Methods of 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 with 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.

Logic systems mostly constitute a theoretical basis for knowledge-based technologies. Logic approaches can be classified on [1-11]:

- the set of truth values
- methods of construction assertions and definition of semantics
- methods of reasoning.

Taking account of the set of truth-values, the following logic systems are used:

- two-valued Boolean logic with the set of truth-values {*false, true*}
- multiple-valued logics with a finite or infinite set *T* of elements
- probability-based logic with the truth-values belonging to interval [0,…1]
- fuzzy logic with the set of truth-values consisting of fuzzy subsets of set *T*, and other logics.

Taking account of methods of constructing assertions and defining semantics, the logic systems are listed as follows:

- two-valued proposition, first-, and higher-order predicate logics
- multiple-valued proposition, first-, and higher-order predicate logics
- probability-based and fuzzy proposition and first-order predicate logics, and others.

The existing logics use the following reasoning methods:

- inference rules (modus ponens and others)
- resolution principle
- Bayes rule
- solution of nonlinear program in fuzzy logic and other methods.

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. Although the value *don't care* has been used for many years, a novel aspect is that the partial logic allows for manipulation of the clause domains and simplification of the clauses during logic inference.

The paper is organized as follows. In section 2 we give a background of knowledge representation and reasoning. Basic concepts of partial proposition logic are considered in section 3. The syntax, semantics, and transformation procedures of partial first order predicate logic are presented in section 4. In section 5 the Robinson's resolution principle is generalized in the partial logic. Methods of transition from the partial to traditional first order predicate logic are proposed in section 6. Section 7 includes some experimental results. We conclude the work in section 8.

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 potential of the functions is 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 $f(x_1,...,x_n)$ is a mapping $f\colon B^n \to B$ where $B=\{0,1\}$. A partial function $g(y_1,...,y_m)$ is a mapping g: $M^m \rightarrow M$ where $M=\{0,1,–\}$. An incompletely specified function $h(x_1,...,x_n)$ 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 $g(y_1,...,y_m)$ 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]:

$$y_i = \begin{cases} 0 & \text{if } v_i=0 \text{ and } di=1 \\ 1 & \text{if } v_i=1 \text{ and } d_i=1 \\ dc & \text{if } v_i\in\{0,1\} \text{ and } d_i=0. \end{cases} \tag{3.1}$$

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 $z=g(y_1,\ldots,y_m)$ of m three-valued arguments is represented as an incompletely specified function $z'=g'((v_1|d_1),...,(v_m|d_m))$ of $2*m$ two-valued arguments. The partial function z' can be considered as a pair $(v|d)$ of total functions depending on the primary variables v_1, d_1, ... ,v_m, d_m. In this case the Boolean space is broken down into three parts as shown in Fig.3.1:

- on-set g^{on} is a part (a set of vector values) where the function takes value 1, the part is described by the expression $g^{on}=(v\&d)^{on}$ where $\&$ is the total *conjunction* operation,
- off-set g^{off} is a part where the function takes value 0, the part is described by the expression $g^{off}=(\sim v\&d)^{on}$ where $\sim$ is the total *negation* operation,
- don't care set g^{dc} is a part where the function takes value *don't care*, the part is described by the expression $g^{dc}=(\sim d)^{on}$.

Logic assertions that can be formulated concerning the third part take value don't care that can be replaced with value *true* or value *false* arbitrarily.

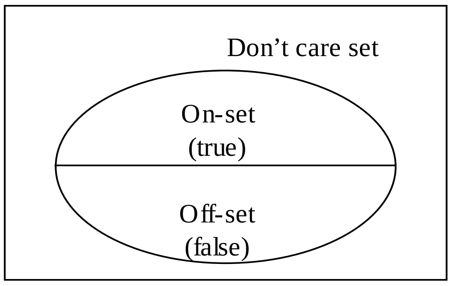

Figure 3.1. Structure of Boolean space

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 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_1.

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 $(v_i|d)$ 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

$$(v\&d)^{on} \subseteq v_i^{on} \subseteq (v+{\sim}d)^{on} \qquad (3.2)$$

holds where + is the total *disjunction* operation. A minimisation operation $min(v|d)$ is a mapping 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 $min(v|d)$ are possible [7]. These depend on which representation forms for v and d are used.

Monadic partial logical operations Table 3.1

N	Operation name	Partial variable y			Operator	Representation
		0	1	–		in VDR
1	Constant 0	0	0	0	$\underline{c0}(y_1)$	$\underline{c0}(v_1\|d_1)$
2	Constant 1	1	1	1	$\underline{c1}(y_1)$	$\underline{c1}(v_1\|d_1)$
3	Constant –	–	–	–	$\underline{c–}(y_1)$	$\underline{c–}(v_1\|d_1)$
4	Identity	0	1	–	$\underline{=}(y_1)$	$\underline{=}(v_1\|d_1)$
5	Negation	1	0	–	$\underline{\sim}(y_1)$	$\underline{\sim}(v_1\|d_1)$

Dyadic partial logical operations Table 3.2

N	Operation name	Partial variables y_1 and y_2									Operator	Representation
		0	1	–	0	1	–	0	1	–		in
		0	0	0	1	1	1	–	–	–		VDR
1	Conjunction	0	0	0	0	1	–	0	–	–	$y_1\underline{\&}y_2$	$(v_1\|d_1)\underline{\&}(v_2\|d_2)$
2	Disjunction	0	1	–	1	1	1	–	1	–	$y_1\underline{+}y_2$	$(v_1\|d_1)\underline{+}(v_2\|d_2)$
3	Sheffer stroke	1	1	1	1	0	–	1	–	–	$y_1\underline{/}y_2$	$(v_1\|d_1)\underline{/}(v_2\|d_2)$
4	Pierce arrow	1	0	–	0	0	0	–	0	–	$y_1\underline{\downarrow}y_2$	$(v_1\|d_1)\underline{\downarrow}(v_2\|d_2)$
5	Implication	1	0	–	1	1	1	1	–	–	$y_1\underline{\rightarrow}y_2$	$(v_1\|d_1)\underline{\rightarrow}(v_2\|d_2)$
6	Consequence	1	1	1	0	1	–	–	1	–	$y_1\underline{\leftarrow}y_2$	$(v_1\|d_1)\underline{\leftarrow}(v_2\|d_2)$
7	Negation of implication	0	1	–	0	0	0	0	–	–	$y_1\underline{\sim>}y_2$	$(v_1\|d_1)\underline{\sim>}(v_2\|d_2)$
8	Negation of consequence	0	0	0	1	0	–	–	0	–	$y_1\underline{<\sim}y_2$	$(v_1\|d_1)\underline{<\sim}(v_2\|d_2)$
9	Exclusive OR	0	1	–	1	0	–	–	–	–	$y_1\underline{\oplus}y_2$	$(v_1\|d_1)\underline{\oplus}(v_2\|d_2)$
10	Equivalence	1	0	–	0	1	–	–	–	–	$y_1\underline{\equiv}y_2$	$(v_1\|d_1)\underline{\equiv}(v_2\|d_2)$

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 operators are used for construction of partial formulas. If the total logical operators such as *negation* (~), *conjunction* (&), *disjunction* (+), *Sheffer stroke* (/), *Pierce arrow* ($\downarrow$), *implication* ($\rightarrow$), *negation of implication* (~>), *consequence* ($\leftarrow$), *negation of consequence* (<~), *exclusive OR* ($\oplus$), and *equivalence* ($\equiv$) are used as well, then the formula will be called a mixed total-partial formula. The formula as follows

$$(x1|x2)\ \underline{\&}\ ((x2|x3)\ \underline{+}\ (x3|x2)\ \underline{\rightarrow}\ (x4|x1)) \qquad (3.3)$$

is a partial formula, and the following formula:

$$(x1\&x2|x2\rightarrow x3)\ \underline{\oplus}\ (x2|x1+x3+x4)\ \underline{+}\ (\sim x3+\sim x4|x1) \qquad (3.4)$$

is a mixed total-partial formula where *x1, x2, x3, x4* are total Boolean variables. The expressions of the considered type represent incompletely specified functions through using partial operations.

Example 3.1. The incompletely specified function defined in Table 3.3 is represented by the following total-partial expression: $f=(\sim x_1+\sim x_2+\sim x_3|x_1\&x_2+\sim x_3\&(x_1+x_2))$.

Example incompletely specified function Table 3.3

x_1	x_2	x_3	f
0	0	0	–
0	0	1	–
0	1	0	1
0	1	1	–
1	0	0	1
1	0	1	–
1	1	0	1
1	1	1	0

Example 3.2. Let we have three total assertions as follows:

"*The student studies at a state university*" – *X*

"*The student will work at a state institution*" – *Y*

"*He / she is a belarusian student*" – *Z.*

Based on the total assertions and the proposition variables *X*, *Y*, and *Z* the following partial assertion can be constructed:

Y if X on Z $\Rightarrow$ $(Y\leftarrow X \mid Z)$.

The partial assertion consists of two parts: the value part *Y if X* (or $Y\leftarrow X$) and the domain part *Z*. The assertion is interpreted with the partial truth function presented in Table 3.4. It is easy to see, the assertion $Y\leftarrow X$ is essential if the student is a belarusian one and is not essential in opposite case.

Partial truth function Table 3.4

Proposition variables			Partial assertion
X	Y	Z	
false	*false*	*false*	*don't care*
false	*false*	*true*	*true*
false	*true*	*false*	*don't care*
false	*true*	*true*	*true*
true	*false*	*false*	*don't care*
true	*false*	*true*	*false*
true	*true*	*false*	*don't care*
true	*true*	*true*	*true*

3.4 PARTIAL LOGIC LAWS

We formulate a partial logic law as

$$Lp = Rp, \qquad (3.5)$$

where *Lp* and *Rp* are the left and right parts respectively in the law. The parts *Lp* and *Rp* are partial formulas. The values of the formulas have to belong the set *M* and have to be the same at any values of total proposition variables v, d, v_1, d_1, v_2, and d_2 occurring in the formulas. In this section we consider three types of laws:

- the laws that maintain transformation of an arbitrary partial proposition logic formula to a pair of total formulas
- the laws that are generalizations for appropriate laws in the traditional logic
- the novel laws of partial logic that maintain an equivalent 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 using the laws as follows:

$$\underline{c0}(v|d) = (0|1) \tag{3.6}$$
$$\underline{c1}(v|d) = (1|1) \tag{3.7}$$
$$\underline{c{-}}(v|d) = (v|0) \tag{3.8}$$
$$\underline{\equiv}(v|d) = (v|d) \tag{3.9}$$
$$\underline{\sim}(v|d) = (\sim v|0). \tag{3.10}$$

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

$$(v_1|d_1)\underline{\&}(v_2|d_2) = (v_1\&v_2|d_1\&d_1+\sim v_1\&d_1+\sim v_2\&d_2), \tag{3.11}$$
$$(v_1|d_1)\underline{+}(v_2|d_2) = (v_1+v_2|d_1\&d_1+v_1\&d_1+v_2\&d_2), \tag{3.12}$$
$$(v_1|d_1)\underline{/}(v_2|d_2) = (v_1/v_2|d_1\&d_1+\sim v_1\&d_1+\sim v_2\&d_2), \tag{3.13}$$
$$(v_1|d_1)\underline{\downarrow}(v_2|d_2) = (v_1\downarrow v_2|d_1\&d_1+v_1\&d_1+v_2\&d_2), \tag{3.14}$$
$$(v_1|d_1)\underline{\rightarrow}\&(v_2|d_2)= (v_1\rightarrow v_2|d_1\&d_1+\sim v_1\&d_1+v_2\&d_2), \tag{3.15}$$
$$(v_1|d_1)\underline{\leftarrow}\&(v_2|d_2)= (v_1\leftarrow v_2|d_1\&d_1+v_1\&d_1+\sim v_2\&d_2), \tag{3.16}$$
$$(v_1|d_1)\underline{\sim>}(v_2|d_2) = (v_1\sim>v_2|d_1\&d_1+\sim v_1\&d_1+v_2\&d_2), \tag{3.17}$$
$$(v_1|d_1)\underline{<\sim}(v_2|d_2) = (v_1<\sim v_2|d_1\&d_1+v_1\&d_1+\sim v_2\&d_2), \tag{3.18}$$
$$(v_1|d_1)\underline{\oplus}(v_2|d_2) = (v_1\oplus v_2|d_1\&d_1), \tag{3.19}$$
$$(v_1|d_1)\underline{\equiv}(v_2|d_2) = (v_1\equiv v_2|d_1\&d_1), \tag{3.20}$$

A proof of equivalence (3.11) is presented in Table 3.5 where two last columns are identical. The values of pairs $(v_1|d_1)$, $(v_2|d_2)$, and $(v_1\&v_2|d_1\&d_1+\sim v_1\&d_1+\sim v_2\&d_2)$ are computed through (3.1), and the values of expression $(v_1|d_1)\underline{\&}(v_2|d_2)$ are computed through using row 1 of Table 3.2. It is easy to see, the proof of other equivalences is performed in the similar way.

Proof of equivalence (3.11) Table 3.5

v_1	d_1	v_2	d_2	$(v_1\|d_1)$	$(v_2\|d_2)$	$v_1\&v_2$	$d_1\&d_1+\sim v_1\&d_1+\sim v_2\&d_2$	$(v_1\|d_1)\underline{\&}(v_2\|d_2)$	$(v_1\&v_2\|d_1\&d_1+\sim v_1\&d_1+\sim v_2\&d_2)$
0	0	0	0	–	–	0	0	–	–
0	0	0	1	–	0	0	1	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	1	0	1	0	0	0	1	0	0
0	1	1	0	0	–	0	1	0	0
0	1	1	1	0	1	0	1	0	0
1	0	0	0	–	–	0	0	–	–
1	0	0	1	–	0	0	1	0	0
1	0	1	0	–	–	1	0	–	–
1	0	1	1	–	1	1	0	–	–
1	1	0	0	1	–	0	0	–	–
1	1	0	1	1	0	0	1	0	0
1	1	1	0	1	–	1	0	–	–
1	1	1	1	1	1	1	1	1	1

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

$$(v_1|d_1)\underline{\&}(v_2|d_2) = (v_1\&v_2|(d_1+d_1)\&(\sim v_2+d_1)\&(\sim v_1+d_2)) \tag{3.21}$$
$$(v_1|d_1)\underline{+}(v_2|d_2) = (v_1+v_2|(d_1+d_1)\&(v_2+d_1)\&(v_1+d_2)) \tag{3.22}$$
$$(v_1|d_1)\underline{/}(v_2|d_2) = (v_1/v_2|(d_1+d_1)\&(\sim v_2+d_1)\&(\sim v_1+d_2)) \tag{3.23}$$
$$(v_1|d_1)\underline{\downarrow}(v_2|d_2) = (v_1\downarrow v_2|(d_1+d_1)\&(v_2+d_1)\&(v_1+d_2)) \tag{3.24}$$
$$(v_1|d_1)\underline{\rightarrow}\&(v_2|d_2)= (v_1\rightarrow v_2|(d_1+d_1)\&(v_2+d_1)\&(\sim v_1+d_2)) \tag{3.25}$$
$$(v_1|d_1)\underline{\leftarrow}\&(v_2|d_2)= (v_1\leftarrow v_2|(d_1+d_1)\&(\sim v_2+d_1)\&(v_1+d_2)) \tag{3.26}$$
$$(v_1|d_1)\underline{\sim>}(v_2|d_2) = (v_1\sim>v_2|(d_1+d_1)\&(v_2+d_1)\&(\sim v_1+d_2)) \tag{3.27}$$
$$(v_1|d_1)\underline{<\sim}(v_2|d_2) = (v_1<\sim v_2|(d_1+d_1)\&(\sim v_2+d_1)\&(v_1+d_2)). \tag{3.28}$$

If $d_1=d_2=1$ then the partial operations become completely specified. In particular, the partial conjunction $(v_1|1)\underline{\&}(v_2|1)$ is transformed to the total traditional conjunction $v_1\&v2$ in the following way: $(v_1|1)\underline{\&}(v_2|1) = (v_1\&v_2|1+\sim v_1\&1+\sim v_2\&1) = (v_1\&v_2|1) = v_1\&v_2$.

3.4.2 *Generalization of traditional logic laws*

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

$$\underline{\sim\sim}(v|d) = (v|d) \tag{3.29}$$
$$(v_1|d_1)\underline{\&}(v_2|d_2) = (v_2|d_2)\underline{\&}(v_1|d_1) \tag{3.30}$$
$$(v_1|d_1)\underline{\&}((v_2|d_2)\underline{\&}(v_3|d_3)) = ((v_1|d_1)\underline{\&}(v_2|d_2))\underline{\&}(v_3|d_3) \tag{3.31}$$
$$(v_1|d_1)\underline{+}(v_2|d_2) = (v_2|d_2)\underline{+}(v_1|d_1) \tag{3.32}$$
$$(v_1|d_1)\underline{+}((v_2|d_2)\underline{+}(v_3|d_3)) = ((v_1|d_1)\underline{+}(v_2|d_2))\underline{+}(v_3|d_3) \tag{3.33}$$
$$(v_1|d_1)\underline{\&}((v_2|d_2)\underline{+}(v_3|d_3)) = ((v_1|d_1)\underline{\&}(v_2|d_2))\underline{+}((v_1|d_1)\underline{\&}(v_3|d_3)) \tag{3.34}$$
$$(v_1|d_1)\underline{+}((v_2|d_2)\underline{\&}(v_3|d_3)) = ((v_1|d_1)\underline{+}(v_2|d_2))\underline{\&}((v_1|d_1)\underline{+}(v_3|d_3)) \tag{3.35}$$
$$\underline{\sim}((v_1|d_1)\underline{\&}(v_2|d_2)) = \underline{\sim}(v_1|d_1)\underline{+\sim}(v_2|d_2) \tag{3.36}$$
$$\underline{\sim}((v_1|d_1)\underline{+}(v_2|d_2)) = \underline{\sim}(v_1|d_1)\underline{\&\sim}(v_2|d_2) \tag{3.37}$$
$$(v_1|d_1)\underline{\&}((v_1|d_1)\underline{+}(v_2|d_2)) = (v_1|d_1) \tag{3.38}$$
$$(v_1|d_1)\underline{+}((v_1|d_1)\underline{\&}(v_2|d_2)) = (v_1|d_1) \tag{3.39}$$
$$(v|d)\underline{\&}(v|d) = (v|d) \tag{3.40}$$
$$(v|d)\underline{+}(v|d) = (v|d) \tag{3.41}$$
$$(v|d)\underline{\&\sim}(v|d) = (0|d) \tag{3.42}$$
$$(v|d)\underline{+\sim}(v|d) = (1|d) \tag{3.43}$$
$$(v|d)\underline{\&}(1|1) = (v|d) \tag{3.44}$$
$$(v|d)\underline{+}(0|1) = (v|d) \tag{3.45}$$
$$(v|d)\underline{\&}(0|1) = (0|1) \tag{3.46}$$
$$(v|d)\underline{+}(1|1) = (1|1) \tag{3.47}$$
$$(0|1)\underline{\rightarrow}(v|d) = (0|1) \tag{3.48}$$
$$(v|d)\underline{\rightarrow}(v|d) = (1|d) \tag{3.49}$$
$$(v|d)\underline{\rightarrow}(1|1) = (1|1) \tag{3.50}$$
$$(v_1|d_1)\underline{\rightarrow}(v_2|d_2) = \underline{\sim}(v_1|d_1)\underline{+}(v_2|d_2) \tag{3.51}$$
$$(v_1|d_1)\underline{\rightarrow}(v_2|d_2) = \underline{\sim}(v_2|d_2)\underline{\rightarrow\sim}(v_1|d_1). \tag{3.52}$$

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 *absorption* laws are represented by equalities (3.38) to (3.50). Laws (3.51) and (3.52) allow for transformation of partial implication.

3.4.3 Novel laws in partial logic

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

$$(v_1|d)\underline{\&}(v_2|d) = (v_1\&v_2|d) \tag{3.53}$$
$$(v_1|d)\underline{+}(v_2|d) = (v_1+v_2|d) \tag{3.54}$$
$$(v|d_1)\underline{\&}(v|d_2) = (v|d_1\&d_2+\sim v\&(d_1+d_2)) \tag{3.55}$$
$$(v|d_1)\underline{+}(v|d_2) = (v|d_1\&d_2+v\&(d_1+d_2)) \tag{3.56}$$
$$(v|v) = (1|v) \tag{3.57}$$
$$(\sim v|v) = (0|v) \tag{3.58}$$
$$(v\&d|d) = (v|d) \tag{3.59}$$
$$(v+d|d) = (1|d) \tag{3.60}$$
$$(v\&\sim d|d) = (0|d) \tag{3.61}$$
$$(v+\sim d|d) = (v|d) \tag{3.62}$$
$$(v|v\&d) = (1|v\&d) \tag{3.63}$$
$$(v|\sim v\&d) = (0|\sim v\&d) \tag{3.64}$$
$$(v|v\&d) = (v|d)\underline{+}(v|0) \tag{3.65}$$
$$(v|\sim v\&d) = (v|d)\underline{\&}(v|0) \tag{3.66}$$
$$(v|v+d) = (v|1)\underline{+}(0|d) \tag{3.67}$$
$$(v|\sim v+d) = (v|1)\underline{\&}(1|d) \tag{3.68}$$
$$(\sim v|v+d) = (\sim v|1)\underline{\&}(1|d) \tag{3.69}$$
$$(\sim v|\sim v+d) = (\sim v|1)\underline{+}(0|d) \tag{3.70}$$
$$(f|x_i) = (f(x_i=1)|x_i) \tag{3.71}$$
$$(f|\sim x_i) = (f(x_i=0)|x_i) \tag{3.72}$$
$$(f(v)|v\oplus d) = (f(\sim d)|v\oplus d) \tag{3.73}$$
$$(f(v)|v\equiv d) = (f(d)|v\equiv d). \tag{3.74}$$

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 to a form that can be efficiently used for logic inference in the partial logic.

4.1 SYNTAX

The *alphabet* of our language consists of:

1) delimiters ‘,’, ‘(’, ‘)’, ‘|’,

2) variables of a set *V*,

3) total logical operators $\sim$, $\&$, $+$, $->$, $\oplus$, and others,

4) partial logical operators $\underline{\sim}$, $\underline{\&}$, $\underline{+}$, $\underline{\rightarrow}$, $\underline{\oplus}$, and others,

5) functional symbols $f_1^{n1},...,f_i^{ni},...$ of degree $n_i \geq 0;$ a functional symbol f^0 of degree $n=0$ is an individual constant,

6) predicate symbols $P_1^{n1},...,P_i^{ni},...$ 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 $\underline{\forall}$, and partial existential quantifier $\underline{\exists}$.

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 $f_i^n(t_1,...,t_n)$ followed by n terms is a term.

Atomic formulas:

1) a proposition variable P^0 is an atomic formula,

2) a predicate symbol $P_i^n(t_1,...t_n)$ followed by n terms is an atomic formula.

Literals:

1) an atomic formula is a literal,

2) if A is an atomic formula then $\sim A$ is a literal.

A literal containing no variables is called a ground literal. Two literals A and $\sim A$ constitute a complementary pair.

Total formulas:

1) an atomic formula is a total formula,

2) if F is a total formula then $(\sim F)$ 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 $(\underline{\sim}R)$ is a partial formula,

3) if R and Q are partial formulas then $(R\underline{\&}Q)$, $(R\underline{+}Q)$, $(R\underline{\rightarrow}Q)$, $(R\underline{\oplus}Q)$, ... are partial formulas,

4) if R is a partial formula and x is a variable, then $(\underline{\forall}x)R$ and $(\underline{\exists}x)R$ are partial formulas.

4.2 SEMANTICS

The semantics of 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 $\sim$, $\&$, $+$, $\rightarrow$, 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 semantics of the universal and existential quantifiers $(\underline{\forall}x)R$ and $(\underline{\exists}x)R$ 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.

Truth function for pair $(F|G)$ Table 4.1

F	G	(F|G)
0	0	-
0	1	0
1	0	-
1	1	1

Functions for $(\underline{\forall}x)R$ and $(\underline{\exists}x)R$ Table 4.2

The set of values of R	$(\underline{\forall}x)R$	$(\underline{\exists}x)R$
{0}	0	0
{1}	1	1
{-}	-	-
{0,1}	0	1
{0,-}	0	-
{1,-}	-	1
{0,1,-}	0	1

Let a set D be the domain. We assign a function $D^n\rightarrow D$ to each n-place functional symbol f_i^n, and assign a function $D^n\rightarrow B$ to each n-place predicate symbol P_i^n. 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:

$$(F|G). \tag{4.1}$$

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 the steps as follows:

- 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:

$$(\underline{\forall}x)((\underline{\forall}y)(\sim Q(c,x)\rightarrow R(a,y) \mid Q(y,b))\ \underline{\&}\ (\underline{\exists}z)\underline{\sim}(\underline{\exists}y)(R(y,z)+P(x) \mid P(b))), \tag{4.2}$$

where x, y, and z are variables, a, d, and c are individual constants, and P, Q, and R are predicate symbols.

Step 1. Eliminating the total logical operators except ~, &, 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:

$$(\underline{\forall}x)((\underline{\forall}y)(Q(c,x)+R(a,y) \mid Q(y,b))\ \underline{\&}\ (\underline{\exists}z)\underline{\sim}(\underline{\exists}y)(R(y,z)+P(x) \mid P(b))). \tag{4.3}$$

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:

$$(\underline{\forall}x)((\underline{\forall}y)(Q(c,x)+R(a,y) \mid Q(y,b))\ \underline{\&}\ (\underline{\exists}z)\underline{\sim}(\underline{\exists}v)(R(v,z)+P(x) \mid P(b))). \tag{4.4}$$

Step 3. Reducing the scope of the total ~ and partial $\underline{\sim}$ 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:

$$\underline{\sim}(F|G) = (\sim F|G) \tag{4.5}$$
$$\underline{\sim}(\underline{\forall}x)(F|G) = (\underline{\exists}x)\underline{\sim}(F|G) \tag{4.6}$$
$$\underline{\sim}(\underline{\exists}x)(F|G) = (\underline{\forall}x)\underline{\sim}(F|G). \tag{4.7}$$

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)).

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:

Proof of equivalences (4.6) and (4.7) Table 4.3

$(F\|G)$	$\underline{\sim}(F\|G)$	$(\underline{\exists}x)(F\|G)$	$\underline{\sim}(\underline{\exists}x)(F\|G)$	$(\underline{\exists}x)\underline{\sim}(F\|G)$	$(\underline{\forall}x)(F\|G)$	$\underline{\sim}(\underline{\forall}x)(F\|G)$	$(\underline{\forall}x)\underline{\sim}(F\|G)$
{0}	{1}	0	1	1	0	1	1
{1}	{0}	1	0	0	1	0	0
{-}	{-}	-	-	-	-	-	-
{0,1}	{0,1}	1	0	1	0	1	0
{0,-}	{1,-}	-	-	1	0	1	-
{1,-}	{0,-}	1	0	-	-	-	0
{0,1,-}	{0,1,-}	1	0	1	0	1	0

$$(\underline{\forall}x)((\underline{\forall}y)(Q(c,x)+R(a,y)\mid Q(y,b))\ \underline{\&}\ (\underline{\exists}z)(\underline{\forall}v)(\sim R(v,z)\&\sim P(x)\mid P(b))). \tag{4.8}$$

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 $(\underline{\exists}z)$ is located after the universal quantifier $(\underline{\forall}x)$. Therefore, the variable *z* is replaced with a function symbol *f(x)*:

$$(\underline{\forall}x)((\underline{\forall}y)(Q(c,x)+R(a,y)\mid Q(y,b))\ \underline{\&}\ (\underline{\forall}v)(\sim R(v,f(x))\&\sim P(x)\mid P(b))). \tag{4.9}$$

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

$$((\forall x)F|G) = (\underline{\forall}x)(F|G) \tag{4.10}$$
$$((\underline{\forall}x)R)\underline{\&}Q = (\underline{\forall}x)(R\underline{\&}Q) \tag{4.11}$$
$$((\underline{\forall}x)R)\underline{+}Q = (\underline{\forall}x)(R\underline{+}Q). \tag{4.12}$$

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 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:

$$(F|(\forall x)G) = (\underline{\forall}x)(F|G). \tag{4.13}$$

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):

$$(\underline{\forall}x)(\underline{\forall}y)(\underline{\forall}v)((Q(c,x)+R(a,y)\mid Q(y,b))\ \underline{\&}\ (\sim R(v,f(x))\&\sim P(x)\mid P(b))) \tag{4.14}$$

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

$$(Q(c,x)+R(a,y)\mid Q(y,b))\ \underline{\&}\ (\sim R(v,f(x))\&\sim P(x)\mid P(b))). \tag{4.15}$$

Proof of equivalence (4.10) Table 4.4

F	G	$(\forall x)F$	$((\forall x)F\|G)$	$(F\|G)$	$(\underline{\forall}x)(F\|G)$
{0}	0	0	-	{-}	-
{1}	0	1	-	{-}	-
{0,1}	0	0	-	{-}	-
{0}	1	0	0	{0}	0
{1}	1	1	1	{1}	1
{0,1}	1	0	0	{0,1}	0

Proof of equivalences (4.11) and (4.12) Table 4.5

R	Q	$(\underline{\forall}x)R$	$((\underline{\forall}x)R)\underline{\&}Q$	$R\underline{\&}Q$	$(\underline{\forall}x)(R\underline{\&}Q)$	$((\underline{\forall}x)R)\underline{+}Q$	$R\underline{+}Q$	$(\underline{\forall}x)(R\underline{+}Q)$
{0}	0	0	0	{0}	0	0	{0}	0
{1}	0	1	0	{0}	0	1	{1}	1
{-}	0	-	0	{0}	0	-	{-}	-
{0,1}	0	0	0	{0}	0	0	{0,1}	0
{0,-}	0	0	0	{0}	0	0	{0,-}	0
{1,-}	0	-	0	{0}	0	-	{1,-}	-
{0,1,-}	0	0	0	{0}	0	0	{0,1,-}	0
{0}	1	0	0	{0}	0	1	{1}	1
{1}	1	1	1	{1}	1	1	{1}	1
{-}	1	-	-	{-}	-	1	{1}	1
{0,1}	1	0	0	{0,1}	0	1	{1}	1
{0,-}	1	0	0	{0,-}	0	1	{1}	1
{1,-}	1	-	-	{1,-}	-	1	{1}	1
{0,1,-}	1	0	0	{0,1,-}	0	1	{1}	1
{0}	-	0	0	{0}	0	-	{-}	-
{1}	-	1	-	{-}	-	1	{1}	1
{-}	-	-	-	{-}	-	-	{-}	-
{0,1}	-	0	0	{0,-}	0	-	{1,-}	-
{0,-}	-	0	0	{0,-}	0	-	{-}	-
{1,-}	-	-	-	{-}	-	-	{1,-}	-
{0,1,-}	-	0	0	{0,-}	0	-	{1,-}	-

Proof of equality (4.13) Table 4.6

N	G	F	$(\forall x)G$	$(F\mid(\forall x)G)$	$(F\mid G)$	$(\underline{\forall}x)(F\mid G)$
1	{0}	0	0	-	{-}	-
2	{1}	0	1	0	{0}	0
3	{0,1}	0	0	-	{0,-}	0
4	{0}	1	0	-	{-}	-
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)) & ~R(v,f(x))&~P(x) | (4.17)
(Q(y,b)+P(b))&(Q(y,b)+R(v,f(x))+P(x))&(P(b)+~Q(c,x))&(P(b)+~R(a,y)))).

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

4.4 KNOWLEDGE REPRESENTATION IN INCOMPLETELY SPECIFIED DOMAIN

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

$$Knowledge=C_1\ \underline{\&}\dots\underline{\&}\ C_m, \tag{4.18}$$

$$C_i = (\underline{\forall}x_1)\dots(\underline{\forall}x_n)\ (P_{i,1}\ \underline{+}\dots\ \underline{+}\ P_{i,m}), \tag{4.19}$$

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. After that the partial clause

$$(L_1|D_1)\underline{+}\dots\underline{+}(L_k|D_k)\underline{+}(\sim L_{k+1}|D_{k+1})\underline{+}\quad\underline{+}(\sim L_m|D_m) \tag{4.20}$$

contains k positive and m-k negative literals in the value part of pairs. 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:

$$(L|D)\underline{+}(\sim L_1|D_1)\underline{+}\quad\underline{+}(\sim L_m|D_m) \tag{4.21}$$

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:

$$\begin{aligned}&(L_1+\dots+L_k+\sim L_{k+1}+\dots+\sim L_m\ |\\ &\qquad D_1\&\dots\&D_m+L_1\ \&D_1+\dots+L_k\&D_k+\sim L_{k+1}\&D_{k+1}+\quad+\sim L_m\&D_m).\end{aligned} \tag{4.22}$$

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

$$\begin{aligned}&(L|D)\underline{+}(\sim L_1|D_1)\underline{+}\quad\underline{+}(\sim L_m|D_m) =\\ &(L|D)\underline{+}\ \underline{\sim}((L_1|D_1)\underline{\&}\quad\underline{\&}(L_m|D_m)) =\\ &(L|D)\underline{\leftarrow}((L_1|D_1)\underline{\&}\quad\underline{\&}(L_m|D_m)).\end{aligned} \tag{4.23}$$

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 $\sim L$.

5.1 GOALS AND THEOREMS

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

$$C_1\ \underline{\&}\ ...\ \underline{\&}\ C_m\ \underline{\rightarrow}\ R. \tag{5.1}$$

The formula R represent a new piece of knowledge that has to be inferred from $C_1,...,C_m$. In order to proof 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 formula (5.1) be satisfiable we use a refutation procedure and prove the formula as follows be unsatisfiabile:

$$C_1\ \underline{\&}\ ...\ \underline{\&}\ C_m\ \underline{\&}\ \underline{\sim}\ R. \tag{5.2}$$

Based on the resolution principle and starting with the clauses C_1, … C_m, and $\underline{\sim}R$.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 for the ground [8] partial clauses. Given two partial clauses

$$C_1 = (L_1|D_1)\underline{+}(L_2|D_2) \tag{5.3}$$

and

$$C_2 = (\sim L_1|D_3)\underline{+}(L_3|D_4) \tag{5.4}$$

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

$$C = (0|D_1\&D_3+\sim L_1\&D_1+L_1\&D_3)\underline{+}(L_2|D_2)\underline{+}(L_3|D_4). \tag{5.5}$$

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 $C_1\underline{\&}C_2$ is satisfied iff $C_1\underline{\&}C_2\underline{\&}C$ is satisfied. The partial conjunction $C_1\underline{\&}C_2$ of two clauses is transformed as follows:

$$\begin{aligned}
&C_1\underline{\&}C_2 = ((L_1|D_1)\underline{+}(L_2|D_2))\ \underline{\&}\ ((\sim L_1|D_3)\underline{+}(L_3|D_4)) = \\
&(L_1+L_2|D_1\&D_2+L_1\&D_1+L_2\&D_2)\ \underline{\&}\ (\sim L_1+L_3|D_3\&D_4+\sim L_1\&D_3+L_3\&D_4) = \\
&((L_1+L_2)\&(\sim L_1+L_3)\ |\ (D_1\&D_2+L_1\&D_1+L_2\&D_2)\&(D_3\&D_4+\sim L_1\&D_3+L_3\&D_4) + D_1\&D_2\&\sim L_1\&\sim L_2 + \\
&D_3\&D_4\&L_1\&\sim L_3) = \\
&((L_1+L_2)\&(\sim L_1+L_3)\ |\ D_1\&D_2\&D_3\&D_4 + D_1\&D_2\&D_3{\sim}\&L_1 + \\
&\qquad D_1\&D_2\&D_4\&L_3 + D_1\&D_2\&\sim L_1\&\sim L_2 + \\
&\qquad D_1\&D_3\&D_4\&L_1 + D_1\&D_4\&L_1\&L_3 + \\
&\qquad D_2\&D_3\&D_4\&L_2 + D_2\&D_3{\sim}\&L_1\&L_2 + \\
&\qquad D_2\&D_4\&L_2\&L_3 + D_3\&D_4\&L_1\&\sim L_3).
\end{aligned} \tag{5.6}$$

The resolvent C can be represented as:

$$\begin{aligned}
C &= (0|D_1\&D_3+\sim L_1\&D_1+L_1\&D_3)\underline{+}(L_2|D_2)\underline{+}(L_3|D_4) = \\
&\quad (L_2 + L_3\ |\ (D_1\&D_3+\sim L_1\&D_1+L_1\&D_3)\&D_2\&D_4 + (L_2\&D_2) + (L_3\&D_4)) = \\
&\quad (L_2 + L_3\ |\ D_1\&D_2\&D_3\&D_4 + D_1\&D_2\&D_4\&\sim L_1 + \\
&\qquad\quad D_2\&D_3\&D_4\&L_1 + L_2\&D_2 + L_3\&D_4).
\end{aligned} \tag{5.7}$$

Performing the partial conjunction operation on the operand $C_1\underline{\&}C_2$ presented by (5.6) and on the operand C presented by (5.7) we obtain the following partial formula:

$$\begin{aligned}
C_1\underline{\&}C_2\underline{\&}C = \ &((L_1+L_2)\&(\sim L_1+L_3)\&(L_2+L_3)\ | \\
&D_1\&D_2\&D_3\&D_4 + D_1\&D_2\&D_3{\sim}\&L_1 + D_1\&D_2\&D_4\&L_3 + \\
&D_1\&D_2\&\sim L_1\&\sim L_2 + D_1\&D_3\&D_4\&L_1 + D_1\&D_4\&L_1\&L_3 + \\
&D_2\&D_3\&D_4\&L_2 + D_2\&D_3{\sim}\&L_1\&L_2 + D_2\&D_4\&L_2\&L_3 + D_3\&D_4\&L_1\&\sim L_3).
\end{aligned} \tag{5.8}$$

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

$$(A+B|\sim C)\ \underline{\&}\ (\sim A|B)\ \underline{\&}\ (C+D|A)\ \underline{\&}\ (\sim D|C), \tag{5.9}$$

we are going to prove that the formula is not satisfiable. The formula is broken down into the set of partial clauses as follows:

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

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

5. $(B \mid B\&\sim C)$	- resolvent for clauses 1 and 2
6. $(1 \mid B\&\sim C) = (\sim C \mid B\&\sim C)$	- from clause 5
7. $(D \mid A\&B\&\sim C+A\&D)=(D \mid A\&(B+D)\&(\sim C+D))$	- resolvent for clauses 3 and 6
8. $(0 \mid A\&D\&C)$	- 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

$$C_1 = (L_1 + L_2 \mid D_1) \tag{5.11}$$

and

$$C_2 = (\sim L_1 + L_3 \mid D_2) \tag{5.12}$$

then the resolvent derived from the clauses is as follows:

$$C = (L_2+L_3 \mid D_1\&D_2+L_2\&D_1+L_3\&D_2). \tag{5.13}$$

To prove (5.13) we transform clauses (5.11) and (5.12) in the following way: $C_1{}^{'}=(L_1|D_1)\underline{+}\ (L_2|D_1)$ and $C_2{}''=(\sim L_1|D_2)\ \underline{+}\ (L_3|D_2)$. Based on expression (5.5), the following resolvent is constructed from the clauses:

$$C = (0|D_1\&D_2+\sim L_1\&D_1+L_1\&D_2)\ \underline{+}\ (L_2|D_1)\ \underline{+}\ (L_3|D_2) = \\ (L_2 + L_3 \mid D_1\&D_2+L_2\&D_1+L_3\&D_2). \tag{5.14}$$

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:

$$(0\leftarrow L_1\&L_2|D_1). \tag{5.15}$$

The clause that is used for transforming the current goal to the next-step goal is also represented in the consequence form as:

$$(L_1\leftarrow L_3|D_2). \tag{5.16}$$

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:

$$(\sim L_2+\sim L_3| D_1\&Q+\sim L_2\& D_1+\sim L_3\& D_2) \tag{5.17}$$

that is translated to the next-step goal:

$$(0\leftarrow L_2\& L_3| D_1\& D_2+\sim L_2\& D_1+\sim L_3\& D_2). \tag{5.18}$$

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

$$(L_1| D_2) \tag{5.19}$$

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

$$(0\leftarrow L_2 \mid D_1\&(D_2+\sim L_2)). \tag{5.20}$$

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

$$F\&G \subseteq Value\_Part \subseteq F+\sim G. \qquad (6.1)$$

The new pair is as follows:

$$(Value\_Part \mid G). \qquad (6.2)$$

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:

$$(Value\_Part \mid 1). \qquad (6.3)$$

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

$$C_1 \ \& \ldots \& \ C_m. \qquad (6.4)$$

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 $F\&G \subset Value\_Part \subset F+\sim G$. 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:

$$(A+B|1)\underline{\&}(\sim A|1)\underline{\&}(C+D|1)\underline{\&}(\sim D|1) \ = \ ((A+B)\&(\sim A)\&(C+D)\&(\sim D) \mid 1) =$$
$$(A+B)\&(\sim A)\&(C+D)\&(\sim D). \qquad (6.5)$$

The following set of four total clauses is derived from (6.5):

1. A+B
2. ~A (6.6)
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:

$$(A+B+C|\sim C)\underline{\&}(\sim A+\sim B|B)\underline{\&}(C+D+\sim A|A)\underline{\&}(\sim D+\sim C|C) =$$
$$(A+B+C|\ 1)\underline{\&}(\sim A+\sim B|\ 1)\underline{\&}(C+D+\sim A|\ 1)\underline{\&}(\sim D+\sim C|\ 1) =$$
$$((A+B+C)\&(\sim A+\sim B)\&(C+D+\sim A)\&(\sim D+\sim C) \mid 1) =$$
$$(A+B+C)\&(\sim A+\sim B)\&(C+D+\sim A)\&(\sim D+\sim C). \qquad (6.7)$$

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

1. A+B+C
2. ~A+~B (6.8)

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

$$\begin{aligned}
&((A+B)\&(\sim C)|\sim C)\underline{\&}((\sim A)\&(B)|B)\underline{\&}((C+D)\&(A)|A)\underline{\&}((\sim D)\&(C)|C) = \\
&((A+B)\&(\sim C)|\ 1)\underline{\&}((\sim A)\&(B)|\ 1)\underline{\&}((C+D)\&(A)|\ 1)\underline{\&}((\sim D)\&(C)|\ 1) = \\
&((A+B)\&(\sim C)\&(\sim A)\&(B)\&(C+D)\&(A)\&(\sim D)\&(C)\ |\ 1) = \\
&(A+B)\&(\sim C)\&(\sim A)\&(B)\&(C+D)\&(A)\&(\sim D)\&(C).
\end{aligned} \tag{6.9}$$

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

1. A+B
2. ~C
3. ~A
4. B (6.10)
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

Now we consider the following rule in the form of partial consequence:

$$(A|1)\ \underline{\leftarrow}\ (C|Q). \tag{6.11}$$

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: *false, true* and *don’t care*. The *don’t care* value can be replaced with *false* or *true* arbitrarily. As a result, one of the completely specified truth functions belonging to the range *C&Q* to *C+~Q* can be obtained. In order to generate all the possible solutions for a given goal, we select the function *C+~Q*. Formula (6.11) is transformed to the following expression:

$$A \leftarrow (C + \sim Q). \tag{6.12}$$

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

$$(A \leftarrow \sim Q)\ \&\ (A \leftarrow C). \tag{6.13}$$

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) model is widely used for modelling and synthesis of digital sequential circuits and systems. The FSM is a five-tuple

$$FSM = (X,Y,S,\varphi,\delta), \tag{7.1}$$

where X is a set of input symbols; Y is a set of output symbols; S is a set of internal states; φ: $S{\times}X{\to}S$ is a next-state function; δ: $S{\times}X{\to}Y$ 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 *don't care* or '–' 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*.

FSM next-state and output functions Table 7.1

State	Input symbol			
	x1	x2	x3	x4
s0	– / y1	s0 / y3	s4 / y2	s1 / y2
s1	s2 / y3	s0 / –	s3 / –	– / y1
s2	– / y2	– / y2	s1 / –	s2 / y4
s3	s1 / –	s4 / –	– / y4	– / y3
s4	– / y4	s3 / y1	s2 / –	s3 / –

7.2 REPRESENTATION OF KNOWLEDGE ABOUT FSM

The *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 a FSM representation in the partial logic requires a less number of clauses.

The input symbols *x1, x2, x3, x4,* the output symbols *y1, y2, y3, y4,* and the internal states *s0, s1, s2, s3, s4* 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 on the FSM, a set of predicates is introduced. Among them there are the predicates as follows: *next_state(s,x,sn), state_specified(s,x,sn), state_dontcare(s,x,sn), output(s, x,y), output_specified(s,x,y),* and *output_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:

(next_state(s,x,sn) | 1) if (state_specified(s,x,sn) | not state_dontcare(s,x,sn)).

According to (6.12) and (6.13) the clause is split to following two-valued clauses:

next_state(s,x,sn) if state_dontcare(s,x,sn).
next_state(s,x,sn) if state_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

mapping(s0,Xl,[y3,y1,y2,y3,y2,y4]) and *write*("Xl=",Xl), described in Fig.7.1 denotes the task consisting in searching for an input symbol sequence *Xl*, given the output symbol sequence *[y3,y1,y2,y3,y2,y4]*.

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
    state=s0;s1;s2;s3;s4
    input_symbol=x1;x2;x3;x4
    output_symbol=y1;y2;y3;y4
    sequence_of_states=state*
    sequence_of_inputs=input_symbol*
    sequence_of_outputs=output_symbol*
predicates
    mapping(state,sequence_of_inputs,
                sequence_of_outputs)
    next_state(state,input_symbol,state)
    state_specified(state,input_symbol,state)
    state_dontcare(state,input_symbol,state)
    sx_y_dc(state,input_symbol)
    each_of_y(output_symbol)
    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).
    output_specified(s4,x1,y4).
    output_specified(s4,x2,y1).
    output_dontcare(S,X,Y) if sx_y_dc(S,X) and
                                each_of_y(Y).
    sx_y_dc(s1,x2).
    sx_y_dc(s1,x3).
    sx_y_dc(s2,x3).
    sx_y_dc(s3,x1).
    sx_y_dc(s3,x2).
    sx_y_dc(s4,x3).
    sx_y_dc(s4,x4).
    each_of_y(y1).
    each_of_y(y2).
    each_of_y(y3).
    each_of_y(y4).
    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. Conclusions

In this paper we have proposed the formalism for modelling the world in which an assertion is essential in one situation and is not essential in other situation. The formalism is the partial proposition and first order predicate logics developed through using three truth-values: *false, true,* and *don't care*. The difference between the partial logics and the traditional three-valued logics consists in that the *don't care* value is replaceable with other values.

Due to encoding the three-valued partial variables with a pair of two-valued Boolean variables, a knowledge is represented as a set of partial clauses constructed on a set of two-valued predicates through using partial operations and quantifiers. In order to perform deductive reasoning in an incompletely specified domain we generalized the Robinson's resolution principle to infer a partial resolvent from partial clauses. The inference rule supports manipulation of clause domains and simplification of clauses during reasoning.

Two approaches are possible for the implementation of the partial logics. The first one consists in development of special software handling partial clauses and performing partial reasoning. The second one is based on transition from the partial logics to the traditional two-valued logics. Three method of transition are proposed in the paper. One of these is the most preferable and speeds up logic inference. Knowledge representation in an incompletely specified domain is illustrated with the incompletely specified finite state machine. The example proves reduction in the number of clauses when the proposed methods have been used.

9. 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.

References

1. Damiani M. and de Micheli G. Don't Care Set Specifications in Combinational and Synchronous Logic Circuits, IEEE Trans. On CAD, 12, 1993, pp.365-388.

2. Filman R. Reasoning with Worlds and Truth Maintenance in a Knowledge-Based Programming Environment. Communications of the ACM, 31(4), 1988, pp.382-401.

3. Pliuskevicius R. *The Saturated Tableaux for Linear Miniscoped Horn-like Temporal Logic*, Journal of Automated Reasoning, 13, 1994, pp.391-407.

4. Prihozhy A.A. *Methods of partial logic*. Proc. Int. Conf. on Neuro Computers and Artificial Intelligence, Brest, 1999.

5. Prihozhy A.A. *Logic Inference in Partial Logic*. Intelligent Systems, Minsk, Belarus, 1998, pp.123-140.

6. Prihozhy A.A. *Algebra of Partial Logic*. New Technologies for Computers and Mechanics, Brest, Belarus, 1998, pp.168-174.

7. Prihozhy A.A. *If-Diagrams: Theory and Application*, PATMOS'97: Proc. European Workshop, Belgium, 1997, pp.369-378.

8. Robinson J.A. *A Machine-Oriented Logic Based on the Resolution Principle*. Journal of the Association for Computing Machinary, 12(1), 1965, pp.23-41.

9. Sakalauskaite J. *Ordered Resolution for Finitely-Valued First-Order Logics*, Preprintas Nr.95-3, Vilnius, Lietuva, 1995, 10 p.

10. Zadeh L.A. *The Role of Fuzzy Logic in the Management of Uncertainty in Expert Systems. Approximate Reasoning in Expert Systems*. Elsevier Science Publishers, North Holland, 1985, pp.3-41.

11. Zavadskas E., Peldschus F., Kaklauskas A. *Multiple Criteria Evaluation of Projects in Construction*. Vilnius, Technika, 1994.