relation)

All systems are sound and complete:

one can prove everything that is true

one cannot prove anything that is false

Prove procedure: The resolution is also applicable in the modal systems, but we must add some rules:

$M(\mathit{false}) \rightarrow \mathit{false}$

non resolution is possible with L(...) and within M(...)

each system has one or two rules to add.

Example 1 of proof by resolution in K: prove $L(p \wedge q) \rightarrow Lp$

the rule to add in K is: if one has Lu and Lv, we can add $M(u \wedge v)$

1. $L(p \wedge q)$ (given)
2. $\neg Lp$ (negation of the conclusion)
3. $M\neg p$ (from 2. by definition)
4. $M(\neg p \wedge p \wedge q)$ (from 1. and 3. by applying the rule in K)
5. $M(\mathit{false})$
6. false

Example 2 of proof by resolution in T: $Lp \rightarrow Mp$

the rule to add in T is: if we have Lp we may add p.

1. Lp (given)
2. ¬Mp (negation of the conclusion)
3. L¬p (by definition)
4. p (from 1, rule of T)
5. ¬p (from 3, rule of T)
6. false (from 4 and 5)

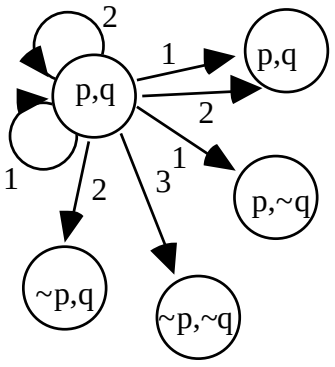
The different classes of the algorithms:

P:	polynomial deterministic
NP:	polynomial non-deterministic
PSPACE:	space polynomial deterministic
EXP:	all others

It is known that $P \subseteq NP \subseteq PSPACE \subseteq EXP$ and $P \neq EXP$.
The satisfiability of a formula

- in propositional logic is in NP
- in first-order logic in PSPACE
- propositional S5 is in NP
- K, D, T, S4 are all in PSPACE

Extensions of modal logic:
1. Extension on more than one process (agent). Modality K_i means "the process i knows that ...". Each process follows the same axioms (satisfiability is in PSPACE).



for K1: p is true
for K2: p is false, but K2 can at least find p possible

Other modalities are:
E : "Everyone knows..." (Ep is equivalent to $K_1 p \wedge \mathrm{K} \wedge K_n p$)
C: "Common sense" (Cp is equivalent to $Ep \wedge EEp \wedge EEEp \wedge \mathrm{K}$
I: "implicit knowledge" (what we would know if our heads would be put

together)
example: if $K_1 p, K_2(p \rightarrow q)$ and if K1 and K2 would communicate then we would know that q is true.

Other extension: modal, first order logic:
Example of a formula: $(\forall x)LP(x) \rightarrow (\exists x)MP(x)$
Formula of Barcan (1945): $(\forall x)LP(x) \rightarrow L(\forall x)P(x)$ is true in S5, otherwise it is true if all world contain the same domain (same number of objects), otherwise it is false.

To make system usable, we should have:

> If Di is the domain of wi and Dj is the domain of wj and wj is accessible from wi then $D_i \subseteq D_j$ (no object can be lost).

Logic of multiple values:
3 values: Lucasiewicz (1920) : true, false, and non-defined (unknown, contradictory)
4 values (Prior): true and we know it, true and we do not know it, false and we know it, false and we do not know it.

Motivations of a logic of multiple values:

1) to avoid the fatalism: How to interpret: "I will be in England tomorrow"
2) to avoid the paradoxes: "This sentence is false"
3) lack of denotation: "the king of France is bold"
4) counterfactuals: "if Colon would have been an English man, he would not have visited the US".
5) non-decidability (and physique of quanta)

NON-MONOTONIC LOGIC

If one defines the following knowledge base in classical logic:

1) All birds can fly
2) All kiwis are birds
3) Titi is a kiwi
4) one may conclude: Titi can fly!

That's ok. There is no probleme with this knowledge base. But now suppose we add the knowledge

5) No kiwi can fly.

The knowledge base becomes inconsistent! So we are now able to prove that Titi can fly and that Titi cannot fly! Classical logic is based on monotony: if knowledge is added to an already existing knowledge base, all conclusions which can been obtained by the former knowledge base remain valid if we add another formula. That means simply that if $A \varnothing p$ is valid then $A \Delta q \varnothing p$ is valid too. How to manage such a situation? A radical step would be to modify the knowledge base. For example, to modify axiom 1), in our example, to "All birds except kiwis can fly". But this solves not really our problem; maybe Titi is not a kiwi but an ostrich. Now the axiom must be modified to "All birds except kiwis and ostriches can fly". Each time an exception is found it would have been added to the general axiom; it is impossible to add all exceptions.

Another solution would be to erase the knowledge 2) that kiwis are birds. This step, in general, is to be avoided.

Other solutions are given by the approach of non-monotonic logic. It belongs to the fast rising new alternative logics, known also as *default reasoning* which can be classified into different approaches:

1) Logic based approaches
   - non-monotonic logic
   - default logic (common sense reasoning)
   - circumscription
   - LEL (least exception logic)
2) Measure basic approaches
   - Bayesian analysis
   - Dempter-Shafer
   - Fuzzy logic
   - confidence factors

It is impossible to give here a survey of them. But they are relevant too for mechanical theorem proving.

REFERENCES

CHANG C., LEE C., [1973], Symbolic Logic and Mechanical Theorem Proving, Academic Press, Inc., Boston.

DAVIS M.D., WEYUKER E. J., [1983], Computability, Complexity, and Languages, Fundamentals of Theoretical Comptuer Science, Academic Press, New York.

DAVIS M., PUTNAM H.A., [1960], A computing procedure for quantification theory, J. Assoc. Comput. Mach. 7 (1960), p. 201-216.

DUFFY D, [1991], Principles of Automated Theorem Proving, Wiley,, Chichester.

FITTING M., [1990], First-order Logic and Automated Theorem Proving, Springer, New York.

HÜRLIMANN T. [1993b], Programming Logic: a Survey, Institute of Informatics (formerly Institute for Automation and Operations Research), Working Paper No 216 , September 1993, Fribourg.

LEWIS H.R., PAPADIMITRIOU C.H., [1981], Elements of the Theory of Computation, Prentice-Hall.

MANNA Z. [1974], Mathematical Theory of Computation, McGraw-Hill Book Company, New York.

ROBINSON J.A., [1992], Logic and Logic Programming, Communications of the ACM (35), (March 1992), p.41-65.

ROBINSON J.A., [1965], A machine-oriented logic based on the resolution principle. JACM 12 (1965), p.23-41.

Presburger arithmetic:

BLEDSOE W.W., [1975], A new method for proving certain Presburger formulas, Advanves Papers, 4th International Joint Conference on Artificial Intelligence, Tibilisi, Georgia, USSR (1975), p.15-21.

CHANDRU V., HOOKER J.N., [1988], Logical Inference: a Mathematical Programming Perspective, in: Workshop "Mathematics and AI", Vol II, FAW Ulm, 19th-22nd December 1988, p.315-375.

COOPER D.C., [1972], Theorem proving in arithmetic without multiplication, in: Machine Intelligence 7, Meltzer B., Michie D., {eds.), American Elsevier, New York.

OPPEN D., [1975], A $2^{2^{2^n}}$ upper bound on the complexity of Presburger arithmetic, Journal of Computer and Systems Science (1975).

PRESBURGER M, [1929], Über die Vollständigkeit eines gewissen Systems der Arithmetik ganzer Zahlen in welchem die Addition als einzige Operation hervortritt, Sprawozdanie z I Kongresu Matematykow Krajow Slowcanskich Warszawa, Warsaw, Poland (1929), p.92-101.

SHOSTACK R.E., [1977], On the SUP-INF method for proving Presburger formulas, Journal of the ACM 24 (1977), p.529-543.

WILLIAMS H.P., [1987], Linear and Integer Programming Applied to the Propositional Calculus, Int. Journal Systems Research and Info. Science, Vol 2, p. 81-100.

WILLIAMS H.P., [1986], Fourier's Method of linear programming and its dual, American Mathematical Monthly, 93, p.681-695.

WILLIAMS H.P., [1976], Fourier-Motzkin elimination extention to integer programming problems, Journal of Combinatorial Theory, 21, p.118-123.

Probabilistic logic

NILSSON N.J., [1986], Probabilistic Logic, Artificial Intelligence, 28 (1986), p.71-87.

HOOKER J.N., [1988], A mathematical programming model for probabilistic logic, Working Paper 5-88-89, Graduate School of Industrial Administration, Carnegie Mellon Univ., Pittsburgh, PA.

BEN-ARI M., [1993], Mathematical Logic for Computer Science, Prentice Hall, New York.

BECKSTEIN C., [1988], Zur Logik der Logik-Programmierung, Springer, Berlin.