A TOOLBOX FOR DESIGN OF DIAGNOSIS SYSTEMS

Erik Frisk, **Mattias Krysander**, **Mattias Nyberg**, **and Jan Åslund**

Department of Electrical Engineering
Linköpings universitet, 581 83 Linköping, Sweden
{frisk,matkr,matny,jaasl}@isy.liu.se

Abstract: Design of diagnosis systems is a complex task that involves many different steps. Full understanding of all different parts of the design procedure requires deep knowledge on theory from a wide variety of subjects. Thus, to encourage the use of results from diagnosis research it is highly desirable to have software support in the design process. This paper describes ongoing work for determining an architecture for such a toolbox. The paper also describes software solutions in the toolbox. In industry as well as in universities, Matlab is probably the most widespread tool used by control engineers. Therefore the toolbox is primarily based upon Matlab but also some computer algebraic tools such as Mathematica and Maple are used.

Keywords: Fault diagnosis, software, toolbox, Matlab

1. INTRODUCTION

Design of diagnosis systems is a complex task that involves many different steps. Full understanding of all different parts of the design procedure requires deep knowledge on theory from a wide variety of subjects. Thus, to encourage the use of results from diagnosis research it is highly desirable to have software support in the design process. This paper describes ongoing work for determining an architecture for such a toolbox and also software solutions to support a diagnosis systems engineer. The work has been carried out partly as a collaboration between Linköping University and Scania CV.

2. USING THE TOOLBOX IN THE DESIGN PROCESS

The process of designing a diagnosis system contains several steps. Important steps are the following:

- Importing and converting models.
- Isolability and detectability analysis.
- Selection of submodels to be used in residual generator design.
- Residual generator design.
- Threshold, possibly adaptive, design for the residual generators.
- Selection of which diagnostic tests to use for optimal fault isolation performance.

When using the toolbox to design a diagnosis system, the first step is to import the model. Models can be described in many ways: Simulink models, linear state-space or DAE-models in the Matlab control toolbox format. For all these model descriptions, the toolbox has support for handling information of how different faults affect the process. Several fault types can be handled such as parameter faults and additive faults. This topic is briefly discussed in Section 4.

When the model is loaded, there is a possibility to analyze fault isolability and fault detectability properties of the model. Depending on the model description, different tools are available. If the model is linear, analytical and exact methods are available. If the model is nonlinear, structural and simulation based methods are available. This is discussed in Sections 3 and 5

The next step is to select submodels to be used in the design of residual generators. Here, the user can specify different fault scenarios, for which the residual design should be based upon. This includes to decide which faults that should be decoupled in each residual.

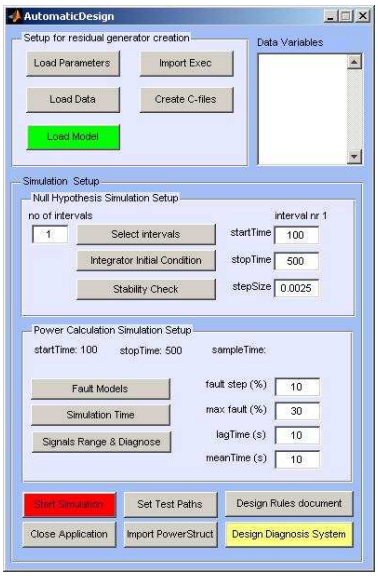

Fig. 1. Screen dump of one window in the user interface.

As an alternative, the toolbox includes an automatic design method which tries to decouple faults in every possible way. This method is based on structural analysis. After this step, a set of submodels is obtained. Section 3 addresses this topic.

Next, residual generators can be designed for each of the submodels. Here the user can choose between a number of methods: linear decoupling via null-space operations, nonlinear parity relations, and nonlinear observers. The toolbox also has support for generation of C-code for the residual generators. This is essential both for the final implementation but also to speed up the analyses done in the next step. Residual generator design is covered in Section 6.

The residual generators need to be supplemented with filtering, normalization, and thresholds. Doing this, a diagnostic test is obtained. In this respect the toolbox has support for analyzing the performance of diagnostic tests, including how different design parameters such as thresholds affect the performance of the diagnostic tests. This part of the toolbox is further described in [2].

The result of the previous steps is a set of diagnostic tests. This set can be quite large and contain many redundant or superfluous tests that are not needed to obtain the diagnosis performance requested. Therefore the toolbox contains as a next step an algorithm for picking out a subset of all diagnostic tests. This is done by estimating the performance of different tests with either Monte-Carlo simulations or, if available, measured data. The goal is that this subset should fulfill a specified isolation requirement, but at the same time, minimize the on-line computational burden needed to compute the residuals. Also this part of the toolbox is further described in [2].

To illustrate the user interface of the toolbox, Figure 1 contains a screen dump of one of the windows. The main purpose of this window is to set up the analysis of residual generator performance.

3. REDUNDANCY ANALYSIS

In model based diagnosis, the diagnosis system construction is based on a model of the technical system to be diagnosed. In order to achieve fault isolation, a common strategy is to pick out submodels with redundancy and to test these separately against measured signals, [3,13,14]. To cope with large differential algebraic models, a systematic structural approach is here used to find these submodels. The basic ideas of how to do this and the software tool are briefly described next.

3.1 *Background and basic definitions*

The structure of a model includes which variables that are included in each equation. Consider for example the dynamic system

$$\begin{aligned} e_1 : \dot{x} &= -2\,x + u \\ e_2 : y_1 &= \dot{x} \\ e_3 : y_2 &= x \end{aligned} \tag{1}$$

where u, y_1, and y_2 are known, and x is an unknown variable. One structural representation of (1) is the following:

equation	unknown	known		
	x	u	y_1	y_2
e_1	X	X		
e_2	X		X	
e_3	X			X

(2)

In this representation an X denotes that the variable, or any of its time-derivatives, appear in the equation. This approach has been used in for example [6,8,9,12]. There are other structural representations of dynamical systems, see e.g. [3], which are also supported by the toolbox. However, for the remainder of this paper, this representation is used.

Redundancy is needed for computing parity relations and residual generators and this motivates the following definition.

Definition 1. (Structurally Overdetermined). A set M of equations is *structurally overdetermined* if M has more equations than unknowns.

The SO sets of equations should be regarded, in the generic case, as the sets of equations containing redundancy. To localize the faults in the model and to obtain good fault isolation small overdetermined sets are important.

Definition 2. (Minimal Structurally Overdetermined). A structurally overdetermined set is a *minimal structurally overdetermined* (MSO) set if no proper subset is a structurally overdetermined set.

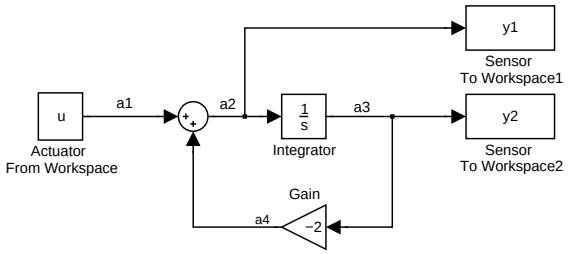

Fig. 2. Example Simulink model

The MSO sets in the structural representation of (1) shown in (2) are $\{e_1, e_2\}$, $\{e_1, e_3\}$, and $\{e_2, e_3\}$. The corresponding parity relations are $\dot{y}_1 + 2y_1 - \dot{u} = 0$, $\dot{y}_2 + 2y_2 - u = 0$, and $\dot{y}_2 - y_1 = 0$ respectively.

3.2 *Software*

In this section we present the algorithms that can be used to find all MSO sets in a model. The input to the algorithms can be a Simulink model, analytical equations, or a structural model like (2). Starting with a Simulink model of the system, the first step in the design procedure is to extract analytical equations from the model file.

To give an example of an input Simulink file consider the model in Figure 2. This is a Simulink model of the system (1). `To Workspace`-blocks tagged with the letters `Sensor` are treated as known sensor signals and `From Workspace`-blocks tagged with the letters `Actuator` are treated as known actuator signals. The Simulink file containing the model in Figure 2 is parsed and transformed into the following analytical system of equations:

```
e1 Actuator From Workspace : a1 = u
e2 Sum                     : a2 = a1 + a4
e3 Integrator              : a3 = Integrator(a2)
e4 Gain                    : a4 = -2 * a3
e5 Sensor To Workspace1    : y1 = a2
e6 Sensor To Workspace2    : y2 = a3
```

Each Simulink block is transformed into a model equation and connections correspond to unknown variables. Note that this model contains more equations and unknowns than necessary. The algorithms facilitates reduction techniques that can be used to reduce the size of the model. However, here we proceed without this reduction.

The next step is to extract the structural information from the model equations, i.e. to identify which variables that are included in each equation. For the example, the structural model obtained by the algorithm is

equations	unknown				known		
	a_1	a_2	a_3	a_4	u	y_1	y_2
e_1	X				X		
e_2	X	X		X			
e_3		X	X				
e_4			X	X			
e_5		X				X	
e_6			X				X

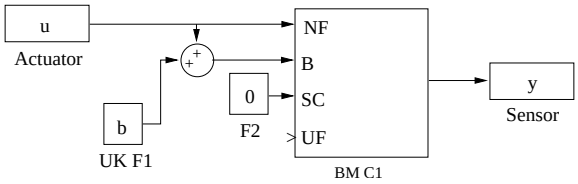

Fig. 3. The Simulink model of (3).

Finally, all MSO sets in the structural model is found using the algorithm presented in [9]. In this case the MSO sets are $\{e_1, e_2, e_3, e_4, e_5\}$, $\{e_1, e_2, e_3, e_4, e_6\}$, and $\{e_3, e_5, e_6\}$.

The software described in this section has been applied to an engine model with 126 equations and 122 unknowns. The model contained 1419 MSO sets and the execution time for all the steps was approximately 10 seconds on a PC with a 1 GHz processor. For more details about the model see [9].

4. MODELING FOR FAULT DIAGNOSIS IN SIMULINK

When modeling faults in Simulink, information about fault behavior must be included. For example, consider the equation $y = u$ where y is a sensor signal and u is an actuator signal. Assume now that the sensor has three possible fault modes: a constant bias fault (B), a short circuit fault (SC) and an unknown fault (UF). Let Ω denote the set of all behavioral modes of the sensor, i.e. $\Omega = \{NF, B, SC, UF\}$ where NF denotes the no fault mode. This model may be written as:

$$\begin{array}{ll} \text{Ass.} & \text{Equation} \\ NF & y = u \\ B & y = u + b \\ SC & y = 0 \\ & \dot{b} = 0 \end{array} \qquad (3)$$

where Ass. is the behavioral mode assumption.

When using the toolbox, model (3) is in Simulink represented as in Figure 3. This solution uses a subsystem block BM-C1 to connect the equations to their corresponding behavioral mode. Subsystems defining behavioral modes are tagged with the letters BM in the beginning of their names. The interpretation of the UK tagged constant block is that b is an unknown constant.

5. FAULT ISOLATION

Structural analysis is a powerful tool for early determination of detectability/isolability possibilities. This is important both to evaluate if the number and placement of sensors is adequate in order to meet diagnosis specifications.

Even though the structural information is very coarse, useful insights can be gained by analyzing the structure. This is also one of the strengths, since useful

information can efficiently be obtained early in the development process before much work has been spent on obtaining detailed analytical models.

To evaluate possible isolability properties of the model, fault sensitivity of all possible parity relations need to be studied. Under some technical model assumptions [10], not included here, the isolability properties of a system can be determined by only considering the fault sensitivities of all MSO sets.

The fault isolability analysis, performed by the software, will be illustrated on the model in Figure 2. Assume that the actuator, the gain block, and the two sensors can fail. The corresponding fault modes are abbreviated A, G, $S1$, and $S2$ and the no-fault mode is denoted by NF. Multiple faults can be handled [11], but for brevity only single faults are considered. By including the behavioral mode information in the Simulink model the computed fault incidence matrix for the MSO sets is

	NF	A	G	$S1$	$S2$
$\{e_1, e_2, e_3, e_4, e_5\}$	0	X	X	X	0
$\{e_1, e_2, e_3, e_4, e_6\}$	0	X	X	0	X
$\{e_3, e_5, e_6\}$	0	0	0	X	X

In a noisy and uncertain environment we have to threshold the residuals such that the probability for false-alarm is low. Thus, conclusions about faults can only be drawn when a residual is larger than its associated threshold. When residuals are below thresholds, no conclusion can be drawn in a sound way. This means that for example $S2$ *may* invalidate the two last MSO sets. Thus, a large enough fault of the type $S2$ will be isolated as $S2$. However, no matter how large, a fault of the type A can not be separated from G. The output of the algorithm is a fault isolability matrix, which summarizes isolability properties of the model. The isolability matrix for the example is

	NF	A	G	$S1$	$S2$
NF	X	X	X	X	X
A	0	X	X	0	0
G	0	X	X	0	0
$S1$	0	0	0	X	0
$S2$	0	0	0	0	X

where an X in position (i, j) means that behavioral mode i can be interpreted as behavioral mode j, i.e. behavioral mode i can not be isolated from behavioral mode j.

6. DESIGN OF RESIDUAL GENERATORS

After an initial redundancy analysis has been made and a subset of equations, with redundancy, has been selected it is time for residual generator design.

There exists many methods for residual generator design with their respective advantages and disadvantages. Two approaches, common within the FDI community, are

- Designs based on parity relations.
- Observer based designs.

If the submodel is linear, design is easy and complete solutions are available regardless of the model structure. An algorithm and software support for this case are discussed further in Section 6.2. For the case of non-linear dynamic models the choice of method is more involved and is discussed next in Section 6.1.

Next is a more detailed look on software support for designs based on parity relations.

6.1 *Parity relation based designs*

The objective of this section is to describe how, in the toolbox, a residual generator is designed for each MSO set. To illustrate this, the example introduced in Section 3.1 will be used. The equations in the example are linear and the approach presented later in Section 6.2 can be applied. However, we will present a method that is applicable also to non-linear dynamic systems. The basic idea in this approach is to consider signals and their time-derivatives as separate independent variables and use algebraic elimination to obtain a parity relation. To do this elimination we need more equations and these can be obtained by differentiation.

We illustrate the method using the MSO set $\{e_1, e_2\}$ in (2). For example the set $\{\dot{e}_1, e_2, \dot{e}_2\}$ contains the unknown variables $\{\dot{x}, \ddot{x}\}$ and has the following structure:

equation	unknown		known		
	$\dot{x}$	$\ddot{x}$	$\dot{u}$	y_1	$\dot{y}_1$
$\dot{e}_1$	X	X	X		
e_2	X			X	
$\dot{e}_2$		X			X

(4)

This is an MSO model and algebraic elimination can be used to obtain the parity relation

$$2\,y_1 + \dot{y}_1 - \dot{u} = 0 \tag{5}$$

In this part of the software, the input is an MSO set of differential equations. Structural methods are used to decide which equations to differentiate and how many times. The theory for general non-linear differential algebraic equations is presented in [8]. After that, standard elimination tools e.g. Gröbner basis theory, can be used to obtain the parity relation. Such algorithms are readily available in standard computer algebra software, like Mathematica and Maple, and the toolbox contains an interface to these packages. When performing the analytical elimination, the order of which variables are eliminated can greatly influence the complexity of the elimination problem.Structural heuristics are implemented to choose elimination orders that correspond to as low elimination trees as possible [4]. The approach described above uses algebraic elimination tools, but also differential algebraic tools are available, see e.g. [15].

Another approach, related to the approaches outlined above, is to find a matching in the bi-partite graph that represents the structure of the MSO. This matching

represents a causality assignment, i.e. a way to compute a residual. Software has been developed using this approach to automatically, for a class of MSO models, generate Simulink implementations of the residual generators. This approach is also closely connected with the approach outlined in the structural analysis parts of the book [3].

6.2 *Linear Residual Generation*

The aim of this section is to outline how the toolbox handles a complete design procedure for any type of linear model. This means that not only state-space models or descriptor models should be considered but a more general class of models. The motivation for this is that modern physically based object-oriented modeling tools like for example Dymola, produce models that are not on any specific form and with no causality, i.e. it is not determined by the model which signals that are inputs and which that are outputs. This fits well with the diagnosis problem where signals rather are separated into known and unknown signals. The objective is to be able to use such models directly, without any need for additional transformations.

The presentation below is entirely for time-continuous systems, but the approach works also for time-discrete systems with only small changes. The approach is based on the theory of polynomial matrices, see e.g.[7] for a thorough description of the theory. The toolbox uses a software package for dealing with polynomial matrices in Matlab [1].

The class of models considered is general linear models in the form

$$H(p)x + L(p)z + F(p)f = 0 \tag{6}$$

where $x(t) \in \mathbb{R}^n$, $z(t) \in \mathbb{R}^{n_z}$, and $f(t) \in \mathbb{R}^{n_f}$. The matrices $H(p)$, $L(p)$, and $F(p)$ are polynomial matrices in the differentiation operator p. The vector x contains all unknown signals, such as internal system states, unknown inputs such as disturbances, and possibly also faults that are to be decoupled. The vector z contains all known signals such as control signals and measured signals, and the vector f contains the fault-signals corresponding to faults that need to be detected. The equations (6) may be an MSO set, but can also be a larger set of equations.

The only assumption imposed on the matrices describing the model (6) is that $[H(s)\ L(s)]$ has full row rank. This is a reasonable assumption since it means that there are no linear dependencies in the model equations when $f = 0$.

The above model structure is directly applicable to e.g. state-space and descriptor models where matrices $H(p)$, $L(p)$, and $F(p)$ then become

$$H(p) = \begin{bmatrix} C & D_d \\ -(pE - A) & B_d \end{bmatrix}, L(p) = \begin{bmatrix} -I & D_u \\ 0 & B_u \end{bmatrix}, F(p) = \begin{bmatrix} D_f \\ B_f \end{bmatrix} \tag{7}$$

The objective is now to outline a design procedure, and software implementation, that finds all residual generators for a given system and also provides the required fault sensitivity in the residual.

To illustrate the design procedure, let the rows of $N_H(s)$ form an irreducible polynomial basis for the left null-space of the matrix $H(s)$. If we let $f = 0$ and multiply (6) from the left with $N_H(p)$, we obtain the expression $N_H(p)L(p)z = 0$. In this equation the influence of the unknown signals x has been decoupled. Thus, by picking one row in $N_H(p)L(p)$, or a linear combination of rows specified by a row vector $\gamma(p)$, we obtain a so called parity relation $\gamma(p)N_H(p)L(p)z = 0$. By adding stable dynamics $d(p)$ of sufficiently high order we obtain a residual generator with transfer operator

$$R(p) = d^{-1}(p)\gamma(p)N_H(p)L(p) \tag{8}$$

which can be realized by an explicit state-space description. This is our basic idea of how residual generators can be constructed for models in the form (6). To achieve the required fault sensitivity, the vector γ need to be chosen with some care.

It is straightforward to prove that expression (8) can be used to form *any* residual generator for a controllable system. For non-controllable systems additional care has to be taken to parameterize all residual generators and the general case is treated in detail in [5].

It is of course important that the residual generator is not only insensitive to unknown inputs but also sensitive to the faults we wish to detect. An intuitive result, also from [5], on fault detectability is

Theorem 1. Fault i is detectable in (6) if and only if $\operatorname{Rank}[H(s)\ F_i(s)] > \operatorname{Rank} H(s)$.

A design procedure can now be outlined as:

1. Form matrices $H(p)$, $L(p)$, $F(p)$ and compute $Q(s) = N_H(s)L(s)$.
2. Determine if all faults are detectable using Theorem 1 and then choose γ such that the residual has required fault sensitivity properties.
3. Choose a stable polynomial $d(s)$ with degree higher than $\gamma Q(s)$ and form the residual generator $R(p)$ according to (8). This expression can easily be written in state-space form for direct implementation.

Design and analysis of residual generators is in the toolbox performed in Matlab using functions from the Polynomial Toolbox [1] which are well suited for residual generator design. It will show that the main design steps are standard operations from polynomial theory and thus all operations rely on well tested and thoroughly analyzed algorithms. Numerical issues are often of concern when polynomial algorithms are considered and numerical motives for the approach is discussed in [5]. The code excerpts shown below illustrates the main steps of the implementation.

Before we start the design, we may want to check if the design goal is at all achievable, i.e. if all faults are

detectable. Using the simple rank condition in Theorem 1 for each fault respectively, it can be checked if they are detectable. A detectability test is done with the commands

```
>> rank([H F(:,1)]) > rank(H)
```

where `F(:,1)` is the column of $F(p)$ corresponding to the fault we wish to analyze. The command will return 1 if the fault is detectable and 0 otherwise. When fault detectability is ensured, the residual generator can be designed, following the design procedure above, with the following set of simple commands:

```
>> Nh = null(H')';
>> Q = Nh*L;
>> gamma = 1; s0 = -3;
>> q = deg(gamma*Q); d = (s-s0)^q;
>> % Compute state-space form of R(s)
>> [A,B,C,D] = lmf2ss(gamma*Q,d);
```

If the model (6) is a minimal over determined subsystem, the left null-space of matrix $H(s)$ will have dimension 1. This further means that the choice of parameter γ disappears, i.e. can always be set to 1 as in the example session above. In the example session, all poles of the residual generator are placed in $s = s_0 = -3$.

7. SUMMARY

Design of diagnosis systems is a complex task that involves many different steps. Full understanding of all different parts of the design procedure requires deep knowledge on theory from a wide variety of subjects. This paper has briefly described a toolbox as a support for the whole design procedure of a diagnosis system. Software solutions include significant parts that are based on structural analysis but also analytical design methods and evaluation based on measured data. The structural analysis is used in several parts of the design process, e.g. in the fault isolability and detectability analysis and also in the residual generator design process.

In industry as well as in universities, Matlab is probably the most widespread tool used by control engineers. Therefore the toolbox is primarily based upon Matlab but also some computer algebraic tools such as Mathematica and Maple are used.

8. REFERENCES

[1] *The Polynomial Toolbox 2.5*. Polyx, Czech Republic. URL: http://www.polyx.com, 2001.

[2] G. Arrhenius and H. Einarsson. Automatic design of diagnosis systems, optimizing isolation and computational load. Master's thesis, Uppsala University, 2005.

[3] M. Blanke, M. Kinnert, J. Lunze, and M. Staroswiecki. *Diagnosis and Fault-Tolerant Control*. Springer-Verlag, 2003.

[4] E. Frisk. Efficient elimination orders for the elimination problem in diagnosis. Technical Report LiTH-R-2532, Department of Electrical Engineering, Linköpings Universitet, SE-581 83 Linköping, Sweden, 2003.

[5] E. Frisk and M. Nyberg. Residual generation for fault diagnosis of systems described by general linear differential-algebraic equations (revised). Technical Report LiTH-R-2531, Department of Electrical Engineering, Linköpings Universitet, SE-581 83 Linköping, Sweden, 2005.

[6] Erik Frisk, Dilek Düştegör, Mattias Krysander, and Vincent Cocquempot. Improving fault isolability properties by structural analysis of faulty behavior models: application to the DAMADICS benchmark problem. In *Proceedings of IFAC Safeprocess'03*, Washington, USA, 2003.

[7] Thomas Kailath. *Linear Systems*. Prentice-Hall, 1980.

[8] Mattias Krysander and Jan Åslund. Graph theoretical methods for finding analytical redundancy relations in overdetermined differential algebraic systems. In *IMACS World Congress*, Paris, France, 2005.

[9] Mattias Krysander, Jan Åslund, and Mattias Nyberg. An efficient algorithm for finding over-constrained sub-systems for construction of diagnostic tests. In *16th International Workshop on Principles of Diagnosis (DX-05)*, Pacific Grove, California, USA, 2005.

[10] Mattias Krysander and Mattias Nyberg. Structural analysis utilizing MSS sets with application to a paper plant. In *Proc. of the Thirteenth International Workshop on Principles of Diagnosis*, Semmering, Austria, May 2002.

[11] Mattias Krysander and Mattias Nyberg. Fault isolability prediction of diagnostic models. In *16th International Workshop on Principles of Diagnosis (DX-05)*, Pacific Grove, California, USA, 2005.

[12] S. Ploix, M. Desinde, and S. Touaf. Automatic design of detection tests in complex dynamic systems. In *Proceedings of 16th IFAC World Congress, Prague*, Prague, Czech Republic, 2005.

[13] B. Pulido and C. Alonso. Possible conflicts, ARRs, and conflicts. In M. Stumptner and F. Wotawa, editors, *Proceedings DX-2002*, pages 122–127, Semmering, Austria, 2002.

[14] L. Travé-Massuyès, T. Escobet, and R. Milne. Model-based diagnosability and sensor placement. application to a frame 6 gas turbine subsystem. In D. T. Dupre' S. Mcilraith, editor, *DX01 twelfth international workshop on principles of diagnosis*, pages 205–212, 2001.

[15] A. D. Wittkopf. *Algorithms and Implementations for Differential Elimination*. PhD thesis, Simon Fraser University, Burnaby, British Columbia, Canada, 2004.