Bunus P., Fritzson P.:
Methods for Structural Analysis and Debugging of Modelica Models
2nd International Modelica Conference, Proceedings, pp. 157-165

Paper presented at the 2nd International Modelica Conference, March 18-19, 2002,
Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR), Oberpfaffenhofen, Germany.

All papers of this workshop can be downloaded from
http://www.Modelica.org/Conference2002/papers.shtml

Methods for Structural Analysis and Debugging of Modelica Models

Peter Bunus, Peter Fritzson
PELAB, Programming Environment Laboratory, Department of Computer and Information Science, Linköping University, SE-581 83, Linköping, Sweden
{petbu,petfr}@ida.liu.se

Abstract

A significant part of the simulation design effort is spent on detecting deviations from the specifications and subsequently localizing the source of errors. Employment of debugging environments that control the correctness of the developed source code is an important factor in reducing the time and cost of software development in classical programming languages. Currently, few or no tools are available to assist developers when debugging declarative equation based modeling languages. To begin to address this need we have developed an efficient debugging framework for Modelica and have adapted traditional debugging techniques and algorithms to it. The developed algorithms and methods help to statically detect and repair a broad range of errors without having to execute the simulation model. Several simulation models and examples are given in this paper in order to illustrate the main situations when over and under-constraining equations can appear in the system. Error detection and error solving strategies for those cases are also given.

1 Introduction

Obviously, each simulation problem is associated with a corresponding mathematical model. In dynamic continuous simulation the mathematical model is usually represented by a mixed set of algebraic equations and ordinary differential equations. A typical problem which appear in physical system modeling and simulation is when too many (or few) equations are specified in the system inevitably leading to an inconsistent state of the simulation model. In such situations numerical solvers fail to produce correct solutions to the underlying system of equations.

For example, a physical system simulation model specified in a declarative object-oriented equation based modeling language may consist of several hundreds of classes resulting in over 100 000 flattened equations. If one of these equations over-constrains the overall system it cannot be simulated. It can be easily imagined that, if a subset of six over-constraining equations can be provided by a static debugger from where the user can choose one equation to eliminate, in order to form a structurally well posed simulation problem, it would be extremely useful. This could greatly reduce the amount of time required to get the simulation working.

Our goal is to contribute to the methodology of algorithmic debugging and automated debugging of object-oriented equation based modeling languages and to develop programming environments to support it. Although, what we present in this paper applies to the whole area of equation based debugging, our primary target is debugging of Modelica models and more specifically static analysis techniques for diagnosability of physical system models specified with Modelica.

The simulation models presented in this paper are so trivial as to be almost beneath consideration, but they serves as a straightforward vehicle for the introduction of several fundamental debugging concepts with the purpose of illustrating concepts of structural analysis. These models are extremely useful from that point of view because they keep the associated structural graphs to a minimum size and complexity, but in the meantime exposing interesting structural and debugging problems.

This paper is organized as follows: Section 2 provides graph theoretical preliminaries necessary to understand the algorithms used in this paper together with the canonical decomposition algorithm. In Section 3 simple over constrained simulation models are diagnosed and debugged with the help of graph decomposition techniques and our algorithmic debugging approach. Details are briefly presented about the structures used to annotate the underlying equations of the simulation model, in order to help the debugger to eliminate the heuristics when multiple choices are available to fix an error. In Section 4 the particulars of debugging an under-constrained systems are given. Implementation details of the debugger are given in Section 5. Finally, Section 6 concludes and summarizes the work.

2 Preliminaries

Many practical problems are examples of a model of interaction between two different types of objects and can be phrased in terms of problems on bipartite graphs. The expressiveness of the bipartite graphs in concrete practical applications has been demonstrated many times in the literature (Dolan and Aldous [2]), (Asratian et al. [1]). We will show that the bipartite graph representations are general enough to efficiently accommodate several numeric analysis methods in order to reason about the solvability and unsolvability of the

flattened system of equations and implicitly about the simulation model behavior. Another advantage of using the bipartite graphs is that it offers an efficient abstraction necessary for program transformation visualization when the equation based declarative specifications are translated to procedural form.

The bipartite graph representation and the associated decomposition techniques are widely used internally by compilers when generating the procedural form from the declarative equation based description of the simulation model (Elmqvist [4]) (Maffezzoni et. al. [9]) but none of the existing simulation systems use them for debugging purposes or expose them visually for program understanding purposes.

Definition 1: A bipartite graph is an ordered triple $G = (V_1, V_2, E)$ such that V_1 and V_2 are sets, $V_1 \cap V_2 = \emptyset$ and $E \subseteq \{\{x, y\}; x \in V_1, y \in V_2\}$. The vertices of G are elements of $V_1 \cup V_2$. The edges of G are elements of E.

Definition 2: A *matching* is a set of edges from graph G where no two edges have a common end vertex.

Definition 3: A matching M of a graph G is *maximal* if it is not properly contained in any other matching.

Definition 4: A vertex v is *saturated* or *covered* by a matching M if some edge of M is incident with v. An unsaturated vertex is called a *free vertex*.

Definition 5: A *perfect matching* P is a matching in a graph G that covers all its vertices.

Definition 6: A path $P = \{v_0, v_1, \cdots, v_k\}$ in a graph G is called an *alternating path* of M if it contains alternating free and covered edges.

In Figure 1 all the possible perfect matchings of a simple bipartite graph are presented. It should be noted that a maximum matching and the perfect matching of a given bipartite graph is not unique.

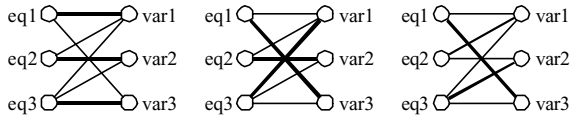


Figure 1. An example bipartite graph with all the possible perfect matchings marked by thick lines.

A structural decomposition of a bipartite graph associated with a simulation model which relies on the above presented vertex coverings is due to (Dulmage and Mendelsohn [3]) and canonically decomposes any maximum matching of a bipartite graph in three distinct parts: *over-constrained, under-constrained*, and *well-constrained* part.

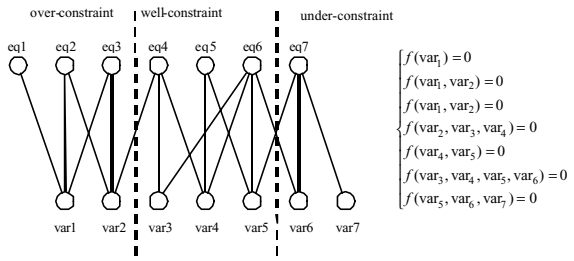


Figure 2. Dulmage Mendelsohn's canonical decomposition of a bipartite graph.

The canonical decomposition algorithm is given below:

Algorithm: Dulmage and Mendelsohn canonical decomposition

Input Data: A biparpartite graph

Result: Three subgraphs: well-constrained , over-constrained and under-constrained .

begin:

- Compute the maximum matching of .
- Compute the directed graph ' by replacing each edge in by two arcs and orienting all other edges from the equations to the variables.
- Let be the set of all descendants of sources of the directed graph '.
- Let be U the set of all ancestors of sink of the directed graph '.
- Calculate = - - .

end.

The *over-constrained* part: the number of equations in the system is greater than the number of variables. The additional equations are either redundant or contradictory and thus yield no solution.

The *under-constrained* part: the number of variables in the system is greater than the number of equations. A possible error fixing strategy would be to initialize some of the variables in order to obtain a well-constrained part or add additional equations to the system.

Over and under-constrained situations can coexist in the same model. In the case of over-constrained model, the user would like to remove the over-constraining equations in a manner which is consistent to the original source code specifications, in order to alleviate the model definition.

The *well-constrained* part: the number of equations in the system is equal to the number of variables and therefore the mathematical system of equations is structurally sound having a finite number of solutions. This part can be further decomposed into smaller solution subsets. A failure in decomposing the well-constrained part into smaller subsets means that this part cannot be decomposed and has to be solved as it is. A failure in numerically solving the well-constrained part means that no valid solution exists and there is somewhere a numerical redundancy in the system.

3 Debugging of Over-Constrained Models

A typical problem which appears in physical system modeling and simulation is when too many equations are specified in the system inevitably leading to an inconsistent state of the simulation model. In such situations numerical solvers fail to compute correct solutions to the underlying system of equations.

In Figure 3 the Modelica source code of a simple simulation model consisting of a resistor connected in parallel to a sinusoidal voltage is given. The intermediate form is also given for explanatory purposes. The `Circuit` model is represented as an aggregation of the `Resistor`, `Source` and `Ground` model instances, `R1`, `AC` and `G` connected together by means of physical ports.

```
connector Pin
  Voltage v;
  Flow Current i;
end Pin;

model TwoPin
  Pin p, n;
  Voltage v;
  Current i;
equation
  v = p.v - n.v; 0 = p.i + n.i; i = p.i
end TwoPin;

model Resistor
  extends TwoPin;
  parameter Real R;
equation
  R*i == v;
end Resistor;

model VsourceAC
  extends TwoPin;
  parameter Real VA=220;  parameter Real f=50;
  protected constant Real PI=3.141592;
equation
  v=VA*(sin(2*PI*f*time));
end VsourceAC;

model Ground
  Pin p;
equation
  p.v == 0
end Ground;

model Circuit
  Resistor R1(R=10); VsourceAC AC; Ground G;
equation
  connect(AC.p,R1.p); connect(R1.n,AC.n);
  connect(AC.n,G.p);
end Circuit;
```

Flat equations

```
1.    R1.v == -R1.n.v + R1.p.v
2.    0 == R1.n.i + R1.pi
3.    R1.i == R1.p.i
4.    R1.i*R1.R == R1.v
5.    AC.v == -AC.n.v + AC.p.v
5.    0 == AC.n.i + AC.p.i
7.    AC.i == AC.p.i
8.    AC.v == AC.VA*Sin[2*time*AC.f*AC.PI]
9.    G.p.v == 0
10.   AC.p.v == R1.p.v
11.   AC.p.i + R1.p.i == 0
12.   R1.n.v == AC.n.v
13.   AC.n.v == G.p.v
14.   AC.n.i + G.p.i + R1.n.i == 0
```

Flat Variables

```
1. R1.p.v   2. R1.p.i   3. R1.n.v
4. R1.n.i   5. R1.v     6. R1.i
7. AC.p.v   8. AC.p.i   9. AC.n.v
10. AC.n.i  11. AC.v    12. AC.i
13. G.p.v   14. G.p.i
```

Flat Parameters

```
R1.R -> 10
AC.VA -> 220
AC.f -> 50
```

Flat Constants

```
AC.PI -> 3.14159
```

Figure 3. Modelica code of a simple electrical circuit and the associated flattened equations

During the first stage of the static analysis the associated bipartite graph of the intermediate flattened form of the equations is constructed and the maximum cardinality matching is computed as it is shown in Figure 4.

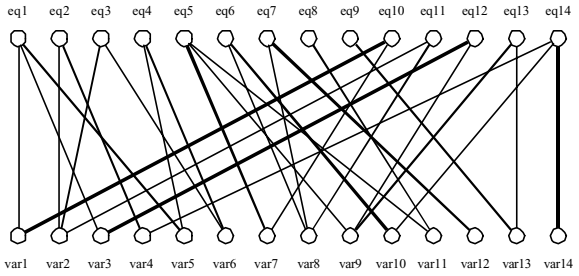


Figure 4. Associated bipartite graph and the corresponding perfect matching (thicker lines) to the simple electrical circuit.

It's worth noting, that in this case, the maximal matching is also a perfect matching of the associated bipartite graph. In this case all the vertices are covered by a matching and the canonical decomposition algorithm will yield to only one well-constrained component without any under or over-constraining part. The well-constrained part can be safely sent to the numerical solver and the simulation can be successfully performed if no other numerical redundancies are present in the system of equations..

Let us now consider the same electrical circuit where an additional equation (`i=23`) was intentionally introduced inside the `Resistor` component in order to obtain a generally over-constrained system. The D&M canonical decomposition will lead to two parts: a well-constrained part and an over-constrained part (see Figure 5). Equation "*eq11*" is a non-saturated vertex of the equation set so it is a source for the over-constrained part. Starting from "*eq11*" which is the non-saturated vertex, the directed graph can be derived from the undirected bipartite graph as is illustrated in Figure 6. by exchanging all the matching edges in bi-directional edges and orienting all other edges from equations to variables. An immediate solution of fixing the over-constrained part is to eliminate "*eq11*" which will lead

to a well-constrained part and therefore the equation system becomes structurally sound.

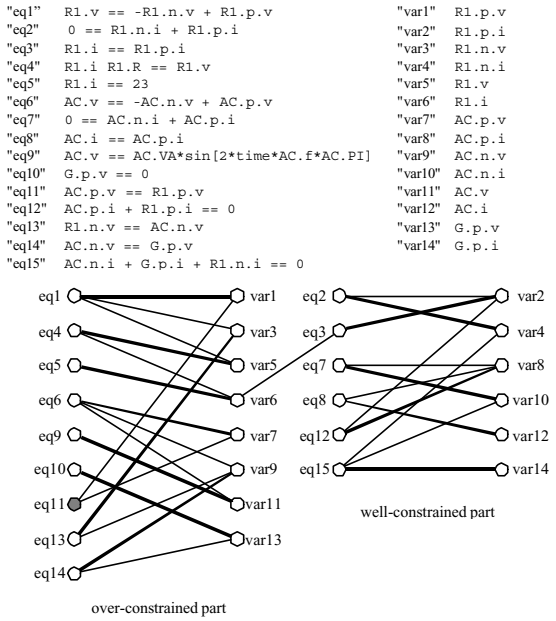


Figure 5. Canonical decomposition of the over-constraining system.

However, examining the equation "*eq11*" one can note that the equation is generated by a `connect` statement from the `Circuit` model, and the only way to remove the equation is to remove the `connect(AC.p, R1.p)` statement. But removing the above-mentioned statement will remove two equations from the flattened model which is unacceptable.

In order to support an automatic reasoning about the equations the flattened equations from the intermediate code are annotated by a structure which resembles the one presented in Table 1.

Table 1. The structure of the annotated equation

Attribute	Value
Equation	`R1.i * R1.R == R1.v`
Name	"*eq4*"
Description	"*Ohm's Law for the resistor component*"
Nr. of associated eq	1
Class Name	"*Resistor*"
Flexibility Level	3
Connector generated	no

The *Class Name* tells from which class the equation is coming. This annotation is extremely useful in exactly locating the associated class of the equation and therefore providing concise error messages to the user.

The *No. of associated eqs.* parameter specify the number of equations which are specified together with the annotated equation. In the above example the *No. of associated eqs.* is equal to one since there are no additional equations specified in the `Resistor` component. In the case of the `TwoPin` component the number of associated equations is equal to 3. If one associated equation of the component need to be eliminated the value is decremented by 1. If, for example, during debugging, the equation `R1.i * R1.R == R1.v` is diagnosed to be an over-constraining equation and therefore need to be eliminated, the elimination is not possible because the model will be invalidated in that way (the *No. of associated eqs.* cannot be equal to 0) and therefore other solutions need to be taken into account.

The *flexibility level*, in a similar way as it is defined in (Flannery and Gonzales [5]), allows the ranking of the relative importance of the constraint in the overall flattened system of equations

The *Connector generated* is a Boolean attribute which tells whether the equation is generated or not by a `connect` statement. Usually these equations have a very low flexibility level.

It is worth nothing that the annotation attributes are automatically initialized by the static analyzer, incorporated in the front end of the compiler, by using several graph representations.

Having the equations annotated, the next step is to traverse the associated directed graph, shown in Figure 6, to the over-constraining part, obtained from the D&M decomposition.

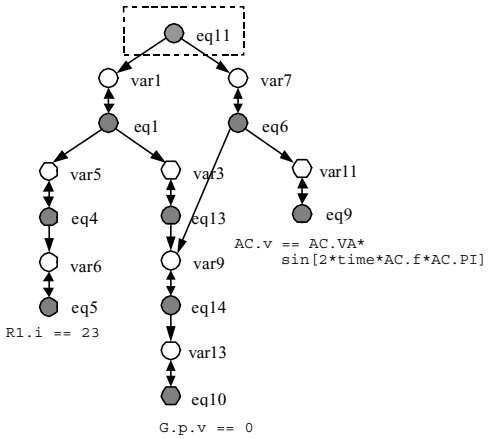


Figure 6. The associated directed graph of the ove-constraining subgraph.

One important property of the over-constrained bipartite graph is that it only contains alternating paths because it is constructed from perfect matchings and a supplementary free edge. We can easily obtain all the maximal matchings in the over-constrained graph by exchanging matching edges with other edges along an alternating path. Therefore eliminating any of the constitutive nodes that represent an equation we can easily find a corresponding matching of the sub-graph will yield to a well-constrained subsystem. But eliminating some of the constitutive nodes that represent equations will disconnect the sub-graph as it is illustrated in Figure 7 where *eq1* was eliminated. Even if this situation, when two disconnected graphs are obtained, are mathematically sound they are not very common from the modeling point of view and therefore they are not further considered.

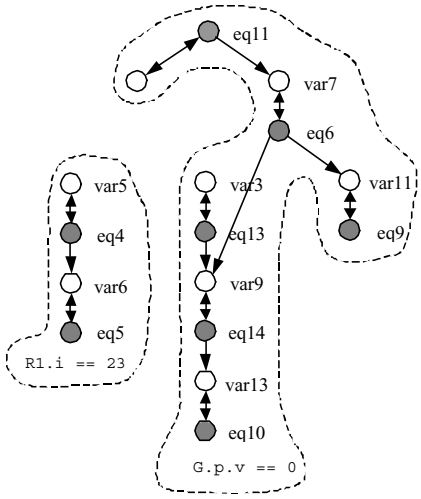
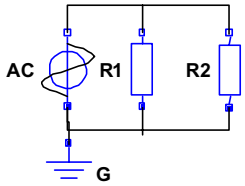


Figure 7. Disconnected graph obtained by eliminating *eq1*.

In our case the set of equivalent over-constraining equations is {*"eq11", "eq13", "eq10", "eq5", "eq9"*} after eliminating those equations which disconnect the bipartite graph. "*eq11"* was already analyzed and therefore can be eliminated from the set. "*eq13"* is eliminated too for the same reasons as equation "*eq11*". Analyzing the remaining equations {*"eq10", "eq5", "eq9"*} one should note that they have the same flexibility level and therefore they are candidates for elimination with an equal chance. But analyzing the value of the *No. of associated eqs.* parameter, equation "*eq10*" and "*eq9*" have that attribute equal to one, which means that they are singular equations defined inside the model. Eliminating one of these equations will invalidate the corresponding model, which is probably not the intention of the modeler.

Examining the annotations corresponding to equation "*eq5*" one can see that it can be safely eliminated because the flexibility level is high and eliminating the equation will not invalidate the model since there is another equation defined inside the model. After choosing the right equation for elimination the debugger tries to identify the associated class of that equation based on the *Class name* parameter defined in the annotation structure. Having the class name and the intermediate equation form `(R1.i=23)` the original equation can be reconstructed `(i=23)` indicating exactly to the user which equation needs to be removed in order to make the simulation model mathematically sound. In that case the debugger correctly locates the faulty equation previously introduced by us in the simulation model.

We now construct a simple electrical circuit model (Figure 8) by connecting two resistors in parallel with a voltage source as is shown in Figure 9. The Modelica definition of the `Ground`, `VsourceAC` and `Resistor` component are reused from the previous examples. The `TwoPin` class is modified by introducing an additional over-constraining equation `(i=10)` in the model definition. This extra equation will be inherited by all the classes which extends the `TwoPin` class. Therefore each instance of the `Resistor` and `VsourceAC` models will contribute to one extra over-constraining equation to the final flattened system of equations.




```
model TwoPin
   Pin p,n;
   Real v,i;
equation
   v=p.v-n.v;
   0=p.i+n.i;
   i=p.i;
   i=10;
end TwoPin
```

Figure 8. An electrical circuit with an over-constraining equation in the `TwoPin` component.

During the model translation the corresponding flattened set of equations from the simulation model is derived and the associated bipartite graph *G* is constructed. The overall flattened model corresponding to the simple electrical circuit contains three extra over constraining equations (*eq9, eq18, eq7*). Therefore three vertices from the equations sets are not covered by a matching, as it is illustrated in the derived directed graph in Figure 9.

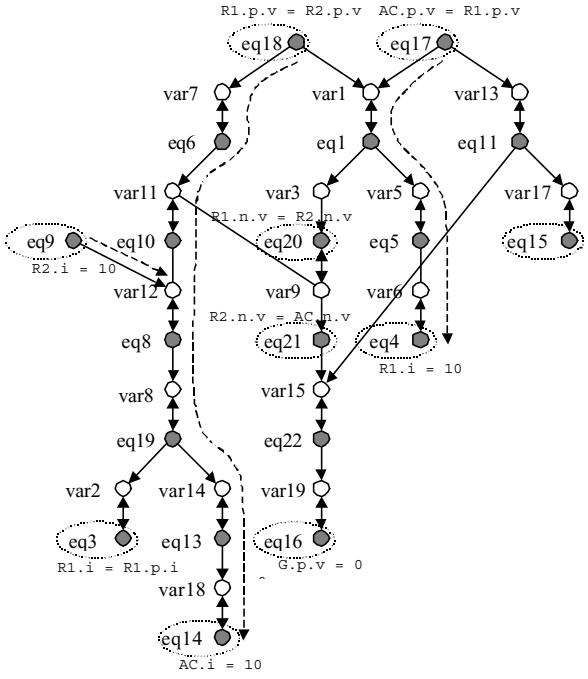


Figure 9. The over-constrained directed graph

While traversing the directed graph, after eliminating all the equations that disconnect the graph and performing reasoning based on the equation annotations we found that the equations *eq9 eq14, eq15* need to be eliminated from the intermediate form which are generated from the equation `i=10` in the `TwoPin` partial component. The elimination of *eq9, eq14* and, *eq15* is safe because they can be made free vertices by exchanging the matching edges with non matching edges along the paths indicated with dashed lines in Figure 9.

4 Debugging of Under-Constrained Models

The issue of under-constrained simulation models considered in an object-oriented declarative equation-based frameworks, has been discussed in (Ramirez [10]). The work presented in (Ramirez [10]) is particularly concerned with the issues involved in the modeling and solutions of conditional models where the system of equations in the model is different for each of the alternatives.

Let us consider the number of equations *m* from a model and the number of variables incident in those equations *n*. For a typical under-constrained situation the number of variables is greater than the number of equations *(n>m).*

Definition 7: We call the *degree of under-constraining* the difference between the number of variables and the number of equations $D_u = n - m$. In a similar way in (Ramirez [1]) D_u is called the number of degrees of freedom of the problem.

In the following we are going to illustrate the possible error fixing solutions for a typical under-constraining situation and the reasoning involved in the graph transformation system. Let us consider the following system of equations with the corresponding bipartite graph presented in Figure 11 and with the degree of under-constraining $D_u = 1$.

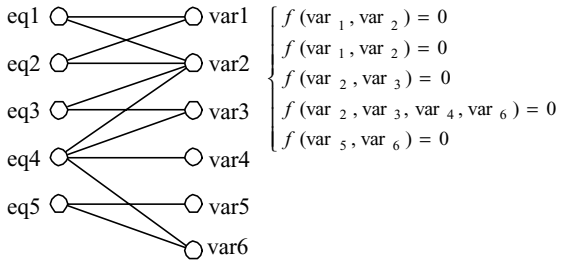


Figure 10. A simple system of equations with the associated bipartite graph.

One possible corresponding maximal matching (represented by thicker edges) to the bipartite graph and the D&M canonical decomposition is presented below:

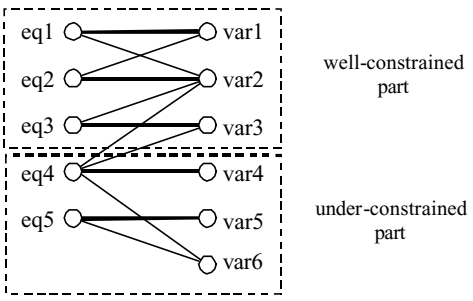


Figure 11. Maximum matching and canonical decomposition of the bipartite graph

In performing the canonical decomposition algorithm the associated directed graph to the bipartite graph was constructed by exchanging all the edges which are part of the maximal matching by bi-directional edges and orienting all other edges from equations to variables. The obtained directed graph is shown in Figure 12:

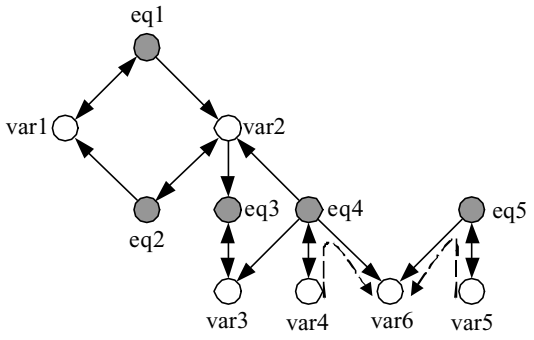


Figure 12. Directed graph associated with the system of equations.

The variables contained in an under-constrained part constitutes the eligibility set. In our small example the elibility set is {*var4, var5, var6*} which means that any of these variable can be taken away and the remaining associated graph will be well constrained.

Variable *var6* is not covered by the maximal matching and therefore is a free vertex. In the directed graph, it can be seen that these are two alternating paths to the free vertex *var6* (indicated by the dashed arrows in Figure 12):

$\{(\text{var}_4, eq_4), (eq_4, \text{var}_6)\}$ and $\{(\text{var}_5, eq_5), (eq_5, \text{var}_6)\}$.

Exchanging the matching edges with normal edges and the normal edges with matching edges along an alternating path a new matching can be obtained which cover the free vertex *var6* but will uncover another vertex from the eligible set. Therefore for an error fixing strategy all the possible combinations should be taken into account.

During the first stage of the error fixing process only those solutions which involve the elimination of a variable from the eligibility set are taken into account. We have the following possible solutions illustrated in Figure 13 .

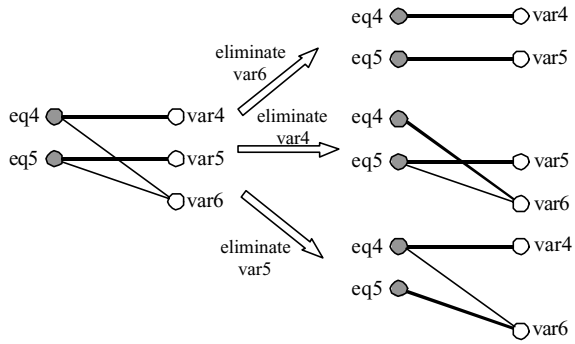


Figure 13. Error fixing solution when one variable is taken away from the eligibility set.

By removing *var6* from the under-constrained sub system the considered maximum matching becomes a perfect matching of the associated bipartite graph and therefore the associated system of equations is structurally sound. However, by removing *var6* the bipartite graph will be disconnected and an independent edge (*eq5,var5*) appears in the system, which is not connected to the main bipartite graph. This situation is extremely unusual in physical system modeling and it means that some variables are computed locally inside a component without contributing to the general behavior of the simulated system. As an example the following Modelica `Resistor` component integrated in a circuit model will produce two disconnected sub-graphs.

```
model Resistor
   extends TwoPin;
   parameter Real R;
   Real s;
equation
   R*i=v;
   s=10;
end Resistor
```

The variable `s` and the equation `s=10` are redundant in the system. Therefore the situation when an extra variable is eliminated and the remaining bipartite graph is disconnected needs to be further analyzed. In our case, for example, a solution which involves the elimination of variable *var6* and the presence of an extra variable *var1, var2, var3* or *var4* in equation *eq5* might be acceptable.

It should also be noted that multiple error fixing strategies are possible in the case of an under-constrained subsystem. Another error fixing situation for the under-constrained systems is to add one extra equation to the system and link the free variable to the added equation instead of eliminating the free variable. This strategy applied for the free variable *var6* is presented in Figure 14.

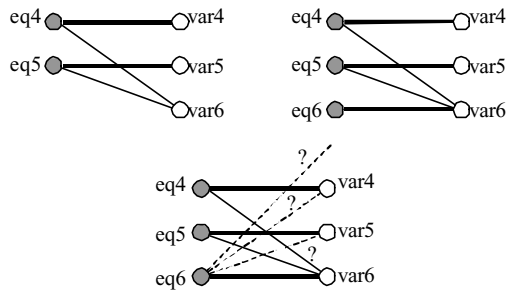


Figure 14. Error fixing strategy involving adding an extra equation.

This strategy involves two steps: at the first step an extra equation is added and linked together with the free variable and then at the second step is checked if other variables from the system might be present in the recently added equation. This last step turns out to be very useful from the users point of view because is helps them to reconstruct missing equations from simulation models.

Let us again analyze the simple circuit model when the `Resistor` component is changed again by declaring an extra variable (`Real s`) and introducing this variable into the `Resistor` component constitutive equation.

```
model Resistor
   extends TwoPin;
   parameter Real R;
   Real s;
equation
   R*i=v*s;
end Resistor
```

The directed graph obtained from the associated bipartite graph of the flattened underlying system of equations and a corresponding maximum cardinality matching, is given below:

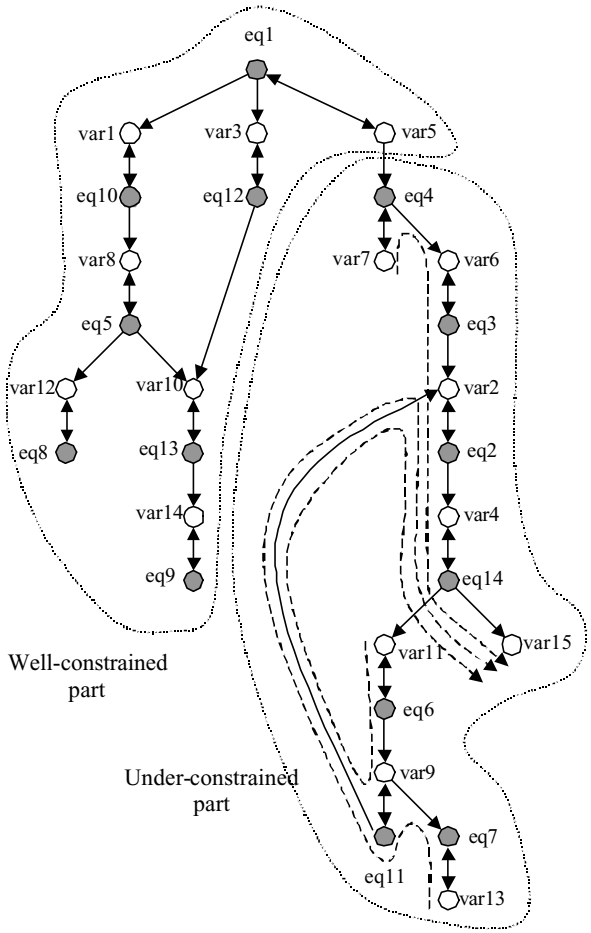


Figure 15. Directed graph corresponding to the under-constrained simple electrical circuit**.**

The uncovered variable by the considered maximum cardinality matching is *var15,* the eligibility set being:

$$\{var_{15}, var_4, var_2, var_6, var_7, var_{11}, var_9, var_{13}\}$$

with the corresponding variables:

{G.p.i, R.n.i, R.p.i, R.i, R.s, AC.n.i, AC.p.i, AC.i}

From the under-constrained subgraph we can derive the following alternating paths (indicated in Figure 15 by the dashed arrows) to the uncovered variable:

$$\{(\mathrm{var}_{15}, eq_{14}), \overline{(eq_{14}, \mathrm{var}_4)}, (\mathrm{var}_4, eq_2), \overline{(eq_2, \mathrm{var}_2)}$$
$$(\mathrm{var}_2, eq_3), \overline{(eq_3, \mathrm{var}_6)}, (\mathrm{var}_6, eq_4), \overline{(eq_4, \mathrm{var}_7)}\}$$

$$\{(\mathrm{var}_{15}, eq_{14}), \overline{(eq_{14}, \mathrm{var}_4)}, (\mathrm{var}_4, eq_2), \overline{(eq_2, \mathrm{var}_2)}$$
$$(\mathrm{var}_2, eq_{11}), \overline{(eq_{11}, \mathrm{var}_9)}, (\mathrm{var}_9, eq_6), \overline{(eq_6, \mathrm{var}_{11})}\}$$

$$\{(\mathrm{var}_{15}, eq_{14}), \overline{(eq_{14}, \mathrm{var}_4)}, (\mathrm{var}_4, eq_2), \overline{(eq_2, \mathrm{var}_2)}$$
$$(\mathrm{var}_2, eq_{11}), \overline{(eq_{11}, \mathrm{var}_9)}, (\mathrm{var}_9, eq_7), \overline{(eq_7, \mathrm{var}_{13})}\}$$

By following each alternating path and eliminating the variables from the eligibility set it can be noticed that eliminating only the variables $\{\mathrm{var}_{15}, \mathrm{var}_4, \mathrm{var}_7, \mathrm{var}_{13}\}$ will not disconnect the bipartite graph. Therefore only this reduced set will be further analyzed at this stage. Based on a similar reasoning as in the over-constrained situations and on variable associated annotations, the results is that only *var7* can be safely removed from the Modelica code in order to obtain a well specified underlying equation system. We call the set of variables obtained after performing the reasoning based on annotations *the reduced eligibility set*

In the above presented situation the fault was detected during the first stage of the debugging of under-constrained equations. But if the user is not satisfied with the given solution or the reduced eligibility set is empty, the debugger can enter in the second stage when possible connections of the adjacent equations to those variables that disconnect the bipartite graph are checked. If a possible coupling of a variable to those equations is found the adjacent disconnecting variable might be also considered for elimination. The possible coupling of variables with equations is performed by a *variable reachability analysis* based algorithms applied to the inheritance graph of the underlying simulation system. The variable reachability analysis computes the list of variables which can be inserted into certain equations. The description of the variable reacheability analysis algorithm is not the subject of this paper.

A third stage in the debugging process of the under-constraining equations is when extra equations need to be added and coupled to the free equation. For example, in our case, adding an extra equation `s=10` in the `Resistor` component is a mathematically sound solution even it might not reflect the modelers intent. In a similar way extra equations can be added to each variable from the eligibility set.

The user has the possibility of specifying which level of debugging he/she would like to perform on the erroneous model, in that way, filtering out some of error messages and performing an incremental error fixing on the modeling source code.

5 Prototype Implementation

A prototype debugger has been built and attached to the *MathModelica* simulation environment as a testbed for evaluating the usability of the above presented graph decomposition techniques for debugging declarative equation based languages. *MathModelica* is an integrated problem-solving environment (PSE) for full system modeling and simulation (Fritzson et. al.[6]) (Jirstrand [7]) (Jirstrand et. al.[8]). The environment integrates Modelica-based modeling and simulation with graphic design, advanced scripting facilities, integration of code and documentation, and symbolic formula manipulation provided via *Mathematica* (Wolfram [11]). Import and export of Modelica code between internal structured and external textual representation is supported by *MathModelica*. The environment extensively supports the principles of literate programming and integrates most activities needed in simulation design: modeling, documentation, symbolic processing, and transformation and formula manipulation, input and output data visualization.

In order to attach the debugger it was absolutely necessary to have access to the intermediate form of the code because the presented algorithm makes use of the intermediate flat form of the equations. The implemented debugger was successfully tested on Modelica models involving several hundreds of algebraic and differential algebraic equations.

The general architecture of the implemented debugger is presented in Figure 16. The debugging algorithm proceeds as follows: based on the original declarative source code the intermediate representation is generated. From the intermediate representation the overall system of equations is extracted and transformed into bipartite graph form. The associated bipartite graph is canonically decomposed. Error-fixing strategies are applied if the decomposition leads to over- or under-constrained components. The debugger will try to solve the errors automatically without explicit intervention of the user. If automatic error solving is not possible due to missing information the user will be consulted regarding the repair strategy.

When the user is interrogated, all valid options that will lead to a structurally sound underlying system of equations are presented. As was mentioned earlier, the error fixing strategies for over- and under-constrained subcomponents might involve several stages, especially for under-constrained situations. Due to the equation and variable annotations the error messages output by the debugger are understandable relative to the user perception of the simulation source code, in our case the Modelica code.The information output by the debugger will of course lead to a mathematically sound system of equations. However, some of the solutions might not be acceptable from the modeling language point of view or from the physical system model perspective. The debugger focuses on those errors whose identification would not require the solution of the underlying system of equations.

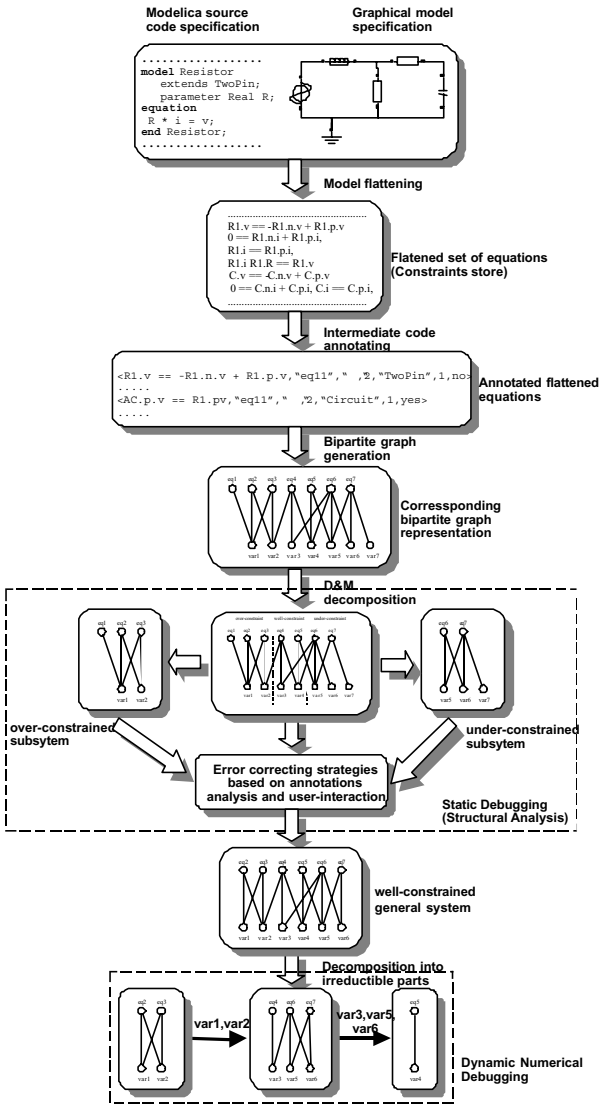


Figure 16. Debugger architecture.

6 Conclusions

Determining the cause of errors in models of physical systems is hampered by the limitations of the current techniques of debugging declarative equation based languages. We have presented a new approach for debugging such languages by employing graph decomposition techniques and have given several usage examples for debugging erroneous models. It has also been demonstrated that it is possible to create a tool with an enhanced user interaction capability that can be used explicitly in understanding complicated simulation models.

The contributions of this paper are twofold: the proposal of integrating graph decomposition techniques for debugging declarative equation based languages and an efficient equation annotation structure which helps the debugger to eliminate some of the heuristics involved in the error solving process. The annotations also provide an efficient way of identifying the equations and therefore helps the debugger in providing error messages consistent with the user's perception of the original source and simulation model. The implemented debugger helps to statically detect a broad range of errors without having to execute the simulation model. Since the simulation system execution is expensive the implemented debugger helps to greatly reduce the number of test cases needed to validate a simulation model.

The merits of the proposed debugging technique are as follows:

- The user is exposed to the original source code of the program and is therefore not burdened with understanding the intermediate code or the numerical artifacts for solving the underlying system of equations.
- The user has a greater confidence in the correctness of the simulation model.
- The error fixing strategies are also prioritized by the debugger, which benefits the user in choosing the right error fixing solution.

References

[1] Asratian A.S.; Denley T. and Häggkvist R. *Bipartite Graphs and their Applications*. Cambridge University Press 1998.

[2] Dolan A. and Aldous J. *Networks and algorithms – An introductory approach*. John Wiley & Sons 1993 England.

[3] Dulmage, A.L., Mendelsohn, N.S. *Coverings of bipartite graphs,* Canadian J. Math., 10, 517-534.

[4] Elmqvist, H. *A Structured Model Language for Large Continuous Systems*. PhD thesis TFRT-1015, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden. 1978.

[5] Flannery, L. M. and Gonzalez, A. J. *Detecting Anomalies in Constraint-based Systems*, Engineering Applications of Artificial Intelligence, Vol. 10, No. 3, June 1997, pages. 257-268.

[6] Fritzson P.; Gunnarsson J; Jistrand M.;. "MathModelica - An Extensible Modeling and Simulation Environment with Integrated Graphics and Literate Programming". *In Proceedings of the 2nd International Modelica Conference* (March 18-19, Munich, Germany, 2002)

[7] Jirstrand, M.; Gunnarsson J. and Fritzson P. "MathModelica – a new modeling and simulation environment for Modelica. " *In Proceedings of the Third International Mathematica Symposium* (IMS'99, Linz, Austria, Aug), 1999.

[8] Jirstrand, M."MathModelica – A Full System Simulation Tool". In Proceedings of *Modelica Workshop 2000* (Lund, Sweden, Oct. 23-24),2000.

[9] Maffezzoni C.; Girelli R. and Lluka P. *Generating efficient computational procedures from declarative models.* Simulation Practice and Theory 4 (1996) pages 303-317.

[10] Vicente Rico Ramirez. Representation, Analysis and Solution of Conditional Models in an Equation-Based Environment. PhD Thesis, Carnegie Melon University, Pittsburgh, Pennsylvania, August 1998.

[11] Wolfram S. *The Mathematica Book* . Wolfram Media Inc. (February 1996)