Transient Analysis of Linear Circuits using Constraint Logic Programming

Archana Shankar, David Gilbert, Michael Jampel

{shankar,drg,jampel}@cs.city.ac.uk

Department of Computer Science, City University
London EC1V 0HB, UK

Abstract

This paper describes the design of a transient analysis program for linear circuits and its implementation in a Constraint Logic Programming language, CLP(R). The transient analysis program parses the input circuit description into a network graph, analyses its semantic correctness and then performs the transient analysis. The test results show that the program is at least 97% accurate when run at two decimal places. We have also compared the performance of our program with a commercial package implemented in an imperative language. The advantages of implementing the analysis program in a CLP language include: quick construction and ease of maintenance and generation of a circuit with given characteristics.

1.0 INTRODUCTION

Electrical circuit analysis is done in order to allow the designer to verify his design and to predict the response of the system under varying conditions of load and excitation. Traditional methods of circuit analysis make use of data driven simulations which are sometimes not adequate as many iterations have to be carried out before the one satisfying the performance requirements of the designed item are met.

Viewed from another angle, the analysis can be considered to be a goal driven task as there is a final state that the system has to attain. Constructing a goal driven simulation is useful in design as it frees the designer from repeated simulation. The final state the system should attain and the composition of the circuit are inputs and the program can work out the component values such that they satisfy the specifications. Using Constraint Logic

Programming (CLP), we have implemented a system that is both data driven as well as goal driven.

A circuit can be analysed at different points in time. Whenever a switch is closed in a circuit, the voltages and currents in the circuit take some time to settle down to their final values; the components of voltages and currents that die down are called as *transients*. *Steady state* analysis of a circuit is done when the system is in a steady state. *Transient analysis* of a circuit is done *at the time of switching* to study the effects of the transients as well as to determine the time taken by the system to settle down.

The behaviour of the circuit as a function of time is studied under transient analysis. The inductors in the circuit are replaced by their equivalent current sources and resistances, and the capacitors in the circuit are replaced by their equivalent voltages sources and resistances. The circuit voltages and currents are calculated at the time of switching (usually at $t = 0$); this is the initial condition solution. The voltages across the capacitors and the currents across the inductors are used to calculate the circuit voltages and currents at each time step; this is done repeatedly for a designated amount of time and the results are then plotted.

In our work, we implement the transient analysis program of linear circuits using Constraint Logic Programming (CLP) [JAFF87]. Constraint programming can help build more powerful simulation environments by providing a flexible way to perform traditional data driven simulations as well as goal driven simulation. Constraints provide an elegant means of stating relationship among objects that should be maintained by the underlying system. The Constraint Logic Programming scheme is an extension to logic programming. Our transient analysis program is implemented in CLP(R) [JAFF92] using the notion of parent and leaf constraints [HEIN92]. This gives us the advantage of being able to use the same program not only being to make an analysis of a given circuit, but also to generate a circuit from a general schema and a description of its required behaviour. The program developed for analysis of a circuit can also be used for design purposes as well by binding the output voltages to certain values and asking the CLP system to generate the necessary components such that the inputs satisfy the output.

OVERVIEW OF THE WORK

This paper discusses transient analysis of linear circuits and its implementation in CLP(R). Section 2 discusses analysis of linear circuits, and develops the relationships required to do transient analysis program of a general network. Section 3 gives the high level design of the transient analysis program which uses the matrix method of node analysis. This method is general in nature and can be extended to include non-linear networks as well. Section 4 is the implementation of the program in CLP(R). Section 5 gives details of the tests conducted as well as the comparison of our program with a commercial package. Section 6 discusses related work and Section 7 discusses some directions for further work. Finally Section 8 summarises the paper and gives our conclusions.

2.0 BACKGROUND

Electrical *circuits* or *networks* consist of an aggregate of electrically (and sometimes magnetically) connected building blocks. Each building block has two or more electrical connections. Examples of building blocks with two connections are resistors, inductors, capacitors, voltage sources and current sources. The junction of two or more such connections is called a *node*. Voltages across the building blocks within the network are measured as differences between the voltages at the nodes. Currents that leave the nodes, the *branch* currents, flow through the circuit components thus generating the voltage differences between the nodes to which the components are connected. The calculation of voltages and currents in a circuit and its response to input excitations form the basis for analysing a network. The mathematical tools involved in analysis are the basic voltage-current relationships for the circuit elements, rules for generating simultaneous equations, and solution procedures for these equations.

A network needs to be analysed in order to derive information about it so that its predicted behaviour can be verified. The task of analysis starts off with a mathematical model chosen by the designer to represent the behaviour of an actual circuit. If the behaviour of the model does not reflect the behaviour of the actual circuit, the circuit parameters of the model are adjusted in order that the response conforms to the specification. Once the response of the model is within acceptable limits, detailed analysis of the circuit responses and detailed calculations of the effects of component variations must be made. If there is a discrepancy, the circuit is re-modelled and re-analysed. The mathematics involved is the solution of a set of simultaneous differential equations. These equations can be linearised for linear circuits (refer to section 2.1.4 for the inductor and capacitor models).

A circuit with an excitation voltage or current of x and an output voltage or current y is ***linear*** *if y is proportional to x i.e. if $y = f(x)$ or $ky = f(kx)$.* Any circuit which is not linear is called *non-linear*. Almost all physical components in the circuit are in fact non-linear owing to effects such as aging, internal heating, voltage breakdown and magnetic core saturation, which alter the component value and depend on the applied voltage and its frequency. However, the linearity assumption greatly simplifies circuit analysis, and so it is important to make this assumption even if the circuits are only approximately linear.

The linearity assumption has several important consequences. If we consider a change of Δx in the excitation x, a corresponding change Δy is produced in y. In a linear circuit the changes are proportional. Therefore

$$\Delta y / y = \Delta x / x = k$$

and $$\Delta y = f(\Delta x).$$

The changes in the circuit voltages and currents produced by variation of the fixed sources are therefore independent of the nominal values which exist in the circuit. Also since

$$1 + (\Delta y / y) = 1 + (\Delta x / x) = 1 + k,$$

$$(y + \Delta y) = f(x + \Delta x),$$

the effects of source variation are additive to the nominal values.

If the variations Δy and Δx are small compared with the nominal values x and y, the analysis for variation only is called small signal analysis. As noted earlier, all practical circuits are non-linear. Fortunately, many circuits operate in regions of near linearity and distortion due to non-linearity is usually small enough to ignore. Circuits in which this cannot be assumed cause major errors when linear analysis is employed, and special analysis techniques are required. Models of non-linear elements such as diodes can be linearised by replacing the non-linear element with a combination of sources and linear elements. The resulting linear circuit is valid, provided that small signals are assumed that do not deviate significantly from the nominal or bias values.

The tools involved in analysis are the basic voltage-current relationships for the circuit elements, rules for generating simultaneous equations, and solution procedures for these equations.

In the following sections, the necessary equations for solving these equations in a linear circuit will be developed. Most of the equations are represented using matrix notation and require a working knowledge of matrix algebra [NERI70].

2.1 TRANSIENT ANALYSIS OF LINEAR CIRCUITS

The "node" method of analysis is used for systematic analysis of *lumped component circuits*. In any circuit, current varies with time, and the electromagnetic energy is radiated and lost. When the wavelength of the generated electromagnetic wave is usually large in comparison with the physical dimensions of the circuit, the energy loss is negligible. *Circuits in which radiated electromagnetic energy may be assumed negligible are called* ***lumped circuits*** *and they are obtained by interconnecting* ***lumped component circuit elements***. The lumped component circuit elements are idealised models of physical elements. The circuit analysed is therefore not the physical circuit, but its mathematical model.

The node method consists of summing the currents at each node in the network with unknown voltages at the nodes and solving the resultant equations for the voltages. The reason for choosing this method of analysis is that in a network, the node voltages are uniquely determined and the current summations at each node yield a sufficient number of equations to permit the solution of the node voltages.

The basic relations for circuit analysis are the first two laws of Kirchoff [SESH63]:

1. The sum of all currents entering a node must equal the sum of all currents leaving it (current law or KCL).

2. The sum of all voltages in a given loop must be equal to zero (voltage law or KVL).

In analysing a network, one or the other of the above laws is applied to every independent node or independent loop of the network. The number of independent nodes is generally less than the number of independent loops in a network (i.e. there are more parallel paths between two nodes than there are serial elements in loops) and hence there are fewer node equations than loop equations. Circuit analysis programs generally use node voltage equations for analysing the network.

2.1.1 BASIC RELATIONSHIPS

Let the branches in the network be labelled uniquely and sequentially starting from one. Let the nodes be labelled uniquely and sequentially starting from 0. The node given the number zero is the **reference node** or the **ground node**. The circuit connections can be represented by:

$$\mathbf{A}_a\mathbf{I}_b = \mathbf{0} \qquad (2.1.1)$$

where $\mathbf{I}_b$ is the branch current matrix (the current i_j is the current flowing in branch j). The elements of the matrix $\mathbf{A}_a$ are given by :

$$\begin{aligned} a_{mn} &= 1 \quad \text{if the current } i_n \text{ leaves node } m \qquad (2.1.2)\\ &= -1 \quad \text{if the current } i_n \text{ enters node } m \\ &= 0 \quad \text{if the current } i_n \text{ neither enters nor leaves node } m. \end{aligned}$$

These valuations hold for all the nodes in the circuit. Each equation is the application of Kirchoff's current law (KCL) for the *m*th node. For N independent nodes, Eq. (2.1.1) is represented by

$$\mathbf{A}\mathbf{I}_b = \mathbf{0} \qquad (2.1.3)$$

where matrix **A** has the elements given by Eq.(2.1.2) and contains *N* columns (number of branches) and *M* rows (number of independent nodes).

Voltage-current relationships for passive elements in a linear circuit are characterised by a single (complex) number (the number is complex for alternating current (AC) circuit analysis):

$$i = y.v \qquad (2.1.4)$$

with y = 1/R for a resistor, 1/(jω)L for an inductor, or (jω)C for a capacitor where R is the resistance, L is the inductance, C is the capacitance, y is the admittance and ω is the frequency; in the AC analysis, i and v are complex valued phasor representations for the current and voltage.

As a result of the branch currents flowing through the various branches, a voltage is developed across the respective branches. These voltages can be represented by:

$$\mathbf{V}_b = \mathbf{B}\,\mathbf{V}_n \qquad (2.1.5)$$

where $\mathbf{V}_n$ represents the node voltage matrix (voltage v_j is the voltage measured at node j) and is a column vector containing n rows. Node voltages are measured with respect to the ground which is at zero potential. The elements of vector **B** are given by

$$\begin{aligned} b_{mn} &= 1 \quad \text{if the branch m leaves node } n \qquad (2.1.6) \\ &= -1 \quad \text{if the branch m enters node } n \\ &= 0 \quad \text{if the branch m neither enters nor leaves node } n. \end{aligned}$$

From Eqs. (2.1.2) and (2.1.6) we find that

$$\mathbf{B} = \mathbf{A}^T \qquad (2.1.7)$$

where $\mathbf{A}^T$ is the transpose of matrix **A**.

This implies that the direction of the current in each branch is the direction of positive voltage to negative voltage in each branch and the number of elements in $\mathbf{I}_b$ and $\mathbf{V}_b$ are the same. (i.e. i_2 refers to the current in branch 2 and v_2 refers to the voltage in branch 2); hence the same branch numbering and the same node numbering apply to both Eqs. (2.1.3) & (2.1.5).

A branch can represent either an active element (i.e. a voltage or current source) or a passive element (i.e. an inductor or a capacitor or a resistor). If a branch has an independent current source (represented by the vector $\mathbf{I}_g$), then the current leaving that branch is the difference of the currents caused by the independent source and all other currents (represented by $\mathbf{I}_b^*$). The branch current will then be

$$\mathbf{I}_b = \mathbf{I}_b^* - \mathbf{I}_g = \mathbf{Y}_b\mathbf{V}_b - \mathbf{I}_g \qquad (2.1.8)$$

where $\mathbf{Y}_b$ is the *branch admittance matrix.*

Combining Eqs. (2.1.8), (2.1.5) and (2.1.3) we get the relationship

$$\mathbf{A}\mathbf{Y}_b\mathbf{A}^T\mathbf{V}_n = \mathbf{A}\mathbf{I}_g \qquad (2.1.9)$$

$\mathbf{A}\mathbf{Y}_b\mathbf{A}^T$ is called the *node-admittance matrix* $\mathbf{Y}_n$.

2.1.2 ANALYSIS OF A NETWORK

In the analysis of linear networks, various element types must be considered. The elements that are used in this analysis is a minimum subset of elements that are present in a network . The elements used in our program are:

(1) Passive Elements

- a) Resistance (R)
- b) Inductance (L)
- c) Capacitance (C)

(2) Active Elements (Sources)

- a) Independent voltage source (V)
- b) Independent current source (I)

There are two types of sources, accompanied and unaccompanied. If a source is accompanied, it implies that there is a small resistance in series with a voltage source and a small resistance in parallel with a current source. Unaccompanied sources are pure current and voltage sources and do not have any accompanying resistances. In our analysis we will consider only accompanied voltage sources as unaccompanied sources present difficulties when forming the branch admittance matrices. Since the accompanying resistance R is zero, the admittance (1/R), becomes infinite and the current through the branch becomes indeterminate.

The element definitions are consistent with the universally used definitions of the basic network elements and are valid for direct current (DC) transient analysis.

A general network of the type shown in Fig. 2.1 and made up of M branches and N nodes will be considered for analysis. The voltages and currents in each branch of the network are related as follows:

$$i_b = i_e - i_g \qquad (2.1.10)$$

$$v_b = v_e - v_g \qquad (2.1.11)$$

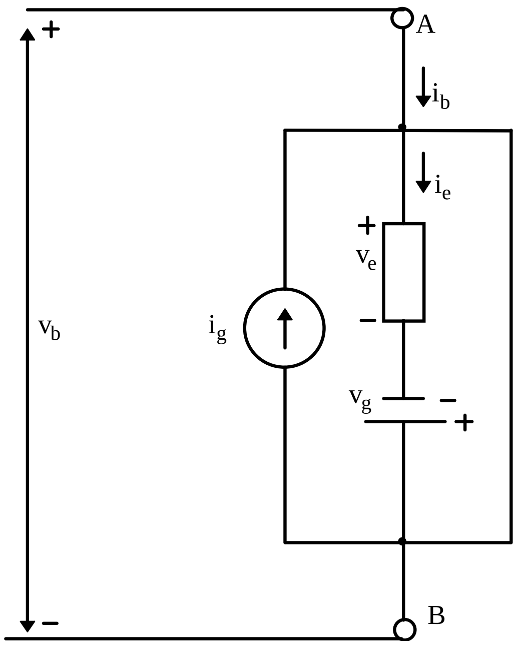
The element voltage and current also satisfy the relationship

$$i_e = y_e . v_e \qquad (2.1.12)$$

where y_e is the conductance of the passive element in the branch. The above three equations hold for all branches in the network.

The branch currents i_{b1}, i_{b2} ... i_{bM} , are represented by a matrix as:

$$\mathbf{I}_b = \begin{bmatrix} i_{b1} \\ i_{b2} \\ . \\ . \\ i_{bM} \end{bmatrix} \qquad (2.1.13)$$



ib	= branch current
vb	= branch voltage
ie	= element current
ve	= element voltage
ig	= independent current
vg	= independent voltage
A	= From-node
B	= To-node

Fig 2.1 A network branch

Similarly the M-vectors for $\mathbf{I}_e$, $\mathbf{I}_g$, $\mathbf{V}_e$, $\mathbf{V}_b$, $\mathbf{V}_g$ are defined. The currents and voltages for all the network branches satisfy the following vector equations:

$$\mathbf{I}_b = \mathbf{I}_e - \mathbf{I}_g \qquad (2.1.14a)$$

$$\mathbf{V}_b = \mathbf{V}_e - \mathbf{V}_g \qquad (2.1.14b)$$

The voltage-current relationships for the passive elements lead to the matrix equation

$$\mathbf{I}_e = \mathbf{Y}_e.\mathbf{V}_e \qquad (2.1.14c)$$

$\mathbf{Y}_e$ is a diagonal matrix of M rows and M columns if the network consists solely of resistors, inductors and capacitors.

Form Eqs (2.1.14 a, b & c) we get

$$\mathbf{I}_b = \mathbf{Y}_e (\mathbf{V}_b + \mathbf{V}_g) - \mathbf{I}_g \qquad (2.1.15)$$

By Kirchoff's current law

$$\mathbf{A}\mathbf{I}_b = \mathbf{0} \qquad (2.1.16)$$

where **A** is the MxN *incidence* matrix of the network. Hence

$$\mathbf{A}[\mathbf{Y}_b(\mathbf{V}_b+\mathbf{V}_g) - \mathbf{I}_g] = \mathbf{0} \qquad (2.1.17)$$

The node voltages are related to the branch voltages by

$$\mathbf{A}^T \mathbf{V}_n = \mathbf{V}_b \qquad (2.1.18)$$

Thus, the node voltage expression becomes

$$\mathbf{A}\mathbf{Y}_b\mathbf{A}^T\mathbf{V}_n + \mathbf{A}\mathbf{Y}_b\mathbf{V}_g - \mathbf{A}\mathbf{I}_g = \mathbf{0} \qquad (2.1.19)$$

If the interconnection of the circuit (i.e. the incidence matrix) and the independent voltages and currents are known, then the node voltages can be solved using the above equation.

If the node voltages are known then, the branch voltages and currents can be solved from Eqs. (2.1.18) and (2.1.15). Finally, the element voltages and currents can be solved for using Eqs. (2.1.14b) and (2.1.14c).

2.1.3 TRANSIENT ANALYSIS RELATIONSHIPS

The analysis in the previous sub-section dealt with voltages and currents in a network which did not vary with time; however an important problem in network analysis is to calculate those voltages and currents as functions of time. One of the approaches used to deal with this problem is to replace the inductors and capacitors by equivalent conductance and source combinations and then solve the resultant set of simultaneous algebraic equations.

THE INDUCTOR MODEL

The voltage-current relationship for an inductor of value L is

$$i(t) = 1/L \int_{t=0}^{t} v(t)dt + i(0) \qquad (2.1.20)$$

where i(0) is the current in the inductor at time t=0. The voltage values are calculated at equally spaced time intervals t_0, t_1 ... t_{k-1}, t_k

The integral can be replaced by trapezoid approximation

$$\int_{t=0}^{t} v(t)dt \approx 1/2\ \Delta t(v_{k-1} + v_k) \qquad (2.1.21)$$

with Δt being the time step. The current is now a linear relationship

$$i_k = i_{k-1} + 1/2\ \Delta t\ (v_{k-1} + v_k) \qquad (2.1.22)$$

The last equation is of the form

$$i = I + g.v \qquad (2.1.23)$$

and represents a parallel combination of a current source and a conductance

$g = \Delta t/2L$.

The value of g will not change during the entire calculation. However, the value of I must be readjusted at every new time value. The voltages and currents in Eq. (2.1.22) are the element voltages and currents across the inductor not the branch voltages and currents.

THE CAPACITOR MODEL

The voltage-current relationship for a capacitor C is

$$i(t) = Cdv(t)/dt \qquad (2.1.24)$$

The current at time $t = t_k$ is

$$i_k = C\ dv/dt\Big|_{t=tk} \qquad (2.1.25)$$

Here again, the voltages and currents will be known only at equally spaced intervals in time. The slope of the voltage vs. time curve may again be approximated. Once such approximation is

$$dv/dt\Big|_{t=tk} \approx 1/\Delta t\ (v_k - v_{k-1}) \qquad (2.1.26)$$

Hence the capacitor current is approximated by

$$i_k = C/\Delta t\ (v_k - v_{k-1}) \qquad (2.1.27)$$

The last equation is of a conductance of value C/Δt in series with a voltage source v_{k-1}. The voltages and currents in the equation are element voltages and currents and not branch voltages and currents. The conductance remains constant as long as the value of the time step does not change.

2.1.4 EQUATIONS FOR A NETWORK

Let us consider again the network with M branches and N nodes. Let each branch contain a single passive element such as an inductor, a resistor or a capacitor. The node-voltage equations for such a network can be developed using the inductor and capacitor models developed in section 2.1.3.

Since each branch contains only one component, the element currents $\mathbf{I}_e$ and element

$$\mathbf{I}_e = \begin{bmatrix} \mathbf{I}_L \\ \mathbf{I}_R \\ \mathbf{I}_C \end{bmatrix} \qquad \mathbf{V}_e = \begin{bmatrix} \mathbf{V}_L \\ \mathbf{V}_R \\ \mathbf{V}_C \end{bmatrix} \tag{2.1.28}$$

voltages $\mathbf{V}_e$ may be separated as follows:

where $\mathbf{I}_L$, $\mathbf{V}_L$ refer to the currents and voltages across the inductors in the network; $\mathbf{I}_R$, $\mathbf{V}_R$ refer to those across the resistors in the network and $\mathbf{I}_C$, $\mathbf{V}_C$ refer to those across the capacitors in the network. This derivation assumes that the branches in the network graph are numbered such that the inductors are in a group of branches followed by the resistors and then by branches containing the capacitors.

The currents and voltages of the various elements are related as follows:

$$\mathbf{I}_R = [1/R]\,\mathbf{V}_R \tag{2.1.29}$$

$$\mathbf{I}_C = [C]\,1/\Delta t\,(\mathbf{V}_{Ci} - \mathbf{V}_{Ci-1}) \tag{2.1.30}$$

$$\mathbf{I}_{Lk} = [L]^{-1}\Delta t/2\,\mathbf{V}_{Lk} + [L]^{-1}\Delta t/2\,\mathbf{V}_{Lk-1} + \mathbf{I}_{Lk-1} \tag{2.1.31}$$

$$\mathbf{I}_{ek} = \begin{bmatrix} (\Delta t/2)[L]^{-1} & 0 & 0 \\ 0 & [1/R] & 0 \\ 0 & (1/\Delta t)[C] & 0 \end{bmatrix} \begin{bmatrix} \mathbf{V}_{LK} \\ \mathbf{V}_{RK} \\ \mathbf{V}_{CK} \end{bmatrix} + \begin{bmatrix} (\Delta t/2)\,[L]^{-1} * \mathbf{V}_{LK} + \mathbf{I}_{LK-1} \\ 0 \\ -(1/\Delta t)[C] * \mathbf{V}_{ck-1} \end{bmatrix}$$

Hence the element currents at time t_k are given by

$$= \mathbf{Y}_e . \mathbf{V}_{ek} + \mathbf{I}_{pk-1} \qquad (2.1.32)$$

The vector $\mathbf{I}_{pk-1}$ contains all the past history of the inductors and capacitors in the circuit.

The branch currents can be calculated by

$$\mathbf{I}_{bk} = \mathbf{I}_{ek} - \mathbf{I}_{gk} \qquad (2.1.33)$$

The element voltage may be expressed in terms of element current

$$\mathbf{V}_{ek} = \mathbf{Y}_e^{-1}\mathbf{I}_{ek} - \mathbf{Y}_e^{-1}\mathbf{I}_{pk-1} \qquad (2.1.34)$$

The inverse of the element admittance matrix $\mathbf{Y}_e^{-1}$ is the element impedance matrix $\mathbf{Z}_e$.

$$\mathbf{Z}_e = \begin{bmatrix} (2/\Delta t)[L] & 0 & 0 \\ 0 & [R] & 0 \\ 0 & 0 & \Delta t[1/C] \end{bmatrix} \qquad (2.1.35)$$

Hence Eq.(2.1.34) can be re-written as

$$\mathbf{V}_{ek} = \mathbf{Z}_e \, \mathbf{I}_{ek} - \mathbf{Z}_e \, \mathbf{I}_{pk-1} \qquad (2.1.36)$$

The branch voltages can be calculated using

$$\mathbf{V}_{bk} = \mathbf{V}_{ek} - \mathbf{V}_{gk} \qquad (2.1.37)$$

And finally the node voltages can be calculated using

$$\mathbf{A}\mathbf{Y}_b\mathbf{A}^T\mathbf{V}_{nk} = \mathbf{A}[\mathbf{I}_{gk} - \mathbf{A}\mathbf{Y}_b\mathbf{V}_{gk} - \mathbf{I}_{pk-1}] \qquad (2.1.38)$$

The vector $\mathbf{I}_g$ includes the inductor equivalent current and the vector $\mathbf{V}_g$ includes the capacitor equivalent voltage. Therefore, these two vectors have to be re-evaluated at each time step. To set the current sources in the inductor equivalent circuits, only the various inductor element currents have to be calculated at each time step. Similarly, the capacitor equivalent voltage sources are set from the element voltages as indicated by Eq.(2.1.32)

Eq. (2.1.38) is solved for $\mathbf{V}_{nk}$. The entire transient analysis consists of stepping k from 1 to K where $K\Delta t$ equals or exceeds t_f, the final time for which the analysis is to be performed.

At time t = 0, the vector $\mathbf{I}_{pk-1}$ does not exist. For this, the initial condition, the inductor is replaced by a very small conductance in parallel with the current source denoted by I_0, the initial current across the inductor. The capacitor is replaced by a small resistance in series with the voltage source V_0, the initial voltage across the capacitor. Then, the capacitor voltages and inductor currents are solved, and these form the basis for calculating the first $\mathbf{I}_p$, so that normal circuit equations may be formed for $t = \Delta t$. Subsequent solutions are then obtained using the method outlined above. The initial conditions in the circuit should not conflict with one another i.e. all capacitors in a loop should have the same initial conditions. Similarly, all inductors connected to a node should have the same initial conditions when that node does not have any other current path to it.

3.0 DESIGN

The transient analysis program is implemented as an analytical tool (an interpreter) to simulate the behaviour of an electrical circuit. Nodal analysis of the network (i.e. calculation of the currents and voltages at different nodes (location) in the network) is carried out and the results are output to the user. Nodal analysis is based on the algebraic manipulations of matrices.

Therefore any input given by the user has to be suitably transformed before the actual analysis can take place. In this program, one of the inputs is a file containing a description of the circuit topology and the other inputs are the analysis time, the nodes at which output voltages should be calculated and the file to which CLP(R) should write the output. The transformations take place in the parser and the analyser, the voltages and currents are calculated in the interpreter, and the results are displayed as output.

The interpreter is the process which does the actual transient analysis of the given circuit. It requires as inputs, the circuit topology the time scale for which the simulation should run and the desired output from the program. The circuit topology is described using a Circuit Definition Language defined by us for this purpose. A parser checks the inputs and reports any syntactic errors back to the user and the output from the parser is in the form of a network graph.

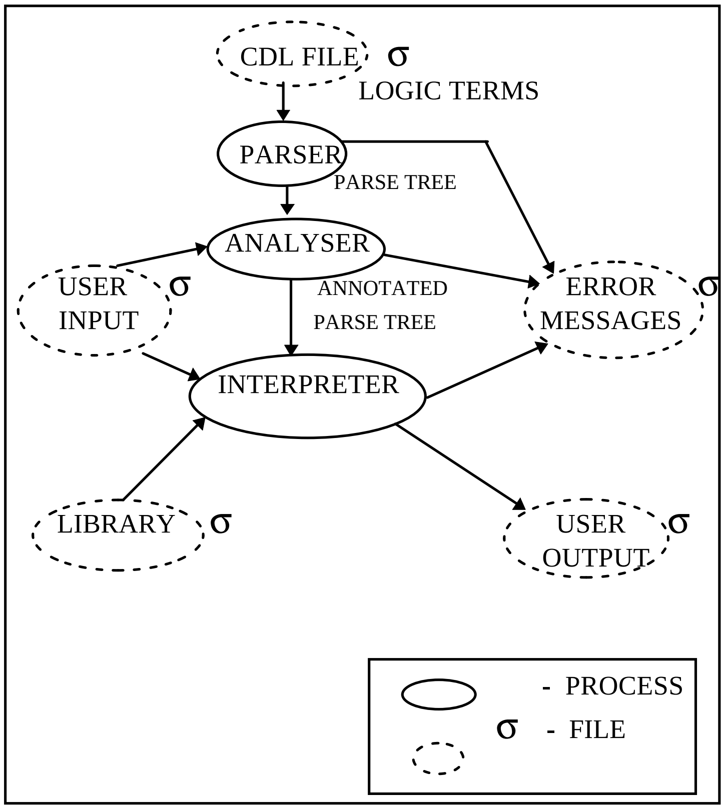


Figure 3-1 Transient analysis program

The network graph is then passed on to the analyser. The analyser performs a semantic analysis of the network, reporting any errors back to the user. It also converts the network graph into matrices and passes them on to the interpreter. The interpreter has access to CLP(R) and to a purpose built library implementing various matrix operations as an abstract data type. The interpreter calculates and outputs the dynamic behaviour of the circuit (i.e. the change of node voltages over a period of time as specified by the user).

Apart from the circuit description, the user has to input other information such as the time step, the start time and end time of analysis, the output nodes which are of interest as well as a file name to which CLP(R) can write the output.

3.1 CIRCUIT DEFINITION LANGUAGE

As stated earlier, the transient analysis program requires the topology of the circuit as one of its inputs. When such an analysis is done manually, the circuit diagram is represented by a set of symbols drawn on a sheet of paper. The person can identify the different elements of the network by the symbols. A computer on the other hand, cannot recognise these symbols and so the program has to be given this information as part of the structure of the input. The circuit details must be presented in a specific manner and in this program, this is done using a language called the circuit definition language which will be defined in this section.

The language has to represent in a unique way all the circuit elements and has to distinguish between the active elements such as the current source and the voltage source (the drivers), and the passive elements such as the inductors, the capacitors and the resistors.

The connection of each element in the network is indicated by the two nodes to which it is connected. A node is represented by the letter n followed by a digit. Numbering of the nodes starts from 0. In electrical engineering, voltages are measured as the potential difference between two any two nodes. When one of such nodes is at zero potential it is called as the ground and is represented as n(0). All voltages are measured relative to the ground. See [SHAN95] for the definition of the language.

As an example, the linear circuit in Fig 3-2 is defined in the circuit description language as:

```
circuit(  [voltage_source(v1,30,5,n(0),n(1))],
              [inductor(l1,10,1,n(2),n(1)),
               resistor(r1,10,n(0),n(2)),
               resistor(r2,10,n(2),n(0))] ).
```

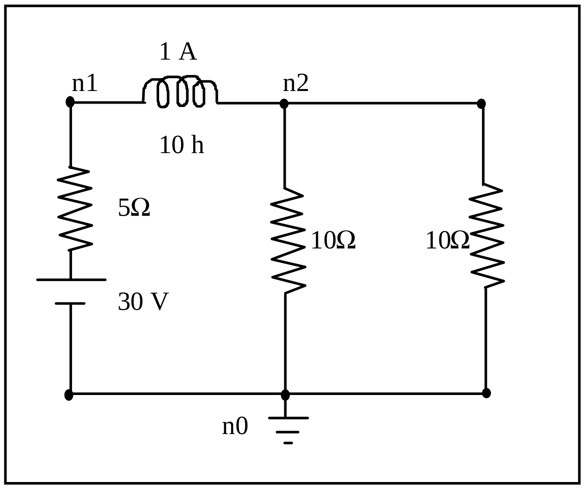


Figure 3-2

Inclusion of the nodes in our circuit definition language makes it context sensitive. This is so because a syntactically correct circuit could still describe a semantically incorrect circuit.

One way of overcoming this difficulty is to perform a semantic analysis on the input file. Another way would be to make the language context free by implicitly declaring a node to be present at the junction of two elements. The circuit is described by specifying the composition of the elements as either series or parallel [BOUT88]. However, describing a circuit as a composition becomes quite cumbersome for the larger circuits, and description of non-planar circuits becomes very difficult (a non-planar circuit is a circuit that is three dimensional, whereas planar circuits are two dimensional). Both methods of circuit description require that the user of these languages should have adequate knowledge of circuit connections.

3.2 PARSER

The parser reads the input file and converts it into a network graph. The network graph we use is represented by two sets: the set of independent nodes in the circuit and set of branches or arcs in the circuit. Each branch in the circuit is given a number by the parser. This number is used later on by the analyser to form the network matrices and then by the interpreter to manipulate the matrices. The branches containing inductors in the circuit are numbered first, followed by the resistors, the capacitors, the voltage sources and finally the current sources. The output of the parser is a positionally ordered set of arcs and an ordered set of nodes. The parse is not unique for the arcs as the ordering in the set of arcs depends on the ordering in the input file. However, the time taken to parse the file does not depend on the ordering of the input file.

3.3 ANALYSER

The analyser transforms the inputs i.e. the set of arcs and the set of nodes into matrices. In addition it also does a semantic analysis of the network, reporting errors back to the user. The matrices form the foundation for the nodal analysis of the network and so the analyser plays an important part in the program.

The outputs of the analyser are the incidence matrix (A), the admittance matrices (Yb and Yb0), the independent current matrix (Ig) and the independent voltage matrix (Vg). The product of the incidence matrix and the branch current matrix represents the network graph at the end of the analysis process. In the semantic analysis, the analyser checks to see if the circuit is connected properly and detects open circuits. The analysis is done by checking that the number of elements in the arc list is greater than the number of elements in the node list.

If the node list is the larger list, the incidence matrix is used to locate which of the branches are open. The results of the semantic analysis are output to the user.

3.4 SEMANTIC ANALYSIS OF THE NETWORK

The circuit definition file is context sensitive due to the inclusion of the nodes in the language. Therefore, the file has to be checked to ensure that it is correct semantically. In semantic analysis, the network is checked to ensure that it is connected properly i.e. the branches are all connected to one another and there are no open circuits. The incidence matrix and the network graph are used in the check. In a properly connected circuit, the number of nodes is always less than or equal to the number of elements. Hence, the first check is made on the number of arcs and the number of nodes in the network graph. If the nodes are greater in number than the arcs, it implies that the circuit is not properly connected. If the network is open, the incidence matrix is used to determine which of the branches are open. This can be found out from the fact that every node has at least two connections to it. In other words, every row in the incidence matrix should have more than one non-zero column in it. If a row has just one entry that is non-zero, then that node is the open node.

3.5 INTERPRETER

The interpreter is the process that performs the actual transient analysis. The matrices from the analyser and the time input by the user are the inputs to this process. The interpreter steps through the analysis, one time step at a time from the initial time T_{ini} to the final time T_{fin} calculating the currents and voltages required by the next iteration. The time step is represented by δt. The past history of the capacitors and the inductors are represented by the matrix Ip. At each iteration, the output voltages and currents are accumulated and at the end of the analysis, are sent to the output process. The interpreter makes use of the operations defined in the matrix library to calculate the voltages and currents. The output process is used to output the results of the analysis. The node voltages of interest to the user are by default printed on to the screen for each time step. A copy of the output is also written to an output file at the request of the user.

4.0 IMPLEMENTATION

In the transient analysis program, a number of iterations have to be carried out before the results can be obtained. The voltages and currents of the circuit have to be calculated for each time step, and the results used in the next time step. For large circuits, this could mean the calculation of hundreds of equations. In our implementation we use constraints to simulate the behaviour of the circuit as well as it’s components. For example, in nodal analysis of circuits, Kirchoff’s current and voltage laws must hold for all the nodes [SESH63].

The network graph is represented by two sets - the set of arcs and the set of nodes in the parser. In the CLP(R) implementation, the sets can be represented by lists. Positionally ordered sets are represented as follows:

Each element in the set of arcs is given a unique branch number. Numbering always starts from 1 and the number given to each branch depends on the ordering of the elements in the input file. All inductor branches are numbered first, followed by all the resistors, then capacitors, then voltage sources and finally the current sources.

The time taken to analyse the circuits is independent of the ordering of the elements in the circuit description file and the output of the parser depends on the ordering of the individual elements in the circuit description file.

4.1 MAIN PROGRAM

The main program corresponds to the process flow diagram shown in Fig 3-1 in section 3. The user is asked for the name of the circuit description file, the start time, time step and end time of analysis, the name of the file into which output should be written (default is the screen), and the nodes for which the output voltages should be calculated.

```
input(CircuitDescriptionList, Time, OutNode, OpFile),
parse(CircuitDescriptionList,ArcList,NodeList, ElemList),
analyse(NodeList,ArcList,IncidenMat,AdMat,InitialAdMat,
        CurrMat,VoltMat),
transient(IncidenMat,AdMat,InitialAdMat,CurrMat,VoltMat,Time,
ElemList,OutNode,Output),
output(OutNode,OpFile,Output).
```

Each of these predicates represent a process and are discussed in the following sub-sections.

4.2 PARSER

The parser converts the circuit description into a network graph which is represented by two lists - the ArcList and the NodeList.

```
parse(CircuitDescriptionList, ArcList, NodeList, ElemList):-
          [DriverList|ElementList]=[CircuitDescriptionList],
          get_arclist(DriverList, ElementList, ArcList),
          get_nodelist(DriverList,ElementList,NodeList),
          separate(ArcList, ElemList).
```

The predicate `get_arclist(DriverList, ElementList, ArcList)` is used to form the ArcList. The elements are separated into five lists, each list containing elements of the same type and the branches are numbered starting from 1. All the inductor branches in the circuit are numbered first, followed by the resistor branches, the capacitor branches, the voltage source branches and finally the current source branches.

The predicate `form_arc(Element,Number,NewElement)` converts the element in the circuit description file format to an arc in the network graph.

4.3 ANALYSER

The analyser is divided into two parts. One part performs semantic analysis of the circuit and the other part converts the network graph into matrices.

The predicate `semantics(NodeList,ArcList,IncidenceMatrix,Error)`

checks the circuit for open connections. This is done by calculating the lengths of the ArcList and the NodeList. If the length of the NodeList is greater, it implies that there is an open circuit. The incidence matrix is then checked using the predicate `connected(Incidence_matrix, Error)` to locate which of the branches are open.

```
semantics(Nl,Al,InMat,Error,Open):-
          length1(Nl,Num1),
          length1(Al,Num2),
          Num1 < Num2,
          Error = 1,
          connected(InMat,Open).
```

The analyser converts the network graph into matrices. The product of the incidence matrix and the branch current matrix now represent the circuit.

4.4 INTERPRETER

Transient Analysis of a circuit consists of stepping the voltage and current calculations from $t=t_{ini}$ to t_{fin} in steps of Δt, where t_{ini} is the initial time, t_{fin} is the final time and Δt is the time step.

The conditions that exist in the circuit at time t_{ini} are calculated using the predicate `initial(Tini,Y0,A,L,Vg,Ig,Op,VgNew,IgNew,Output)` where `Y0` and `L` are functors of arity 3. The arguments of `Y0` are the initial condition branch admittance matrix (Y_{B0}), the inverse of the node admittance matrix (Y_n^{-1})and the element admittance matrix (Y_e). The arguments of `L` are IndList, CapList, and RestList. IndList is a list containing the branch numbers of all inductors in the circuit. CapList is a list containing the branch numbers of all the capacitors in the circuit and RestList is a list containing the branch numbers of all other elements in the circuit. These lists act as place markers while calculating `Ip, VgNew and IgNew`. `VgNew` is the matrix `Vg` with the latest capacitor equivalent sources. `IgNew` is the matrix `Ig` with the latest inductor equivalent sources. `Ip` is a vector containing the past history of all the inductors and capacitors in the circuit. This is calculated using the predicate `pastH(Ve,Ie,Ye,L,Ip)`, where `Ve` is the element voltage matrix, `Ie` is the element current matrix, `Ye` is the element admittance matrix, `L` is the list of place markers.

```
pastH(Ve,Ie,Ye,L,Ip):-
           extract3(L,IndList,CapList,RestList),
           length1(Ye,Num),
           putR(Num,[],Newl),
           ipL(IndList,Ve,Ie,Ye,Newl,Tmp1),
           ipC(CapList,Ve,Ye,Tmp1,Tmp2),
           ipR(RestList,Tmp2,Ip).
```

Once the initial conditions are calculated the remaining analysis is done using the predicate `transient(Tnext,T,Y,A,L,P,VgNew,IgNew,OutputNew)`, where Tnext is the time step Δt, and `T`/3, `Y`/3, `L`/3 and `P`/3 are functors, `A` is the incidence matrix and `Outputnew` is the list of lists containing the output node voltage values, and the time. At each iteration, the output is stored in Outputnew using the predicate `storeOP(User_output,Node_voltages,Time,Input, Output).`

4.5 MATRIX LIBRARY

To implement the abstract data type of matrices, operations such as multiplication, addition, subtraction, transpose, determinant and inverse have been defined.

The definition of matrix inverse is: if A is the matrix to be inverted, I is a unit matrix, and AxB = I , then B is the inverse of A. This can be implemented very elegantly and economically in CLP(R) by making use of the multi-directionality of logic programs. Since matrix multiplication has already been defined, it can be used to find the inverse and a separate predicate need not be defined. This is also very efficient.

5.0 TESTING

The transient analysis program was tested using the examples given in Staudhammer [STAU75]. For the analysis, the inductors and capacitors in the circuits are replaced by their equivalent models as discussed in Section 2. The calculations start with the determination of the initial conditions existing in the network, prior to the application of the time-iterations. The initial conditions should be consistent to gain a meaningful result: for example, if we replace a 0.5 farad capacitor by two 1 farad capacitors, we would expect the initial voltage across the two separate capacitors to be equal.

As an example, let us consider the RL circuit of Fig. 3-2. The inductor connected between node 1 and node 2 has an initial current of 1 ampere flowing through it. The solution to this network is given by the following equations in [STAU75]:

$$V_1(t) = 15 + 10e^{-T/\tau}$$

$$V_2(t) = 15 - 10e^{-T/\tau}$$

with $\tau = L/R_T = 1$ since, R_T, the total resistance that exists in the current loop of the network is equal to 10 ohms.

The analysis was carried out with an initial time of 0, a time step of 0.1, and a final time of 5 seconds. Table 1 shows a comparison between the expected results (V1 and V2) and the actual results obtained (n(1) and n(2) represent the voltages at node 1 and node 2 respectively) for selected values, presented at time intervals of 0.5 seconds for reasons of clarity.

Time (sec)	V1 (volts)	n(1) (volts)	V2 (volts)	n(2) (volts)
0.0	25.00	25.00	5.00	5.00
0.5	21.06	21.06	8.94	8.94
1.0	18.68	18.68	11.32	11.32
1.5	17.23	17.23	12.77	12.77
2.0	16.35	16.35	13.65	13.65
2.5	15.82	15.82	14.18	14.18
3.0	15.50	15.50	14.50	14.50
3.5	15.30	15.30	14.70	14.70
4.0	15.18	15.18	14.82	14.82
4.5	15.11	15.11	14.90	14.89
5.0	15.07	15.07	14.93	14.93

Table 1- Comparison of expected results with actual results

Comparison of the expected results with the actual results shows that the program is 99.99% accurate at 2 decimal places; the only discrepancy occurring when time is equal to 4.5 seconds. See [SHAN95] for detailed results of this and other examples, where the minimum accuracy is 97%.

The test results show that CLP(R) can be used to perform transient analysis of circuits with accuracy and ease. The response time of CLP(R) is very fast. For example, in the test, CLP(R) had to solve 700 equations and this was done within 2 seconds.

5.1 BACKWARD EVALUATION

One of the main benefits of implementing the transient analysis program in CLP(R) is that in addition to the analysis, we can use the same program for design. For example for the circuit in Fig 3-2, we can query the program as to what should be the value of the voltage source and the initial current across the inductor so that the node voltages at n1 and n2 are 25 and 5 respectively.

```
circuit([voltage_source(v1,V,5,n(0),n(1))],
        [inductor(l1,10,I,n(2),n(1)),
        resistor(r1,10,n(0),n(2)),
        resistor(r2,10,n(2),n(0))]).
```

The program comes out with the answer `V = 30` and `I = 1`. To accomplish the same result in an imperative language either a different program has to be written or the simulation has to be run several times with different values of V and I and the answer to the question found by trial and error.

5.2 COMPARATIVE TEST

The time taken by the transient analysis program to evaluate a 13 element circuit was compared against the time taken by PSPICE to evaluate the same circuit. A larger example was used in order to get better precision in comparison. See [SHAN95] for full details.
PSPICE is designed to be the micro computer version of SPICE. SPICE is the acronym for Simulation Program with Integrated Circuit Emphasis. As the name suggests it was originally developed to perform analysis of AC and DC circuit, frequency response and, transient analysis of integrated circuit chips by the University of Berkeley, California.
The input used by PSPICE is very similar to the circuit description that we have defined even though the circuit description was defined before looking at PSPICE. Each element in the circuit has to be identified by its name, the two nodes to which it is connected as well as its impedance. For transient analysis, the analysis time, the initial current across inductors, and the initial voltage across capacitors have to be specified [NILS93].

The circuit that was used for the test is shown here in the circuit description format:

```
circuit([voltage_source(v1,30,5,n(0),n(1))],
        [inductor(l1,10,1,n(2),n(1)),
        resistor(r1,10,n(0),n(2)),
        resistor(r2,10,n(2),n(3)),
        resistor(r3,10,n(3),n(4)),
        resistor(r4,10,n(4),n(5)),
        resistor(r5,10,n(5),n(6)),
        resistor(r6,10,n(6),n(7)),
        capacitor(c1,0.5,10,n(3),n(0)),
        capacitor(c2,0.5,0,n(4),n(0)),
        capacitor(c3,0.5,0,n(5),n(0)),
        capacitor(c4,0.5,0,n(6),n(0)),
        capacitor(c5,0.5,0,n(7),n(0))] ).

Initial time = 0 , tstep = 0.1, final time = 5.
```

PSPICE took about 5 seconds to analyse the circuit while our program took about 16 seconds. The tests were run on a 486 DX machine and the time taken by the two programs is the total time and not CPU time alone. The reasons for the slower response time could be as follows: firstly our program being a prototype, is not optimised for efficiency and speed while PSPICE is an optimised commercial package. Secondly, our program is not compiled, but is interpreted by the CLP(R) interpreter while PSPICE is an compiled executable. There could also be a penalty for calculating inverses at each time step. Since our matrix library is a standalone module, it can be replaced by other faster more efficient library routines.

6.0 RELATED WORK

CLP(R) has been used to in the design and analysis of circuits by [HEIN92] and also used by [FATT94] to model dynamic systems in general.

In [HEIN92], CLP(R) was used for the analysis of circuits such as RLC circuits, transistor circuit design etc. Steady state, or static analysis, is done on RLC circuits containing voltage sources, current sources, resistors, inductors and capacitors.

A general purpose approach has also been adopted by [FATT94] to model dynamic systems using bond graphs and CLP(R). The structure of the dynamic system is described using a bond graph language. The circuit is analysed using state space analysis and the resulting differential equations are solved using a relational differential equations solver implemented in CLP(R).

In [HONG94] a solver for differential equations described, included an example program to solve simple relationships over electrical circuits. We believe that this system could provide a good implementation platform for extensions to our work.

7.0 FURTHER WORK

We have shown for the first time that CLP(R) and its multi-directional capabilities can be used in a particular application domain which is generally considered to be more suitable for imperative languages. Our work concentrated on the DC analysis of linear RLC networks and as it stands can be used as a teaching aid for undergraduates. To make it a commercially viable package the following have to be added:

- Providing an integrated windows environment for the program
- Addition of further circuit elements to the basic set considered here.
- Adding constraints to the circuit definition language (the circuit description) used in the program.

Adding constraints to the input and output and implementing the program in an integrated graphics environment will be of great use to the designer since the cost of circuit components is related to the tolerances (i.e. constraints) to which they are manufactured.

8.0 SUMMARY AND CONCLUSIONS

This paper investigates the implementation of transient analysis of linear circuits in constraint logic programming. Our work provides a declarative alternative to the procedural approach usually adopted to model electrical circuits.

The user provides the analysis start time, end time and time step, and the nodes for which the voltages and currents should be calculated. The parser converts the circuit description into a network graph represented by two sets - the set of arcs and the set of nodes. The analyser checks the network graph for open connections and converts the network graph into matrices required by the interpreter. The interpreter steps through the analysis calculating the required voltages and currents and then informs the user about the results of the analysis. The tests conducted show that the accuracy of the program was 99% at two decimal places.

We have also shown that implementing the analysis package in CLP(R) enables us to use the same package for *designing* a circuit as well. Using the desired output as the constraint, the necessary inputs to the circuit were calculated. Multi-directionality of CLP(R) was also used to define the inverse of a matrix in terms of multiplication alone, thereby avoiding the task of defining a new predicate to calculate the inverse.

Future work should include additional elements such as transformers and dependent sources, thereby extending the range of circuits that can be analysed. Another important extension would be the inclusion of constraints as part of the circuit description itself.

The process of implementing the program in CLP(R) has enabled us to reason about the circuit itself as well as the analysis, thereby increasing our understanding of the problem. The declarative language used has proved to be very useful in the implementation. Backward evaluation makes the analysis tool a design tool as well. Given a known output, the program was able to deduce the values of the input excitation. The same technique was used to find the inverse of a matrix. Coding of the entire program was followed easily from the logical design. The code is readable and hence maintenance is easy. Given that our system was a interpreted prototype, its performance compares very well with a commercial package which has been implemented in a compiled imperative language.

Finally, an advantage of our approach is that we are able to generate instances of circuits from general schemas and descriptions of their required behaviour.

REFERENCES

[BOUT88] Boute Raymond T., ‘Systems Semantics : Principles, Applications, and Implementation’ in *ACM Transactions on Programming Languages and Systems*, Vol. 10, No. 1, January 1988, pp. 118-155.

[FATT94] El Fattah Y. and Holzbaur C., ‘Constraint Logic Programming for Modelling and Simulation of Dynamic Systems’, in ‘*Proceedings ILPS ‘94 workshop on Constraint Languages/Systems & their use in Problem Modelling*’, Vol 1, Nov 1994.

[HONG94] Hoon Hong, ‘RISC-CLP(CF) Constraint Logic Programming over Complex Functions’, LPAR94 (?)

[JAFF87] Jaffar J. and Lassez J.L., ‘Constraint Logic Programming’, in *Proceedings of the Fourteenth ACM Principles of Programming Languages Conference,* Munich, January 1987.

[JAFF92] Jaffar J., Michaylov S., Stuckey P., and Yap R., ‘The CLP(R) Language and System’ in *ACM Transactions on Programming Languages and Systems,* Vol 14 No 3, July 1992, pp. 339-395.

[HEIN92] Heintze N., Michaylov S., Stuckey P., ‘CLP(R) and Some Electrical Engineering Problems’ *Journal of Automated Reasoning,* Vol 9, 1992, pp 231-260.

[NERI70] Nering E.D., *Linear Algebra and Matrix Theory*, John Wiley and Sons Inc., New York, 1970.

[NILS93] Nilsson J.W., and Riedel S.A., *Introduction to PSPICE*, Addison-Wesley Publishing Company Inc., 1993.

[STAU75] Staudhammer John, *Circuit Analysis by Digital Computer* Prentice-Hall Inc., 1975.

[SESH63] Seshu S., and Balabanian N., *Linear Network Analysis*, John Wiley & Sons, Inc., New York, 1963.

[SHAN95] Shankar A., Gilbert D., Jampel M., *Transient Analysis of Linear Circuits using Constraint Logic Programming*, Technical Report TCU/CS/95/17, City University Computer Science Department, London, 1995