Overview of proposition and predicate logic

Jan Kuper
jankuper@cs.utwente.nl

October 5, 2005

Introduction

The subject of logic is to examine human reasoning and to formulate rules to ensure that such reasoning is correct. Modern logic does so in a formal mathematical way, hence names like “symbolic logic”, “formal logic”, “mathematical logic”. The logical approach includes the expression of human knowledge in a formal language (often called “knowledge representation language”), and second the application of reasoning rules to these formal representations. This formal character opens the possibility that the human reasoning process can be implemented on a computer. That does not necessarily mean that a logical language can model every aspect of human reasoning in a satisfactory way, but it does explain why formal logic is important for artificial intelligence, and for automated reasoning in general.

Below we describe two logical systems:

– in **proposition logic** we can express statements as a whole, and combinations of them. Intuitively, a statement is a sentence in which something is told about some reality, and which can be **true** or **false** about that reality. For example, if \( p \) is the statement "Albert is at home", and \( q \) means "the door is locked", then \( q \rightarrow \neg p \) says: "if the door is locked, then Albert is not at home".

– in **first order predicate logic** a statement has a specific inner structure, consisting of terms and predicates. Terms denote objects in some reality, and predicates express properties of, or relations between those objects. For example, the same example as before might be expressed as \( \text{Locked}(d) \rightarrow \neg \text{AtHome}(a) \). Here, \( \text{Locked} \) and \( \text{AtHome} \) are predicates, and \( d \) and \( a \) are terms, all with obvious meanings.

One further possibility on predicate logic is that we can speak about **some** and **all** objects. For example, \( \forall x. \text{AtHome}(x) \) means that every \( x \) is at home (leaving open for the moment whether \( x \) is a person or something else).

In the descriptions below we start with the formal language definition, both syntactically and semantically, and then give rules according to two approaches: natural deduction and semantic tableaux. The latter ones are treated at the end for proposition and predicate logic together. In addition there are other systems for reasoning rules like Hilbert style systems, sequent calculus, resolution, but we won’t discuss them here.

This text is very concise and only describes some basic definitions. For further questions about logical systems, like the strength or equivalence of several systems of rules, we refer the reader to the literature.
1 Proposition logic

A proposition is a statement of language that formulates something about an external world. A proposition can either be true or false. Thus questions, commands, promises, etc., are not propositions.

Formal language definition

The formal language of proposition logic is defined recursively:

- there is an infinite list of elementary propositions $p, q, r, \ldots$,
- if $\varphi$ is a proposition, then $\neg \varphi$ is a proposition too,
- if $\varphi, \psi$ are propositions, then $\varphi \lor \psi, \varphi \land \psi, \varphi \rightarrow \psi, \varphi \leftrightarrow \psi$ are propositions too,
- nothing else is a proposition.

The symbols $\neg, \lor, \land, \rightarrow, \leftrightarrow$ are (logical) connectives, called negation (for “not”), disjunction (“or”), conjunction (“and”), implication (“if · · · then”), equivalence (“if and only if”), respectively. Some examples of propositions are

$$p \lor q, \quad p \rightarrow (q \lor r), \quad (p \land q) \leftrightarrow (r \rightarrow (\neg q \lor p)).$$

In these examples brackets indicate in what order the proposition is constructed by the rules from the definition. In order to avoid too many brackets, connectives can be ordered according to their binding strength as follows (from strong to weak): $\neg, \land, \lor, \rightarrow, \leftrightarrow$. Thus, the final example above might also have been written as

$$p \land q \leftrightarrow r \rightarrow \neg q \lor p.$$ 

However, for reasons of clarity, the use of brackets may be helpful. Probably

$$(p \land q) \leftrightarrow (r \rightarrow \neg q \lor p)$$

expresses the essential structure of the same proposition more clearly.

Truth tables

The question whether a proposition is true or false depends on the truth or falsity of the elementary propositions and the way in which they are combined by the connectives, as expressed in the following truth tables (0 stands for false, 1 for true):

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\neg p$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$p$</th>
<th>$q$</th>
<th>$p \lor q$</th>
<th>$p \land q$</th>
<th>$p \rightarrow q$</th>
<th>$p \leftrightarrow q$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Remark. The language of proposition logic is a formal language, i.e., the propositions form an inductively defined set of syntactic expressions, and the truth tables give the semantics of these expressions. For predicate logic (see below) we will define this more formally by means of a “semantic value function”.

Natural deduction for proposition logic

Below $\varphi, \psi, \chi$ are formulas in the language of proposition logic, i.e., they are constructed from elementary propositions $p, q, r, \ldots$, and from the logical connectives $\neg, \land, \lor, \rightarrow$.

For every connective there are one or two introduction rules (on the left) and elimination rules (on the right). We will not present the rules for “$\leftrightarrow$”, since $\varphi \iff \psi$ can easily be considered as shorthand notion for $(\varphi \rightarrow \psi) \land (\psi \rightarrow \varphi)$.

The symbol $\bot$ is called falsum, and stands for a contradiction, i.e., a formula which is always false. For example, we may take $\bot$ as shorthand notation for any formula of the form $\varphi \land \neg \varphi$.

In the rules $(\lor E), (\rightarrow I), (\neg I), (\neg E)$ there are formulas between “[” and “]”. That means that these formulas are used as (temporary) assumptions, which are canceled at the moment that such a rule is applied.
The rules for negation could also have been formulated without using \( \perp \):

\[
\begin{array}{c}
\psi \\
\hline
\neg \psi \quad (\neg \mathbf{I}'')
\end{array}
\quad
\begin{array}{c}
\neg \varphi \\
\hline
\varphi \quad (\neg \mathbf{E}_{c}')
\end{array}
\]

Thus, the rule \( \neg \mathbf{I}''' \) says that if some proposition \( \psi \) and its negation \( \neg \psi \) can both be derived from the same assumption \( \varphi \), this assumption leads to a contradiction. Hence, \( \varphi \) cannot be true, and we may conclude \( \neg \varphi \). The rule \( \neg \mathbf{E}_{c}' \) can be explained in the same way.

However, note that first applying the \( \mathbf{\& I} \)-rule leads to

\[
\psi \land \neg \psi.
\]

Since \( \perp \) is considered as shorthand notation for a proposition of this form (see above), we may also choose the simpler original formulation with \( \perp \). For example, the rule \( \neg \mathbf{I} \) now directly says that if \( \varphi \) leads to a contradiction, we may conclude \( \neg \varphi \).

The above set of rules characterizes so-called classical logic (hence the index “c” in the \( \neg \mathbf{E}_{c} \)-rule). Based on a different notion of truth, intuitionistic logic replaces this rule by the following rule

\[
\begin{array}{c}
\perp \\
\hline
\psi \\
\hline
\neg \varphi \quad (\neg \mathbf{E}_{i})
\end{array}
\]

In classical logic formulas like \( \neg \neg \varphi \rightarrow \varphi \) and \( \varphi \lor \neg \varphi \) are provable, whereas in intuitionistic logic they are not. Intuitionistic logic and its background are outside the scope of this text.

**Example.** Proofs in natural deduction take the form of a tree. The tree below proves \( \varphi \land \psi \rightarrow \chi \) from the assumption \( \varphi \rightarrow \psi \rightarrow \chi \).

\[
\begin{array}{c}
\varphi \rightarrow \psi \rightarrow \chi \\
\hline
[\varphi \land \psi] \quad (\land \mathbf{E}_{1}) \\
\hline
\varphi \quad (\rightarrow \mathbf{E}) \\
\hline
[\varphi \land \psi] \quad (\land \mathbf{E}_{2}) \\
\hline
\psi \\
\hline
\chi \quad (\rightarrow \mathbf{E}) \\
\hline
\varphi \land \psi \\
\hline
\chi \quad (\rightarrow \mathbf{I})
\end{array}
\]

In this proof the formula \( \varphi \land \psi \) is a (temporary) assumption that is used by the \( \land \)-elimination rules (twice). This assumption is canceled by the \( \neg \mathbf{I} \)-rule. After this cancellation the only remaining assumption (axiom) in this proof is the formula \( \varphi \rightarrow \psi \rightarrow \chi \). When there are several assumptions which are canceled during a proof, it has to be indicated at which step in the proof these assumptions are canceled.
Semantic tableaux

In natural deduction a proof is “built up” from the premises towards the conclusion. With a semantic tableau we proceed the other way, i.e., we “break down” the formulas. The method of semantic tableaux is mechanic in nature, whereas the method of natural deduction requires some creativity and understanding.

A semantic tableau is a tree in which every node consists of a bullet with some propositions on its left-hand and right-hand side. Formulas on the left-hand side of the bullet are supposed to be true, on the right-hand side they are supposed to be false.

A tree in the semantic tableau method starts by stating that all the premises are true, but the conclusion that has to be proved, is not true. The aim then is to find in a systematic way a truth value of the elementary propositions, which makes this starting point possible. If no such truth values can be found, the starting point cannot be the case, and thus the conclusion must be true whenever the premises are true.

For every logical connective there is a left rule and a right rule, saying that if a compound formula is true/false, then a certain truth value follows for the constituents of the compound formula. Thus, a rule has to be read from top to bottom. Sometimes there are a few possibilities, indicated by a “split” of the rule.

Example. We take the same example as before, i.e., the tree below proves \( \varphi \land \psi \rightarrow \chi \) from the premise \( \varphi \rightarrow \psi \rightarrow \chi \). However, the tree now starts with the assumption that the conclusion does not follow from the premise. Thus the tree starts with the premise on the left hand side (the true side), and the conclusion on the right hand side (the false side). The proof then proceeds by breaking the formulas into their constituent parts according to the rules above. A branch
closes (indicated with two horizontal lines), when somewhere in that branch the
same formula occurs at the left-hand side as well as at the right-hand side. That
is, a branch closes when it represents an impossible situation. In the case of this
example all branches close, i.e., the tree as a whole closes (see below). Hence,
the starting assumption that the premise is true and the conclusion is false, is
not possible. In other words, whenever the premise is true, the conclusion must
also be true. Thus the premise implies the conclusion, which was to be proved.
2 Predicate Logic

Predicate logic assumes that the world consists of individual objects which may have certain properties and between which certain relations may hold (the general name for a property or a relation is *predicate*). Besides, there are operations which may be performed on these objects, the result of which is an object in the world again. For example, if the objects in the world are individual human beings, then there might be predicates like “male”, “female”, “older-than”, “married-to”. Note that the first two examples are about one person, the others about two. An example of an operation on human beings is “father-of”. Applying this operation to an individual delivers the father of that individual\(^1\).

Distinguishing separate objects in the world gives the possibility to quantify over these objects, i.e., to make statements about all or some objects. In higher order predicate logic it is also possible to speak about all (or some) sets of objects, such as all families, some school classes. We restrict ourselves to first order logic in which one can only quantify over the individual objects themselves.

Formal language definition

People usually reason about specific topics, so in order to formulate sentences about such a topic in a formal language, one first has to fix the alphabet of the language:

- a constant \((a, b, c, \cdots)\) is intended to denote a specific individual object,
- a function symbol \((f, g, h, \cdots)\) denotes an operation that may be performed on (sequences of) individual objects to yield another object,
- a predicate symbol \((P, Q, R, \cdots)\) denotes a property or relation that holds for (sequences of) individual objects.

Every function and predicate symbol has an *arity*, indicating how many arguments it requires.

The alphabet determines a language which is intended to express facts about a given world. In addition, every language of predicate logic contains *variables* which denote individual objects. The difference between a variable and a constant is that a constant denotes a fixed object, whereas the object denoted by a variable may vary (not within the same context or sentence, though).

The formal language of predicate logic consists of two parts: *terms* are intended to denote objects in the world, *propositions* are intended to express facts about these objects. Both terms and propositions are defined recursively:

**Terms:**

- Variables and constants are terms,
- if \(f\) is an \(n\)-ary function symbol, and \(t_1, \ldots, t_n\) are terms, then \(f(t_1, \ldots, t_n)\) is a term too,
- nothing else is a term.

\(^1\)Note that the word “operation” in this context is pretty technical.
Propositions:

- if \( P \) is an \( n \)-ary predicate symbol, and \( t_1, \ldots, t_n \) are terms, then the expression \( P(t_1, \ldots, t_n) \) is an (atomic) proposition,

- if \( \varphi \) is a proposition, then \( \neg \varphi \) is a proposition too,

- if \( \varphi, \psi \) are propositions, then \( \varphi \lor \psi, \varphi \land \psi, \varphi \rightarrow \psi, \varphi \leftrightarrow \psi \) are propositions too,

- if \( x \) is a variable, and \( \varphi \) is a proposition, then \( \forall x \varphi \) and \( \exists x \varphi \) are propositions too,

- nothing else is a proposition.

The symbols \( \forall \) and \( \exists \) are quantifiers, called the universal quantifier (for “for all”) and the existential quantifier (“for some”, i.e. “at least one”), respectively.

Some examples of propositions are

\[
(\forall x P(x)) \land (\exists y Q(y)), \quad \forall x P(x) \rightarrow \exists y Q(x, y, z), \quad \forall x P(x) \land \exists x Q(x).
\]

When a variable \( x \) is inside the “scope” of a quantifier with the same variable \( x \), then \( x \) is said the be bound by that quantifier. In the second example above \( x \) is bound by the quantifier \( \forall x \), \( y \) is bound by \( \exists y \), and \( z \) is not bound at all. The variable \( z \) is said to occur free in this proposition. Note that in the third example the \( x \) in \( P(x) \) is bound by \( \forall x \), whereas the \( x \) in \( Q(x) \) is bound by \( \exists x \). Outside the scope of a quantifier which binds a variable \( x \), this variable \( x \) is not meaningful.

With respect to the use of brackets it is agreed that the scope of quantifiers extends to the right as far as possible.

As a more concrete example, take normal arithmetic with the following alphabet (we restrict ourselves to just a few of the standard symbols): \( 0, 1, 2, \) etc., are constants, \( + \) and \( \ast \) are two-place function symbols, \( > \) and \( = \) are two-place predicate symbols. Using infix notation, the following expressions are propositions according to the definitions above:

\[
\forall x (x + 1 > x), \quad \forall x \forall y (x + y = y + x),
\]

\[
\forall x \forall y \forall z (x \ast (y + z) > (x \ast y) + (x \ast z)),
\]

\[
\forall x \forall y (x \ast y = 0 \rightarrow x = 0 \lor y = 0).
\]

These examples express true or false propositions about normal arithmetic.

The following examples are expressions which are not correctly formulated according to the above syntactical rules:

\[
\forall \exists y f(x = gy), \quad P(Q(a) \land R(b)).
\]

Substitution

The proposition

\[\forall x \forall y (x + y = y + x)\]
is a true formula of arithmetic. One of the rules of natural deduction for predicate logic says that then

\[ \forall y (2 + y = y + 2) \quad \text{and} \quad 2 + 3 = 3 + 2 \]

also are true propositions of arithmetic. These formulas are obtained from the first one by removing the leading universal quantifier, and then by substitution of 2 for \( x \) and 3 for \( y \). Substitution is a tricky operation, for example

\[ \forall x \exists y (y > x) \]

is a true proposition, but substituting \( y \) for \( x \) in

\[ \exists y (y > x) \]

yields

\[ \exists y (y > y) \]

which is a false formula. The problem here is that the substituted \( y \) is bound by the existential quantifier, whereas before the substitution \( x \) was not bound by that quantifier.

Substituting a term \( t \) for a variable \( x \) in an expression \( e \) is written as \( e[x:=t] \) and formally defined as follows (recursively, following the definition of terms and propositions):

- \( a[x:=t] \equiv a \),
- \( x[x:=t] \equiv t \),
- \( y[x:=t] \equiv y \),
- \( f(t_1,\ldots,t_n)[x:=t] \equiv f(t_1[x:=t],\ldots,t_n[x:=t]) \),
- \( P(t_1,\ldots,t_n)[x:=t] \equiv P(t_1[x:=t],\ldots,t_n[x:=t]) \),
- \( (\neg \varphi)[x:=t] \equiv \neg(\varphi[x:=t]) \), likewise for the other propositional connectives,
- \( (\forall \varphi)[x:=t] \equiv \forall x \varphi \),
- \( (\forall y \varphi)[x:=t] \equiv \forall y'(\varphi'[x:=t]) \), where \( \varphi' \equiv \varphi[y:=y'] \), \( y' \) not in \( t \) or \( \varphi \),
- ibid for the existential quantifier.

In this definition, \( \equiv \) stands for syntactical equality. It is assumed that \( a \) is a constant, and \( x, y \) are (different) variables.

Remark. It is amazing how complex the definition of something so simple as substitution is. It is only given for completeness’ sake – don’t try to learn it by heart. It formulates what you would expect in practical situations anyway, but when a computer has to perform a substitution correctly, it is absolutely necessary to formulate the definition explicitly.

The only thing that should be carefully watched is whether the binding pattern of a proposition remains the same after some substitution.
Semantics

The syntax of a language is concerned with formulating expressions in the language correctly, semantics deals with the meaning of the expressions. Since the formal syntactical definition considers expression as abstract objects, which have no meaning by themselves, semantics can only be given to expressions by a function which maps syntactical objects to objects in some real world. A world consists of

- a universe $\mathcal{U}$ of objects,
- a collection $\mathcal{P}$ of predicates on $\mathcal{U}$, i.e., subsets of (ordered $n$-tuples of) objects of $\mathcal{U}$,
- a collection $\mathcal{F}$ of functions on $\mathcal{U}$, i.e., subsets of ordered $(n+1)$-tuples of objects in $\mathcal{U}$. Notice that the $(n+1)$-st element of those tuples must be uniquely determined, given the first $n$ elements of the tuple.

An assignment function $\alpha$ is a function from the language to the world, and maps every individual variable to an object in $\mathcal{U}$.

An interpretation function $I$ also is a function from the language to the world, and maps every individual constant to an object in $\mathcal{U}$, every predicate symbol to a predicate in $\mathcal{P}$, and every function symbol to a function in $\mathcal{F}$.

A valuation function $V$ is a function which assigns objects in $\mathcal{U}$ to terms, and which assigns truth values to propositions.

Let an assignment function $\alpha$ and an interpretation function $I$ be given. Then the corresponding valuation function $V$ is defined for terms as:

- $V(x) = \alpha(x)$ for every individual variable $x$,
- $V(a) = I(a)$ for every individual constant $a$,
- $V(f(t_1, \ldots, t_n)) = d$ if $\langle V(t_1), \ldots, V(t_n), d \rangle$ is in $I(f)$.

For propositions the valuation function is defined as follows:

- $V(P(t_1, \ldots, t_n)) = 1$ if $\langle V(t_1), \ldots, V(t_n) \rangle$ is in $I(P)$,
- for combinations of propositions by means of the propositional connectives the valuation function is defined exactly as in proposition logic (by the truth tables),
- $V(\forall x. \varphi) = 1$ if for all $d$ in $\mathcal{U}$ we have that $V(\varphi) = 1$, where $\alpha$ is replaced by $\alpha[x\mapsto d]$,
- $V(\exists x. \varphi) = 1$ if there is some $d$ in $\mathcal{U}$ for which $V(\varphi) = 1$, where $\alpha$ is replaced by $\alpha[x\mapsto d]$.

The notation $\alpha[x\mapsto d]$ denotes a function which does the same as $\alpha$, except for the argument $x$. The result for $x$ is $d$.

If a formula $\varphi$ is true in a world $W$ for some given assignment function $\alpha$ and interpretation function $I$, we write

$\langle W, I \rangle \models_\alpha \varphi$. 
A formula $\varphi$ “follows from” a theory $\Sigma$ (i.e. a set of propositions) if for every $W$, $I$, $\alpha$ such that all formulas in $\Sigma$ are true, we also have that $\varphi$ is true. We write

$$\Sigma \models \varphi.$$ 

Some technical names for this form of “follows from” are entailment, semantic consequence.

Formulated somewhat sloppy, $\Sigma \models \varphi$ says that in any world where the theory $\Sigma$ holds, $\varphi$ must also hold.

**Remark.** The same remark as with substitution applies for the definition of semantics: for humans the meaning of a predicate logic expression is more or less clear by itself (provided a person has a sufficient amount of experience). But in the context of a computer, an expression becomes meaningless, and meaning has to be given explicitly by the mathematical function $V$.

**Natural deduction for predicate logic**

For the propositional connectives in predicate logic, the rules are the same as before. In the quantifier rules we write $\varphi(x)$ to indicate that the variable $x$ may occur free in the formula $\varphi$. Then $\varphi(a)$, $\varphi(t)$ are the results of substituting $a$, $t$ (respectively) for $x$ at all relevant positions in $\varphi$. Here we intend $a$ to be a name, i.e., a variable or a constant, and $t$ to be an arbitrary term.

For the rules with the quantifiers we need a little care. These extra precautions have to do with the following two points:

- **substitution:** it may occur that after substitution some unwanted bindings of variables to quantifiers arise. Take, for example, the proposition $\varphi(x) \equiv \exists y. y > x$, and consider $\varphi(y)$.

- **arbitrariness:** names have to be arbitrary, i.e., nothing may be assumed about those names. Technically speaking, such names may not occur free in any of the assumptions.

**Exercise.** Formulate these restrictions more carefully and precisely.

$$\begin{align*}
\frac{\varphi(t)}{\exists x. \varphi(x)} & \quad (\exists I) \\
\frac{\varphi(a)}{\forall x. \varphi(x)} & \quad (\forall I)
\end{align*}$$

$$\begin{align*}
\frac{[\varphi(a)]}{\exists x. \varphi(x) \psi} & \quad (\exists E) \\
\frac{\exists x. \varphi(x) \psi}{\psi} & \quad (\forall E)
\end{align*}$$

If a formula $\varphi$ can be proved from some theory $\Sigma$ by rules of natural deduction, we write

$$\Sigma \vdash \varphi.$$
There is the important completeness theorem:
\[ \Sigma \models \varphi \iff \Sigma \vdash \varphi. \]

From left to right this is called “soundness”, expressing that with the proof rules no false conclusions from true theories can be drawn. From right to left is “completeness” proper: it says that all semantic consequences of a theory can be proved by the proof rules.

**Semantic tableaux**

In natural deduction a proof is “built up” from the assumptions towards the conclusion. With a semantic tableau we proceed the other way, i.e., we “break down” the formulas. Hence, the method of semantic tableaux is mechanic in nature, whereas the method of natural deduction requires some creativity and understanding.

A semantic tableau is a tree in which every node consists of a bullet with some propositions on its left-hand and right-hand side. Formulas on the left-hand side of the bullet are supposed to be true, on the right-hand side they are supposed to be false. For every logical connective there is a left rule and a right rule, saying that if a compound formula is true/false, then a certain truth value follows for the constituents of the compound formula. Sometimes there are a few possibilities, indicated by a split of the rule.

With the quantifier rules some names are introduced: a name \( c \) may be any of the previously introduced names, name \( a \) is supposed to be new.
Example. The proof below assumes $\forall x. A(x) \rightarrow B(x)$ and $\forall x. B(x) \rightarrow C(x)$, and proves $\forall x. A(x) \rightarrow C(x)$. The technique of proving is the same as with proposition logic, i.e., let the assumptions be true, and suppose that the formula that has to be proved is false. Then all formulas are broken down according to the rules of semantic tableau, yielding various paths in a tree. If a path contains the same formula on the left (i.e., the true side) and on the right (the false side), then a contradiction arises and the path is closed (indicated by a double horizontal line). If all paths are closed, the proof is complete. If there remains an open path (i.e., a path which is not closed), there is no proof, and that open path provides a counter-example to the proposition to be “proven”. In the example below all paths are closed.

\[
\begin{align*}
\forall x. A(x) &\rightarrow B(x) , \forall x. B(x) \rightarrow C(x) \\
\forall x. A(x) &\rightarrow C(x) \\
A(a) & \rightarrow C(a) \\
B(a) & \rightarrow C(a) \\
A(a) & \rightarrow B(a) \\
\end{align*}
\]

Acknowledgments
Thanks to Peter Asveld for several corrections.

Some literature
