Constraint Programming for Constructive Abduction. A Case Study in Diagnostic Model-Based Reasoning

Antoni Ligęza(✉)

AGH University of Science and Technology, Kraków, Poland
ligeza@agh.edu.pl

Abstract. Diagnostic reasoning is often based on abduction. Abductive inference consists in generation of hypotheses which explain the current behavior of the system under investigation. Such a reasoning is based on accessible background knowledge and the results must be consistent with all auxiliary observations. Efficient abductive diagnosis is carried out as Model-Based Reasoning. The knowledge about the model defines the search-space for diagnostic hypotheses. Unfortunately, use of classical consistency-based reasoning leads to rough, qualitative results only, even if good knowledge of the correct model is available. In this paper and attempt to use Constraint Programming as a tool for diagnostic reasoning is presented. The ultimate goal is to provide more precise diagnoses. Two case studies, one concerning fault parameter evaluation, and the second concerning structural fault localization are presented.

Keywords: Model-Based Reasoning · Automated diagnosis · Consistency-based reasoning · Constraint programming · Abduction · Constructive abduction · Parametric fault identification · Structural fault identification

1 Introduction

Model-Based Diagnostic Reasoning (MBDR) [3,6] requires formal definition of the model of correct work of a system under investigation. The search for diagnoses incorporates both Consistency-Based Diagnosis and abductive reasoning [3,6,7,16]. *Abduction* can be considered as one of principal ways of reasoning for problem solving. In fact, abductive reasoning consists in *search for hypotheses* which provide a satisfactory explanation of the observed faults. Simultaneously, the diagnostic hypotheses must be consistent with auxiliary observations. Note that accessible background knowledge must be taken into account.

Search for potential diagnoses can be performed with experience-based approaches based on *shallow knowledge* or with Model-Based Reasoning (MBR) [6] taking into account the *deep knowledge* of system components, structure and parameters. In this paper we follow the MBR approach where diagnostic

J.M. Kościelny et al. (eds.), *Advanced Solutions in Diagnostics and Fault Tolerant Control*, Advances in Intelligent Systems and Computing 635,
DOI 10.1007/978-3-319-64474-5_8

hypotheses are generated with help of Consistency-Based Reasoning (CBR) [16] and *abduction*. Unfortunately, use of this classical methods lead to binary evaluation of component faults only. Moreover, the number of admissible diagnostic hypotheses can be relatively large, especially in the case of many-element diagnoses. In more complex cases, where values of certain variables are to be found as well, simple abduction based on pure search (e.g. Backtracking Depth-First Search or search on AND-OR graphs) becomes inefficient; some attempt to use SAT-based methods has been reported in [5] and Constraint Programming in application in enhanced diagnosis in [15].

In this paper we attempt to employ Constraint Programming to generate more detailed diagnoses. The main goal is to put the abductive inference into a formal framework of *Constraint Programming* in order to enable the use of constraint propagation techniques and tools. The main aim is to make abduction more *informative*; we shall call this approach *Constructive Abduction*. Constructive Abduction means that (i) the generated hypotheses should be as precise as possible (e.g. numerical models of faults for further evaluation), and (ii) inconsistent hypotheses should be eliminated in a more efficient way [12]. Moreover, localization of structural faults should be as precise as possible.

Existing methods of diagnostic inference are diversified. There are algebraic, graph-based, and logical expert-like or model-based diagnostic approaches. Some of the popular models include extended diagnostic matrices [8,9], Consistency-Based Reasoning [6,10,16], logical causal graphs [10], and many other [3,7]. A recent survey of various approaches is given by [17]. Following classical CBR, this paper explores mainly the abductive approach and it is focused on employing Constraint Programming for developing *constructive abduction*, where purely logical abductive hypotheses are enriched with exact numerical solutions.

The main focus of this work consists in employing Constraint Programming in abductive diagnosis of technical systems. An analysis of applying Constraint Programming in modeling diagnostic reasoning is carried out. The ideas are illustrated with a diagnostic example of a multiplier-adder system. Two case studies are provided. The first one concerns finding more precise, numerical models of faults. The second one is concerned on precise localization of structural faults (e.g. a break in the circuit).

The paper is organized as follows. In Sect. 2 a simple motivation example from [16] is recalled. Section 3 covers some minimal information on Constraint Programming. Section 4 provides a note on abduction and a formal definition of diagnosis within the Consistency-Based Approach and Constraint Programming framework. Section 5 reports on an approach to precise numerical fault modeling. Finally, Sect. 6 presents a case study of Constraint Programming application to structural fault localization. The paper follows the ideas of the author [11,13] and is a further development of ideas first presented in [12].

2 Motivation Example

In this section we briefly recall a classical diagnostic example of a feed-forward arithmetic circuit. This is the multiplier-adder example presented in the seminal

paper by R. Reiter [16]. This example was further re-explored in numerous papers, including selected readings [6] and diagnostic handbook (Chapter [10]). It was further explored in the discussion carried out in domain literature concerning comparative analysis of diagnostic approaches [1,2,17]. Here we shall base on an in-depth analysis presented in [14], and also explored in [11,13].

The basic, intuitive schema of the system is presented in Fig. 1.

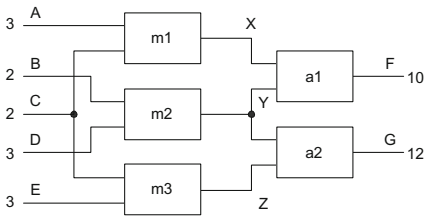

Fig. 1. The example multiplier-adder system.

The system is composed of five components and two layers. The first one contains three multipliers $m1$, $m2$, and $m3$, and receiving the input signals A, B, C, D and E. The second layer is composed of two adders, namely $a1$ and $a2$, producing the output values of F and G. Only inputs (of the first layer) and outputs of the system (of the second layer) are directly observable. The intermediate variables, namely X, Y and Z, are hidden and cannot be observed.

Observe that the current state of the system is defined by the input values; they are: $A = 3$, $B = 2$, $C = 2$, $D = 3$ and $E = 3$. It is easy to check — under the assumption of correct work of all the system elements — that the outputs should be $F = 12$ and $G = 12$. Note also that they *should be equal* to each other, which is due to the symmetry of the system and the symmetry of the input vales; this observation will be important for the analysis and we shall see later on why.

Now, since the current value of F is incorrect, namely $F = 10$, the system is faulty. At least one of its components must be faulty[1]. At this stage, for simplicity, we consider only *correct* components and *faulty* ones; no details about the type of fault are taken into consideration so far.

Through the Consistency-Based Reasoning [16] two *conflict sets* can be identified. These are: $DCF_1 = \{m1, m2, a1\}$ and $DCF_2 = \{m1, a1, a2, m3\}$. Conflict sets are sets of components, such that under the assumption of correct system model and the current fault manifestations, at least one element of such a set must be faulty.

Note that the types of DCF_1 and DCF_2 are different; this is due to their origin. DCF_1 is of *causal* type — all the elements directly influence the conflicting variable. On the other hand, DCF_2 is of *constraint* type; this is a kind of mathematical

[1] In Model-Based Diagnosis it is typically assumed that faulty behavior is caused by a fault of a named component or a simultaneous fault of a set of such components; no faults caused by faulty links, parameter setting or the internal structure are considered.

constraint which must be satisfied, but there is not necessary causal dependency between the components and the value of the faulty variable (F).

In the analyzed case, i.e. F being faulty and G correct, the final diagnoses for the considered case are calculated as reduced elements of the Cartesian product of $DCF_1 = \{m1, m2, a1\}$ and $DCF_2 = \{m1, m3, a1, a2\}$ [14]. There are the following potential diagnoses: $D_1 = \{m1\}$, $D_2 = \{a1\}$, $D_3 = \{a2, m2\}$ and $D_4 = \{m2, m3\}$. They all are shown in Fig. 2.

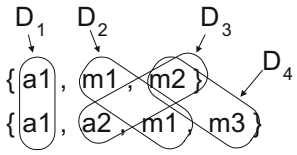

Fig. 2. Generation of potential diagnoses

Note that so far only binary faults were considered (i.e. a component may be faulty or not). In [14] and further in [11] an attempt at introducing qualitative diagnoses was undertaken. After calculation of possible binary diagnoses their qualitative forms were considered, and with use of inference rules representing simple constraints inconsistent qualitative diagnoses were eliminated. In the next section a still more precise, in-depth, numerical analysis will be carried out. The model of the system will be used as constraints. The binary logical diagnoses (i.e. *faulty* or *correct*) will be further refined with exact numerical characteristics.

3 Constraint Programming

In this section a brief note on Constraint Programming is presented. We aim at explaining the basic ideas of this promising technology for solving complex, combinatorial problems. Our presentation is based on [11].

A Constraint Satisfaction Problem (CSP) is one where the goal consists in finding a legal assignment of values to a set of predefined variables so that a set of given constraints is satisfied.

More formally, after [4] let $X = \{X_1, X_2, \ldots, X_n\}$ denote a set of variables, $V = \{V_1, V_2, \ldots, V_n\}$ is a set of domains for the variables in X and C is a set of constraints. Each constraint is given by a pair (S_i, R_i), where S_i is referred to as the *scope* (or scheme) and consists of a selection of variables from X, while R_i is a relation defined over a Cartesian Product of domains appropriate for the variables in the scope. The Constraint Satisfaction Problem is given by the triple (X, V, C).

A solution to CSP given by (X, V, C) is any assignment of values to variables of X of the form $\{X_1 = v_1, X_2 = v_2, \ldots, X_n = v_n\}$, such that $v_i \in V_i$, and for any constraint in $(S_i, R_i) \in C$, R_i is satisfied by the appropriate projection of the solution vector $(v_1, v_2, \ldots, v_k)$ over variables of S_i. Obviously, all the constraints

must be satisfied, and there can be more than one admissible solution; no solution may exist for an over-constrained problem.

If the domains of V are *finite* (e.g. binary or decimals digits are allowed) we speak about Finite Domains (FD) problems. Obviously, such problems suffer from *combinatorial explosion* while attempting at solving them. Constraint Programming (CP) is a set of techniques (or a technology) for efficient dealing which constraints.

The basic technique for solving a CSP given by (X, V, C) consists in subsequent assignment of admissible values to variables of X (e.g. by *backtracking search*); the order is chosen in an arbitrary way, and it can influence how fast a solution is found.

The principal technique proposed to solve diagnostic problems is to use constraint programming in a specific way. There are two levels of constraints to be used: (i) basic level constraints concerning variables and functional operation, and (ii) meta-level constraints concerning existence (or not) of basic level constraints.

The specification of basic level constraints is straightforward. Below we show some intuitive examples using the SWI-Prolog notation of the constraint programming library over finite domains. The typical relation symbols are used but they are preceded with the # sign. For example

```
A #= B,  Y #< Z,  X #= A*C
```

define the requirements that `A` must be equal to `B`, `Y` must be less than `Z` and `X` must be equal to the result of multiplication of `A` and `C`. The specification of constraints as such is typical for *declarative programming*, where the user specifies some knowledge to be interpreted by the system in contrast to typical procedural programming.

The way to specify the meta-level constrains – aimed in fact in encoding connections within the network structure – is by *reification*. Below we have some three examples:

```
P #==> A #= B
P #==> X #= A*C
P #==> G #= Y+Z
```

The meaning is that if `P` is True (or 1) then the constraint of the right-hand side must also hold. Instead of `P` its negation in the form `#\P` can be used.

4 Diagnosis by Abduction as Constraint Programming

The key message of this section is to show that abduction can be solved in a constructive way with efficient approach of Constraint Programming. The obtained diagnoses are more precise than in the case of purely logical abduction of Consistency-Based Diagnosis. In case of more than one potential diagnoses, the exact values of variables can be used for further analysis, and — if applicable — elimination of some spurious diagnoses due to impossible values predicted

by CP or thanks to introduction of additional measurements for confirming or rejecting the generated values.

Abductive inference consists in search for hypotheses which explain — or logically imply — the manifested faults taking into account the system model and all accessible background knowledge. The generated hypotheses must also be *consistent* with all other auxiliary observations.

Search for diagnostic hypotheses is typically restricted to a predefined set of components that can become faulty. In abductive reasoning diagnosis is carried out with a spectrum of trial-and-error or search methods and tools. A most typical approach is repeated backtracking search. In case of purely logical statements the hypotheses take the form of a set of facts, both positive and negative ones. For example, in case of Model-Based Diagnostic Reasoning (MBDR) such diagnostic hypotheses can be generated by Consistency-Based Reasoning (CBR) with reasonable search effort.

Let us consider the standard abduction scheme:

$$\frac{\phi \Longrightarrow \psi, \psi}{\phi}. \tag{1}$$

This basic scheme for abductive inference states that $\phi \Longrightarrow \psi$, and observing that ψ holds, a possible explanation of it is ϕ; both ϕ and ψ can be some arbitrarily complex formulas covering specifications of faults and fault manifestations.

Abduction is not a *legal inference rule*, i.e. one preserving *logical consequence*. In our case, ϕ is not the *logical consequence* of the faults manifested with ψ. However, it is still a kind of production rule, often used in practice. The main use of abduction is the search for *hypotheses* explaining the current observations.

In case of diagnostic reasoning, abduction is often performed by detection and elimination of inconsistencies, i.e. the so-called Consistency-Based Reasoning (CBR). The main idea of Model-Based Consistency-Based Diagnosis was presented in the seminal paper [16], and widely explored in the domain literature [3,6,7,10]. It rests in generation of diagnostic hypotheses stating which components of the system may be faulty (abduction), so that assuming them faulty explains the current observations with the model in mind in a consistent way (deduction).

At this point let us explain the CBR approach [16] from logical point of view. Consider a theory SD (*System Description*; this is a set of logical formulas (constraints) describing in a formal way behavior of the system under discourse i.e. the *background knowledge*) and a set OBS of some current observations to be explained.

To put things in a more precise formal framework, let us define the current observations as a set of variable-value pairs; hence, $OBS = \{M_1 = m_1, M_2 = m_2, \ldots M_k = m_k\}$. Each pair $M_i = m_i$ denotes an observation that variable M_i takes value m_i, $i = 1, 2, \ldots k$. Similarly, let $EXP = \{X_1 = x_1, X_2 = x_2, \ldots X_n = x_n\}$ be a set representing the fault defining variables and their values; the value x_i denotes the type of fault associated with variable X_i, $i = 1, 2, \ldots n$. The constraints are represented by the system model SD.

Definition 1. *A diagnosis for the system defined by SD and the fault observations OBS is the set EXP, such that*

$$SD \cup EXP \models OBS \tag{2}$$

and

$$SD \cup EXP \cup OBS \not\models \bot \tag{3}$$

Typically, it is also required that *EXP* is *minimal*, i.e. only the absolutely necessary hypotheses are specified within *EXP*. Further, if several competitive hypotheses are available, the most likely ones may be selected with some auxiliary tests, heuristics or statistical information.

The main problem with abduction supported by CBR is that the generated diagnoses are only *potential* fault explanations. Moreover, typically they are (i) numerous, (ii) imprecise, and perhaps (iii) uncertain (in more complex systems exhibiting stochastic behavior). Especially the case of *structural faults* may lead to extensive number of potential diagnoses and is hardly tractable.

5 Parametric Fault Identification: A Case Study

In this section a case study of precise numerical fault modeling is sketched out. As the starting point consider the four *logical* potential diagnoses: $D_1 = \{m1\}$, $D_2 = \{a1\}$, $D_3 = \{a2, m2\}$ and $D_4 = \{m2, m3\}$. They are logical in the sense that the only information provided is of logical value: *True* (or 1) if a diagnosis is valid or *False* (or 0) if a diagnosis is invalid. A constraint model for the diagnostic case presented in Fig. 1 is aimed at obtaining more detailed, numerical characteristics can be abducted.

First, the set of common observations *OBS* is modeled with simple constraints as follows[2]:

```
A #= 3, B #= 2, C #= 2, D #= 3, E #= 3,
F #= 10, G #=12,
```

In this code excerpt comma is used to separate constraints, while #<op> (like # = or # >) is used to express constraints; several typical relations can be used in place of <op>.

Now, consider the first case of $m1$ being faulty. We assume that a faulty multiplier produces the incorrect output and its value can be expressed as multiplication of the correct value by a factor K1/M1, where both the numbers are integers. The system model *SD* takes the following form:

```
A * C * K1 #= X *M1,
B * D #= Y, C * E #= Z,
X + Y #= F, Y + Z #= G,
K1 #> 0, M1 #> 0.
```

[2] The constraints are direct codes of SWI-Prolog; for constraint modeling we use the clp(fd) package.

The first constraint models the fault of $m1$. For the sake of operation in the domain of integers, `M1` is placed as a multiplication factor on the right-hand side. The other constraints correspond directly to the operation and connections of the components. The produced output is:

```
EXP = {X=4, Y=6, Z=6, K1=2, M1=3}
```

and it can be easily checked by hand.

As the second case consider the diagnosis $a1$ being faulty. We assume that a this time it is the adder that produces the incorrect output and its value can be expressed as subtraction from the correct value a factor `A1` (an integer). The model SD takes the following form:

```
A * C #= X, B * D #= Y, C * E #= Z,
X + Y - A1 #= F, Y + Z #= G.
```

The fourth constraint models the fault of $a1$. The other constraints correspond directly to the operation and connections of the components. The produced output is:

```
EXP = {X=6, Y=6, Z=6, A1=2}
```

and again, it can be easily checked by hand.

The third, a bit more complex case is the one of active diagnosis $\{a2, m2\}$. The model (SD) for this case is as follows:

```
A * C #= X, B * D * K2 #= Y * M2, C * E  #= Z,
X + Y  #= F, Y + Z +A2 #= G,
K2 #> 0, M2 #> 0.
```

Variables `K2/M2` model the multiplicative fault of $m2$ and variable `A2` models the fault of $a2$. The produced output is:

```
EXP = {X=6, Y=4, Z=6, K2=2, M2=3, A2=2}
```

and again, it can be easily checked by hand.

The fourth and perhaps the most complex case is the one of active diagnosis $\{m2, m3\}$. Here we have to introduce four variables, namely `K2/M2` and `K3/M3` for modeling two multiplicative faults, namely the one of $m2$ and $m3$, respectively. The model (SD) is as follows:

```
A * C #= X, B * D * K2 #= Y * M2, C * E * K3 #= Z * M3,
X + Y #= F, Y + Z #= G,
K2 #> 0, M2 #> 0, K3 #> 0, M3 #> 0.
```

The produced output is:

```
EXP = {X=6, Y=4, Z=8, K2=2, M2=3, K3=4, M3=3}
```

and again, it can be checked by hand.

This case study shows, that Constraint Programming allows to obtain precise numerical models of faults. They can be used for further analysis and refinement of potential diagnoses.

6 Structural Fault Identification. A Case Study

Structural faults are generally hard to identify basing on the observation of input and output signals. This is so because even a single structural fault can drastically change the overall behavior of the system. In practice, a simple break or shortcut in an electrical network can result in behavior very different from the expected one.

In the case study below we shall consider possible brakes of connections (e.g. a wire break) among system components in the system presented in Fig. 1. The simple experimental model covers 10 possible breaks, 6 of them concerning input connections (between the inputs A, B, C, D, E and the 6 inputs of the multipliers $m1, m2$ and $m3$), as well as 4 internal connections (between the outputs of the three multipliers X, Y, Z and the four inputs of the two adders, namely $a1$ and $a2$).

In order to build a formal model, the following naming convention of connection points will be applied:

- input signals: A, B, C, D, E,
- system inputs: AM1, CM1, BM2, CM2, CM3, EM3; the first letter refers to an appropriate input signal, and the rest of the string (M1, M2, M3) identifies the multiplier,
- outputs of the multipliers: M1X, M2Y, M3Z,
- inputs of the adders: XA1, YA1, YA2, ZA2.

In general, the component is identified by its name (in capitals). Its input by an appropriate signal name; when in front of the component – it refers to an input, and when following the component – it refers to the output. The observed signals are just F and G.

Now the convention applied to modeling connection breaks is as follow: let P and Q be two connection points. A string of the form `P_Q` denotes a propositional formula with the intended meaning that P is connected to Q. So, for example, `D_DM3` is a propositional formula such that when true, there is the connection between D and DM3 (the case of correct connection). Simultaneously, `#\D_DM3` denotes negation of the formula (the connection is broken).

Now, the principal modeling paradigm consist in use of the so-called *reification*: the fact that a connection constraint holds (`P #= Q`) is conditioned by the existence of the connection (`P_Q` must be true). This is denoted as:

```
P_Q \#==> P \#=Q.
```

Below, the specification of the complete set of constraints defining the SD is provided. For the input level both the case of correct work and the broken connection are covered.

```
% Definition of correct and incorrect connections by reification:
A_AM1 #==> A #= AM1, #\A_AM1 #==> AM1 #= 0,
C_CM1 #==> C #= CM1, #\C_CM1 #==> CM1 #= 0,
B_BM2 #==> B #= BM2, #\B_BM2 #==> BM2 #= 0,
```

```
D_DM2 #==> D #= DM2, #\D_DM2 #==> DM2 #= 0,
C_CM3 #==> C #= CM3, #\C_CM3 #==> CM3 #= 0,
E_EM3 #==> E #= EM3, #\E_EM3 #==> EM3 #= 0,
```

In an analogous way the correct and faulty work of the 4 inter connections is defined in the model below:

```
% Inter connections - correct and incorrect work by reification:
M1X_XA1 #==> M1X #= XA1, #\M1X_XA1 #==> XA1 #= 0,
M2Y_YA1 #==> M2Y #= YA1, #\M2Y_YA1 #==> YA1 #= 0,
M2Y_YA2 #==> M2Y #= YA2, #\M2Y_YA2 #==> YA2 #= 0,
M3Z_ZA2 #==> M3Z #= ZA2, #\M3Z_ZA2 #==> ZA2 #= 0,
```

The work of the multipliers is defined with usual constraints:

```
% Multipliers definition:
M1X #= AM1 * CM1,
M2Y #= BM2 * DM2,
M3Z #= CM3 * EM3,
```

Finally, the correct work of adders is defined as follows:

```
% Definition of adders:
XA1 + YA1 #= A1F,
YA2 + ZA2 #= A2G,
A1F #= F,
A2G #= G.
```

Note that a restriction of the number of errors can be defined in a straightforward way, by imposing a constraint on the minimal number of correct connections that must be still kept, e.g.:

```
A_AM1+C_CM1+B_BM2+D_DM2+C_CM3+E_EM3+M1X_XA1+
M2Y_YA1+M2Y_YA2+M3Z_ZA2 #>= 9,
```

means, that there can be at most one connection broken (there are 10 connections, at least 9 of them must work correct).

Now we can investigate possible faults of connections within the model. As first example consider simulation of a fault consisting in a break of connection between the output of multiplier $m3$ and the second input of adder $a2$; in such a case we expect formula `M3Z_ZA2` to be false (equal to 0). For the input signal values of the system presented on Fig. 1 let change the input values to [A,B,C,D,E] = [1,3,5,7,11]. In such a case one could expect the following output: F = 26, and G = 21 (taking into account the existence of the fault). In fact, the model leads to three admissible single-element diagnoses; these are:

- `C_CM3` = 0: the connection between input C and the first input of multiplier $m3$ is broken (and so its output is 0),

- E_EM3 = 0: the connection between input E and the second input of multiplier $m3$ is broken (and so its output is 0),
- M3Z_ZA2 = 0: the internal connection between output of the multiplier $m3$ and the second input of adder $a2$ is broken (and so the input of adder $a2$ receives signal 0).

Now, an experiment when we search for two-element diagnoses (the number of correct connections is constrained to be 8) results with three admissible diagnoses; these are: (C_CM3 = 0 and E_EM3 = 0), (C_CM3 = 0 and M3Z_ZA2 = 0), and finally (E_EM3 = 0 and M3Z_ZA2 = 0). Note that all the two-element diagnoses are not *minimal*; they are combinations of the single element ones.

It is symptomatic in this simple case that after removing the constraint on a number of diagnoses, we obtain only one diagnosis more, i.e. (C_CM3 = 0 and E_EM3 = 0 and M3Z_ZA2 = 0), which obviously is not minimal either.

Now, in order to show another example, let us assume that the observed outputs are F = 5 and G = 55. In this case we obtain as much as 13 admissible diagnoses (of 1–4 elements). This single element diagnoses are: B_BM2, D_DM2.

Finally, consider a case when there are no single-element diagnoses. Assume the observations are F = 21 and G = 55. In fact, no single diagnosis can explain this observation. In general there are as many as 3 admissible two-element diagnoses: (A_AM1 and M2Y_YA2), (C_CM1 and M2Y_YA2), and (M1X_XA1, and M2Y_YA2) (and 7 including the non-minimal).

7 Conclusions

The paper presents an idea of applying Constraint Programming for enriching abductive diagnostic reasoning. Two case studies were explored. The first one concerns the exact (numerical) knowledge for modeling the faults. In this way not only logical or qualitative solutions are obtained (of the form yes/no), but detailed numerical characteristics of the generated solutions are provided. This kind of approach we call *constructive abduction*. The second one consists in application of Constraint Programming to precise localization of structural faults. The technique of *reification* was proposed to deal with this problem.

Possible further work may concentrate on the following issues. First, in order to reduce the number of potential diagnoses more constraints are necessary. This can be archived by providing more input-output data. The selection of potential tests seem to be worth investigating. Second, a complex model - covering components faults, component behavior and structural faults could be embedded within a single model. This may require further constraint refinement and use of auxiliary heuristic or probabilistic knowledge. Third, different type of structural faults - shortcuts, switching, etc. can be incorporated in the model. In summary, current Constraint Programming tools seem to open a new frontier of research with in abductive diagnostic inference.

Acknowledgments. The presented research was carried out within AGH University of Science and Technology Internal Project No. 11.11.120.859.

References

1. Cordier, M.O., et al.: AI and automatic control approaches of model-based diagnosis: links and underlying hypotheses. In: Edelmayer, A.M. (ed.) Preprints: SAFEPROCESS 2000, 4th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes, pp. 274–279. IFAC (2000)
2. Cordier, M.O., et al.: A comparative analysis of AI and control theory approaches to model-based diagnosis. In: Horn, W. (ed.) ECAI 2000. 14th European Conference on Artificial Intelligence, pp. 136–140. IOS Press (2000)
3. Davis, R., Hamscher, W.: Model-Based Reasoning: Troubleshooting. Morgan Kaufmann Publishers, San Mateo (1992)
4. Dechter, R.: Constraint Processing. Elsevier Science, New York (2003)
5. Feldman, A., Pietersma, J., van Gemund, A.: A multi-valued sat-based algorithm for faster model-based diagnosis. In: González, C.A., Escobet, T., Pulido, B. (eds.) DX 2006: 17-th International Workshop on Principles of Diagnosis, pp. 93–100 (2006)
6. Hamscher, W., Console, L., de Kleer, J. (eds.): Readings in Model-Based Diagnosis. Morgan Kaufmann, San Mateo (1992)
7. Korbicz, J., Kościelny, J., Kowalczuk, Z., Cholewa, W. (eds.): Fault Diagnosis. Models, Artificial Intelligence, Applications. Springer, Berlin (2004)
8. Kościelny, J.M.: Methodology of Process Diagnosis, Chap. 3, pp. 57–114. In: [7]. Springer (2004)
9. Kościelny, J.M.: Models in Process Diagnosis, Chap. 2, pp. 29–43. In: [7]. Springer (2004)
10. Ligęza, A.: Selected Methods of Knowledge Engineering in System Diagnosis, Chap. 16, pp. 633–668. In: [7]. Springer (2004)
11. Ligęza, A.: A Constraint Satisfaction Framework for Diagnostic Problems, pp. 255–262. Control and Computer Science. Information Technology, Control Theory, Fault and System Diagnosis. Pomeranian Science and Technology Publisher PWNT, Gdańsk (2009)
12. Ligęza, A.: Towards constructive abduction: solving abductive problems with constraint programming. In: Proceedings of the International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management, *IC3K*, vol. 2-KEOD, pp. 352–357. SCITEPRESS - Science and Technology Publications, Lisbon, Portugal (2015)
13. Ligęza, A.: Towards knowledge compilation for automated diagnosis: a qualitative, model-based approach with constraint programming. In: Kowalczuk, Z. (ed.) Advanced and Intelligent Computations in Diagnosis and Control. AISC, vol. 386, pp. 355–367. Springer, Cham (2016). doi:10.1007/978-3-319-23180-8_26
14. Ligęza, A., Kościelny, J.M.: A new approach to multiple fault diagnosis. Combination of diagnostic matrices, graphs, algebraic and rule-based models. The case of two-layer models. Int. J. Appl. Math. Comput. Sci. **18**(4), 465–476 (2008)
15. Puig, V., Escobet, T., Ocampo-Martinez, C., Tornil-Sin, S.: Robust fault diagnosis of non-linear systems using constraints satisfaction. In: Preprints of the 7th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes, pp. 1138–1143 (2009)
16. Reiter, R.: A theory of diagnosis from first principles. Artif. Intell. **32**, 57–95 (1987)
17. Travé-Massuyès, L.: Bridges between diagnosis theories from control and AI perspectives. In: Korbicz, J., Kowal, M. (eds.) Intelligent Systems in Technical and Medila Diagnosis, pp. 3–28. Springer (2014)