A Combination of Petri-Nets and Linear Programming in Design for Dependability

A. Pataricza
P. Urbán

5/95, 12/97

Contents

1 Motivation

In this report a new approach to system diagnosis is investigated which is mainly based on operation research methods and the theory of Petri nets. It will be incorporated into a framework aiming the creation of an integrated hardware-software codesign environment. The plans for the framework, HIDE, were proposed as an ESPRIT Open LTR project and are currently under evaluation. HIDE will expose practitioning designers only to a standardised system modelling paradigm, UML, while it will allow them to use more formal methods capable of analysing and validating the quality of services and the dependability aspects of the design. These formal methods make heavy use of different, highly abstract mathematical formalisms which prevent non-expert designers from applying them and necessitate a workload intensive manual re-modelling of the system. HIDE will deal with this problem by automatic transformation of the UML model into the formal models.

The one origin of the results presented here is the field of **operation research**, widely used since WW II in modeling large scale systems, before all in different economics problems. This field of the science offers well-proven methods for handling extremely large systems of constraints and optimization algorithms for a large field of objective functions. One of the most well-developed topics in this field of science is the so-called linear integer programming dealing with the search of optimal solutions of linear inequality systems over positive integers with a linear objective function.

Models based on this formalism might be the only computationally feasible alternative in cases where a dependability aspect is investigated which requires the handling of the entire fault set. They deliver relatively short answers to questions, thus their main use will be to verify certain properties and tune certain parameters of the design; such are checking the existence of a single point of failure, delivering maximum likelihood diagnostics and compute the coverage of a test set, for instance. These questions would necessitate an exhaustive search in the case of fault oriented methods.

The other apparatus this approach relies on is the theory of **Petri nets**, which is one of the rising stars in modeling of computer systems. They are primarily used to reduce the problem size during the transformation to an optimisation problem. A great advantage of this formalism is that it offers a broad spectrum of modeling possibilities even for parallel and non-deterministic system — and in the field of diagnosis, diagnostic uncertainty plays a key role, partially to express random fault effects. These properties have made it a frequently used technique in order to keep the model complexity relatively low. Some papers were published in the most recent years on the modeling simple diagnostic problems with the help of Petri nets and solving them with the help of integer linear algebra. The report generalizes the known methodology in order to cover realistic problems as well.

The paper focuses on the problematic of formulating diagnostics problems with the help of this apparatus, as having relatively small and well structured models is a prerequisite for solving the problems efficiently. It was not the intention of the paper, however, to be fully self contained by summarising all the standard solution methods within these fields, as they are used only as a toolbox for diagnostics, and are published already in numerous excellent textbooks. Similarly, pure technical proofs of mathematical theorems and lemmata are omitted too for an easier readability. This is in accordance with one of the main guidelines of HIDE: extensive re-use of existing — commercially available and academic — tools, in particular those used for the evaluation of mathematical models.

A special emphasis was given to the automatic construction of the formal models from the ones used in the engineering practice, like data-flow models which has gained a rapidly increasing importance in hardware-software co-design. Details and an experimental implementation of such a transformation is presented.

The implementations of the algorithms were extended by conversion tools and tools of other kind. Together they form a system of loosely interconnected software components. This software system interfaces to optimisation software and — in future — the rest of the framework.

The report is structured in the following way:

- The first section summarises some basic definitions used in the field of diagnostics.
- The next section summarises the basics of the mathematics of Petri nets used in the further parts of the report, along with an overview on the previous results of their use in the field of diagnostics.
- Subsequently, the refined mathematical model is presented, allowing for a broader spectrum of diagnostics rules to be used.
- An automatic transformation of dataflow models to the refined Petri net models is discussed. Modeling aspects are dealt with and an experimental implementation is presented.
- In the following section the new solution algorithm of the much more complex model is described in details, along with the implementation and its analysis.
- A refined model is introduced in the subsequent section and it is shown that the overwhelming majority of typical problems in the diagnostics can be formulated by means of this model.
- A summary on the most important applications of optimization in the diagnostics follows the previous discussion. Examples are given and the viabiliy of the approach is investingated by performance measurements of an implementation.
- The software system is presented in the next section. Only the main design decisions and components are described.

- An investigation of the possibility in using other modeling formalisms and thoughts on their automatic transformations close the report.

2 Basic models of fault diagnostics

The process of diagnostics is in the general sense a backward inference process, deducing from the observed failures their potential sources, the faults. At first a general mathematical model is presented for the information flow of such system, while simultaneously the most important definitions and categorization used in the subsequent parts of the report are introduced.

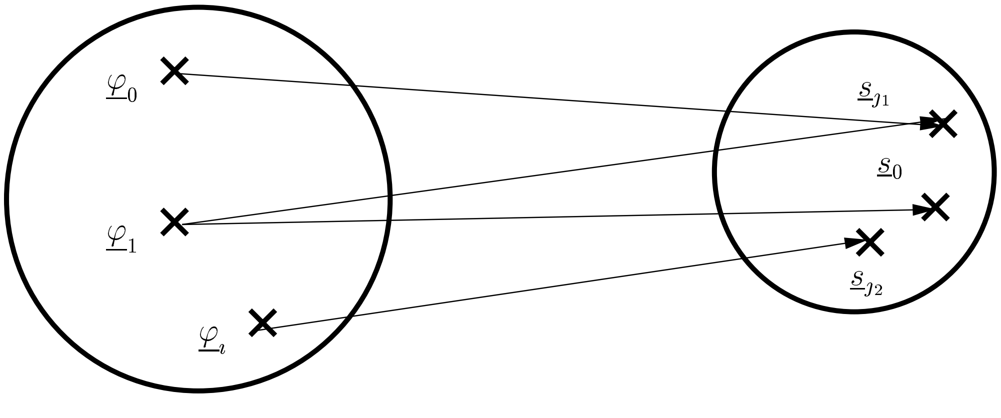
2.1 Mathematical formulation

The general form of fault diagnostics can be defined in the following way:

Let us assume that the fault model of the *unit under test* (UUT) comprises all single and multiple faults from the elementary fault set $\mathcal{F} = \{f_1, \ldots, f_f\}$. The actual state of the UUT can be characterized by the fault state vector $\underline{\varphi} = [\varphi_1, \ldots, \varphi_f]$, composed of indicator bits having a value of 1, if the corresponding elementary fault is present. Note that the cardinality of the fault state space depends exponentially on the number of elementary faults, as without further restrictions on the fault model the occurrence of any binary f-tuple from the set $\mathbb{B}^f$ is allowed as fault state vector.

The basis of the diagnostics is provided by the *syndrome* composed of the elementary test outcomes, like test results, observed failure modes etc. This observation is described in the form of the *syndrome vector* $\underline{s}_f = [s_1, s_2, \ldots, s_o]$. A 1 denotes a failure indication, a 0 its absence. In the case of a partial observation (some test results are still not available) the value u =*unknown* is allowed too to appear in $\underline{s}_f$.

It can be assumed that *no false alarms* will be generated in the overwhelming majority of technical diagnosis systems, especially in the field of digital diagnostics, i.e. a fault-free system delivers the deterministically determined proper results and no failures will occur. This way the fault-free state $\underline{\varphi}_0 = \underline{0}$ will be always mapped to the fault-free syndrome $\underline{s}_0 = \underline{0}$.



$\underline{\varphi}_0$
$\underline{\varphi}_1$
$\underline{\varphi}_\imath$
$\underline{s}_{\jmath 1}$
$\underline{s}_0$
$\underline{s}_{\jmath 2}$

Figure 1: The model of fault to failure mapping

However, the correlation between the fault states and syndrome is expressed typically only by *relations* in a faulty system, instead of the stronger category of *function*s. There is no deterministic mapping in many practical cases between the fault and the relevant failure due to the following reasons:

- **Inactive faults** can remain without any manifestation as error during the entire testing or observation phase, thus producing a syndrome identical with the fault-free case.
- **Latent errors** produce the syndrome corresponding to the fault-free case, as well. For instance, the incomplete coverage of the tests delivering the syndrome elements may result in a conditional error detection. If the syndrome element s_ℓ is conditionally detected only for the fault state vector $\underline{\varphi} = \{\varphi_1, \ldots, \varphi_f\}$, then both $[s_1, \ldots, s_{\ell-1}, 0, s_{\ell+1}, \ldots, s_o]$ and $[s_1, \ldots, s_{\ell-1}, 1, s_{\ell+1}, \ldots, s_o]$ can appear as its syndrome image. Note that in the general case this uncertainty in the detection relates to each elementary test outcome individually, i.e. to each syndrome element independently of the others. Thus, if a given fault is *conditionally detected* by multiple tests, then anyone of them can either detect it or even not, independently of the outcomes of the others.
- **Transient faults** manifesting in the form of temporal errors are another important factor resulting in a conditional error detection. This can even appear and vanish during the evaluation of the failure indicators transforming the diagnostic task to a crucial time dependent one.

The different cases of the fault state to syndrome mapping will be referred in the subsequent description as:

- *Deterministic fault manifestation* denotes the case, when this mapping is a function, i.e. to each fault state only a single syndrome can appear as image.
- *Uncertain fault manifestation* can occur, if the mapping is a relation, but not a function.

The problem of diagnostics can be formulated in the following form:

The set of fault states consistent with the eventually only partial syndrome is to be estimated[1]. There are basically two kinds of basic steps applied during the evaluation of the syndrome [7]:

- *Positive inference* estimates the fault state subspace $\mathcal{T}^+$ potentially activating the particular fault indication. The candidate diagnostic hypotheses must lie in the intersection of these inferred subspaces of the different syndromes, if available.

[1]Some syndrome elements can be of a still unknown value.

- *Negative inference* drawn from an inactive fault indication in the form of a rejection of all diagnostic hypotheses $\mathcal{T}^-$ from the further investigations, which would deterministically activate at least one of the actually inactive failures. Note that a negative inference can be drawn from a particular syndrome, only if the corresponding fault states would be unconditionally detected by it.

We speak on a *symmetrical* test in the specific case, if both sets included in the set of candidate diagnostic hypotheses by positive inference and excluded from it by negative inference coincide. If these two sets differ, then the test is said to be *asymmetrical.*

Example: If the fault-to error mapping is a single-output function, then $\mathcal{T}^+$ corresponds to the sub-domain $\mathcal{F}_1$ of the fault space mapped into the syndrome $s_1 = 1$. All remaining fault states in $\mathcal{F} \setminus \mathcal{F}_1 = \mathcal{F}_0$ are mapped into $s_0 = 0$, under a closed world assumption. This way, if a $s_0 = 0$ was observed, then the entire subspace $\mathcal{F}_0$ can be excluded from the set of fault hypotheses, restricting the set of candidate hypotheses to the complementary fault subspace $\mathcal{T}^- = \mathcal{F} \setminus \mathcal{F}_0 = \mathcal{F}_1$ identical with $\mathcal{T}^+$.

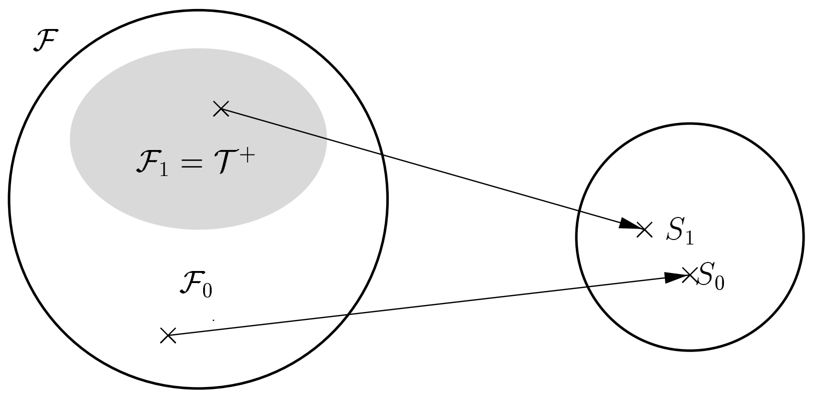
However, when the fault-to error mapping is only a relation, then the fault space is partitioned into three disjunct subspaces: $\mathcal{F}_1$ and $\mathcal{F}_0$ for those states, which result surely in a 1 and 0 syndrome value, respectively, and $\mathcal{F}_u$, which can produce a syndrome 0 or 1. Thus, when the observation was s_1, then the set of its potential sources is $\mathcal{T}^+ = \mathcal{F}_1 \cup \mathcal{F}_u$. If 0 was observed, then the set of incompatible fault states, which can be excluded as hypothesis consists only of $\mathcal{T}^- = \mathcal{F}_1$, as any non-manifested fault in $\mathcal{F}_u$ can potentially produce this 0 value as well.

Example: The property of symmetry depends basically on the level of abstraction and diagnostic granularity used by the modeler. Let us assume that our UUT is a memory protected by an error detecting code.

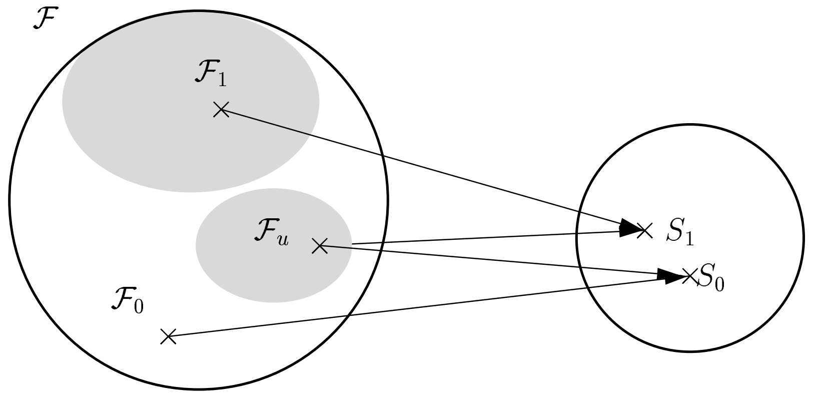
In the roughest model, the fault space consists only the states "GOOD" and "FAULTY".

An error detecting code of a low error detection probability, like a simple parity code has to be modeled as a relation, if multiple faults can occur, as an odd number of faults in the same word are detected by it, but an even number of errors remains undetected. This way parity checking is an asymmetrical test. On the other hand, a good EDC of a high error detection probability can be treated as a symmetric test in an idealized view by assuming that all single and multiple errors included into the fault model will be detected it.

However, when all fault combinations are modeled by a separate fault state, then the mapping becomes to a function, and the test is be symmetrical in both cases of EDC application.



a/ Deterministic fault manifestation



b/ Uncertain fault manifestation

Figure 2: Symmetrical and asymmetrical tests

Example: The Preparata-Metze-Chien model of system level testing uses an asymmetrical test model, as if the fault state vector is described in the form of

$$\underline{\varphi} = [\varphi_{tester}, \varphi_{UUT}] \tag{1}$$

- from the test outcome of a value 0 we can exclude the combination

$$\mathcal{T}^- = \{[0,1]\} \tag{2}$$

  from the set of candidate diagnostic hypotheses but

- from a test outcome of 1 we have to keep the hypotheses

$$\mathcal{T}^+ = \{[0,1], [1,0], [1,1]\} \tag{3}$$

  as candidates.

3 Petri net based diagnostics

Petri nets are favorite candidates for problem modeling in a wide variety of different subfields in computer science, thanks to their great expressive power in describing concurrency and non-determinism.

In this chapter a short overview on the application of Petri net based formulation and solution of diagnostic problems is given, without any intention to serve as a comprehensive overview on the theory of Petri nets, published already in numerous publications, like [3, 2].

3.1 Background mathematics

3.1.1 Definition of the Petri net

A *Petri net structure* is a directed bipartite graph with vertex subsets $\mathcal{P}$ and $\mathcal{T}$ named *places* and *transitions*, respectively. Pictorially, places are represented by circles and transitions by bars.

Places and transitions are joined by directed arcs, forming the edge set $\mathcal{E} \subseteq (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$. The predecessors and successors of a vertex are defined as follows:

- the set of *input places* of the transition $t \in \mathcal{T}$: $\bullet t = \{p|(p,t) \in \mathcal{E}\}$
- the set of *output places* of the transition $t \in \mathcal{T}$: $t\bullet = \{p|(p,t) \in \mathcal{E}\}$
- the set of *input transitions* of the place $p \in \mathcal{P}$: $\bullet p = \{t|(t,p) \in \mathcal{E}\}$
- the set of *output transitions* of the place $p \in \mathcal{P}$: $p\bullet = \{t|(p,t) \in \mathcal{E}\}$

The definition given above can be generalized in a straightforward manner for vertex subsets $\mathcal{S} \subseteq \mathcal{P} \cup \mathcal{T}$., in the form of $\bullet\mathcal{S} = \bigcup\limits_{x \in \mathcal{S}} \bullet x$ and $\mathcal{S}\bullet = \bigcup\limits_{x \in \mathcal{S}} x\bullet$, respectively.

A *source* (*sink*) transition is a transition without input (output) places, i.e. $\bullet t = \emptyset$, and $t\bullet = \emptyset$, respectively.

Places are marked with nonnegative integers, referred as *token count*. The actual state of a Petri net is characterized by the distribution of the tokens at the places. This *marking* of the Petri net is described in the form of the marking vector $\underline{M} = [m_0, \ldots, m_r]^T$, where $r = |\mathcal{P}|$.

A *Petri net* is defined by its structure and its *initial state* $\underline{M}_0$.

The edges of a Petri net structure are labeled by nonnegative integers. Place to transition arcs $(p,t) \in \mathcal{E}$ are labeled by *input weights* $w^-(p,t) \in I\!N$, while *output weights* $w^+(t,p) \in I\!N$ are associated to arcs $(t,p) \in \mathcal{E}$ joining a transition with a place. A Petri net is said to be *ordinary* if for every edge the weight is 1. By convention, if $(p,t) \notin \mathcal{E}$, then $w^-(p,t) = 0$ and if $(t,p) \notin \mathcal{E}$, then $w^+(t,p) = 0$.

A transition $t \in \mathcal{T}$ is *enabled*, if each of its input places $p \in \bullet t$ contains at least $w^-(p,t)$ tokens. An enabled transition can fire, but must not do it anyway. If the transition $t \in \mathcal{T}$ fires, then the firing removes $w^-(p,t)$ tokens from all of its input places $p \in \bullet t$ and adds $w^+(t,p)$ tokens to all of its output places $p \in t\bullet$ (*firing rule*), thus the token count of a place $p \in \bullet t \cup t\bullet$ adjacent to t changes by $w^+(t,p) - w^-(t,p)$.

The *incidence matrix* of a Petri net is an $|\mathcal{P}| \times |\mathcal{T}|$ matrix $\underline{\underline{W}} = \|w^+(t,p) - w^-(p,t)\|$.

A *firing sequence* $\sigma = \langle \underline{M}_{i_0} t_{i_1} \underline{M}_{i_1} \ldots t_{i_n} \underline{M}_{i_n} \rangle$ or in a shorter notation $\sigma = \langle t_{i_1}, \ldots t_{i_n} \rangle$ is executable, if each of its firings fulfills the precondition of the actual firing rule. In this case $\underline{M}_{i_n}$ is said to be *reachable* from $\underline{M}_{i_0}$ by the sequence σ, abbreviated as $\underline{M}_{i_0}[\sigma > \underline{M}_{i_n}$.

3.1.2 Quantitative characterization of firing sequences

The *firing count vector* $\underline{\sigma}$ of dimension $q = |\mathcal{T}|$ consists of the number of occurrences of the individual transitions in the firing sequence σ ordered according to some fixed order of the transitions. The individual elements of $\underline{\sigma}$, i.e. the number of times that transition t_i fires in σ will be referred as the *firing count* of t_i in σ, and denoted by σ_i.

The *support* $\sup(\sigma)$ of a firing sequence is the subset of transitions activated by it, i.e. the subset of transitions, with strictly positive firing count in $\underline{\sigma}$. The *occurance vector* $\underline{X}(\sigma)$ of $\underline{\sigma}$ is the characteristic vector of its support, i.e. it contains a value of 1 in exactly those positions, where the firing count vector has a positive value, while in the remaining positions the support vector element is 0. By introducing the sign function restricted for non-negative integers $z \in I\!N_0$ in the form of

$$\operatorname{sig}(z) = \begin{cases} 1 & : \quad z > 0 \\ 0 & : \quad z = 0 \end{cases} \tag{4}$$

this definition becomes

$$\underline{X}(\sigma) = \left[\operatorname{sig}(\sigma_1) \ldots \operatorname{sig}\left(\sigma_q\right)\right]^T . \tag{5}$$

The domain of the sign function will be extended for a notational simplicity to vectors over the non-negative integers $\underline{z} = [z_1, \ldots, z_n] \in I\!N_0^n$ in the form of:

$$\operatorname{sig}(\underline{z}) = [\operatorname{sig}(z_1), \ldots, \operatorname{sig}(z_n)] \tag{6}$$

This way, Equ. (5) becomes to

$$\underline{X}(\sigma) = \operatorname{sig}(\sigma) . \tag{7}$$

Theorem 1 (State equation) *If $\underline{M}_{i_0}[\sigma > \underline{M}_{i_n}$, then the so-called* state equation *of the Petri net defines a necessary condition for the corresponding firing count vector*

$$\underline{M}_{i_n} - \underline{M}_{i_0} = \underline{\underline{W}}^T \underline{\sigma}, \tag{8}$$

where the subscript T denotes transposition.

In the very special case, if

$$\underline{\underline{W}}^T \underline{\tau} = \underline{0} \tag{9}$$

the execution of the corresponding firing sequence -if possible- does not alter the marking of the Petri net. Such a nonempty firing sequence $\underline{\tau}$ is called *transition invariant* or shortly *T-invariant.*

It must be noted that the conditions formulated in Equ. (8) and as special case in Equ. (9) are only of a sufficient type, accordingly their fulfillment does not guarantee the fireability.

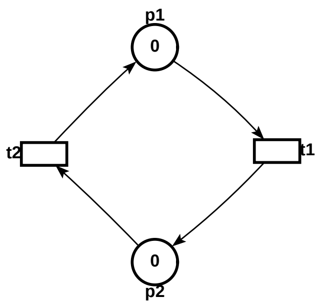


Figure 3: Sample net

For instance, the firing count vector $\underline{\tau} = [1,1]^T$ is a T-invariant of the net depicted in Fig. (3), but it is not fireable from the initial marking $\underline{M}_0 = [0,0]^T$. Moreover, the firing count vector contains less information, as the firing sequence itself. For instance, $\sigma_0 = \langle t_1, t_2 \rangle$ is fireable from $\underline{M} = [1,0]^T$, but $\sigma_1 = \langle t_2, t_1 \rangle$ not, despite the fact that their joint firing count vector is $\underline{\sigma} = [1,1]^T$.

A T-invariant $\underline{\tau}$ is of a minimal support, if there exist no T-invariant $\underline{\tau}_1$ having a support as a true subset of it, or of and identical subset, but less of firing count(s), i.e.

$$\nexists \underline{\tau}_1 : \underline{X}(\underline{\tau}) \neq \underline{0} \wedge \underline{X}(\underline{\tau}_1) \leq \underline{X}(\underline{\tau}) \tag{10}$$

.

3.1.3 Estimation of T-invariants

Obviously, T-invariants form a vector space, as any linear combination of T-invariants is itself a T-invariant, if its elements lie all in $\mathbb{N}_0$. The basis -or in other terminology the generators of the subspace- depend on the set of the allowed coefficients, when forming the linear combination. The most important characteristics of the different possibilities are summarized in Table 1.

Level	Domain	Coefficient domain	Linear independence	Uniqueness	Solution algorithm
1	$x \in \mathbb{Z}$	$\mathbb{Q}$	Yes	No	Gauss elimination
2	$x \in \mathbb{Z}$	$\mathbb{Z}$	Yes	No	Reduction to Hermite normal form
3	$x \in \mathbb{N}_0$	$\mathbb{Q}_0$	Not assured	Yes	Martinez-Silva Memmi Alawain-Toudic Pascoletti-Jaxy
4	$x \in \mathbb{N}_0$	$\mathbb{N}_0$	Not assured	Yes	Pascoletti
5	$x \in \mathbb{B}$	$\mathbb{B}$	Not assured	Yes	Jaxy

Table 1: Invariants and bases

3.2 Modeling of predicates

T-invariants analysis provides an adequate means for the solution of (negation-free) *clause systems*, composed of implications of the form $A_1, A_2, \ldots, A_\ell \rightarrow B$, where $\ell \geq 0$. Here $A_1, A_2, \ldots, A_\ell$ and B are ponate logic variables and implication means that if every *predicate variable* $A_\imath$ $(1 \leq \imath \leq \ell)$ is TRUE, then the consequence B holds.

In the special case of $\ell = 0$, i.e. the implication is of the form $\rightarrow B$, then B is a *fact*, or primary input logic variable. A *goal statement* has the form of $A_1, A_2, \ldots, A_\ell \rightarrow$.

An ordinary Petri net model of such logic programs can be constructed in the following way (Fig. (4)):

- *Logic variables* of the system of clauses are represented by places. A token at a place corresponds to a TRUE logic value of the corresponding logic variable.
- A source transition is associated to each *fact* connected to a single place storing the value of the variable.
- An *implication* is mapped to a transition with input arcs from the premise variables and an output arc to the consequence variable [2].
- *Goal statements* are modeled by a sink transition.

An input combination implying the target statement corresponds to the

- *firing* of the relevant input transitions the flow;

[2]Note that as the precondition of the firing is the presence of a token at each input place denoting a TRUE value, the transition implements simply a logic AND function. A logic OR relation can be represented by multiple input arcs to the place of the result from the transitions realizing the operands.

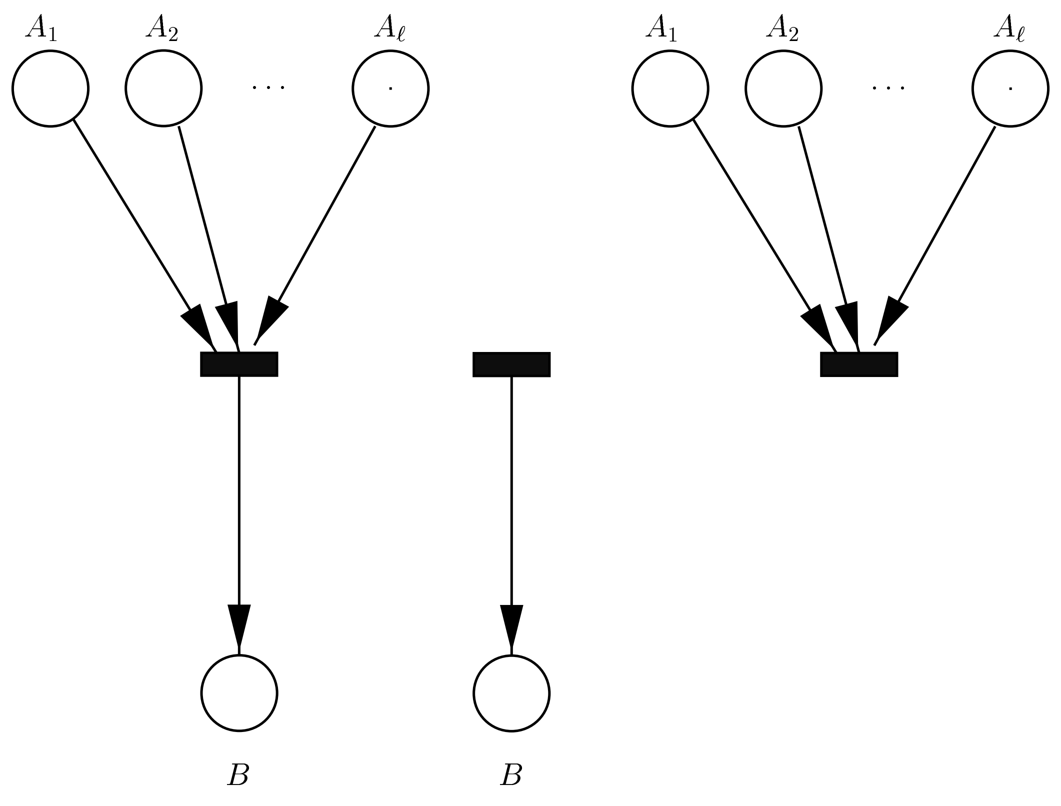


Figure 4: Modeling of clauses by Petri net elements

- the *flow* of these tokens across the Petri net;
- a *final firing* of the goal transition leaving the Petri net empty.

This way, a solution of the clause system executes a firing sequence corresponding to a T-invariant. Moreover, the estimation of the T-invariants provides a sufficient condition in this special case, as by a theorem of Murata [4]:

Theorem 2 (Murata) *IF* $\forall t \in \mathcal{T} : |t \bullet| \leq 1$ *in a Petri net, then any firing sequence* τ *corresponding to a T-invariant is fireable from the empty marking* $\underline{M}_0 = [0 \ldots 0]^T$.

i.e. the fulfillment of the state equation is a sufficient condition of fireability for this restricted special class of Petri net structures.

The preconditions of this theorem are satisfied in the Petri net model of a clause system, as facts and implications are mapped to transitions with one, goal statements to transitions with no output edges.

Theorem 3 (Murata) *A goal statement in a logic program can be deduced, iff it is fired by a sequence corresponding to a T-invariant.*

Accordingly, after estimating a basis for all T-invariants, the clause system can be solved in the following way:

Algorithm 1 (Murata) *The algorithm solves the clause system by estimating the supports of the minimal T-invariants and re-mapping these to the values of the logic variables, by applying the following steps:*

1. The set of all T-invariants of *minimal support* are estimated, for example by the Martinez-Silva algorithm [6]. Minimality is required for the avoidance of unnecessary bindings of don't care primary variables.

2. All T-invariants with supports covering sink transition(s) of the goal statement(s) fired are selected.

3. All primary input logic variables with source transitions included in the support of the T-invariant are set TRUE in a solution corresponding to a particular T-invariant. In a simpler form, the value of the support vector of the T-invariant is assigned to the logic variable.

This way the clause system can be reduced to the solution of the *Diophantine equation system* generating the T-invariants of the Petri net.

Remark: Note that negation-freedom is essential for the application of the algorithm described above. The (not necessarily minimal) T-invariants form a vector space, so the sum of two T-invariants is a T-invariant too. If the algorithm is applied for instance on the simple implication $A_1 \oplus A_2 \rightarrow B$ or in an equivalent form: $A_1\bar{A}_2 + \bar{A}_1 A_2 \rightarrow B$, the two T-invariants of minimal support correspond obviously $A_1\bar{A}_2$ and $\bar{A}_1 A_2$, respectively. The logic expression $A_1\bar{A}_2\bar{A}_1 A_2 \equiv 0$ corresponding to the sum of the T-invariants, however, is obviously not a true solution of the clause system.

3.3 Diagnostic problem formulation and solution

When formulating diagnostics problems in the form of a clause system:

- The primary inputs correspond to the -typically unobservable- internal fault states of the system under test;
- the clauses describe the error propagation mechanism;
- the goal statements denote the observable failure effects.

The diagnostic task is in this sense nothing else, then a search for those primary input combinations, i.e. source transitions that would trigger the sink transitions justifying the actual pattern of active failures.

3.3.1 The Portinale method

Symmetric tests are implicitly assumed in the method published by Portinale [5]. The algorithm builds the Petri net model of the fault $\Rightarrow$ error $\Rightarrow$ failure syndrome propagation chains using the algorithm described in Section 3.2.

As this model includes only the effects of active faults and errors, negative inferences (i.e. typically exclusion of some fault hypotheses based on the inactivity of some failure indications) are handled in an indirect way. The difficulties originate in the facts that

- a "transition is fired in the T-invariant"-like result has different interpretations at the different parts of the Petri net modeling the logic program;
- multiple goal statements are to be handled simultaneously;

In more details, the interpretation problem is the following one:

- At the sink transitions a simple *binary* logic is valid, thus a firing is made absolutely identical with the active failure indication, its missing with the inactive indication.
- However, at the source transitions a firing and the associated logic value of 1 means defines only the *necessary* conditions for the firing of the goal transitions. A missing firing can cover both of the cases that the actual logic variable *must* be set to 0, or it can have optionally a value of 1 in a non-minimal T-invariant covering the actual minimal one (don't care variable). This way a two-valued logic is used for representing a ternary problem.

Portinale uses a separate phase for solving these problems:

Sink transitions representing failure observations are grouped into two sets: $\mathcal{T}^+$ denotes the set of sink transitions corresponding to active failure indications during the actual diagnosis, while the set $\mathcal{T}^-$ denotes those belonging to inactive failure signals. $\mathcal{T}^+$ and $\mathcal{T}^-$ correspond to the transitions, which associated support vector element is of a value of 1 and 0, respectively.

Now, there are three disjunct cases for any T-invariant $\underline{\tau}$ estimated by the algorithm described in Section 3.2 :

- The T-invariant would trigger both active and inactive failure signals, i.e.

$$[\sup(\underline{\tau}) \cap \mathcal{T}^+ \neq \emptyset] \wedge [\sup(\underline{\tau}) \cap \mathcal{T}^- \neq \emptyset] \tag{11}$$

  leading to a contradiction with the actual syndrome. Such T-invariants do not deliver a diagnosis and are rejected from the further investigation.

- The T-invariant triggers only sink transitions associated with active fault signals, i.e. $\sup(\underline{\tau}) \cap \mathcal{T}^- = \emptyset$. This delivers a partial diagnosis consistent with the active failure indications by setting all primary input logic variables, which source transition belongs to $\sup(\underline{\tau})$ TRUE, as described already above.

- In a similar way if the T-invariant triggers only sink transitions associated with inactive fault signals, i.e. $\sup(\underline{\tau}) \cap \mathcal{T}^+ = \emptyset$, a partial diagnosis consistent with the inactive failure indications is provided by setting all primary input logic variables with source transition belonging to $\sup(\underline{\tau})$ to FALSE.

Any intersection of a pair of partial solutions from the sets of active and inactive partial diagnoses provides a diagnosis compatible with the syndrome.

3.3.2 Evaluation of the method

The major advantage in the T-invariant based modeling results from the reduction of diagnostic problems to the solution of a linear Diophantine equation system. The major insufficiencies of the approach result from the following basic limitations:

- The method presented is capable only of handling symmetric failure indications and requires external post-processing for negative inferences.

- There is no way to express don't care and unknown values, thus a failure indication not known yet can not be handled in an integrated way.

- The main reason of the limited capabilities of the model described above results from the insufficient representational power of the state space handling only the TRUE/FALSE logic values without allowing the use of negate variables.

- A simple introduction of the negate variables, however, would not guarantee a mutual exclusion of the firings of the transitions representing the ponate and negate variables, thus it would result in a potentially inconsistent diagnosis. Similarly, the introduction of the notation of inhibitor arcs would prohibit the use of the T-invariant analysis methodology.

4 The refined mathematical model

In Section (3.3.1) the basic idea of the model introduced by L. Portinale was presented. In this section the previous models will be refined in order to come much more close to the typical needs in computer system diagnostics. The basic requirements for the modeling technique will be summarized at first. The solution methodology of the model class will be described in the next section. The diagnostic model will be further refined to include all important factors involved in (pure) logic diagnosis. The last section in this chapter examines an experimental implementation and its performance.

4.1 Basic requirements

4.1.1 Stationarity

It is assumed that the system is a *stationary* fault state. The terminology of stationarity is used here for a clear distinction of the notion of permanent faults. The meaning of this limitation prohibiting changes in the fault state during diagnostics is the reduction of the problem complexity. Note that temporal faults can be still modeled by diagnostic uncertainty, as test manifestation relations can be handled too.

4.1.2 Fault model

The mathematical model should be able to properly handle multiple valued logic, due to the following reasons:

- **Single and multiple faults:** The importance of diagnosing simultaneously appearing independent faults is rapidly decreasing due to the increasing reliability of electronic components, thus many diagnostic algorithms take the advantage of the use of a single, or few faults assumption. However, when coping with correlated faults, like those resulting from a single defect in the form of follow-up defects without a detailed knowledge of the defect propagation chain, the system has to take the pessimistic approach to allow any combination of faults.
- **Handling of arbitrary test manifestation classes:** The limitation to handling symmetrical test manifestation models is too restrictive for many practical applications. The mathematical model should be able to describe arbitrary test manifestation models, inclusive asymmetrical and conditional ones.
- **Handling of uncertain fault manifestation effects:** Diagnostic uncertainty plays an important role in the simplification of the fault model, as this offers an appropriate means to avoid the use of over-detailed models.

- **Unknown value handling:** The number of potential failure indicators exceeds the limit of a reasonable simultaneous observation in numerous practical applications. Moreover, as testing is always a time-requiring process, *diagnostics on-the-fly* offers the single possibility for keeping the recovery time in systems sufficiently low, where testing related time redundancy is costly either in the terms of financial-like costs (e.g. the computing time in massively parallel systems), or the risk of an uncontrolled, or ill-functioning equipment (safety relevant real-time process control applications). However, the processing of such partial information necessitates an efficient processing of unknown values. A similar requirement arises, when using adaptive diagnostics in large scale systems.

- **Model compactness:** Beyond doubt, modern computers allow the processing of huge models, but the computational complexity of the solution algorithms of Petri-net based diagnostic models depend by a high exponent on the model size, as it will be described later on. Thus, a model size reduction by a factor of two results typically a decrease in the computing time by one-two orders of magnitude. Obviously, a key factor in assuring the compactness of the entire model is a compact representation of the domain of the faults. If different fault modes can be associated with a component of the UUT, or with the entire system, which form a disjunct event space, then the corresponding representation in the Petri-net should effectively utilize this kind of reduction as well.

- **Interfacing to high level descriptions:** One of the key problems in the use of formal methods, like Petri-nets originates in the requirement for a very special description methodology strange for practitioner engineers. Moreover, manual model transformations are a major source of human errors in the design of diagnostics. Thus, a Petri-net based methodology must be able to process other kinds of models automatically generated by usual design methodologies, like data-flow models in hardware-software co-design, for instance.

4.1.3 Constraint handling

When describing a diagnostic problem in the form of a Petri net, two basic classes of constraints can be distinguished:

- **Internal constraints** express requirements for the consistency of the model. A key factor is the description of mutual exclusion and deterministic dependence relations between different states of the search space. For instance, in the case of a fully deterministic fault manifestation complementary failure modes can not occur, at least so far as the diagnostic model properly describes the reality. Thus, the model has to express that only a single one from the disjunct failure events can occur simultaneously, additionally to the causal fault-failure relations embedded into the unconstrained Petri net model of the fault propagation. This problem will be examined in details in the subsequent section.

- Problem depending **external constraints** limit the solution space further on typically by using a priori information on the distribution of the faults. A typical representative of such a constraint is the t-limit, widely used in the system level diagnostics of multiprocessor systems. Such a limitation reduces the search space implicitly to those fault combinations, which have an occurrence probability above a predefined threshold.

The modeling methods which need a homogeneous mathematical apparatus for solution will be preferred for both classes of constraints.

4.2 Modeling of dichotomy

When using a clause system consisting of both ponate and negate variables, or more generally, multiple valued domains, at most a single one from the corresponding transitions has to fire in a consistent solution.

As first step, a model is built neglegting all internal constraints For instance, mutually excluding variables are used, as if they were independent ones. This model will be referred further on as the *unconstrained model* of the problem. The generated solutions, e.g. the set of all T-invariants of this model form a superset of the consistent solutions, thus a further restriction expressing mutual exclusion have to be applied on the solution space as a second phase.

It should be noted before describing the solution of this problem that such a constraint exceeds the limits of the problem field solvable with T-invariant analysis. Remember that the sum of two invariants, each one covering a complementary value of the same variable, would be a T-invariant, but an inconsistent solution of the clause system.

The basic idea of the modeling of the constraints is borrowed from the theory of linear programming. It will be presented at first for simple binary choice, and further extended for an arbitrary number of alternatives.

4.2.1 Binary choice

A single variable of a binary domain and a single source transition for each state is to be modeled (Fig. (5)) in the simplest case. A fully determined solution is consistent only in the following disjunct cases:

- $x_P = 1$, $x_N = 0$, i.e. the logic variable is TRUE;

- $x_P = 0$, $x_N = 1$, i.e. the logic variable is FALSE.

The goal is now, to express this mutual exclusion constraint in the terms of linear algebra in order to have a homogeneous mathematical background.

The *dichotomy problem* can be formulated in the form of a linear inequality system. Let denote by P_P and P_N the places corresponding to the different value assignments to this binary variable. t_P and t_N are the transitions corresponding to the assignment of the TRUE and FALSE value to a logic variable respectively. The firing counts of these transitions in the T-invariant are τ_P and τ_N. It is assumed for simplicity that the Petri net model is an ordinary one, i.e. all edges have the weight of 1, the number of tokens M_P^+ and M_N^+ supplied to the places P_P and P_N equal to the firing count, i.e.

$$M_P^+ = \tau_P \tag{12}$$

$$M_N^+ = \tau_N \tag{13}$$

The variables $x_P, x_N \in \mathbb{B}$ are the corresponding elements of the occurance vector.

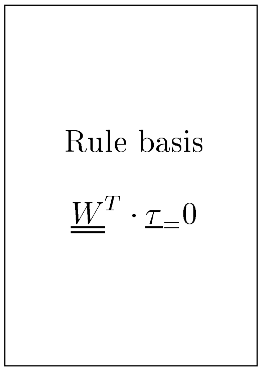


Figure 5: Binary choice

The mutual exclusion constraint prescribing the firing of a single transition has the form of

$$\text{sig}\left(M_P^+\right) + \text{sig}\left(M_P^+\right) = \text{sig}\left(\tau_P\right) + \text{sig}\left(\tau_N\right) = 1, \tag{14}$$

or in the terms of the occurance vector elements

$$x_P + x_N = 1. \tag{15}$$

The major problem in the reformulation to a linear inequality system originates in the fact that the sign function is non-linear in the general case. Accordingly, it is assumed that an upper bound is known for the firing count of each individual transition, i.e. it is known that $\forall t_\ell \in \mathcal{T} \in \{t_P, t_N\}$ what is the maximal number of times that transition t_ℓ fires in a consistent solution [3]. Thus expectedly a priori known upper bounds on σ_P and σ_N are denoted by S_P and S_N. The inequalities describing the constraints on the individual variables are the following ones, where $\ell \in \{P, N\}$:

[3]A justification of this assumption will be presented together with the complete solution algorithm in Section (6.2).

- Definition of the domains of the occurance vector elements:

$$x_\ell \geq 0 \tag{16}$$
$$x_\ell \leq 1 \tag{17}$$

- Definition of the lower bound of the firing counts:

$$\tau_\ell \geq 0 \tag{18}$$

- Definition of the occurance vector elements of the firing counts:

$$\tau_\ell \geq x_\ell \tag{19}$$
$$\tau_\ell \leq x_\ell S_\ell \tag{20}$$

Note that Equ. (18) can be omitted, as it is a consequence of Equ. (16) and Equ. (19). These inequalities are equivalent with the assignment $x_\ell = \mathrm{sig}\,(\tau_\ell)$, as Equ. (19) forces $x_\ell = 0$, if $\tau_\ell = 0$ and Equ. (20) $x_\ell = 1$, if $\tau_\ell > 0$, respectively.

At the first glance, the use of inequalities seems to result in the use of much more sophisticated solution methods than those for equation systems, however, remember that the implicit assumption on the positivity of the components of the T-invariant already involved the use of such methods.

4.2.2 Multiple choices

A similar system of inequalities can be easily constructed not only for the case of a simple binary choice, but for any number of disjunct alternatives as well. If the domain elements are represented by P_ℓ, where $\ell = 1, \ldots, k$, the mutual exclusion constraint becomes to

$$\sum_{\ell=1}^{k} \mathrm{sig}\left(\underline{M}_\ell^+\right) = \tag{21}$$
$$\sum_{\ell=1}^{k} x_\ell = 1, \tag{22}$$

by substituting the left side of Equ. (15) with the summation over all occurance vector elements $\{x_\ell | \ell = 1, \ldots, k\}$.

4.2.3 Embedded transitions

Naturally, places embedded in a Petri net can have multiple transitions supplying tokens into them. Mutual exclusion has to express in such cases that if any input transition of a place was fired, then all input transitions of all other places denoting alternative value assignments to the same variable must remain inactive during the entire firing sequence τ. In a much more formal fashion, let denote the input places corresponding to the individual elements of the domain of a variable by $P_1, \ldots P_k$. (Single indices are used here for a better readability only). The number of tokens $\underline{M}_\ell^+$ supplied to P_ℓ by τ is:

$$\underline{M}_\ell^+ = \tag{23}$$

$$\sum_{t \in \bullet P_\ell} w^+(t, P_\ell) \cdot \tau_t = \tag{24}$$

$$\sum_{t \in \mathcal{T}} w^+(t, P_\ell) \cdot \tau_t = \tag{25}$$

$$\underline{W}^+(P_\ell)^T \cdot \tau, \tag{26}$$

where $\underline{W}^+(P_\ell)$ denotes the column of $\underline{\underline{W}}^+$ corresponding to P_ℓ.

Remember that $w^+(t, P_\ell) = 0$, if $t \notin \bullet P_\ell$.

The required mutual exclusion relation can be written as:

$$\forall \imath, \jmath \in \{1, \ldots, k\}, (\imath \neq \jmath) : \left[\underline{W}^+(P_\imath)^T \cdot \underline{\tau}_t > 0\right] \to \left[\underline{W}^+(P_\jmath)^T \cdot \underline{\tau}_t = 0\right]. \tag{27}$$

The binary variable $x_\ell = \text{sig}\left(\underline{W}^+(P_\ell)^T \cdot \underline{\tau}\right)$ will denote now, whether τ was firing into place P_ℓ or not. The constraints expressing mutual exclusion have the form of:

$$\sum_{\ell=1}^{k} \text{sig}\left(\underline{M}_\ell^+\right) = \tag{28}$$

$$\sum_{\ell=1}^{k} \text{sig}\left(\underline{W}^+(P_\ell)^T \cdot \underline{\tau}\right) = \tag{29}$$

$$\sum_{\ell=1}^{k} x_\ell = 1 \tag{30}$$

4.2.4 Multiple simultaneous consequences

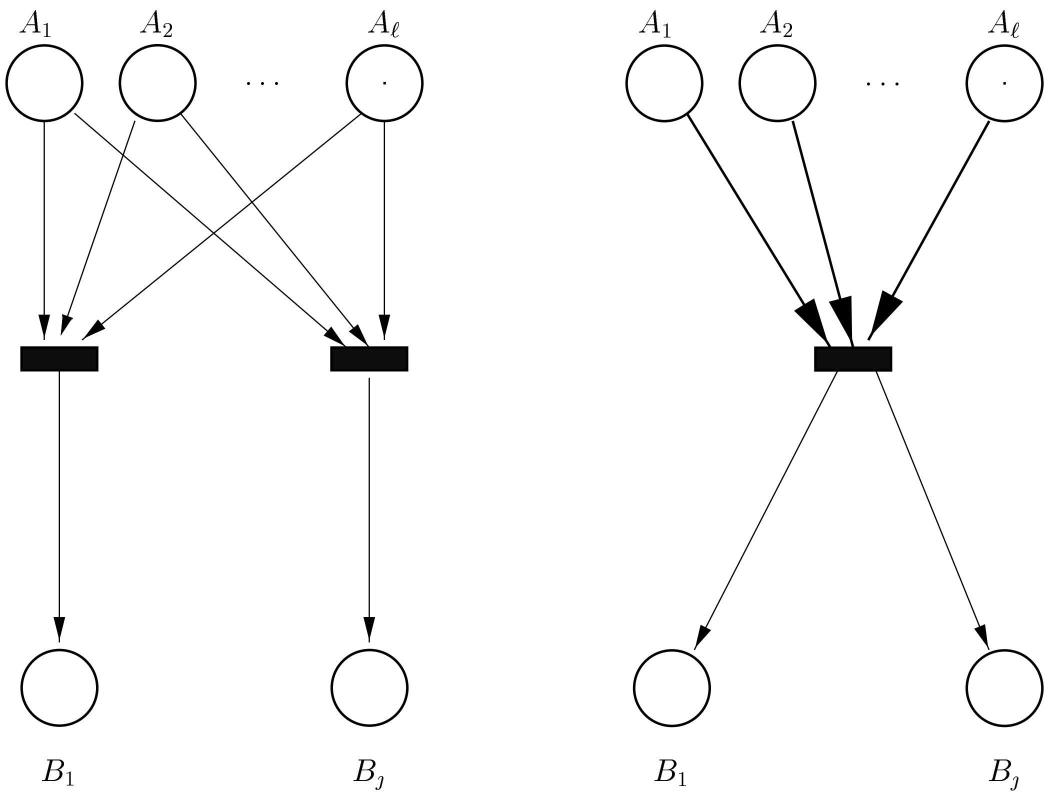
The very same premise may result deterministically in multiple consequences in many diagnostic problems. For instance, an active fault can simultaneously cause multiple errors, like a bus error can corrupt all bus devices etc. The limitation of the use of maximally single output transitions was postulated in the basic unconstrained Petri net model, as one of the preconditions of the basic Murata theorem. Note that the requirement for expressing multiple simultaneous consequences of the form

$$A_1, A_2, \ldots, A_\ell \to (B_1 \wedge \ldots \wedge B_j math) \tag{31}$$

exceeds even the class of clauses as well.

The simple introduction of transitions with multiple output arcs (t on the left side of Fig. (6)) would result in theoretical troubles with a potential loss of solutions, as it would force the number of tokens $M_1^+, \ldots, M_\jmath^+$ passing through these output arcs equal in a still unjustifiable way.

The core of the problem is that the use of independent transitions ($t_1, \ldots, t_\jmath$ on the right side of Fig. (6)) is in itself unable to express an obligatory simultaneous firing, as fireable transitions fire at will, by the very basic attribute of the Petri nets.



a/ Independent transitions b/ Multi-output transition

Figure 6: Modeling of multiple consequences

Fortunately, an external constraint offers an elegant way to express such deterministic logic relations between transitions. If single output transitions having the same predecessor places (premises) are used for modeling the individual conclusions, we have to assure that if any one of them is firing during the T-invariant firing sequence, then all of them have to do it at least once. But this is nothing else, then an equivalence relation over the occurance vector elements belonging to these transitions. This can be simply formulated by

$$x_1 = \ldots = x_\jmath, \tag{32}$$

for all of the consequence transitions. $A_1, A_2, \ldots, A_\ell \rightarrow B$

A similar problem arises, if the premises of a clause form a subset of another one, e.g. in the form of:

$$A_1, \ldots, A_\ell \rightarrow B_1 A_1, \ldots, A_\imath \rightarrow B_2, \tag{33}$$

with $\imath \geq \ell$. This way, if all premises of the second clause are fulfilled, then the same holds for the first one, as

$$A_1, \ldots, A_\imath \rightarrow A_1, \ldots, A_\ell \rightarrow B_1. \tag{34}$$

Consequently, if it was fired into B_2, then it is required that the same happens with B_1. This can be expressed simply by the occurance vector elements in the form of

$$x_2 \rightarrow x_1 \tag{35}$$

or in the terms of a linear inequality

$$x_2 \leq x_1 \tag{36}$$

forcing $x_1 = 1$, if $x_2 = 1$. Note that all transitions with such a coverage relation can be automatically extracted form the pre-incidence matrix, as the sign of its elements is identical with the characteristic vector of the predecessors $\bullet t$ of a particular transition t, thus

$$\forall t \in \mathcal{T} : p \in \bullet t \leftrightarrow \text{sig}\left(\underline{W}^-(t)\right) = 1 \tag{37}$$

. Accordingly, the premises of t_1 are contained in the set of premises of t_2, then

$$\text{sig}\left(\underline{W}^-(t_2)\right) \geq \text{sig}\left(\underline{W}^-(t_1)\right) \tag{38}$$

4.2.5 Uncertain fault manifestation

Relations can result in uncertain fault manifestations due to the ambiguity of the failure effects related with a particular fault. Two contradictory values can appear, if rules resulting in different consequences are invoked multiple fold. In this case, the mutual exclusion relation has to be relaxed at the corresponding transitions and places.

4.2.6 Unknown and don't care values

In the case, if unknown or don't care values can appear in a solution, the requirement "*exactly one* of the alternatives has to occur" should be weakened to "*at most one* of the alternatives has to occur". This can be simply expressed by reformulating Equ. (30) to

$$\sum_{\ell=1}^{k} x_\ell \leq 1 \tag{39}$$

This way an all-zero combination denotes a logic value, which is indeterminate.

Another problem in the handling of indeterminate values arises, when such values are to be compared. As example, let us assume that a specific failure indicator was not observed during the previous diagnostic data acquisition, but a costly model reduction by omitting the parts related to this unobserved failure indicator is to be avoided. (E.g. when performing an intermediate phase of an adaptive diagnostic algorithm). This case an compatibility relation allowing both the unknown and equal value assignments to the occurance vectors is to be used instead of the strict equation between the occurance vectors representing the observed and derived partial syndromes.

Let us assume now that compatibility is to be checked for two characteristic vectors of the same logic variable denoted by $\underline{x} = [x_1, \ldots, x_k]$ and $\underline{y} = [y_1, \ldots, y_k]$. Then the allowed compatible combinations are:

- $\underline{x} = 0$ corresponding to an unknown value with an arbitrary $\underline{y}$, or vica versa
- $\underline{y} = 0$ with an arbitrary $\underline{x}$, or
- $\underline{y} = \underline{x}$.

This is equivalent to

$$\underline{z} = \underline{x} \vee \underline{y} \tag{40}$$

is either of a value unknown, or consists in at most a single position a 1, thus

$$\sum_{\ell=1}^{k} z_\ell = \tag{41}$$

$$\sum_{\ell=1}^{k} x_\ell \vee y_\ell = \tag{42}$$

$$\sum_{\ell=1}^{k} \mathrm{sig}\,(x_\ell + y_\ell) \leq 1. \tag{43}$$

5 Automatic transformation of data flow networks

5.1 Data flow networks

Data flow networks belong to the most popular tools used in hardware-software co-design. Typically, system engineering environments use different forms of the data flow paradigm at the highest levels of abstraction for performance modeling and behavioral specification of the target system. The basic idea of using data-flow models was already published in [1]. Here we focus only on the aspect of automatic model transformation between data flow and Petri net based models.

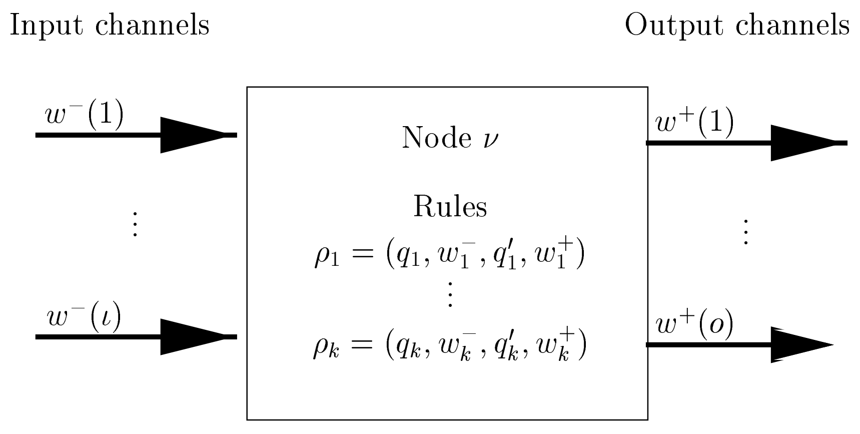
Formally, a *data flow network* is a set of nodes $\mathcal{N}$, which execute concurrently and exchange data over point-to-point communication channels C.The *data flow node* represents the functional elements of the system and describes their signal propagation behavior by a simple relation between input and output, eventually depending on the previous state of the node. The use of relations instead of input-output functions allows the modeling of non-deterministic behavior. For instance in case of diagnostics this provides a proper mean to express diagnostic uncertainty. The *channels* of the data flow network symbolize the interaction between the functional elements of the system. Internal channels link two nodes. Input (output) channels connect a single node to the outside world representing the primary inputs (outputs) of the system. *Communication events* occur when data items (subsequently called tokens) are inserted into an input channel (input event describing the arrival of some data to the primary inputs) or data items are removed from an output channel (output event denoting the appearance of results on a primary output of the system).

The functional behavior of a node $\nu \in \mathcal{N}$ is defined by the set of firing rules $\mathcal{R}_\nu$ over the input domain and over $\mathcal{Q}_\nu$, the set of possible states of the node. A node is ready to execute as soon as the data required by one of its firing rules are available and the node is in a proper state. The meaning of firing rule $\rho \in \mathcal{R}_\nu$, denoted by $\rho = (q, w^-, q', w^+)$ is that if the node ν is in state $q \in \mathcal{Q}$, each of the input channels $i \in \mathcal{I}_\nu$ holds at least $w^-(i)$ data items, then firing rule ρ is potentially selected for execution. The execution of firing rule ρ removes $w^-(i)$ data items from each input channel $i \in \mathcal{I}_\nu$ and outputs $w^+(j)$ data items on each output channel $j \in \mathcal{O}_\nu$, while the node changes its state from q to q'.

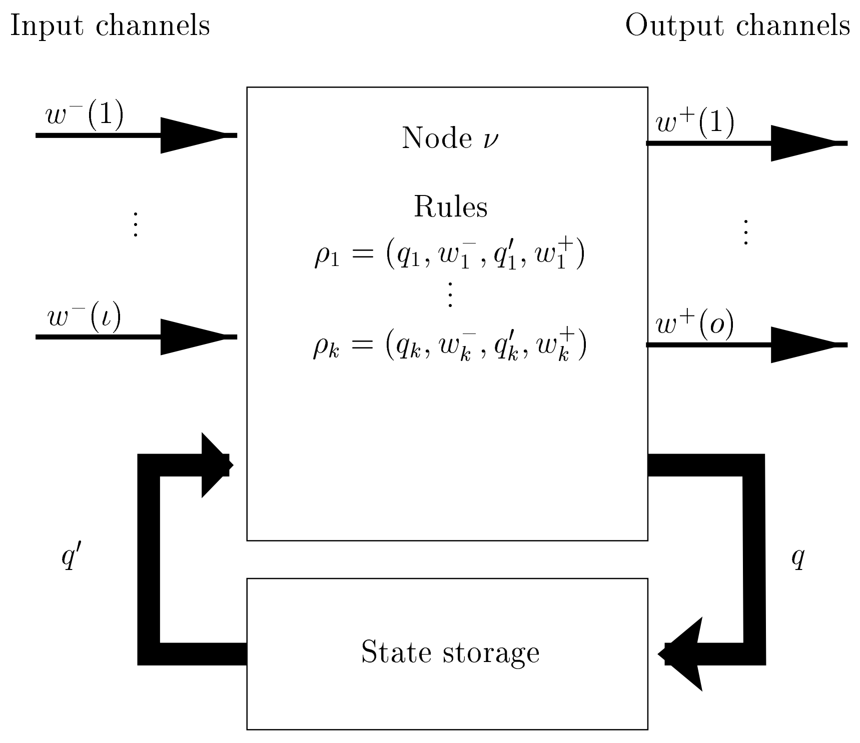
A data flow network shows an essential structural similarity to a Mealy-automaton, with the main difference of having relations in both of the next state and output mappings instead of functions (Fig. (7)).

5.2 Transformation to a Petri net

There is a straightforward transformation between data flow networks and Petri nets, where input and output channels, and state storage elements are represented by places.



a/ Basic view



b/ Non-deterministic Mealy-automaton

Figure 7: Data flow networks

Rules are mapped to transitions with inputs connected to the places standing for input channels and state storage, while the outputs of the transition fire into the places on the output channel and state storages places, respectively, as depicted in Fig. (8).

If the data flow node is of a combinational type without any internal states, then places representing state storage and arcs related to the next state relation are omitted.

Note that feedback in this direct form inhibits the search for a T-invariant based solution

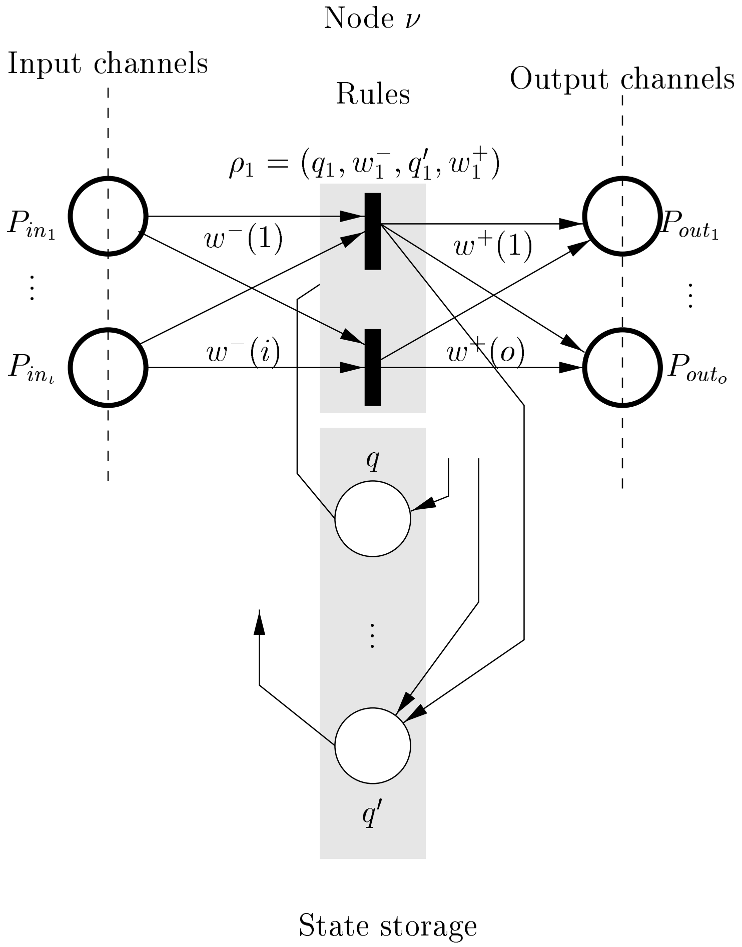


Figure 8: Transformation of data flow networks to Petri nets

in the case of a sequential data flow node. The only way found till yet is the transformation of the Petri net into an iterative array model by cutting the feedback loops and repeating the Petri net for each time frame (Fig. (9)).

Fortunately, the search for T-invariants has to be done for a single phase only, due to the identity of the models used in the individual frames. On the other hand, the number of frames is usually as low as two-three according to the experiences of a current development work aiming at the construction of a diagnostic model of the MEMSY multiprocessor [9].

Moreover, the use of data-flow networks as input description languages offers an additional advantage of a hierarchical problem decomposition. The token flow corresponding to a T-invariant can be partitioned into two disjunct parts: token flow between and within of data-flow nodes. As a system-level T-invariant is naturally a partial T-invariant for all data-flow nodes, intra-node T-invariants can be estimated individually, and the system-level T-invariant can be estimated as a composition of these partial solutions.

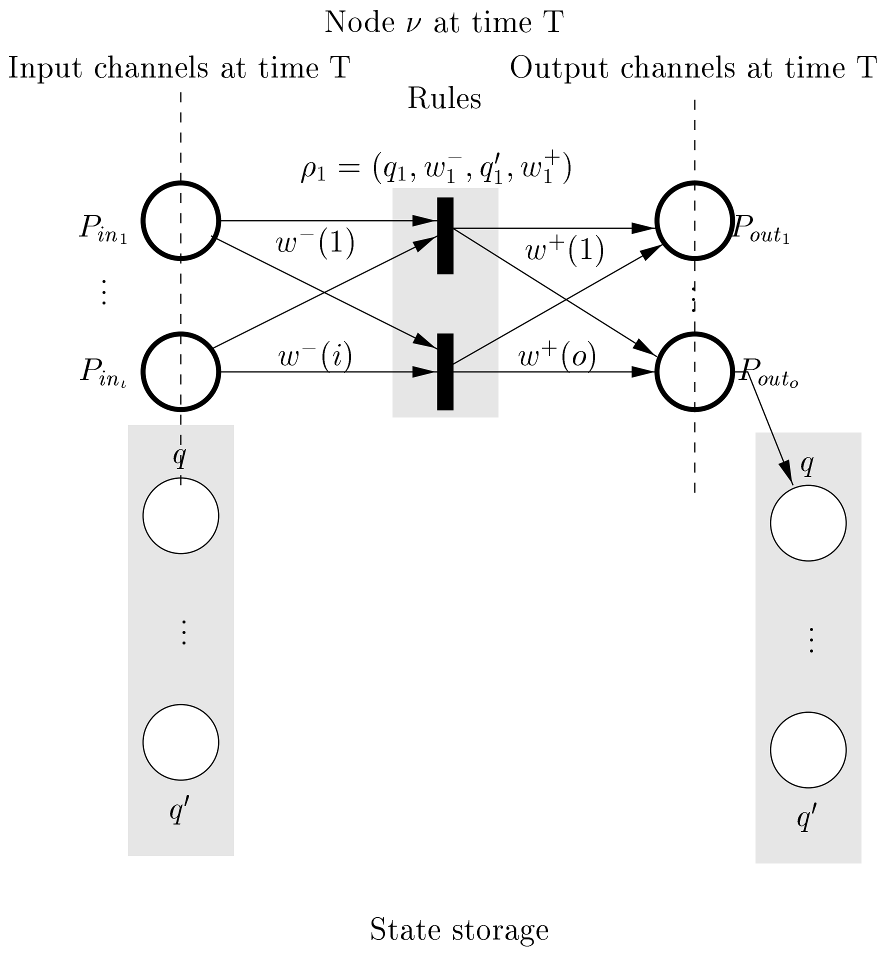


Figure 9: Iterative Petri net array model for sequential data-flow elements

5.3 Transformation of a higher level data flow network

The implementation of the transformation handles a more general class of data flow networks. In these, channels may deliver several types of tokens, as opposed to one. Firing rules may depend on the type of the tokens: they are generally of the form $\rho = (q, w^-, q', w^+)$ just like above, but $w^-(i)$ and $w + (j)$ no longer contain the number of tokens for the corresponding channels but a vector, the number of tokens for each token type.

The transformation of the previous section works without change, but it produces *coloured Petri nets*, Petri nets with several different types of tokens. They offer a number of advantages over simple Petri nets when used as a modeling tool; these aspects are discussed in Section (10.1).

Coloured Petri nets are *unfolded* to simple Petri nets here, for this is the class on which the algorithms of this report work. Unfolding means that a substructure of the Petri net is repeated for each token type: input and output channels are represented by several places and the marking of a place indicates the number of data items of a certain type which flow on the channel.

The resulting structure, shown in Fig. (10), contains transitions with multiple outputs in case a firing rule puts tokens on several different channels or it outputs tokens of different type.

As already discussed in Section (4.2.4), this should be modelled with repeating the transition once for each output arc, so that $|t\bullet| \leq 1$ and the T-invariant algorithm can be used. A constraint must be added that ensures that each of the transitions either fires at least once or does not fire at all.

In the present solution algorithm described in Section (6), constraints are formulated in terms of the occurance vector, which indicates those places into which the firing sequence corresponding to a T-invariant fired. Constraints on transition firing counts can be formulated indirectly; the transition must be replaced by a transition $\rightarrow$ place $\rightarrow$ transition structure. This way the transition has a place which no other transition fires into.

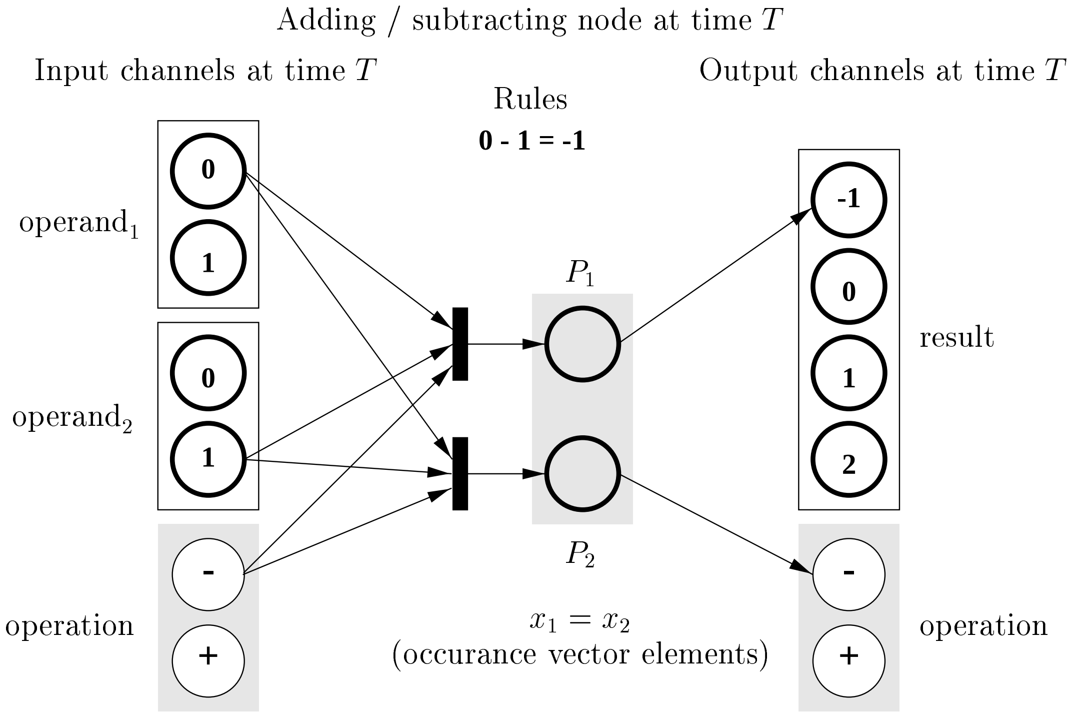

Adding / subtracting node at time T
Input channels at time T
Output channels at time T
Rules
0 - 1 = -1
operand$_1$
0
1
operand$_2$
0
1
operation
-
+
P_1
P_2
-1
0
1
2
result
-
+
operation
$x_1 = x_2$
(occurance vector elements)

Figure 10: Petri net equivalent of a firing rule of a coloured data flow network

5.4 Implementation performance

The crucial factor which determines the overall efficiency of solving a diagnosis problem is the size of the Petri net model built by the transformation. The running times of solution algorithms depend on this by a high exponent. The transformation was tested with the components of the dataflow models of the MEMSY multiprocessor ; there are 35 different kinds of components. Fig. (2) shows the sizes of the original and the transformed models for the smallest, the biggest and moderate sized components.

Component	**Dataflow model**				**Petri net model**			
	Channels		**State**	**Firing rule**	**Place**	**Trans.**	**Arc**	**Constraint**
	In	**Out**						
S_arb	46	42	10	495	2543	4919	17724	1971
Cpu	6	10	20	136	782	1494	4507	594
Disc	4	2	5	33	94	167	383	28
Reset	4	14	2	8	66	110	179	56
Term	2	2	2	8	30	46	178	14

Component	**Description**
S_arb	Arbitration on a VME bus with 8 memory units
Cpu	CPU – read, write and fetch cycles may fail.
Disc	A disc with CRC checking
Reset	Reset of the system
Term	A terminal

Table 2: Sizes of the dataflow and the Petri net models

As one can see, Petri net models are bigger by a large factor. On the one hand, it comes from the fact that a higher level coloured description is unfolded. On the other hand, little effort was given to eliminate unnecessary components. The reason is that the structures which are easy to remove by extending the algorithm can also be removed by automatic preprocessing the last model in the chain of transformations. This last model is a 0-1 optimisation problem here; the effectiveness of preprocessing is measured in Section (8.5) and the results are compared to the original model size.

The efficiency of the transformation algorithm is of low importance, it is about linear in the size of the dataflow component. Memory consumption can be reduced in a straightforward manner: only channel names and the corresponding group of Petri nettransitions must be stored in memory.

6 Linear algebraic reformulation

This section describes the algorithm transforming a Petri net model of the system into a system of inequalities. Important problem reduction steps are performed during this process.

6.1 Problem structure

Before presenting the proposed algorithm, an overview of the problem structure is given. As depicted in Fig. (11), to each input variable $V_\jmath$, where $\jmath = 1, \ldots, k$, a part of the occurrence vector $\underline{x}$ is associated as characteristic vector of its domain[4]. The mutual exclusion constraint assures that exactly one element is to be selected from the domain, or if it is written in the weaker form of an inequality, then at most one. (Cf. Equ. (39)). The occurance vector $\underline{x}$ indicates those places, in which the firing sequence corresponding to the T-invariant fired. This way it relates the value of the logic variables with the token flow $\underline{M}^+$ into the places representing them. The relation between the firing counts in the firing sequence corresponding to the T-invariant $\underline{\tau}$ and $\underline{M}^+$ is defined with the help of the post-incidence matrix $\underline{\underline{W}}^+$. The fault manifestation constraints embedded into the diagnosis rules are expressed with the help of the state equation.

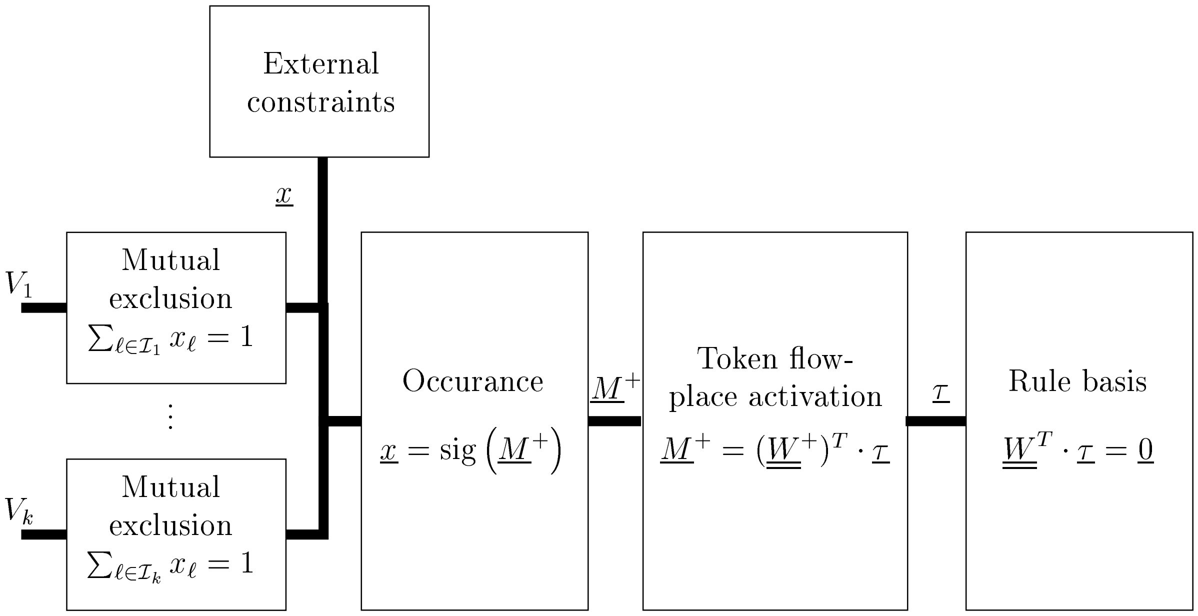

External constraints
$\underline{x}$
V_1
Mutual exclusion
$\sum_{\ell \in \mathcal{I}_1} x_\ell = 1$
$\vdots$
V_k
Mutual exclusion
$\sum_{\ell \in \mathcal{I}_k} x_\ell = 1$
Occurance
$\underline{x} = \text{sig}\left(\underline{M}^+\right)$
$\underline{M}^+$
Token flow-place activation
$\underline{M}^+ = (\underline{\underline{W}}^+)^T \cdot \underline{\tau}$
$\underline{\tau}$
Rule basis
$\underline{\underline{W}}^T \cdot \underline{\tau} = \underline{0}$

Figure 11: Constraint structure

[4]In Fig. (11) $I_\jmath$ denotes the index set of the element in the occurance vector belonging to $V_\jmath$. For simplicity, constraints corresponding to the simultaneous firing of multiple transitions are not included and all mutual exclusion constraints are depicted as equations.

6.2 The basic idea

Before formulating the solution algorithm in a formal fashion, the most important steps are individually presented.

6.2.1 Estimation of the basis

As the first step a basis $\{\underline{\tau}_1, \ldots, \underline{\tau}_c\}$ over $\mathbb{Q}_0$ consisting of minimal support T-invariants is estimated by solving the state equation for instance by the Martinez-Silva algorithm. This way, if a total number of c independent basis vectors are found, then any one of them $\underline{\tau}_\ell$, for $\ell = 1 \ldots c$ satisfies:

$$\underline{\underline{W}}^T \underline{\tau}_\ell = \underline{0}. \tag{44}$$

Now, as they form a basis over $\mathbb{Q}_0$ any T-invariant $\underline{\tau}$ can be expressed in the form of

$$\underline{\tau} = \sum_{\ell=1}^{c} \alpha_\ell \cdot \underline{\tau}_\ell, \tag{45}$$

with $\underline{\alpha} \in \mathbb{Q}_0^c$ or in a much more compact form:

$$\underline{\tau} = \underline{\underline{B}} \cdot \underline{\alpha}, \tag{46}$$

where $\underline{\underline{B}} = \|\underline{\tau}_1, \ldots, \underline{\tau}_c\|$ and $\underline{\alpha} = |\alpha_1, \ldots, \alpha_c|$.

On the other hand, the selection of an arbitrary $\underline{\alpha} \in \mathbb{B}^c$ results in a $\underline{\tau}$ satisfying the homogeneous state equation.

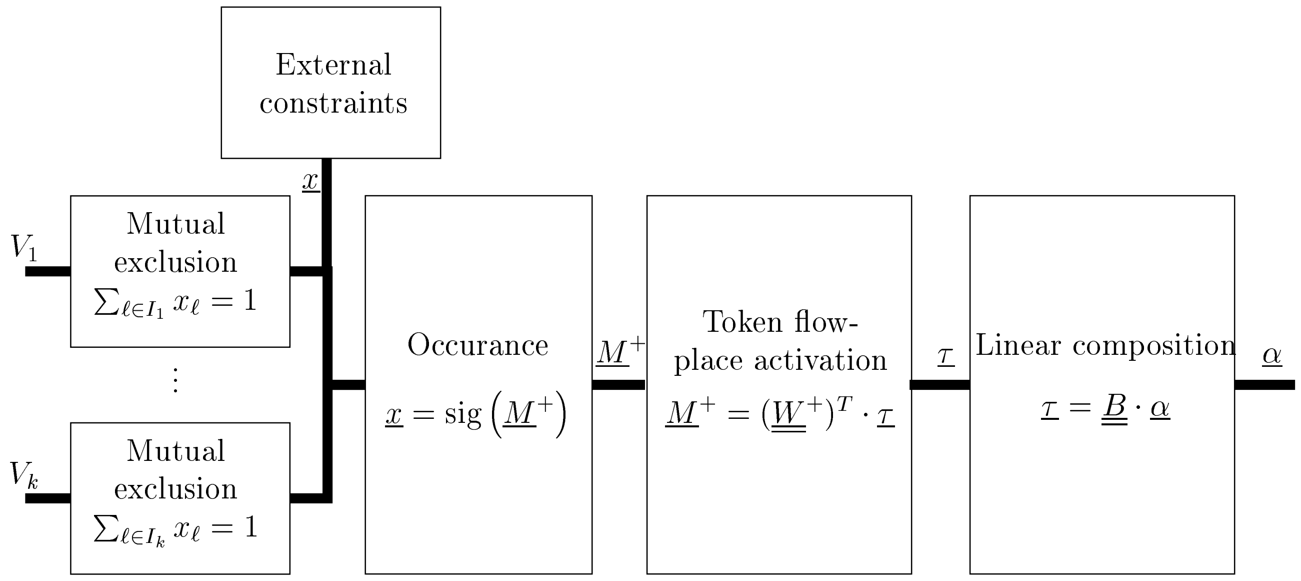

External constraints
$\underline{x}$
V_1
Mutual exclusion
$\sum_{\ell \in I_1} x_\ell = 1$
$\vdots$
V_k
Mutual exclusion
$\sum_{\ell \in I_k} x_\ell = 1$
Occurance
$\underline{x} = \mathrm{sig}\left(\underline{M}^+\right)$
$\underline{M}^+$
Token flow-place activation
$\underline{M}^+ = (\underline{\underline{W}}^+)^T \cdot \underline{\tau}$
$\underline{\tau}$
Linear composition
$\underline{\tau} = \underline{\underline{B}} \cdot \underline{\alpha}$
$\underline{\alpha}$

Figure 12: Constraint structure after estimation of the basis

6.2.2 Reduction of the variable set

Theoretically, as both $\underline{\tau}$ and $\underline{M}^+$ are determined by matrix equations by $\underline{\alpha}$, at the first glance nothing else is necessary, than to subsequently substitute $\underline{\alpha}$ into these equations. However — without going into implementation details of the solution algorithms — an efficient solution of the inequalities expressing the internal (mutual exclusion) and eventually even external constraints, we aim simultaneously at a reduction of the search space of the feasible solution by reducing the domains of the variables used.

$\underline{\tau}$ and $\underline{M}^+$ are playing only the role of a temporary variable and all mutual exclusion and external constraints are formulated in the terms of the occurance vector. Thus we can eliminate these vectors by a simple substitution of the form

$$\underline{M}^+ = (\underline{\underline{W}}^+)^T \cdot \underline{\underline{B}} \cdot \underline{\alpha}, \tag{47}$$

resulting in the simplified constraint structure depicted in Fig. (13).

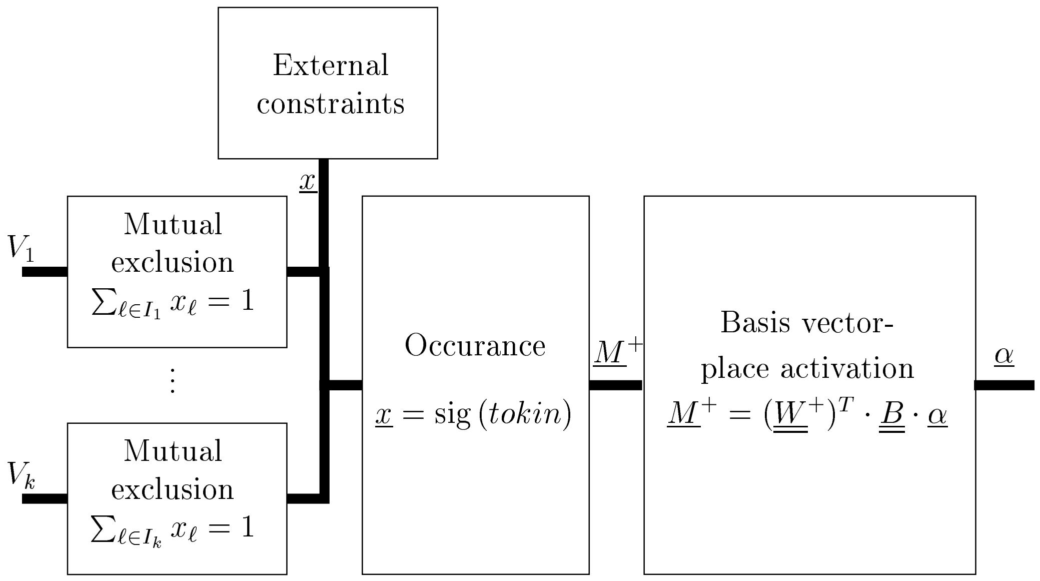

External constraints
$\underline{x}$
V_1
Mutual exclusion $\sum_{\ell \in I_1} x_\ell = 1$
$\vdots$
V_k
Mutual exclusion $\sum_{\ell \in I_k} x_\ell = 1$
Occurance $\underline{x} = \mathrm{sig}\,(tokin)$
$\underline{M}^+$
Basis vector-place activation $\underline{M}^+ = (\underline{\underline{W}}^+)^T \cdot \underline{\underline{B}} \cdot \underline{\alpha}$
$\underline{\alpha}$

Figure 13: Constraint structure after elimination of the firing count vector

As next step, a new variable $\underline{\mu}^+$ will be introduced instead of $\underline{M}^+$ having a smaller domain in such a way that

$$\underline{x} = \mathrm{sig}\left(\underline{M}^+\right) = \mathrm{sig}\left(\underline{\mu}^+\right) \tag{48}$$

still holds, i.e. the logic solution remains unaltered.

Lemma 1 *For any matrices (or as special case vectors) $\underline{\underline{X}}$ and $\underline{\underline{Y}}$:*

$$\mathrm{sig}\left(\underline{\underline{X}} \cdot \underline{\underline{Y}}\right) = \mathrm{sig}\left(\mathrm{sig}\left(\underline{\underline{X}}\right) \cdot \mathrm{sig}\left(\underline{\underline{Y}}\right)\right) \tag{49}$$

Lemma 2 () *The occurance vector of the token flow into the individual places can be rewritten to the following form, when the T-invariant is composed as a linear combination of basis vectors with coefficients in $\underline{\alpha}$:*

$$\underline{x} = \tag{50}$$
$$\mathrm{sig}\left(\underline{M}^+\right) = \tag{51}$$
$$\mathrm{sig}\left((\underline{\underline{W}}^+)^T \cdot \underline{\underline{B}} \cdot \underline{\alpha}\right) = \tag{52}$$
$$\mathrm{sig}\left(\mathrm{sig}\left((\underline{\underline{W}}^+)^T\right) \cdot \mathrm{sig}\left(\underline{\underline{B}}\right) \cdot \underline{\alpha}\right) = \tag{53}$$
$$\mathrm{sig}\left(\underline{\underline{A}} \cdot \underline{\alpha}\right) = \tag{54}$$
$$\mathrm{sig}\left(\underline{\mu}^+\right) \tag{55}$$

where

$$\underline{\underline{A}} = \mathrm{sig}\left(\mathrm{sig}\left((\underline{\underline{W}}^+)^T\right) \cdot \mathrm{sig}\left(\underline{\underline{B}}\right)\right) \tag{56}$$

is a binary matrix and

$$\underline{\mu}^+ = \underline{\underline{A}} \cdot \underline{\alpha} \tag{57}$$

is a vector over $\mathbb{N}_0$.

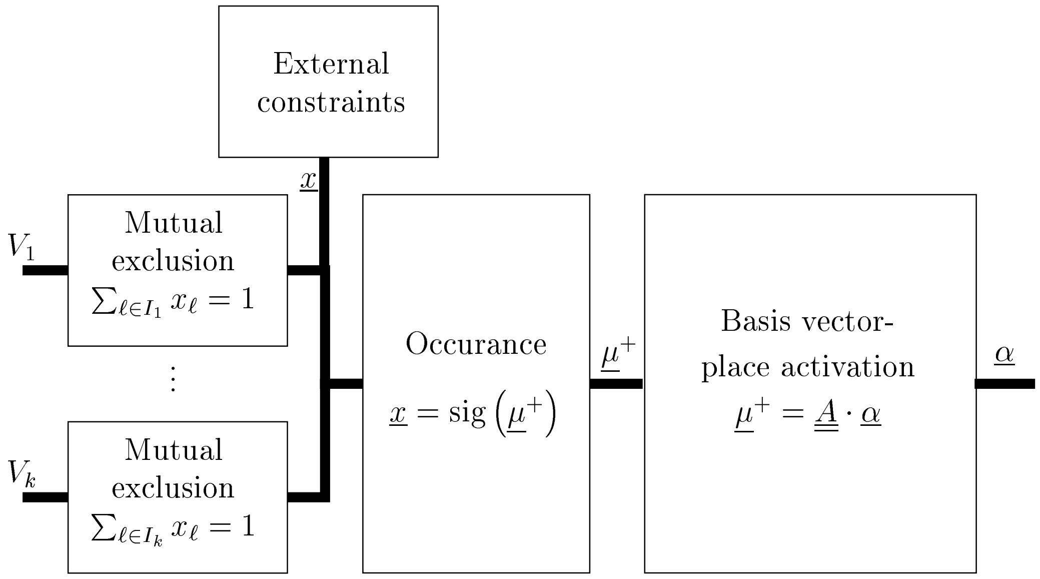

External constraints
$\underline{x}$
V_1
Mutual exclusion
$\sum_{\ell \in I_1} x_\ell = 1$
$\vdots$
V_k
Mutual exclusion
$\sum_{\ell \in I_k} x_\ell = 1$
Occurance
$\underline{x} = \mathrm{sig}\left(\underline{\mu}^+\right)$
$\underline{\mu}^+$
Basis vector-place activation
$\underline{\mu}^+ = \underline{\underline{A}} \cdot \underline{\alpha}$
$\underline{\alpha}$

Figure 14: Constraint structure after the first phase of the domain reduction

This step which seems to be a purely technical one at the first glance plays an important role in the further reduction steps. Now, the occurance vector can be estimated as the sign of the product of a binary matrix with a binary vector[5](Fig. (14)).

[5]Note that the sign of product of matrices expression on the right hand side of Equ. (56) can be estimated simply by performing the multiplication with the operations of logic AND as multiplication and OR as addition over the component sign matrices, as well. This observation can help to reduce the computation time needed.

It is easy to prove that

Lemma 3 *If $\underline{a}$ and $\underline{b}$ are binary vectors of dimension d, i.e. $\underline{a}, \underline{b} \in \mathbb{B}^d$, then for their scalar product $z = \underline{a}^T \cdot \underline{b} = \sum_{\ell=1}^{d} a_\ell \cdot b_\ell$*

$$\operatorname{sig}(z) = \tag{58}$$

$$\operatorname{sig}\left(\underline{a}^T \cdot \underline{b}\right) = \tag{59}$$

$$\operatorname{sig}\left(\sum_{\ell=1}^{d} a_\ell \cdot b_\ell\right) = \tag{60}$$

$$\bigvee_{\ell=1}^{d} (a_\ell \wedge b_\ell), \tag{61}$$

where $\vee$ and $\wedge$ denote the logic OR and AND operations respectively.

Corollary 1 (Reduction of the matrix $\underline{\underline{A}}$) *A simple generalization of the previous lemma shows that $\underline{x} = \operatorname{sig}\left(\underline{\mu}^+\right) = \operatorname{sig}\left(\underline{\underline{A}} \cdot \underline{\alpha}\right)$ is nothing else, than the logic sum (OR operation) of the columns of $\underline{\underline{A}}$ in which positions $\underline{\alpha}$ has a value of 1. This way all columns can be omitted from $\underline{\underline{A}}$, which:*

1. *can be generated as a linear combination of other columns or;*
2. *contradict to some of the mutual exclusion constraints;*

Naturally, $\underline{\alpha}$ has to be reduced to the proper dimension after deleting the columns from $\underline{\underline{A}}$.

Proof: 1. If a column $\underline{A}_\ell$ of $\underline{\underline{A}}$ can be written as a linear combination of the other columns with an $\underline{\alpha}'$ in the form of

$$\underline{A}_\ell = \sum_{\jmath=1}^{c} \underline{A}_\jmath \cdot \alpha'_\jmath, \tag{62}$$

where $\alpha'_\ell = 0$, and an arbitrary $\underline{x} = \operatorname{sig}\left(\sum_{\jmath=1}^{c} \underline{A}_\jmath \cdot \alpha_\jmath\right)$ for some α, then

$$\underline{x} = \tag{63}$$

$$\operatorname{sig}\left(\sum_{\jmath=1}^{c} \underline{A}_\jmath \cdot \alpha_\jmath\right) = \tag{64}$$

$$\bigvee_{\jmath=1}^{c} \left(\underline{A}_\jmath \wedge \alpha_\jmath\right) = \tag{65}$$

$$\left\{\alpha_\ell \wedge \left[\bigvee_{\substack{\jmath=1\\ \jmath\neq\ell}}^{c} \left(\underline{A}_\jmath \wedge \alpha'_\jmath\right)\right]\right\} \vee \left[\bigvee_{\substack{\jmath=1\\ \jmath\neq\ell}}^{c} \left(\underline{A}_\jmath \wedge \alpha_\jmath\right)\right] = \tag{66}$$

$$\bigvee_{\substack{\jmath=1\\ \jmath\neq\ell}}^{c} \underline{A}_\jmath \wedge \left[\left(\alpha_\ell \wedge \alpha'_\jmath\right) \vee \alpha_\jmath\right] \tag{67}$$

2. If some column $\underline{\underline{A}}_\ell$ of $\underline{\underline{A}}$ does not satisfy the constraints, no $\underline{\alpha}$ belonging to the feasible space can have in the corresponding position $\alpha_\ell = 1$, otherwise the mutual exclusion constraint would be violated by the obvious property of the logic OR relation. Thus in all solutions $\alpha_\ell = 0$, and both $\underline{\underline{A}}_\ell$ and the ℓ-th position of $\underline{\alpha}$ can be omitted.

Q.E.D.

Now, after the deletion of the superfluous columns of $\underline{\underline{A}}$ and positions in $\underline{\alpha}$, the constraint structure remains the same, as depicted in Fig. (14), but with a new matrix $\underline{\underline{A}}^*$ and $\underline{\alpha}^*$ of a reduced dimension c^*. Naturally, for any $\underline{\alpha}^*$ the new

$$\underline{\mu}^* = \underline{\underline{A}}^* \cdot \underline{\alpha}^* \tag{68}$$

may differ for the corresponding $\underline{\mu}^+$ in value, but their sign vectors remain equal.

6.2.3 Formulation of the occurance vector

As a result of the previous steps, the occurance vector can be re-written in the following form:

$$\underline{x} = \mathrm{sig}\left(\underline{\underline{A}}^* \cdot \underline{\alpha}^*\right). \tag{69}$$

The next step constitutes of the formulation of the inequality system corresponding to the sign function. Remember that for each element of the occurance vector these constraints are of the following form (c.f. Equ. (16)-20).

$$x_\ell \in I\!B \tag{70}$$

$$\mu_\ell^* \geq x_\ell \tag{71}$$

$$\mu_\ell^* \leq x_\ell S_\ell \tag{72}$$

In order to formulate the third inequality, the upper bound on the firing counts is to be estimated.

Corollary 2 Upper bound on the firing counts
The upper bounds on the firing counts required for inequalities 72 are defined by substituting $\underline{\alpha}^* = \underline{1}_c^*$ into Equ. (68), where $\underline{1}$ denotes the all-one vector of dimension c^*, i.e.

$$\underline{S} = \underline{\underline{A}}^* \cdot \underline{1}_c^*, \tag{73}$$

This way the constraint system to be satisfied gains the following form:

$$\underline{x} \in \mathbb{B}^q \tag{74}$$

$$\underline{\alpha}^* \in \mathbb{B}^{c^*} \tag{75}$$

$$\underline{\underline{A}}^* \cdot \underline{\alpha}^* \geq \underline{x} \tag{76}$$

$$\forall \ell \in \{1, \ldots, q\} : \left(\underline{\underline{A}}^* \cdot \underline{\alpha}^*\right)_\ell \leq \underline{x}_\ell \cdot \left(\underline{\underline{A}}^* \cdot \underline{1}\right)_\ell , \tag{77}$$

where the lower index ℓ refers to the ℓ-th element of the corresponding vector. This way the structure of the inequality system to be solved becomes to that depicted in Fig. (15).

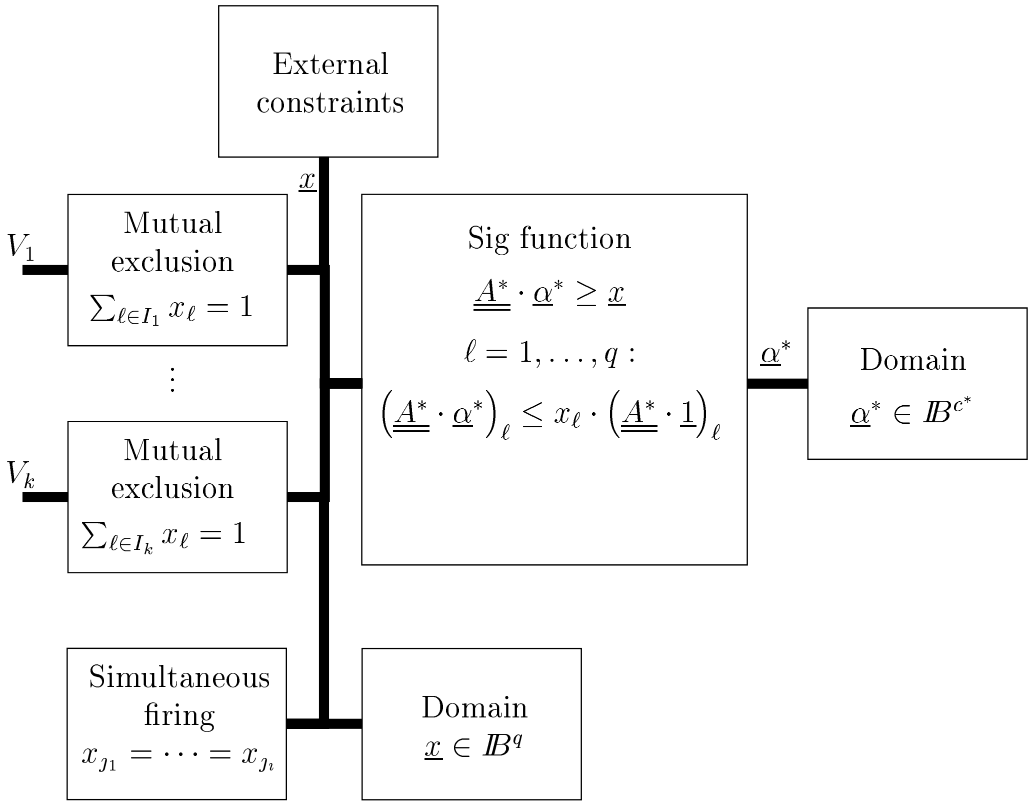

External constraints
$\underline{x}$
V_1
Mutual exclusion
$\sum_{\ell \in I_1} x_\ell = 1$
V_k
Mutual exclusion
$\sum_{\ell \in I_k} x_\ell = 1$
Sig function
$\underline{\underline{A}}^* \cdot \underline{\alpha}^* \geq \underline{x}$
$\ell = 1, \ldots, q :$
$\left(\underline{\underline{A}}^* \cdot \underline{\alpha}^*\right)_\ell \leq x_\ell \cdot \left(\underline{\underline{A}}^* \cdot \underline{1}\right)_\ell$
$\underline{\alpha}^*$
Domain
$\underline{\alpha}^* \in \mathbb{B}^{c^*}$
Simultaneous firing
$x_{\jmath_1} = \cdots = x_{\jmath_\imath}$
Domain
$\underline{x} \in \mathbb{B}^q$

Figure 15: The structure of the linear inequality system to be solved

6.2.4 The solution algorithm for the extended model

Algorithm 2 () *The constrained Petri net model can be solved by the following steps:*

1. Construct the uninterpreted constraint model of the system
2. Estimate the T-invariants over $\mathbb{Q}_0$
3. Determine the token flow matrix into the individual places
4. Reduce the basis

5. Estimate the maximal value of the firings
6. Solve the inequality system, describing the mutual exclusion, eventual simultaneous firing and other external constraints.

6.3 Implementation

The constrained Petri net model is built automatically from dataflow descriptions. The T-invariants are estimated using a C language implementation of the Martinez-Silva algorithm[8]. The Petri net is represented by a tableau matrix. All the other steps are implementations of the ideas of this section. The program prints the resulting inequality system. The solution is postponed to other programs. The solution process is described in Section (8).

Performance measurements were conducted on the dataflow components of the MEMSY multiprocessor (see Section (5.4)). Again, the size of the output model — the inequality system — is most important. The running time and memory usage of the transformation must also be considered, for the algorithm contains several non-trivial reduction steps.

6.3.1 Model sizes

Fig. (3) shows the sizes of the original and the transformed models for selected components.

Component	**Petri net model**				**Inequality system**	
	Place	**Trans.**	**Arc**	**Constraint**	**Variable**	**Constraint**
S_arb	2543	4919	17724	1971	4919	7056
Cpu	782	1494	4507	594	1494	2157
Disc	94	167	383	28	167	215
Reset	66	110	179	56	110	187
Term	30	46	178	14	46	73

Table 3: Sizes of the Petri net models and the inequality systems

One can see from the examples that the transformation reduces the problem size indeed. It did not seem worthwhile to conduct more measurements, for the automatically generated Petri net description and — to a lesser extent — the generated inequality system is redundant. Fair measurements, where the original dataflow model is compared to the preprocessed inequality system, are given in Section (8.5).

6.3.2 Efficiency

The worst case efficiency of the algorithm is exponential in both time and memory as there can be an exponential number of minimal support T-invariants[8]. In practice, however, this part of the algorithm performs much better.

In the following, execution times and memory usage will be measured for the MEMSY components and bottlenecks will be identified.

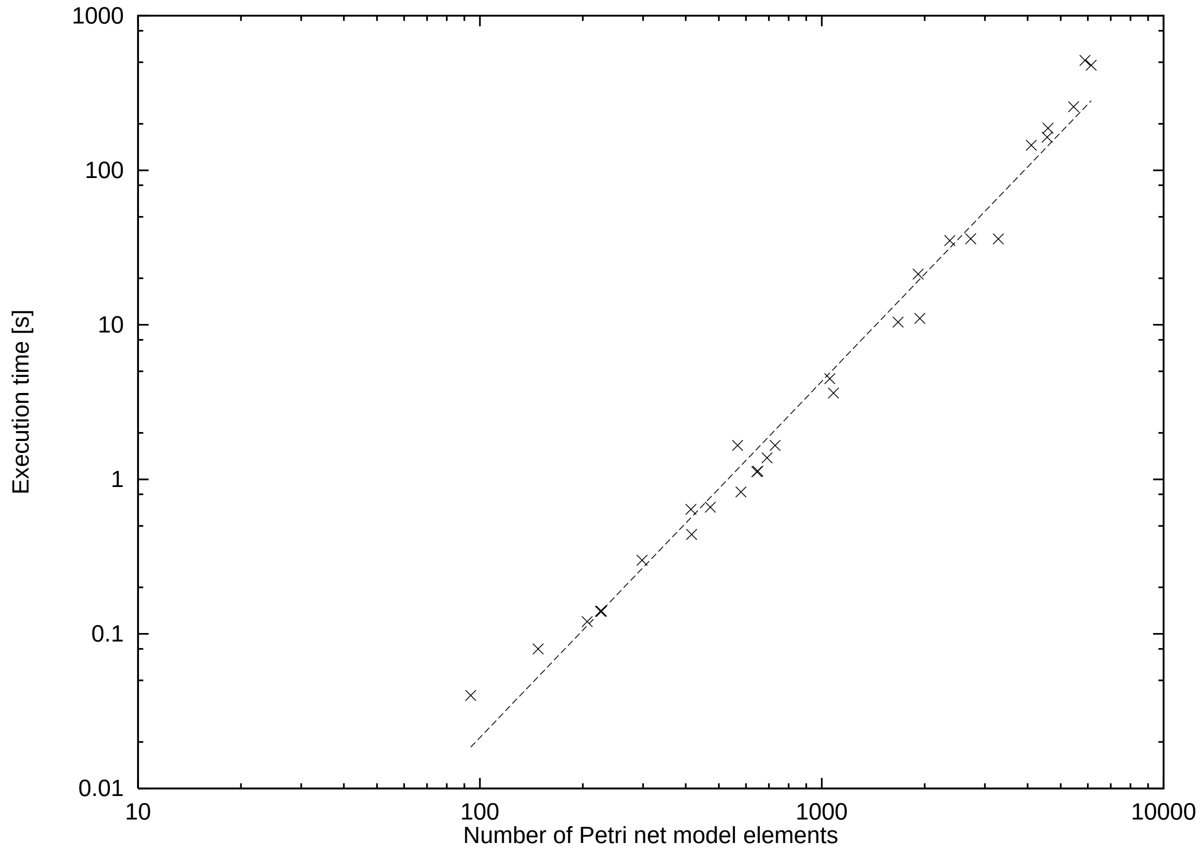


Figure 16: Execution times vs. number of variables

Fig. (16) shows execution times measured on an SGI workstation with a 250 MHz IP22 processor. The fitted curve is $c{\cdot}x^{2.3044}$, but this value is not to be relied on when estimating the complexity for several reasons:

- The algorithm will be significantly improved (see below).
- The size of the input is a very rough estimate of the difficulty of the problem. The components used vary greatly in the kind of their Petri net models: there are ones with excessively high numbers of arcs and ones with "sparse" structures, for example.

- The Petri net model is redundant, and the majority of reduction steps will only be performed after the transformation.

Most of the time is spent in the Martinez-Silva algorithm and in the matrix multiplication performed to obtain the matrix of the token flow into the individual places. As the incidence matrix of the Petri net, the T-invariant vectors and the operands of the multiplication are sparse, the current tableau implementation is clearly inefficient and significant improvements can be expected from a sparse implementation.

Memory usage is relatively high. This comes from the fact that the incidence matrix and the T-invariant vectors must be stored in memory. Again, much is to be expected from a sparse implementation.

7 Refined model for diagnostics

The method elaborated by Portinale and extended in Section (4) reflects only the properties of fault propagation without concerning other important factors influencing the behavior of the UUT.

A general purpose system model will be presented in this section at first. Subsequently, the mathematical formulation the most basic questions formulated during diagnosis using this model will be presented. Finally, some more possible extension ideas are mentioned.

7.1 Model of the system behavior

The most important factors contributing to the diagnostics are typically the following ones:

System input, like the selection of test or operational input sequences applied during the observation period; which error detection mechanism are enabled during this period, etc.

The structure and fault model of the system. For simplicity, the – still general – assumption is postulated that the system is decomposed into N components. An own elementary fault set is associated to each individual element $\imath$. Each component can be in one of this fault states or in the fault-free state, thus forming the fault state subspace $\mathcal{F}_\imath$. Note that multiple physical faults can be modeled by introducing a separate fault state for each combination. In the system any number of faults is allowed.[6]

The syndrome observed serving as information source on the behavior of the system. It can be thought of as a key to a part of the observable subset of the fault states.

The fault propagation chain can be described in several ways: by a dataflow diagram, by its Petri net model, as described in Section (3.3.1) or by an inequality system. In fact, the previous sections of the report are concerned with modeling with these descriptions and transforming one into another.

Note the similarity of this system model with a dataflow component. System input corresponds to input channels, the syndrome to the output channels and the fault states in the fault model to the state variable. The fault propagation chain describes the system behaviour, thus it is the counterpart of the firing rules. All this is shown in Fig. (17).

[6]It should be pointed out that the restrictions on the fault model described above serve only an easier understanding. All further results and algorithms can be extended in a straightforward way to incorporate any fault combinations within a component and correlated faults affecting multiple components as well.

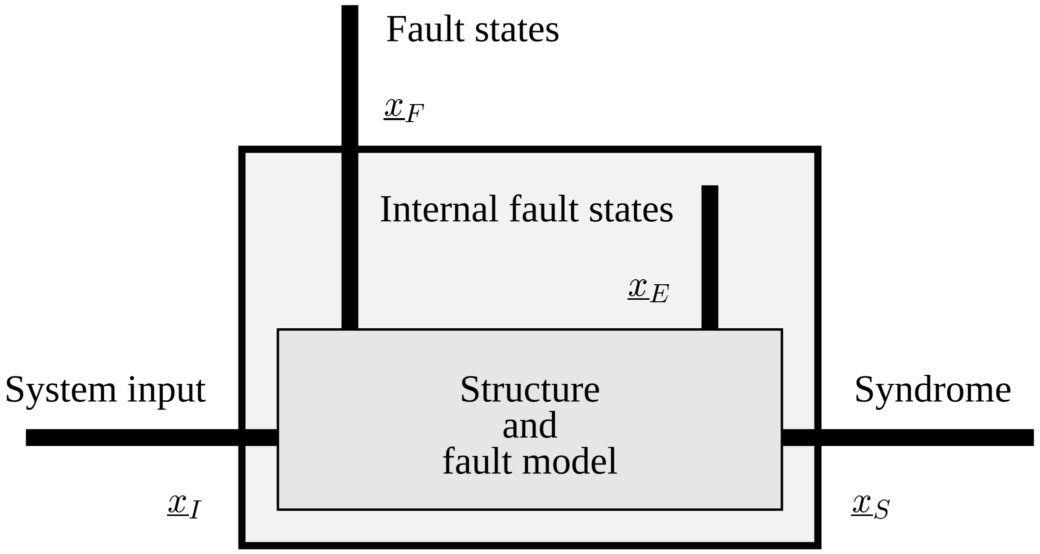


Figure 17: The system model as a dataflow component

The system variable descriptors can be subdivided into the same form groups as described above, by selecting a proper index ordering of the model elements of the fault propagation chain. They should be called

$$\underline{x} = [\underline{x}_I, \underline{x}_F, \underline{x}_E, \underline{x}_S] \tag{78}$$

respectively in the above order. For instance, in the final inequality system model they are the elements of the occurance vector of the token flow into the individual places belonging to a T-invariant of the Petri net model.

The subsets of the occurance vector allowed by the system model are denoted by $\mathcal{X}_I$, $\mathcal{X}_F$, $\mathcal{X}_E$ and $\mathcal{X}_S$, respectively.

Note that for the diagnostic engineer only the occurance vector has a direct meaning, as the constraints in the model of the fault propagation chain are only necessary to the solution of the constraint system, and are more or less illegible to humans — the more automatic transformations were performed, the more illegible they are. Similarly, the internal variables $\underline{x}_E$ — corresponding to internal error manifestations — are only of a secondary importance, as they are used only in a detailed analysis of the fault effect propagation mechanism.

Accordingly, in the external view of the system, its behavior is defined by the mapping

$$(\underline{x}_I, \underline{x}_F) \rightarrow \underline{x}_S \tag{79}$$

or more precisely the *logic diagnosis constraint*

$$\Re_{\mathcal{L}} (\underline{x}_I, \underline{x}_F, \underline{x}_S) \tag{80}$$

describing the relation between them.

The notion introduced in Equ. (79) can be extended for subsets as well in the following form:

Let $\mathcal{X}_I^{'}\subseteq\mathcal{X}_I, \mathcal{X}_F^{'}\subseteq\mathcal{X}_F,$ and $\mathcal{X}_S^{'}\subseteq\mathcal{X}_S$.

$$\Re_{\mathcal{L}}\left(\mathcal{X}_I^{'}, \mathcal{X}_F^{'}, \mathcal{X}_S^{'}\right) \text{ iff } \forall \underline{x}_I \in \mathcal{X}_I^{'}, \forall \underline{x}_F \in \mathcal{X}_F^{'}, \forall \underline{x}_S \in \mathcal{X}_S^{'} : \Re_{\mathcal{L}}\left(\underline{x}_I, \underline{x}_F, \underline{x}_S\right) \tag{81}$$

is satisfied.

7.2 Diagnostic problems

The major diagnostic problems one can investigate with the help of this model can be grouped according to which ones of the three vectors or their subsets are known in the logic diagnosis constraint. This is summarised in Fig. (4).

Input	**Faults**	**Syndrome**	**Operation**
$\underline{x}_I$	$\underline{x}_F$	?	Fault simulation
$\underline{x}_I$	$\mathcal{X}_F$	?	Parallel fault simulation
$\underline{x}_I$	?	$\underline{x}_S$	Fault diagnosis
$\underline{x}_{I1}, \ldots, \underline{x}_{Iy}$	?	$\underline{x}_{S1}, \ldots, \underline{x}_{Sy}$	Sequential diagnosis
$\underline{x}_I$	?	$\underline{x}_S$ (partial)	Partial diagnosis
$\underline{x}_I$	?	$\underline{x}_{S0}$ (fault free)	Estimation of non-detected faults
?	$\underline{x}_F$	$\underline{x}_S$	Test generation
			Compensation of error latency
			Fail safe testing

Table 4: Applications of the system model

7.2.1 Known input and fault state, unknown syndrome

Fault simulation estimates the maximal $\mathcal{X}_S^{'}$ satisfying $\Re_{\mathcal{L}}\left(\underline{x}_I, \underline{x}_F, \mathcal{X}_S^{'}\right)$.

The syndrome set $\mathcal{X}_S^{'} \subseteq \mathcal{X}_S$ compatible with the given input $\underline{x}_I$ and fault $\underline{x}_F$ is determined by using the logic diagnosis constraint in the forward direction. After substituting these vectors into the Diophantine inequality system all its solutions can be estimated. This process corresponds to the simulation of a single fault in logic testing.

Parallel fault simulation searches for $\mathcal{X}_S' : \Re_{\mathcal{L}}\left(\underline{x}_I, \underline{x}_F, \mathcal{X}_S'\right)$

When $\underline{x}_F$ is unbounded, i.e. any $\underline{x}_F \in \mathcal{X}_F$ can occur, then the entire subset of syndromes is to be estimated, which can occur, when applying the input $\underline{x}_I$ in a similar fashion, as parallel fault simulation [7] does it at the logic level.

7.2.2 Known input and syndrome, unknown fault state

Fault diagnosis: The most typical diagnostic task is the estimation of the fault hypotheses consistent with the given input $\underline{x}_I$ and syndrome $\underline{x}_F$. The Petri net based model delivers all feasible hypotheses, as described earlier.

Sequential diagnosis: A very frequently used approach performs diagnosis in a sequential way, thus multiple tests are successively applied to the UUT, and the corresponding syndrome sequence is processed. It is assumed that the inputs applied during different measurement time frames do not interfere, i.e. the syndrome resulting in a particular observation time frame is independent of the actions taken during the previous frames. When assuming the stationarity of the fault state during the entire measurement series of a length of y observations, the set of the resulting fault hypotheses $\mathcal{H}$ consistent with all syndromes is the intersection of the hypotheses $\mathcal{H}_1, \ldots, \mathcal{H}_y$ consistent with the individual syndromes $\underline{x}_S^1, \ldots, \underline{x}_S^y$.

$$\mathcal{H} = \bigcup_{\ell=1}^{y} \mathcal{H}_\ell \tag{82}$$

This case has a simple formulation by using the model developed before.

For each observation time frame $\ell \in \{1, \ldots, y\}$ the input and syndrome vectors $\underline{x}_I^\ell$, and $\underline{x}_S^\ell$ are known. The elements of the error occurance vector $\underline{x}_E^\ell$ are free variables. Due to the assumed stationarity of the fault state in each time frame the same fault occurance vector is to be assigned[8]. The final diagnosis is the joint solution of the system constraint

$$\begin{array}{ll} \Re_{\mathcal{L}} & (\underline{x}_I^1, \underline{x}_F, \underline{x}_S^1) \\ & \vdots \\ \Re_{\mathcal{L}} & (\underline{x}_I^\imath, \underline{x}_F, \underline{x}_S^\imath) \\ & \vdots \\ \Re_{\mathcal{L}} & (\underline{x}_I^o, \underline{x}_F, \underline{x}_S^o)\,, \end{array}$$

[7]The adjective "parallel" expresses here the simultaneous processing of the effects of all faults and it is not intended to refer to a particular simulation technique, like parallel deductive or concurrent fault simulation.

[8]If don't care variables can appear in the occurance vector, then the constraint has to be constructed according Section (4.2.6).

containing y times as much inequalities, as a single frame logic system constraint. Note that the number of free variables is

$$\dim(\underline{x}_F) + y \cdot [\dim(\underline{x}_I) + \dim(\underline{x}_E)] \,. \tag{83}$$

The resulting effect shows a virtually nearly quadratic increase in the problem size, however many steps, like the estimation of the solution of the unconstrained system are still identical during the necessary calculations.

Partial diagnosis In numerous technical applications not all the syndromes are collected in each time frame, thus the diagnosis process has to deal with incomplete information as well. This can be performed by a simple relaxation of constraints belonging to unobserved syndrome elements. Note that the simple assignment of the unknown value to such syndrome variables were too restrictive, as it would exclude from the set of candidate fault hypotheses such a fault, which would definitely trigger a failure indication, even if that remains unobserved.

Estimation of non-detected faults A crucial task of the test design process is the determination of the set of faults still undetected by the till yet generated test input sequence. Such an examination controls not only the test generation process, but it delivers some basic qualitative characteristics such as the fault coverage.

Exactly those faults remain undetected by an input sequence $\langle \underline{x}_I^1, \ldots, \underline{x}_I^\ell \rangle$, which result in the same syndrome sequence $\langle \underline{x}_{s_0}^1, \ldots, \underline{x}_{s_0}^\ell \rangle$, as the fault-free system characterized by the occurance vector $\underline{x}_{F0}$. As a deterministic behavior of the fault-free system is assumed, the syndrome sequence is unique. This way the same algorithm, as used for sequential diagnosis, applied for this syndrome sequence delivers the set of all undetected faults.

7.2.3 Known fault state and syndrome, unknown input

At the first glance, this case seems to be a not very meaningful one, as usually it can be assumed that the environment is known during the observation of the failures. However, there are three important cases, where this model finds an important use:

Test generation Here one intends to find those combinations of input which trigger the activation of a certain type of fault, so that its presence can be observed in the syndrome.

Compensation of error latency Frequently, the effects of an error appear as failures on after such a long latency that an inference from the failure mode to the fault becomes rather difficult. For instance, if the use of a fault produces a latent error stored in the system somewhere, then the failure can appear already in such a phase of the operation, where the faulty resource is not used anymore. In such cases the estimation of the set of input combination can be used as a guiding heuristics for the fault isolating test strategy.

Fail safe testing When diagnosing faulty systems, one of the main requirements is that the diagnostic process should avoid further catastrophic failures both in the sense of follow-up physical defects or data losses. For instance, in a faulty computer usually no test action can be started after a crash, which involves a danger of losing disk data which is not backed up yet. The inhibited test combinations can be estimated with the diagnostic model described above, by including into the set of syndromes those, which correspond to the initiation of a fatal action.

7.3 Further possibilities for model refinement

The diagnostic model described above can be further refined by introducing any additional factor influencing the diagnostic process. For instance, when introducing the notion of resources, the mutual exclusion relation limiting the use to a single test at a time can be easily included in to model as an external constraint. Similarly, complex interferences between faults and test can be modeled as well. A test invalidation relation, like that used in the Russel-Kime type of model requires not even the introduction of a new model attribute at all.

8 Optimal solutions of diagnostic problems

The formulation of the diagnostic problem in the form of Diophantine equations or inequalities describing the constraints between the system structure and observations allows for an efficient estimation of all solutions. However, the set of all candidate diagnostic hypotheses grows frequently so large that it becomes practically unmanageable. In this case, typically a single or a few of solutions are searched for, which fulfill some optimization criteria.

The introduction of such decision criteria offers new possibilities allowing optimization of the diagnostic process as well, using standard methods of the operation research allowing for the solution of problems of a huge dimension.

This special chapter of operation research — integer programming — plays a key role in integer and combinatorial optimization. The basic definitions used in this field will be summarised first. The formulation of some basic diagnostic problems in the form of linear integer programs will be presented in the following sections. Finally, measurements on the compactness of the formulation and the performance of the experimental implementation are presented.

8.1 Linear integer programming

The general *linear integer programming problem* has the form of:

$$\max\{\underline{c}^T\underline{x} : \underline{\underline{A}}\underline{x} \leq \underline{b}, \underline{x} \in I\!N_0^n, \underline{c} \in I\!R^n, \underline{\underline{A}} \in I\!R^{n\times m}, \underline{b} \in I\!R^m\} \tag{84}$$

The set defined in Equ. (84) over which the maximum is searched for is called the *feasible region*, and an $\underline{x}$ in it a *feasible solution.* The function $\underline{c}^T\underline{x}$ is called *the objective function.*

Note that an equality constraint can be described by simultaneously introducing two inequalities for the "less or equal" and the "greater or equal" constraint. A minimization problem can be formulated as the maximization of the negate objective function. The modeler may use certain non-linear constructs in his model, those which can be transformed to some linear constructs so that the problem size does not increase in an unacceptable manner.

An important subclass of integer programming problems are the 0-1 IP problems. Here variables are constrained to be equal to 0 or 1; in the definition, $\underline{x} \in I\!N_0^n$ is replaced by $\underline{x} \in I\!B^n$. For *combinatorial optimization problems* there is no generally accepted definition, but most such problems are 0-1 IPs that deal with finite sets and collections of subsets.

8.1.1 Solution

This section only aims at providing an overview. For more thorough handling of the subject numerous textbooks are available, such as [15] or those mentioned in [16].

Linear integer programs are hard to solve, in general they are in the class of NP-complete problems, in contrast to linear programs, which are in P. Most general-purpose large-scale IP codes use a procedure called *branch-and-bound* to search for an optimal integer solution and to prove its optimality. This procedure works by solving a sequence of related linear programming *relaxations* of the original problem, problems from which the integrality constraint $\underline{x} \in I\!N_0^n$ was removed. Good codes distinguish themselves primarily by solving shorter sequences of LPs, and secondarily by solving the individual LPs faster. They also exploit that the similarities between successive LPs in the search tree can speed up the LP solution process considerably.

A different solution approach known generally as *constraint logic programming* has drawn increasing interest of late. Having their roots in studies of logical inference in artificial intelligence, CLP codes typically do not proceed by solving any LPs. As a result, they search "harder" but faster through the tree of potential solutions. Their greatest advantage lies in their ability to tailor the search to many constraint forms that can be converted only with difficulty to the form of an integer program; their greatest success tends to be with "highly combinatorial" problems such as scheduling, sequencing, and assignment, where the construction of an equivalent IP would require the definition of large numbers of 0-1 variables.

Another burden to solving IPs is that the difficulty of any particular IP problem is hard to predict in advance. Problems in no more than a hundred variables can be challenging, while others in tens of thousands of variables solve readily. The best explanations of why a problem is difficult often rely on some insight into the modeled system and tend to appear only after running a lot of computational tests. A related observation is that the way of model formulation can be more crucial than the choice of the algorithm and the solver. Thus a large scale IP problem should be approached with a certain degree of caution and patience. A willingness to experiment with alternative formulations and with an IP code's numerous search options often pays off in greatly improved performance.

In the hardest cases, it may be worthwhile abandoning the goal of a provable optimum. By terminating a IP code prematurely, one can often obtain a high-quality solution along with a provable upper bound on its distance from optimality. Indeed, it may be an optimal solution,as procedures for IP may not be able to prove optimality of a solution until long after it has been found[9]. This choice also opens up a broad area of approximate methods, probabilistic methods and heuristics, as well as modifications to branch-and-bound.

[9]The problem of determining whether an IP has an objective value less than a given target is NP-complete.

Subclasses of IP problems may be much easier to solve. There are certain models whose LP solution always turns out to be integer, assuming the input data consists of integers[10] The best-known and most widely used models of this kind are the *pure network flow* models. The network linear programming problem is to minimize the — linear — total cost of flows along all arcs of a network, subject to conservation of flow at each node, and upper and/or lower bounds on the flow along each arc. It is a subclass of LPs. They can be solved much faster than general LPs of the same size, by use of specialized optimization algorithms, such as the *network simplex method.* Latter has the property that — if given integer data — it will return integral optimal flows.

Unfortunately, many different network problems of practical interest do not have a formulation as a network LP. Contrary to many people's intuition, the statement of a hard problem may be only marginally more complicated than the statement of some easy problem.

8.1.2 Modeling

As already highlighted in the previous section, formulating a "good" model is of crucial importance to solving the model. What "good" means is dealt with in several books[15, 14]; here only some aspects of the problem are shown.

Usually, there is a huge number of choices of formulating a model for a given problem. Typically, defining the variables and the objective function is the first question addressed. This often derives from the unknown parameters of the desired solution. Thus the most important choice is defining the constraints, i.e. $\underline{\underline{A}}$ and $\underline{b}$, in an appropriate way. Two observations are presented here:

- Most integer programming algorithms require an upper bound on the value of the objective function, and their efficiency is very dependent on the sharpness of the bound. An upper bound is determined by solving the LP relaxation of the IP. Thus the closer the objective value of the IP to that of the LP, the better.
- It is instinctive to believe that computation time increases as the number of constraints increases. But trying to find a formulation with a small number of constraints is often a bad strategy. In fact, one of the main algorithmic approaches involves the systematic addition of constraints, so-called *cutting planes.*

8.1.3 Preprocessing

Preprocessing refers to elementary operations that can be performed to impove or simplify a formulation. It can be thought of as a phase between formulation and solution. It can greatly enhance the speed of a sophisticated solution algorithm.

[10] In general these models have a "unimodular" constraint matrix of some sort.

Sometimes, small problems can be solved with just preprocessing or preprocessing combined with a kind of enumeration. It has been the general opinion of researchers, though, that usually there is a need for both a preprocessing phase and a sophisticated solution phase.

8.2 Formulation of diagnostic problems

Some recurring problems in diagnostics are presented in this section along with their integer programming formulations. They are summarized in Fig. (5). The comparison with Fig. (4) might be interesting. Note that the constraints of the problems can be produced with the algorithm of Section (6) and by formulating the ideas of Section (7); here only objective functions are added and explained.

Input	**Faults**	**Syndrome**	**Objective function**	**Operation**	
$\underline{x}_I$	?	$\underline{x}_S$	$\sum_{u \in F_{\text{fault free}}} \underline{x}_{Fu}$	Maximum likelihood diagnosis	Minimal number of faults
			$\sum_{u \in F} P(t) \cdot \underline{x}_{Fu}$		Probabilistic diagnosis
			$-$ checking costs		Practice oriented testing
?	? $\sum_{u \in F} \underline{x}_{Fu} \leq t$	?	Deviation from the reference	Validation of fault tolerance	

Table 5: Diagnostics with optimization problems

8.3 Maximum likelihood diagnosis

8.3.1 Minimal number of faulty elements

The probability of the occurrence of a fault state decreases with the number of faults within. In the case of independent faults, each of them is assumed to have a probability less than 0.5. Upon this assumption, the maximum likelihood diagnosis is achieved by a fault hypothesis, which is

- consistent with the syndrome assured by imposing the constraints;
- minimal in the terms of the number of faults involved into the generation of the syndrome.

The objective function consists simply the complement of the total number of faults. This can be expressed as the sum of the support vector bits of the source transitions corresponding to faults, i.e.

$$\sum_{t \in \mathcal{F} \setminus \mathcal{G}} x_F(t), \tag{85}$$

where $\mathcal{G} \subset \mathcal{F}$ denotes the set of the fault-state sink transitions corresponding to the fault-free states.

8.3.2 Probabilistic diagnosis

A further refined diagnosis optimization can be performed if the probabilities of the individual states in the fault state space are known. Maximal likelihood diagnostics is achieved in this case by simply selecting the fault state with maximal probability which is also consistent with the system constraints.

- If the system is not decomposed into units, then the fault occurance vector $\underline{x}_F$ describes directly the occurrence of the candidate faults (or fault combinations). Probabilities $P(f_\ell)$ are assigned directly to each individual fault $f_\ell \in \mathcal{F}$ and source transitions t_f representing them. The objective function to be maximized is the probability itself, thus

$$\sum_{t \in \mathcal{F}} x_F(t) \cdot P(t). \tag{86}$$

- If the system is decomposed to into disjunct fault state components, e.g. by decomposing the system into N components each one having its own fault state f_ell $\ell = 1 \dots N$ independently of the other ones, then the probability of a particular fault pattern $(f_1, \dots, f_N)$ is

$$P_{f_1,\dots,f_N} = \prod_{\ell=1}^{N} P_{f_\ell} \tag{87}$$

  The negative logarithm of the probability will be used instead of the probability itself in order to have a linear objective function. These two functions have their maxima simultaneously. Obviously,

$$-\log P_{f_1,\dots,f_N} = -\sum_{\ell=1}^{N} \log P_{f_\ell} \tag{88}$$

  The objective function is simply the scalar product of the occurance vector and a vector containing the negative logarithms of the individual faults. Note that the mutual exclusion constraints assure that for each component only a single element in the fault occurance vector has the value 1.

8.3.3 Practice oriented objective function

Both measures described above aim at the expression of the costs of testing in terms of some theoretical cost estimators. Different, explicitly financial cost oriented measures are used in the testing practice. Without recapitulating this to the full extent[11], some representative examples are the following ones:

- Average execution time of a test
- Costs of a test originating in instrumentation, working time of skilled personnel etc.

Note that all these costs can be expressed as a linear objective function weighted with the particular financial cost and the fault occurrence probability, thus an optimization can be performed using the very same method, as for the theoretical measures.

8.4 Validation of fault tolerance

Optimization methods are also an efficient means for verifying some properties of the entire fault set. It must be possible to guess whether the property holds from the objective value or the feasible solution belonging to the objective value. The latter can be used to identify elements violating the property.

Note that this is not a "classical" optimization problem in which the choice of the objective function is more or less fixed by the problem statement. In most cases, one can change the objective function, for instance to speed up the solution process. In other words, there is now yet another tunable parameter of the model and one has the choice of modelling parts of the model with either constraints or the objective function. This is the reason why the entries of the tasks in this section in Fig. (5) contain mostly generalities.

In the field of diagnostics, an important task is to validate the fault tolerance in the system. Interesting subproblems are e.g. to prove that no single point of failure exists or that the fault coverage of the test set used in a diagnosis algorithm is 100% under certain conditions.

A modeling possibility is to define the objective function as a measure of deviation from some reference, derived from the fault tolerance requirements of the specification. The fault set should be constrained by conditions that hold when the system is supposed to function properly. An example of such an assumption is the diagnostic *t-limit*:

$$\sum_{t \in F} \underline{x}_F(t) \leq t$$

The input and the syndrome should not be constrained and the fault set should not be constrainted further as it might be interesting to know

[11]A comprehensive overview of the modeling of the economic aspects in diagnostics can be found in [7].

- which occurance of faults cannot be tolerated;
- how to reproduce the inacceptable behavior;
- what is the reaction of the system to the situation.

Note that the implementation of a complicated objective function may require significant extensions to the model. In this example, "deviation from the reference" would probably be expressed partly by allowed value combinations of input and syndrome, by introducing additional variables and constraints.

8.5 Model compactness

8.5.1 The measurement method

Parts of the dataflow model of the MEMSY multiprocessor are under investigation once again; see Section (5.4). The system structure and fault model are built according to the algorithm in Section (6). An objective function and some more constraints must be added so that the power of the optimization approach can be investigated with the help of the components.

Let us find a suitable diagnostics task. It will be the estimation of non-detected faults in a system S; see Section (7.2.2). The inequality description of the system — the structure and fault model — is copied. The fault state variables of the copy S_R, $\underline{x}_{FR}$, are forced to the fault free state. This way, S_R acts as a fault-free reference, its input and output is only allowed to have legal combinations. Let the inputs and outputs of the two systems be equal. The objective function is maximizing the number of faults in S. The problem structure is shown in Fig. (18).

The system S is always a single dataflow component of the multiprocessor. Let the state variable of the component is taken as the only fault state. The first element of its domain is the *fault free* state. The higher the number of a state, the more severe is the related fault supposed to be. (One often has a state variable like {`cpu_ok_data_ok`, `cpu_ok_data_faulty`, `cpu_faulty_data_ok`, `cpu_faulty_data_faulty`}.) The following objective function will maximize the severity of the fault:

$$\max\{1 \cdot x_1 + 2 \cdot x_2 + 3 \cdot x_3 + \ldots + k \cdot x_k\}$$

where the domain of the state variable is $\mathcal{S} = \{s_0, s_1, \ldots, s_k\}$ and $x_i = 1$ if and only if the component is in state s_i.

The model is sincerely not the most relevant one when performing some diagnostics of the multiprocessor. It is believed, though, that the choice of the task is not of great importance at this early stage of measurements, therefore a simple, automatically generated

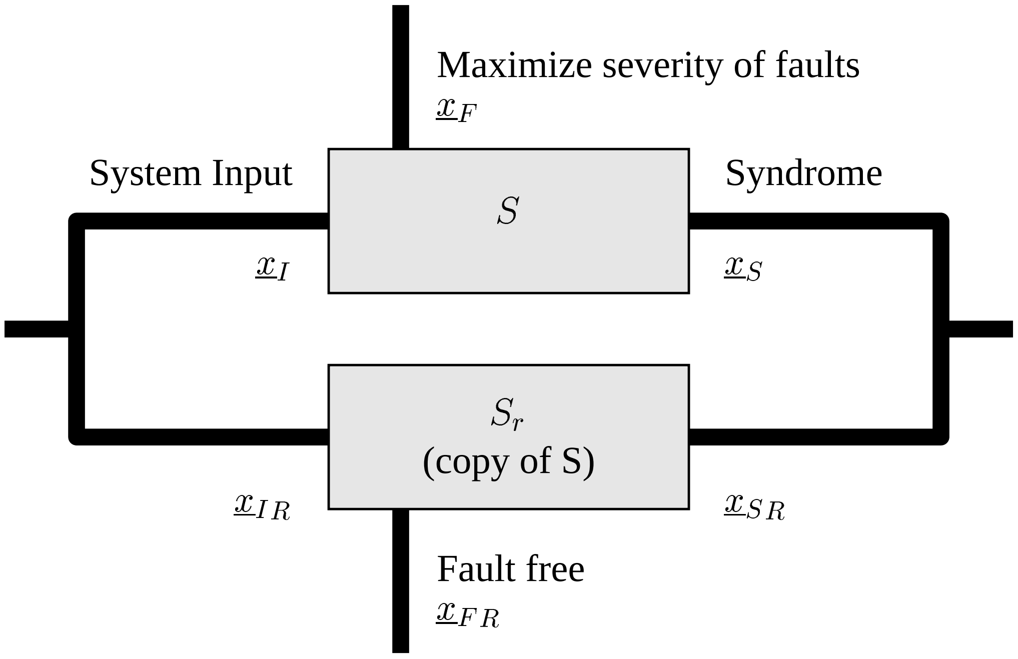


Figure 18: Integer programming model of the problem of non-detected faults

task is chosen. Also, the task does not have to be highly realistic. Note also that the task dependent additions form a relatively small part of the problem, compared with the formulation of the fault model, hence they are not expected to render the measurements useless. This is especially true when looking at the preprocessing phase, where only a small part of the problem is considered at a time.

The next sections measure the compactness of the model, by preprocessing the problem and comparing its size with that of the original dataflow model and the intermediate models. The problems will be solved in the section on performance measurements, Section (8.6).

As IP preprocessor and solver, the codes OPBDP and CPLEX are used. They are described in Section (9.3.6) and Section (9.3.5).

8.5.2 Results

Fig. (6) shows the effect of preprocessing on the model sizes for selected components – see Section (5.4). "Term" is the number of terms in both the constraints and the objective function. The running time is also shown.

On the way from the dataflow formulation to the integer programming formulation, the automatic model transformations introduced a lot of redundancy. This prevented meaningful comparisons of model sizes. But the comparison of the sizes of the original dataflow models with the sizes of the preprocessed integer programming models can be done now,

Component	Original model			OPBDP preprocessed model			
	Var.	Constr.	Term	Var.	Constr.	Term	Time [s]
S_arb	9838	14262	76087	N/A			
Cpu	2988	4347	18715	396	765	2634	27.82
Disc	334	444	2330	172	265	1223	1.75
Reset	220	295	1191	212	432	1208	0.71
Term	92	159	459	48	108	248	0.36

Quantity	Percentage of OPBDP preprocessed / original
Number of variables	56.2418 ± 21.3035
Number of constraints	68.2409 ± 23.8061
Number of terms	58.5825 ± 22.7967

Table 6: IP model sizes before and after preprocessing

as preprocessing removed of the easily removable part of the redundacy. The number of firing rules in the dataflow model and the constraints in the IP model is taken as a measure of model size, respectively. Results are shown in Fig. (19)

The number of constraints is approx. 12 times as much as the number of firing rules. Model growth can be accounted to several factors:

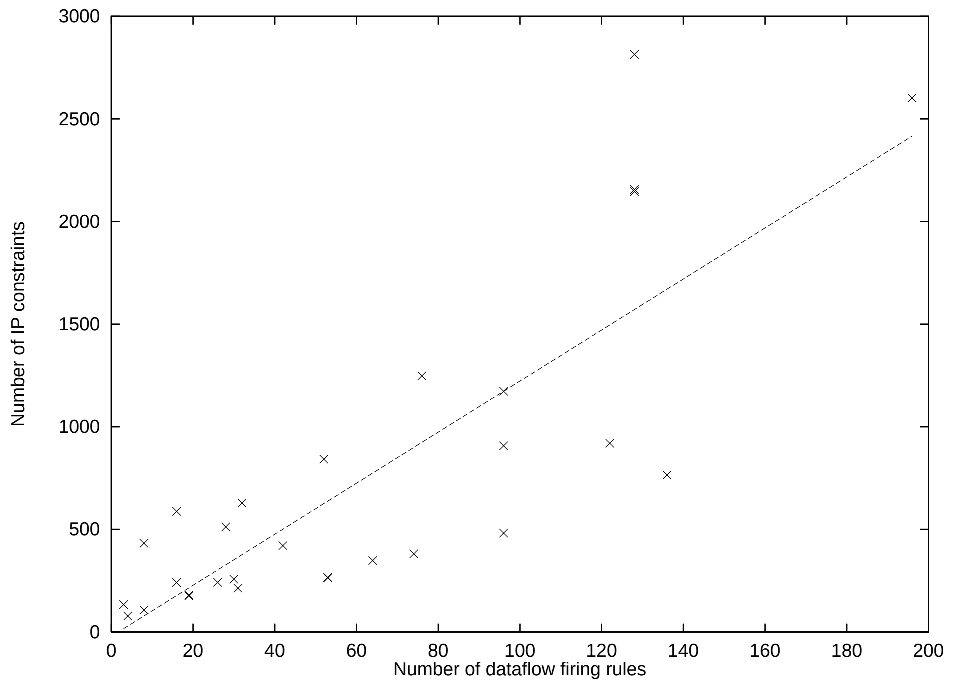
- The original description is higher-level, dataflow tokens are of several kinds. For this reason, one firing rule has more expressive power than one constraint.
- Tranformations should produce "good" integer programming models. To achieve this will be a major piece of future work.

Model growth is lower for large models. This can be attributed to the fact that large models generated by hand tend to contain more redundancy, for the sake of easier understanding and because the designer has less overview which would enable him to model more tersely.

8.6 Performance measurements

The measurement method and the problems are the same as in the previous section. Fig. (20) shows the running times as a function of the number of constraints in the problem.

The $c \cdot x^n$ shaped curves which fit the data best have $n = 1.3781$ and $n = 1.8172$ respectively for the solution phase and both phases. The measurement data is rather irregular; this irregularity is expected to increase for larger models, especially for the



Component	**Dataflow firing rules**	**Constraints in preprocessed IP model**	
		OPBDP	**CPLEX**
S_arb	495	14262 (not preprocessed)	
Cpu	136	765	
Disc	33	265	
Reset	8	432	
Term	8	108	

Figure 19: Model sizes before and after all automatic transformations

solution phase. More large examples are needed to explore the efficiency of solving the models.

As mentioned before, running time depends to a large extent on the way of formulating the model; a lot of experimentation will be necessary. The same goes for probing the different parameters infuencing the algorithms; most solvers provide a large number of these.

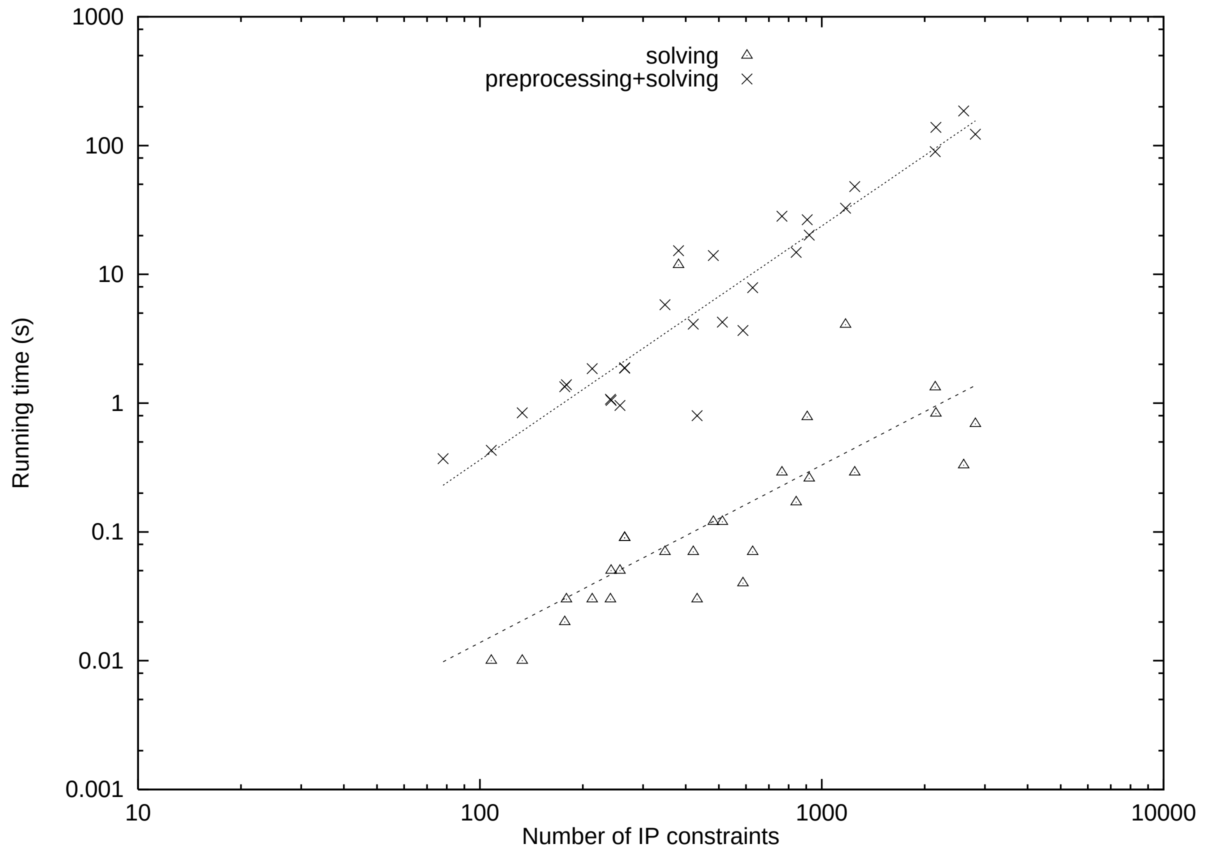


Figure 20: Running times of solving the IP problems

9 The software system

This section does not go into details of the implementation. Such information is available in the source code and the documentation.

9.1 Design goals and choices

The experimental implementation is a collection of loosely coupled programs which communicate by means of files and/or UNIX pipes (|). This solution was chosen because of its extensibility: new parts can be added and other systems may be interfaced to easily, as long as they are "plug compatible", i.e. the file formats are the same. The task of conversion — if needed — needs writing a simple parser in the worst case. Several such parsers or their skeletons are provided already.

This way of communication needs that the interchange formats are specified: see Section (9.2). All the files are plain text files.

The implementation languages of the programs are perl, the Bison parser generator and C++. Perl is good for rapid prototyping and the source can be easily modified, thus it was chosen for solving the simpler problems and for implementing unstable requirements, such as tools specific to a chosen file format. C++ is extendible and efficient, if the program is well designed, thus it is the implementation language of sophisticated algorithms. The Bison parser generator is used for file format conversions.

The source code and the documentation is managed by the CVS versioning system and a set of generic makefiles. Documentation includes diagrams of the Booch object-oriented design notation.

The structure of the software system is summarised in Fig. (21). , Boxes correspond to programs and arrows to data flows, with shaded boxes indicating the interchange file format.

As to the size of the source code (excluding OPBDP source): there are ≈ 4700 lines of C and C++, ≈ 1100 lines of Flex/Bison source, and ≈ 500 lines of Perl and other scripts.

9.2 File formats

9.2.1 Overview

Fig. (7) summarises the used interchange formats and the file extensions used for each type.

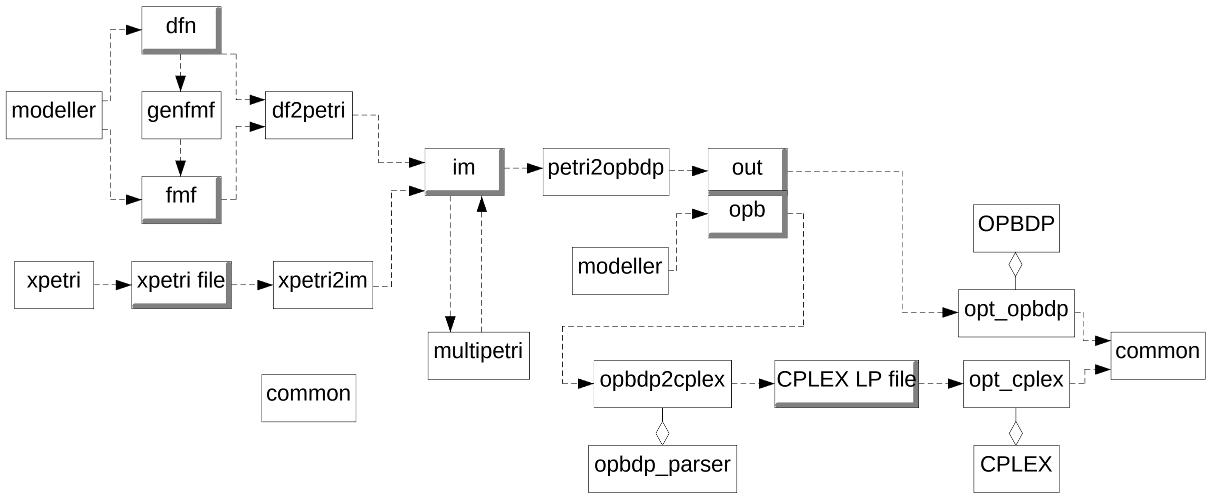


Figure 21: Overview of the structure

File extension	Short description
`.dfn`	Models of dataflow components
`.fmf`	Fault models of dataflow components
`.im`	Descriptions of Petri nets
`.out`, `.opb`	Optimisation problems

Table 7: Interchange file formats

9.2.2 Models of dataflow components

The file format is the one used in [9]. It contains the description of a colored dataflow component; see Section (5.3) for its description. It has the input and output variables, the possible states and the firing rules. Firing rules are of the form

$$i_1(t) = iv_1, \ldots, i_k(t) = iv_k, s(t) = sv \Rightarrow o_1 = ov_1, \ldots o_l(t) = ov_l, s(t+1) = sv_{new}$$

where the i-s are some input variables, the o-s are some output variables and s is the state variable. The other symbols are arbitrary values legal for the corresponding variable.

The file format includes priority and delay values for firings. These are unused by the software system.

9.2.3 Fault models of dataflow components

The dataflow to Petri net tranformation needs classification of input and output variables and the state variable in order to allow incorporation of a fault model for the component. Various classes express various kinds of non-determinism. This information is not fully present in the `.dfn` file, and only appears in the naming of constructs, hence it is not accessiable to the computer.

The classification scheme currently in use by the software system is found below. It must only be viewed as an example, for it may be exchanged to an appropriate model specific scheme easily. Only the dataflow to Petri net transformation (**df2petri**) must be slightly modified to achieve that.

Variable class	**Description**	**Treatment**
normal	describes no fault state.	has at most one value at a time. Having no value means "don't care".
fault	describes the fault state of a system component	one element in the domain is marked as "fault free".
permanent	with permanent fault mode.	has exactly one value.
transient	with transient fault mode.	has either one value or two, one of which means "fault free".

9.2.4 Descriptions of Petri nets

The representation is sparse. Linear constraints on 0-1 variables representing the occurance of token flow into a specific place can be added. Special comments are used to store information on the fault model and the contents of the dataflow component — if the file comes from a dataflow to Petri net transformation. See Section (9.2.3).

9.2.5 Optimisation problems

A description of a linear 0-1 optimisation problem with integer coefficients. The format is similar to the widespread LP format of e.g. the public domain tool `lpsolve`. The fact that the variables are 0-1 need not be expressed. This format is easily convertible to other widely used formats.

The concatenation of `.opb` and `.out` is a full problem statement. Of those, `.out` is the part generated by the reformulation algorithm of Section (6).

9.3 Programs

9.3.1 `df2petri`

A program transforming dataflow models into an equivalent Petri net model. The input is a dataflow component description file along with some additional information on the fault model. Section (5) is devoted to the transformation algorithm.

9.3.2 `genfmf`

Generates a default fault model information file for a dataflow component description, in order to ease the creation of these files. In the new file all variables are normal variables but the state variable, which is "permanent" (see 9.2.3).

9.3.3 `multipetri`

Constructs the iterative array model of a dataflow component (see Section (5.2)), by concatenating the output an dinput state variables of the models at consecutive time intervals.

9.3.4 `petri2opbdp`

Transforms the Petri net model — extended by some constraints — into a linear inequality system of 0-1 variables. The algorithm is described in Section (6). The latter must be extended by an objective function and additional constraints to obtain a formulation of a specific dependability problem; for possible ways see Section (8).

9.3.5 CPLEX

CPLEX[13] is commercial software, solving linear mixed integer programming problems. It is one of the most widespread and performant tools on the market. It has numerous parameters for the solving and preprocessing phase; the system is tunable to a high extent.

9.3.6 OPBDP

OPBDP[11] is a freely available tool implementing an implicit enumeration algorithm for solving linear 0-1 optimisation problems[12] with integer coefficients, as well as a large selection of preprocessing algorithms.

9.3.7 `opbdp2cplex`

Transforms an OPBDP format problem file into a CPLEX LP format problem file. Uses **opbdp_parser**.

[12]Note that the result of model transformation is an optimisation problem satisfying these requirements.

9.3.8 Tools for the programs

A C++ library A library containing classes used by several of the executables, e.g. container class templates.

Container class templates are planned to be replaced by STL[10].

`opbdp_parser` A generic parser for OPBDP format problem files. Parsing actions can be implemented by subclassing an abstract class.

`opt_opbdp`, `opt_cplex` These tools are wrappers around OPBDP and CPLEX, resp. . They run the optimisation software with some default options and interpret the output: values of variables in the optimal solution will be translated into values of the dataflow component variables from which the model to be solved was created.

10 Use of high-level descriptions

Specific high-level modeling techniques are shortly addressed in this section. Coloured Petri nets are discussed as a problem description language. There are also plans to use statecharts — hierarchical finite state machines expressing also parallelism, as described in [17] — in the modeling process.

10.1 Coloured Petri nets

Coloured Petri nets are popular modeling tools in the description of complex problems. They show a good efficiency especially in those cases, where -at least locally- similarities are present in the problem, as this way the repetition of the identical parts can be avoided in the Petri net representation.

In the case of diagnostics the following main reasons state the necessity of further investigations on application of coloured nets in a similar diagnostic algorithm:

- Our model is essentially nothing else than the unfolded version of a coloured net, in such a way that mutual exclusion relations prescribe the uni-color firing of the variables.

  A special attractivity of going to the higher description level results from the fact that the Murata theorem forming the basis of the developed algorithms is valid even for coloured nets, thus the search for T-invariants can be performed eventually using a much more compact representation. The key problem still unsolved is, how to express the constraints in a manageable form.

- A practical benefit is assumed in the coloured modeling of the fault-error-failure mechanism due to the following reasons:

  - The unfolded model handles faulty and fault-free cases in symmetrical tests, as if they were fully independent ones.
  - The reduction of typical test structure elements, like dominant and equivalent faults, require an extensive preprocessing in the non-coloured models.
  - Traditional Petri nets offer a fine granular modeling capability in hierarchical modeling for the detailed description of the local fault-to-error transformation mechanism within a subunit. However, as error propagation mechanisms through the remaining parts of the systems depend only on the type of the error on the outputs of this faulty subunit, this mechanism is largely independent of the particular fault.

- If the UUT has a regular structure, like a massively parallel processor, this regularity can be easily expressed by means of the association of algebraic attributes with the tokens, in such a general form that a rule can describe "processor$_\ell$ tests processor$_{\ell+1}$".

11 Conclusion and future work

The main result of its report is that the combination of Petri nets as modeling tool and of linear algebra and integer optimization as well-proven solution methodology for large scale problems is a promising way for handling realistic diagnostics problems.

An experimental algorithm was implemented. It is not yet fully mature; its running time and memory usage must be improved. This can be achieved by relatively little implementation effort.

Note that the run-time efficiency of the solver for the integer linear programming problem to be used in the last phase heavily depends on the ordering of inequalities and in general on the model formulation. A lot of experimentation will be necessary to guess the right modelling choices.

Several high level descriptions — in addition to dataflow networks — have to be interfaced to the software system, so that the latter smoothly integrates into the greater HIDE framework. The greatest challenge is to keep the size of the model resulting from each transformation low.

A more detailed list of improvement ideas are given near the ends of Section (5) and Section (6).

References

[1] G. Csertán, A. Pataricza, and E. Selényi. Dependability analysis in hw-sw co-design. In *IEEE international Computer Performance and Dependability Symposium 1995-IPDS'95*, pages 306–315. IEEE Computer Society Press, Apr. 1995.

[2] R. David and H. Alla. *Petri nets and Grafcet.* New York u.a. Prentice Hall, 1992.

[3] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–557, Apr. 1989.

[4] T. Murata and D. Zhang. A predicate-transition model for parallel interpretation of logic programs. *IEEE Transactions on Software Engineering*, 14(4):481–497, Apr. 1988.

[5] L. Portinale. Exploiting t-invariant analysis in diagnostic reasoning on a petri net model. In M. A. Marsan, editor, *Application and theory of Petri nets 1993*, volume 691 of *Lecture notes in computer science*, pages 339–356, Berlin..., 1993. 14th International Conference, Chicago, Illinois, USA, June 21 - 25, 1993, Springer. PND/1.

[6] M. Silva, J. Martinez, P. Ladet, and H.Alla. Generalized inverses and the calculation of symbolic invariants in coloured petri nets. *Technique et Science Informatiques*, 4:113–126, 1985.

[7] J. W. Simpson, W. R. ; Sheppard. *System Test and Diagnostic.* Kluwer Academic Publishers, 1994. ISBN 0-7923-9475-5.

[8] J. Martínez, M. Silva. A simple and fast algorithm to obtain all invariants of a Generalised Petri Net. Bad Honnef, Germany, *Procs. of the 2nd European Workshop on Application and Theory of Petri Nets*, 411-421, 1981.

[9] Csertan Gyuri MEMSY modelljerol valami.

[10] Working Paper for Draft Proposed International Standard for Information Systems – Programming Language C++. Public Review Document. ISO/IEC JTC1/SC22/WG21, 1997.

[11] Peter Barth. Logic Based 0-1 Constraint Programming. Kluwer Academic Publishers, Boston, ISBN: 0-7923-9663-4, 1995. For OPBDP: World Wide Web `http://www.mpi-sb.mpg.de/ barth/opbdp/opbdp.html`, 1995.

[12] Unified Modeling Language Summary, Version 1.1. World Wide Web `http://www.rational.com/uml/`, 1 September 1997.

[13] Using the CPLEX Callable Library. Version 4.0, 1995.

[14] H. P. Williams. Model building in mathematical programming. Wiley and Sons, New York, 1990.

[15] G. L. Nemhauser, L. A. Wolsey. Integer and Combinatorial Optimization. John Wiley & Sons, New York, 1988.

[16] Fourer, Robert (4er@iems.nwu.edu), Gregory, John W. (ashbury@skypoint.com). Linear Programming FAQ. World Wide Web http://www.mcs.anl.gov/home/-otc/faq/linear-programming-faq.html, Usenet sci.answers, anonymous FTP ftp://rtfm.mit.edu/pub/usenet/sci.answers/linear-programming-faq, 1997.

[17] David Harel. Statecharts: A Visual Formalism for Complex Systems. Science of Computer Programming, 8(3), pp. 231-274, June 1987.