Partial Logic for Knowledge Representation and Deductive Reasoning in Incompletely Specified Domains

Anatoly Prihozhy and Liudmila Prihozhaya

Information Technologies and Robotics Department, State Politechnical Academy, Minsk, Belarus
aprihozhy@bspa.unibel.by

Abstract
This paper presents a new partial logic that generalizes the traditional proposition and first order predicate logics for incompletely specified domains. Three values are considered in the partial logic (false, true, and don’t care) instead of two values considered in the traditional logic. The operations, formulas, laws, and inference rules constitute a basis for knowledge representation and deductive reasoning when the world is not completely specified. The Robinson’s resolution principle is generalized for situation when a clause can take the don’t care value. Methods for the transition from partial deductive reasoning to inference in the first order predicate logic are proposed. The presented theoretical results are illustrated by examples. The proposed models and methods speed up the deductive reasoning process and decrease the number of clauses needed for knowledge representation.

Keywords: don’t care value, partial logic, incompletely specified domain, knowledge representation, deductive reasoning, generalization of Robinson’s resolution principle.

1. Introduction
The use of knowledge-based systems technology simplifies modeling a large class of complex situations involving symbolic reasoning and eases the task of stating complicated things about irregular domains. Several knowledge-based systems environments have been developed, both in research institutions and commercially. These environments use technologies such as pattern-action rules, frames, variants of procedural attachment and others [2]. Such environments provide not only the internal representational structures of their chosen paradigm, but also interface facilities that understand and can manipulate these structures.

Very often we deal with incompletely specified domains where assertions can be made for a part of situations. In this paper we propose a new partial logic that is based on three truth values: false, true, and don’t care. The logic is a generalization for the traditional proposition and first order predicate logics. The key results of Robinson’s work [8] hold in the partial logic as well. A novel aspect is that the partial logic allows for manipulation of the clause domains and simplification of the clauses during logic inference.

2. Methods of knowledge representation and reasoning: Background

In a knowledge-based system [2,8,10,11] the fundamental assumption is “knowledge is power”. The key idea is to separate knowledge of the task area as much as possible from the procedures that manipulate it.

A representation is a set of conventions for describing the world. The results of artificial intelligence research have been used to establish convenient ways of describing parts of the world. The current representation methods are not the final word. However, they are well enough developed that they can be used for problem solving in interesting domains. By separating a knowledge base from the inference procedures that work with the knowledge, a lot of systems were built that are understandable and extendable. The basic requirements on a knowledge representation scheme are extendibility, simplicity, and explicitness. To achieve these goals, three types of representation framework have been used: rule-based, frame-based, and logic-based systems [2-8].

Instead of viewing computation as a pre-specified sequence of operations, production systems view computation as the process of applying transformation rules in a sequence determined by the
data. A classical production system has three major components: a global data base, rule base, and rule interpreter.

One approach to representing knowledge that allows rich linkages between facts is a generalization of semantic nets known as frames. A frame is an encoding of knowledge about an object, including not only properties (slots) and values, but points to other frames and attached procedures for computing values.

A logic-based representation scheme is one in which knowledge about the world is represented as assertions in logic, usually first order predicate logic or a variant of it. This mode of representation is normally coupled with an inference procedure based on theorem proving. The rigor of logic is an advantage in specifying precisely what is known and knowing how the knowledge will be used. Besides the proposition and first order predicate logics, the multiple-valued [9], temporal [3], and other logics constitute very important tools for solving various problems of computer science and artificial intelligence. A disadvantage is difficulty in dealing with the imprecision and uncertainty of plausible reasoning.

For many years, artificial intelligence research has focused on heuristic reasoning [10-11]. Heuristics are an essential key to intelligent problem solving because computationally feasible, mathematically precise methods are known for only a relatively few classes of problems.

Incompletely specified functions that take values of the set \{false, true, don't care\} are a very useful mechanism for modeling incompletely specified domains, in particular, the process of logic synthesis of digital systems [1]. The application possibilities of the functions are restricted by the fact that their arguments can take only the truth values true and false.

3. Partial proposition logic

The proposition variables and truth values are the primitive elements that are used for definition of more complex concepts in the partial proposition logic.

3.1 Partial values, variables and functions

The traditional total logic considers two values: true (1) and false (0). The partial logic [5-8] considers three values: true (1), false (0), and don't care (dc or -). The don't care value can be replaced with true or false arbitrarily. A total function \( f(x_1,\ldots,x_n) \) is a mapping \( f: B^n \rightarrow B \) where \( B=\{0,1\} \). A partial function \( g(y_1,\ldots,y_m) \) is a mapping \( g: M^m \rightarrow M \) where \( M=\{0,1,\ldots\} \). An incompletely specified function \( h(x_1,\ldots,x_n) \) is a mapping \( h: B^n \rightarrow M \). A variable that takes values from the set \( B \) will be called a total variable, and a variable that takes values from the set \( M \) will be called a partial variable.

A partial function \( g(y_1,\ldots,y_m) \) is valid if for all vectors \( a',a'' \in M^m \) such that \( g(a')=1 \) and \( g(a'')=0 \) the vectors \( a' \) and \( a'' \) are orthogonal. The vectors \( a'=(a'_1,\ldots,a'_m) \) and \( a''=(a''_1,\ldots,a''_m) \) are orthogonal if an integer \( j \) exists such that \( a'_j=0 \) and \( a''_j=1 \), or \( a'_j=1 \) and \( a''_j=0 \).

A Value-Domain Representation (VDR) is the following encoding of a partial variable \( y_i \) with a pair \((v_i|d_i)\) of total variables [4-7]:

\[
y_i = \begin{cases} 
0 & \text{if } v_i=0 \text{ and } d_i=0 \\
1 & \text{if } v_i=1 \text{ and } d_i=1 \\
dc & \text{if } v_i \in \{0,1\} \text{ and } d_i=0.
\end{cases}
\] (3.1)

The variable \( v_i \) is called a value variable and the variable \( d_i \) is called a domain variable. Due to VDR, a partial function \( z=g(y) \) of \( m \) three-valued arguments is represented as an incompletely specified function \( (v|d)=g'((v_1|d_1),\ldots,(v_n|d_n)) \) of \( 2m \) two-valued arguments. In this case the Boolean space is broken down into three parts as shown in Fig.3.1:

- on-set is the part where the function takes value 1, it is described by the expression \( f_{on}=(v\&d)^m \)
• off-set is the part where the function takes value 0, it is described by the expression $f^{\text{off}} = (\neg v \& d)^{on}$.

• don’t care set where the function takes don’t care, it is described by the expression $f^{\text{dc}} = (\neg d)^{on}$.

Logic assertions that can be formulated concerning the third part may take the value true or the value false arbitrarily.

![Figure 3.1. Structure of Boolean space](image)

### 3.2 Partial logic operations

Monadic and dyadic partial operations are basic ones in the partial logic. They allow for representation of all the partial functions through using a composition operation. There exist 27 monadic partial operations. Five of them are defined in Table 3.1: constant 0, constant 1, constant $dc$, identity, and negation. The third to fifth columns of the table describe the operation values for various values of the three-valued argument $y$. The sixth column indicates an operator that is used for a corresponding operation, and the seventh column presents a Value-Domain Representation of the operation, where $v_1$ and $d_1$ are Boolean variables that encode the partial variable $y$.

Exactly 19683 dyadic partial operations are possible. Ten of them that generalize the traditional dyadic logical operations are defined in Table 3.2. For the partial operations we use underlined operators that are similar to the operators used for the traditional Boolean operations. The dyadic operations are as follows: conjunction, disjunction, Sheffer stroke, Pierce arrow, implication, implication negation, consequence, consequence negation, exclusive OR, and equivalence. The third to eleventh columns of the table present the operation values for various values of partial variables $y_1$ and $y_2$. The thirteenth column presents a Value-Domain Representation of each dyadic partial operation, where $v_1$, $d_1$, $v_2$, and $d_2$ are Boolean variables that encode the partial variables $y_1$ and $y_2$.

In the pair $(v|d)$ function $d$ is fixed. The function $v$ can be replaced with another total function $v_i$ such that $(v_i|d) = (v|d)$ or $v_i \& d = v \& d$. In other words, the VDRs $(v|d)$ and $(v|d)$ represent the same incompletely specified function. If $V$ is the set of functions $v_i$ then for each $v_i \in V$ the inequality as follows

$$
(v \& d)^{on} \subseteq v_i^{on} \subseteq (v + \neg d)^{on}
$$

holds. A minimisation operation $\text{min}(v|d)$ is a mapping $\text{min}: F \times F \rightarrow F$ where $F$ is the set of total functions $f: B^n \rightarrow B$. The operation selects one function from the set $V$. Different definitions for $\text{min}(v|d)$ are possible [7]. They depend on which representation forms for $v$ and $d$ are used.

<table>
<thead>
<tr>
<th>Monadic partial logical operations</th>
<th>Partial variable $y$</th>
<th>Operator</th>
<th>Representation in VDR</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Operation name</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>Constant 0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>Constant 1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Constant –</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
3.3 Expressions

A formula that is interpreted using a two-valued truth function will be called a total formula. A formula that is interpreted using a truth function taking a value from the set $M$ will be called a partial formula. As we allow for using only Boolean primary variables, a partial formula has to contain at least one pair $(F|G)$ of total formulas $F$ and $G$. The partial logical operations are used for construction of partial formulas. If the total logical operations are used as well, then the formula will be called a mixed total-partial formula. The formula as follows

$$(x1|x2) & ((x2|x3) + (x3|x2) \rightarrow (x4|x1)) \quad (3.3)$$

is a partial formula, and the following formula:

$$(x1\&x2|x2 \rightarrow x3) @ (x2|x1+x3+x4) \pm (\sim x3+\sim x4|x1) \quad (3.4)$$

is a mixed total-partial formula. The expressions of the considered type represent incompletely specified functions through partial operations.

Example 3.1. The incompletely specified function presented in Table 3.3 is represented by the following partial expression: $f=(\sim x1+\sim x2+\sim x3|x1\&x2+\sim x3\&(x1+x2))$.

3.4 Partial logic laws

We formulate a partial logic law as

$$Lp = Rp, \quad (3.5)$$
where $L_p$ and $R_p$ are the left and right parts respectively in the law. The parts $L_p$ and $R_p$ are partial formulas. The values of the formulas have to belong the set $M$ and have to be the same at any values of proposition variables occurring in the formulas. In this section we consider three types of laws:

- the laws that maintain the transformation of an arbitrary partial proposition logic formula to a pair of total formulas
- the laws that are generalizations of appropriate laws in the traditional logic
- the novel laws of partial logic that maintain a transformation of a partial formula.

### 3.4.1 Transforming partial operations to pairs of total formulas

The monadic partial logical operations are transformed to pairs of total formulas through the laws as follows:

$$c_0(v|d) = (0|1)$$  \hspace{1cm} (3.6)
$$c_1(v|d) = (1|1)$$  \hspace{1cm} (3.7)
$$c_-(v|d) = (v|0)$$  \hspace{1cm} (3.8)
$$c_+(v|d) = (v|d)$$  \hspace{1cm} (3.9)
$$c_\sim(v|d) = (\sim v|0)$$  \hspace{1cm} (3.10)

The following laws constitute a basis for transformation of the dyadic partial operations to pairs of total formulas:

$$c_1(v_1|d_1)(v_2|d_2) = (v_1 v_2|d_1 d_2),$$  \hspace{1cm} (3.11)
$$c_1(v_1|d_1) + (v_2|d_2) = (v_1 + v_2|d_1 + d_2),$$  \hspace{1cm} (3.12)
$$c_1(v_1|d_1) / (v_2|d_2) = (v_1 / v_2|d_1 d_2),$$  \hspace{1cm} (3.13)
$$c_1(v_1|d_1) \downarrow (v_2|d_2) = (v_1 \downarrow v_2|d_1 d_2),$$  \hspace{1cm} (3.14)
$$c_1(v_1|d_1) \rightarrow (v_2|d_2) = (v_1 \rightarrow v_2|d_1 d_2),$$  \hspace{1cm} (3.15)
$$c_1(v_1|d_1) \leftarrow (v_2|d_2) = (v_1 \leftarrow v_2|d_1 d_2),$$  \hspace{1cm} (3.16)
$$c_1(v_1|d_1) \nearrow (v_2|d_2) = (v_1 \nearrow v_2|d_1 d_2),$$  \hspace{1cm} (3.17)
$$c_1(v_1|d_1) \searrow (v_2|d_2) = (v_1 \searrow v_2|d_1 d_2),$$  \hspace{1cm} (3.18)
$$c_1(v_1|d_1) \oplus (v_2|d_2) = (v_1 \oplus v_2|d_1 d_2),$$  \hspace{1cm} (3.19)
$$c_1(v_1|d_1) \equiv (v_2|d_2) = (v_1 \equiv v_2|d_1 d_2).$$  \hspace{1cm} (3.20)

Proof of equivalence (3.11) Table 3.4

| $v_1$ | $d_1$ | $v_2$ | $d_2$ | $(v_1|d_1)$ | $(v_2|d_2)$ | $v_1 \& v_2$ | $d_1 d_2 + \sim v_1 d_1 + \sim v_2 d_2$ | $(v_1|d_1) \& (v_2|d_2)$ | $(v_1\&v_2|d_1\&d_2)$ | $(v_1\&v_2|d_1\&d_2)$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | - | - | 0 | 0 | - | - | - |
| 0 | 0 | 0 | 1 | - | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | - | - | 0 | 0 | - | - | - |
| 0 | 0 | 1 | 1 | - | 1 | 0 | 0 | - | - | - |
| 0 | 1 | 0 | 0 | - | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | - | 0 | 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | - | - | 0 | 0 | - | - | - |
| 1 | 0 | 0 | 1 | - | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | - | 1 | 0 | - | - | - | - |
| 1 | 0 | 1 | 1 | - | 1 | 1 | 0 | - | - | - |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | - | - | - |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

A proof of equivalence (3.11) is presented in Table 3.4 where two last columns are identical.

The values of pairs $(v_1|d_1)$, $(v_2|d_2)$, and $(v_1\&v_2|d_1\&d_2)$ are computed through $(3.1)$,
and the values of expression \((v_1|d_1) \& (v_2|d_2)\) are computed through using row 1 of Table 3.2. It is easy to see, the proving other equivalences is performed in the similar way.

The domain parts of the pairs appearing in the right parts of (3.11)-(3.20) are represented as a sum of products. In the following versions of the laws the domain parts are represented as a product of sums:

\[
(v_1|d_1) \& (v_2|d_2) = (v_1 \& v_2|(d_1+d_1) \& (\neg v_2+d_1) \& (\neg v_1+d_2))
\] (3.21)

\[
(v_1|d_1) + (v_2|d_2) = (v_1+v_2|(d_1+d_1) \& (v_2+d_1) \& (v_1+d_2))
\] (3.22)

\[
(v_1|d_1) / (v_2|d_2) = (v_1 / v_2|(d_1+d_1) \& (\neg v_2+d_1) \& (\neg v_1+d_2))
\] (3.23)

\[
(v_1|d_1) \downarrow (v_2|d_2) = (v_1 \downarrow v_2|(d_1+d_1) \& (v_2+d_1) \& (v_1+d_2))
\] (3.24)

\[
(v_1|d_1) \rightarrow (v_2|d_2) = (v_1 \rightarrow v_2|(d_1+d_1) \& (v_2+d_1) \& (\neg v_1+d_2))
\] (3.25)

\[
(v_1|d_1) \leftarrow (v_2|d_2) = (v_1 \leftarrow v_2|(d_1+d_1) \& (\neg v_2+d_1) \& (v_1+d_2))
\] (3.26)

\[
(v_1|d_1) \sim> (v_2|d_2) = (v_1 \sim> v_2|(d_1+d_1) \& (v_2+d_1) \& (\neg v_1+d_2))
\] (3.27)

\[
(v_1|d_1) <\sim (v_2|d_2) = (v_1 <\sim v_2|(d_1+d_1) \& (\neg v_2+d_1) \& (v_1+d_2)).
\] (3.28)

If \(d_1=d_2=1\) then the partial operations become completely specified. In particular, the partial conjunction \((v_1|1) \& (v_2|1)\) is transformed to the total traditional conjunction \(v_1 \& v_2\) in the following way:

\[
(v_1|1) \& (v_2|1) = (v_1 \& v_2|1+\neg v_1 \& 1+\neg v_2 \& 1) = (v_1 \& v_2|1) = v_1 \& v_2
\]

3.4.2 Generalization of traditional logic laws

The key laws of the traditional two-valued logic are generalized in the partial logic as:

\[
\neg\neg(v|d) = (v|d)
\] (3.29)

\[
(v_1|d_1) \& (v_2|d_2) = (v_2|d_2) \& (v_1|d_1)
\] (3.30)

\[
(v_1|d_1) \& ((v_2|d_2) \& (v_3|d_3)) = ((v_1|d_1) \& (v_2|d_2)) \& (v_3|d_3)
\] (3.31)

\[
(v_1|d_1) \& (v_2|d_2) = (v_2|d_2) \& (v_1|d_1)
\] (3.32)

\[
(v_1|d_1) \& (v_2|d_2) = (v_1|d_1) \& (v_2|d_2)
\] (3.33)

\[
(v_1|d_1) \& (v_2|d_2) = (v_1|d_1) \& (v_2|d_2)
\] (3.34)

\[
(v_1|d_1) \& (v_2|d_2) = (v_1|d_1) \& (v_2|d_2)
\] (3.35)

\[
(v_1|d_1) \& (v_2|d_2) = (0|d)
\] (3.36)

\[
(v_1|d_1) \& (v_2|d_2) = (1|d)
\] (3.37)

\[
(v_1|d_1) \& (v_2|d_2) = (v_1|d_1)
\] (3.38)

\[
(v_1|d_1) \& (v_2|d_2) = (v_1|d_1)
\] (3.39)

\[
(v_1|d_1) \& (v_2|d_2) = (v_1|d_1)
\] (3.40)

\[
(v_1|d_1) \& (v_2|d_2) = (v_1|d_1)
\] (3.41)

\[
(v_1|d_1) \& (v_2|d_2) = (v_1|d_1)
\] (3.42)

\[
(0|d) \& (v_1|d_1) = (0|d)
\] (3.43)

\[
(0|d) \& (v_1|d_1) = (l|d)
\] (3.44)

\[
(0|d) \& (v_1|d_1) = (0|d)
\] (3.45)

\[
(0|d) \& (0|1) = (0|1)
\] (3.46)

\[
(0|d) \& (l|1) = (l|1)
\] (3.47)

\[
(l|d) \& (0|1) = (0|1)
\] (3.48)

\[
(l|d) \& (v_1|d_1) = (v_1|d_1)
\] (3.49)

\[
(l|d) \& (l|1) = (l|1)
\] (3.50)

\[
(v_1|d_1) \& (v_2|d_2) = (v_1|d_1) \& (v_2|d_2)
\] (3.51)

\[
(v_1|d_1) \& (v_2|d_2) = (v_2|d_2) \& (v_1|d_1)
\] (3.52)

Equivalence (3.29) is the "double negation" law. Equivalences (3.30) and (3.32) are the "commutative" laws for partial conjunction and partial disjunction respectively. Equivalences (3.31) and (3.33) are the "associative", and equivalences (3.34) and (3.35) are the "distributive" laws for partial conjunction and disjunction. Equivalences (3.36) and (3.37) are the "de Morgan's" laws. The
3.4.3 Novel laws in partial implication

The partial logic has own novel laws that constitute a mechanism for manipulation of value and domain parts of pairs representing incompletely specified functions:

\[
\begin{align*}
(v_1|d) & \& (v_2|d) = (v_1\&v_2|d) \\
(v_1|d) & \lor (v_2|d) = (v_1\lor v_2|d) \\
(v|d_1) & \& (v|d_2) = (v|d_1\&d_2 + v\&(d_1+d_2)) \\
(v|d_1) & \lor (v|d_2) = (v|d_1\lor d_2 + v\&(d_1+d_2)) \\
(v|v) & = (1|v) \\
(\neg v|v) & = (0|v)
\end{align*}
\]

Equivalences (3.53) and (3.54) prove that partial logical operations do not modify the domain part if the part is the same in the both operands. Equivalences (3.55) and (3.56) show how the domain part is determined when the partial conjunction is applied to two source pairs with the same value part. Laws (3.57) to (3.64) allows for simplification of the value part in special cases. Equivalences (3.65) to (3.70) are expansions of an incompletely specified function on partial conjunction and partial disjunction. Laws (3.71) to (3.74) constitute a mechanism for transformation of the total function appearing in the value part.

4. Partial first order predicate logic

The syntax and semantics of the partial first order predicate logic are described in this section. We also present a method for transforming a formula in the logic to a form that can be efficiently used for partial logic inference.

4.1 Syntax

The alphabet of our language consists of:

1) delimiters ’,’ ’,’ ’’,
2) variables of a set \(V\),
3) total logical operators \(\neg, \&, \lor, \rightarrow, \oplus\), and others,
4) partial logical operators \(\neg, \&, \lor, \rightarrow, \oplus\), and others,
5) functional symbols $f_1^{n_1},...,f_i^{n_i},...$ of degree $n_i \geq 0$; a functional symbol $f^0$ of degree $n=0$ is an individual constant,
6) predicate symbols $P_1^{n_1},...,P_i^{n_i},...$ of degree $n_i \geq 0$; a predicate symbol $P^0$ of degree $n=0$ is a proposition variable,
7) total universal quantifier $\forall$, total existential quantifier $\exists$, partial universal quantifier $\forall$, and partial existential quantifier $\exists$.

The alphabet symbols are used for building expressions.

Terms:
1) an individual constant $f^0$ is a term,
2) a variable $v \in V$ is a term,
3) a functional symbol $f_i^{n_i}(t_1,...,t_n)$ followed by $n$ terms is a term.

Atomic formulas:
1) a proposition variable $P^0$ is an atomic formula,
2) a predicate symbol $P_i^{n_i}(t_1,...,t_n)$ followed by $n$ terms is an atomic formula.

Literals:
1) an atomic formula is a literal,
2) if $A$ is an atomic formula then $\neg A$ is a literal.
A literal containing no variables is called a ground literal. Two literals $A$ and $\neg A$ constitute a complementary pair.

Total formulas:
1) an atomic formula is a total formula,
2) if $F$ is a total formula then $\neg F$ is a total formula,
3) if $F$ and $G$ are total formulas then $(F \& G)$, $(F + G)$, $(F \rightarrow G)$, $(F \oplus G)$, ... are total formulas,
4) if $F$ is a total formula and $x$ is a variable, then $(\forall x)F$ and $(\exists x)F$ are total formulas.
Sometimes we will omit the parentheses, taking into account the priority of operations.

Partial formulas:
1) a pair $(F|G)$ of total formulas is a partial formula,
2) if $R$ is a partial formula then $\neg R$ is a partial formula,
3) if $R$ and $Q$ are partial formulas then $(R \& Q)$, $(R + Q)$, $(R \rightarrow Q)$, $(R \oplus Q)$, ... are partial formulas,
4) if $R$ is a partial formula and $x$ is a variable, then $(\forall x)R$ and $(\exists x)R$ are partial formulas.

4.2 Semantics

The semantics for our language is defined as follows.
With each total logical operator we associate a truth function $B^n \rightarrow B$ where $n$ is the arity of operator. The truth functions for operators $\neg$, $\&$, $\rightarrow$, and $\oplus$ are well known.
The incompletely specified function $B \times B \rightarrow M$ defined in Table 4.1 is put into accordance with the pair $(F|G)$. With each partial logical operator we associate a partial function $M^n \rightarrow M$ from Table 3.1 and Table 3.2.

| The truth function for pair $(F|G)$ Table 4.1 |
|----------|----------|----------|
| $F$  | $G$  | $(F|G)$ |
| 0   | 0   | -       |
| 0   | 1   | 0       |
| 1   | 0   | -       |
| 1   | 1   | 1       |

| The functions for $(\forall x)R$ and $(\exists x)R$ Table 4.2 |
|----------|----------|----------|
| The set of values of $R$ | $(\forall x)R$ | $(\exists x)R$ |
| {0}   | 0       | 0       |
The semantics of the universal and existential quantifiers \((\forall x)R\) and \((\exists x)R\) is represented by the functions described in Table 4.2. The functions are mappings \(M^+ \rightarrow M\), where \(M^+\) is the set of all subsets of set \(M\) excluding the empty set. The ordinary universal and existential quantifiers \((\forall x)R\) and \((\exists x)R\) are defined in the traditional way.

Let a set \(D\) be the domain. We assign a function \(D^n \rightarrow D\) to each \(n\)-place functional symbol \(f^n_i\), and assign a function \(D^n \rightarrow B\) to each \(n\)-place predicate symbol \(P^n_i\). Each variable of \(V\) is mapped to an element of \(D\).

An evaluation function \(v_I(R)\) computes the value of a partial formula \(R\) for interpretation \(I\). An interpretation \(I\) satisfies a partial formula \(R\) iff \(v_I(R) \in \{1, -\}\). The formula \(R\) is called unsatisfiable iff there is no interpretation that satisfies \(R\).

### 4.3 Transforming a partial formula

Now we describe a method of transforming any partial formula to the following special form:

\((F|G)\). \hspace{1cm} (4.1)

The parts \(F\) and \(G\) are total formulas without any quantifiers. Each total formula is a conjunction of sentences and each sentence is a disjunction of literals. The transformation procedure includes a number of steps:

- eliminating the total logical operators except negation, conjunction, and disjunction
- introducing a unique variable for each quantifier
- reducing the scope of negation operations
- eliminating the existential quantifiers
- moving the universal quantifiers to the formula prefix
- removing the formula prefix
- eliminating all the partial logical operators
- transforming the partial formula matrix to a pair of sets of total clauses.

**Example 4.1.** Each step of the transformation procedure will be illustrated through using the following example partial formula:

\[(\forall x)((\forall y)(\neg Q(c,x) \rightarrow R(a,y) | Q(y,b)) \& (\exists z)(\neg(\exists y)(R(y,z)+P(x) | P(b))),\] \hspace{1cm} (4.2)

where \(x, y,\) and \(z\) are variables, \(a, d,\) and \(c\) are individual constants, and \(P, Q,\) and \(R\) are predicates.

**Step 1.** Eliminating the total logical operators except \(\neg, \&\), and \(+\). The known equivalences are used for this purpose. Performing the step for formula (4.1) we eliminate implication in the value part of first pair and obtain the formula as follows:

\[(\forall x)((\forall y)(Q(c,x)+R(a,y) | Q(y,b)) \& (\exists z)(\neg(\exists y)(P(x) | P(b))),\] \hspace{1cm} (4.3)

**Step 2.** Introducing a unique variable for each quantifier. Formula (4.3) uses three variables: \(x, y,\) and \(z\). Each of them is under a quantifier. The variable \(y\) is used twice. Therefore, we rename the variable at the second existential quantifier and have the following formula:

\[(\forall x)((\forall y)(Q(c,x)+R(a,y) | Q(y,b)) \& (\exists z)(\neg(\exists y)(P(x) | P(b))),\] \hspace{1cm} (4.4)

**Step 3.** Reducing the scope of the total \(\neg\) and partial \(\neg\) negation operators. The double negation and de Morgan’s laws for conjunction and disjunction are used. Besides, one has to use the equivalences as follows:
\[ \neg(F \mid G) = (\neg F \mid G) \]  
(4.5)

\[ \neg(\forall x)(F \mid G) = (\exists x)(\neg F \mid G) \]  
(4.6)

\[ \neg(\exists x)(F \mid G) = (\forall x)(\neg F \mid G). \]  
(4.7)

It is easy to proof equivalence (4.5). Equivalences (4.6) and (4.7) are proved by Table 4.3. Columns 5 and 7 are identical (this proves (4.6)) and columns 4 and 8 are identical as well (this proves (4.7)).

**Proof of equivalences (4.6) and (4.7)**  
<table>
<thead>
<tr>
<th>(F\mid G)</th>
<th>(\neg(F\mid G))</th>
<th>(\neg x)(F\mid G)</th>
<th>(\neg x)(\neg F\mid G)</th>
<th>(x)(\neg F\mid G)</th>
<th>(\neg(x)(F\mid G))</th>
<th>(\neg x)(\neg F\mid G)</th>
</tr>
</thead>
<tbody>
<tr>
<td>{0}</td>
<td>{1}</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>{1}</td>
<td>{0}</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>{}\</td>
<td>{}</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>{0,1}</td>
<td>{0,1}</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>{0,-}</td>
<td>{1,-}</td>
<td>-</td>
<td>-</td>
<td>1</td>
<td>0</td>
<td>-</td>
</tr>
<tr>
<td>{1,-}</td>
<td>{0,-}</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>0</td>
</tr>
<tr>
<td>{0,1,-}</td>
<td>{0,1,-}</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-</td>
</tr>
</tbody>
</table>

Formula (4.4) contains one total negation operator. Applying equivalences (4.7), (4.5), and one of the de Morgan’s laws to the formula, we generate the formula as follows:

\[ (\forall x)((\forall y)(Q(c,x)+R(a,y) \mid Q(y,b)) \& (\exists x)(\neg R(v,z)\&\neg P(x) \mid P(b))). \]  
(4.8)

**Step 4.** Eliminating the total and partial existential quantifiers by means of introducing a function symbol instead of the variable under the quantifier. If the existential quantifier is located after \(n\) universal quantifiers then the symbol is of degree \(n\). If there are no universal quantifiers before the existential quantifier then \(n=0\). Analyzing formula (4.8), we conclude that the existential quantifier \((\exists x)\) is located after the universal quantifier \((\forall x)\). Therefore, the variable \(z\) is replaced with a function symbol \(f(x)\):

\[ (\forall x)((\forall y)(Q(c,x)+R(a,y) \mid Q(y,b)) \& (\exists x)(\neg R(v,f(x))\&\neg P(x) \mid P(b))). \]  
(4.9)

**Step 5.** Moving the total and partial universal quantifiers to the formula prefix. To perform the moving, equivalences as follows are used:

\[ ((\forall x)(F\mid G) = (\forall x)(F\mid G) \]  
(4.10)

\[ ((\forall x)R \& Q = (\forall x)(R \& Q) \]  
(4.11)

\[ ((\forall x)R \& Q = (\forall x)(R \& Q). \]  
(4.12)

Equivalence (4.10) allows for moving the ordinary total quantifier from the value part of a pair and replacing the total quantifier with a partial quantifier. The equivalence is proved by the contents of Table

**Proof of equivalence (4.10)**  
<table>
<thead>
<tr>
<th>(F) (G)</th>
<th>((\forall x)F)</th>
<th>((\forall x)F \mid G)</th>
<th>((F \mid G))</th>
<th>((\forall x)(F \mid G))</th>
</tr>
</thead>
<tbody>
<tr>
<td>{0}</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>{1}</td>
<td>0</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>{0,1}</td>
<td>0</td>
<td>0</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<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>
</tr>
<tr>
<td>{0,1}</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>{0,1}</td>
</tr>
</tbody>
</table>

Proof of equivalences (4.11) and (4.12)  
<table>
<thead>
<tr>
<th>(R) (Q)</th>
<th>((\forall x)R)</th>
<th>((\forall x)R &amp; Q)</th>
<th>(R &amp; Q)</th>
<th>((\forall x)(R &amp; Q))</th>
<th>((\forall x)(R &amp; Q))</th>
<th>(R &amp; Q)</th>
<th>((\forall x)(R &amp; Q))</th>
</tr>
</thead>
<tbody>
<tr>
<td>{0}</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>{0}</td>
<td>0</td>
<td>{0}</td>
<td>0</td>
</tr>
<tr>
<td>{1}</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>{0}</td>
<td>1</td>
<td>{1}</td>
<td>1</td>
</tr>
<tr>
<td>{}\</td>
<td>0</td>
<td>-</td>
<td>0</td>
<td>{0}</td>
<td>-</td>
<td>{}</td>
<td>-</td>
</tr>
</tbody>
</table>
4.4: columns 4 and 6 of the table are identical. Equivalences (4.11) and (4.12) allow for moving the quantifier from the operands of conjunction and disjunction operators. They are proved by the contents of Table 4.5: columns 4 and 6 as well as columns 7 and 9 are identical. While moving to the prefix, the ordinary quantifiers may be replaced with partial quantifiers.

In order to move the universal quantifier from a partial formula to the prefix we additionally need the equivalence as:

$$(F|\forall x G) = (\forall x)(F|G).$$

(4.13)

Analyzing Table 4.6, we conclude that the equality holds in all the cases except case 3, where the formula $(F|\forall x G)$ takes value '-' and the formula $(\forall x)(F|G)$ takes value 0. The transition from value '-' to value 0 is eligible; therefore equivalence (4.13) can be used during the formula transformation. Applying equivalence (4.11) to formula (4.9), we generate formula (4.14):

$$(\forall x)(\forall y)(\forall v)((Q(c,x)+R(a,y) | Q(y,b)) \& (~R(v,f(x))&~P(x) | P(b)).$$

(4.14)

Step 6. Removing the formula prefix. After the removing, the formula consists of a matrix. Formula (4.14) is directly transformed to

$$((Q(c,x)+R(a,y) | Q(y,b)) \& (~R(v,f(x))&~P(x) | P(b)).$$

(4.15)

Table 4.6

| N | G   | F   | $$(\forall x)G$$ | $$(F|\forall x G)$$ | $$(F|G)$$ | $$(\forall x)(F|G)$$ |
|---|-----|-----|------------------|---------------------|------------|---------------------|
| 1 | \{0\} | 0   | 0                | -                  | \{0\}      | -                   |
| 2 | \{1\} | 0   | 1                | 0                  | \{0\}      | 0                   |
| 3 | \{0,1\} | 0   | 0                | -                  | \{0,-\}    | 0                   |
| 4 | \{0\} | 1   | 0                | -                  | \{1,-\}    | -                   |
| 5 | \{1\} | 1   | 1                | 1                  | \{1\}      | 1                   |
| 6 | \{0,1\} | 1   | 0                | -                  | \{1,-\}    | -                   |

Step 7. Eliminating all the partial logical operators. The elimination is based on laws (3.6) to (3.28). As a result we obtain a pair of total formulas. Applying equivalence (3.21) to formula (4.15), the following formula without partial operators is generated:

$$((Q(c,x)+R(a,y) \& ~R(v,f(x))&~P(x) | (Q(y,b)+P(b))&$$

$$((Q(y,b)+(~R(v,f(x))&~P(x)))&P(b)+~(Q(c,x)+R(a,y)))).$$

(4.16)
Step 8. Transforming the total formulas in the value and domain parts of the pair to a conjunction of disjunctions of literals. Two sets of total clauses are obtained. Performing step 8 for formula (4.16), the resulting sets contain 3 and 4 total clauses respectively:

\[(Q(c,x)+R(a,y)) \land \neg R(v,f(x)) \land \neg P(x)]
\[(Q(y,b)+P(b)) \land (Q(y,b)+R(v,f(x))\land P(x)) \land (P(b)+\neg Q(c,x)) \land (P(b)+\neg R(a,y))].

In this paper, we also use other representation forms for a partial formula.

4.4 Knowledge representation in incompletely specified domain

In addition to representation form (4.1), we shall also use for knowledge representation a partial conjunction of sentences that are partial clauses:

\[Knowledge=C_1 \land \ldots \land C_m, \quad (4.18)\]

\[C_i = (\forall x_1)\ldots(\forall x_n) \ (P_{i,1} \pm \ldots \pm P_{i,m}), \quad (4.19)\]

where \(P_{i,j}\) is a pair \((F|G)\) of total formulas. In particular, formulas \(F\) and \(G\) can be literals. The universal quantifiers will be omitted. The partial clause

\[(L_1|D_1)\pm \ldots \pm (L_k|D_k)\pm (\neg L_{k+1}|D_{k+1})\pm \ldots \pm (\neg L_m|D_m) \quad (4.20)\]

contains \(k\) positive literals in the value parts of pairs and \(m-k\) negative literals. The empty clause includes one pair \((0|D_i)\).

Similarly to the Horn clause in the traditional first order predicate logic we introduce a Horn partial clause with one positive literal:

\[(L|D)\pm \neg (L_1|D_1)\pm \ldots \pm \neg (L_m|D_m) \quad (4.21)\]

It is easy to see that partial clause (4.20) is transformed to a pair the value part of which is a total clause and the domain part is a total formula:

\[(L_1+\ldots+L_k+\neg L_{k+1}+\ldots+\neg L_m \mid D_1 \land \ldots \land D_m \land L_1 \land D_1 + \ldots + L_k \land D_k + \neg L_{k+1} \land D_{k+1} + \ldots \neg L_m \land D_m). \quad (4.22)\]

Horn partial clause (4.21) is transformed to a partial consequence:

\[(L|D)\pm \neg (L_1|D_1)\pm \ldots \pm \neg (L_m|D_m) = (L|D)\leftarrow ((L_1|D_1)\land \ldots \land (L_m|D_m)) \quad (4.23)\]

Consequence (4.23) will be called a partial rule. One pair \((L|D)\) will be called a partial fact.

5. Deductive reasoning in partial logic

Presented in work [8] is a formulation of first order logic that is specifically designed for use as the basic theoretical instrument of a computer theorem-proving program. Traditionally, a single step in a deduction has been required, for pragmatic and psychological reasons, to be simple enough. In the system described in [8], one inference principle is used. It is called the resolution principle. It is machine-oriented and forms a complete system of first order predicate logic. In this paper the resolution principle is generalized for incompletely specified domains.

We use the deductive reasoning formalism that is based upon the notions of Herbrand interpretation, unsatisfiability and refutation. A set of clauses is satisfiable if there is a model containing no complementary pair of the two literals \(L\) and \(\neg L\).
5.1 Goals and theorems

Given a set \( S = \{C_1, \ldots, C_m\} \) of partial clauses (pieces of knowledge) and a target partial formula (goal) \( R \), we formulate the following theorem:

\[
C_1 \land \ldots \land C_m \rightarrow R. \quad (5.1)
\]

The formula \( R \) represents a new piece of knowledge that has to be inferred from \( C_1, \ldots, C_m \). In order to prove the theorem we find a model being a set of variable values satisfying formula (5.1). The formula is satisfied if it takes value 1 or value don’t care that can be replaced with value 1.

5.2 Proving by refutation

To prove the formula (5.1) be satisfiable we use the refutation procedure and prove the formula as follows be unsatisfiable:

\[
C_1 \land \ldots \land C_m \land \neg R. \quad (5.2)
\]

Based on the resolution principle and starting with the clauses \( C_j, \ldots, C_m \) and \( \neg R \), a sequence of additional clauses (resolvents) is generated until a clause consisting of one pair \((0|L)\) appears.

5.3 Generalization of resolution principle

First, we consider a generalization of the resolution principle ground partial clauses. Given two partial clauses

\[
C_1 = (L_1|D_1) + (L_2|D_2) \quad (5.3)
\]

and

\[
C_2 = (\neg L_1|D_3) + (L_3|D_4) \quad (5.4)
\]

where \( L_1 \) and \( \neg L_1 \) is a complementary pair of ground literals, the following additional partial clause (resolvent) is added to the source clauses:

\[
C = (0|D_1&D_3+\neg L_1&D_1+L_1&D_3) + (L_2|D_2) + (L_3|D_4). \quad (5.5)
\]

The pairs \((L_2|D_2)\) and \((L_3|D_4)\) from the source clauses are included in the resolvent. An additional pair containing the logical value 0 in the value part appears.

To prove \( C \) be the resolvent we have to prove that \( C_1 \land C_2 \) is satisfied iff \( C_1 \land C_2 \land C \) is satisfied. The partial conjunction \( C_1 \land C_2 \) of two clauses is transformed as follows:

\[
\begin{align*}
C_1 \land C_2 &= ((L_1|D_1) + (L_2|D_2)) \land ((\neg L_1|D_3) + (L_3|D_4)) = \\
&= (L_1 + L_2|D_1&D_2+L_1&D_1+L_2&D_2) \land (\neg L_1+L_3|D_3&D_4+\neg L_1+L_3&D_3+L_3&D_4) = \\
&= ((L_1+L_2)&\neg L_1+L_3) \land ((D_1&D_2+L_1&D_1+L_2&D_2)&(D_3&D_4+\neg L_1+L_3&D_3+L_3&D_4) + \\
&\quad \land D_1&D_2&\neg L_1+L_3&D_1&D_2&\neg L_1+L_3 = \\
&= (L_1+L_2)&\neg L_1+L_3 \quad D_1&D_2&D_3&D_4 & D_1&D_2&D_3&\neg L_1+L_3 = \\
&\quad \land D_1&D_2&D_4&D_5 & D_1&D_2&D_3&D_4&D_5 &= (5.6)
\end{align*}
\]

The resolvent \( C \) can be represented as:

\[
C = (0|D_1&D_2+\neg L_1+L_1&D_1+L_1&D_3) + (L_2|D_2) + (L_3|D_4) = \\
(L_2+L_3) \land (D_1&D_2+L_1&D_1+L_1&D_3) + (D_2&D_4 + (L_2+L_3) & (D_2&D_4) = \\
(L_2+L_3) \land (D_1&D_2&D_3&D_4 + D_1&D_2&D_3&D_4 & L_1) + \\
D_2&D_3&D_4 & L_2+L_3 = (5.7)
\]

Performing the partial conjunction operation on the operand \( C_1 \land C_2 \) presented by (5.6) and on the operand \( C \) presented by (5.7) we obtain the following partial formula:
\[
C_1 \& C_2 \& C = ((L_1 + L_2) \& (\neg L_1 + L_3) \& (L_2 + L_3)) \\
D_1 \& D_2 \& D_3 \& D_4 + D_1 \& D_2 \& D_3 \& \neg L_1 + D_1 \& D_2 \& D_3 \& L_3 + \\
D_1 \& D_2 \& \neg L_1 \& \neg L_2 + D_1 \& D_3 \& D_4 \& L_1 + D_1 \& D_3 \& L_1 \& L_3 + \\
D_2 \& D_3 \& D_4 \& L_2 + D_2 \& D_3 \& \neg L_1 \& L_2 + D_2 \& D_4 \& L_2 \& L_3 + D_3 \& D_4 \& L_3 \& \neg L_3).
\]

Comparing the formulas (5.6) and (5.8), it is easy to see that the domain parts of the pairs are the same, and the value part of formula (5.8) contains an additional total resolvent compared to the value part of formula (5.6). Therefore, formula (5.6) is satisfied iff the formula (5.8) is satisfied.

**Example 5.1.** Given the partial formula
\[
(A+B|\neg C) \& (~A|B) \& (C+D|A) \& (~D|C),
\]
the goal is to prove that the formula is not satisfiable. We brake down the formula into the set of partial clauses as follows:

1. \((A+B|\neg C)\) - source clause
2. \((\neg A|B)\) - source clause (5.10)
3. \((C+D|A)\) - source clause
4. \((\neg D|C)\) - source clause

Applying the resolution method to pairs of partial clauses, we generate partial resolvents until the empty partial clause appears:

5. \((B|B\&\neg C)\) - resolvent for clauses 1 and 2
6. \((A|B\&\neg C)\) = \((\neg C|B\&\neg C)\) - from clause 5
7. \((D|A\&B\&\neg C\&A\&D)\) = \((D|A\&B\&D\&\neg C\&D)\) - resolvent for clauses 3 and 6
8. \((0|A\&D\&C)\) - resolvent for clauses 4 and 7

Clause 8 is the empty partial clause, therefore formula (5.9) is not satisfiable.

### 5.4 Generation of a resolvent

If the source partial clauses look like
\[
C_1 = (L_1 + L_2 \mid D_1)
\]
and
\[
C_2 = (\neg L_1 + L_3 \mid D_2)
\]
then the resolvent derived from the clauses is as follows:
\[
C = (L_2 + L_3 \mid D_1 \& D_2 + L_2 \& D_1 + L_3 \& D_2).
\]

To prove (5.13) we transform clauses (5.11) and (5.12) in the following way: \(C_1 = (L_1 \mid D_1) \oplus (L_2 \mid D_1)\) and \(C_2 = (\neg L_1 \mid D_2) \oplus (L_3 \mid D_2)\). Based on expression (5.5), the following resolvent is constructed from the clauses:
\[
C = (0 \mid D_1 \& D_2 + \neg L_1 \& D_1 \& L_2 \& D_2 \& D_1 + L_3 \& D_2) = \(L_2 + L_3 \mid D_1 \& D_2 + L_2 \& D_1 + L_3 \& D_2\).
\]

If the clauses are not ground then the unification algorithm from [8] has to be used.

### 5.5 Resolution principle for clauses in consequence form

Starting with a set of rules (clauses in the consequence form) and a target formula (goal), we transform the current goal to another one step by step through using the resolution principle until an empty clause appears. The current goal is represented in the consequence form as follows:
\[
(0 \leftarrow L_1 \& L_2 \mid D_1).
\]
The clause that is used for transforming the current goal to the next-step goal is also represented in the consequence form as:

\[(L_1 \leftarrow L_3)[D_2].\]  \hspace{1cm} (5.16)

We translate the goal and rule to the form of disjuncts \((\sim L_1 + \sim L_2)[D_1]\) and \((L_1 + \sim L_3)[D_2]\).

Applying the resolution principle, we obtain the following resolvent:

\[(-L_2 + \sim L_3)[D_1 \& Q + \sim L_2 \& D_1 + \sim L_3 \& D_2].\]  \hspace{1cm} (5.17)

that is translated to the next-step goal:

\[(0 \leftarrow L_2 \& L_3 \| D_1 \& D_2 + \sim L_2).\]  \hspace{1cm} (5.18)

The value part of the goal can be simplified through using laws (3.57-3.63). If a fact as

\[(L_1 \| D_2)\] \hspace{1cm} (5.19)

is used instead of rule (5.16) to generate a resolvent, then the next-step goal is as follows:

\[(0 \leftarrow L_2 \| (D_2 + \sim L_2)).\] \hspace{1cm} (5.20)

6. Transition from partial to traditional total logic

In order to use the proposed mechanism in practice for knowledge representation and deductive reasoning, an appropriate software or hardware should be developed. We propose a method of transition from the partial to the traditional first order predicate logic consisting of four steps.

1. Transforming the partial formula to the pair \((F|G)\) of total formulas.
2. Replacing the value part \(F\) of the pair with a total formula \(Value_Part\) from the range \(F \& G \subseteq Value_Part \subseteq F + \sim G.\) \hspace{1cm} (6.1)

The new pair is as follows:

\[(Value_Part \| G).\] \hspace{1cm} (6.2)

3. Transition from the pair with the domain part \(G\) to the pair with the domain part 1 and replacing all the partial universal quantifiers (that are omitted) with the traditional total quantifiers:

\[(Value_Part \| 1).\] \hspace{1cm} (6.3)

4. Transforming the \(Value_Part\) to a set of clauses in the traditional first order predicate logic:

\[C_1 & \ldots & C_m.\] \hspace{1cm} (6.4)

Three methods are possible when selecting the \(Value_Part\) at step 3:

- to use both 0 and 1 for replacing ‘-’ (method 1),
- to replace the value ‘-’ with the value 1 (method 2),
- to replace the value ‘-’ with the value 0 (method 3).

6.1 Transition to total logic (method 1)

When we use the both values 0 and 1 for replacing the value ‘-’, the \(Value_Part\) is selected from the range \(F \& G \subseteq Value_Part \subseteq F + \sim G.\) In particular, the \(Value_Part=F\) can be used.

Example 6.1. This example continues example 5.1. We transform partial formula (5.9) to a total conjunction of total disjuncts in the following way:

\[(A+B \| 1) \&(\sim A \| 1) \&(C+D \| 1) \&(\sim D \| 1) = ((A+B) \& \sim A \& (C+D) \& \sim D \| 1) = (A+B) \& (\sim A) \& (C+D) \& (\sim D).\] \hspace{1cm} (6.5)

The following set of four total clauses is derived from (6.5):
1. A+B
2. ~A
3. C+D
4. ~D.

It is easy to see that it is impossible to infer the empty clause from clauses (6.6).

### 6.2 Transition to total logic (method 2)

If we use only the value 1 for replacing the value ‘-’, then Value_Part = F+~G.

**Example 6.2.** Let us transform partial formula (5.9) to a total conjunction of disjuncts by means of replacing the value part of pairs with disjunction of their value and domain parts:

\[(A+B+C|\sim C)&((\sim A+\sim B|B)\&((C+D+\sim A|A)\&((\sim D+\sim C|C) =
\]
\[(A+B+C|\sim A+\sim B|B)\&((C+D+\sim A|A)\&((\sim D+\sim C|C) I) =
\]
\[((A+B+C)\&((\sim A+\sim B)&(C+D+\sim A)&(\sim D+\sim C) | I) =
\]
\[(A+B+C)\&((\sim A+\sim B)&(C+D+\sim A)&(\sim D+\sim C). \quad (6.7)
\]

The following set of four clauses is derived from (6.7):

1. A+B+C
2. ~A+~B
3. C+D+~A
4. ~D+~C.

It is easy to see that it is impossible to infer the empty clause from clauses (6.8).

### 6.3 Transition to total logic (method 3)

If we use only the value 0 for replacing the value ‘-’, then the Value_Part = F&G.

**Example 6.3.** Now we replace the value part of pairs in partial formula (5.9) with total conjunction of the pair’s value and domain parts:

\[((A+B)&(\sim C)&((\sim A)&(B))\&((C+D)&(A)|(A)\&((\sim D)&(C)|C) =
\]
\[((A+B)&(\sim C)&((\sim A)&(B))\&((C+D)&(A)|(A)\&((\sim D)&(C)|C) I) =
\]
\*)((A+B)&(\sim C)&((\sim A)&(B))\&((C+D)&(A)|(A)\&((\sim D)&(C) | I) =
\]
\[(A+B)&(\sim C)&((\sim A)&(B))\&((C+D)&(A)|(A)\&((\sim D)&(C). \quad (6.9)
\]

The following set of eight clauses is derived from (6.9):

1. A+B
2. ~C
3. ~A
4. B
5. C+D
6. A
7. ~D
8. C.

It is easy to see that it is sufficient to generate only one resolvent in order to infer the empty clause from clauses (6.10). In particular, the literals C and ~C constitute a complementary pair.

Method 3 seems to be the most preferable, because it speeds up the generation of the empty clause and the technique for transition from the partial formula to a set of clauses in the traditional logic is the most efficient. Comparing methods 1, 2, and 3, we conclude that only method 3 can be used in practice.
6.4 Translation of partial consequence

Consider the following rule in the form of partial consequence:

\[(A|1) \leftarrow (C|Q).\] (6.11)

In pair \((A|1)\) the domain function equals 1. Therefore, the pair value is equal to the value of literal \(A\). The pair \((C|Q)\) takes one of three values: \textit{false}, \textit{true} and \textit{don’t care}. The \textit{don’t care} value can be replaced with \textit{false} or \textit{true} arbitrarily. As a result, one of the completely specified truth functions belonging to the range \(C\&Q\) to \(C+\neg Q\) can be obtained. In order to generate all the possible solutions for a given goal, we select the function \(C+\neg Q\). Formula (6.11) is transformed to the following expression:

\[A \leftarrow (C + \neg Q).\] (6.12)

It is easy to see that expression (6.12) is equivalent to the following conjunction of two total consequences:

\[(A \leftarrow \neg Q) \& (A \leftarrow C).\] (6.13)

Therefore, one consequence (6.11) in the partial logic is translated to two total consequences in the first order predicate logic.

7. Results

Now we consider an example of knowledge representation and deductive reasoning in an incompletely specified domain. The proposed methods of knowledge description and logic inference allow for reduction in the number of clauses and in the number of inference steps.

7.1 Incompletely specified finite state machine (FSM)

The finite state machine (FSM) is widely used for modeling and synthesis of digital sequential circuits and systems. The FSM is a five-tuple

\[FSM = (X,Y,S,\varphi,\delta),\] (7.1)

where \(X\) is a set of input symbols; \(Y\) is a set of output symbols; \(S\) is a set of internal states; \(\varphi: S\times X \to S\) is a next-state function; \(\delta: S\times X \to Y\) is an output function. The FSM is incompletely specified if the next-state and output functions are incompletely specified, that is the functions take additionally the value \textit{don’t care} or \textit{‘--’} value. An example FSM is presented in Table 7.1 where the pair \(s/y\) defines the next state \(s\) and the output symbol \(y\).

<table>
<thead>
<tr>
<th>State</th>
<th>x1</th>
<th>x2</th>
<th>x3</th>
<th>x4</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0</td>
<td></td>
<td>s1</td>
<td>s2</td>
<td>s3</td>
</tr>
<tr>
<td>s1</td>
<td>y2</td>
<td>y1</td>
<td>y3</td>
<td>y2</td>
</tr>
<tr>
<td>s2</td>
<td>y3</td>
<td>y3</td>
<td>y1</td>
<td>y4</td>
</tr>
<tr>
<td>s3</td>
<td>y2</td>
<td>y2</td>
<td>y3</td>
<td>y2</td>
</tr>
<tr>
<td>s4</td>
<td>y3</td>
<td>y3</td>
<td>y1</td>
<td>y1</td>
</tr>
</tbody>
</table>

where \(X\) is a set of input symbols; \(Y\) is a set of output symbols; \(S\) is a set of internal states; \(\varphi: S\times X \to S\) is a next-state function; \(\delta: S\times X \to Y\) is an output function. The FSM is incompletely specified if the next-state and output functions are incompletely specified, that is the functions take additionally the value \textit{don’t care} or \textit{‘--’} value. An example FSM is presented in Table 7.1 where the pair \(s/y\) defines the next state \(s\) and the output symbol \(y\).

7.2 Representation of knowledge about FSM

The \textit{don’t care} value used in the FSM definition is rather a value of the domain variables next state and output symbol, but is not a logical value. We could formulate a set of clauses for FSM without the theoretical results obtained in this paper. We show that FSM representation in partial logic requires less number of clauses.
The input symbols \( x_1, x_2, x_3, x_4 \), the output symbols \( y_1, y_2, y_3, y_4 \), and the internal states \( s_0, s_1, s_2, s_3, s_4 \) are represented by function symbols of degree 0, i.e. by individual constants. The sequences of input symbols, output symbols, and internal states are represented as lists.

In order to define clauses about the FSM, a set of predicates is introduced. Among them there are the predicates as follows: \( \text{next} \_\text{state}(s,x,sn) \), \( \text{state} \_\text{specified}(s,x,sn) \), \( \text{state} \_\text{dontcare}(s,x,sn) \), \( \text{output}(s, x,y) \), \( \text{output} \_\text{specified}(s,x,y) \), and \( \text{output} \_\text{dontcare}(s,x,y) \), where \( s \) is the current state, \( sn \) is the next state, \( x \) is the input symbol, and \( y \) is the output symbol. Using these predicates, we could construct the three-valued clause as follows:

\[
\text{(next} \_\text{state}(s,x,sn) \mid 1) \quad \text{if} \quad (\text{state} \_\text{specified}(s,x,sn) \mid \text{not} \text{ state} \_\text{dontcare}(s,x,sn)).
\]

According to (6.13) the clause is splitted to following two-valued clauses:

\[
\text{next} \_\text{state}(s,x,sn) \quad \text{if} \quad \text{state} \_\text{dontcare}(s,x,sn).
\]
\[
\text{next} \_\text{state}(s,x,sn) \quad \text{if} \quad \text{state} \_\text{specified}(s,x,sn).
\]

Following this way, we obtain the Prolog program shown in Fig.7.1.

### 7.3 Logic inference tasks concerning FSM

The knowledge presented in Fig.7.1 allows for solving various logic inference tasks, including as follows:

- what is the output symbol sequence for the given input symbol sequence
- what is the input symbol sequence if any exists, given the output symbol sequence
- what is the FSM state sequence, given the input or output symbol sequence
- are there any unreachable states,

and others.

The goal

\[
\text{mapping}(s_0,Xl,[y_3,y_1,y_2,y_3,y_2,y_4]) \quad \text{and} \quad \text{write}("Xl=",XI),
\]

described in Fig.7.1 denotes the task consisting in searching for an input symbol sequence \( Xl \), given the output symbol sequence \( [y_3,y_1,y_2,y_3,y_2,y_4] \).

Comparing the models of knowledge representation and deductive reasoning, constructed through using the proposed methods of partial logic, with the existing logic inference techniques, we can conclude that our models speed up the logic inference process and increase the efficiency of knowledge representation in incompletely specified domains. Thus, the FSM representation presented in Fig.7.1 contains 55 clauses. If the proposed methods are not used then 90 clauses are needed in order to describe the FSM.

### domains

\[
\begin{align*}
\text{state} &= s_0; s_1; s_2; s_3; s_4 \\
\text{input} \_\text{symbol} &= x_1; x_2; x_3; x_4 \\
\text{output} \_\text{symbol} &= y_1; y_2; y_3; y_4 \\
\text{sequence} \_\text{of} \_\text{states} &= \text{state}^* \\
\text{sequence} \_\text{of} \_\text{inputs} &= \text{input} \_\text{symbol}^* \\
\text{sequence} \_\text{of} \_\text{outputs} &= \text{output} \_\text{symbol}^*
\end{align*}
\]

### predicates

\[
\begin{align*}
\text{mapping} \_\text{if} & \quad \text{state}, \text{sequence} \_\text{of} \_\text{inputs}, \text{sequence} \_\text{of} \_\text{outputs} \\
\text{next} \_\text{state} \_\text{if} & \quad \text{state}, \text{input} \_\text{symbol}, \text{state} \\
\text{state} \_\text{specified} \_\text{if} & \quad \text{state}, \text{input} \_\text{symbol}, \text{state} \\
\text{state} \_\text{dontcare} \_\text{if} & \quad \text{state}, \text{input} \_\text{symbol}, \text{state} \\
\text{sx} \_\text{y} \_\text{dc} \_\text{if} & \quad \text{state}, \text{input} \_\text{symbol} \\
\text{each} \_\text{of} \_\text{y} \_\text{if} & \quad \text{output} \_\text{symbol}
\end{align*}
\]

\[
\begin{align*}
\text{output} \_\text{specified} \_\text{if} & \quad s_4, x_1, y_4 \\
\text{output} \_\text{specified} \_\text{if} & \quad s_4, x_2, y_1 \\
\text{output} \_\text{dontcare} \_\text{if} & \quad \text{state}, \text{X}, \text{Y} \quad \text{if} \quad \text{sx} \_\text{y} \_\text{dc} \_\text{if} \quad \text{state}, \text{X} \_\text{and} \_\text{each} \_\text{of} \_\text{y} \_\text{if} \quad \text{Y}. \\
\text{sx} \_\text{y} \_\text{dc} \_\text{if} & \quad s_1, x_2 \\
\text{sx} \_\text{y} \_\text{dc} \_\text{if} & \quad s_1, x_3 \\
\text{sx} \_\text{y} \_\text{dc} \_\text{if} & \quad s_2, x_3 \\
\text{sx} \_\text{y} \_\text{dc} \_\text{if} & \quad s_3, x_1 \\
\text{sx} \_\text{y} \_\text{dc} \_\text{if} & \quad s_3, x_2 \\
\text{sx} \_\text{y} \_\text{dc} \_\text{if} & \quad s_4, x_3 \\
\text{sx} \_\text{y} \_\text{dc} \_\text{if} & \quad s_4, x_4 \\
\text{each} \_\text{of} \_\text{y} \_\text{if} & \quad y_1 \\
\text{each} \_\text{of} \_\text{y} \_\text{if} & \quad y_2 \\
\text{each} \_\text{of} \_\text{y} \_\text{if} & \quad y_3 \\
\text{each} \_\text{of} \_\text{y} \_\text{if} & \quad y_4.
\end{align*}
\]
output(state,input_symbol,output_symbol)
output_specified(state,input_symbol,
output_symbol)
output_dontcare(state,input_symbol,
output_symbol)
sx_s_dc(state,input_symbol)
each_of_s(state)
goal
mapping(s0,Xl,[y3,y1,y2,y3,y2,y4]) and
write("Xl=".Xl).
clauses
mapping(_,[],[]).
mapping(S,[X|Xl],[Y|Yl]) if output(S,X,Y)
and
next_state(S,X,Sn) and
mapping(Sn,Xl,Yl).
output(S,X,Y) if output_specified(S,X,Y)
or
output_dontcare(S,X,Y).
output_specified(s0,x1,y1).
output_specified(s0,x2,y3).
output_specified(s0,x3,y2).
output_specified(s0,x4,y2).
output_specified(s1,x1,y3).
output_specified(s1,x4,y1).
output_specified(s2,x1,y2).
output_specified(s2,x2,y2).
output_specified(s2,x4,y4).
output_specified(s3,x3,y4).
output_specified(s3,x4,y3).
next_state(S,X,Sn) if state_specified(S,X,Sn) or
state_dontcare(S,X,Sn).
state_specified(s0,x2,s0).
state_specified(s0,x3,s4).
state_specified(s0,x4,s1).
state_specified(s1,x1,s2).
state_specified(s1,x2,s0).
state_specified(s1,x3,s3).
state_specified(s2,x3,s1).
state_specified(s2,x4,s2).
state_specified(s3,x1,s1).
state_specified(s3,x2,s4).
state_specified(s4,x2,s3).
state_specified(s4,x3,s2).
state_specified(s4,x4,s3).
state_dontcare(S,X,Sn) if sx_s_dc(S,X) and
each_of_s(Sn).
sx_s_dc(s0,x1).
sx_s_dc(s1,x4).
sx_s_dc(s2,x1).
sx_s_dc(s2,x2).
sx_s_dc(s3,x3).
sx_s_dc(s3,x4).
sx_s_dc(s4,x1).
each_of_s(s0).
each_of_s(s1).
each_of_s(s2).
each_of_s(s3).
each_of_s(s4).

Figure 7.1. Prolog-description of incompletely specified finite state machine

8. Acknowledgements
The authors are grateful to Prof. E. Zavadskas and Prof. A. Kaklauskas who encouraged submitting the paper to the BalticDB&IS’2000 Conference. Our thanks are also due to Prof. R. Pliuskevicius and A. Pliuskeviciune for their interest to the topic and useful comments.

9. References


