REASONING ABOUT LINEAR CIRCUITS IN SINUSOIDAL STEADY STATE

by

JUAN JOSE FLORES ROMERO

A DISSERTATION

Presented to the Department of Computer
and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

August 1997

"Reasoning About Linear Circuits in Sinusoidal Steady State," a dissertation prepared by Juan Jose Flores Romero in partial fulfillment of the requirements for the Doctor of Philosophy degree in the Department of Computer and Information Science. This dissertation has been approved and accepted by:

[illegible]

Dr. Art Farley, Chair of the Examining Committee

20 August 1997

Date

Commitee in charge: Dr. Art Farley, Chair
Dr. Andrzej Proskurowski
Dr. David Etherington
Dr. Davison Soper

Accepted by:

[illegible]

Vice Provost and Dean of the Graduate School

An Abstract of the Dissertation of

Juan Jose Flores Romero for the degree of Doctor of Philosophy

in the Department of Computer and Information Science

to be taken August 1997

Title: REASONING ABOUT LINEAR CIRCUITS IN SINUSOIDAL STEADY STATE

Approved: [illegible]

Dr. Art Farley

Most of the work on behavior prediction on the field of Qualitative Reasoning has focused on transient behavior and responses to perturbations; very little has been done about systems in steady state. A large class of systems, especially in the area of power systems, are designed for sinusoidal steady-state operation. Thus, an understanding of the steady state of electrical circuits is very important.

This dissertation presents a framework for reasoning about linear electrical circuits in sinusoidal steady state. The reasoning process relies on a constraint-based model of the circuit, derived from electro-magnetic theory and generated automatically from the structure of the circuit. In a linear circuit operating in steady state, all quantities are sinusoidals of the same frequency as the source. Since any sinusoidal can be expressed as the real part of a complex exponential, we use the complex form, which simplifies computations; this complex form, characterized by magnitude and angle, is called a phasor. In order to capture magnitude and phase angle information in the model, all constraints operate on phasor variables.

Constraint propagation (CP) is the main inference mechanism. The CP module reasons with as much information and precision as the user provides, ranging from qualitative to quantitative. Intervals provide a general representation mechanism.

The framework presented in this dissertation has been implemented in a program called Qualitative Phasor Analysis (QPA), which performs the following reasoning tasks: analysis, parameter design, diagnosis, control design, and structure simplification. Circuits with multiple sources are solved using the superposition principle.

Power systems can be modeled as linear circuits and normally operate in steady state. A power system problem is translated to a circuit problem and solved by QPA; the results are then translated back to the original power system.

By extending the circuit ontology to include phasor information, this dissertation extends the range of problems that can be solved by qualitative reasoning.

CURRICULUM VITA

NAME OF THE AUTHOR: Juan Jose Flores Romero

PLACE OF BIRTH: Zamora, Michoacan, Mexico

DATE OF BIRTH: June 2nd, 1961

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon
Centro de Investigacion y Estudios Avanzados, IPN
Univesidad Michoacana de San Nicolas de Hidalgo

DEGREES AWARDED:

Doctor of Philosophy in Computer and Information Science, 1997,
University of Oregon
Master of Science in Computer Science, 1986,
CIEA, IPN, Mexico,
B.Sc. in Electrical Engineering, 1983,
UMSNH, Morelia, Mexico.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Professor Art Farley, for his guidance and encouragement to achieve my goals. He spent so many hours of discussion about different issues in my research, proof-read every paper I submitted and this dissertation.

I would also like to thank Professors Andrzej Proskurowski, David Etherington, and Davison Soper for serving on my Ph.D. committee, reading the drafts, attending to meetings, and giving valuable comments. James Crawford had to leave the University of Oregon and therefore my committee, but he still kept reviewing and commenting my work. Elizabeth Bradley, from the University of Colorado, and Edmundo Barrera, from the University of Michoacan, took the time to read the drafts and held various discussions about the applicability of this work in Electrical Engineering. I appreciate the cheers and help from several Mexican graduate students in other Universities and my fellow grad students at the University of Oregon.

I want to thank my wife, Gina, for her love, understanding, and support. I thank my children: Gina, Mayra, and Pablo, for understanding so many times that I had to go back to work, instead of staying with them. Thanks to my father for always encouraging me to keep studying, and to my mother (que en paz descanse) for teaching me, by example, to live in God's way. Thanks to my brothers and sisters, who always helped me with all sorts of paper work back home. Thanks to my aunts and my wife's family, who kept alive the desire to go back home.

This research was supported by a Conacyt-Fulbright scholarship, the Universidad Michoacana de San Nicolas de Hidalgo, and the University of Oregon

DEDICATION

To my wife: Gina

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

One of the main objectives of qualitative reasoning is to derive the behavior of a system from a description of its components and their interrelationships [12, 17, 28, 52]. Prediction of behavior has been achieved by traditional physics at different levels and in different areas. In particular, in the area of circuit analysis there are a number of numerical methods to analyze circuits of different kinds and under different conditions [27, 49]. Those methods take as input a circuit topology and exact values for the parameters, perform some computation (mainly based on linear algebra or iterative methods to solve non-linear or differential equations), and return exact values for the variables representing the unknown quantities. In this process, causality and explanation are discarded; the only goals are precision and efficiency.

This dissertation presents a framework for performing qualitative analysis of linear circuits in sinusoidal steady state. We call this approach Qualitative Phasor Analysis (QPA). QPA is based on three main ideas regarding modeling and qualitative physics. First, it develops a constraint-based model of the circuit, where the constraints are derived from general knowledge of circuit theory and the circuit's topology. A constraint-based model can be generated automatically, this feature makes the system able to analyze any series/parallel decomposable circuit. Another feature provided by constraint-based reasoning is that the order of computation is not fixed, i.e., any variable can be used as input or output. This property allows us to perform parameter design, based on circuit analysis.

Second, it is able to propagate constraints of different kinds (e.g., algebraic, value, order of magnitude, etc.). Constraints of different types are kept in different sets, and the rules to propagate those constraints are different. Value propagation is extended to deal with interval values. Mixed constraint propagation allows us to derive order of magnitude relations from values and to refine values from order of magnitude relations.

Third, phasor analysis can be done with as much information about circuit parameters and quantities as provided by the user, ranging from qualitative (sign) to numerical (real) values. This characteristic is implemented by uniformly representing all values as intervals. Signs can be represented as intervals with open infinite extremes, and reals can be represented as point intervals. Value propagation can also handle phasors with interval magnitudes and phase angles.

Based on the results of circuit analysis, several reasoning tasks can be performed. The reasoning tasks we have in mind are qualitative/quantitative circuit analysis, parameter design based on first-order reasoning, circuit analysis, circuit simplification by order of magnitude reasoning, the derivation of all possible qualitatively different phasor diagrams (corresponding to qualitatively different states of the circuit), fault diagnosis, and control design.

This chapter explains the basic problems this dissertation addresses, states the thesis, and identifies some of the important research issues. It then discusses implementation and evaluation.

Background

The foundational contributions to the field of qualitative reasoning are those by DeKleer [12], Kuipers [28], and Forbus [17]. DeKleer proposed an approach based upon confluences, which captures a sense of causality and reasons about change. Forbus developed qualitative process theory, QPT, a mechanism that allows us to reason about physical objects and their interaction modeled as processes. Kuipers developed QSIM, a system that predicts all possible transient behaviors of physical systems, for systems that can be described by sets of ordinary differential equations.

Several systems have been built to reason about and derive the qualitative behavior of electrical circuits. Most of them focus on either digital circuits or DC analog circuits (see for example, DeKleer [10], Hamscher [22], Williams [52]). None has addressed the analysis of linear circuits in sinusoidal steady state. QSIM can simulate the behavior of linear circuits, but since it is based on differential equations, its scope is limited to transient state analysis. Also, QSIM's formalism is not able to represent a sinusoidal source in terms of the given types of constraints. The response description given by QSIM is at a microscopic level with respect to time, describing the possibilities at each distinguished time point. It is a well known fact that all variables in a circuit in steady state will be steady sinusoidals; there is no point in trying to find out if a peak (defined by a landmark) will be greater, equal or less than the next one. That microscopic view prevents us from getting the big picture of what is happening in the circuit and generates unnecessary ambiguity.

DeKleer's confluences allows us to reason about change, but only in terms of magnitudes of *scalar* quantities. Since the main tool used to solve steady state problems is *phasors* (i.e., a particular kind of vector), we need a way to represent angular information and the interaction between the magnitudes of different quantities and their phase angles.

Trying to describe the behavior of an electrical circuit in terms of processes is awkward. Similar to Kuiper's approach, the kind of description that QPT yields would be at the microscopic level. This kind of representation would involve charges, and how the process of moving charges (i.e., electrical current) would result from the application of an electrical field. We need something at a higher level of abstraction, where the existence of a stable oscillation of alternating currents is already known and not the goal to be established. Thus, QPT is not suited to solving the problems addressed in this dissertation.

Few researchers have worked on qualitative analysis of power systems. Struss [42, 43] developed a system to diagnose faults in power transmission networks. He follows a relational approach to the modeling of power system components; consistency-based diagnosis is used to find faults in the system, based on the reading of "distance protection relays". The analysis he performs is not based on circuit theory; it focuses on what relays have tripped and if the observations are consistent with the models of those relays.

QPA is able to simplify a circuit, based on order of magnitude relations. If QPA determines (based on order of magnitude relations given by the user) that a current (voltage) is negligible, it can discard that part of the circuit, replacing it by an open (short) circuit. We call this feature *structural exaggeration*. A similar kind of transformation is presented by Liu's ARC [32], which eliminates parts of a circuit if the operating region of an element indicates it behaves like an open circuit. The system is then recasted according to its new topological configuration. Also related to model changes, Struss presents a diagnosis system that works with models at different levels of abstraction. The simplifications presented in his work deal with the internal model of each device, refining it to yield more accurate results when necessary. The overall structural description of the system does not change with the use of different models. QPA's structural exaggeration, by contrast, can simplify the overall structure of the circuit, supported on the circuit's operating conditions.

Bibliographic surveys are given at the end of each chapter to contrast the contributions presented in this work with previous results in related fields.

Qualitative Phasor Analysis

The electrical engineering community has been very successful in predicting behavior of linear circuits in steady state. The main tool they use in circuit analysis is the phasor. Phasors [27, chapter 5] are a mathematical transformation that maps sinusoids from the time domain to the frequency domain, allowing us to replace complicated simultaneous differential equations by simultaneous algebraic equations in the complex domain. In addition to their power to solve linear circuits, phasors can be expressed in an intuitive graphical form; the so called *phasor diagrams.* These diagrams allow electrical engineers to have a better understanding of what happens inside a circuit and can be used to produce causal explanations of physical phenomena. This characteristic of phasors is the one that we want to capture in a suitable representation, allowing us to reason about linear circuits in the same way as in explanations found in books.

A mathematical model of a circuit includes a set of algebraic relations that constrain its behavior. For instance, we know that current and voltage are in phase in a resistor or that the currents of two parallel branches add to the total current of the combination. We can capture this information and represent it as a set of qualitative constraints, which will enable us to reason about a circuit's behavior.

To determine the set of algebraic constraints, we will represent the circuit as a recursive structure of series/parallel clusters [32]. We can recursively traverse that clustering structure, generating constraints for each cluster and single element we encounter. The resultant set of constraints can be partitioned into constraints of several types: Algebraic (e.g., $V_{R2} = Z_{R2} * I_{R2}$), Ordering (e.g., $I_{R1} > I_C$), Order of Magnitude (e.g., $I_{R1} \gg I_C$), Phase Angle (e.g., V_{R2} InPhase I_{R2}), and Confluences[1] (e.g., $\partial V_R - \partial Z_R - \partial I_R = 0$). These constraints are derived from basic electro-magnetic and circuit theories. Confluences, which represent the dynamic properties of the circuit, can be derived from algebraic constraints.

Let us consider the circuit of Figure 1, which shows a circuit and its topological configuration in terms of series/parallel clusters. Figure 1 also shows examples of the type of constraints that can be generated for each component and cluster of the circuit.

The set of all constraints that can be derived from the circuit topology, based solely on electrical circuit theory, are what we call the Basic Set of Constraints (BSOC). Once the BSOC has been generated, propagation is used to maintain the transitive closure of the constraints and their implications. For instance, from the constraints $I_{S1} = I_{R1}$ and $I_{S1} = I_L$ we can derive that $I_{R1} = I_L$. If the user has any further information about features of the circuit, he or she can express them in the form of additional constraints. For instance, the user can say that $Z_{R2} < Z_C$; propagation will return to the user the implied constraints, informing him or her that $I_C < I_{R2}$.

[1]A confluence has the usual meaning and use [12]. For instance, for Ohm's law in a resistor $V_R = Z_R I_R$, we have the qualitative counterpart $\partial V_R - \partial Z_R - \partial I_R = 0$. This confluence indicates (among other things) that if Z_R decreases and V_R does not change, I_R increases.

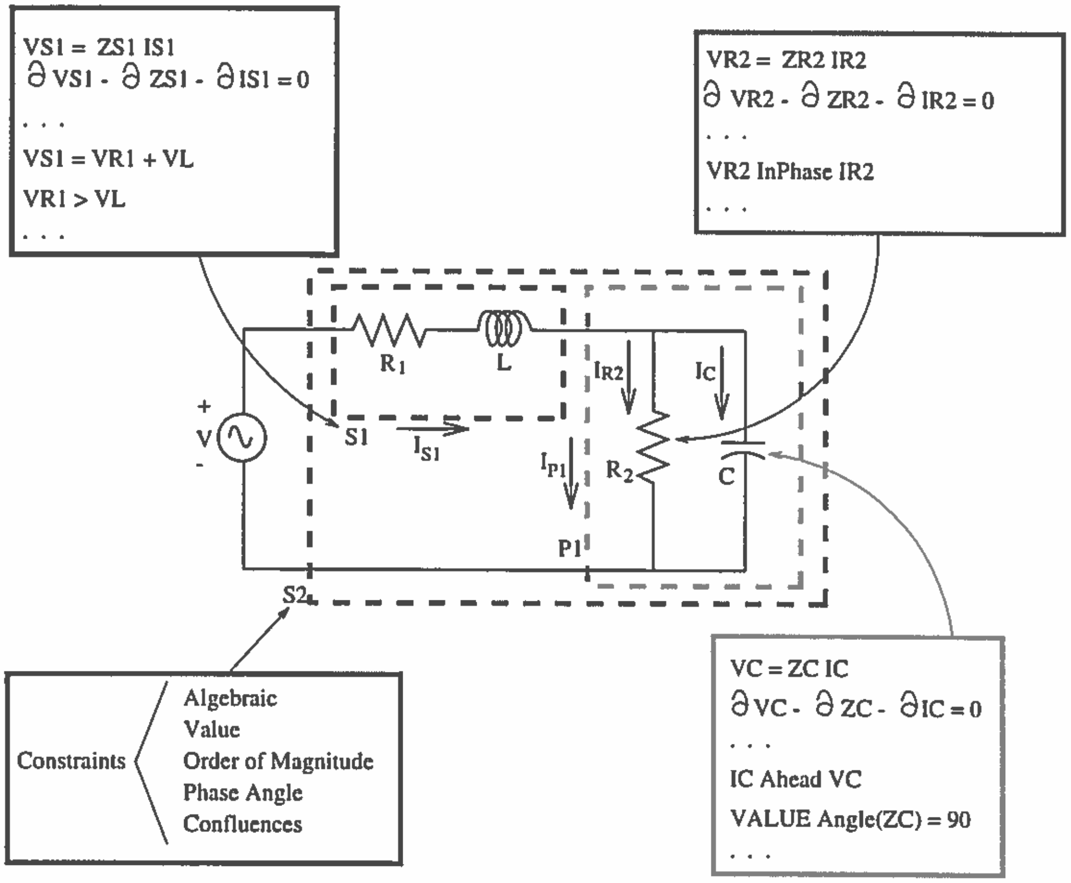


FIGURE 1. An Electrical Circuit, its Configuration, and Constraints

The values that variables can take on range from qualitative signs to reals. In between those extremes, one can specify values by intervals, with as much precision as needed. Of course, the more precise the provided information is, the more precise the results will be. While analyzing a circuit, one can specify values for some of the variables, and propagation will compute the resulting values for the rest of them. For instance, for the circuit of Figure 1, one can specify $W = 60$, $V_S = 100$, $C = [0.0024, 0.0026]$, $R_2 = [14.9, 25.1]$, etc. QPA will solve for the rest, e.g., $V_{S2} = 100$, $I_C = [1.4857, 1.9539]\angle[301.37, 307.32]$, $I_{R2} = [0.3867, 0.5365]\angle[211.37, 217.32]$, etc. (all these quantities are phasors; the first interval represents the magnitude and the second one the angle –see Chapter II)

If the user then asks if $I_C < I_{R2}$ can be true, the system can respond with the following answer:

Contradiction detected. $I_C < I_{R2}$ contradicts the constraints

$I_C = [1.4857, 1.9539]\angle[301.37, 307.32]$, and

$I_{R2} = [0.3867, 0.5365]\angle[211.37, 217.32]$.

The set of all quantities in a circuit represents the set of possible behaviors the circuit can exhibit. When the quantities are allowed to range in intervals (magnitudes and angles of phasors as well), each possible combination of assignments of values to variables represent a behavior. Thus, the set of constraints represents a set of acceptable behaviors of the circuit. Reducing the interval domains of the variables (by asserting new constraints) reduces the number of possible behaviors. If all quantities take on real number values, the behavior of the circuit is unique. By refining the interval values of quantities until all of them become point intervals (i.e., real numbers), the set of predicted behaviors reduces to one and coincides with the behavior predicted by a conventional circuit solver.

Mixed propagation can also be performed. That is, from value constraints, order of magnitude relations can be discovered. The converse is also possible, given order of magnitude constraints, the values of the involved variables can be refined to make sure the given relation does not result in an inconsistency.

Causal reasoning, also known as First Order Reasoning[2] (FOR) is important if we want to be able to explain a circuit's behavior. Confluences capture the interaction among different variables in the circuit, and how changes in one variable can produce changes in other variables. For example, if the user asks "what happens if C increases?" (expressed as the constraint $\partial C = +$), the system replies[3]

> "If C increases, Z_C decreases, which causes Z_{P1} and Z_{S2} to decrease. This makes I_{S2}, I_{S1}, and I_{P1} increase. If I_{S1} increases, V_{S1} increases, and since V_{S2} is constant, this causes V_{P1}, V_{R2}, and V_C to decrease. If V_{R2} decreases, I_{R2} decreases, causing IC to increase. Since I_{R2} decreases and IC increases, the phase angle $\angle(I_{R2}, I_{P1})$ will increase and the angle $\angle(I_{P1}, I_C)$ will decrease."

Confluences are equations, whose variables are rates of change, taking on values from the domain of qualitative signs (i.e., -,0,+). Our value propagation mechanism propagates changes in variable values through a network of algebraic equations, where the values variables take on can be reals, intervals, or qualitative signs, all of them uniformly represented as intervals. Therefore, first order reasoning is a particular case of value propagation with interval values.

The BSOC is a Partially Constrained Model (PCM) of the circuit, which will

[2]The term First Order Reasoning comes from the fact that we are dealing with the first order derivatives of quantitites. Similarly, value propagation is called Zeroth Order Reasoning

[3]No natural language interface has been built or used. In the rest of this document, we paraphrase all questions and answers. Example 5 (figures 51 and 52, on pages 98 and 99) show a transcript of the system interaction and the results of explanation.

allow us to perform the reasoning task we have in mind. A PCM corresponds to a set of circuit behaviors; the more constrained the circuit model is, the smaller the set of possible behaviors is. For instance, we can tell that the phase angle of S_1 lies in the interval (0, 90), represented by the constraint $\angle Z_{S1} = (+\angle[0\ 90])$. After asserting that $I_C < I_{R2}$, we know that $\angle Z_{S1} = (+\angle[0\ 45])$. The set of constraints represents the set of possible behaviors a circuit can exhibit. Figure 2 shows one phasor diagram (behavior of the circuit), of the many possible for the circuit model of Figure 1. That phasor diagram was drawn under the added assumptions $I_{R2} > I_C$, $V_L > V_{R1}$, and $V_{S1} > V_{P1}$.

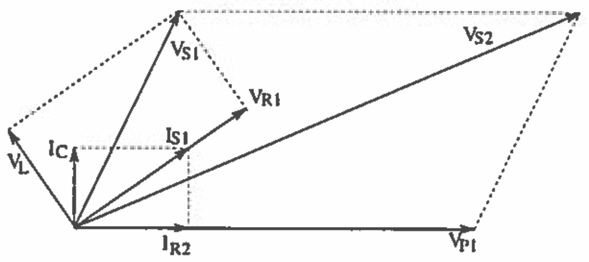


FIGURE 2. A Possible Phasor Diagram for Circuit in Figure 1

After the user has provided a number of constraints, it is more likely that further constraints are rejected as inconsistent with the partial solution. At that point the user can say "OK, give me all possible, fully constrained models ...". Even if the user was able to provide all constraints available, one of the goals of the system is to produce all possible, fully constrained models of the circuit (i.e., all possible qualitative behaviors of the circuit under the actual set of constraints). Traversing the circuit's structure, QPA determines what variables are interrelated and produces all relevant constraints. If no constraint involving those variables is found in the

PCM, all possible constraints that include those variables are produced. Nodes with constraints that produce inconsistencies are pruned and not included in the tree. The result is a tree like the one shown in Figure 3[4] where the leaves correspond to sets of constraints representing fully constrained models of the circuit.

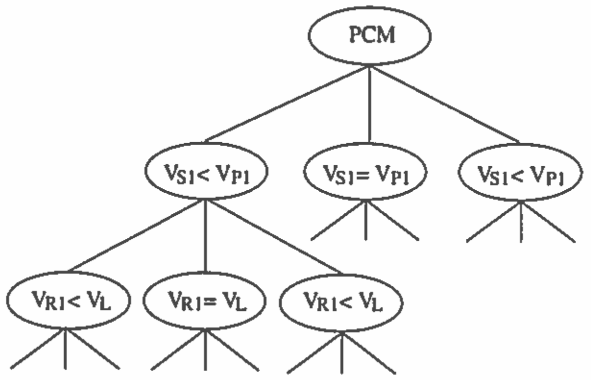


FIGURE 3. Tree Containing All Fully Constrained Models

QPA also handles order of magnitude constraints. Order of magnitude constraints can be used to simplify a circuit, when appropriate. Returning to the circuit shown in Figure 1, if the user tells the system that $Z_C \gg Z_{R2}$, the system responds that $I_C \ll I_{R2}$. This is interpreted by QPA in the appropriate way; the current through that branch is negligible, therefore, the whole branch can be omitted (open circuited). The resulting circuit is shown in Figure 4. In general, if after running propagation it is determined that the current through a branch in a parallel cluster is negligible, the element can be substituted by an open circuit. A similar simplification can be done for the case where the voltage through an element in a series cluster is

[4]The tree in Figure 3 was formed using only operators <, =, and >. The system is general enough to accept any other set (e.g., the set of order of magnitude operators)

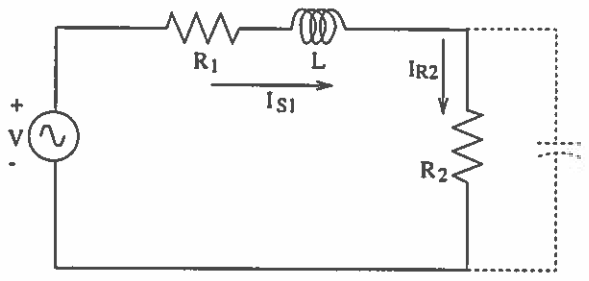


FIGURE 4. Structural Simplification by Order of Magnitude Reasoning

negligible; in that case, the element is substituted by a short circuit. Opening an element or cluster is equivalent to removing it from the circuit; short-circuiting is equivalent to removing that element or cluster and to collapsing its end-points into one. After these structural modifications, a new model of the circuit is rebuilt, and propagation on the given constraints must be recomputed.

Now consider the process of measurement interpretation or diagnosis, based on a QPA representation. Suppose that the observed state of the circuit is shown in Figure 5. The observed state, measured by physical instruments, can be easily

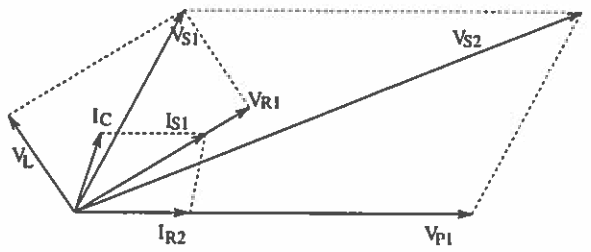


FIGURE 5. Faulty Observed Behavior

translated to a set of constraints. For instance, we observe that $V_{S_1} = (95\angle 3.5)$, $V_C = (7.5\angle 309)$, $I_C = (3.8\angle 323)$, etc. Those constraints can be verified against the circuit model by following the circuit's topology in a top down manner. If a contradiction is found, the circuit is considered faulty and a set of possible faults are suggested to the user. In this example we start by checking the constraints for cluster S_2; while no contradictions were found at this level, we need to continue verifying the rest of the circuit, traversing its structure. We continue checking S_1, P_1, and inside them, until we find that the phase angle of the current and voltage in the capacitor does not correspond to the model of that element. By the characteristics of the observation, we conclude that "*the capacitor is leaking*". In other words, it is shorted by a small resistance R_F (see Figure 6).

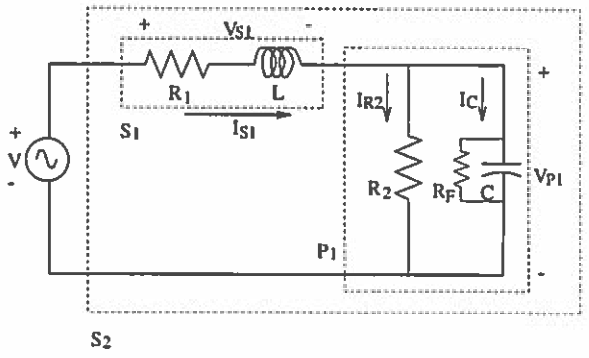


FIGURE 6. Diagnosed Fault

Order of magnitude reasoning, as proposed above, should allow the system, for example, to prove that a short-circuit with resistance, for which the fault resistance is very small (negligible), is equivalent to a circuit with a perfect short (i.e., with zero resistance). First, QPA infers $I_C \ll I_{Rf}$, which leads it to eliminate C. Then, it infers that $V_{S1} \gg V_{P1}$, which eliminates the whole cluster P_1. The final circuit is equivalent to the situation where there is an ideal short circuit in C.

Another problem we are addressing is Control Design. Control design answers questions similar to those in FOR (e.g., "How can I get the phase angle of cluster

S_1 to decrease?"). In some cases, if one or more parameters are variable, we can solve those questions by FOR alone. In some other cases, even if there are some variable parameters, there is no setting of those parameters that would make us achieve the goal(s). Furthermore, we might want to further constrain the solution to those problems (e.g., "How can I get the phase angle of cluster S_1 to decrease, without changing V_{S1}?"). The approach we take is similar to means-ends analysis, where the initial situation is represented by the working model, and the goal situation by the goals and the design constraints. The operators are circuit modifications, that, when applied to the circuit, change its conditions. We design by inserting one modification at a time until all goals are satisfied and no constraint is violated.

Given these basic reasoning capabilities, one domain of application suitable for using QPA is Power Systems Analysis and Control. Power Systems are modeled by linear circuits [20, 19], with lumped, constant parameters, and are normally operated under sinusoidal steady state; those are exactly the kinds of circuits QPA reasons about. By using QPA, we can solve basic problems in the area of power system analysis. Some of those problems are power factor correction and power distribution. Industrial loads are typically composed of resistive and inductive elements, therefore having a lagging power factor. Connecting a capacitor bank in parallel with the load corrects its power factor. The power distribution problem arises in situations where the transmitted power increases, and one of the lines is not capable of holding the resulting amount of current. Power redistribution can be accomplished by installing capacitors, or tap changing or phase shifting transformers in series with the transmission lines. QPA can reason from first principles to determine such solutions to power system problems.

Thesis Statement and Research Issues

Research in qualitative circuit analysis (and even in behavior prediction in the field of qualitative reasoning) to date focuses on analysis of transient response. Previous work cannot be applied to reasoning about the response of such systems in steady state, because it focuses only on magnitudes of *scalar* quantities. Quantities in electrical circuits in steady state are represented as a special kind of vector, called *phasors*, where not only magnitudes, but also *angles* play an important role. To develop a computational framework capable of reasoning about such systems, this dissertation investigates the following thesis statement:

> By extending the circuit ontology to include phasors and by using a constraint-based model of the circuit, we can extend the range of problems that can be solved by qualitative reasoning about complex systems.

To explore the development of such a computational framework, this dissertation addresses the following research issues.

- Can the circuit ontology be extended to adequately represent phasor and phase angle information?

- Can we use this representation to reason effectively about electrical circuits in steady state?

- What modifications to normal constraint propagation procedures are needed to deal with constraints of different kinds?

- How can the structure of a circuit be simplified, based on order of magnitude information derived from constraint propagation?

- Can we, by using a single representation, reason about the circuit at several degrees of abstraction, from qualitative to quantitative information?

- Can we design solutions for the problems of operation, diagnosis, and control of power transmission systems, based on first principles of phasor analysis?

Implementation and Evaluation

The system has been implemented in Allegro Common Lisp for Sun Workstations. The system consists of three layers: Power System Analysis and Design (PSAD), Qualitative Phasor Analysis (QPA), and Hybrid Representation Constraint Propagation (HRCP). Figure 7 shows the proposed architecture of the system.

The interface for this project is a textual symbolic description of the input and output. The input is the topological configuration of the circuit or power system, which includes the definition of each of the elements and their interconnections. The input constraints are of the form mentioned in the preceding section. The output

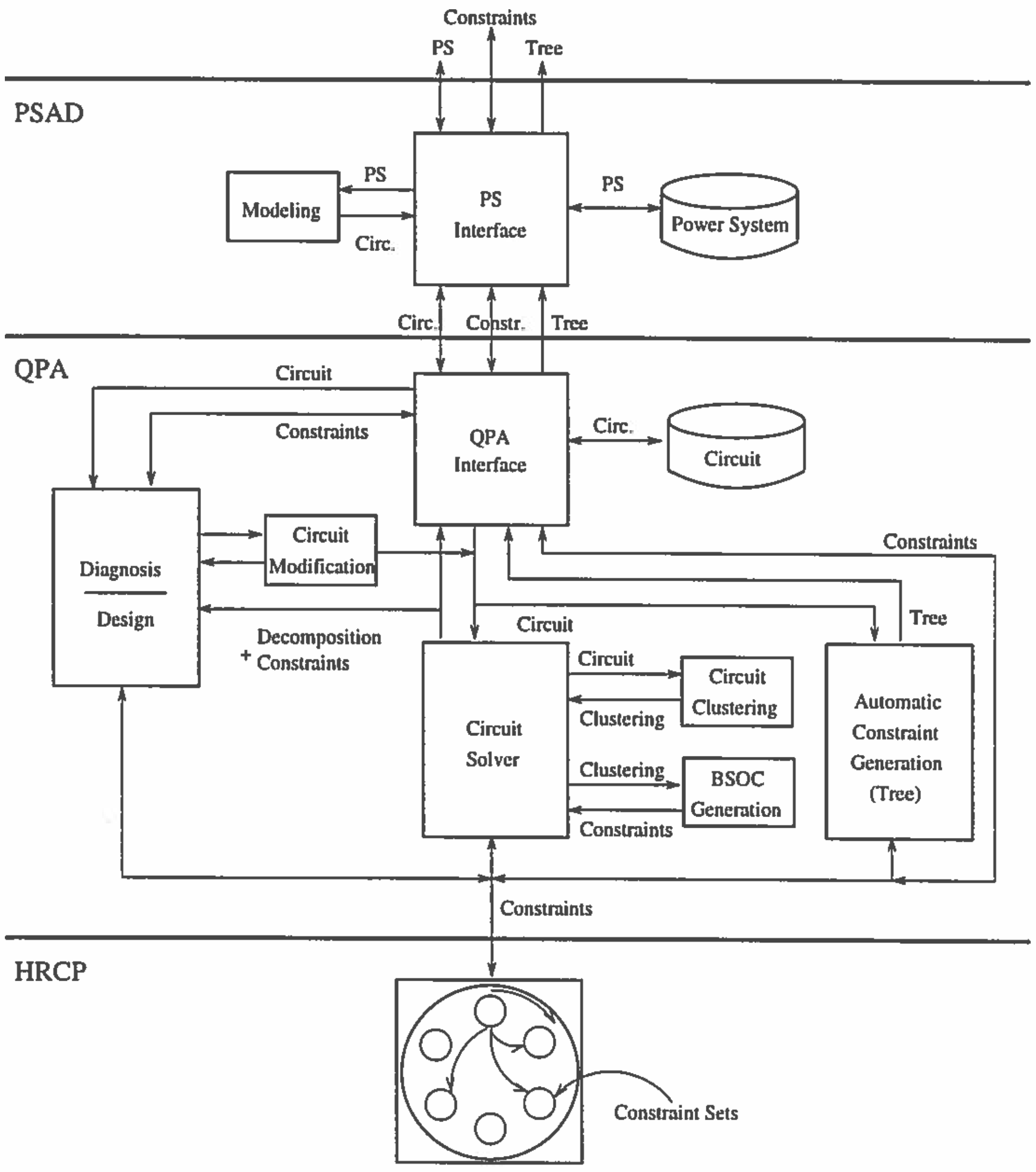


FIGURE 7. QPA's Architecture

of the system is a set of constraints, representing the partially constrained model of the circuit or power system. Value propagation, which includes first-order reasoning, forms a trace of the propagation process, which can be used to provide explanations like the ones mentioned above. In the case of diagnosis or control design, the new topological configuration of the circuit will be returned to the user.

The performance of the system was evaluated by comparing its results with examples found in textbooks, and with the results of the application of standard techniques used in circuit analysis. The usability and applicability of the system has been evaluated by experts in the area of Electrical Engineering, Artificial Intelligence, and Power Systems. For that purpose, we contacted a power engineer from Bonneville Power Administration, a faculty member from the University of Colorado, and a faculty member from the School of Electrical Engineering of the University of Michoacán.

Dissertation Organization

Chapter II presents an overview of the basic concepts in electrical engineering, phasors and our proposed representation, called *complex fans*. Chapter III focuses on the proposed modifications to constraint propagation to deal with constraints of different kinds, including order of magnitude relations, and confluences. Chapter IV addresses the reasoning tasks QPA (Qualitative Phasor Analysis) performs about linear circuits in sinusoidal steady state. Chapter V gives an introduction to QPA's main field of application, Power Systems, as well as examples of how the techniques developed for QPA can be used to solve problems of importance to the field. Chapter VI concludes the dissertation by discussing its contributions and limitations, as well as directions for future research. Each chapter includes a brief survey of related research, as appropriate.

CHAPTER II

ELECTRICAL CIRCUITS AND PHASORS

This chapter covers basic concepts in electric circuits, focusing on the solution of circuits under sinusoidal steady state. The main representational tool used in electrical engineering to perform circuit analysis in sinusoidal steady state is the phasor. We review the concept of phasor and its use in the solution of linear circuits, and present a proposed representation, called complex fans, that allows us to handle phasor information under uncertain conditions.

Phasors

The behavior of a linear circuit can be characterized by an ordinary differential equation (ODE) [4, 27, 49, 39]. If the forcing function is a sinusoid, the analytical solution of the equation, in the steady state regime, indicates that the response will also be a sinusoid. When performing circuit analysis, instead of using a real-valued driving function, we use a complex one, whose projection on the real axis (when rotating) produces the same sinusoidal. This complex quantity is called a *phasor*. By the use of phasors, the solution to a linear circuit in sinusoidal steady state can be obtained by relatively easy manipulation of complex numbers.

A phasor diagram is a graphical representation of complex exponentials, in the complex plane [27, chapter 5]. Consider the function $Ce^{j\omega t}$[1], whose characteristics are described in equation II.1 and represented graphically in Figure 8.

$$\begin{aligned}
Ce^{j\omega t} &= (a + jb)e^{j\omega t} \\
|Ce^{j\omega t}| &= \sqrt{a^2 + b^2} \\
\angle Ce^{j\omega t} &= \omega t + \angle C \\
\angle C &= \arctan \tfrac{b}{a}
\end{aligned} \tag{II.1}$$

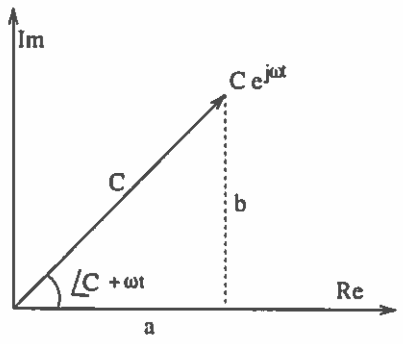


FIGURE 8. Graphical Representation of $Ce^{j\omega t}$

In a circuit excited by a sinusoidal voltage source of frequency ω, all variables (i.e. currents and voltages) are also sinusoids oscillating at the same frequency. If we represent each variable by a phasor, they will rotate at the same angular frequency, as if fastened together; what changes between each variable are its magnitude and phase angle. So a phasor diagram can be seen, at any given moment, as a snapshot

[1]In electrical engineering, the complex operator $j = \sqrt{-1}$ is used instead of the mathematical symbol i. ω is the frequency of a sinusoidal wave, see equation II.2, and t is time.

of the set of rotating phasors that represent all quantities of the circuit.

Let us now consider a circuit with a pair of terminals, for which a voltage V and a current I can be defined

$$\begin{aligned} V(t) &= Re(Ve^{j\omega t}) = |V|\cos(\omega t + \angle V) \\ I(t) &= Re(Ie^{j\omega t}) = |I|\cos(\omega t + \angle I) \end{aligned} \tag{II.2}$$

where $V(t)$ and $I(t)$ represent (real) functions of time, and V and I represent phasors in the frequency domain. The ratio between voltage and current will be called *impedance*, denoted by Z, and its inverse *admittance*, denoted by Y; i.e.,

$$\begin{aligned} Z(jw) &= V/I \\ Y(jw) &= I/V = 1/Z(jw) \end{aligned} \tag{II.3}$$

The impedance and admittance for individual resistors, inductors, and capacitors are given in the Table 1.

Element	$Z(jw)$	$Y(jw)$	$\frac{\|V\|}{\|I\|}$	$\angle V - \angle I$
R	R	$1/R$	R	0
L	jwL	$1/jwL$	wL	+90
C	$1/jwC$	jwC	$1/jwC$	−90

TABLE 1. Properties of Circuit Elements

For series elements, the current phasor is common and the voltage phasor is the sum of the element voltage phasors; so impedances in series add as in the case of resistors in series. The analogous case holds for elements in parallel

$$Z_{Ser}(jw) = \sum_i Z_i(jw)$$
$$Y_{Par}(jw) = \sum_i Y_i(jw) \tag{II.4}$$

Impedances and admittances can be used to perform circuit analysis using the same techniques as for networks with direct current sources, except that now all the quantities will be complex numbers; in the end, only the real parts will be considered as the result. This permits a graphical representation of Kirchoff's laws in terms of addition of phasors. This can be illustrated with the circuit in Figure 9, for which one phasor diagram (of the many possible) is shown in Figure 10.

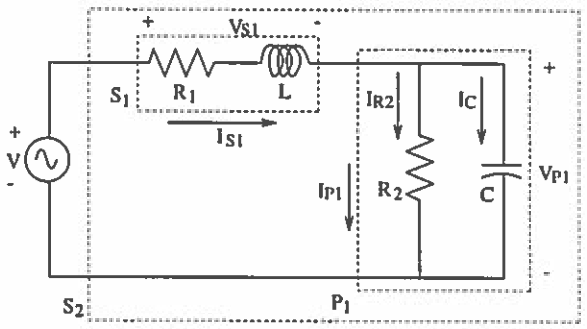


FIGURE 9. An Electrical Circuit

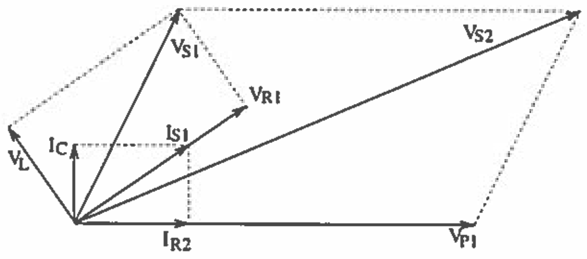


FIGURE 10. A Possible Phasor Diagram for Circuit in Figure 9

Power Factor and Phase Angle

An important concept that deserves consideration at this point is that of power factor. *Power* is defined as the rate of energy transfer; in terms of a circuit quantities, it is defined as the product of the instantaneous voltage and the instantaneous current that appears through an element.

$$p = VI \tag{II.5}$$

If the voltage and current of a circuit element are expressed by

$$\begin{aligned} V &= V_{max}\cos(\omega t) \\ I &= I_{max}\cos(\omega t - \theta) \end{aligned} \tag{II.6}$$

the instantaneous power is

$$p = V_{max} I_{max}\cos(\omega t)\cos(\omega t - \theta) \tag{II.7}$$

A positive value of P indicates power is being absorbed by the element. That is, the element is acting as a load or consumer of energy. This situation is encountered when the current has the same sign as the voltage drop in the element(i.e. current is flowing in the direction of voltage drop). In the other case, when voltage and current have different signs (i.e. current is flowing in the direction of voltage rise), energy is being transferred from the element to the system.

For a purely resistive element, voltage and current are in phase; current always flows in the direction of voltage drop, therefore, it is always absorbing power. In a pure inductor or capacitor, the current is always 90 degrees out of phase with respect to the voltage. In that case, power alternates flowing to and from the element (i.e. acting as a load and a source, respectively).

By using trigonometric identities, the expression of equation II.7 is reduced to

$$p = \frac{V_{max} I_{max}}{2} \cos\theta (1 + \cos 2\omega t) + \frac{V_{max} I_{max}}{2} \sin\theta \sin 2\omega t \qquad \text{(II.8)}$$

The first term of equation II.8 is always positive, and is normally referred to as the *real* or *active* power. The second term alternates between positive and negative values; it is known as the *reactive* power and expresses the flow of energy alternately toward the load and toward the system. The cosine of the phase angle θ between the voltage and current is called the *power factor*. An inductive circuit is said to have a lagging power factor, and a capacitive circuit is said to have a leading power factor. A resistive load has a unit power factor (i.e. voltage and current are in phase, so θ is zero). Power, active power, and reactive power are very useful terms in describing the operation of a Power System (see section V).

Figures 11 and 12 show the current, voltage, and power sine waves for circuits with unity and lagging power factors, respectively. Figure 11 represents a circuit with a net resistive effect. Power in a resistive circuit is always positive, that is, resistors have no means to store energy and are always dissipating it in form of heat. Figure 12 represents a circuit with net resistive and inductive effects, like the one in Figure 13.

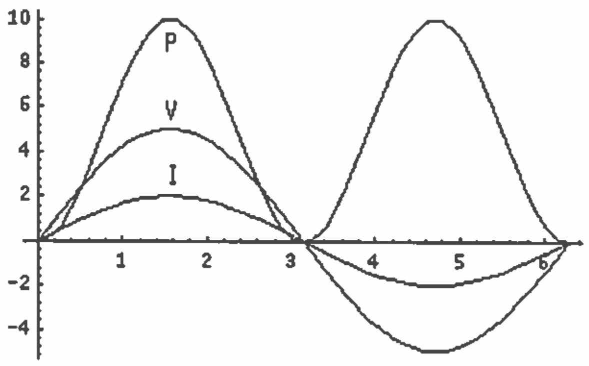


FIGURE 11. Power in a Circuit with Unity Power Factor

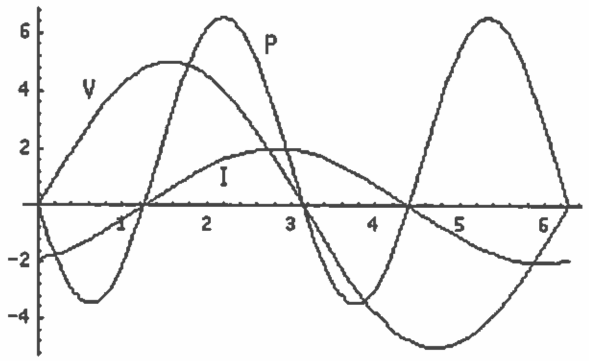


FIGURE 12. Power in a Circuit with Lagging Power Factor

In that case, the inductor is storing energy (in the form of a magnetic field) when p becomes positive, and returning it to the system when p becomes negative. Notice that the average consumed power is the part dissipated by the resistor. Although the dissipated power depends only on the resistive part of the circuit, the magnitude of the current does increase with the reactive power. This increase in current will be reflected by an increase in losses and higher capacity and size are required in all transmission equipment.

This problem can be corrected by inserting a capacitor in the circuit, as shown in Figure 14.

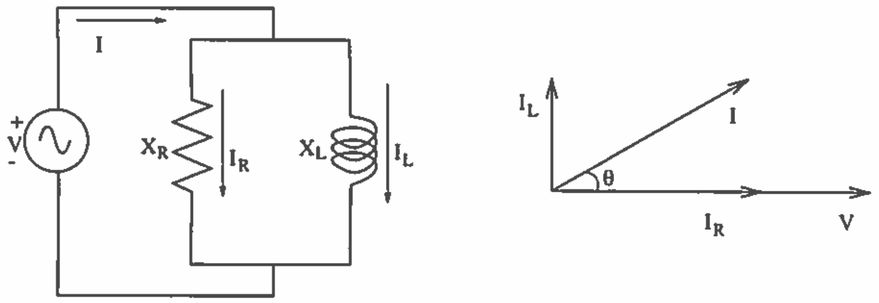


FIGURE 13. RL Circuit and Corresponding Phasor Diagram

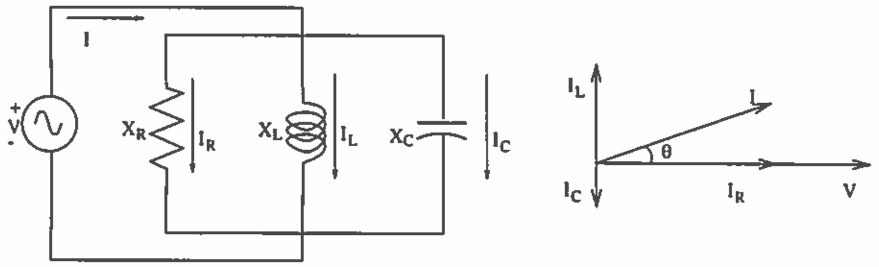


FIGURE 14. RLC Circuit and Corresponding Phasor Diagram

In that case, (part of) the reactive power needed by the inductor is supplied by the capacitor. As a result, there is no reactive power traveling back and forth from the source to the load, and the magnitude of the resulting current is smaller. We can say that the power factor has been corrected.

Complex Fans

An important feature of our reasoning engine is the ability to transition smoothly from qualitative to quantitative values in the process of reasoning. This can be accomplished by representing every value as an interval and performing all operations using interval algebra. Phasors are complex quantities, so we need two numbers to represent such a quantity. If we choose the rectangular representation, those two numbers are the real and imaginary components of the complex number. In polar representation, a phasor is expressed by its magnitude and angle. Traditionally, in the field of Electrical Engineering, complex numbers are represented in rectangular form, because arithmetic operations are simpler in this representation. Our representation language talks about magnitudes and phase angles; for instance, given values for quantities V_1 and V_2, the order relation $V_1 > V_2$ can be used to limit the possible values of the magnitude of both quantities. If those quantities were represented in rectangular form, we can reduce the real, the imaginary, or both components to reduce the resulting magnitude. Similarly with angle values and angle ordering constraints. So, we decided to represent phasors in polar form.

If we allow the magnitude and angle of a phasor to range over intervals, the resulting objects, instead of denoting a point in the complex domain, now span over circular segments. We call those circular segments *complex fans.* For example, fan V of Figure 15 is a phasor whose magnitude ranges from a to b, and whose angle ranges from α_1 to α_2.

In this section, we develop the complex fan arithmetic that will be needed to perform value propagation on the domain of interval phasors, or complex fans. In developing the complex fan arithmetic, we pay special attention to providing minimality in the results. That is, the results of an operation is the smallest complex fan that encloses all possible results of that operation.

The basic arithmetic operations that need to be defined are $+$, $-$, $*$, and $/$.

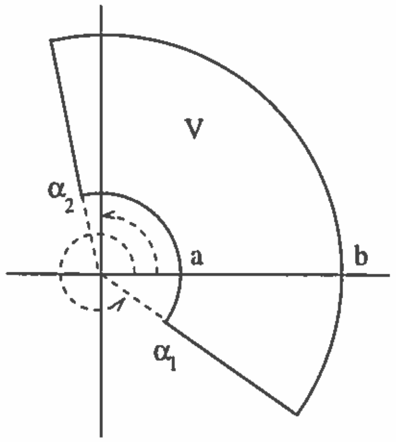


FIGURE 15. Example of a Complex Fan

Other operations can be defined in terms of the basic ones. The formulae for product, division, and negation (i.e. unary −), operate independently on magnitudes and angles. So the formulae for phasors can be used, performing standard interval operations, instead of real-valued ones. The formulae for addition (and subtraction, which is defined in terms of addition and negation) are more complicated, and we cannot just apply interval operators in a straightforward manner.

In this section, we develop an algorithm to evaluate complex fan addition, yielding the minimum complex fan that encloses all possible results. The derivation of the algorithm is justified by the mathematics for deriving the formulas presented here.

Notation

The complex fan shown in Figure 15 can be represented as

$$V = Vm\angle V_\alpha \tag{II.9}$$

where V_m represents V's magnitude, and V_α its angle. V_m ranges over the interval $0 \le a \le V_m \le b$, and V_α ranges over the interval $\alpha_1 \le V_\alpha \le \alpha_2$. Then, the complex fan V can also be expressed as

$$V = [a, b]\angle[\alpha_1, \alpha_2] \tag{II.10}$$

Note that if $a = b$ and $\alpha_1 = \alpha_2$, the complex fan reduces to a phasor (i.e. a point in the complex plane).

Throughout the rest of the chapter, when we deal with two fans, we assume

$$\begin{aligned} V_1 &= V_{1m}\angle V_{1\alpha} = [a, b]\angle[\alpha_1, \alpha_2] \\ V_2 &= V_{2m}\angle V_{1\alpha} = [c, d]\angle[\alpha_3, \alpha_4] \end{aligned} \tag{II.11}$$

In the figures of section II.3, we (arbitrarily) number the corners of the fans as indicated in Figure 16. Also, the addition of two corners i and j will be indicated by point ij. For example, addition of points 2 and 5 yields point 25, as shown in the same figure.

Product

As mentioned above, formulae for product, division, and negation of complex fans can be obtained by simply replacing real-valued operations by interval operations in the respective formulae for phasor arithmetic.

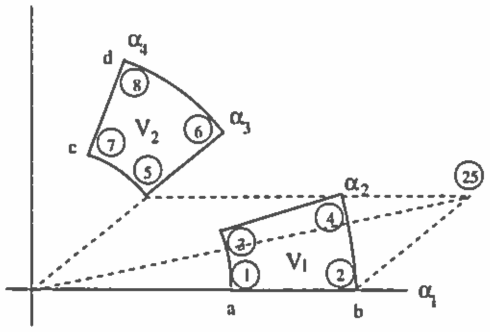


FIGURE 16. Notation

The product of two phasors $V = V_1 * V_2$ is given by the formula

$$V = ([a,b] * [c,d])\angle([\alpha_1,\alpha_2] + [\alpha_3,\alpha_4]) \tag{II.12}$$

Division

The quotient of two phasors $V = V_1/V_2$ is analogous to the product

$$V = ([a,b]/[c,d])\angle([\alpha_1,\alpha_2] - [\alpha_3,\alpha_4]) \tag{II.13}$$

Negation

The negation of a complex fan is another complex fan, with the same magnitude, and whose angle is the complement of the angle of the original fan.

$$V = -V_1 = [a,b]\angle([\alpha_1,\alpha_2] + [180,180]) \tag{II.14}$$

Subtraction

Subtraction can be defined in terms of addition and negation as follows

$$V = V_1 - V_2 = V_1 + (-V_2) \tag{II.15}$$

Addition

Addition presents a complication. There is no formula to add two phasors in polar form. In one approach, the addends (i.e. phasors) are converted to rectangular form, the addition is performed and then the result is transformed back into polar form. When we generalize the idea to intervals, the smallest rectangle that encloses a fan includes some points not present in the original fan. Then, when we convert the resulting rectangle to a fan, more points are included than are needed. This is illustrated in Figure 17

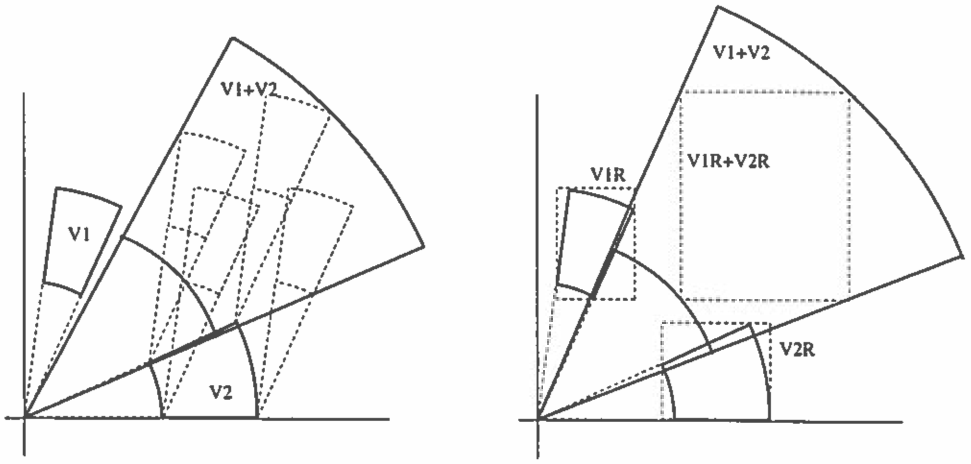


FIGURE 17. Imprecision Added by Representation Changes

We will present an algorithm that computes the smallest possible fan that encloses the result of $V_1 + V_2$, where

$$V_1 + V_2 = \{v + w \mid (v, w) \in V_1 \times V_2\} \tag{II.16}$$

The result of the addition must be complete and minimal. To be complete, it must contain all possible results of the addition, as defined above. To be minimal, its boundaries (both, magnitude and phase angle) must be such that the intervals they represent cannot shrink without leaving possible results out. This implies that at least one result must fall on each of the boundaries of the resulting complex fan.

Clearly, one possible value can be obtained by adding the corners of the fans. Nevertheless, these extreme values do not necessarily produce the extreme values of the result. The only case when the corners produce the extreme results is when the fans span less than 90 degrees. So, we will decompose V_1 and V_2 into four fans, representing their intersection with each quadrant, taking as reference the smallest angle of the first fan (i.e. V_1 always starts at zero). Ensuring this assumption might require a rotation, in which case, the final results will need to be corrected back to the original reference. For instance, V_1 will be

$$V_1 = \bigcup_{i=1\ldots4} V_{1i} \tag{II.17}$$

where

$$V_{1i} = (0,\infty)\angle[90(i-1), 90i] \cap V_1 \tag{II.18}$$

Thus, addition can be expressed as

$$\begin{aligned} V &= V_1 + V_2 \\ &= (V_{11} \cup V_{12} \cup V_{13} \cup V_{14}) + (V_{21} \cup V_{22} \cup V_{23} \cup V_{24}) \\ &= \{v + w \mid (v, w) \in (V_{11} \cup V_{12} \cup V_{13} \cup V_{14}) \times (V_{21} \cup V_{22} \cup V_{23} \cup V_{24})\} \end{aligned} \tag{II.19}$$

and since Cartesian product distributes over union,

$$\begin{aligned} V &= \{v + w \mid (v, w) \in (V_{11} + V_{21} \cup V_{11} + V_{22} \ldots)\} \\ &= \bigcup_{i,j=i\ldots4} V_{1i} + V_{2j} \end{aligned} \tag{II.20}$$

Each partial addition is performed with fans that do not extend more than 90 degrees. We can categorize each partial addition into one of three cases: when the two addends are in the same, adjacent, or opposite quadrants. We analyze each case and determine a procedure to compute the partial results. The final result is the union of all the partial results. For each case considered, we will demonstrate that the minimality condition is met.

Case 1: Same Quadrant

In this part, we consider the case when both fans are in the same quadrant. The analysis is made for the first quadrant; for other quadrants, we just apply a rotation, which will be corrected at the end of the process.

Consider the addition $V = V_1 + V_2$, where $\alpha_1 = 0$, since we take V_1 as our reference. Vector algebra states that

$$\begin{aligned} V_m &= \sqrt{V_{1m}^2 + V_{2m}^2 + 2V_{1m}V_{2m}\cos\theta} \\ V_\alpha &= \tan^{-1}\left(\frac{V_{1m}\sin V_{1\alpha} + V_{2m}\sin V_{2\alpha}}{V_{1m}\cos V_{1\alpha} + V_{2m}\cos V_{2\alpha}}\right) \end{aligned} \tag{II.21}$$

where $\theta = V_{1\alpha} - V_{2\alpha}$.

Since square root is a monotonic function, V_m reaches an extreme when V_m^2 does. Since addition monotonically increases with its addends, and cosine is monotonically decreasing in θ's domain (i.e. $0 \leq \theta \leq 90$), we have the following,

$$\begin{aligned} V_{m_{max}}^2 &= V_{1m_{max}}^2 + V_{2m_{max}}^2 + 2V_{1m_{max}}V_{2m_{max}}\cos\theta_{max} \\ &= b^2 + d^2 + 2bd\cos\theta_{min} \end{aligned}$$

$$\begin{aligned} V_{m_{min}}^2 &= V_{1m_{min}}^2 + V_{2m_{min}}^2 + 2V_{1m_{min}}V_{2m_{min}}\cos\theta_{min} \\ &= a^2 + c^2 + 2ac\cos\theta_{max} \end{aligned} \tag{II.22}$$

The final computation for the extremes of θ and V_m (see Figure 18), can be expressed as follows

$$\begin{aligned}
\theta_{max} &= \max(\alpha_4 - \alpha_1, \alpha_2 - \alpha_3) \\
\theta_{min} &= \begin{cases} 0 & V_{1\alpha} \cap V_{2\alpha} \neq \phi \;\; (\alpha_2 \geq \alpha_3) \\ \alpha_3 - \alpha_2 & otherwise \end{cases} \\
V_{mmax} &= \sqrt{V_{mmax}^{\,2}} \\
V_{mmin} &= \sqrt{V_{mmin}^{\,2}} \qquad \text{(II.23)}
\end{aligned}$$

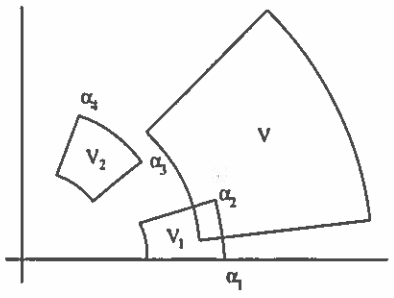


FIGURE 18. Phasor Addition, Case 1

For the computation of V_α, $\tan^{-1}$ is a monotonic function. To find the extremes of the function, all we need to do is find the extrema of its argument. A change of variables simplifies the expressions,

$$f(x, y, u, v) = \frac{x \sin u + y \sin v}{x \cos u + y \cos v} \qquad \text{(II.24)}$$

The parallelogram rule for vector addition [46] suggests that the smallest possible angle occurs adding points 2 and 5, denoted by point 25 in Figure 19, and the largest angle occurs adding points 3 and 8, denoted by point 38 in the same figure.

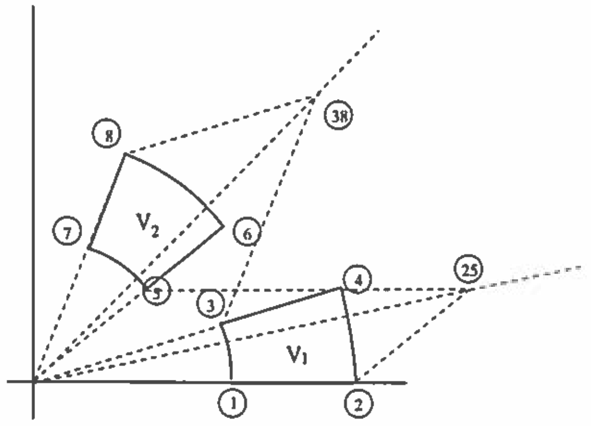


FIGURE 19. Maximum and Minimum Angles in Phasor Addition

To prove our intuition, we first show that the minimum angle has to occur when adding some points in the boundaries of minimum angles. For a given x_0 and y_0 (i.e. fixing the magnitudes of both fans),

$$g_1(u,v) = \frac{x_0 \sin u + y_0 \sin v}{x_0 \cos u + y_0 \cos v} \tag{II.25}$$

The extremes are found where the gradient is zero,

$$\begin{aligned} \nabla g_1 &= 0 \qquad\qquad (\text{II.26}) \\ \frac{\partial g_1}{\partial u} &= \frac{x_0 \cos u (x_0 cos u + y_0 \cos v) - (-x_0 \sin u)(x_0 \sin u + y_0 \sin v)}{(x_0 \cos u + y_0 \cos v)^2} = 0 \\ \frac{\partial g_1}{\partial v} &= \frac{y_0 \cos v (x_0 cos u + y_0 \cos v) - (-y_0 \sin v)(x_0 \sin u + y_0 \sin v)}{(x_0 \cos u + y_0 \cos v)^2} = 0 \end{aligned}$$

$$x_0 + y_0 \cos(u - v) = 0 \tag{II.27}$$

$$x_0 = -y_0 \cos(u - v)$$

Since x_0, y_0, and $\cos(u - v)$ are positive, there is no solution to equation II.27. Therefore, the function has no extremes in the given domain. So, we analyze how the function behaves on each variable. For a given u_0 (i.e. fixing the angle of the second fan),

$$\begin{aligned} h_1(v) &= g_1(u_0, v) \\ h_1'(v) &= \frac{x_0^2 + x_0 y_0 \cos(u_0 - v)}{(x_0 \cos u_0 + y_0 \cos v)^2} \end{aligned} \tag{II.28}$$

and for a given v_0,

$$\begin{aligned} h_2(u) &= g_1(u, v_0) \\ h_2'(u) &= \frac{y_0^2 + x_0 y_0 \cos(u - v_0)}{(x_0 \cos u + y_0 \cos v_0)^2} \end{aligned} \tag{II.29}$$

Since

$$-90 \leq u_0 - v \leq 90, \; 0 \leq \cos(u_0 - v) \leq 1 \tag{II.30}$$

therefore

$$\begin{aligned} h_1'(v) &> 0 \\ h_2'(u) &> 0 \end{aligned} \tag{II.31}$$

This indicates that f (the angle of the result) grows with u and v (the angles of the addends), so it reaches its maximum (minimum) value in the domain, when u and v are maxima (minima).

We proceed to determine f's behavior with respect to x and y (the magnitudes of the addends).

$$\begin{aligned} g_2(x,y) &= f(x,y,u_0,v_0) \\ &= \frac{x\sin u_0 + y\sin v_0}{x\cos u_0 + y\cos v_0} \end{aligned} \tag{II.32}$$

$$\begin{aligned} \nabla g_2(x,y) &= 0 \\ \frac{\partial g_2}{\partial x} &= \frac{\sin u_0(x\cos u_0 + y\cos v_0) - (\cos u_0)(x\sin u_0 + y\sin v_0)}{(x\cos u_0 + y\cos v_0)^2} \\ \frac{\partial g_2}{\partial y} &= \frac{\sin v_0(x\cos u_0 + y\cos v_0) - (\cos v_0)(x\sin u_0 + y\sin v_0)}{(x\cos u_0 + y\cos v_0)^2} \end{aligned}$$

$$\begin{aligned} y\sin(u_0 - v_0) &= 0 \\ x\sin(v_0 - u_0) &= 0 \end{aligned} \tag{II.33}$$

Since x and y are both positive, solving equations II.33 reduces to solving $\sin(u_0 - v_0) = 0$ and $\sin(v_0 - u_0$. The solution to the first is found for $u_0 - v_0 = k\pi,\ k = 0, 1, 2\ldots$, which is located outside the domain of the angle difference for the case we are analyzing. We analyze how the function behaves with respect to x and y (magnitudes). For a given y_0 (i.e. fixing the magnitude of the second fan),

$$\begin{aligned} h_3(x) &= g(x, y_0) = \frac{x \sin u_0 + y_0 \sin v_0}{x \cos u_0 + y_0 \cos v_0} \\ h_3'(x) &= \frac{y_0 \sin(u_0 - v_0)}{(x \cos u_0 + y_0 \cos v_0)^2} \end{aligned} \tag{II.34}$$

and for a given x_0

$$\begin{aligned} h_4(y) &= g(x_0, y) = \frac{x_0 \sin u_0 + y \sin v_0}{x_0 \cos u_0 + y \cos v_0} \\ h_4'(y) &= \frac{x_0 \sin(v_0 - u_0)}{(x_0 \cos u_0 + y \cos v_0)^2} \end{aligned} \tag{II.35}$$

From equations II.31, we see that to compute the maximum angle, we need to use α_2 and α_4. Assuming $u_0 = \alpha_2$ and $v_0 = \alpha_4$, we can distinguish three cases. When $\alpha_2 > \alpha_4$, $\sin(u_0 - v_0) > 0$, which makes $h_3'(x) > 0$ and $h_4'(y) < 0$, we compute the maximum angle using $x = b$ (the maximum) and $y = c$ (the minimum). When $\alpha_2 < \alpha_4$, which makes $h_3'(x) < 0$ and $h_4'(y) > 0$. We compute the maximum angle using $x = a$ and $y = d$. When $\alpha_2 = \alpha_4$, the maximum angle is equal to $\alpha_2 = \alpha_4$, i.e. it does not depend on the values of of x and y.

To compute the minimum angle, we assume $u_0 = \alpha_1$ and $v_0 = \alpha_3$. Here, we have only one case, $\alpha_1 \leq \alpha_3$, and the minimum angle is computed using $x = b$ and $y = c$.

Figure 20 shows a summary of the partial computation of fan addition for case 1.

```
AddCase1($V_1$, $V_2$)
    if $\alpha_2 \geq \alpha_3$
        $\theta_{min} = 0$
    else
        $\theta_{min} = \alpha_2 - \alpha_3$
    $\theta_{max} = \max(\alpha_4 - \alpha_1, \alpha_2 - \alpha_3)$

    $V_{mmin} = \sqrt{a^2 + c^2 + 2ac\cos\theta_{max}}$
    $V_{mmax} = \sqrt{b^2 + d^2 + 2bd\cos\theta_{min}}$

    $V_{\alpha min} = \tan^{-1}\left(\frac{b\sin\alpha_1 + c\sin\alpha_3}{b\cos\alpha_1 + c\cos\alpha_3}\right)$

    if $\alpha_2 < \alpha_4$
        $V_{\alpha max} = \tan^{-1}\left(\frac{a\sin\alpha_2 + d\sin\alpha_4}{a\cos\alpha_2 + d\cos\alpha_4}\right)$
    else if $\alpha_2 > \alpha_4$
        $V_{\alpha max} = \tan^{-1}\left(\frac{b\sin\alpha_2 + c\sin\alpha_4}{b\cos\alpha_2 + c\cos\alpha_4}\right)$
    else if $\alpha_2 > \alpha_4$
        $V_{\alpha max} = \alpha_2$

return($[V_{mmin}, V_{mmax}]$, $\angle[V_{\alpha min}, V_{\alpha max}]$)
```

FIGURE 20. Pseudo-Code for Case 1

Case 2: Adjacent Quadrants

We can assume that fan V_1 will be in the first quadrant, and fan V_2 will be in the second. If that is not the case, we can make a rotation, and the results will be corrected at the end. So, we have the following conditions for case 2:

$$\begin{aligned} 0 \leq \alpha_1 \leq \alpha_2 \leq 90 \\ 90 \leq \alpha_3 \leq \alpha_4 \leq 180 \\ and\ (\alpha_4 - \alpha_1) > 90 \end{aligned} \tag{II.36}$$

If the last condition $(\alpha_4 - \alpha_1) > 90$ is not satisfied, the problem can be reduced to Case 1 by a simple rotation. From equations II.21, if we fix the magnitudes, V_m depends solely on θ.

$$V_m = f_2(x, y, \theta) = x^2 + y^2 + 2xy\cos\theta \tag{II.37}$$

Since $0 \leq \theta \leq 180$, and cosine is monotonically decreasing with respect to θ, V_m reaches a maximum when θ is at its minimum value. When $\theta \leq 90$, $0 \leq \cos\theta \leq 1$, and f_2 reaches a maximum when x and y reach their maxima. When $\theta > 90$, $-1 \leq \cos\theta < 0$, we have a more complex situation. Analyzing how f_2 changes with θ,

$$\begin{aligned} \frac{\partial f_2}{\partial \theta} &= 2xy(-\sin\theta) \\ 90 &< \theta \leq 180 \\ \frac{\partial f_2}{\partial \theta} &\leq 0 \end{aligned} \tag{II.38}$$

we have a minimum where the derivative is zero (i.e. $\theta = 180$). If that point does not belong to θ's domain, we know that the function is monotonically decreasing with respect to θ. That is, V_{mmax} occurs for θ_{min}, and V_{mmin} occurs for θ_{max}.

Now, let us see how V_m behaves with respect to x and y, for a given θ.

$$\begin{aligned} V_m &= f_3(x,y) = f(x,y,\theta_0) \\ \nabla f_3 &= 0 \\ \frac{\partial f_3}{\partial x} &= 2x + 2y\cos\theta_0 = 0 \\ \frac{\partial f_3}{\partial y} &= 2y + 2x\cos\theta_0 = 0 \end{aligned} \tag{II.39}$$

$$\begin{aligned} x &= -y\cos\theta_0 \\ y &= -x\cos\theta_0 \end{aligned} \tag{II.40}$$

To compute V_{mmax} we consider $\theta_0 = \theta|_{min}$. We distinguish three regions.

$$\theta_0 \begin{cases} < & 90 \\ = & 90 \\ > & 90 \end{cases} \tag{II.41}$$

For the case when $\theta_0 \leq 90,\ \cos\theta_0 \geq 0$, so we see that

$$\frac{\partial f_3}{\partial x} > 0$$

$$\frac{\partial f_3}{\partial y} > 0 \tag{II.42}$$

therefore, V_m reaches a maximum at x_{max}, y_{max}. That is,

$$V_{mmax} = \sqrt{b^2 + d^2 + 2bd\cos\theta_{min}} \tag{II.43}$$

For the case when $90 < \theta_0 < 180$, $-1 \leq \cos\theta_0 < 0$,

$$\begin{aligned} \frac{\partial f_3}{\partial x} &= 2x + 2y\cos\theta_0 = 0 \\ x &= -y\cos\theta_0 \end{aligned} \tag{II.44}$$

We note that $\frac{\partial^2 f_3}{\partial x^2} = 2$, which implies the extrema above is a minimum. This in turn implies that the maximum of V_m has to be in one of the corners of $V_{1m} \times V_{2m}$, when the angle is minimum. That is (see Figure 21),

$$V_{mmax} = \max(p35_m, p36_m, p45_m, p46_m) \tag{II.45}$$

To compute f_{min} we consider $\theta_0 = \theta_{max} = \alpha_4 - \alpha_1$. We find only one case, $90 < \theta_0 \leq 180$ (see the case definition, equation II.36), and the minimum is not necessarily in one of the corners. From equation II.44, if x and y are reals, we find a minimum at $x = -y\cos\theta_0$. In general, x and y are real intervals, so if $I_x = x \cap -y\cos\theta_0 \neq \emptyset$, we

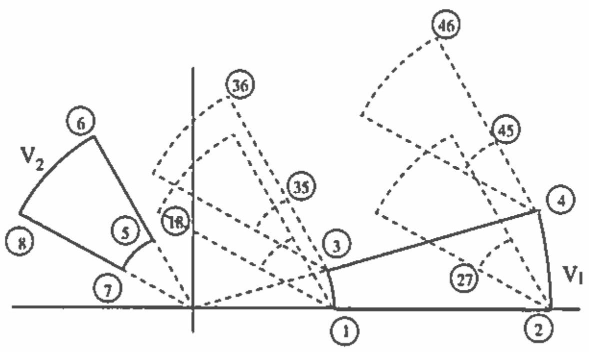


FIGURE 21. Fan Addition in the Second Case.

have a minimum at $x_m = I_{xmin}$. If they do not intersect, depending on whether x falls to the left or right of $-y\cos\theta_{max}$, we can use x_{max} or x_{min}, respectively (see Figure 22). The situation is symmetric for y, depending on whether y intersects, or falls to the left

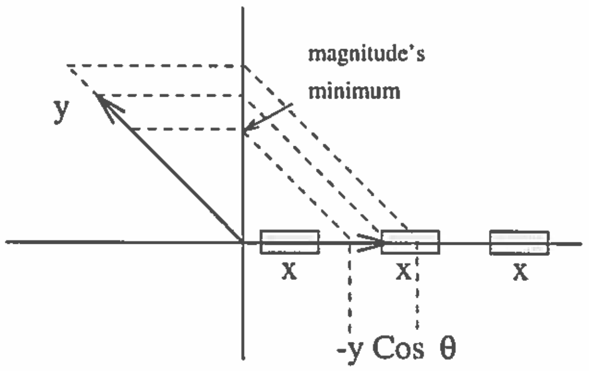


FIGURE 22. Minimum Magnitude of Phasor Addition; x and y Are Intervals.

or right of $-x\cos\theta_{max}$, we can use $y_m = I_{y_{min}}$, y_{max}, or y_{min}, respectively. Figure 23 shows the algorithm to compute the magnitude for fan addition, for Case 2.

We now proceed to derive the minimum and maximum values for the angle. From equation II.31, we have that $h_1'(v) > 0$ and $h_2'(u) > 0$. This indicates that the maximum angle has to be computed using α_2 and α_4.

From equation II.40, we have that $x_0 = -y_0\cos\theta$, where $0 \leq \theta \leq 180$. For $\theta =$

```
MagnitudeCase2($V_1$, $V_2$)
    $\theta_{min} = \alpha_3 - \alpha_2$
    $\theta_{max} = \alpha_4 - \alpha_1$
    if $\theta_{min} \leq 90$
        $V_{mmax} = \sqrt{b^2 + d^2 + 2bd\cos\theta_{min}}$
    else
        $V_{mmax} = \max(p35_m, p36_m, p45_m, p46_m)$

    $I_x = V_{1m} \cap (-V_{2m}\cos\theta_{max})$
    if $I_x \neq \emptyset$
        $x_m = I_{xmin}$
    else if $a > -d\cos\theta_{max}$
        $x_m = a$
    else if $b < -c\cos\theta_{max}$
        $x_m = b$

    $I_y = V_{2m} \cap (-V_{1m}\cos\theta_{max})$
    if $I_y \neq \emptyset$
        $y_m = I_y min$
    else if $c > -b\cos\theta_{max}$
        $y_m = c$
    else if $d < -a\cos\theta_{max}$
        $y_m = d$

    $V_{mmin} = \sqrt{x_m^2 + y_m^2 + 2x_m y_m \cos\theta_{max}}$
return([$V_{mmin}, V_{mmax}$])
```

FIGURE 23. Pseudo-Code for Magnitude, Case 2

90 there is no solution, since $x_0 > 0,\ y_0 > 0$, and $\cos\theta = 0$. For $\theta < 90,\ 0 < \cos\theta \leq 1$, so there is no solution in the domain either, since $x_0 > 0,\ y_0 > 0$, and $\cos\theta > 0$. Later on, we will show that point 38 yields the maximum angle.

For the case when $\theta > 90,\ -1 \leq \cos\theta < 0$, from equation II.40, we can determine the angle for which the extreme happens.

$$\theta = \cos^{-1}\left(-\frac{y_0}{x_0}\right) \tag{II.46}$$

This is the value of θ that produces the maximum angle. Figure 24 shows a case where the extremes for the angle cannot be computed using the corners of the fans. The same figure shows a diagram of the computation of θ and ϕ. Depending on the

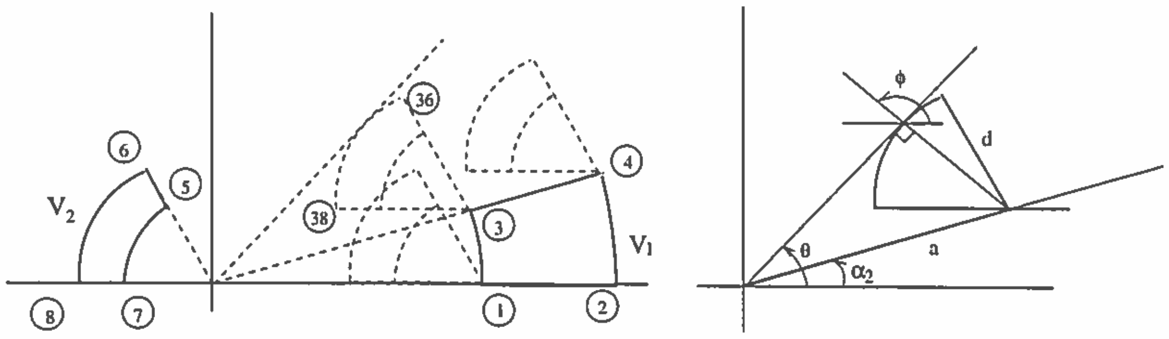


FIGURE 24. Case where Correction Is Needed, and Computation of ϕ

values of ϕ, we can determine the maximum magnitude as follows

$$\phi \begin{cases} \in & V_2 \text{ use } \theta \text{ to compute max angle} \\ < & V_2 \text{ use point38} \\ > & V_2 \text{ use point36} \end{cases} \tag{II.47}$$

To prove that points 38 and 36 yield the maximum angle, we have to analyze how the angle behaves with respect to x and y. From equation II.33, we find a solution for $k = 1,\ \theta = 180$. This is a special case, when angle doesn't change with x or y.

From equations II.34 and II.35, with $0 \leq \theta \leq 180$, we have that

$$\begin{aligned} h_3'(x) &< 0 \\ h_4'(y) &> 0 \end{aligned} \tag{II.48}$$

which shows that V_α reaches a minimum for x_{min} and y_{max}. Similarly, V_α reaches a maximum for x_{max} and y_{min}.

The algorithm to compute the minimum angle is the mirror image of the one for the maximum angle. Summarizing, the algorithm to compute the angle of fan addition for the second case is shown in Figure 25.

Case 3: Opposite Quadrants

This section covers the partial computation of $V = V_1 + V_2$, for the case where V_1 and V_2 are in opposite quadrants. Again, we can assume that fan V_1 will be in the first quadrant, and fan V_2 in the third one. If that is not the case, we can make a rotation, and the results will be corrected at the end. So, the following conditions

```
AngleCase2($V_1$, $V_2$)
    $V_{\alpha min} = \min p25_a, p27_a$
    $V_{\alpha max} = \max p18_a, p38_a$
    if $V_{\alpha max} < 90$
        $\theta = \alpha_2 + \sin^{-1}(d/a)$
        $\phi = \theta + 90$
        if $\phi \cap V_{2a}$
            $V_{\alpha max} = \theta$
        else if $\phi < \alpha_3$
            $V_{\alpha max} = p36_\alpha$
        else if $\phi > \alpha_4$
            $V_{\alpha max} = p38_a)$
    if $V_{\alpha min} > 90$
        $\theta = \alpha_3 - \sin^{-1}(b/c)$
        $\phi = \theta - 90$
        if $\phi \cap V_{1\alpha}$
            $V_{\alpha min} = \phi$
        else if $\phi < \alpha_1$
            $V_{\alpha min} = p25_\alpha$
        else if $\phi > \alpha_2$
            $V_{\alpha min} = p45_\alpha$
return($[V_{\alpha min}, V_{\alpha max}]$)
```

FIGURE 25. Pseudo-Code for Angle, Case 2

define this case:

$$\begin{gathered} 0 \leq \alpha_1 \leq \alpha_2 \leq 90, \\ 180 \leq \alpha_3 \leq V_{2a} \leq \alpha_4 \leq 270, \\ 90 \leq (\alpha_3 - \alpha_2) \leq 180, \\ 90 \leq (\alpha_4 - \alpha_1) \leq 180 \end{gathered} \tag{II.49}$$

If any of the last two conditions ($90 \leq (\alpha_3-\alpha_2) \leq 180$ or $90 \leq (\alpha_4-\alpha_1) \leq 180$) is not satisfied, we can make a rotation and reduce the problem to case 2. A consequence of those two last conditions is that the maximum angle $\theta_{max} = 180$. An example of case 3 is shown in Figure 26.

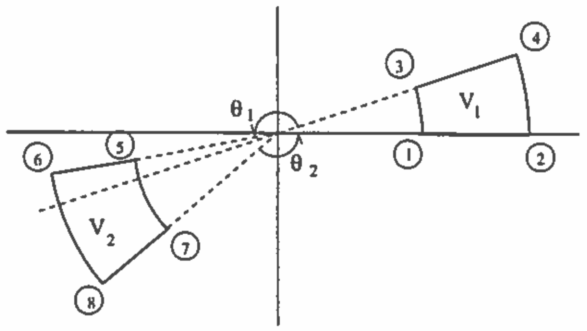


FIGURE 26. Fan Addition, Case 3

To compute the extremes of the magnitude, considering $90 \leq \theta_1 < 180$, we can compute V_{mmax} following the procedure for case 2, using the minimum of angles $\theta_1 = \alpha_3 - \alpha_2$ and $\theta_2 = \alpha_4 - \alpha_1$. V_{mmin} can also be computed using the procedure for case 2, with $\theta_{\max} = 180$. Figure 27 shows the algorithm to compute the magnitude for case 3.

For the determination of the extremes of the resulting angles, from equations II.24 to II.27, and using interval computation, the solution to equation II.27 turns to

$$0 \in x - y \cos \theta_{max} \tag{II.50}$$

Since $\theta_{max} = 180$, we distinguish three sub-cases here.

$$x + y \begin{cases} > 0 \\ \ni 0 \\ < 0 \end{cases} \tag{II.51}$$

```
MagnitudeCase3($V_1$, $V_2$)
    $\theta_1 = \alpha_3 - \alpha_2$
    $\theta_2 = 360 - (\alpha_4 - \alpha_1)$
    if $\theta_1 < \theta_2$
        $V_{mmax} = \max(p35_m, p36_m, p45_m, p46_m)$
    else
        $V_{mmax} = \max(p17_m, p18_m, p27_m, p28_m)$
    $I = V_{1m} \cap V_{2m}$
    if $I \neq \emptyset$
        $x_m = I_{min}$
        $y_m = I_{min}$
    else if $a > d$
        $x_m = a$
        $y_m = d$
    else if $b < c$
        $x_m = b$
        $y_m = c$
    $V_{mmin} = \sqrt{x_m^2 + y_m^2 - 2x_m y_m} = x_m - y_m$
return($[V_{mmin}, V_{mmax}]$)
```

FIGURE 27. Pseudo-Code for Magnitude, Case 3

In the first sub-case, the magnitude of one of the fans is large enough to *pull* the resultant to *its side* (the third sub-case is symmetrical to this one). In the second sub-case, the resulting fan has points scattered at all angles. So, the resulting angle can be anything, i.e. $[0, 360]$. Figure 28 illustrates these cases.

Angle correction for sub-cases 1 and 3 (and the angle computation) is similar to that performed in case 2, except that, in these cases, we might need to perform angle correction in both extremes. See Figure 29.

In summary, Figure 30 shows the pseudo-code for the computation of the angle for case 3.

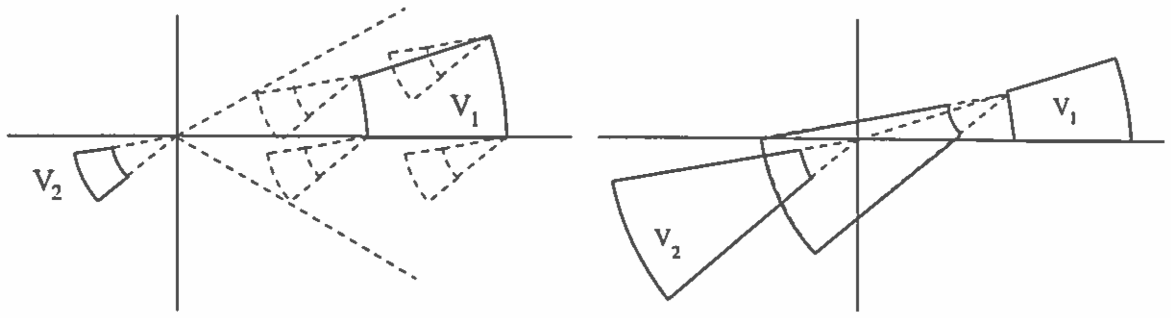


FIGURE 28. Fan Addition, Case 3

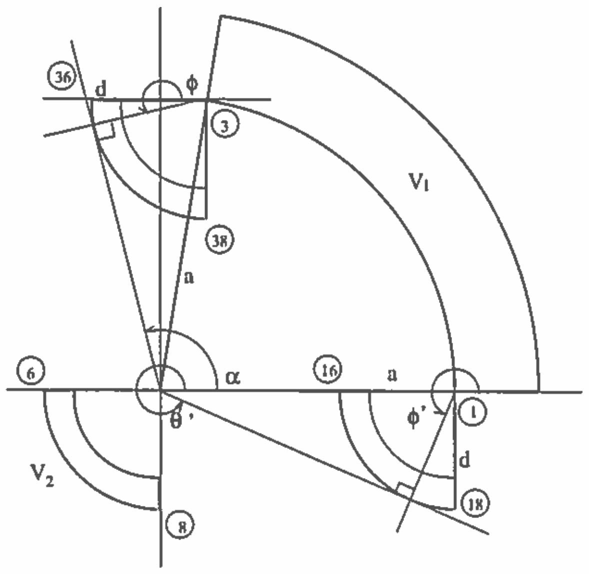


FIGURE 29. Angle Correction for Case 3

```
AngleCase3($V_1$, $V_2$)
    $\theta_{max} = 180$
    MagDiff $= V_{1m} - V_{2m} \cos \theta_{max}$
    if $0 \in$ MagDiff
        return($[0, 360]$)
    else if MagDiff $> 0$
        $V_{\alpha max} = p36_\alpha$
        if $V_{\alpha max} > 90$
            $\theta = \sin^{-1} \frac{d}{\alpha} + \alpha_2$
            $\phi = \theta + 90$
            if $\phi \in V_{2a}$
                $V_{\alpha max} = \theta$
            else if $\phi < \alpha_3$
                $V_{\alpha max} = p36_\alpha$
            else if $\phi > \alpha_4$
                $V_{\alpha max} = p38_\alpha$
        $V_{\alpha min} = p18_\alpha$
        if $V_{\alpha min} > 270$
            $\theta' = 360 - \sin^{-1} \frac{d}{\alpha}$
            $\phi' = \theta' - 90$
            if $\phi' \in V_{2a}$
                $V_{\alpha min} = \theta'$
            else if $\phi' < \alpha_3$
                $V_{\alpha min} = p16_\alpha$
            else if $\phi' > \alpha_4$
                $V_{\alpha min} = p18_\alpha$
    else
        ... case MagDiff< 0 is symmetrical
return($[V_{\alpha min}, V_{\alpha max}]$)
```

FIGURE 30. Pseudo-Code for Angle, Case 3

Completeness and minimality

Completeness and minimality are the two desired properties of our algorithm. Completeness refers to the fact that the solution fan V encloses all possible points in the operation $V_1 + V_2$. By minimality we understand that the result V cannot be reduced and still comprise all possible results.

Equation II.20 guarantees completeness if each partial result is complete. For each case we have shown completeness, since the resulting fan includes the minimum and maximum of magnitude and angle of all possible results of the complex fan addition. Therefore, all possible results are included in the answer.

Given that each case has been proven to guarantee completeness, all we need to guarantee completeness of the algorithm is an adequate definition for complex fan union. Let $V_1 = [a, b]\angle[\alpha_1, \alpha_2]$ and $V_2 = [c, d]\angle[\alpha_3, \alpha_4]$ be complex fans. We define complex fan union as follows:

$$V = V_1 \cup V_2 = [a, b] \cup [c, d]\angle[\alpha_1, \alpha_2] \cup [\alpha_3, \alpha_4]. \tag{II.52}$$

If we see a phasor as a pair (magnitude, angle), a complex fan is the set of all phasors in the Cartesian product of the magnitude and phase angle of that complex fan. Given that definition, it is clear that

$$([a, b] \cup [c, d]) \times ([\alpha_1, \alpha_2] \cup [\alpha_3, \alpha_4]) \supseteq ([a, b] \times [\alpha_1, \alpha_2]) \cup ([c, d] \times [\alpha_3, \alpha_4]). \tag{II.53}$$

so, complex fan union guarantees completeness and so does our algorithm.

Each partial case of complex fan addition guarantees that the partial result has at least one of the possible results on each of its boundaries. The union of two complex fans is defined as the independent union of their magnitudes and phase angles, and computed by interval union, which is set union. This makes sure that the final result still contains at least one point on each of its boundaries. Therefore, those partial results cannot be reduced without missing some of the possible results.

The preceding algorithm was derived considering all extremes of the fans as closed. If the fans have open extremes, we still have to do the computation as if they were closed. At the end, we have to take into consideration what points are producing the extreme of the intervals and verify whether those points have to be included in the result or not.

For example, in computing the maximum angle for the second case (see Figure 25), we use point $p36$ (in one of the cases), which is the result of adding points 2 and 6. We can say that a point is included in the fan if both boundaries are closed; in the case of point 3, if the minimum magnitude and maximum angle of fan V_1 are closed, point $p3$ is included. If both points, $p3$ and $p6$ are included in fans V_1 and V_2, respectively, the right extreme of the angle of the result will be closed.

There is an important issue regarding the evaluation of expressions involving more than one addition. The above results guarantee minimality with respect to a addition in the sense that there is no smaller fan that encloses all the results of the operation. Even if the result is expressed with the minimum fan possible, it contains, most of the time, spurious results. That is, there are regions that are enclosed by the resulting fan, and do not form part of the actual result. Figure 31 shows two complex fans, its real addition (a very irregular shape), and the computed complex fan. All points not in the irregular shape are spurious results, produced by complex fan addition.

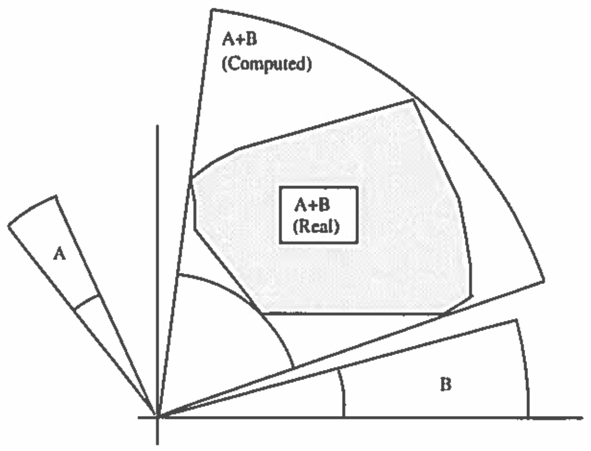


FIGURE 31. Spurious Results in Complex Fan Addition

In most cases, the extremes of the resulting case are computed using the corners of the addend fans, and that is precisely where the spurious result lie in the resulting fan of an addition. When the result of an addition is added to another fan, it is very likely that spurious behaviors lying in the corners will be used to compute the new result. This problem is caused by the uncertainty contained in the representation, which propagates through arithmetic operations. How much uncertainty is being generated depends on the order of evaluation. In other words, complex fan addition is commutative, but not associative. Figure 32 shows an example of how how the order of evaluation alters the result; we first define three complex fans more equidistant in angles; we evaluate $a + b + c$ in all possible ordering combinations. Figure 32 proves that the evaluation order may affect the results.

```
USER(957): (setq a (polar (interval [ 10 20 ])
                          (interval [ 10 20 ])))
([10.0000, 20.0000] L [10, 20])
USER(958): (setq b (polar (interval [ 10 20 ])
                          (interval [ 130 140 ])))
([10.0000, 20.0000] L [130, 140])
USER(959): (setq c (polar (interval [ 1 2 ])
                          (interval [ 250 260 ])))
([1.0000, 2.0000] L [250, 260])

USER(960): (setq abc (polar+ (polar+ a b) c))
([6.4524, 22.1928] L [28.8175, 121.1825])
USER(961): (setq acb (polar+ (polar+ a c) b))
([6.3229, 22.1141] L [26.3283, 117.1285])
USER(962): (setq bac (polar+ (polar+ b a) c))
([6.4524, 22.1928] L [28.8175, 121.1825])
USER(963): (setq bca (polar+ (polar+ b c) a))
([6.3229, 22.1141] L [32.8715, 123.6717])
USER(964): (setq cab (polar+ (polar+ c a) b))
([6.3229, 22.1141] L [26.3283, 117.1285])
USER(965): (setq cba (polar+ (polar+ c b) a))
([6.3229, 22.1141] L [32.8715, 123.6717])
```

FIGURE 32. Complex Fan Addition Is Not Associative

From our minimality results, we have that each evaluation contains all the possible results of addition $a + b + c$. Therefore, the intersection of all of them must as well enclose all the results, and it is at least as small as any of the results for the different evaluation orders. Taking the intersection in this example, reveals that none of the evaluation orders provides a minimal result. This indicates that there is not a straightforward method to provide an evaluation ordering which provides optimal results for addition expressions of complex fans. More work needs to be done in this direction.

Reasoning about change

In the previous subsections, we derived arithmetic operators for the complex fan domain. Those operators will be used in evaluating expressions while performing value propagation (see Chapter III). Another form of reasoning that we need to provide as part of the complex fan abstract data type is first order reasoning (i.e. reasoning about change).

The problem to solve in first order reasoning is to infer how a quantity changes as a result of the changes in other quantities. For instance, in Ohm's law, $V = ZI$, we know that if Z does not change and V increases, I increases, and we can say that the change in I was a result of the change in V. This notion is captured by DeKleer's confluences [12], which are a qualitative version of equations involving derivatives. For instance, the confluence equation for Ohm's law is $\partial V - \partial Z - \partial I = 0$, where $\partial X = \left[\frac{dX}{dt}\right]$, for any quantity X.

DeKleer's model deals with scalar quantities and their rate of change. Phasors, in the other hand, are vectors in two dimensions, and the confluence model needs to be extended to deal with them. A phasor has magnitude and angle, which can vary and can be affected by other phasors independently. A phasor derivative will be represented as a phasor $\partial X = (\partial X_m \angle \partial X_a)$, where the magnitude (∂X_m) and angle(∂X_a) represent the rate of change of magnitude and angle of the corresponding phasor X.

In the general case, we have a confluence involving two or more phasors and we need to derive how the changes in magnitude and phase angle of all but one phasor affect the remaining phasor. In our model we have equality, product, addition, and parallel confluences, and their inverse relations. Equality confluences arise from the fact that series currents and parallel voltages are equal (e.g. $I_S = I_1$). Product confluences arise from Ohm's law (see example in preceding paragraph). Addition confluences arise from the fact that voltages in series and currents in parallel add (i.e. $V_S = V_1 + V_2$); also, impedances in series add. Parallel confluences result from impedances connected in parallel (i.e. $Z = par(X,Y) = \frac{XY}{X+Y}$).

Equality

For equality constraints, it is obvious that changing the magnitude of one changes the magnitude of the other, without affecting the angle, and vice-versa.

$$\begin{aligned} \partial X &= \partial Y \\ \partial X_m &= \partial Y_m \\ \partial X_a &= \partial Y_a \end{aligned} \tag{II.54}$$

Product and Division

Let us consider the phasor product

$$\begin{aligned} Z &= XY \\ Z_m \angle Z_a &= (X_m Y_m) \angle (X_a + Y_a) \end{aligned} \tag{II.55}$$

It is clear that the magnitude of the result does not depend on the angle and vice-versa. So, one phasor product confluence can be seen as two independent confluences

$$\begin{aligned} \partial Z_m - \partial X_m - \partial Y_m = 0 \\ \partial Z_a - \partial X_a - \partial Y_a = 0 \end{aligned} \tag{II.56}$$

Division $X = Z/Y$ can be derived in the same way, or directly solved from the above confluences.

$$\begin{aligned} \partial X_m - \partial Z_m + \partial Y_m = 0 \\ \partial X_a - \partial Z_a + \partial Y_a = 0 \end{aligned} \tag{II.57}$$

Addition and Subtraction

As in the case of complex fan operations, complex fan confluence addition is considerably more complex. In this case, we see that changes in magnitudes and angles are not independent; changing a magnitude or angle can affect the magnitude and/or angle of the resultant. Those effects add, and can be computed independently. That is if we have $Z = X + Y$, to compute the final effect on ∂Z_m, we can add the influences of $\partial X_m, \partial X_a, \partial Y_m$, and ∂Y_a. This leads to the consideration of four cases of influences, magnitude to magnitude, magnitude to angle, angle to magnitude, and angle to angle. In all those cases, knowing how ∂X and ∂Y is not enough to compute ∂Z, we also need to know the values of X and Y, because the confluences change in different regions of the complex plane.

For the derivation of confluences in the four cases, consider the phasor addition $Z = X + Y$. First we consider magnitude to magnitude influences. In this case, our intuition misguides us. For example, if the magnitude of Y remains steady and the magnitude of X increases (no changes in the angles), we might think that the magnitude of Z must increase. That is correct for some values of X and Y, however, it is not always true. Figure 33 shows in the left the case where ∂Z_m decreases with ∂X_m, and on the right the case where ∂Z_m increases.

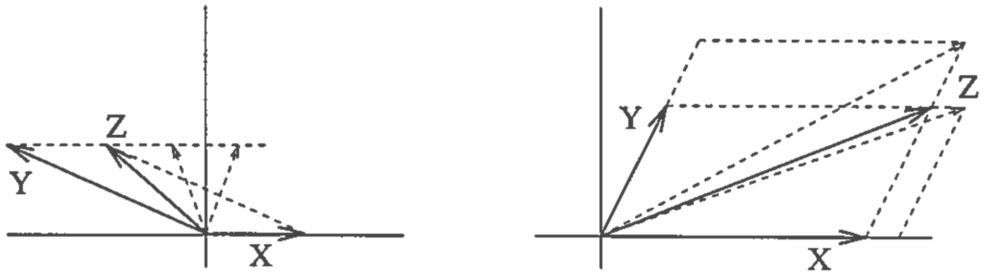


FIGURE 33. Magnitude to Magnitude Influence

There is a point that limits both regions. Since $\partial Z_m = -$ in the left region and $\partial Z_m = +$ on the right one, there must be a minimum for Z_m somewhere in between. The expression that relates magnitudes in phasor addition is $Z_m^2 = X_m^2 + Y_m^2 + 2X_mY_m \cos \alpha$, where $\alpha = Y_a - X_a$. Considering only the magnitude of X changes (i.e. $\partial Y_m = 0, \partial Y_a = 0$, and $\partial X_a = 0$), we can find the place where the derivative of

Z_m is zero,

$$\frac{dZ_m^2}{dt} = 2X_m\frac{dX_m}{dt} + 2Y_m\cos\alpha\frac{dX_m}{dt} = 0$$
$$\frac{dX_m}{dt}(2X_m + 2Y_m\cos\alpha) = 0 \tag{II.58}$$
$$\alpha = \cos^{-1}(-\frac{X_m}{Y_m}), X_m \leq Y_m$$

To compute how Y_m influences Z_m, we follow the same procedure, obtaining the condition,

$$\alpha = \cos^{-1}(-\frac{Y_m}{X_m}), Y_m \leq X_m \tag{II.59}$$

Let us consider complex fan X; X_m and X_a denote its magnitude and angle, and ∂X_m and ∂X_a the sign of the rates of change of its magnitude and angle, respectively. We will use the notation $I+(\partial Z_m, \partial X_m), I-(\partial Z_m, \partial X_m)$, and $I\pm(\partial Z_m, \partial X_m)$ to denote positive, negative and both influences of ∂X_m in ∂Z_m; same for other variables and subscripts. Figure 34 shows the pseudo-code to compute the magnitude to magnitude influences in phasor addition.

Now, we consider magnitude to angle influences. When the magnitude of one of the addends increases, it tends to "pull" the resultant to its side, influencing the angle towards it (e.g. right part of Figure 33). To prove it, we start from the expression for the angle of the resultant, taking X as reference ($X_a = 0$),

$$\begin{aligned} Z_a &= \tan^{-1}\left(\frac{Y_m\sin Y_a}{X_m + Y_m\cos Y_a}\right) \\ &= \tan^{-1}(e) \end{aligned} \tag{II.60}$$

```
Magnitude-to-Magnitude($\partial X_m, \partial Y_m, X_m, Y_m$)
    if $X_m < Y_m$
        $\alpha_c = \cos^{-1}(-Y_m/X_m)$
        if $\alpha > \alpha_c$
            $I - (\partial Z_m, \partial X_m)$
        else if $\alpha < \alpha_c$
            $I + (\partial Z_m, \partial X_m)$
        else
            $I \pm (\partial Z_m, \partial X_m)$
    else if $X_m > Y_m$
        $I + (\partial Z_m, \partial X_m)$
    else
        $I \pm (\partial Z_m, \partial X_m)$

    if $Y_m < X_m$
        $\alpha_c = \cos^{-1}(-X_m/Y_m)$
        if $\alpha > \alpha_c$
            $I - (\partial Z_m, \partial Y_m)$
        else if $\alpha < \alpha_c$
            $I + (\partial Z_m, \partial Y_m)$
        else
            $I \pm (\partial Z_m, \partial Y_m)$
    else if $Y_m > X_m$
        $I + (\partial Z_m, \partial Y_m)$
    else
        $I \pm (\partial Z_m, \partial Y_m)$
```

FIGURE 34. Pseudo-Code for Magnitude to Magnitude Influence

Since tangent is a monotonic function, it presents an extreme where its argument does (if there is one in its domain). So, assuming only the X_m changes,

$$\begin{aligned} \frac{de}{dt} &= \frac{-Y_m \sin Y_a \frac{dX_m}{dt}}{(X_m + Y_m \cos Y_a)^2} \\ -Y_m \sin Y_a \frac{dX_m}{dt} &= 0 \end{aligned} \tag{II.61}$$

Since Y_m cannot be zero, we find extremes only when $\sin Y_a = 0$ or $\sin Y_a = 180$, which is when both addends are parallel. In this case, a change in magnitude in the addends does not affect the angle of the resultant. The pseudo-code for this case is shown in Figure 35.

```
Magnitude-to-Angle($\partial X_m, \partial Y_m, X_m, Y_m$)
    if not parallel($X, Y$)
        $I - (\partial Z_a, \partial X_m)$
        $I + (\partial Z_a, \partial Y_m)$
```

FIGURE 35. Pseudo-Code for Magnitude to Angle Influence

Let us now consider the effect of a change in the angle of one addend to the magnitude of the resultant. Assuming the magnitudes do not change, the resultant will grow when the angle α decreases and vice-versa.

$$\begin{aligned} Z_m^2 &= X_m^2 + Y_m^2 + 2X_m Y_m \cos\alpha \\ \frac{dZ_m^2}{dt} &= -2XY \sin\alpha \\ \sin\alpha &= 0 \end{aligned} \tag{II.62}$$

The roots are found at $\alpha = 0$ and $\alpha = 180$. Since $\alpha = Y_a - X_a$, Z_a will decrease when Y_a decreases and when X_a increases, and vice-versa. When the angles are equal, any change in angles would decrease the magnitude of the resultant. When the angles are opposite, any change would increase the magnitude of the resultant. The pseudo-code for this case is shown in Figure 36,

```
Angle-to-Magnitude($\partial X_m, \partial Y_m, X_m, Y_m$)
    if not parallel($X, Y$)
        $I + (\partial Z_m, \partial X_a)$
        $I - (\partial Z_m, \partial Y_a)$
    else if $X_a = Y_a$
        $I - (\partial Z_m, -(|\partial X_a| + |\partial Y_a|))$
    else if $X_a = -Y_a$
        $I + (\partial Z_m, -(|\partial X_a| + |\partial Y_a|))$
```

FIGURE 36. Pseudo-Code for Angle to Magnitude Influence

Finally, we consider angle to angle influences. When the angle of Y changes, the resultant travels along the circle described by Y. For a region of Y_a's domain, increasing Y_a will decrease Z_a and for the rest, it will increase. See Figure 37.

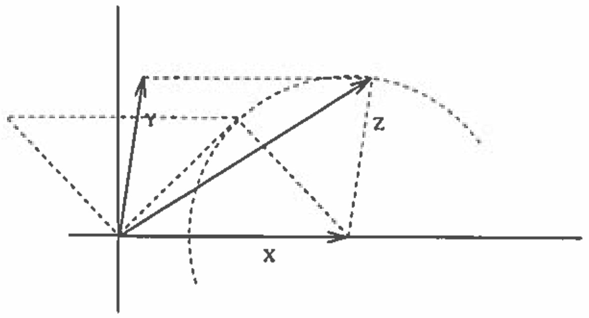


FIGURE 37. Angle to Angle Influence

Taking the derivative of equation II.60, considering everything constant except

Y_a, we have:

$$\frac{de}{dt} = \frac{(X_m + Y_m \cos Y_a) Y_m \cos Y_a \frac{dY_a}{dt} + Y_m \sin Y_a Y_m \sin Y_a \frac{dY_a}{dt}}{(X_m + Y_m \cos Y_a)^2}$$

$$Y_m \frac{dY_a}{dt}(X_m \cos Y_a + Y_m) = 0$$

$$X \cos Y_a + Y_m = 0$$

$$Y_a = \cos^{-1} - \frac{Y_m}{X_m}, Y_m \leq X_m \qquad \text{(II.63)}$$

A similar situation is present when analyzing the effects of ∂X_a in ∂Z_a. The pseudo-code for this part is shown in Figure 38.

Note that the pseudo-code presented in this section is in declarative form, indicating how changes in the addends affect the resultant. To compute the overall change in the resultant, we add all partial influences.

Phasor subtraction can be expressed in terms of addition, adding the minuend plus the negation of the subtrahend. When negating a phasor, the rates of change of its magnitude and angle are not affected. So, to compute confluence subtraction, we compute addition with the same rates of change and the negation of the subtrahend.

Parallel and Parallel Inverse

The only remaining constraint to be solved is the parallel confluence for impedances. Since the parallel constraint can be expressed in terms of product, addition, and division, all of which have been solved, we just compute partial results for the numerator and denominator, and then perform the division. Similarly, parallel inverse can be

```
Angle-to-Angle($\partial X_m, \partial Y_m, X_m, Y_m$)
    if $X_m < Y_m$
        $\alpha_c = \cos^{-1}(-Y_m/X_m)$
        if $\alpha > \alpha_c$
            $I - (\partial Z_a, \partial X_a)$
        else if $\alpha < \alpha_c$
            $I + (\partial Z_a, \partial X_a)$
        else
            $I \pm (\partial Z_a, \partial X_a)$
    else if $X_m > Y_m$
        $I + (\partial Z_a, \partial X_a)$
    else
        $I \pm (\partial Z_a, \partial X_a)$

    if $Y_m < X_m$
        $\alpha_c = \cos^{-1}(-X_m/Y_m)$
        if $\alpha > \alpha_c$
            $I - (\partial Z_a, \partial Y_a)$
        else if $\alpha < \alpha_c$
            $I + (\partial Z_a, \partial Y_a)$
        else
            $I \pm (\partial Z_a, \partial Y_a)$
    else if $Y_m > X_m$
        $I + (\partial Z_a, \partial Y_a)$
    else
        $I \pm (\partial Z_a, \partial Y_a)$
```

FIGURE 38. Pseudo-Code for Angle to Angle Influence

expressed in terms of parallel.

Related Work

Although there has been a considerable amount of work in the area of interval computation, no one has dealt with interval complex numbers expressed in polar form (i.e. complex fans). Alefeld [1] and Moore [35] provide very comprehensive texts on interval arithmetic, calculus, and interval-based numerical methods; the need to deal with complex numbers with interval attributes is mentioned, but restricted to rectangular form. Bandemer [2] gathered several papers on modeling uncertain data through interval computation, and none of them mention any work related to interval complex numbers represented in polar form (i.e. complex fans). Davis [7], in his article on constraint propagation with interval labels, presents a good survey, including expressions for complexity of constraint propagation, languages used at the time, and several examples. Still, none of the examples he presents include anything similar to the methods developed in this chapter. R. Baker Keafott [26] presents an implementation of an interval abstract data type. Our complex fan ADT relies on a complex interval ADT; their implementation is more general, since it is aimed for a more general use, our is more limited in the number of functions but it works with intervals with symbolic limits and considers closed and open extremes.

DeKleer and Brown [12] developed the confluence model, which allows reasoning about change and accounts for causality. Those concepts were developed to deal with scalar quantities. The concept of confluence has been extended here to deal with two dimensional quantities expressed in polar form (e.g. phasors).

In developing the algorithms for first order reasoning, we realized that the state of the involved variables may change the confluences, raising the need to deal with different models for different instances of the same problem. Other work [17, 31] has solved problems where devices exhibit different behaviors in different operating regions. In our case, the number of possible state combinations to be considered would be very large, and the system would need to be constantly switching from model to model as the state of the variables change. Instead, we developed the concept of conditional confluence, where the sing of the variables in a confluences depend on the state of the involved variables.

Chapter Conclusions

The first part of this chapter gives a brief introduction to the electrical engineering terms used in this dissertation, defining phasors and stating why they are important in the solution of linear circuits. The second part defines complex fans and derives algorithms for implementing the necessary arithmetic operations. We have proven that fan addition is complete but not sound. Nonetheless, if the information provided is precise, the algorithm converges to the same result as polar computation with real values for magnitude and angle, being therefore complete and sound.

We have also proven that the procedure to compute fan addition determines the smallest possible fan that contains all results of the addition. By developing this algorithm, we stay in the polar form representation, handling uncertainty for magnitude and angle values and generating the least number of spurious results.

To complete the work on complex fans, in the last part of this chapter, we presented a confluence model applicable to phasors. This is the first work on applying confluences and causal reasoning to non-scalar quantities. The form of the confluences changes depending on the state of the involved variables. We solved this problem by developing conditional confluences, which are confluences that are activated when a given condition is met.

In phasor confluences, changes in both magnitude and angle in one of the variables may affect the magnitude and/or variable of another variable; this results in a complex model of interactions. The way we solved the problem is by considering the different influences independently and adding their effects. The final result must, of course, be consistent with results for that variable computed through other paths in the constraint network.

In summary, we developed a representation for vectors with interval magnitude and angle attributes, providing the arithmetic operations needed to perform value propagation. Also, the confluence model was extended to cope with vectors expressed in polar form.

CHAPTER III

HYBRID REPRESENTATION CONSTRAINT PROPAGATION

Constraint propagation is an important inference engine for a variety of AI applications that reason about quantities. Many systems [12, 17, 28, 52, 45] represent the knowledge they reason about in terms of mathematical relations on qualitative or quantitative values of quantities.

As mentioned in the previous section, we model electrical circuits as a set of constraints and perform constraint propagation to infer as much (qualitative and/or quantitative) information about the circuit as possible. Thus, constraint propagation has to be implemented efficiently to allow our system to react in a reasonable time. The constraints that form the circuit model can be classified into disjoint sets; this allows us to implement specialized propagation algorithms, which are more efficient than general ones. This section presents the design and implementation of HRCP (Hybrid Representation Constraint Propagation), a constraint propagation engine that separates constraints into sets presenting different domains and types of constraints with different representations. All this, of course, must be transparent to the final user of the propagation engine, who sees the constraints in a uniform representation. A circuit can be modeled in terms of the Kirchoff and Ohm's laws that govern its behavior; these mathematical relations are normally recorded as algebraic constraints. Besides algebraic constraints, the user will provide the inference engine with additional information. In general, constraints can be algebraic expressions, qualitative or quantitative values for the involved variables, ordering and order of magnitude relations, confluences, etc.

Several observations led us to the current implementation. A constraint-based circuit model contains different types of constraints. While propagating ordering constraints, we do not need to use value information. Also, ordering constraints never compare quantities of different types; if we keep them separate, the problem size reduces. If we keep constraints of different kinds separate, one can carefully implement constraint propagation algorithms that perform efficiently for specialized sets. Worst case analysis of constraint propagation shows this to be a hard problem [7]. These facts suggest the separation of constraints into sets that do not interact with each other. By separating the constraints, we are applying the divide-and-conquer principle to reduce the complexity of the algorithm.

Constraint Propagation

The observation that constraints can be classified into different sets, with different properties, leads to a heterogeneous representation. If we can make such a classification and design a propagation algorithm for each set to be as efficient as possible, we will obtain better overall performance. Although the sets of constraints are disjoint, the sets are not totally independent. Some propagation procedures will need constraints from other sets to complete the propagation, and some sets will have (propagated) effects on other sets. Thus, we need to find a way to establish an order of propagation in the different sets, to record what sets have been modified, (i.e. need to be propagated), and when propagation has reached quiescence.

The idea to order propagation on sets of constraints has been taken from Waltz's algorithm [50], which is shown in Figure 39. Waltz's algorithm registers the con-

```
Propagation(constraints_to_be_propagated)
result ← empty
queue ← constraints_to_be_propagated
while not(empty(queue))
    constraint ← remove(queue)
    new_constraints ← propagate(constraint)
    insert new_constraints into result
    insert new_constraints into queue
return result
```

FIGURE 39. Waltz's Algorithm

straints to be propagated in a queue. It then enters a cycle until quiescence is reached. In the cycle, it removes a constraint from the queue, propagates its effects (when possible), determines what variables were affected and what constraints can be affected by those variables, and registers them as constraints to be propagated. Quiescence is reached when there are no more constraints to be propagated. In our case, we will have a set of constraints of some type that will be propagated by some algorithm and can result in new constraints being posted to different sets. Those constraints will produce subsequent changes in those sets and possibly in other sets. The idea is to register the constraint sets to be propagated in the same way Waltz's algorithm registers individual constraints. When a different constraint set is modified, it is added to the sets to be propagated. Quiescence is reached when there are no more sets to be propagated.

Instead of keeping track of the constraint sets to be propagated in a queue, we decided to propagate sequentially, in a round-robin fashion. In the beginning, all slots are marked as needing propagation. Each slot is visited in order; if the set needs to be propagated, the propagation function is called and the propagation flag reset. If a constraint set is visited and it is not marked, it is skipped. If constraints are posted to a constraint set, that set will be marked. The system reaches quiescence when it has visited all sets and finds no constraint sets to propagate. The result of each partial propagation is recorded and the union of all produced constraints is returned. This algorithm is shown in Figure 40.

Since we will have various implementations for our constraint sets, we need a system flexible enough to accept any implementation of a constraint set that uses its own propagator as defined by the *expert* (i.e. knowledge engineer). Assuming the set of constraints represent a model of the device we are reasoning about, we will call the whole set of constraints a model. A subset of the model will be called a *constraint set.* A model will have several slots to store each constraint set. The slots will be defined by the *expert*; a slot definition consists of a slot name, a class, an initialization function,

```
Global_propagation(model)
change ← true
result ← nil
while change
   change ← nil
   for each constr_set in model
      partial ← nil
      when marked(constr_set)
         partial ← partial_propagation(constr_set)
      append partial to result
      when partial
         for each constr in partial
            mark constraint_set of constr
         change ← true
return result
```

FIGURE 40. Global Propagation Algorithm

and matching templates. The *expert* is responsible for providing a class definition and an initialization function for the object that will store each constraint set. The matching templates will be used to determine the set where a new constraint will be placed. HRCP will take those definitions and create a device model, with all the slots initialized and ready to start receiving constraints.

In addition to the structure of each slot and its internal representation, the *expert* must provide a propagation function. This propagation function will be responsible for propagating effects of new constraints to its own and possibly to other constraint sets. We have implemented several representations for constraints and tested the system on some examples that will be discussed in section III.8.

Order of Magnitude Constraints

Ordering constraints impose a partial order on the quantity space. Examples of this kind of constraint are $A = B$, $A < B$, and $A > B$, where A and B are any two quantities that belong to the model. Given the constraints $A < B$ and $B < C$ the system should be able to infer $A < C$ (all valid inferences need to be sanctioned, of course). We know this problem is equivalent to that of computing the transitive closure of a labeled graph, where the vertices are variables and the edges represent relations among any two vertices. The Floyd-Warshall algorithm [6] can be used to solve this problem in time bounded by $O(n^3)$, where n is the number of variables involved in all constraints in the set. This algorithm is easier to code if the constraint set is represented as a matrix M, indexed by variable and where $M_{A,B} = r$ iff ArB for any A, B variables in the model. Figure 41 shows the algorithm, where n is the order of the matrix (i.e. number of variables), and T and T_{old} are temporal variables.

The main idea of the algorithm is to compute the relation between two variables i, and j, based on an auxiliary variable k. Returning to the example in the previous paragraph, let $i = A$, $j = B$, $k = C$, $T_{i,j} = <$ (i.e. $A < B$), and $T_{j,k} = <$ (i.e. $B < C$) at the time of the assignment statement in the algorithm. $AND(<,<) = <$ represents the intuitive knowledge $A < B,\ B < C \rightarrow A < C$. $OR(?,<) = <$ means that if we did not know anything about the relation between A and C, and we discover that $A < C$, we can consider the discovered relation a valid one.

```
Transitive_closure(M)
$T \leftarrow M$
For $k = 1$ to $n$
  $T[k,k] \leftarrow =$
For $k = 1$ to $n$
  $T\_old \leftarrow T$
  For $i = 1$ to $n$
    For $j = 1$ to $n$
      If $i \neq j$ then
        $T[i,j] \leftarrow$ OR($T\_old[i,j]$,
                        AND($T[i,k]$, $T[k,j]$))
Return $T$
```

FIGURE 41. The Floyd-Warshall Algorithm

The same algorithm can be used to represent Order of Magnitude (OM) constraints between variables. The modifications involve developing consistent semantics for the order of magnitude operators, and code the functions *AND* and *OR*. In our case, we have developed a semantics that does not correspond exactly to that of Mavrovouniotis and Stephanopoulous [33], but is very similar. The allowed operators are shown in Figure 42.

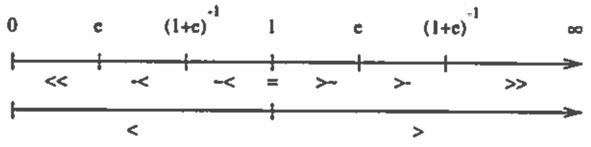


FIGURE 42. Order of Magnitude Operators

All operators have the intuitive semantics, expressed in [33]. $A = B$ has the standard meaning. $A \sim< B$ means A is slightly less than B; $A- < B$ means A is significantly less than B; $A \ll B$ means A is negligible with respect to B; $A < B$

means A is less than B, and represents the union of the three intervals between operators $\sim<$ and $<<$). The semantics we sanction for the AND and OR functions are expressed in tables 2 and 3. In those tables sections are separated by double lines; for each section, the first two columns are the operands and the third one the result. For instance, given $A \sim< B$ and $B \ll C$, using Table 2 (fourth section, row 1), we can infer $A \ll C$; if we previously knew that $A < C$, using Table 3 (section 2, row 1), we can update our knowledge to reflect that $A \ll C$. On the other hand, if we knew that $A \sim< C$, Table 3 (section 4, row 1), would detect that our new discovery is inconsistent with our previous knowledge. The inference rules in tables 2 and 3 are intuitively correct and allow the system to draw reasonable inferences.

Phase angle ordering constraints between phasor variables can also be represented in the same way; the same algorithm can be used for constraint propagation. The allowed constraints of this kind are Ahead, In_Phase, and Behind. These constraints are defined on the phase angle of phasors, whose domain is the interval $[0, 360]$. This domain has the particularity that is circular, i.e. $0 = 360$ and all angles are expressed modulo 360. We need to be a little careful here; we can say that a phasor A is ahead of B only if all values of A's angle are ahead of all values of B's angle by no more than 180 degrees. This definition made us not sanction situations like $A\ Ahead\ B$ and $B\ Ahead\ C \rightarrow A\ Ahead\ C$.

The table for the AND and OR functions for phase angle ordering constraints are shown in figures 4 and 5.

<<	<<	<<	<	<<	<<	−<	<<	<<	∼<	<<	<<
<<	<	<<	<	<	<	−<	<	<	∼<	<	<
<<	−<	<<	<	−<	<	−<	−<	<	∼<	−<	<
<<	∼<	<<	<	∼<	<	−<	∼<	<	∼<	∼<	<
<<	=	<<	<	=	<	−<	=	−<	∼<	=	∼<
<<	>∼	<	<	>∼	?	−<	>∼	<	∼<	>∼	?
<<	>−	<	<	>−	?	−<	>−	?	∼<	>−	>
<<	>	?	<	>	?	−<	>	?	∼<	>	?
<<	>>	?	<	>>	?	−<	>>	>	∼<	>>	>
<<	?	?	<	?	?	−<	?	?	∼<	?	?
=	<<	<<	>∼	<<	<	>−	<<	<	>	<<	?
=	<	<	>∼	<	?	>−	<	?	>	<	?
=	−<	−<	>∼	−<	<	>−	−<	?	>	−<	?
=	∼<	∼<	>∼	∼<	?	>−	∼<	>	>	∼<	?
=	=	=	>∼	=	>∼	>−	=	>−	>	=	>
=	>∼	>∼	>∼	>∼	>	>−	>∼	>	>	>∼	>
=	>−	>−	>∼	>−	>	>−	>−	>	>	>−	>
=	>	>	>∼	>	>	>−	>	>	>	>	>
=	>>	>>	>∼	>>	>>	>−	>>	>>	>	>>	>>
=	?	?	>∼	?	?	>−	?	?	>	?	?
			>>	<<	?	?	<<	?			
			>>	<	?	?	<	?			
			>>	−<	>	?	−<	?			
			>>	∼<	>	?	∼<	?			
			>>	=	>>	?	=	?			
			>>	>∼	>>	?	>∼	?			
			>>	>−	>>	?	>−	?			
			>>	>	>>	?	>	?			
			>>	>>	>>	?	>>	?			
			>>	?	?	?	?	?			

TABLE 2. AND Table for OM Constraints

<<	<<	<<	<	<<	<<	− <	<<	*fail*	∼<	<<	*fail*
<<	<	<<	<	<	<	− <	<	− <	∼<	<	∼<
<<	− <	*fail*	<	− <	− <	− <	− <	− <	∼<	− <	*fail*
<<	∼<	*fail*	<	∼<	∼<	− <	∼<	*fail*	∼<	∼<	∼<
<<	=	*fail*	<	=	*fail*	− <	=	*fail*	∼<	=	*fail*
<<	>∼	*fail*	<	>∼	*fail*	− <	>∼	*fail*	∼<	>∼	*fail*
<<	> −	*fail*	<	> −	*fail*	− <	> −	*fail*	∼<	> −	*fail*
<<	>	*fail*	<	>	*fail*	− <	>	*fail*	∼<	>	*fail*
<<	>>	*fail*	<	>>	*fail*	− <	>>	*fail*	∼<	>>	*fail*
<<	?	<<	<	?	<	− <	?	− <	∼<	?	∼<
=	<<	*fail*	>∼	<<	*fail*	> −	<<	*fail*	>	<<	*fail*
=	<	*fail*	>∼	<	*fail*	> −	<	*fail*	>	<	*fail*
=	− <	*fail*	>∼	− <	*fail*	> −	− <	*fail*	>	− <	*fail*
=	∼<	*fail*	>∼	∼<	*fail*	> −	∼<	*fail*	>	∼<	*fail*
=	=	=	>∼	=	*fail*	> −	=	*fail*	>	=	*fail*
=	>∼	*fail*	>∼	>∼	>∼	> −	>∼	*fail*	>	>∼	>∼
=	> −	*fail*	>∼	> −	*fail*	> −	> −	> −	>	> −	> −
=	>	*fail*	>∼	>	>∼	> −	>	> −	>	>	>
=	>>	*fail*	>∼	>>	*fail*	> −	>>	*fail*	>	>>	>>
=	?	=	>∼	?	>∼	> −	?	> −	>	?	>
			>>	<<	*fail*	?	<<	<<			
			>>	<	*fail*	?	<	<			
			>>	− <	*fail*	?	− <	− <			
			>>	∼<	*fail*	?	∼<	∼<			
			>>	=	*fail*	?	=	=			
			>>	>∼	*fail*	?	>∼	>∼			
			>>	> −	*fail*	?	> −	> −			
			>>	>	>>	?	>	>			
			>>	>>	>>	?	>>	>>			
			>>	?	>>	?	?	?			

TABLE 3. OR Table for OM Constraints

$ahead$	$ahead$	$?$	$in - phase$	$ahead$	$ahead$
$ahead$	$in - phase$	$ahead$	$in - phase$	$in - phase$	$in - phase$
$ahead$	$behind$	$?$	$in - phase$	$behind$	$behind$
$ahead$	$?$	$?$	$in - phase$	$?$	$?$
$behind$	$ahead$	$?$	$?$	$ahead$	$?$
$behind$	$in - phase$	$behind$	$?$	$in - phase$	$?$
$behind$	$behind$	$?$	$?$	$behind$	$?$
$behind$	$?$	$?$	$?$	$?$	$?$

TABLE 4. AND Table for Angle Order Constraints

$ahead$	$ahead$	$ahead$	$in - phase$	$ahead$	$fail$
$ahead$	$in - phase$	$fail$	$in - phase$	$in - phase$	$in - phase$
$ahead$	$behind$	$fail$	$in - phase$	$behind$	$fail$
$ahead$	$?$	$ahead$	$in - phase$	$?$	$in - phase$
$behind$	$ahead$	$fail$	$?$	$ahead$	$ahead$
$behind$	$in - phase$	$fail$	$?$	$in - phase$	$in - phase$
$behind$	$behind$	$behind$	$?$	$behind$	$behind$
$behind$	$?$	$behind$	$?$	$?$	$?$

TABLE 5. OR Table for Angle Order Constraints

Value Propagation

Value propagation (also known as label propagation [7]) is often needed, so we implemented it as well. Variables can take on qualitative signs, real intervals, or precise values for scalar magnitudes, or complex fans for phasor quantities (see section II.3). The sign of a number can be $-$, 0, or $+$. These qualitative values can be expressed as intervals (e.g. $+ = (0, \infty)$ notice the interval is open on both ends). A precise value is represented as a point interval (e.g. $5 = [5,5]$). So, we can express any value a variable takes on as an interval and have a uniform representation for the different levels of abstraction. By using this representation, we can mix information available at different levels of abstraction. In order to do that, we have developed an interval arithmetic package that can deal with qualitative values (e.g. ∞ in one of the limits). The qualitative interval package relies on a qualitative arithmetic package, which has an extensible structure. If any application needs to deal with other qualitative landmarks, they can be easily incorporated into the qualitative arithmetic and automatically handled by the interval package. So, if a variable can take on values from those domains, we can reason about devices at different levels of abstraction, from qualitative reasoning provided by sign algebra to quantitative reasoning like that provided in traditional numeric simulators. The more precise the information the user provides, the more accurate the answer to queries will be.

An important feature of value constraint propagation is its multi-directional nature. That is, if we have a constraint $A = B + C$, any of A, B, or C can be computed given values for the other two variables.[1] Generally, if we have a constraint involving n variables, we can propagate values only if one of the variables is undefined. This does not happen in the case of interval value propagation. That is, we could have the three values defined and still infer additional information from the constraint. For example, let $A = B + C$, $A = [0,\ 20]$, $B = [3,\ 5]$, and $C = [1,\ 20]$. Applying propagation, we could derive $A = [4,\ 20]$, and $C = [1,\ 15]$. Note that the computed interval for A was $[4,\ 25]$, but we took the intersection between that and the old value. An interval can be seen as another constraint; therefore, the computed value must be consistent with the old one. This guarantees that the intervals always shrink

[1]This feature is not present in imperative languages, where the order of computation is fixed by the program designer.

and that the propagation procedure will converge for linear functions [45].

Our implementation of value propagation maintains a list of variables, associating with each variable a value and a list of expressions in which it is found. For the previous example, A would contain the value $A = [0,\ 20]$, and the two expressions $B = A - C$, and $C = A - B$. When we discover a new value for A, we can efficiently determine what variables might be affected by that change. With each constraint set, we store the constraints that have not been propagated yet, i.e. the variables whose values have changed and need to be propagated. The propagation algorithm starts with the list of unpropagated variables; it takes one variable at a time, propagates its value, and place any modified values at the end of the list. The algorithm is shown in Figure 43.

```
Value_propagation(variables_to_be_propagated)
result ← empty
queue ← variables_to_be_propagated
while not(empty(queue))
   this_var ← remove(queue)
   new_variables ← propagate_variable(this_var)
   append new_variables to queue
   append new_variables to result
return result
```

FIGURE 43. Value Propagation Algorithm

First-Order Reasoning

Reasoning about change can be accomplished if we represent the model of a device in terms of *confluences* or qualitative differential equations. Confluences represent constraints about change [12]. For instance, for Ohm's law in a resistor $V_R = Z_R I_R$, we have the qualitative counterpart $\partial V_R - \partial Z_R - \partial I_R = 0$, where ∂X represents the sign of the derivative of variable X with respect to time (i.e. $\partial X = [dX/dt]$). This confluence indicates, for example, that if Z_R decreases and V_R does not change, I_R increases. The values ∂X can take on, for any variable X, are signs. Therefore, ∂X can be considered as a simple variable, and confluences can be viewed like any other algebraic constraint. Given this, the same representation and propagation scheme used in value propagation can be used in first order reasoning.

In Chapter II the concept of confluence was extended to deal with phasors. In the case of phasor confluences, the form of the confluence (i.e., the sign of the terms in a confluence) depends on the value of the variables involved in the confluence. A phasor is a complex quantity, represented by a couple containing magnitude and angle. In a phasor variable, each component (magnitude or angle) of the involved variables can vary independently. In the cases of product and division, magnitude affects magnitude, and angle affects angle, and the expressions to compute the resulting effects are simple. In addition, magnitude and angle can be affected independently; to compute the final change in each component of the result variable we have to consider how the magnitude and angle of the other two variables influence the magnitude and angle of the result variable. See Chapter II

Mixed Propagation

Another type of inference needed in our application domain is the discovery of order of magnitude relations from numeric values (reals or intervals) and vice-versa. For instance, from $A = 0.0001$, and $B = 5000$, we can conclude $A \ll B$. Our implementation follows the semantics expressed in [33]. These semantics express a relation $A \ll B$ in terms of the relative value of one variable to the other (i.e. the ratio of the variables). How big or small can the ratio of the two variables be, so that we can assert a given relation? For example, $A \ll B$ iff $A/B < e$, for some small e. This leads to the definition of the different OM relational operators as intervals on the ratio of the two related quantities. Figure 42 (page 78) shows the different operators and the intervals on the ratios they represent. The parameter e can be defined independently for each constraint set.

For any two given numbers, it is easy to verify if they satisfy a given relationship, but in the presence of intervals, the discovery of these relations can be tricky. To be able to assert an order of magnitude relation between two variables we need to make sure that the relation is preserved for any allowed value. As an example, consider $A = [1,\ 10]$, and $B = [5,\ 5000]$. If $A = 1$, and $B = 5000$, they satisfy the relation $A \ll B$, but if $A = 10$, and $B = 5$, $B < A$. Therefore we cannot say anything about the relation between those variables. If, on the other hand, $A = [1,\ 10]$, and

$B = [15,\ 5000]$, we can at least say that $A < B$. In that case, if we want to find all possible relations between any two variables, every time a variable is updated it must be compared with the rest of the variables.

The phasor domain is not an ordered set, therefore a relation like $A < B$ makes no sense if A and B are polar variables. The real semantics of $A < B$ in that case is magnitude(A) < magnitude(B). Similarly, we have other ordering operators (Ahead, In_Phase, and Behind) that relate angles of polar quantities. So, A In_Phase B really means angle(A) In_Phase angle(B).

In the discovery of phase angle relations, phase angle ordering operators have different properties than OM operators. For instance, the condition for an angle A to be *Ahead* of an angle B, is related to addition, not to product. That is, A *Ahead* B iff $A - B \in [0, 180]$.

Inferences can be drawn in the other direction, too. If you have two variables, an OM constraint that involves those two variables can *refine* their values. For instance, $A = [15, 900]$, $B = [15, 1500]$, and the user asserts $A \ll B$ (interval($\ll$) = $[0, 0.05]$). The largest value of A which is much less than the largest value of B is 75, and the smallest value of B which is much greater than the smallest value of A is 300. So, we can reduce the intervals to $A = [300, 1900]$ and $B = [15, 75]$ (see Figure 44).

In general, if $A = [a,\ b]$, $B = [c,\ d]$, $A\ op\ B$ and $interval(op) = [op_l,\ op_r]$, A can be refined as indicated in equation III.64 (an analogous expression can be derived

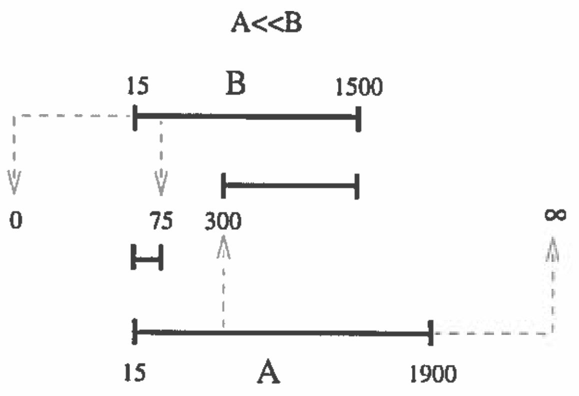


FIGURE 44. Value Refinement from OM relations

for B).

$$\begin{aligned}
\tfrac{A}{B} &\subseteq interval(op) \\
A &\subseteq interval(op)B \\
[a,\ b] &\subseteq [op_l,\ op_r][c,\ d] \\
A_{new} &= [\max(a, c\ op_l),\ \min(b, d\ op_r)]
\end{aligned} \tag{III.64}$$

Propagation Across Algebraic Constraints

Besides computing the transitive closure of OM relations, we sometimes need to propagate them across algebraic expressions. For instance, for a parallel cluster with a resistor and a capacitor we have the constraints $V_R = Z_R I_R$, $V_C = Z_C I_C$, $V_R = V_C$. If we add $Z_R \ll Z_C$, HRCP must be able to infer $I_R \gg I_C$. Pairs of equations related by a third one, like in the precious example, will be called equivalent equations. To perform this type of inference in an efficient manner, we need to keep track of all the pairs of equivalent equations, as they are introduced into the model. These pairs are associated with each variable on the right hand sides of both equations, such that when a new ordering relation involving one of those variables is discovered, equations are at hand and the new computation can be done efficiently. This procedure will generate new OM relations to be propagated later, using the transitive closure algorithm.

User Interface

HRCP has two kinds of users: *the expert* (i.e. a knowledge engineer), who defines what sets of constraints are available, their semantics, and the propagation and verification functions for each set of constraints; and *the client*, who uses the reasoning engine as a tool in the AI application. HRCP is a flexible and extensible reasoning engine that allows the *expert* to define how the set of constraints is to be partitioned into disjoint sets. All this manipulation is transparent to the *client*, who sees a uniform representation for all constraints.

To define new constraint sets, the *expert* must provide a definition of the class that represents each constraint set, and a function that creates such an object. Besides the class definition, the *expert* has to provide function definitions for insertion, consistency checking, and propagation of constraints.

For the *client*, the functions HRCP provides are *assert*, *verify*, and *consistent?*. *Assert* includes a constraint (or list of constraints) in the model and then runs propagation, and returns the result of propagation: a list of constraints that are consequences of the new constraint plus the previous model. If the constraint is inconsistent, an error message is returned, indicating what constraints are contradicted.

Verify checks if a constraint is included in the model. Since the inference process takes place at the moment of asserting a constraint, this step reduces to checking whether the constraint is in the respective constraint set of the model. *Consistent?* verifies that a constraint is consistent with the model, without actually inserting it. The constraint is inserted in a copy of the model; if any inconsistency arises, we can answer no. If not, the answer is yes.

For *assert*, first a copy of the actual model is made, and the constraint is then inserted in the copy. If a contradiction is found it is reported to the user and the model remains unaltered. Otherwise, the consequences of the insertion are returned to the user and the model is updated.

There are two desirable features in a constraint propagator: one is the ability to retract a constraint, and the other is the ability to explain the effects of the assertion of a constraint. Both problems involve keeping track of the chain of effects the assertion of a given constraint has; perhaps using a mechanism similar to the ones used to maintain dependencies in ATM's [11, 34]. This process is expensive in terms of memory and time, and would also complicate and obscure the code. Instead of implementing this bookkeeping mechanism, we chose to implement a history of the events, with the respective copies of the model, as it evolves with those events. If we keep track of that information, we can undo the effects of an assertion, by just retrieving the copy of the model before that assertion happened.

To obtain the second feature, we implemented a limited explanation facility, based on the trace of the propagation process. While propagation is taking place, every time a variable is updated, the new value is recorded, together with the cause (i.e. what variable produced the change through what constraint). When propagation finishes, the user can ask HRCP to explain how a variable got its final value. HRCP analyses the record of changes to variables and responds with the path of latest update to the enquired-about variable. This produces an explanation of the events occurring in the device.

Examples

HRCP has been tested on a number of circuits. In this section we present examples that show how adding more and more constraints to the circuit model changes the state of the circuit, reducing the number of possible behaviors the circuit can exhibit under those conditions. Other examples show that HRCP can indeed derive the kind of knowledge we will use in the different reasoning tasks proposed in this dissertation. The circuit we will refer to is the one shown in Figure 1, repeated here as Figure 45. When the circuit topology is provided to QPA, the circuit analyzer

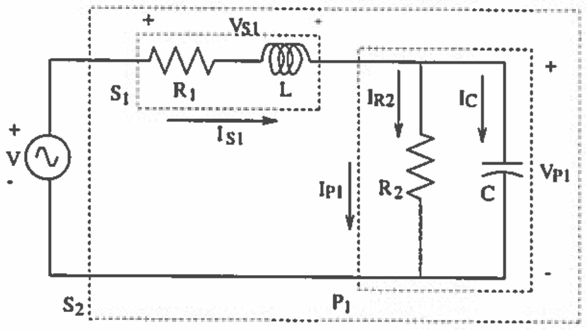


FIGURE 45. An Electrical Circuit

(see Chapter IV) forms the BSOC, i.e. a constraint-based circuit model. The user then interacts with HRCP, adding more constraints, until the analysis task has been completed or an inconsistency is found.

Example 1

This example shows how HRCP allows us to do circuit analysis at different levels of abstraction. Intervals, our general data representation, allow the circuit parameters and variables to take on values that range from qualitative to quantitative, passing through intervals. Qualitative values are represented as intervals with (possibly) infinite extremes, i.e. $- = [-\infty, 0], 0 = (0, 0), and+ = [0, \infty]$. Intervals with finite limits allow us to express different levels of uncertainty, depending on their lengths. Real values can be expressed as point intervals, e.g. $5 = [5, 5]$.

We first assert a set of constraints that assign interval values to parameters and/or variables of the circuit. We also assign real values to parameters and variables; for instance ω, the frequency of the system, was assigned the value 60, and V_{S2}, the voltage of the series cluster S_2, was assigned the value 100. Figure 46 shows the constraints and important details of the results. Since plotting the complex fans of all variables in the circuit gets complicated, we are focusing on variables I_C, I_{R2}, and I_{P1}. Figure 47 shows how the complex fans for those variables reduce as we reduce the intervals of the input parameters, until they converge to a single point, when all parameters are reals.

```
Asserting:
((variables (value W 60))                    (variables (value VS2 100))
 (variables (value L [0.5, 1.5 ]))           (variables (value R1 [15, 20 ]))
 (variables (value C [0.002, 0.003 ]))       (variables (value R2 [20, 30 ])))

Result:
((variables (value VV 100))                  (variables (value ZL ([6, 120] ∠ 90)))
 (variables (value ZR1 [5, 25]))             ...
 (variables (value VL ([60.238, 191.722] ∠ [10.247, 54.149])))
 ...
 (variables (value IC ([0.474, 6.249] ∠ [290.739, 346.769])))
 (variables (value IR2 ([0.131, 2.603] ∠ [200.739, 256.769])))
 (variables (value IP1 ([1.107, 4.425] ∠ [280.247, 324.149])))
 ...)

Asserting:
((variables (value L [0.9, 1.1]))            (variables (value R1 [16.9, 17.1]))
 (variables (value C [0.0024, 0.0026]))      (variables (value R2 [24.9, 25.1])))

Result:
( (variables (value IC ([1.485, 1.953] ∠ [301.371, 307.320])))
 (variables (value IR2 ([0.386, 0.536] ∠ [211.371, 217.320])))
 (variables (value IP1 ([1.583, 1.970] ∠ [287.044, 291.736])))
 ...)

Asserting:
((variables (value L 1))                     (variables (value R1 17))
 (variables (value C 0.0025))                (variables (value R2 25)))

Result:
( (variables (value IC (1.697 ∠ 304.067)))
 (variables (value IR2 (0.452 ∠ 214.067)))
 (variables (value IP1 (1.756 ∠ 289.136)))
 ...)
```

FIGURE 46. Example 1. Reducing the Number of Possible Behaviors

Notice that, as opposed to a regular circuit solver, constraints allow all variables to be seen as input or output variables, without any previous declaration or modification of the model. For instance, in example 1, the input is V_{S2}, which is a voltage variable not an input parameter. Based on the value of V_{S2}, the program computes the value for the voltage of the source, seen by circuit solvers as the input parameter.

Example 2

This example shows how the inclusion of an order of magnitude constraint can refine the values of the variables involved. Figure 48 asserts the same values as the above example, only this time the constraints $V_L < V_{S2}$ and $75 Ahead I_{P1}$ were included. Note that the inclusion of the constraint $V_{P1} < V_{R1}$ also generates other OM constraints. The second constraint does not allow I_{P1}'s angle to take on values that exceed 75, i.e. I_{P1} has to be Behind 75. Since our constraint language does not allow inclusion of constants in constraint expressions, we had to create a dummy variable. The effects of those constraints on the involved variables modify their values and those changes are propagated through the constraint network. You can see how the variables for those variables changed relative to example 1.

Example 3

In the event that the value of a variable becomes the empty set, or that by propagating order of magnitude constraints we get to a contradiction (e.g. $A > B$

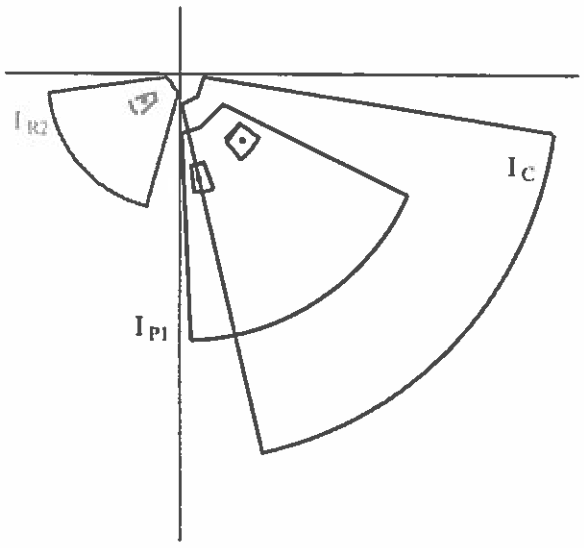


FIGURE 47. Graphical Representation of Complex Fans for Example 1

```
Asserting:
((variables (value W 60))                              (variables (value VS2 100))
  (variables (value L [0.5, 1.5]))                     (variables (value R1 [15, 20]))
  (variables (value C [0.002, 0.003]))                 (variables (value R2 [20, 30]))
  (variables (value IDUM (10 ∠ 75)))
  (Ahead IDUM IP1)                                     (< VP1 VR1))

Result:
((< VC VR1)                                            (< VR2 VR1)
  (variables (value VC ([68.404, 100) ∠ [10.247, 48.728])))
  (variables (value VP1 ([ 3.953, 52.077] ∠ [200.739, 256.769])))
  (variables (value VR1 ([16.617, 88.507] ∠ [280.247, 324.149])))

  (variables (value IC ([ 0.474, 6.249] ∠ [290.739, 346.769])))
  (variables (value IR2 ([ 0.131, 2.603] ∠ [200.739, 256.769])))
  (variables (value IP1 ([ 1.107, 4.425] ∠ [280.247, 324.149])))
  ...)
```

FIGURE 48. Example 2. Value Refinement by Order of Magnitude Constraints

```
Asserting:
((variables (value W 60))              (variables (value VS2 100))
 (variables (value L 1))               (variables (value R1 17))
 (variables (value C 0.0025))          (variables (value R2 25)))

Result:
Inconsistency:
    Type: Values violate OM constraint
    Executing: Value propagation
    What: ((value VP1 (11.316 ∠ 214.067))
                (value VR2 ([105.408, ∞) ∠ [0.0d0, 360.0d0])))
    Contradicts: (= VP1 VR2)
```

FIGURE 49. Example 3. Detecting Inconsistencies

and $B < A$), the program does not insert the given constraints and the state of the circuit remains untouched. Figure 49 shows an example where the values provided by the user are inconsistent. HRCP computes values for the variables V_{P1} and V_{R2} which contradict with the constraint $V_{P1} = V_{R2}$ (derived from the fact that R_2 is an element of the parallel cluster P_2, therefore, they exhibit the same voltage).

Example 4

Figure 50 illustrates how HRCP discovers order of magnitude constraints based on the values of the variables.

Constraint $ZL >\!\!- ZR1$ indicates that ZL is greater than $ZR1$, but they are not really close together nor very far apart. On the other hand, the constraint $ZP1 < ZS2$ is indicating that there are values for $ZP1$ and $ZS2$ that can make $ZP1 \ll ZS2$ or

```
Asserting:
((variables (value W 60))                 (variables (value VS2 100))
 (variables (value L [0.5, 1.5]))         (variables (value R1 [15, 20]))
 (variables (value C [0.002, 0.003]))     (variables (value R2 [20, 30])))

Result:
((variables (value ZL ([30, 90] ∠ 90)))
 (variables (value ZR1 [15, 20]))
 (variables (value ZP1 ([3.568, 12.043] ∠ [280.491, 292.619])))
 (variables (value ZS2 ([22.59, 90.264] ∠ [ 35.850, 79.752])))
 (variables (value IR2 ([0.131, 2.6039] ∠ [200.739, 256.769])))
 (variables (value IS1 ([1.107, 4.4253] ∠ [280.247, 324.149])))
 (variables (value IC ([0.474, 6.2493] ∠ [290.739, 346.769])))

 (> − ZL ZR1)                             (< ZP1 ZS2)
 (Behind IR2 IS1)                         (Ahead IC IR2)
 ··· )
```

FIGURE 50. Example 4. Discovering OM Constraints from Value Constraints

$ZP1- < ZS2$ valid. Only the more general constraint $ZP1 < ZS2$ can be asserted, because it hold for all values in $ZP1$ and $ZS2$.

The reader can also verify the angle constraints discovered from the values displayed above. E.g., the maximum value that $IR2$'s angle can take on is 256.769, while the minimum possible angle of $IS1$ is 280.247, which indicates that $IR2$ will always be behind $IS1$.

Example 5

Another important kind of reasoning that QPA needs to exhibit is first order reasoning. As explained in preceding sections, reasoning about change can be implemented in terms of value propagation, where the variables represent derivatives and take on qualitative (sign) values. Figure 51 shows the response of the system to the query "what happens if the value of C increases?" (assuming the rest will remain without change). Note that in Figure 51, $(P\ V)$ stands for $\partial\ V$.

```
Asserting constraints:
((confluences (value (P VV) (0 L 0)))
 (confluences (value (P R1) (0 L 0)))
 (confluences (value (P L) (0 L 0)))
 (confluences (value (P R2) (0 L 0)))
 (confluences (value (P C) (+ L 0))))

Result:
((confluences (value (P VL) (+ L ?)))
 (confluences (value (P VR1) (+ L ?)))
 (confluences (value (P IL) (+ L ?)))
 (confluences (value (P IR1) (+ L ?)))
 (confluences (value (P VS1) (+ L ?)))
 (confluences (value (P IV) (+ L ?)))
 (confluences (value (P IP1) (+ L ?)))
 ...)
```

FIGURE 51. Example 5. First Order Reasoning

In most cases, a simple change in a variable's rate produces many paths of ramifications, and HRCP just returns the set of all consequences of the asserted constraint. To help the user understand those results and separate the different causal paths, we have developed a simple explanation mechanism. Figure 52 shows the explanation of how the change in C affects $\partial\ I_{P1}$.

The program output can be paraphrased as follows:

"When C increases, Z_C decreases, causing Z_{P1} to decrease, which in turn

```
Explaining variable (P IP1)
((P IP1) (+ L ?) (P IS2)
 (confluences (== (P IP1) (P IS2))))
((P IS2) (+ L ?) (P ZS2)
 (confluences (== (P IS2) (DERIV/ (P VS2) (P ZS2)))))
((P ZS2) (- L ?) (P ZP1)
 (confluences (== (P ZS2)
                  (DERIV+ (P ZS2) (P ZS1) (P ZP1)
                          ZS2 ZS1 ZP1))))
((P ZP1) (- L -) (P ZC)
 (confluences (== (P ZP1)
                  (DERIV-PARALLEL (P ZP1) (P ZR2) (P ZC)
                                  ZP1 ZR2 ZC))))
((P ZC) (- L 0) (P C)
 (confluences (== (P ZC) (DERIV-NEGATE (P C))))
 User provided: (P C) = (+ L 0)
```

FIGURE 52. Explanation for Example 5

causes Z_{S2} to decrease. The decrease in Z_{S2} causes I_{S2} to increase, causing I_{P1} to increase as well, since they are the same current."

Related Work

This chapter presented HRCP, an extensible reasoning engine that performs constraint propagation. This reasoning engine is the main inference engine behind the programs that reason about electrical circuits. In this section, we contrast our work with the work in the fields of constraint programming and interval constraint propagation.

In the area of constraint programming [29], and constraint logic programming [5], several languages have been developed, to solve problems expressed as a set of constraints. Screamer [40], CLP(R) [25], and Ilog [37] are examples of such languages. These languages are more geared toward discrete constraint satisfaction problems (CSP). In a CSP you have a set of constraints on variables with discrete finite domains. The problem is to find an assignment to each variable such that all constraints are satisfied. In contrast to CSP, our problem domain does not require generating a particular solution to a given problem; determining the sets of possible values the variables can take on is what is needed. For a completely specified problem, the precise (single) solution will be generated, though. Since we are not performing backtracking, the main feature of constraint programming languages would be of no use at all to solve our problem. Besides, we would need to program our own propagators to deal with our data types, specially complex fans. That is why we decided to implement HRCP instead of using an existing system.

HRCP can be characterized as an "anytime algorithm", as defined in [48]; that is, the solution improves with time. As intervals and fans can only decrease in size, the solution can only get better, until the system reaches quiescence.

Efficiency of constraint propagation depends on the implementation of the particular propagators for each constraint set. For the case of order relations, we have implemented the Floyd-Warshall algorithm, which has time complexity $O(n^3)$. Our implementation of value propagation is comparable to AC-3 [47], where only constraints affected by variables whose values have changed are revisited.

There are other systems that perform constraint propagation over interval values. Davis [8] provides a complete study on constraint propagation with interval or sign values. In his paper, Davis includes examples of applications to several domains, characteristics of different constraint languages, their strengths and weaknesses, and a complexity analysis for various cases. TMM [13] keeps a set of events, denoting particular instants of time. The time at which a given event happens, or the difference between events are bounden by intervals. Constraint propagation is used to infer as much as possible about time relationships between events. CLP(intervals) [3] implements a constraint logic programming language on interval domains. The authors of CLP(R) focus on the application of the Newton interval method to solve systems of non-linear equations; they are solving a different problem, our main concern is just constraint propagation with values in the interval and complex fan domains. Hyvönen [24] presents a numerical constraint propagation system that works with intervals. He deals with closed intervals only; in our problem, we need to represent intervals with open ends, such that $-\infty$, 0, and ∞ are never part of a variable's domain. Ward et al [51] present a formalism to perform constraint propagation with interval values, and applications to mechanical design. Although many people have worked on constraint propagation before, their implementations are not readily available, so we could not really evaluate if their work would have been useful to solve our problem or not.

Chapter Conclusions

Constraint languages allow us to model a device by a set of constraints. The device model normally contains constraints of different kinds, which can be partitioned into several sets with (possibly) different representations. We take advantage of this fact to develop a framework for constraint propagation in which the user can define how to partition the constraints in the model. Different representations allow us to take advantage of the characteristics of each constraint set and to implement a special propagation algorithm for each set.

A higher-level constraint propagation engine is provided, which ensures the completeness of the propagation process. HRCP's implementation is, in essence, an eager reasoning mechanism. That is, as soon as the user enters a constraint, all its consequences are computed. So, when the user enquires about a constraint, all the system has to do is verify whether it belongs to the model. The system provides a transparent representation for the client user, who sees an engine capable of recording constraints in a declarative fashion, propagating them, and preserving consistency.

In this domain of application, we encounter several kinds of constraints: ordering constraints, value constraints, value-difference constraints, confluences, etc. The system accepts all data in a uniform format (i.e. a constraint is expressed in prefix form).

Our representation allows variables to have sets of values represented as contiguous intervals (or complex fans, in the case of phasors). This constrains the kind of constraints we can handle; for instance, we allow equality, but not inequality, since propagation across inequality constraints can produce non-contiguous intervals (i.e. intervals with holes). The value propagation mechanism is general and independent of the domain of the variables in it. That is, the same mechanism is used to propagate polar values for circuit variables as to propagate qualitative sign values for first order reasoning.

In Section II.3 we mentioned the way uncertainty propagates through complex fan addition, making them larger than the real result. We showed that complex fan addition is commutative, but not associative; that is, when we have more than two operations, the result depends on the order of execution. If we had all operations in a single expression, we could try to find an optimal evaluation order, and reduce the number of spurious results. In constraint propagation the operations need to be evaluated when a constraint is being propagated. We do not have all the operations, therefore we cannot change the order of evaluation. In performing value propagation, complex fans are seen as constraints. If we compute a larger complex fan as a result of propagating a constraint, this result will represent a weaker constraint, and when intersected with the old value of the variable the recently introduced spurious results will not contribute to make it grow (i.e. complex fans will never grow). This property stops the propagation of uncertainty, which would grow unwieldly otherwise. Still, the final values will contain spurious results, but they will not diverge due to uncertainty.

Ernest Davis presents a complete discussion on complexity of constraint propagation with interval labels [8]. In his paper, Davis shows that propagation on linear constraints with unit coefficients always quiesces if the starting state is consistent. In our case all constraints are additions and products with unitary coefficients. A problem arises in special cases, where propagation of one constraint iterates through successively more refined intervals. Figure 53 shows the constraint, the initial values for the variables, and part of the trace where the iteration takes place. This prob-

```
    ...
    Propagating variable ZP1
     evaluating : (== ZS1 (- ZS2 ZP1))
                  (- ([9.782, INFINITY} L [287.612, 43.788])
                     ([5.901, 8.300] L [287.612, 292.619]))
     ZS1=([1.481, INFINITY} L [39.266, 43.788])
    inserting ZS1 in queue
    Propagating variable ZS2
    Propagating variable ZS1
     evaluating : (== ZS2 (+ ZS1 ZP1))
                  (+ ([1.481, INFINITY} L [39.266, 43.788])
                     ([5.901, 8.300] L [287.612, 292.619]))
     ZS2=([9.782, INFINITY} L [297.478, 43.788])
    inserting ZS2 in queue
    Propagating variable ZS2
     evaluating : (== ZS1 (- ZS2 ZP1))
                  (- ([9.782, INFINITY} L [297.478, 43.788])
                     ([5.901, 8.300] L [287.612, 292.619]))
     ZS1=([1.667, INFINITY} L [39.266, 43.788])
    inserting ZS1 in queue
    Propagating variable ZS1
    ...
```

FIGURE 53. Iteration Loop in Value Propagation

lem arises because non-linear functions (i.e. trigonometric functions) are involved in the computation of fan addition. Since the arithmetic is being done at a given degree of accuracy ϵ, either the values change at least that amount, or our comparison function indicates no change, in which case iteration terminates. Let us say X is involved in a constraint that is iterating, and only its maximum limit X_r is decreasing from its initial value X_{ro}, tending to X_{rf}. The maximum number of iterations will be $(X_{ro} - X_{rf})/\epsilon$. If both limits are changing, the maximum number of iterations for a given constraint is the maximum number for both limits. Even if the initial value is $(X_{ro} = \infty$, the process always terminates in a finite number of iterations; in this case, the first reduction takes the infinite value to a finite one, and the case reduces to the previous one. Davis also mentions the FIFO heuristic we are using to select constraints to be propagated in Waltz's algorithm is much better than LIFO or best-first, in which case the number of evaluations (interval refinements) can grow exponentially.

The implementation of the algorithm makes constraint propagation usable in both on-line and off-line schemes. The main function of this inference engine is to provide the expert user with a scheme to easily and rapidly implement efficient constraint propagators to suit the application needs. On the other hand, the client user sees a transparent interface, with a uniform representation.

The flexibility that HRCP offers to the user has proven to be of great value in debugging the modeling language for our domain of application. We needed to find a representation language that captures all information we want to express about a circuit. In the process of finding such a language, we have changed the representation several times, without having to change the constraint propagation algorithm.

CHAPTER IV

REASONING ABOUT ELECTRICAL CIRCUITS

In this chapter we address the reasoning tasks we wish to perform regarding linear circuits in sinusoidal steady state. The main reasoning tasks we have developed are Analysis, First Order Reasoning (reasoning about change), Diagnosis, and Incremental Design. Other reasoning tasks include the reduction of a circuit by order of magnitude reasoning, and the generation of all qualitatively different states for a given circuit. The set of programs that perform these reasoning tasks about circuits in steady state have been grouped in a module called Qualitative Phasor Analysis (QPA).

All reasoning tasks that QPA addresses are based on the concept of constraint-based model of the circuit. First, the system generates the the BSOC. Next, the user asserts a set of constraints that represent the circuit's operating conditions. Propagation of those constraints reduces the number of possible behaviors of the circuit, yielding the Working Model of the circuit (WMC). Based on the WMC, QPA performs the reasoning tasks described in the following paragraphs.

First Order Reasoning (FOR) is reasoning about change; i.e. determining what the effects on the circuit variables are when we allow one or more circuit parameters or variables to change. Typical questions solved by this module are "What happens if R_1 increases?", or "How can I get the phase angle of cluster S_1 to decrease?". These problems can be modeled and solved by confluences. The representation of confluences can be seen as a particular case of the more general problem of value propagation, with the characteristic that the algebraic expressions only involve addition and subtraction and the values the variables take on are signs (i.e. $-$, 0, $+$).

Diagnosis can be seen as the process of measurement interpretation. The systems constantly probes a subset of the circuit variables, if any symptom is detected, a set of candidate faults are generated. Those candidate faults can be verified by modifying the circuit (according to the suggested fault), simulating the new circuit and verifying whether the observed behavior is an expected behavior of the new circuit.

The last problem we are addressing is Control Design. Control design answers questions similar to those in FOR (e.g. "How can I get the phase angle of cluster S_1 to decrease?"). In some cases, if one or more parameters are variable, we can solve those questions by FOR alone. In some other cases, even if there are some variable parameters, there is no setting of those parameters that would make us achieve the goal(s). Furthermore, we might want to further constrain the solution to control design problems (e.g. "How can I get the phase angle of cluster S_1 to decrease, without changing V_{S1}?"). The approach we take is based on means-ends analysis; the initial situation is represented by the working model, the goal situation by the constraints in the goal plus the design constraints, and the operators are circuit modifications that, when applied to the circuit, change its conditions. We design by inserting one modification at a time until all goals are satisfied and no constraint is violated.

This module, responsible for reasoning about circuits in steady state, is called QPA (Qualitative Phasor Analysis), and is sketched in the diagram in Figure 54, illustrating its interaction with the constraint propagation engine (see section III).

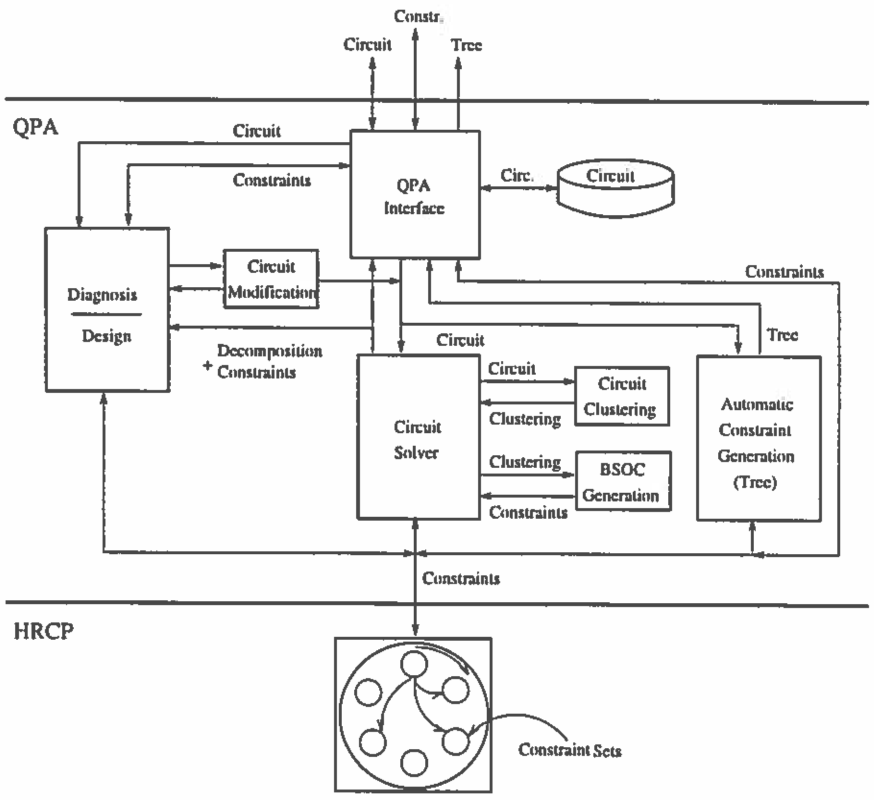


FIGURE 54. Qualitative Phasor Analysis (QPA)

The QPA interface receives ciruit descriptions and requests from the user or another module and routes them to the appropriate component or to the propagation module. The interface is also responsible for keeping a record of the circuit being solved and its (possibly partial) solution. The different reasoning tasks that QPA can perform are explained in the following sections.

Constraint-Based Circuit Model

The main idea behind our approach to qualitative circuit analysis and the other reasoning tasks is to express the problem as a constraint propagation problem. In that way, the user can assert facts about the circuit, and the constraint propagation engine will compute the logical consequences of the assertions in the circuit model. Those logical consequences can be used to verify that a given property or relation among variable values holds, or how a change in the circuit parameters or topology would affect the predicted behavior.

General knowledge [27, 49, 39] about the domain can be expressed in terms of constraints (algebraic equations) obtained from basic electro-magnetic theory, Kirchoff's laws, and vector algebra. From those algebraic constraints, we can also derive the respective confluences. We can express all that knowledge about the domain in constraints of the following types: algebraic, value, phase angle, order of magnitude, and confluences. Algebraic, value, order of magnitude, and phase angle constraints will be used to determine the circuit behavior. Confluences will allow us to reason about how changes in some variables cause change in other variables of the system. Order of magnitude constraints can be used to simplify the circuit structure, when possible.

To determine the set of algebraic constraints, we represent the circuit as a recursive structure of parallel/series clusters [32]. From that structure, we can derive the necessary algebraic and qualitative constraints that allow us to perform the reasoning tasks we have in mind. For each single element, we can derive the constraints shown in Table 6.

Element	Algebraic Constraints	Phasor Information	Confluences
R	$Z_R = R$ $V_R = Z_R I_R$	$\angle Z_R = 0$ In-Phase(I_R, V_R)	$\partial Z_R = \partial R$ $\partial \angle Z_R = 0$ $\partial V_R = \partial Z_R + \partial I_R$
C	$Z_C = \frac{1}{\omega C}$ $V_C = Z_C I_C$	$\angle\ Z_C = -90$ Ahead(I_C, V_C)	$\partial Z_C = -\partial C$ $\partial \angle Z_C = 0$ $\partial V_C = \partial Z_C + \partial I_C$
L	$Z_L = \omega L$ $V_L = Z_L I_L$	$\angle\ Z_L = 90$ Ahead(V_L, I_L)	$\partial Z_L = \partial L$ $\partial \angle Z_L = 0$ $\partial V_L = \partial Z_L + \partial I_L$

TABLE 6. Constraints for Single Elements

The constraints we derive for each cluster are shown in Table 7. In the table, we assume a series cluster S with elements 1 and 2, and a parallel cluster P with elements 1 and 2. Impedances, currents, and voltages are indexed accordingly. Notice we are not sanctioning constraints like $V_1 < V_S$ for the series cluster, since voltages are phasors and the constraint does not hold under all conditions.

Figure 1, on page 8, graphically represents the idea of hierarchical clustering and the production of constraints for each element and cluster in the circuit. Chapter III mentions how the different constraints can be represented and how the constraint propagation mechanism works. Chapter II mentions how addition and parallel con-

Cluster	Algebraic Constraints	Confluences
Series S	$I_S = I_1$	$\partial I_S = \partial I_1$
	$I_S = I_2$	$\partial I_S = \partial I_2$
	$V_S = V_1 + V_2$	$\partial V_S = \partial V_1 + \partial V_2$
	$V_S = Z_S I_S$	$\partial V_S = \partial Z_S + \partial I_S$
	$Z_S = Z_1 + Z_2$	$\partial Z_S = \partial Z_1 + \partial Z_2$
Parallel P	$V_P = V_1$	$\partial V_P = \partial V_1$
	$V_P = V_2$	$\partial V_P = \partial V_2$
	$I_P = I_1 + I_2$	$\partial I_P = \partial I_1 + \partial I_2$
	$V_P = Z_P I_P$	$\partial V_P = \partial Z_P + \partial I_P$
	$Z_P = \frac{Z_1 Z_2}{Z_1 + Z_2}$	$\partial Z_P = \text{parallel}(\partial Z_1, \partial Z_2)$

TABLE 7. Constraints for Series and Parallel Clusters

fluences can be computed.

QPA provides a function that takes a circuit description and creates a *circuit object.* The circuit is defined by means of its components and their interconnections. For example, the circuit in Figure 1 (page 8) would be defined as shown in Figure 55

```
(define-circuit
    :name test-circuit-1
    :components
         ((voltage-source V 1 0)
          (resistor R1 1 2)
          (inductor L  2 3)
          (resistor R2 3 0)
          (capacitor C 3 0)))
```

FIGURE 55. Circuit Definition

Circuit Analysis

We have a mechanism to represent and propagate the information in the constraint-based circuit model. Given that, the process of circuit analysis can be seen as solving the set of constraints that govern the circuit behavior. We start with a basic set of constraints, the Basic Circuit Model (BCM), which represents the set of possible behaviors the given circuit may exhibit.

Figure 9, repeated in Figure 56 illustrates one way to cluster our example cir-

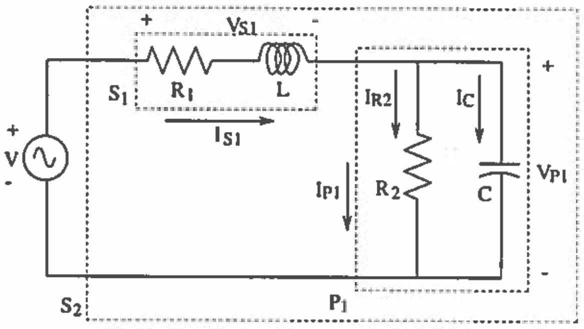


FIGURE 56. Simple Circuit

cuit, and Table 8 shows how the circuit's structure is traversed, and the constraints generated.

Every time the user posts a new constraint, the set of expected behaviors is (possibly) reduced by constraint propagation. The user-provided constraints (normally) represent the operating conditions of the circuit. After asserting the operating conditions in the BCM, and running propagation, we obtain the WMC.

QPA provides functions to assert and verify constraints related to a given circuit. Asserting constraints consists of passing the same constraint to the model of the circuit and running propagation. To verify if a given constraint holds in a model, we check if it is explicitly recorded; if it is, the constraint holds. If the constraint does not exist in the model, it can be verified via constraint propagation. The constraint is asserted to (a copy of) the model, and propagation is run. If propagation detects an inconsistency, the constraint clearly does not hold. If propagation does not fail, we can

Region	Algebraic	Phase Angle	Confluences
Series	$I_{S_2} = I_{S_1}$		$\partial I_{S_2} = \partial I_{S_1}$
S_2	$I_{S_2} = I_{P_1}$		$\partial I_{S_2} = \partial I_{P_1}$
	$V_{S_2} = V_{S_1} + V_{P_1}$		$\partial V_{S_2} = \partial V_{S_1} + \partial V_{P_1}$
	$V_{S_2} = Z_{S_2} I_{S_2}$		$\partial V_{S_2} = \partial Z_{S_2} + \partial I_{S_2}$
	$Z_{S_2} = Z_{S_1} + Z_{P_1}$		$\partial Z_{S_2} = \partial Z_{S_1} + \partial Z_{P_1}$
Series	$I_{S_1} = I_{R_1}$		$\partial I_{S_1} = \partial I_{R_1}$
S_1	$I_{S_1} = I_L$		$\partial I_{S_1} = \partial I_L$
	$V_{S_1} = V_{R_1} + V_L$		$\partial V_{S_1} = \partial V_{R_1} + \partial V_L$
	$V_{S_1} = Z_{S_1} I_{S_1}$		$\partial V_{S_1} = \partial Z_{S_1} + \partial I_{S_1}$
	$Z_{S_1} = Z_{R_1} + Z_L$		$\partial Z_{S_1} = \partial Z_{R_1} + \partial Z_L$
R_1	$Z_{R_1} = R_1$	$\angle Z_{R_1} = 0$	$\partial Z_{R_1} = \partial R_1$
	$V_{R_1} = Z_{R_1} I_{R_1}$	In-Phase (I_{R_1}, V_{R_1})	$\partial V_{R_1} = \partial Z_{R_1} + \partial I_{R_1}$
L	$Z_L = \omega L$	$\angle Z_L = 90$	$\partial Z_L = \partial L$
	$V_L = Z_L I_L$	Behind (I_L, V_L)	$\partial V_L = \partial Z_L + \partial I_L$
Parallel	$V_{P_1} = V_{R_2}$		$\partial V_{P_1} = \partial V_{R_2}$
P_1	$V_{P_1} = V_C$		$\partial V_{P_1} = \partial V_C$
	$I_{P_1} = I_{R_2} + I_C$		$\partial I_{P_1} = \partial I_{R_2} + \partial I_C$
	$V_{P_1} = Z_{P_1} I_{P_1}$		$\partial V_{P_1} = \partial Z_{P_1} + \partial I_{P_1}$
	$Z_{P_1} = \frac{Z_{R_2} Z_C}{Z_{R_2} + Z_C}$		$\partial Z_{P_1} = parallel(\partial Z_{R_2}, \partial Z_C)$
R_2	$Z_{R_2} = R_2$	$\angle Z_{R_2} = 0$	$\partial Z_{R_2} = \partial R_2$
	$V_{R_2} = Z_{R_2} I_{R_2}$	In-Phase (I_{R_2}, V_{R_2})	$\partial V_{R_2} = \partial Z_{R_2} + \partial I_{R_2}$
C	$Z_C = \frac{1}{\omega C}$	$\angle Z_C = -90$	$\partial Z_C = -\partial C$
	$V_C = Z_C I_C$	Ahead (I_C, V_C)	$\partial V_C = \partial Z_C + \partial I_C$

TABLE 8. Generation of the BSOC

say the constraint is holds, and even show the user the consequences of that constraint in the model (the resulting constraints of propagation). Chapter III presented several examples of how propagation works and how it can be used to perform circuit analysis and first order reasoning. Figure 57 shows part of the set of constraints printed following the circuit structure (i.e. clustering). Confluences and other algebraic constraints are not shown in the figure.

QPA presents several advantages over conventional circuit solvers. Verifying that a relationship among circuit variables (a constraint) holds can be accomplished

```
Component1:
  Single VOLTAGE-SOURCE: V
  . . .
Component2:
  SERIES cluster: S2
  nodes: (1, 0)
  ZS2 = ([20.3477, 22.3792] L [19.7045, 29.6023])
  VS2 = 100.0000
  IS2 = ([4.4684, 4.9146] L [330.3977, 340.2955])
  OMC = ((= VS2 VV) (= IS2 IV) (= IS2 IP1) (= IS2 IS1))
  Component1:
    SERIES cluster: S1
    . . .
  Component2:
    PARALLEL cluster: P1
    . . .
    Component1:
      Single RESISTOR: R2
      nodes: (3, 0)
      R2 = [20.0000, 21.0000]
      ZR2 = [20.0000, 21.0000]
      VR2 = ([26.3706, 40.7957] L [258.0103, 272.9154])
      IR2 = ([1.2557, 2.0398] L [258.0103, 272.9154])
      OMC = ((= VR2 VP1))
    Component2:
      Single CAPACITOR: C
      nodes: (3, 0)
      C = [0.0020, 0.0025]
      ZC = ([6.6667, 8.3333] L 270.0000)
      VC = ([26.3706, 40.7957] L [258.0103, 272.9154])
      IC = ([3.7853, 5.0618] L [348.0103, 2.9154])
      OMC = ((= VP1 VC)
```

FIGURE 57. Printout of a Circuit Model

by verifying that constraint against the circuit model; if it preserves the model's consistency, we say it holds, otherwise it does not. A circuit model becomes inconsistent if we have two contradictory ordering constraints (e.g. $V_{S_1} > V_{P_1}$ and $V_{S_1} < V_{P_1}$), or whenever the range of values for a variable becomes empty.

In this process, the values of the variables can range from qualitative to quantitative, in an intermixed form. That is, some variables may be precisely specified (real numbers), while others may be partially specified by the use of intervals, and yet others be left totally unspecified (i.e. we know all values must be positive). This characteristic allows the user to provide the system with as much information as available at a given time. The system produces results as specific as its knowledge about the circuit being analyzed. Of course, if all parameters are precisely specified, the result is precise and coincides, in the numeric results, with any conventional circuit analyzer.

An interesting characteristic found in constraint programming is that any variables can be input or output at any time in the computation; even change its role in one execution. This makes the system ideal for situations where the user knows some desired operating conditions and some parameters, and wants to design an acceptable range of values for other parameters. The user can express the desired operating conditions as value constraints, and QPA computes the range of values that the rest of the parameters may take on to achieve the goal state. The point stated in the preceding paragraph also applies here, the more precise the provided information is, the more precise the design of the parameters will be. Figure 58 shows an example of parameter design for the circuit in Figure 56. In this example, the user provides values for all parameters but C, and some values for currents and voltages. A range for C and some other circuit variables are computed.

In specifying the operating conditions, or in designing ranges of parameters, the information may be available incrementally. Perhaps the user asserts that the value of the resistance $R2 = [5, 100]$; later on, (s)he decides that $V_{S_1} > V_{P_1}$; later, more information is obtained about $R2$, refining its value to $R2 = [10, 15]$. The computed interval for the capacitor will be refined as information arrives, getting more and more precise. Examples of these properties were given in Chapter III.

First-Order Reasoning

The process of determining the impacts of changes in certain parameters to the rest of the variables is known as First-Order Reasoning (FOR). The name comes from the fact that the first derivative expresses the rate of change of a given variable; so, FOR is the process of reasoning about first-order derivatives. Following the same idea, the process in which we compute the possible values the circuit variables can take on is called zeroth order reasoning.

Chapter I introduces the idea of FOR with an example query. In Table 8 we derive some confluences directly from algebraic constraints. For instance, Ohm's law for a resistor states that $V = ZI$. Deriving both sides of the equation and taking

```
(assert-constraint
 '((value W    60)         (value VS2 100)
   (value L    [ 0.1  2 ]) (value R1   [ 5   25 ])
   (value R2   [10   50 ])
   (value IS1 ([1.583,  1.970] L [287.044, 291.736]))
   (value IR2 ([0.386,  0.536] L [211.371, 217.320]))
   (value VR2 ([9.707, 13.360] L [211.371, 217.320])))
  ex01)

SERIES cluster
name: S2
nodes: (1, 0)
...
Component1:
   SERIES cluster
   name: S1
   ... values for other variables were computed as well

Component2:
   PARALLEL cluster
   name: P1
   nodes: (3, 0)
   ZP1 = ([4.9255, 8.4355] L [279.6346, 290.2753])
   VP1 = ([9.7073, 13.3605] L [211.3713, 217.3201])
   IP1 = ([1.5839, 1.9708] L [287.0449, 291.7367])
   Component1:
      Single RESISTOR
      name: R2
      nodes: (3, 0)
      ...
   Component2:
      Single CAPACITOR
      name: C
      nodes: (3, 0)
      C = [0.0019, 0.0034]
      ZC = ([4.9680, 8.9924] L 270.0000)
      VC = ([9.7073, 13.3605] L [211.3713, 217.3201])
      IC = ([1.4858, 1.9540] L [301.3713, 307.3201])
```

FIGURE 58. Example of Parameter Design

the signs we have $\partial V = \partial Z + \partial I$, where $\partial X = \left[\frac{dX}{dt}\right]$, and [ ] is the sign operator. Also, if V, Z, and I are phasors, we can relate the change rate of their angles by the confluence $\partial\angle V = \partial\angle Z + \partial\angle I$

In Chapter II we derived the conditional confluences needed for phasor addition, product, and to compute the equivalent impedance of a parallel cluster. Conditional confluences express how derivative variables are related under different operating conditions of the circuit. The same concept was captured in other work [17, 32] by switching models when simulation reached a different operating region of a device. In the case of linear circuits, we do not have many changing confluences, or models that change drastically in the different states of the device. So, instead of creating a different model for each possible combination of the variables involved in those situations, we express the model of the circuit as a single set of constraints, where only a subset of them can be used; determining whether a confluence can be applied or not depends on the current situation of the circuit (i.e. the set of variable values).

A set of confluences allows us to derive the impacts of changes in magnitudes on angles and vice-versa. This enables a complete first-order reasoning scheme, where we can derive all possible consequences a change in a parameter or variable can generate.

FOR itself answers some design questions like "What can I do to make $\angle Z_{S_1}$ smaller?". If there is a combination in changes in the parameters that can produce that constraint as an effect, those changes are the answer to the problem.

Behavior Tree Generation

At any point, the user can ask QPA to automatically generate all possible order of magnitude constraints consistent with the actual model of the circuit. For example, if two elements are in parallel, a fully constrained model must include some ordering constraint that relates the currents of both elements (since they share the same voltage).

The Automatic Constraint Generation module traverses the circuit clustering, verifying for each cluster, whether its variables (currents for parallel, and voltages for series clusters) are already involved in a constraint. If they are not, it generates a branch for each possible constraint and verifies if it is consistent, forming a tree; inconsistent branches are pruned. This module returns a tree with all possible fully constrained models of the circuit, which represent all qualitatively different solutions for the circuit. For example, for the circuit model shown in Figure 57, we can see that current I_C, in cluster P_1, can be smaller, equal, or greater than current I_{R_2}. Figure 59 shows part of the behavior tree for the same example.

Figure 59 shows only the order of magnitude constraints and the condition at which branching takes place. The tree shown is very skinny, since the model did not contain much imprecision. At the third level, only branch number one succeeded; i.e. only $I_{R_2} < I_C$ is consistent with the model, and branches containing $I_{R_2} = I_C$ and $I_{R_2} > I_C$ were pruned. Since the tree can become very large, the nodes are numbered in a systematic way.

```
Id            : 0
Condition     : NIL
Voltage OMC : ((= VV VS2) (= VP1 VC) (= VP1 VR2))
Current OMC : ((= IS2 IV) (= IS2 IP1) (= IS1 IS2)
                (= IS1 IL) (= IS1 IR1))
Children      :
   Id            : 3
   Condition     : (> VS1 VP1)
   Voltage OMC : ((= VV VS2) (< VP1 VS1) (= VP1 VC)
                   (= VP1 VR2))
   Current OMC : ((= IS2 IV) (= IS2 IP1) (= IS1 IS2)
                   (= IS1 IL) (= IS1 IR1))
   Children      :
      Id            : 33
      Condition     : (> VR1 VL)
      Voltage OMC : ((> VR1 VL) (= VV VS2) (< VP1 VS1)
                      (= VP1 VC) (= VP1 VR2))
      Current OMC : ((= IS2 IV) (= IS2 IP1) (= IS1 IS2)
                      (= IS1 IL) (= IS1 IR1))
      Children      :
         Id            : 331
         Condition     : (< IR2 IC)
         Voltage OMC : ((> VR1 VL) (= VV VS2) (< VP1 VS1)
                         (= VP1 VC) (= VP1 VR2))
         Current OMC : ((< IR2 IC) (= IS2 IV) (= IS2 IP1)
                         (= IS1 IS2) (= IS1 IL) (= IS1 IR1))
         Children      :
```

FIGURE 59. Part of the Behavior Tree for Model of Figure 57

Diagnosis

Now let us consider the problem of diagnosis as measurement interpretation. Given an electrical circuit, for which we have its description, model and operating conditions. QPA can derive the set of expected behaviors the circuit may exhibit under those conditions. We have measurement devices to probe a subset of the circuit's variables, and probe the circuit periodically to determine if the exhibited behavior belongs to the set of predicted behaviors. If that is the case, we continue probing and testing. If at some point we find the behavior unacceptable (e.g. some currents or voltages are out of range), we have to determine and report the possible fault or set of faults that made the circuit behave abnormally.

All quantities in a circuit are tightly related; the slightest fault may affect the currents and voltages in most elements. As a result, magnitudes and phase angles of voltages may be out of range even in normal elements or clusters, making them look faulty. This fact complicates the process of diagnosis.

Let us think of a series cluster with two elements e_1 and e_2, forming part of any linear circuit, as illustrated in Figure 60. Now assume a fault occurs in element e_1,

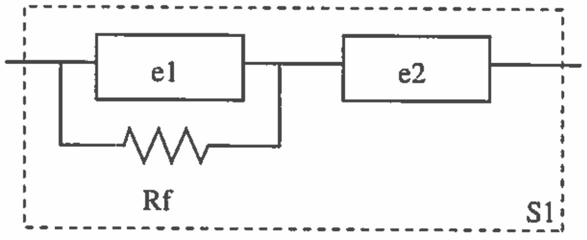


FIGURE 60. Fault in an Element of a Series Cluster

short-circuiting it through a small resistance (a non-ideal short circuit). The total impedance Z_{e_1} will decrease and the current I_{S_1} will increase, making V_{e_2} increase and V_{e_1} decrease. Depending on the nature of e_1, its current's phase angle may also change, affecting the phase angle of e_2's current, since the clusters are in series. The rest of the circuit's variables will exhibit similar effects.

Now, looking at element e_2, we see that its current's phase angle changes, but its voltage's phase angle changes also, indicating that its main characteristics have not changed. Therefore, it is not a faulty element. An element's or cluster's phase angle is defined as by $\angle V - \angle I$. Looking at the definition of impedance, assuming phasor quantities, we have that the phase angle of a component (element or cluster) corresponds to the angle of the component's impedance

$$\begin{aligned} Z &= \frac{V}{I} = \frac{V_m \angle V_a}{I_m \angle I_a} \\ Z &= Z_m \angle Z_a = \frac{V_m}{I_m} \angle (V_a - I_a) \end{aligned} \tag{IV.65}$$

We can measure absolute magnitudes of voltage and current on any element, and phase angle difference (i.e. $\angle V_a - \angle I_a$) as well. That information can be used to compute the magnitude and angle of the element/cluster's impedance. The impedance is then compared with the expected impedance of the element to determine if there is a fault and, if so, of what kind.

To detect a symptom, we compare the observed magnitude and angle of a variable with their expected values. If they intersect, we can say there is not an inconsistency; otherwise, the element is symptomatic. Now consider a series cluster S_1 (a similar consideration can be done for a parallel cluster), consisting of two elements e_1 and e_2, with impedances Z_{e_1} and Z_{e_2}. If both components of S_1 are normal (not faulty), their observed impedances can be used to compute what we call the *expected-observed impedance* of cluster S_1. In this situation, instead of checking Z_{S_1} against its expected impedance, we verify it against its expected-observed impedance, which may be a smaller set. This increases the accuracy of results in the presence of uncertain information.

The determination of a fault is done using a set of production rules, derived from first principles. That is, for each kind of element and cluster, we analyze the consequences of adding each kind of element in series and parallel. For instance, Figure 61 shows a situation where a capacitor C is short-circuited with a resistor R_f in parallel. Note that the capacitor was renamed C_o and the parallel cluster corresponds

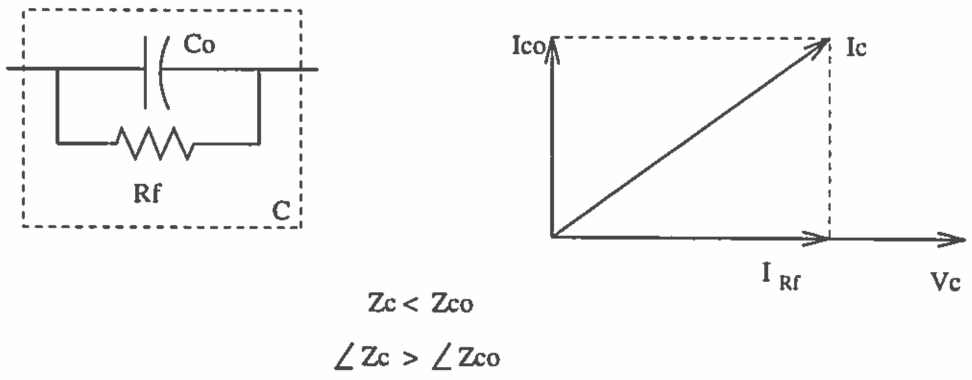


FIGURE 61. A capacitor Being Short Circuited by a Resistor

to the faulty capacitor, now called C, because it is the capacitor we are observing. The phasor diagram shows the current before the fault I_{C_o}, the fault current I_{R_f} and the total current of the faulty capacitor I_C. The current's angle and magnitude are drawn relative to the cluster's voltage V. Based on the change in the ratio V_C/I_C, we observe a decrease in impedance magnitude and a decrease in impedance angle. Each possible situation corresponds to one rule, where the consequences of the situation will be the condition of the rule and the added element the consequent of the rule.

We can consider ideal short and open circuit as special situations. In both, the current in the element is zero; the difference is that in short circuit, the voltage is also zero. There is still one more detail to take into consideration. When two or more elements are connected in series, and they are open-circuited, all currents are zero. Since no current is circulating by any of those elements, their voltages are zero, except for the one that has the open circuit. The reason for the voltage of the faulty element not to be zero can be derived from Kirchoff's voltage law applied to the series cluster. Since the voltage of all elements is zero (because their currents are zero), the voltage of the faulty element must equal the voltage applied to the whole series cluster. The dual situation is observed for short circuit of elements in a parallel cluster. Some of the diagnosis rules are illustrated in Table 9.

Element	Preconditions	Fault
Any	$V = 0$ $I > 0$	Short circuit
	$I = 0$ $V > 0$	Open circuit
Capacitor	$Z < Z_o$ $\angle Z > \angle Z_o$	Parallel Resistor
	$Z < Z_o$ $\angle Z = \angle Z_o$	Parallel Capacitor
Inductor	$Z < Z_o$ $\angle Z < \angle Z_o$	Parallel Resistor
	$Z > Z_o$ $\angle Z = \angle Z_o$ $or \angle Z = \angle Z_o - 180$	Parallel Capacitor
Inductive Cluster	$Z > Z_o$ $\angle Z < \angle Z_o$	Series Resistor
	$Z > Z_o$ $\angle Z < \angle Z_o$	Parallel Capacitor

TABLE 9. Selected Diagnosis Rules

One of the claims of this work is to be able to reason about the circuit with as much information as available. This idea translates to diagnosis in two dimensions:

imprecise information and incomplete knowledge. Imprecise information captures errors in measuring devices, and can be expressed by intervals. In this case, the procedure outlined above works, using the interval and complex fan ADTs.

Incomplete knowledge occurs when the set of probed variables is a subset of the circuit's variables, i.e. we do not know anything about some variables. This situation produces another set of possibilities. If we had no readings from e_1 in the series cluster in Figure 60, none of the above rules would apply (i.e. the values of voltages and currents are undefined, therefore they agree with any expected behavior); the fault would go undetected. A fault in one of the components propagates upward in the cluster hierarchy; a faulty element affects the impedance of the cluster it belongs to, and so on recursively. How much a faulty element affects its parent cluster (i.e. how fat do the effects propagate) depends on how precise the information is. The effects of a faulty element vanish when it is combined with other elements; as the effect travels upwards in the clustering tree, there must be a point where they go undetected. For this example, if we find cluster S_1 with symptoms, and e_1 is not being measured, there is a possibility that the fault resides in e_1, or in S_1, or in both.

If the two elements in Figure 60 exhibit observed impedances Z_{e1_o} and Z_{e2_o}, respectively, we can compute the expected value for the observed impedance of cluster S_1; namely, $Z_{S1_{eo}} = Z_{e1_o} + Z_{e2_o}$. Since our observations are normally real values, comparing Z_{S1} with $Z_{S1_{eo}}$ will produce a discriminarion at least as good as the one obtained by comparing it with the expected impedance Z_{S1_e}, which may be an interval.

This computed value $Z_{S1_{eo}}$, is called *expected-observed impedance.*

The diagnosis process can be decomposed into several steps. First, we transform the observations into a hierarchical data structure similar to the circuit clustering, where each node will have the expected, observed, and expected-observed impedances. This structure is called the *monitor tree.* While forming the monitor tree, the diagnosis rules are applied to each element and cluster, associating a set of possible faults to each element or cluster. Next, the monitor tree is traversed computing all combinations of possible faults; each combination (i.e. a conjunction of faults) is called a fault candidate. Finally, each fault candidate will be applied to the original circuit to model the diagnosed faulty circuit; the faulty circuit's working model is formed and the observations verified. Verification is done by asserting the observations (as value constraints) to the faulty circuit model. If no inconsistency is detected, we say the faulty circuit models the faulty behavior, and the fault candidate is promoted to fault, otherwise, the fault candidate is rejected. The set of all faults that pass the test are returned to the user. Figure 62 shows the pseudo-code for the diagnosis process.

Under this scheme, results improve with the amount of information available. The more elements that are probed, the better is the diagnosis we can produce. Our diagnosis mechanism compares the observed impedance with the expected-observed impedance first, and then with the expected impedance. The more elements we probe, the more nodes will have a value for expected-observed impedance, and the diagnosis will be more accurate. Also, for a given set of probed variables, the more precise

```
Model-Diagnosis(circuit, observations)
    monitor-tree = gen-monitor(circuit, observations)
    fault-cands = all-possible-faults(monitor)
    result = nil
    for each fault in fault-cands
          new-circuit = modify-circuit(circuit, fault)
          generate-model(new-circuit)
          if not error(assert-constraints(new-circuit, observations))
                result += (fault, new-circuit)
    return result
```

FIGURE 62. Pseudo-Code for Diagnosis

our working model is and the more precise the readings are, the more accurate our diagnosis is.

Figure 63 shows a set of observations and figures 64 shows part of the monitor tree and the resulting diagnoses. Notice that since there is no information about capacitor C, which is the faulty element, the diagnosis process takes into consideration the possibilities of a fault in C, in P_1, or in both. The result is sorted by number of faulty elements in each fault expression, so the fault (PARALLEL C R) is the first candidate to be tested; the test succeeds and we conclude that capacitor C is being shorted by a resistor. Also, notice that node P_1 is the only one with fault candidates; consequently, the result of the procedure all-possible-faults is equal to the fault candidates in P_1.

```
Observations:
  ((VV  100)
   (IV  ( 4.207 L 322.031))
   (VS2 100)
   (IS2 ( 4.207 L 322.031))
   (VS1 (95.396 L   3.455))
   (IS1 ( 4.207 L 322.031))
   (VR1 (71.531 L 322.031))
   (IR1 ( 4.207 L 322.031))
   (VL  (63.116 L  52.031))
   (IL  ( 4.207 L 322.031))
   (VP1 ( 7.474 L 309.723))
   (IP1 ( 4.207 L 322.031))
   (VR2 ( 7.474 L 309.723))
   (IR2 ( 0.373 L 309.723)))
   (VC  ( 7.474 L 309.723))
   (IC  ( 3.843 L 323.219)))
```

FIGURE 63. Observations for Circuit of Figure 57

Control Design

Now let us consider the problem of circuit design. A *control design task* can be defined in terms of an existing circuit, the goals to be achieved, and a set of design constraints. The goals are to be specified as changes in the circuit's variables, involving either magnitudes, angles, or both. For instance, for the circuit of Figure 56, a design goal could be $\partial \angle Z_{S_1} = -$, subject to the design constraint $\partial V_{S_1} = 0$. In words, we want the phase angle of cluster S_1 to decrease, without altering its voltage. We are to find modifications to the current circuit that *entail the goals without violating the design constraints.*

The control design problem can be seen as a planning problem, where the current circuit is the initial state, the current circuit plus the goals and the constraints are

```
MONITOR:

   SERIES cluster
   name: S2
   ... no faults found in S2

   Component1:
      SERIES cluster
      name: S2
      ... no faults found in S1

   Component2:
      PARALLEL cluster
      name: P1
      nodes: (3, 0)
      Expected   voltage: ([10.095, 88.010] L [220.761, 289.974])
                 current: ([2.829, 7.307] L [300.270, 357.355])
                 impedance: ([3.568, 12.044] L [280.491, 292.619])
      Observed   voltage: (7.474 L 309.723)
                 current: (4.207 L 322.031)
                 impedance: (1.776 L 347.692)
                 expected/observed impedance: (+ L ???)
      Fault candidates: (((PARALLEL C R)) ((PARALLEL P1 R))
                          ((PARALLEL C R) (PARALLEL P1 R)))
      Component1:
         Single RESISTOR
         name: R2
         ... no faults found in R2

      Component2:
         Single CAPACITOR
         name: C
         ... no faults found in C2 (no observations)

Diagnosis = (((PARALLEL C R)) ((PARALLEL P1 R))
             ((PARALLEL C R) (PARALLEL P1 R)))
```

FIGURE 64. Results of Diagnosis

the goal state, and the modification actions are the operators. The problem is to find a sequence of operators (a set of modifications to the circuit), such that the resulting circuit satisfies the task definition.

To define the operators, we observe what happens to each element and cluster when another element is connected in series or parallel. The consequences of the modification will be the consequences of the application of that operator. The set of operators (we call them design rules) are exactly the same as the set of diagnosis rules, but the semantics of use are inverted. In the case of diagnosis the consequences are seen as preconditions, to be matched with the observations. For a given design task, we first select one goal to be achieved, and select the design rules that contain that goal as a consequence. When applying an operator, the consequences indicate what changes that operation causes to the current circuit. To compute all the ramifications of the action, we can assert the rule's consequents in the circuit model, but doing so would cause inconsistencies. For instance, if we know that the impedance of a cluster does not change (under normal conditions) and the rule says it will decrease (under the faulty assumption), an inconsistency will be discovered immediately. To solve this problem, we suspend all confluence value constraints for the cluster in question and all its components; i.e. we make it a black box. That way, we could assert the consequents and run propagation without producing inconsistencies.

The asserted consequences may entail some of the goals or violate design constraints. The application of each rule reduces the number of goals to be achieved, and may satisfy or deny some of the design constraints; thus yielding a reduced design task.

Each design task keeps a record of what design goals and constraints have been satisfied so far. If a new design step violates any previously satisfied goal, we prune the search tree at that node. This decision was taken to avoid design loops; e.g. rule r_1 that ensures goal g_1 and violates g_2, and r_2 that ensures g_2 and violates g_1 will take us back to the beginning. At each design step, we record the rule used and its relevant consequences. Relevant consequences are the intersection of the consequences of the rule with goals and constraints. This information is used to generate an explanation sequence of the design process.

A design task can be formalized as a tuple <C,Rg,Sg,DC,IC,H>, where C is a circuit, Rg are the remaining goals, Sg the satisfied goals, DC the design constraints, IC the initial conditions, and H the history list. The procedure *Design-Step*, shown in Figure 65, takes as input dtask, a design task, and returns a list of all design tasks derived by satisfying one of dtask's remaining goals.

We implemented a breadth first exploration of the search space. A queue of remaining design tasks starts with only one element, the initial design task. Each application of a *Design-Step* produces a set of new design steps, one for each applicable rule. A node is removed from the queue; using *Design-Step*, its descendants are computed and then enqueued to be explored later.

If all the goals of a design task have been achieved we then check that none of

```
Design-Step(dtask<C,Rg,Sg,DC,IC,H>, goal)
    if Rg
          rules = select-rules(goal)
          result = nil
          for each (rule $\in$ rules)
                init-conds = suspend-values(this-goal,IC)
                new-constraints = assert(init-conds+consequences(rule)+Sg)
                if not(error?(new-constraints))
                      rel-constraints = (consequences(rule) + new-constraints)
                                                    $\cap$ (Rg + Sg)
                      new-Rg = Rg − rel-constraints
                      new-Sg = Sg + rel-constraints
                      new-H = H + <rule,rel-constraints>
                      result + = <new-C,new-Rg,new-Sg,init-conds,new-H>
```

FIGURE 65. Pseudo-Code for Design-Step

the constraints are violated. If any design constraint is violated, the system generates a new goal that would reverse those undesirable effects.

In the search process, we have a parameter to indicate the maximum depth to be searched (i.e. the maximum number of elements to be included in the design). We can also indicate how many solutions to find; the number of solutions is typically one or all, but may be any number. Figure 66 shows the search algorithm for incremental or control design.

Figure 67 shows an example of the definition of a design task, together with a transcript of two of the design solutions. The first solution causes Z_{S_1} to decrease, leaving the magnitude undetermined. This causes the voltage V_{S_1} to be undetermined, therefore not violating the constraint. That is, there must be a setting for the inserted capacitor that make the angle of S_1 decrease, without altering its voltage.

```
Incremental-Design(dtask<C,Rg,Sg,DC,H>)
    result = nil
    queue = dtask
    while q
          dt<C,Rg,Sg,DC,H> = remove(q)
          selected-goal = select-goal(Rg)
          new-designs = design-step(dt, selected-goal)
          for each new-dt ∈ new-designs
                if not violates-constraints(sol-dt Sg(new-dt))
                      look-for-satisfied-goals(new-dt)
                      if not(Rg(new-dt))
                            if |H| <= *max-depth*
                                  q + = Design-Step(dt)
                      else
                            if not (violates-constraints C, DC)
                                  if |result| = *num-designs*
                                        return result
                                  else
                                        result + = dt
                            else
                                  q + = violated-constraint-to-goal(dt)
    return result
```

FIGURE 66. Pseudo-Code for Incremental Design

The second solution causes the phase angle of S_1 to decrease, but also makes V_{S_1} decrease. Another resistor is inserted in parallel to P_1, which makes S_1's current and voltage increase. This effect adds to the previous one, making the total outcome ambiguous; this indicates that there must be a setting for the parameters of the design elements such that the constraint is preserved (not violated).

Figure 68 shows another design example, with two goals and one constraint and part of its solution. It is important to note that every time the design algorithm inserts an element, it computes all its consequences, and it may be the case that

```
(def-dtask
  :circuit     ex01
  :rem-goals   '((confluences (value (P ZS1) (? L -))))
  :constraints '((confluences (value (P VS1) (0 L ?)))))

Solution:
     (PARALLEL S1 C)
     provides ((CONFLUENCES (VALUE (P ZS1) (? L -)))
               (CONFLUENCES (VALUE (P VS1) (? L ?))))
Solution:
     (PARALLEL S1 R)
     provides ((CONFLUENCES (VALUE (P ZS1) (? L -)))
               (CONFLUENCES (VALUE (P VS1) (- L ?))))
     (PARALLEL P1 R)
     provides ((CONFLUENCES (VALUE (P ZP1) (- L ?)))
               (CONFLUENCES (VALUE (P IP1) (+ L ?)))
               (CONFLUENCES (VALUE (P IS1) (+ L ?)))
               (CONFLUENCES (VALUE (P VS1) (+ L ?))))
...
```

FIGURE 67. Design Task #1 and Solution

```
(def-dtask
  :circuit     ex01
  :rem-goals   '((confluences (value (P ZS1) (? L -)))
                  (confluences (value (P IP1) (+ L ?))))
  :constraints '((confluences (value (P VS1) (0 L ?)))))

Solution:
     (PARALLEL S1 R)
     provides ((CONFLUENCES (VALUE (P ZS1) (- L -)))
               (CONFLUENCES (VALUE (P ZS2) (- L -)))
               (CONFLUENCES (VALUE (P IS1) (+ L +)))
               (CONFLUENCES (VALUE (P IP1) (+ L +))))
     (PARALLEL P1 P)
     provides ((CONFLUENCES (VALUE (P ZP1) (- L +)))
               (CONFLUENCES (VALUE (P ZS2) (- L +)))
               (CONFLUENCES (VALUE (P VS1) (? L ?))))
...
```

FIGURE 68. Design Task #2 and Solution

some additional goals and/or constraints can be satisfied by the same operator. In this example we see that inserting a resistor in parallel with S_1 causes $\partial Z_{S_1} = (-\angle -)$ which in turn causes $\partial Z_{S_2} = (-\angle -)$. This makes the currents I_{S_1} and I_{P_1} increase. By trying to meet one goal we have satisfied both. The constraint is violated and an action is needed to satisfy that constraint. The system suggests the insertion of a resistor in parallel with P_1, completing the design for this particular solution.

Related Work

There are numerous publications about analysis, design, and diagnosis of electrical systems using qualitative reasoning, or model-based approaches. Most of them are designed to solve problems with digital or DC analog circuits. Very few mention analysis of linear circuits in sinusoidal steady state, and none of them have developed and formalized a system to reason about linear circuits.

Stallman and Sussman [41] developed EL, a rule-based system capable of doing circuit analysis. Their approach is based on constraints, expressed as rules. Those rules fire when they have enough information to produce a result. This is one of the first works where this kind of inference is called constraint propagation. The applications they present are limited to circuit analysis with DC sources. Their main focus is on controlling the search by use of dependencies derived from the constraints.

Sussman and Steele [45] present a language for describing constraint networks and computing via those constraints. Their main field of application is electrical circuits; their language allows the user to define circuit elements and their models. Slicing is a mechanism they provide to have different views of a portion of a circuit; this is equivalent to our clustering mechanism. They mention analysis of linear circuits in sinusoidal steady state, but they do not provide an automatic way to generate the constraint-based model; instead, they derive all constraints and do some algebraic manipulation by hand. They mention some aspects of parameter design explicitly. QPA provides modules to perform analysis and design based on the constraint representation of the circuit. Heintze and Michaylov perform numerical analysis of linear circuits in steady state using CLP(R) [23]. They generate the circuit model automatically by using constraint templates that are instantiated at run time. They use a complex arithmetic module, using rectangular representation. Their analysis is purely numeric, their intent is to illustrate the applicability of CLP(R) in the electrical engineering domain; they do not perform any reasoning about the circuit.

DeKleer's EQUAL [10] applies confluences to figure out how circuits work. His work uses first-order reasoning to derive causal chains due to perturbations in DC circuits. We have extended the concept of confluences to work with variables in the complex fan domain. This enables us to perform first-order reasoning about linear circuits in steady state. Williams [52] presents Temporal Qualitative Analysis, a technique to derive transient behavior of non-linear circuits. This technique performs large signal analysis of non-linear circuits by representing each circuit element by its different operating regions, and switching models when crossing boundaries.

In the area of diagnosis, there are quite a few publications, but most of them deal with digital circuits, or DC non-linear circuits. Davis [9] presents a methodology for diagnosis of digital circuits. His structural representation includes electrical, physical, thermal, and electro-magnetic adjacency; he claims that many of the faults are not only due to misbehavior inside an element, but also to interaction among elements. He develops the concept of constraint suspension for troubleshooting. The idea of constraint suspension is to find a constraint (element) whose retraction would leave the model in a consistent state. We use a similar concept in design, when we eliminate the derivative values of the element being modified, to avoid inconsistencies. Davis uses a discrepancy detection criteria; when a discrepancy between the expected and observed behavior is found the program traces back the discrepancies to the input, verifying every element in every possible path. That approach is conceivable because combinatorial circuits have a sense of direction from input to output. QPA also uses a discrepancy criteria, following the clustering paths on the tree that present a symptom, in the context of steady-state circuit behavior.

Genesereth [18] makes use of design descriptions to diagnose faults in combinatorial circuits. Given a set of symptoms, the program DART generates tests, simulates the results and generates a diagnosis. Hamscher [22] presents a system to diagnose combinatorial circuits. The paper focuses on how to model specific devices to get better results on diagnosis. That is, they focus on representing structure, behavior, and faults.

QPA's diagnosis process is limited in scope; dealing only with linear circuits. Other systems [14, 44] have used more general modeling languages, like the one used in QSIM. Nevertheless, most of them are unable to work with multiple faults, and none of them, to my knowledge, works with incomplete and uncertain information using intervals as the basic representation.

Subramanian and Dechter [44] present a diagnosis algorithm based on circuit structure. The circuits they diagnose are combinatorial circuits, and they claim their system works for circuits with cycles (sequential circuits). The diagnosis process is presented as an optimization process to yield minimal-cardinality solutions. QPA also provides minimal cardinality solutions to diagnosis problems.

In summary, most of the work in the field of circuit analysis, diagnosis, and design has focused on combinatorial and non-linear circuits, not involving linear circuits and the interaction of phasors. Our work shares many basic features of others and extends some other points. In particular, the characteristic of improving the diagnosis with the amount and accuracy of information available is unique to this work.

Chapter Conclusions

This chapter showed the usefulness of our modeling approach. By using a constraint-based model of the circuit, QPA can reason about linear circuits in sinusoidal steady state. QPA is capable of performing several reasoning tasks, such as circuit analysis, diagnosis, and design, circuit simplification by order of magnitude constraints, and the derivation of all possible qualitatively different states of the circuit.

All reasoning in QPA is based on a circuit model. Since the circuit model is formed by decomposing the circuit in series/parallel clusters, the kind of problems we can solve are limited to those that are series/parallel reducible; for other circuits we can use delta-star transformations (see [30]) and implement a more general solver. Delta-star transformations are well known in the field of electrical engineering; the extension of this work to perform those transformations does not seem to be difficult. Only the module that clusters the circuit needs to be modified, and the rest of the modules are general enough to support the extension. Delta-star transformations would raise other issues around our reasoning processes; one problem is that these transformations eliminate some nodes and elements, and include new ones. In first-order reasoning, for instance, causal chains would include variables corresponding to elements that do not exist in the original circuit. This problem is not present in series/parallel transformations, since the voltages and currents remain essentially the same; the transformation is very intuitive. Another problem is that the formulas for this transformations are more complicated; that would make value propagation considerably slower, and more chattering in propagating constraints would be present. Also, the more complex formulas, the more ambiguity will be present in confluence propagation, making the results less useful.

Using constraint propagation we can perform circuit analysis, which can be seen from several points of view. The use of intervals allows us to deal with information at several degrees of abstraction, ranging from qualitative (sign) to quantitative (floating point) values. The same mechanism can be used to express uncertainty. Measuring devices, for example, do not yield accurate readings; the values they represent can be expressed as the measured value plus/minus an error. That imprecision can be expressed by intervals.

Constraint propagation gives all variables a double role; any variable can be seen as input or output. This feature allows us to perform normal circuit analysis, where we specify the parameters and the value for the sources, and all currents and voltages are computed. Another possibility is to specify a subset of the parameters and a subset of the variables, and let QPA compute values for the rest of the parameters and variables. This feature can be used in parameter design, where we want to determine in what range the unspecified parameters can be set, so that the circuit exhibits the desired behavior.

An algorithm for diagnosis was presented in this chapter. This algorithm works with as much information as it is available from the user. Imprecise and/or incomplete information in the readings are tolerated by QPA. AS with analysis, the results of diagnosis will be as accurate as the information we provide. The diagnosis process will detect only those faults that maintain the circuit's topology series/parallel-reducible.

Control design or incremental design consists of the insertion of a few elements into an existing circuit to change its behavior. A design task specifies goals to be satisfied and design constraints, which have to be preserved (not violated). The method we present is basically means-ends analysis, where the initial situation is the current circuit, the operators are the modification actions, and the goal state is specified by the goal and constraints. The algorithm presented here provides a qualitative solution to the design problem. Once the design module returns a qualitative solution to a design problem, the user can use parameter design to specify the value of the parameters of the newly inserted elements, completing the design cycle.

The next chapter will explore the usefulness of our representation and reasoning scheme one step further, applying it to the solution of problems from the domain of power transmission systems.

CHAPTER V

POWER SYSTEMS

One domain of application suitable for using QPA is Power Systems Analysis, Diagnosis, and Control Design [19, 20, 15]. Power Systems are composed of linear circuits, with lumped, constant parameters, normally operated under sinusoidal steady state. Typical components of a power system are generators, transformers, transmission lines, capacitor banks, and loads (normally containing resistive and inductive components). Generators are energy sources that will deliver power to the system at a fixed voltage. Transformers and transmission lines are responsible for transporting energy from the production to the consumption sites. The loads are an abstraction of the different devices consuming energy from the system.

Some of the problems that require solution in power systems are:

Fault Analysis. Given a power system under faulty conditions (normally a short circuit), determine the voltage and currents in all elements and nodes. The information derived from the solution of this problem is used in determining adequate protection for the system against high currents and voltages.

Contingency Analysis. Similar to Fault Analysis, except that the changes in this case are inclusion and/or disconnection of lines, transformers, etc. The problem is to determine the changes in currents and voltages in all elements and nodes of the system.

Power Factor Correction. There are problems in the operation of power systems that require a slight modification in the system to achieve the desired behavior. For instance, if the power factor in a load site is low (large phase angle), a capacitor can be inserted in parallel with the load, with the resulting increase in power factor (decreasing in phase angle). A low power factor will increase the reactive power circulation in the transmission lines and transformers, which implies higher losses, the need for larger protection and switching devices, etc. This situation is, of course, undesirable.

Power Distribution. Another problem that requires structural changes to be designed is that of distribution of power transmission between parallel transmission lines. This problem arises in situations where the transmitted power increases, or one of the parallel transmission lines fails and one of the remaining lines is not capable of holding the resulting amount of current. In that case, the problem is solved by rerouting part of the current through transmission lines that still have some capacity. That is normally accomplished by installing capacitors, tap changing or phase shifting transformers in series with the transmission lines.

Load Studies. Given the voltages on the sources (generators) and the required power on the loads, determine the necessary power (real and reactive) the sources must deliver to the system, and the voltages on the loads. This information is important in systems planning.

Economical Load Distribution. Given the costs of generation and transmission, determine how much of the required load each generator is to produce. This information is very important for an economical operation of the system.

Fault and contingency analysis can be done by indicating the modifications to the circuit, and letting QPA do the analysis work. The new currents and voltages can then be compared to the original ones. Power factor correction and power distribution are special cases of incremental design. The last two problems require a model that handles power. We are not modeling power, but it would not be hard to include it in the model, and our system is flexible enough to adapt to those modeling changes.

To address the above problems, another layer, called Power System Analysis and Design (PSAD), is built on top of QPA, as shown in figure 7 and repeated as figure 69. The rest of this section illustrates how QPA can be used to solve some of the problems in Power Systems.

Power Systems Modeling

Power systems, in their vast majority, are three-phase systems. Nonetheless, their topology is usually depicted in a *one-line diagram*, which shows only one of those phases, and a simplified version of the interconnections, with the ground omitted.

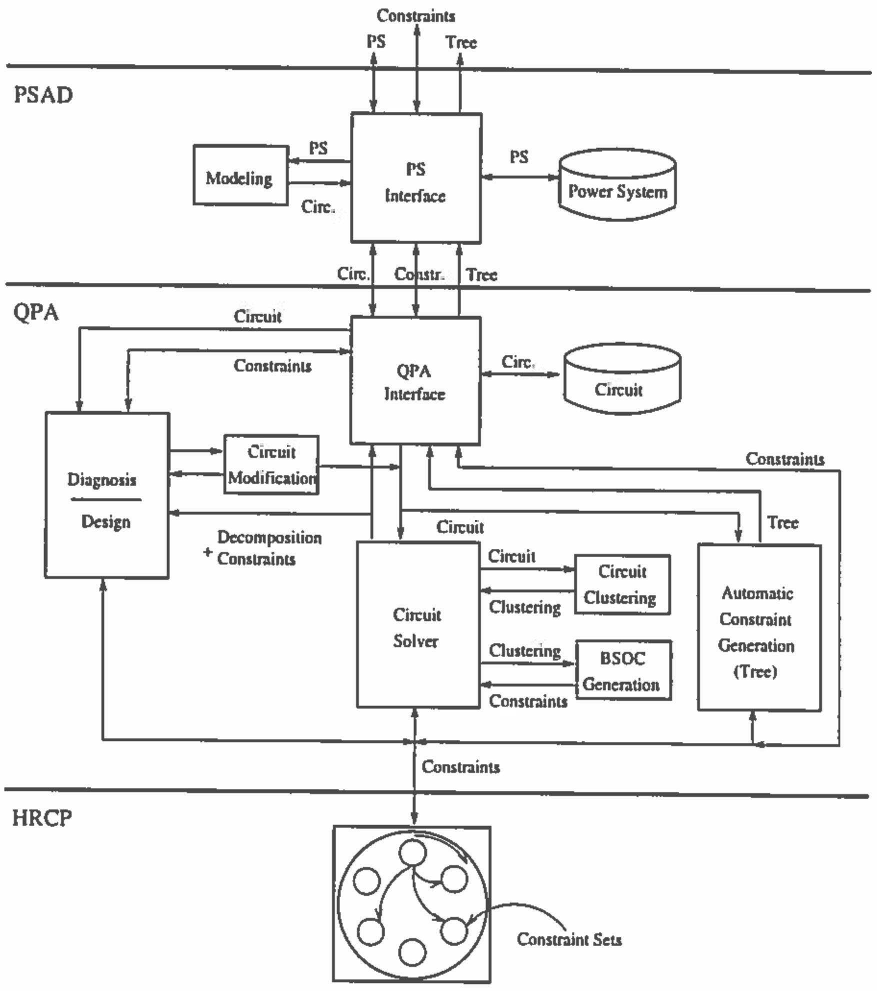


FIGURE 69. Architecture of the Complete System

Figure 70 shows a simple power system, represented by a one-line diagram.

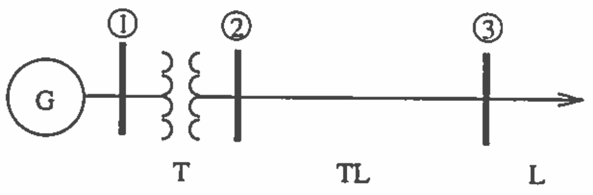


FIGURE 70. One-Line Diagram of a Simple Power System

Mapping from power systems to electrical circuits is straightforward. First, we must decide what models we are going to use to represent each component. The choice of what model to use depends on the precision we need in the answer. At this point, the model we have chosen is the one typically used for fault analysis: the simplest model that accounts for the inductive reactance of transformers and transmission lines, without paying any attention to losses or capacitive effects. We chose this model because the resulting circuit has relatively few elements. Since the number of constraints and variables grow with the number of elements, choosing simpler models will make constraint propagation be faster. Simpler models simplify the confluences; since we will be using sign values in confluences, there is no reason to use complicated models when, at the end, our methods are going to lose the model's precision. Also, the inclusion of more elements would unnecessarily increase ambiguity in confluence propagation, without giving any insight to behavior prediction. Since our reasoning methods for design and diagnosis depend mainly on confluences, it is appropriate to choose the simplest model. Figure 71 shows all the power system elements and their respective models in the circuit domain.

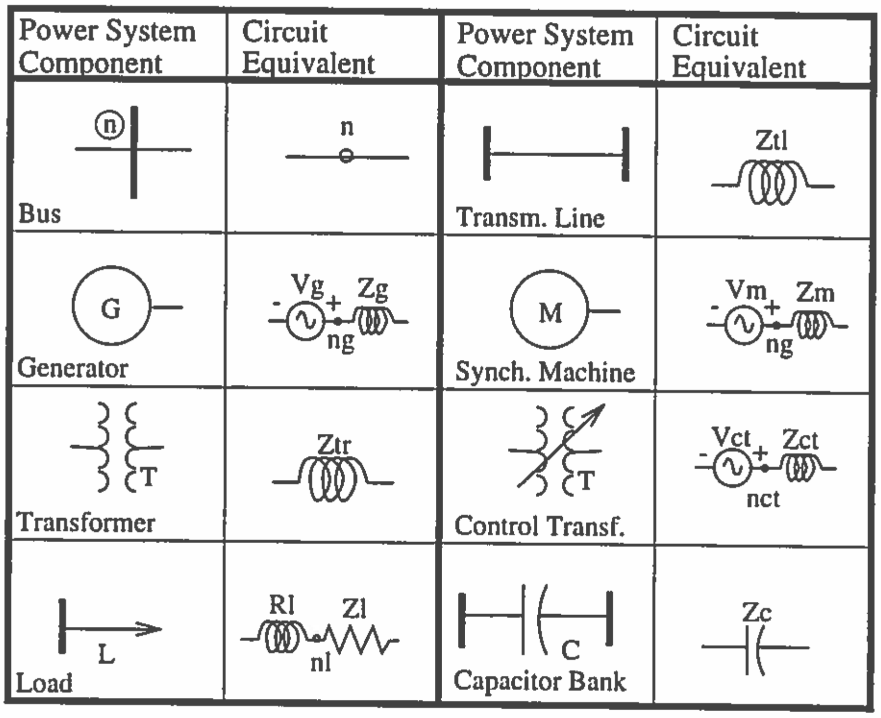


FIGURE 71. Modeling of PS Elements

At the time we define a power system, we can specify what model is to be used and, at the end, compare the results of using different models. This makes the system amenable to modification and extension. Although these are all the elements considered at this point, and those are the only models we have used in our analysis, the modeling module of PSAD provides a flexible way of declaring more power system elements and different models for each power system element. If the user needs more accuracy, the modeling mechanism is open to modifications; the model may be changed, or new power system elements may be added.

Figure 72 shows the definition of the power system of figure 70; figure 73 shows the circuit definition that PSAD generated for that power system, and its corresponding circuit diagram.

```
(def-PS-element
   :type   GENERATOR
   :name   G
   :models fault-analysis))

... other elements are defined here

(define-PS
   :name PS1
   :elements
      ((G (1))
       (T (1 2))
       (TL (2 3))
       (L  (3))))
```

FIGURE 72. Definition of the Power System of Figure 70

```
(def-circuit
  :name PS1-circ
  :elements
      ((Vg  (0  ng))
       (Xg  (ng 1))
       (Xt  (1  2))
       (Xtl (2  3))
       (Xl  (3  nl))
       (Rl  (nl 0))))
```

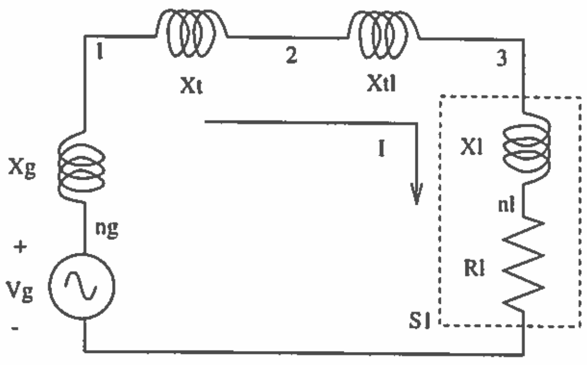


FIGURE 73. Circuit Model of PS of Figure 70

Reasoning about a power system is done by producing the circuit model of the power system and then calling QPA to perform the analysis of the given circuit under the set of constraints the user defined. When the power system module receives the results from QPA, those have to be *translated* back to their original meaning in the power system. This process is straightforward, but represents an important concept in the user interface; it would be really disconcerting for the user to receive an answer in terms of unknown circuit elements.

Control Design

The kind of problems the Control Design Module addresses are Power Factor Correction and Power Distribution [20]. To solve these problems, the power system is modeled as a linear circuit, all operating conditions are passed to QPA, and then the design task is translated to circuit terms and handed over to QPA for solution. The results are translated back into power system terms and returned to the user.

Our design mechanism handles general design specifications. In particular, it has been used to solve the two incremental design problems mentioned above.

Power Factor Correction

As pointed out in Chapter II, the electrical equipment that must be installed to supply a given load is determined by its total power, and therefore affected by the load power factor. Normal industrial loads operate at a lagging power factor, which can be improved or corrected by the connection of capacitor banks in parallel with the load.

As an example, consider the power system shown in figure 70; we want to decrease the phase angle of the load. The design goal is $\partial Z_L = (? \angle -)$. PSAD translates the design task to $\partial Z_{S_1} = (? \angle -)$. The circuit and the design task are passed to QPA, which solves the problem. Figure 74 shows the design task definition and figure 75 shows the results of the translation to circuit terms, performed by PSAD.

```
(def-PS-dtask
  :PS          PS1
  :rem-goals   '((confluences (value (P ZL) (? L -))))
  :constraints nil
```

FIGURE 74. Design Task Definition for Power Factor Correction

```
(def-dtask
  :circuit     PS1-circuit
  :rem-goals   '((confluences (value (P ZS1) (? L -))))
  :constraints nil
```

FIGURE 75. Circuit Design Task Definition Produced by PSAD

Among the solutions returned by QPA is the connection of a capacitor in parallel with the series cluster S_1; figure 76 shows the results of QPA and figure 77 shows the circuit diagram. The translated solution is shown in figure 78 and its corresponding one-line diagram is shown in figure 79.

```
Solution:
     (PARALLEL S1 C)
     provides ((CONFLUENCES (VALUE (P ZS1) (? L -))))
...
```

FIGURE 76. QPA Solutions to a Power Factor Correction Problem

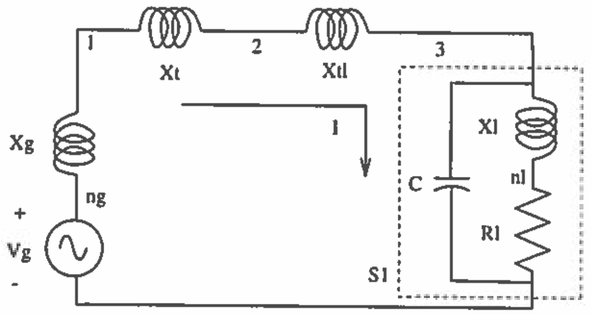


FIGURE 77. Circuit Solution to the Power Factor Correction Problem

```
Solution:
     (PARALLEL L C)
     provides ((CONFLUENCES (VALUE (P ZL) (? L -))))
...
```

FIGURE 78. Translation of QPA's Solution

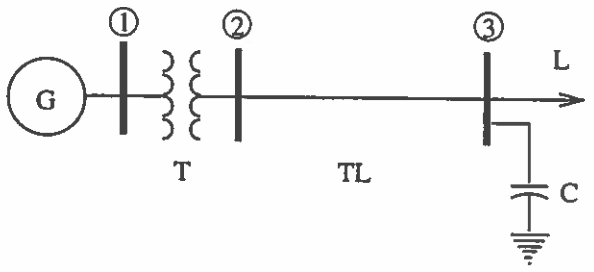


FIGURE 79. Power System Translation of Circuit in Figure 77

Power Distribution

There is another problem that can be solved by slight modification of the power system: the problem of power distribution on transmission lines. All methods to reroute power over parallel transmission lines require the insertion of a new element. Those elements can be Capacitors, Tap Changing Transformers, or Phase Shifting Transformers.

Consider two parallel transmission lines with equal inductive reactance, as shown in Figure 80. The currents through both lines are equal and we want to design

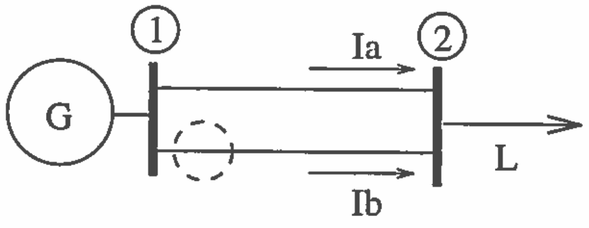


FIGURE 80. A Power Distribution Problem

a solution that ensures that $I_a < I_b$. Figure 81 shows the design task definition for this problem in power system terms.

In all solutions, the correction element goes where the dashed circle is placed. The first solution involves inserting a capacitor in series with the line whose current is to be increased. Alternative methods to redistribute the current are by insertion of a tap changing transformer (TCT), or a phase shifting transformer (PST)[1]. Both transformers are modeled by the insertion of a voltage source that accounts for the

[1][20, chapter 2] explains how these devices work

```
(define-PS
   :name PS2
   :elements
      ((G (1))
       (A (1 2))
       (B (1 2))
       (L (2))))

(def-PS-dtask
  :PS          PS2
  :rem-goals   '((confluences (value (P IA) (- L ?))))
  :constraints '((confluences (value (P IB) (+ L ?)))))
```

FIGURE 81. Design Task Definition for a Power Distribution Problem

increase in voltage or phase shift produced by the control transformer, compared to the other line.

The insertion of a capacitor is already covered in the design rules, but not the insertion of transformers. Table 10 shows how control transformers affect the behavior of inductances. It shows, from left to right, the inserted element, the phasor diagram, and the consequences, or expected changes. V_{Lo} is the old voltage in the inductor, while V_L is the voltage of the inductor after the insertion of the control element (same notation for currents); V_T is the voltage of the source modeling the control transformer. For the case of a TCT, the voltage source is normally in phase with the main source and for the case of a PST, they are very close to 90 degrees. Similar rules have been derived for how those transformers affect other elements.

The design task is passed to PSAD, which models the power system and translates the design task in circuit terms, and then passes the problem to QPA. Among

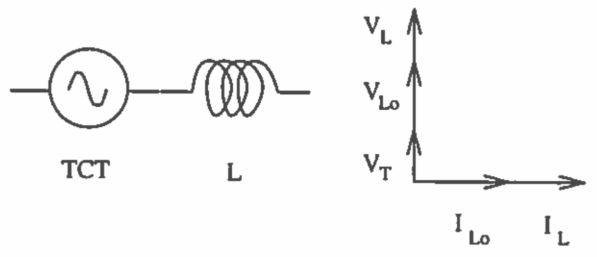
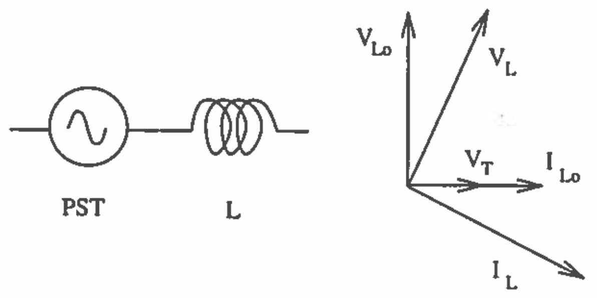
	$\partial Z_L = 0$ $\partial \angle Z_L = 0$ $\partial V_L = +$ $\partial \angle V_L = 0$ $\partial I_L = +$ $\partial \angle I_L = 0$
	$\partial Z_L = 0$ $\partial \angle Z_L = 0$ $\partial V_L = +$ $\partial \angle V_L = -$ $\partial I_L = +$ $\partial \angle I_L = -$

TABLE 10. Design Rules for Control Transformers

the solutions that QPA produces are the three mentioned above (see figure 82).

The first solution includes a capacitor in series with transmission line b, as shown in figure 83. The model of the line is a single inductor; including a capacitor in series is equivalent to reducing the magnitude of the impedance of the line. Therefore, the current in that branch of the solution circuit will be greater than the current in the other branch. Note that the phase angle of the current is not altered by the insertion of the capacitor. This explanation was paraphrased from the causal chain, derived by QPA from the design operation.

Figure 84 illustrates both solutions, inserting a TCT and inserting a PST; since both solutions are qualitatively equivalent, what makes them different is the numeric values of magnitude and phase angle of the voltage source that models the transformer. At some point in the design process, while exploring the design search space,

```
Solution:
     (SERIES B C)
     provides ((CONFLUENCES (VALUE (P ZB) (- L 0)))
               (CONFLUENCES (VALUE (P IB) (+ L 0)))
               (CONFLUENCES (VALUE (P IA) (- L 0))))

     (SERIES B TCT)
     provides ((CONFLUENCES (VALUE (P IB) (+ L -)))
               (CONFLUENCES (VALUE (P IA) (- L +))))

     (SERIES B PST)
     provides ((CONFLUENCES (VALUE (P IB) (+ L -)))
               (CONFLUENCES (VALUE (P IA) (- L +))))

     . . .
```

FIGURE 82. Translation of QPA's Solution

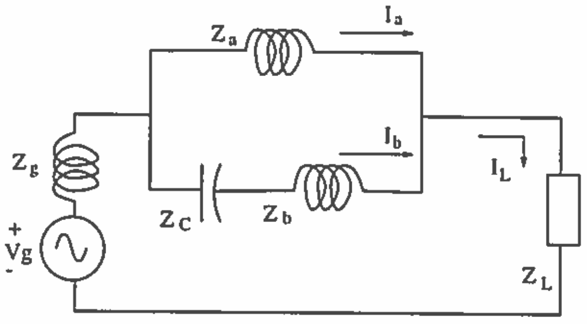


FIGURE 83. Rerouting Current by Inserting a Capacitor

QPA tries to satisfy the goal $\partial I_b = +$, suggesting the insertion of a control transformer. The consequences of that rule are asserted to the circuit model; as a result, we have that $\partial I_a = -$ is a also satisfied. The design is complete and returned to the user; no more modifications need to be done.

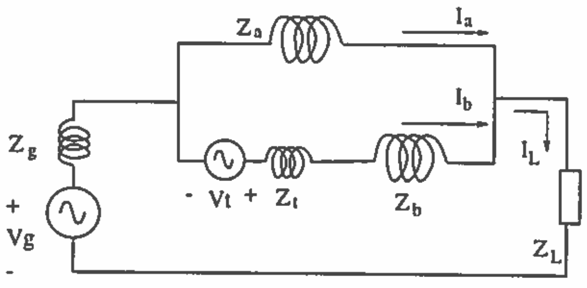


FIGURE 84. Rerouting Current by Inserting a Control Transformer

Power System Analysis

As mentioned in the previous chapter the design module provides a qualitative solution to the problem. To complete the design task, we can perform parameter design for the new elements.

In performing to the analysis of the circuit in figure 84, we encounter a new situation: it contains two voltage sources. To solve circuits containing multiple sources, QPA uses the superposition principle [27]. The circuit is decomposed into as many circuit components as it has sources. In each component, only one source appears and the rest are short-circuited. The idea of superposition is to compute the contribution of each source to the solution, and then add all the components to get the final result. QPA automatically performs the circuit decomposition, generates a partial model for each component and the necessary relations to gather all partial results to yield the overall solution. The algorithm for circuit solution is shown in figure 85.

Applying the superposition principle to this example, we short-circuit V_T, the modified circuit is shown in figure 86. Short-circuiting V_g, yields the circuit in figure 87.

```
Circuit Decomposition:
   for each source Vi in the original circuit
      form circuit component by short-circuiting all other sources
      label this component Cvi

Component Solution:
   for each component Cvi
      Model component using QPA
      Each variable X will be renamed Xvi

Gather solutions:
   put all models (constraints) together
   for each variable X in the circuit
      add constraint X = sum(Xvi)
```

FIGURE 85. Superposition Algorithm

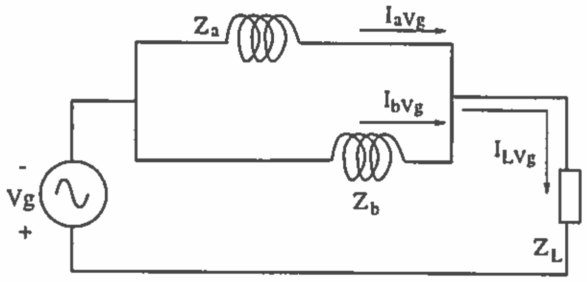


FIGURE 86. Component Due to V_g

QPA generates the partial models and the links that put the partial models together to form the total model. Note that the current I_{aVT} is in the opposite direction to the current I_{aV_g}. So, the totaling constraint is $I_a = I_{aV_g} - I_{aVT}$. QPA considers the first source as the reference source and determines in what direction the current flows through each element. For the rest of the sources, QPA verifies whether the current flows in the same or opposite direction. Those polarities are recorded with each circuit component, so that the circuit modeling module can determine the

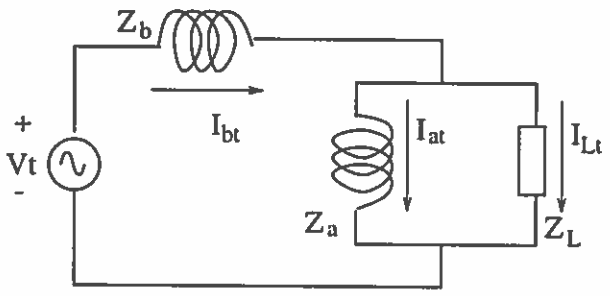


FIGURE 87. Component Due to V_T

right totaling constraints. Figure 88 and 89 show the partial components formed in the circuit decomposition of the circuit of figure 84.

By providing precise values to each of the designs, we can see the difference in behavior, even though they all satisfy the design. Figure 90 shows a comparison of those results.

The main difference between those methods are in the final distribution of power (current) over the lines. Using a Tap Changing Transformer, the *real* component of the current (responsible for the real power) stays the same, and the *imaginary* component (responsible for the reactive power) is redistributed. Phase Shifting Transformers do not touch the *imaginary* part, but redistributes the *real* part of the current. Capacitors redistribute both components of the current, without changing their phase angles. These characteristics were in fact found by QPA, which derived the right phase angle constraints. For instance, for the capacitor solution, it found that I_a InPhase I_L, while in the TCT solution, I_a Ahead I_L, and in the PST solution, I_a Behind I_L.

After the design module returns the solution "insert a tap changing transformer in series with **line b**", the user can ask: How can I make the current in **line a**

```
Circuit component: Vg
Polarities: ((La Vg +) (Lb Vg +) (S1 Vg +) (LL Vg +) (RL Vg +))
SERIES-PARALLEL cluster
name: SP1
nodes: (0, 2)
Component1:
   Single VOLTAGE-SOURCE
   name: Vg
   nodes: (0, 2)
Component2:
   SERIES cluster
   name: S2
   nodes: (2, 0)
   Component1:
      PARALLEL cluster
      name: P1
      nodes: (2, 3)
      Component1:
         Single INDUCTOR
         name: La
         nodes: (2, 3)
      Component2:
         Single INDUCTOR
         name: Lb
         nodes: (2, 3)
   Component2:
      SERIES cluster
      name: S1
      nodes: (3, 0)
      Component1:
         Single INDUCTOR
         name: LL
         nodes: (3, 4)
      Component2:
         Single RESISTOR
         name: RL
         nodes: (4, 0)
```

FIGURE 88. Printout of Component Due to V_g

```
Circuit component: VT
Polarities: ((Lb VT +) (La VT -) (S1 VT +) (LL VT +) (RL VT +))
SERIES-PARALLEL cluster
name: SP2
nodes: (1, 2)
Component1:
   Single VOLTAGE-SOURCE
   name: VT
   nodes: (1, 2)
Component2:
   SERIES cluster
   name: S3
   nodes: (2, 1)
   Component1:
      Single INDUCTOR
      name: Lb
      nodes: (2, 3)
   Component2:
      PARALLEL cluster
      name: P2
      nodes: (3, 1)
      Component1:
         Single INDUCTOR
         name: La
         nodes: (3, 1)
      Component2:
         SERIES cluster
         name: S1
         nodes: (3, 1)
         Component1:
            Single INDUCTOR
            name: LL
            nodes: (3, 4)
         Component2:
            Single RESISTOR
            name: RL
            nodes: (4, 1)
```

FIGURE 89. Printout of Component Due to V_T

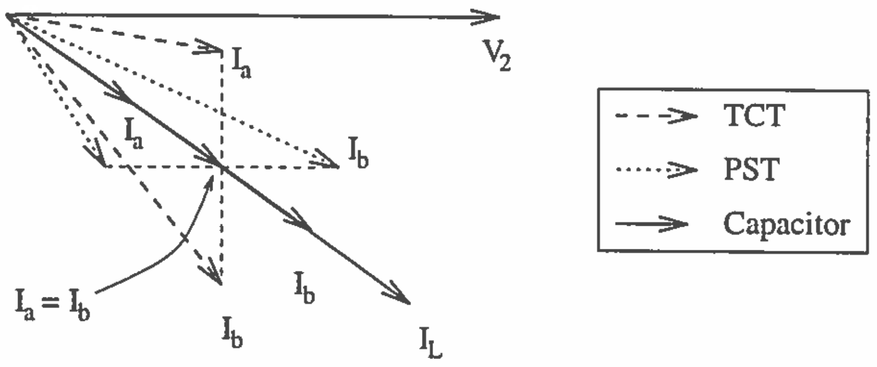


FIGURE 90. Different Solutions to the Power Distribution Problem

decrease even more? A transcript of the explanation is shown in figure 91. What the explanation shows is the affected variable, its value, the variable that affected it, and the constraint that produced a change in it. What the query really asks is if there is a change in V_T that can cause an increase in magnitude of I_b. The result of this query indicates an increase in the voltage of source V_T would cause that change. Translating this to power systems terms means increase the tap position in the TCT. The explanation can be paraphrased as follows:

> "An increase in I_{La} causes I_{LaVT} to increase,[2] this makes V_{LaVT} increase and the same happens to V_{P2VT}, the parallel equivalent. This in turn causes the current I_{P2VT} to increase, which causes the current I_{S3VT} to increase. As a result V_{S3VT} increases, which makes V_T increase."

The question could have been asked in the other direction: "what happens when the tap changes, increasing the voltage in the corresponding voltage source?". The

[2]This is because I_{LaV_g} does not change. I_{LaV_g} does not change because Vg doesn't change and none of the impedances change. This part was also included in the explanation, but not shown here for brevity.

```
Asserting constraints:
((CONFLUENCES (VALUE (P Vg) (0 L 0)))
 (CONFLUENCES (VALUE (P VT) (? L ?)))
 (CONFLUENCES (VALUE (P La) (0 L 0)))
 (CONFLUENCES (VALUE (P Lb) (0 L 0)))
 (CONFLUENCES (VALUE (P LL) (0 L 0)))
 (CONFLUENCES (VALUE (P RL) (0 L 0)))
 (CONFLUENCES (VALUE (P ILa) (- L +))))

Result:
((CONFLUENCES (VALUE (P VT) (+ L ?)))
 ...)

Explaining variable (P VT)
((P VT) (+ L ?) (P VS3VT)
 (CONFLUENCES T (== (P VT) (P VS3VT))))
((P VS3VT) (+ L ?) (P IS3VT)
 (CONFLUENCES T (== (P VS3VT) (DERIV* (P ZS3VT) (P IS3VT)))))
((P IS3VT) (+ L ?) (P IP2VT)
 (CONFLUENCES T (== (P IS3VT) (P IP2VT))))
((P IP2VT) (+ L ?) (P VP2VT)
 (CONFLUENCES T (== (P IP2VT) (DERIV/ (P VP2VT) (P ZP2VT)))))
((P VP2VT) (+ L ?) (P VLaVT)
 (CONFLUENCES T (== (P VP2VT) (P VLaVT))))
((P VLaVT) (+ L ?) (P ILaVT)
 (CONFLUENCES T (== (P VLaVT) (DERIV* (P ZLaVT) (P ILaVT)))))
((P ILaVT) (+ L ?) (P ILa)
 (CONFLUENCES T (== (P ILaVT)
                    (DERIV- (P ILaVT) (P ILa) (P ILaV)
                            (- ILaVT) ILa ILaV))))
User Provided: (P ILa) = (- L +)
```

FIGURE 91. Verifying Design Using FOR

system would predict a decrease in current I_{La} and an increase in current I_{Lb}.

Topological Simplifications

In Chapter IV, we mentioned QPA's ability to simplify circuits based on order of magnitude constraints. Consider the circuit in figure 87, the component due to the source in the control transformer. If the user asserts that the impedance of the load is much greater than the impedance of the transmission lines, i.e. $X_L \gg X_a$, QPA can reduce the circuit to that of figure 92. In that case we infer that $I_{Lt} \ll I_{at}$ and the load branch can be eliminated.

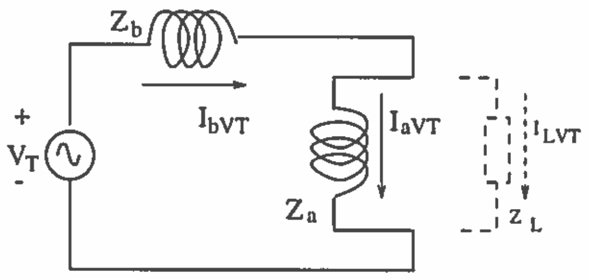


FIGURE 92. Reduction of Circuit of Figure 87 by Order of Magnitude Reasoning

Related Work

In the area of power system analysis, most of the work relies on matrix methods, for efficiency reasons [20]. Even some later work using constraint programming languages bases their computations on linear algebra [38]. All those methods, while gaining in efficiency, are sacrificing reasoning capabilities. Our methods have a greater inference capacity, but are not as efficient as theirs.

Few researchers have worked on qualitative analysis of power systems. Struss [42, 43] developed a system to diagnose faults in power transmission networks. He uses a relational approach to model power system components; consistency-based diagnosis is used to find faults in the system, based on the reading of “distance protection relays”. The representation he uses is not based on circuit theory; it focuses on what relays tripped and whether the observations are consistent with the models of those relays. Our system, QPA, is more focused on the understanding of behavior of electrical circuits, and supports diagnosis based on its components’ behavior and on phase angle information.

Hagman [21] addresses the diagnosis of power systems, focusing on the voltage decay problem. When there is a short circuit and the protection devices do not operate, most voltages present a decay, some branches present high currents, and others present almost zero currents. Hagman uses a qualitative approach to solve the problem, based on currents. His program uses the heuristic that sensors after the fault will show decreased or zero currents, those before the fault will show increased currents, and sensors in other branches will show a decrease in current. In Hagman’s approach, the circuit is traversed following the direction of current flow, comparing observations with expected behavior. Starting at the highest element, a list is made with all the element’s successors. If the current at a successor increased, the process is repeated recursively, starting at that element. Otherwise, the element is discarded. Diagnosis is performed by simple comparison, based on his heuristics. His approach does not handle uncertain or incomplete information; reading for all the elements in the path of a fault need to be available in order to diagnose the fault. Furthermore, his reasoning capabilities are limited to diagnosis.

Pettersson [36] developed an approach to power system diagnosis similar to ours. Based on readings of voltage and current, impedances are computed. The elements that present discrepancies between predicted and observed impedances are considered faulty. Our approach has the advantage that it deals with incomplete and/or uncertain information. Pettersson's approach is limited to real numbers.

Chapter Conclusions

Notice that reasoning about circuits in terms of phase angles is crucial to the solution of these problems. This kind of reasoning is done by extending the circuit ontology to include phase angles and phasor components. This capability is unique to QPA; previous work in the field could not solve these kind of problems because they did not include these components in the representation.

PSAD contains an interface to deal with the user's requests and to keep track of the power systems and the different results. Keeping track of that information allows the user to easily compare and contrast different behaviors for the different control options PSAD produces.

The application presented in this chapter shows that QPA can solve a set of problems of interest to the power systems community. Those problems have been studied extensively by electrical engineers and there are many different numerical solutions to them. Nevertheless, those solutions only do numerical processing, where all the knowledge to produce them is given by the engineer or programmer and the results are analyzed by the power systems engineer. QPA captures the basic knowledge about electrical circuits that allows PSAD to reason about the circuit, not only to yield numerical information.

The methods in this dissertation and traditional methods are not rivals; they complement each other. Automating the reasoning processes is necessary if we want to have intelligent agents that assists electrical engineers in taking better and faster decisions in design/operation of power systems. On the other hand, exact numeric information is always necessary to determine protections for the system, forecast load, etc.

In addition to the application of QPA to power systems, we envision the application of QPA or PSAD to education, training of operators, etc. Simulation programs only yield numerical results, giving the student no information about why results appear in the solution of a problem, or how a given solution was found. It will be very useful for a student to get a chain of causal effects to questions like "What happens to V_5 if there is a short-circuit in cluster S_3?", "What happens to V_3 when V_5 increases?", or "Suppose transformer $T1$ changes to a higher tap, how does the current (flow of power) in line from buses 5 to 3 change?" In fact, those questions appear all through the books on Power System Analysis, see for example [20, problem 3.13 on page 139;

problem 7.16 on page 282; problem 9.17 on page 379].

CHAPTER VI

CONCLUSION

In this chapter we discuss the evaluation of the implementation, the limitations of this work, and future directions of research, the main contributions made by this dissertation, and the final conclusion.

Evaluation

All ideas presented in this dissertation have been implemented in Allegro Common Lisp for Sun workstations. The different modules have been tested with numerous examples. To demonstrate that our representation captures enough detail and allows us to reason about circuits and power systems in the same terms we found in books, we have tested the programs several examples from circuit and power systems books. We did all the examples and problems found in chapter 5 of Kerr's book *Electrical Network Science* [27]; as an example, problem 5.10 gives a solution to a phase angle correction problem, and asks the student to determine the value of the capacitor to be inserted. Our system was able to produce that particular solution (among others) to the design problem, and then perform parameter design to determine the appropriate size for the designed capacitor. We also solved selected problems from

Walton's [49] and Lancaster's [49] circuit books.

To test the power system layer, we chose several examples from chapters 1, 2, 4, 7, and 10 of Grainger's book *Power System Analysis* [20]. Some of those examples involved transient analysis, but we were able to determine values for transient currents and voltages using steady state analysis. Fault analysis can be performed by means of steady state analysis by using the appropriate models (see [20] for a derivation of short-circuit models) for each element, and analyzing the resulting circuit. Some other problems had to be slightly modified, since they were not series/parallel reducible. In some cases by removing one element, our system was able to solve them.

Several of the solved problems involved two or more sources. The problem mentioned in Section V.2 involved two sources in the design solution and our system proved to be useful in performing circuit analysis, parameter design, and first-order reasoning with the multi-source circuit.

All of the above problems were small in size, i.e. less than 10 elements per circuit. However, we tested the program with a system with 20 elements and three sources. The set of test problems are presented in the Appendix.

Limitations and Future Work

There are several limitations in the representation and implementation we present in this dissertation. The first of them is that the system is restricted to solving circuits that are series/parallel reducible. Circuits that are not series/parallel reducible can be still analyzed by using delta-star transformations. When the series/parallel reductions get stalled, we perform a delta-star (or the converse) transformation, which allows us to continue performing the series/parallel reductions. The main reason why we decided to restrict the system in this aspect is that series/parallel reductions do not change the circuit topology; when we reduce two parallel or series elements, what we are doing is adding a different way of seeing the combination of those two elements (i.e. adding the constraints that allows us to model them in a different way). Delta-star transformations, on the other hand, do change the topology of the circuit, adding or removing nodes and connections. This fact complicates the reasoning process; we are not only interested in getting numerical results, but an intelligible account of how things occur in the circuit, based on first principles. We need to further investigate the ramifications of including those transformations in the modeling process.

In the development of the representation and modeling process, we had to face several trade-off situations. In constraint propagation, storing justifications of what caused a variable to change gives us the ability to retract constraints, but it takes more memory and time resources. The decision not to store that information led to two main losses: constraints cannot be retracted, and results cannot be explained. Although we are not storing a complete history of what variables changed, the causes, etc., in the way TMSs do, constraint propagation records a trace of the last propagation process; this propagation trace represents the paths followed in the hyper-graph of constraints in the propagation process. A path from one node (i.e. a variable) to another represents the causal chain that explains how the first node affects the other one. So, we could use those paths to generate explanations; however, the graph may contain many paths from one node to another, or even have loops. Displaying all paths would overwhelm the user and would not be useful; so, we decided to use the path of latest update, which has proven to be useful in many situations but does not provide the most interesting chain of causal effects in others. It may even be the case that more than one path is necessary. This exposes another avenue for future research, how to produce better explanations of a given result.

When modeling power systems with multiple sources we encountered several problems. The superposition principle makes us decompose the circuit into several components, for which different series/parallel clusterings are formed; this makes us lose the sense of structure we have in single-source circuits. Our diagnosis mechanism relies heavily on the structure of the circuit; in order to still be able to solve some diagnosis problems, we do as much clustering as possible before decomposing the circuit, so that at least some structure common to all components is preserved and the diagnosis procedure still (partially) works for these type of circuits. We need to investigate better ways to diagnose circuits with multiple sources.

In power systems control design, the power system modeler translates the design tasks directly into circuit terms, disregarding what elements are involved and physical restrictions on the kind of solutions that can be considered acceptable for certain problems. For instance, QPA generates solutions like "connect a capacitor in parallel with a transmission line", which is obviously an infeasible solution. PSAD could provide QPA with some topological constraints to avoid those problems and prune the search space accordingly.

In the development of the system, we did not pay much attention to the user interface, since it was not the main issue of this dissertation. However, to make the system practically usable, we need to develop a much better graphical interface. There are many tools available to develop WYSIWYG circuit editors, and to display the result graphically. Unless all parameters are precisely specified, most quantities in the system will be complex fans, which do not have a clear graphical representation. Up to this point we have not searched for a single solution to the circuit analysis problem under uncertain conditions; we limited ourselves to computing the domain from which each variable can take on values (i.e. complex fan or interval values). A solution is a member of the set of behaviors represented by the circuit model under the operating conditions. A precise solution does have a clear graphical representation: a phasor diagram. We envision a graphical interface where all changing parameters can be interactively changed (e.g. slide bars to assign values in the allowed ranges), and a window showing the corresponding phasor diagram updated for the given set of parameters. The main problem with this scheme is time; constraint propagation on interval labels is an NP-complete problem [7], the computation time would make this solution infeasible. Perhaps restricting constraint propagation to allow only certain operations, or lowering the accuracy of the system could yield faster implementations.

In Chapter V we mentioned the possibility of applying this work to education. This extension would involve an incursion into the fields of intelligent tutoring systems, student modeling, etc. This is closely related to the topic of explanation; a human expert gives different explanations to different persons, depending on the amount of knowledge the student has about the area or the particular system in question.

An interesting extension would be to implement the physical interface to probe real circuits, and experiment with the monitoring/diagnosis process in a power systems laboratory. This work would involve the development of measuring devices, which would be probing the system constantly and providing their readings to the monitor process. We believe this is a feasible plan, since constraint propagation can be left out of the loop monitor/diagnosis; propagation would be performed only when the operating conditions of the system change, to predict the set of expected behaviors, and, at the end, to validate the fault candidates.

Contributions

In this dissertation we have developed a representation that enables us to reason about linear circuits in sinusoidal steady state. This representation uses one of the main tools in the field of electrical engineering, *phasor analysis*. The main idea is to represent the circuit by a set of constraints that limits the set of allowed behaviors of the circuit. The set of constraints involves magnitude and phase angle information for each of the circuit variables. Based on this representation, we can perform qualitative analysis of electrical circuits, covering both zeroth- and first-order reasoning.

An interesting capability of this system is the ability to perform circuit analysis with as much information as provided by the user. This allows us to smoothly move from totally qualitative to completely quantitative information as the user acquires more knowledge about the conditions of operation of the circuit. This reasoning at different levels of abstraction is realized by representing all quantities as intervals; sign values can be represented as open intervals with infinite ends, uncertain information can be expressed by means of interval values; finally, precise information (i.e. real values) can be expressed as point intervals. Representing all values as intervals can be used to express knowledge about the circuit at different levels of abstraction, or it can be seen as a mechanism that allows QPA to perform analysis in the presence of incomplete information. If the user has no knowledge about the value of a variable, he or she just does not provide anything and QPA considers it simply to be positive, as all values in this domain are positive.

Since QPA is based on a constraint-based model, its inference mechanism is constraint propagation. Constraint propagation provides the feature that any variable can be an input or output variable, or even change roles in the same execution of the program; a feature not present in traditional circuit solvers. This feature allows the user to perform parameter design by specifying the desired values that current and voltage variables take on, as well as a subset of the parameters; QPA computes the corresponding values for the rest of the parameters. The price to pay for these features is computational efficiency; constraint propagation on arbitrary intervals is known to be NP-hard. To provide an efficient implementation, a general constraint propagation module was developed. This module stores constraints of different sets in data structures with different representations; this allows us to develop specialized constraint propagators, which are more efficient than general ones. The result of this implementation, called Hybrid Representation Constraint Propagation (mentioned in Chapter III) was published in an international conference [16].

The reasoning tasks we have in mind make polar representation of phasors more adequate than the rectangular counterpart. The combination of interval computation with phasors led us to the complex fan representation. In this representation, we developed algorithms that guarantee completeness, correctness, and minimality of results for the defined algebraic operations.

The proposed representation and modeling of linear circuits enables us to do a number of reasoning tasks that were not possible previously with the state of the art in qualitative reasoning and or electrical engineering. With QPA we can perform:

- circuit analysis at different levels of abstraction
- parameter design by means of circuit analysis
- the characterization of all qualitatively different phasor diagrams (i.e. states of the circuit)

- reduction of the circuit topology, based on order of magnitude reasoning
- circuit diagnosis based on measurement interpretation
- control design as circuit modification

All these reasoning techniques take full advantage of QPA's ability to deal with uncertain and incomplete information. For instance, diagnosis with uncertain or missing readings guarantees diagnoses as accurate as the provided information allows, modulo losses from the complex fan representation.

A main field of application of QPA, Power System Analysis, was introduced in Chapter V. That chapter mentions two design problems solved using QPA, power factor correction and power distribution on transmission lines. Power systems diagnosis was also mentioned in the same chapter. Contingency and fault analysis can be performed by modeling the original system, modifying the system topology, solving both systems, and comparing the results (this comparison must be done by the user at this point). First order reasoning can also be used to plan control actions, provided the system has one or more variable elements. After QPA determines which control action is to be done, circuit analysis can be used to design the value of the parameters to be changed, completing the design process.

QPA is an analytical tool that not only returns numerical results from a circuit simulation, but is also able to reason about the circuit using the same terms found in the explanations given in text books. This makes QPA an important tool to help the student of electrical circuits to really understand what is happening inside the circuit, as well as what would happen if parts of the circuit, or its operating conditions change.

We conclude this dissertation stating that by extending the circuit ontology to include phasor information, we have been able to broaden the range of systems that can be analyzed and the number of problems that can be solved by qualitative reasoning techniques.

APPENDIX

EVALUATION EXAMPLES

In this appendix, we present a series of examples used to evaluate the system. These examples were useful in verifying the expressive power of our representation, knowing the limitations of the system, evaluating the efficiency of our implementation, and debugging our code. This appendix is divided in two sections, the first one presents circuit examples and the second one power system examples. The circuit examples were taken from Kerr [27], Walton [49] and Lancaster [30]; the power system examples were taken from Grainger [20].

Circuit Examples

This section will be subdivided by circuit; in some cases, we have more than one problem related to the given circuit. In each example we will show a diagram of the circuit's topology, a transcript of the input and output of the program, and a brief discussion highlighting the important point of the given examples. In some cases, the limitations of our system do not allow us to solve all parts of a problem; in those cases, we changed slightly the problem statement, or solved only the parts we could. In some other problems, the book provided partial designs and asked to complete the design numerically; we went beyond the problem statement, designing the solutions provided by the book and then solving the parameter design problems. Very seldom the books provide design problems, they do not provide diagnosis either, and of course, do not even mention interval computations. We made up some design, diagnosis, and analysis problems to cover those cases. The first examples show more or less complete transcripts of the input and output. In subsequent cases, we will show only the interesting parts, basing our discussions at the circuit level and providing explanations similar to those found in text books.

Circuit #1

The circuit that we made more experiments with is the one presented on the examples throughout the dissertation. We performed nine circuit analysis problems, three design problems, and seven diagnosis simulations. That circuit is not presented here, since it was extensively discussed throughout the dissertation.

Circuit #2

This circuit corresponds to problem 5.6 of page 187 of Kerr's book. The problem statement asks to draw a phasor diagram; our system lacks that capability, but it can perform the circuit analysis with the data the book provides, and the user can draw the diagram from the results. Figure 93 shows the problem definition and diagram. Notice that the clustering, unique in this example, corresponds to the one of Figure 93. Based on the clustering, QPA produced the BSOC, which is not shown here, for brevity. This problem was also solved using mixed information: real and interval values. The introduction of a few interval values, causes most of the rest of the quantities to take on interval values. Figure 94 shows the results of the computation for the second case. We are not showing the first case for brevity.

Circuit #3

This circuit corresponds to problem 5.5 of page 187 of Kerr's book. This problem asks to find the voltage transfer ratio $V_{C_2}(j\omega)/V_S(j\omega)$ for the circuit of Figure 95. The only symbolic manipulation that QPA performs is solving a constraint for all variables, when an algebraic constraint is inserted in the model. Thus, QPA was unable to solve this part of the problem. The second part of the problem asks to compute V_{C_2} for a given value of V_S. This part was solved, as illustrated in Figure 96.

Circuit #4

This circuit corresponds to problem 5.10 of pages 187-188 of Kerr's book. This problem asks to design the value of the capacitor to make the power factor unity. In this example, we started with the circuit without the capacitor, and executed the control design module. The solutions that control-design returned, included the one proposed by the book. To complete the design process, we asserted value constraints for all parameters, except the capacitor, which was the unknown in the process. Figure 97 shows the circuit and task definition; the diagram includes the designed capacitor, drawn with dashed lines. Figure 98 shows part of the trace of the control design process and the parameter design query and answer. Notice that S_1's current's phase angle is zero, as specified in the statement of the problem, and the appropriate value for the capacitor was computed.

Circuit #5

This is the circuit of problem 5.20 of page 189 of Kerr's book. This is an interesting problem, because it includes two sources, one of which is a current source. The superposition principle is applied, decomposing the circuit in two components. Each component contains only one source, the rest are eliminated (i.e. voltage sources are short circuited, and current sources are open circuited). Figure 99 shows the

```
(def-circuit  :name 'ex21
  :elements '((r1 1 2)
              (c  2 3)
              (r2 3 4)
              (l  4 0)
              (r3 3 0)
              (s  1 0))))
```

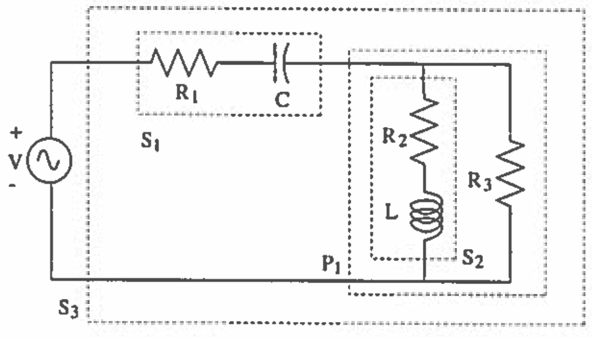


FIGURE 93. Circuit #2

```
Asserting constraints:
((variables (value W   1))          (variables (value VS 1))
 (variables (value R1 [0.50, 1.50])) (variables (value R2 2))
 (variables (value R3 [2.00, 4.00])) (variables (value L  3))
 (variables (value C  [1.00, 3.00])))
```

```
SERIES-PARALLEL cluster
name: SP1
nodes: (1, 0)
Component1:
   Single VOLTAGE-SOURCE
   name: S
   nodes: (1, 0)
   VS = 1
   IS = ([0.22, 0.83] L [337.17, 34.58])
Component2:
   SERIES cluster
   name: S3
   nodes: (1, 0)
   ZS3 = ([1.20, 4.51] L [325.41, 22.82]
   VS3 = 1
   IS3 = ([0.22, 0.83] L [337.17, 34.58]
   Component1:
      SERIES cluster
      name: S1
      nodes: (1, 3)
      ZS1 = ([0.60, 1.80] L [296.56, 347.47])
      VS1 = ([0.13, 1.49] L [273.74,  22.05])
      IS1 = ([0.22, 0.83] L [337.17,  34.58])
      Component1:
         Single RESISTOR
         name: R1
         nodes: (1, 2)
         R1  = [0.50, 1.50]
         ZR1 = [0.50, 1.50]
         VR1 = ([0.11, 1.24] L [337.17, 34.58])
         IR1 = ([0.22, 0.83] L [337.17, 34.58])
      Component2:
         Single CAPACITOR
         name: C
         nodes: (2, 3)
         C  = [1, 3]
         ZC = ([0.33, 1.00] L 270)
         VC = ([0.07, 0.83] L [247.17, 304.58])
         IC = ([0.22, 0.83] L [337.17,  34.58])
```

```
   Component2:
      PARALLEL cluster
      name: P1
      nodes: (3, 0)
      ZP1 = ([1.07, 2.88] L [ 19.44, 29.74])
      VP1 = ([0.23, 1.74] L [356.61, 64.32])
      IP1 = ([0.22, 0.83] L [337.17, 34.58])
      Component1:
         Single RESISTOR
         name: R3
         nodes: (3, 0)
         R3  = [2, 4]
         ZR3 = [2, 4]
         VR3 = ([0.23, 1.74] L [356.61, 64.32])
         IR3 = ([0.05, 0.87] L [356.61, 64.32])
      Component2:
         SERIES cluster
         name: S2
         nodes: (3, 0)
         ZS2 = (3.60 L 56.30)
         VS2 = ([0.23, 1.74] L [356.61, 64.32])
         IS2 = ([0.06, 0.48] L [300.30,  8.01])
         Component1:
            Single RESISTOR
            name: R2
            nodes: (3, 4)
            R2  = 2
            ZR2 = 2
            VR2 = ([0.13, 0.96] L [300.30, 8.01])
            IR2 = ([0.06, 0.48] L [300.30, 8.01])
         Component2:
            Single INDUCTOR
            name: L
            nodes: (4, 0)
            L = 3
            ZL = (3 L 90)
            VL = ([0.19, 1.45] L [ 30.30, 98.01])
            IL = ([0.06, 0.48] L [300.30,  8.01])
```

FIGURE 94. Results for Circuit #2

```
(def-circuit
  :name 'ex31
  :elements '((r1 1 2)
              (r2 2 3)
              (c1 2 0)
              (c2 3 0)
              (s  1 0)))
```

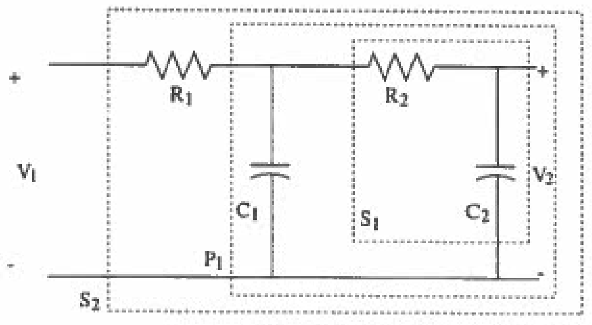


FIGURE 95. Circuit #3

```
Asserting constraints:
((variables (value W  1))
 (variables (value VS 1))
 (variables (value R1 1))
 (variables (value R2 1))
 (variables (value C1 2))
 (variables (value C2 2)))

SERIES-PARALLEL cluster
name: SP1
nodes: (1, 0)
Component1:
   Single VOLTAGE-SOURCE
   name: S
   nodes: (1, 0)
   VS = 1
   IS = (0.84 L 18.43)
Component2:
   SERIES cluster
   name: S2
...
   Component1:
      Single RESISTOR
      name: R1
      nodes: (1, 2)
...
```

```
   Component2:
      PARALLEL cluster
      name: P1
      nodes: (2, 0)
      ZP1 = (0.39 L 288.43)
      VP1 = (0.33 L 306.86)
      IP1 = (0.84 L  18.43)
      Component1:
         Single CAPACITOR
         name: C1
         nodes: (2, 0)
...
      Component2:
         SERIES cluster
         name: S1
         nodes: (2, 0)
         ZS1 = (1.11 L 333.43)
         VS1 = (0.33 L 306.86)
         IS1 = (0.29 L 333.43)
         Component1:
            Single RESISTOR
            name: R2
            nodes: (2, 3)
...
         Component2:
            Single CAPACITOR
            name: C2
            nodes: (3, 0)
            C2 = 2
            ZC2 = (0.50 L 270.00)
            VC2 = (0.14 L 243.43)
            IC2 = (0.29 L 333.43)
```

FIGURE 96. Results for Circuit #3

```
(def-circuit
  :name 'ex41d
  :elements '((r1 1 2)
              (r2 2 0)
              (l  2 0)
              (v  1 0)))

(design
 (def-dtask
   :circuit    ex41d
   :rem-goals '((confluences
                (value (P ZS1) (? L -))))))
```

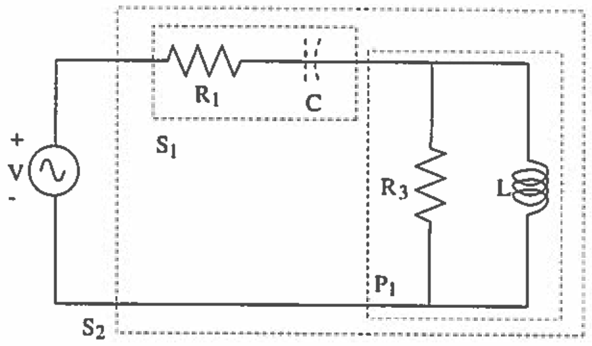


FIGURE 97. Circuit #4 and Design Task

```
*** looping *** |q| = 0
Remaining goals:
  ((CONFLUENCES (VALUE (P ZS1) (? L -))))
Goal selected:
  (CONFLUENCES (VALUE (P ZS1) (? L -)))
  Executing Design Step:
  # rules selected: 4
    ...
    Rule:
      (INDUCTIVE-CLUSTER?
       ((CONFLUENCES (VALUE (P ZS1) (? L -)))
        (CONFLUENCES (VALUE (P ZCD1) (+ L 0))))
       (SERIES S1 C))
    Consequences:
      ((CONFLUENCES (VALUE (P ZS1)  (? L -)))
       (CONFLUENCES (VALUE (P ZCD1) (+ L 0))))
    Sat goals: NIL
    Asserting constraints:
    ((CONFLUENCES (VALUE (P ZS1)  (? L -)))
     (CONFLUENCES (VALUE (P ZCD1) (+ L 0))))
    Result of propagation:
    ((CONFLUENCES (VALUE (P ZS2)  (? L -))))
  ...
  Design Step returns 4 successor design tasks
Analyzing Design Task:
  no additional satisfied goals
  no remaining goals
  does not violate constraints
  Solution found
...
Final Result:
Solution:
     (PARALLEL S1 R) provides
     ((CONFLUENCES (VALUE (P ZS1) (? L -))))
Solution:
     (PARALLEL S1 C) provides
     ((CONFLUENCES (VALUE (P ZS1) (? L -))))
Solution:
     (SERIES S1 R)   provides
     ((CONFLUENCES (VALUE (P ZS1) (? L -))))
Solution:
     (SERIES S1 C)   provides
     ((CONFLUENCES (VALUE (P ZS1) (? L -))))
```

```
(assert-constraint
 '((variables (value VV   1))
   (variables (value W    2))
   (variables (value L    3))
   (variables (value R1   1))
   (variables (value R2   2))
   (variables (value IS2 (+ L 0))))
 (fourth ex41d))

SERIES-PARALLEL cluster
name: SP1
nodes: (1, 0)
Component1:
   Single VOLTAGE-SOURCE
   name: V
   nodes: (1, 0)
   ...
Component2:
   SERIES cluster
   name: S2
   nodes: (1, 0)
...
   Component1:
      SERIES cluster
      name: S1
      nodes: (1, 3)
      ZS1 = (1.1661 L 329.0307)
      VS1 = (0.4165 L 329.0307)
      IS1 = 0.3571
      Component1:
...
      Component2:
         Single CAPACITOR
         name: C
         nodes: (2, 3)
         C = 0.8331
         ZC = (0.6000 L 270.0000)
         VC = (0.2142 L 270.0000)
         IC = 0.3571
   Component2:
   ...
```

FIGURE 98. Results for Circuit #4

circuit definition and diagram. Note that the component currents of R_1 subtract, while the component currents of R_2 add; this is shown in the polarities, and reflects in the numeric computation. Figure 100 shows the components, and the numeric value of the parameters for the different elements. The book asks for the voltage V_{R2}, which was correctly determined. Note that in the final results, all voltages are positive; the voltage of the $V_{VS} = 1$, the voltage of $V_{R2} = 1.3333$, and $V_{R1} = 0.3333$, but the nodes polarities indicate that the voltage V_{R1} goes from node 1 to node 0.

```
(def-circuit
  :name 'ex51
  :elements '((is 1 0)
              (r1 1 0)
              (vs 2 1)
              (r2 2 0)))
```

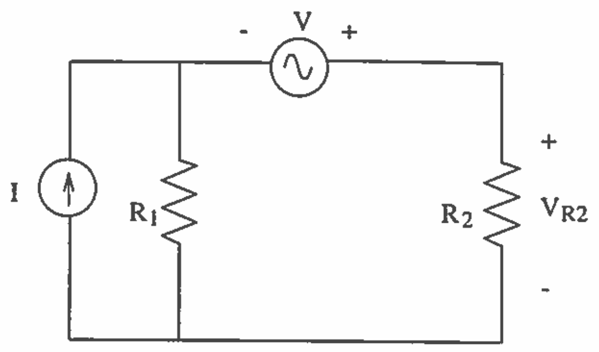


FIGURE 99. Circuit #5

Circuit #6

Problem 5.20 (pages 189-190) of Kerr's book is another multiple source circuit, whose solution requires the use of superposition. The problem asks for the total voltage for V_{RL}. Figure 101 shows the circuit definition, diagram, the input, and final output, and Figure 102 shows the components.

Circuit #7

Problem 5.21 (pages 189-190) of Kerr's book presents a coupled mass-spring system and its equivalent electrical circuit. In this analogy, mass is equivalent to inductance, the spring to a capacitor, the damping coefficient to resistance, and force to voltage. Figure 103 shows the mechanical system, its analogous circuit, the circuit definition and the value constraints to be asserted. Figure 104 shows the results.

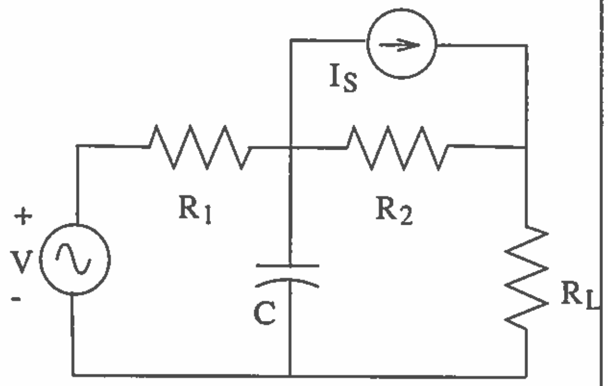
Circuit #8

Problem 3.9, pages 189-190 of Lancaster's book asks the student to show that the equivalent impedance of the circuit in Figure 105 is independent of the frequency if $R^2 = L/C$. Our system lacks the ability to the symbolic manipulation, therefore, this problem cannot be solved. What we did is to verify their assertion, by simulating the circuit at several frequencies, and check the resulting impedance. Figure 106 shows

```
Circuit component: VS
Polarities: ((R2 VS +) (R1 VS -))
SERIES-PARALLEL cluster
name: SP2
nodes: (2, 1)
Component1:
   Single VOLTAGE-SOURCE
   name: VS
   nodes: (2, 1)
   VVSVS = 1.0000
   IVSVS = 0.3333
Component2:
   SERIES cluster
   name: S1
   nodes: (2, 1)
   ZS1VS = 3.0000
   VS1VS = 1.0000
   IS1VS = 0.3333
   Component1:
      Single RESISTOR
      name: R2
      nodes: (2, 0)
      R2VS  = 2.0000
      ZR2VS = 2.0000
      VR2VS = 0.6667
      IR2VS = 0.3333
   Component2:
      Single RESISTOR
      name: R1
      nodes: (0, 1)
      R1VS  = 1.0000
      ZR1VS = 1.0000
      VR1VS = 0.3333
      IR1VS = 0.3333

Circuit component: IS
Polarities: ((R1 IS +) (R2 IS +))
SERIES-PARALLEL cluster
name: SP1
nodes: (1, 0)
Component1:
   Single CURRENT-SOURCE
   name: IS
   nodes: (1, 0)
   VISIS = 0.6667
   IISIS = 1.0000
```

```
Component2:
   PARALLEL cluster
   name: P1
   nodes: (1, 0)
   ZP1IS = 0.6667
   VP1IS = 0.6667
   IP1IS = 1.0000
   Component1:
      Single RESISTOR
      name: R1
      nodes: (1, 0)
      R1IS  = 1.0000
      ZR1IS = 1.0000
      VR1IS = 0.6667
      IR1IS = 0.6667
   Component2:
      Single RESISTOR
      name: R2
      nodes: (1, 0)
      R2IS  = 2.0000
      ZR2IS = 2.0000
      VR2IS = 0.6667
      IR2IS = 0.3333
Final values
Single CURRENT-SOURCE
name: IS
nodes: (1, 0)
VIS = 0.3333
IIS = 1.0000
Single RESISTOR
name: R1
nodes: (1, 0)
R1  = 1.0000
ZR1 = 1.0000
VR1 = 0.3333
IR1 = 0.3333
Single VOLTAGE-SOURCE
name: VS
nodes: (2, 1)
VVS = 1.0000
IVS = 0.6667
Single RESISTOR
name: R2
nodes: (2, 0)
R2  = 2.0000
ZR2 = 2.0000
VR2 = 1.3333
IR2 = 0.6667
```

FIGURE 100. Results for Circuit #5



```
(def-circuit
  :name 'ex61
  :elements '((r1 1 2)
              (c  2 0)
              (r2 2 3)
              (rl 3 0)
              (vs 1 0)
              (is 3 2))))

Asserting constraints:
((VARIABLES (VALUE W   1))
 (VARIABLES (VALUE VVS 1))
 (VARIABLES (VALUE IIS 1))
 (VARIABLES (VALUE R1  1))
 (VARIABLES (VALUE C   1))
 (VARIABLES (VALUE R2  1))
 (VARIABLES (VALUE RL  1)))
```

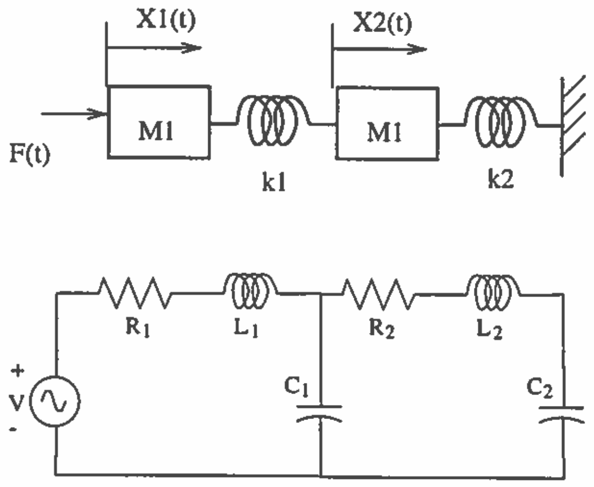
```
Final values
Single RESISTOR
name: R1
nodes: (1, 2)
R1  = 1.0000
ZR1 = 1.0000
VR1 = (0.7845 L 11.3099)
IR1 = (0.7845 L 11.3099)
Single CAPACITOR
name: C
nodes: (2, 0)
C  = 1.0000
ZC = (1.0000 L 270.0000)
VC = (0.2774 L 326.3099)
IC = (0.2774 L  56.3099)
Single RESISTOR
name: R2
nodes: (2, 3)
R2  = 1.0000
ZR2 = 1.0000
VR2 = (0.3922 L 191.3099)
IR2 = (0.3922 L 191.3099)
Single RESISTOR
name: RL
nodes: (3, 0)
RL  = 1.0000
ZRL = 1.0000
VRL = (0.6202 L 352.8750)
IRL = (0.6202 L 352.8750)
Single VOLTAGE-SOURCE
name: VS
nodes: (1, 0)
VVS = 1.0000
IVS = (0.7845 L 11.3099)
Single CURRENT-SOURCE
name: IS
nodes: (3, 2)
VIS = (0.3922 L 191.3099)
IIS = 1.0000
```

FIGURE 101. Circuit #6, Definition, Input, and Output

```
Circuit component: VS
Polarities:
 ((R1 VS +) (C VS +) (R2 VS +) (RL VS +))
SERIES-PARALLEL cluster
name: SP1
nodes: (1, 0)
Component1:
   Single VOLTAGE-SOURCE
   name: VS
   nodes: (1, 0)
   VVSVS = 1.0000
   IVSVS = (0.6202 L 29.7449)
Component2:
   SERIES cluster
   name: S2
   nodes: (1, 0)
   ZS2VS = (1.6125 L 330.2551)
   VS2VS = 1.0000
   IS2VS = (0.6202 L 29.7449)
   Component1:
      Single RESISTOR
      name: R1
      nodes: (1, 2)
      ZR1VS = 1.0000
      VR1VS = (0.6202 L 29.7449)
      IR1VS = (0.6202 L 29.7449)
   Component2:
      PARALLEL cluster
      name: P1
      nodes: (2, 0)
      ZP1VS = (0.8944 L 296.5651)
      VP1VS = (0.5547 L 326.3099)
      IP1VS = (0.6202 L  29.7449)
      Component1:
         Single CAPACITOR
         name: C
         nodes: (2, 0)
         ZCVS = (1.0000 L 270.0000)
         VCVS = (0.5547 L 326.3099)
         ICVS = (0.5547 L  56.3099)
      Component2:
         SERIES cluster
         name: S1
         nodes: (2, 0)
         ZS1VS = 2.0000
         VS1VS = (0.5547 L 326.3099)
         IS1VS = (0.2774 L 326.3099)
         Component1:
            Single RESISTOR
            name: R2
            nodes: (2, 3)
            ZR2VS = 1.0000
            VR2VS = (0.2774 L 326.3099)
            IR2VS = (0.2774 L 326.3099)
         Component2:
            Single RESISTOR
            name: RL
            nodes: (3, 0)
            ZRLVS = 1.0000
            VRLVS = (0.2774 L 326.3099)
            IRLVS = (0.2774 L 326.3099)
```

```
Circuit component: IS
Polarities:
 ((R2 IS -) (RL IS +) (R1 IS +) (C IS -))
SERIES-PARALLEL cluster
name: SP2
nodes: (3, 2)
Component1:
   Single CURRENT-SOURCE
   name: IS
   nodes: (3, 2)
   VISIS = (0.6202 L 352.8750)
   IISIS = 1.0000
Component2:
   PARALLEL cluster
   name: P3
   nodes: (3, 2)
   ZP3IS = (0.6202 L 352.8750)
   VP3IS = (0.6202 L 352.8750)
   IP3IS = 1.0000
   Component1:
      Single RESISTOR
      name: R2
      nodes: (3, 2)
      ZR2IS = 1.0000
      VR2IS = (0.6202 L 352.8750)
      IR2IS = (0.6202 L 352.8750)
   Component2:
      SERIES cluster
      name: S3
      nodes: (3, 2)
      ZS3IS = (1.5811 L 341.5651)
      VS3IS = (0.6202 L 352.8750)
      IS3IS = (0.3922 L  11.3099)
      Component1:
         Single RESISTOR
         name: RL
         nodes: (3, 0)
         ZRLIS = 1.0000
         VRLIS = (0.3922 L 11.3099)
         IRLIS = (0.3922 L 11.3099)
      Component2:
         PARALLEL cluster
         name: P2
         nodes: (0, 2)
         ZP2IS = (0.7071 L 315.0000)
         VP2IS = (0.2774 L 326.3099)
         IP2IS = (0.3922 L  11.3099)
         Component1:
            Single RESISTOR
            name: R1
            nodes: (0, 2)
            ZR1IS = 1.0000
            VR1IS = (0.2774 L 326.3099)
            IR1IS = (0.2774 L 326.3099)
         Component2:
            Single CAPACITOR
            name: C
            nodes: (0, 2)
            ZCIS = (1.0000 L 270.0000)
            VCIS = (0.2774 L 326.3099)
            ICIS = (0.2774 L  56.3099)
```

FIGURE 102. Results for Circuit #6



```
(def-circuit
  :name 'ex71
  :elements '((r1 1 2)
              (l1 2 3)
              (c1 3 0)
              (r2 3 4)
              (l2 4 5)
              (c2 5 0)
              (v  1 0)))

Asserting constraints:
((VARIABLES (VALUE W  1))
 (VARIABLES (VALUE VV 1))
 (VARIABLES (VALUE R1 1))
 (VARIABLES (VALUE L1 1))
 (VARIABLES (VALUE C1 2))
 (VARIABLES (VALUE R2 1))
 (VARIABLES (VALUE L2 1))
 (VARIABLES (VALUE C2 2)))
```

FIGURE 103. Mechanical System and Equivalent Circuit

the results for frequency $\omega = 1$; we performed the simulation, for $\omega = 10$ and $\omega = 60$ as well.

Circuit #9

Section 4.22, on page 56, of Lancaster's book provides an example that asks the student to determine the value of C such that the phase angle of the circuit is zero. Figure 107 shows the circuit definition and diagram. This is a parameter design problem; it tried solving it by using circuit analysis, asserting value constraint for all parameters, except C, indicating that the source's current's phase angle is zero. Nevertheless, the model has too many unknowns, and propagation is unable to compute a value for C. QPA can solve the problem if we provide the magnitude of the total current as well.

The problem can be solved by deriving an expression of the resulting impedance, equating its angle to zero, and solving for C. This process is normally performed using rectangular representation, which produces simpler algebraic expressions. The impedance for the parallel cluster is

$$Z_P = \frac{(R_2 + j\omega L)(-\frac{j}{\omega C})}{(R_2 + j\omega L) - (\frac{j}{\omega C})} \tag{A.66}$$

The angle of this expression is zero when its imaginary part is zero

$$\omega L^2 C - \omega L + R_2^2 \omega C = 0 \tag{A.67}$$

and solve for C. For the conditions stated in the problem, $C = 1/2$. We simulated the circuit with those values; the simulation proved that our results are correct.

```
SERIES-PARALLEL cluster
name: SP1
nodes: (1, 0)
Component1:
   Single VOLTAGE-SOURCE
   name: V
   nodes: (1, 0)
   VV = 1.0000
   IV = (0.7428 L 338.1986)
Component2:
   SERIES cluster
   name: S4
   nodes: (1, 0)
   ZS4 = (1.3463 L 21.8014)
   VS4 = 1.0000
   IS4 = (0.7428 L 338.1986)
   Component1:
      SERIES cluster
      name: S1
      nodes: (1, 3)
      ZS1 = (1.4142 L  45.0000)
      VS1 = (1.0505 L  23.1986)
      IS1 = (0.7428 L 338.1986)
      Component1:
         Single RESISTOR
         name: R1
         nodes: (1, 2)
         R1  = 1.0000
         ZR1 = 1.0000
         VR1 = (0.7428 L 338.1986)
         IR1 = (0.7428 L 338.1986)
      Component2:
         Single INDUCTOR
         name: L1
         nodes: (2, 3)
         L1 = 1.0000
         ZL1 = (1.0000 L  90.0000)
         VL1 = (0.7428 L  68.1986)
         IL1 = (0.7428 L 338.1986)
   Component2:
      PARALLEL cluster
      name: P1
      nodes: (3, 0)
      ZP1 = (0.5590 L 296.5651)
      VP1 = (0.4152 L 274.7636)
      IP1 = (0.7428 L 338.1986)
      Component1:
         Single CAPACITOR
         name: C1
         nodes: (3, 0)
         C1  = 2.0000
         ZC1 = (0.5000 L 270.0000)
         VC1 = (0.4152 L 274.7636)
         IC1 = (0.8305 L   4.7636)
      Component2:
         SERIES cluster
         name: S3
         nodes: (3, 0)
         ZS3 = (1.1180 L  26.5651)
         VS3 = (0.4152 L 274.7636)
         IS3 = (0.3714 L 248.1986)
         Component1:
            SERIES cluster
            name: S2
            nodes: (3, 5)
            ZS2 = (1.4142 L  45.0000)
            VS2 = (0.5252 L 293.1986)
            IS2 = (0.3714 L 248.1986)
            Component1:
               Single RESISTOR
               name: R2
               nodes: (3, 4)
               R2 = 1.0000
               ZR2 = 1.0000
               VR2 = (0.3714 L 248.1986)
               IR2 = (0.3714 L 248.1986)
            Component2:
               Single INDUCTOR
               name: L2
               nodes: (4, 5)
               L2 = 1.0000
               ZL2 = (1.0000 L  90.0000)
               VL2 = (0.3714 L 338.1986)
               IL2 = (0.3714 L 248.1986)
         Component2:
            Single CAPACITOR
            name: C2
            nodes: (5, 0)
            C2 = 2.0000
            ZC2 = (0.5000 L 270.0000)
            VC2 = (0.1857 L 158.1986)
            IC2 = (0.3714 L 248.1986)
```

FIGURE 104. Results for Circuit #7

```
(def-circuit
  :name 'c8
  :elements '((r1 1 2)
              (l  2 0)
              (r2 1 3)
              (c  3 0)
              (vs 1 0)))
```

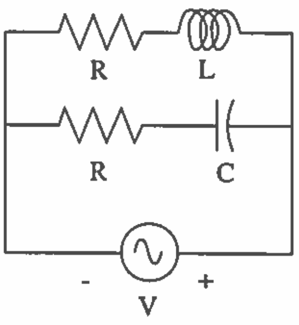


FIGURE 105. Circuit #8

```
SERIES-PARALLEL cluster
name: SP1
nodes: (1, 0)
Component1:
   Single VOLTAGE-SOURCE
   name: VS
   nodes: (1, 0)
   VVS = 1.0000
   IVS = 0.5000
Component2:
   PARALLEL cluster
   name: P1
   nodes: (1, 0)
   ZP1 = 2.0000
   VP1 = 1.0000
   IP1 = 0.5000
   Component1:
      SERIES cluster
      name: S1
      nodes: (1, 0)
      ZS1 = (240.0083  L 89.5225)
      VS1 = 1.0000
      IS1 = (  0.0042 L 270.4775)
      Component1:
         Single RESISTOR
         name: R1
         nodes: (1, 2)
         R1  = 2.0000
         ZR1 = 2.0000
         VR1 = (0.0083 L 270.4775)
         IR1 = (0.0042 L 270.4775)
      Component2:
         Single INDUCTOR
         name: L
         nodes: (2, 0)
         L  = 4.0000
         ZL = (240.0000 L  90.0000)
         VL = (  1.0000 L   0.4775)
         IL = (  0.0042 L 270.4775)
   Component2:
      SERIES cluster
      name: S2
      nodes: (1, 0)
      ZS2 = (2.0001 L 359.5225)
      VS2 = 1.0000
      IS2 = (0.5000 L   0.4775)
      Component1:
         Single RESISTOR
         name: R2
         nodes: (1, 3)
         R2  = 2.0000
         ZR2 = 2.0000
         VR2 = (1.0000 L 0.4775)
         IR2 = (0.5000 L 0.4775)
      Component2:
         Single CAPACITOR
         name: C
         nodes: (3, 0)
         C  = 1.0000
         ZC = (0.0167 L 270.0000)
         VC = (0.0083 L 270.4775)
         IC = (0.5000 L   0.4775)
```

FIGURE 106. Results for Circuit #8

```
(def-circuit
  :name 'c9
  :elements '((r1 1 2)
              (c  2 0)
              (l  2 3)
              (r2 3 0)
              (vs 1 0)))
```

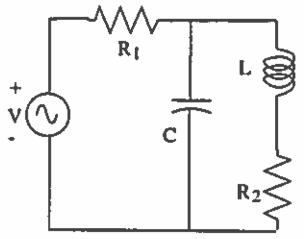


FIGURE 107. Circuit #9

Circuit #10

Chapter 10 of Walton's book *Network Analysis and Practice* studies attenuators and filters. The chapter starts by mentioning how voltage dividers work and then gives examples of divider circuits with resistors. Unfortunately, it is not possible to build a perfect resistor; all resistors include a capacitive component. Those capacitive components make voltage dividers change the reduction on magnitude and introduce a phase shift at high frequencies. The problem is solved by introducing variable capacitors in parallel with the resistors (see Figure 108), and setting them to the appropriate values to avoid phase shift. Again, doing some algebraic manipulation, it can be shown that if $R_1C_1 = R_2C_2$, the voltage divider's performance is independent of the frequency. Figure 108 shows the voltage divider circuit and its definition. Figure 109 shows an example that tests such condition for frequency $\omega = 1$; the same exercise was done with several other frequencies, and in all of them, the voltage divider's phase angle was zero, which means that the voltage drop across each resistor-capacitor combination will have a zero phase angle, therefore, dividing the voltage proportionately.

```
(def-circuit
  :name 'c9
  :elements '((r1 1 2)
              (c  2 0)
              (l  2 3)
              (r2 3 0)
              (vs 1 0)))
```

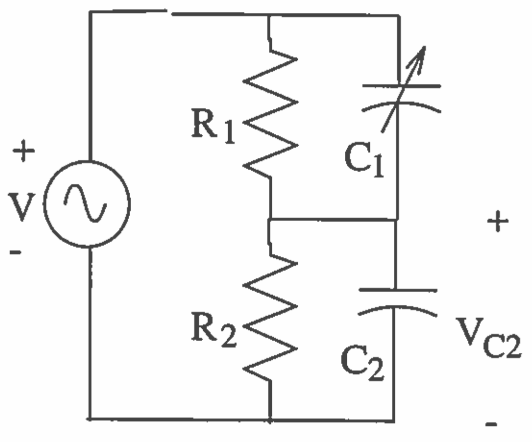


FIGURE 108. Circuit #10

```
SERIES-PARALLEL cluster
name: SP1
nodes: (1, 0)
Component1:
   Single VOLTAGE-SOURCE
   name: VS
   nodes: (1, 0)
   VVS = 1.0000
   IVS = (6.6750 L 87.1376)
Component2:
   SERIES cluster
   name: S1
   nodes: (1, 0)
   ZS1 = (0.1498 L 272.8624)
   VS1 = 1.0000
   IS1 = (6.6750 L 87.1376)
   Component1:
      PARALLEL cluster
      name: P1
      nodes: (1, 2)
      ZP1 = (0.0499 L 272.8624)
      VP1 = 0.3333
      IP1 = (6.6750 L 87.1376)
      Component1:
         Single RESISTOR
         name: R1
         nodes: (1, 2)
         R1  = 1.0000
         ZR1 = 1.0000
         VR1 = 0.3333
         IR1 = 0.3333
```

```
      Component2:
         Single CAPACITOR
         name: C1
         nodes: (1, 2)
         C1  = 2.0000
         ZC1 = (0.0500 L 270.0000)
         VC1 = 0.3333
         IC1 = (6.6667 L 90.0000)
   Component2:
      PARALLEL cluster
      name: P2
      nodes: (3, 2)
      ZP2 = (0.0999 L 272.8624)
      VP2 = 0.6667
      IP2 = (6.6750 L 87.1376)
      Component1:
         Single RESISTOR
         name: R2
         nodes: (3, 2)
         R2  = 2.0000
         ZR2 = 2.0000
         VR2 = 0.6667
         IR2 = 0.3333
      Component2:
         Single CAPACITOR
         name: C2
         nodes: (3, 2)
         C2  = 1.0000
         ZC2 = (0.1000 L 270.0000)
         VC2 = 0.6667
         IC2 = (6.6667 L  90.0000)
```

FIGURE 109. Results for Circuit #10

Power Systems Examples

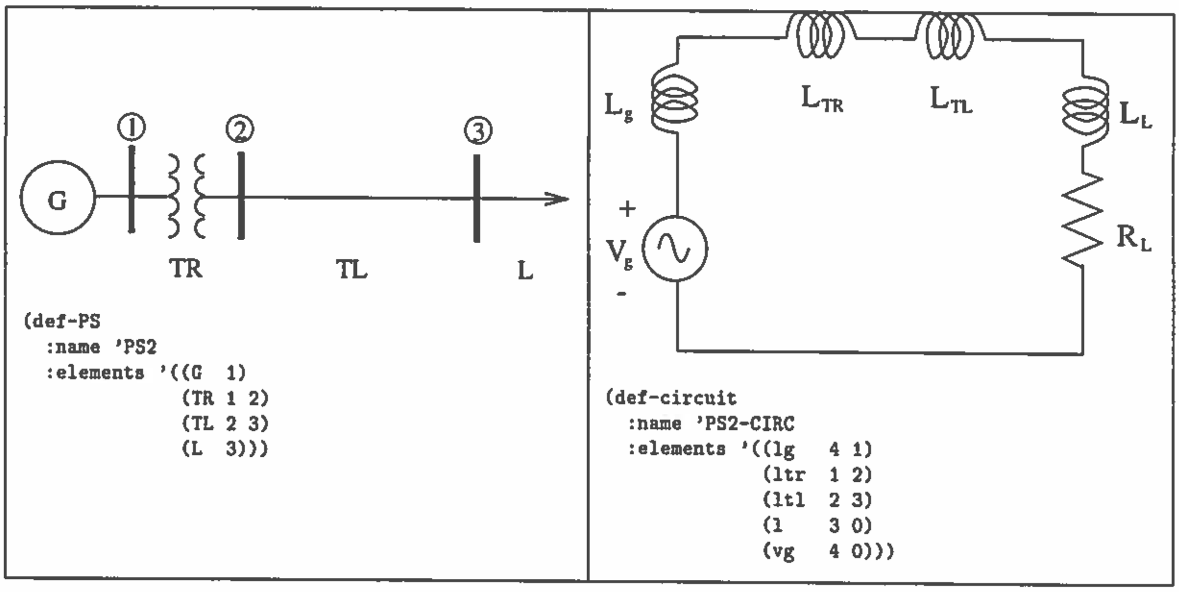
The problems in this section were extracted from Grainger's book *Power Systems Analysis* [20]. Most of the problem in Grainger's book ask for analysis under different conditions, mainly steady state and fault analysis. There are some problems where certain parameters have to be designed, normally capacitors for power factor correction, or control transformers for power distribution problems. For the control problems, we not only provided parameter design solutions, but let PSAD and QPA design the proposed solutions and then performed parameter design. In the case of fault analysis, we did not implement a power system modification function, we transformed the power system by hand and supplied it to PSAD for analysis. A fault modification function would not be difficult, since we have already developed one for the circuit module, so this extension would be straightforward.

Power System #1

The power distribution problem, mentioned in Chapter V (Figure 80 on page 152) is the same as the one mentioned in problem 2.21 on page 85 of Grainger's book. That problem was extensively studied in that chapter and is not mentioned again here. For that example, a design problem was solved (to redistribute power), each design solution was analyzed, and numerical values for the parameters of the design elements were determined. Also, causal reasoning provided a qualitative verification of effects the design elements have on the currents of the parallel transmission lines.

Power System #2

This example corresponds to an illustration (Chapter 2, page 70, of Grainger's book) about the role of transformers in power systems. In that chapter Grainger illustrates the use of circuit models to perform power systems analysis; the model generated by our system corresponds to the fault-analysis model given in the book. More complex models can be provided by the user, if the circuit needs to be analyzed at higher levels of accuracy. Figure 110 shows the power system's one-line diagram and definition, as well as its circuit equivalent and the definition generated automatically by PSAD. Note that L is a pre-clustered combination of R_L and L_L. The reason to cluster this combination before doing the general clustering, is to preserve the notion that those two elements are inseparable. Besides, this way the user can refer to the load cluster as L. Figure 111 shows the results of PSAD.



```
(def-PS
  :name 'PS2
  :elements '((G  1)
              (TR 1 2)
              (TL 2 3)
              (L  3)))
```

```
(def-circuit
  :name 'PS2-CIRC
  :elements '((lg   4 1)
              (ltr  1 2)
              (ltl  2 3)
              (l    3 0)
              (vg   4 0)))
```

FIGURE 110. Power System #2, and Its Circuit Model

Besides analysis, we performed a design task for this power system. The query was "How can we increase the power factor of the load?", which in terms of impedance means to decrease the load's impedance's phase angle. Among other solutions, PSAD determines the expected one: connect a capacitor in parallel with the load. Figure 112 shows the design task and the design solutions provided by PSAD.

```
SERIES-PARALLEL cluster
name: SP1
nodes: (4, 0)
Component1:
   Single VOLTAGE-SOURCE
   name: VG
   nodes: (4, 0)
   VVG = 1.0000
   IVG = (0.1000 L 330.0000)
Component2:
   SERIES cluster
   name: S3
   nodes: (4, 0)
   ZS3 = (10.0000 L 30.0000)
   VS3 = 1.0000
   IS3 = (0.1000 L 330.0000)
   Component1:
      SERIES cluster
      name: S2
      nodes: (4, 3)
      ZS2 = (0.3000 L  90.0000)
      VS2 = (0.0300 L  60.0000)
      IS2 = (0.1000 L 330.0000)
      Component1:
         SERIES cluster
         name: S1
         nodes: (4, 2)
         ZS1 = (0.2000 L  90.0000)
         VS1 = (0.0200 L  60.0000)
         IS1 = (0.1000 L 330.0000)
         Component1:
            Single INDUCTOR
            name: LG
            nodes: (4, 1)
            LG = 0.0017
            ZLG = (0.1000 L  90.0000)
            VLG = (0.0100 L  60.0000)
            ILG = (0.1000 L 330.0000)
```

```
         Component2:
            Single INDUCTOR
            name: LTR
            nodes: (1, 2)
            LTR = 0.0017
            ZLTR = (0.1000 L  90.0000)
            VLTR = (0.0100 L  60.0000)
            ILTR = (0.1000 L 330.0000)
      Component2:
         Single INDUCTOR
         name: LTL
         nodes: (2, 3)
         LTL = 0.0017
         ZLTL = (0.1000 L  90.0000)
         VLTL = (0.0100 L  60.0000)
         ILTL = (0.1000 L 330.0000)
   Component2:
      SERIES cluster
      name: L
      nodes: (3, 0)
      ZL = (9.8534 L  28.4891)
      VL = (0.9853 L 358.4891)
      IL = (0.1000 L 330.0000)
      Component1:
         Single RESISTOR
         name: RL
         nodes: (3, 5)
         RL  = 8.6603
         ZRL = 8.6603
         VRL = (0.8660 L 330.0000)
         IRL = (0.1000 L 330.0000)
      Component2:
         Single INDUCTOR
         name: LL
         nodes: (5, 0)
         LL = 0.0783
         ZLL = (4.7000 L  90.0000)
         VLL = (0.4700 L  60.0000)
         ILL = (0.1000 L 330.0000)
```

FIGURE 111. Results of Power System #2

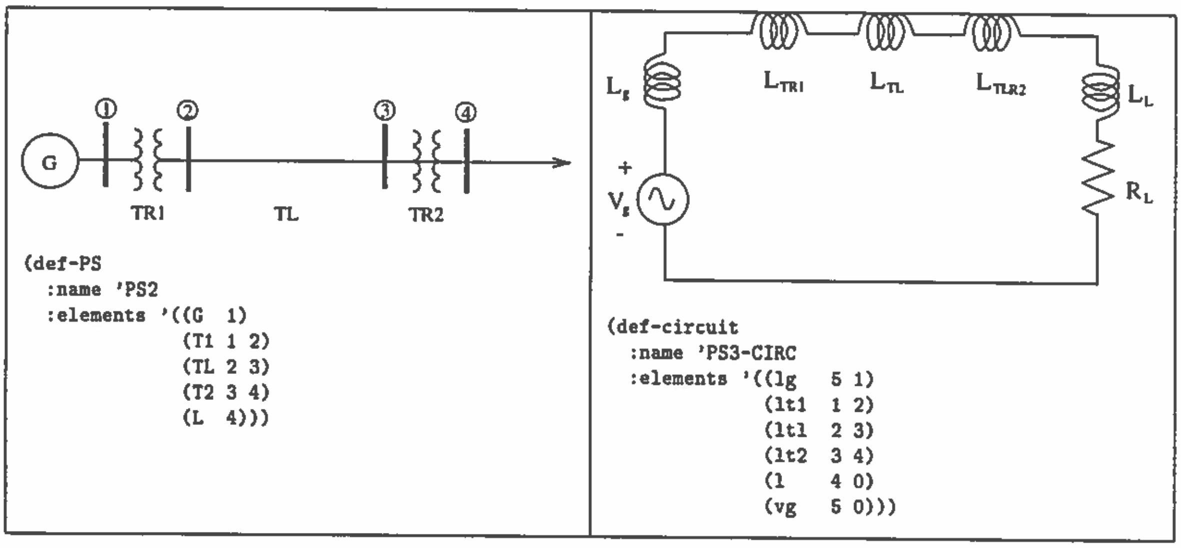
```
(design
  (def-PS-dtask
     :PS ps12
     :rem-goals
        '((confluences (value (P  ZL)
                              (? L -)))))

Solution found ...
History:
(((INDUCTIVE-CLUSTER?
    ((CONFLUENCES (VALUE (P ZL) (- L -)))
     (CONFLUENCES (VALUE (P ZRD1) (- L 0))))
    (PARALLEL L R))
  ((CONFLUENCES (VALUE (P ZL) (? L -)))))
```

```
Solution found ...
History:
(((INDUCTIVE-CLUSTER?
    ((CONFLUENCES (VALUE (P ZL) (? L -)))
     (CONFLUENCES (VALUE (P ZCD1) (- L 0))))
    (PARALLEL L C))
  ((CONFLUENCES (VALUE (P ZL) (? L -)))))
Solution found ...
History:
(((INDUCTIVE-CLUSTER?
    ((CONFLUENCES (VALUE (P ZL) (+ L -)))
     (CONFLUENCES (VALUE (P ZRD1) (+ L 0))))
    (SERIES L R))
  ((CONFLUENCES (VALUE (P ZL) (? L -)))))
```

FIGURE 112. Design Task on Power System #2

Power System #3

This example illustrates the typical situation, where a generator produces power, and a transformer raises the voltage for transmission. At the end of the transmission line, another transformer reduces the voltage, to provide usable energy to the load (page 69 of Grainger's book). For this example, we only provided power system analysis under steady state conditions. We illustrate the use of circuit models to perform power systems analysis; Figure 113 shows the power system's one-line diagram and definition, as well as its circuit equivalent and the definition generated automatically by PSAD. Figure 114 shows the results of the analysis performed by PSAD.



```
(def-PS
  :name 'PS2
  :elements '((G  1)
              (T1 1 2)
              (TL 2 3)
              (T2 3 4)
              (L  4)))
```

```
(def-circuit
  :name 'PS3-CIRC
  :elements '((lg   5 1)
              (lt1  1 2)
              (ltl  2 3)
              (lt2  3 4)
              (l    4 0)
              (vg   5 0)))
```

FIGURE 113. Power System #3, and Its Circuit Model

Power System #4

This example is a little larger in size (i.e. number of elements), and has the interesting characteristic of having two voltage sources. PSAD models this power system as a single circuit, and QPA is in charge of decomposing it into its two components. Figure 115 shows the power system's one-line diagram and definition, as well as its circuit equivalent, generated automatically by PSAD. Figure 116 shows the final results of the analysis performed by PSAD.

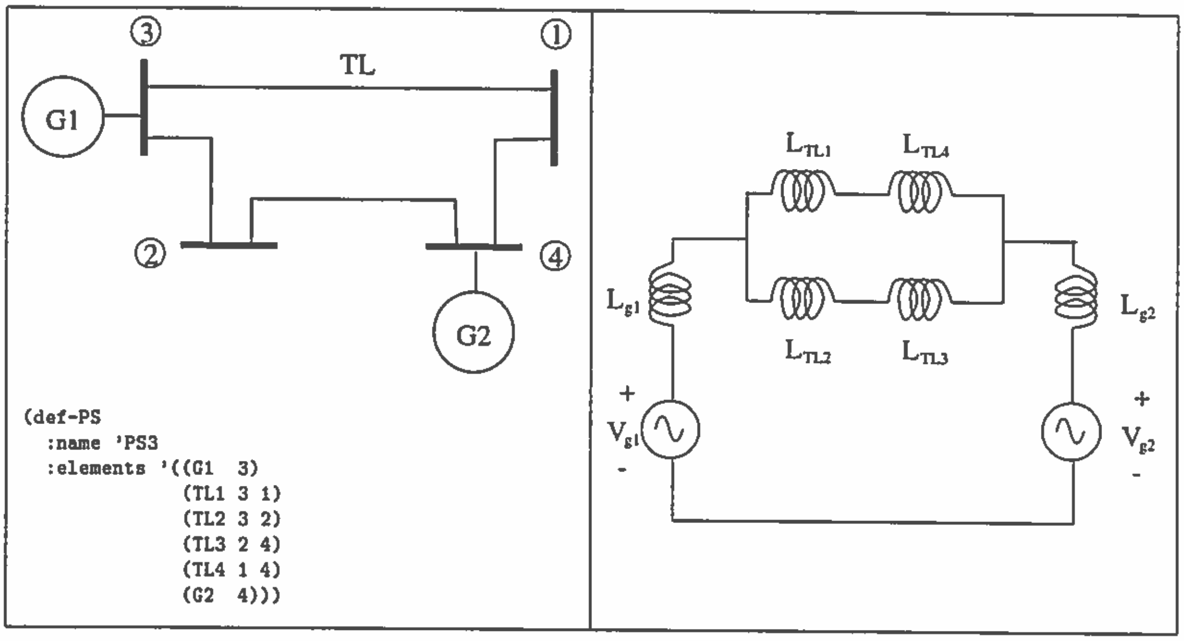
Power System #5

On Section 1.13, page 35, while explaining the role of one-line diagrams, Grainger presents the example power system of Figure 117. This example contains three sources; PSAD generates the circuit model, which is decomposed by QPA into its 3 components. Figure 116 shows the final results of the analysis of this power system,

```
SERIES-PARALLEL cluster
name: SP1
nodes: (5, 0)
Component1:
   Single VOLTAGE-SOURCE
   name: VG
   nodes: (5, 0)
   VVG = 1.0000
   IVG = (0.1000 L 330.0000)
Component2:
   SERIES cluster
   name: S4
   nodes: (5, 0)
   ZS4 = (10.0000 L 30.0000)
   VS4 = 1.0000
   IS4 = (0.1000 L 330.0000)
   Component1:
      SERIES cluster
      name: S3
      nodes: (5, 4)
      ZS3 = (0.4000 L  90.0000)
      VS3 = (0.0400 L  60.0000)
      IS3 = (0.1000 L 330.0000)
      Component1:
         SERIES cluster
         name: S2
         nodes: (5, 3)
         ZS2 = (0.3000 L  90.0000)
         VS2 = (0.0300 L  60.0000)
         IS2 = (0.1000 L 330.0000)
         Component1:
            SERIES cluster
            name: S1
            nodes: (5, 2)
            ZS1 = (0.2000 L  90.0000)
            VS1 = (0.0200 L  60.0000)
            IS1 = (0.1000 L 330.0000)
            Component1:
               Single INDUCTOR
               name: LG
               nodes: (5, 1)
               LG = 0.0017
               ZLG = (0.1000 L  90.0000)
               VLG = (0.0100 L  60.0000)
               ILG = (0.1000 L 330.0000)
```

```
            Component2:
               Single INDUCTOR
               name: LT1
               nodes: (1, 2)
               LT1 = 0.0017
               ZLT1 = (0.1000 L  90.0000)
               VLT1 = (0.0100 L  60.0000)
               ILT1 = (0.1000 L 330.0000)
         Component2:
            Single INDUCTOR
            name: LTL
            nodes: (2, 3)
            LTL = 0.0017
            ZLTL = (0.1000 L  90.0000)
            VLTL = (0.0100 L  60.0000)
            ILTL = (0.1000 L 330.0000)
      Component2:
         Single INDUCTOR
         name: LT2
         nodes: (3, 4)
         LT2 = 0.0017
         ZLT2 = (0.1000 L  90.0000)
         VLT2 = (0.0100 L  60.0000)
         ILT2 = (0.1000 L 330.0000)
   Component2:
      SERIES cluster
      name: L
      nodes: (4, 0)
      ZL = (9.8061 L  27.9756)
      VL = (0.9806 L 357.9756)
      IL = (0.1000 L 330.0000)
      Component1:
         Single RESISTOR
         name: RL
         nodes: (4, 6)
         RL  = 8.6603
         ZRL = 8.6603
         VRL = (0.8660 L 330.0000)
         IRL = (0.1000 L 330.0000)
      Component2:
         Single INDUCTOR
         name: LL
         nodes: (6, 0)
         LL = 0.0767
         ZLL = (4.6000 L  90.0000)
         VLL = (0.4600 L  60.0000)
         ILL = (0.1000 L 330.0000)
```

FIGURE 114. Results of Power System #3



```
(def-PS
  :name 'PS3
  :elements '((G1  3)
              (TL1 3 1)
              (TL2 3 2)
              (TL3 2 4)
              (TL4 1 4)
              (G2  4)))
```

FIGURE 115. Power System #4, and Its Circuit Model

produced by PSAD.

Power System #6

This is the first of several examples on short-circuit analysis of power systems. The diagram in Figure 119 (example 3.8, page 134, of Grainger's book) can be considered as a part of a power system under a short circuit. This example asks to compute the voltage at node 1, and the currents of the generators and the transformer, when the indicated fault has occurred. Figure 120 shows the polarities for all inductances, and the final results of the analysis.

Power System #7

Problem 10.8, on page 414 of Grainger's book, asks to perform short-circuit analysis on the system of Figure 121. We performed the analysis under normal operation, and then under the presence of the fault. Figures 122 and 123 show the results of both analysis.

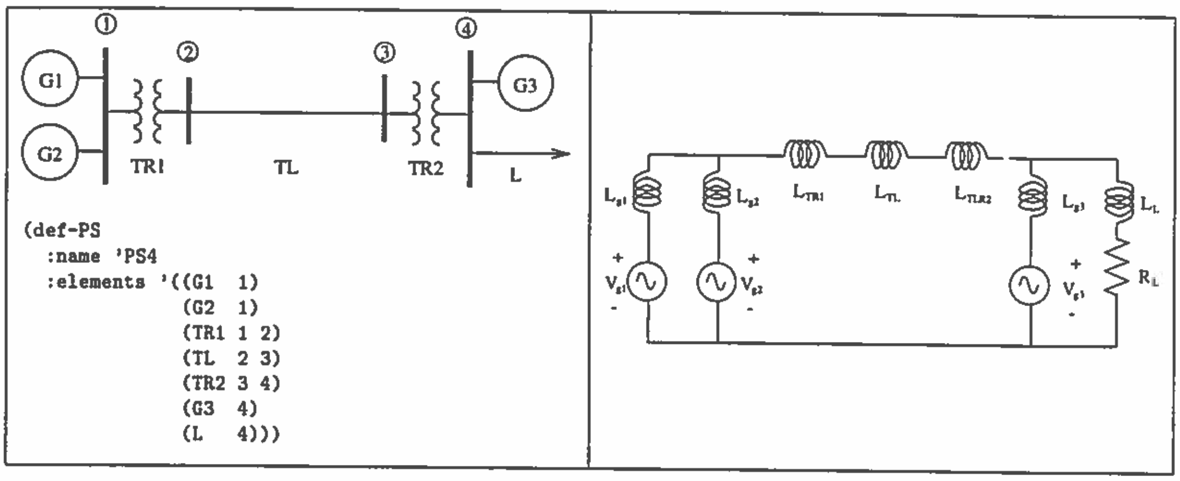
Power System #8

Problem 3.12, page 139, of Grainger's book, presents another short-circuit analysis problem (see Figure 124). This power system is of interest, because when the fault is applied, the system becomes a set of three isolated circuits (sharing the reference node). Since QPA does not handle disconnected circuits, we had to model the

```
Final values
Single VOLTAGE-SOURCE
name: VG1
nodes: (5, 0)
VVG1 = 1.0000
IVG1 = (3.3333 L 270.0000)
Single VOLTAGE-SOURCE
name: VG2
nodes: (6, 0)
VVG2 = 2.0000
IVG2 = (3.3333 L 270.0000)
SERIES cluster
name: S4
nodes: (6, 5)
ZS4 = (0.3000 L 90.0000)
VS4 = 1.0000
IS4 = (3.3333 L 270.0000)
Component1:
   Single INDUCTOR
   name: LG2
   nodes: (6, 4)
   LG2  = 0.0017
   ZLG2 = (0.1000 L 90.0000)
   VLG2 = 0.3333
   ILG2 = (3.3333 L 270.0000)
Component2:
   SERIES cluster
   name: S3
   nodes: (5, 4)
   ZS3 = (0.2000 L 90.0000)
   VS3 = 0.6667
   IS3 = (3.3333 L 270.0000)
   Component1:
      Single INDUCTOR
      name: LG1
      nodes: (5, 3)
      LG1 = 0.0017
      ZLG1 = (0.1000 L 90.0000)
      VLG1 = 0.3333
      ILG1 = (3.3333 L 270.0000)
   Component2:
      PARALLEL cluster
      name: P1
      nodes: (3, 4)
      ZP1 = (0.1000 L 90.0000)
      VP1 = 0.3333
      IP1 = (3.3333 L 270.0000)
```

```
      Component1:
         SERIES cluster
         name: S1
         nodes: (3, 4)
         ZS1 = (0.2000 L 90.0000)
         VS1 = 0.3333
         IS1 = (1.6667 L 270.0000)
         Component1:
            Single INDUCTOR
            name: LTL1
            nodes: (3, 1)
            LTL1 = 0.0017
            ZLTL1 = (0.1000 L 90.0000)
            VLTL1 = 0.1667
            ILTL1 = (1.6667 L 270.0000)
         Component2:
            Single INDUCTOR
            name: LTL4
            nodes: (1, 4)
            LTL4 = 0.0017
            ZLTL4 = (0.1000 L 90.0000)
            VLTL4 = 0.1667
            ILTL4 = (1.6667 L 270.0000)
      Component2:
         SERIES cluster
         name: S2
         nodes: (3, 4)
         ZS2 = (0.2000 L 90.0000)
         VS2 = 0.3333
         IS2 = (1.6667 L 270.0000)
         Component1:
            Single INDUCTOR
            name: LTL2
            nodes: (3, 2)
            LTL2 = 0.0017
            ZLTL2 = (0.1000 L 90.0000)
            VLTL2 = 0.1667
            ILTL2 = (1.6667 L 270.0000)
         Component2:
            Single INDUCTOR
            name: LTL3
            nodes: (2, 4)
            LTL3 = 0.0017
            ZLTL3 = (0.1000 L 90.0000)
            VLTL3 = 0.1667
            ILTL3 = (1.6667 L 270.0000)
```

FIGURE 116. Results of Power System #4



```
(def-PS
  :name 'PS4
  :elements '((G1  1)
              (G2  1)
              (TR1 1 2)
              (TL  2 3)
              (TR2 3 4)
              (G3  4)
              (L   4)))
```

FIGURE 117. Power System #5, and Its Circuit Model

three parts of the power system independently. We believe that the modifications required to make QPA work with disconnected circuits are minimum, but they have not been implemented yet. Figure 125 shows the results of the analysis under normal conditions, and Figure 126 shows the results for the three sources. The problem asks to draw the impedance diagram; PSAD does not draw it, but it determines its components and interconnections correctly. The second part asks to determine the short-circuit currents for each generator; those are shown in the partial results for the disconnected components. Finally, the problem asks to compute the power supplied by each generator. Our system does not model power, but the indicated power quantities can be computed directly, using the phasor values for voltage and current on each generator.

Power System #9

Problem 3.13, pages 139-140, of Grainger's book, presents another short-circuit analysis problem (see Figure 127). This problem illustrates one of the limitations of our system: the system's model is not series/parallel reducible. Even though the system's model in normal operation is not series/parallel reducible, under faulty conditions, it is reducible, and can be analyzed by QPA. Figure 128 shows the results of the analysis of the part containing the two generators. The problem asks to compute the short-circuit currents and voltages for both generators.

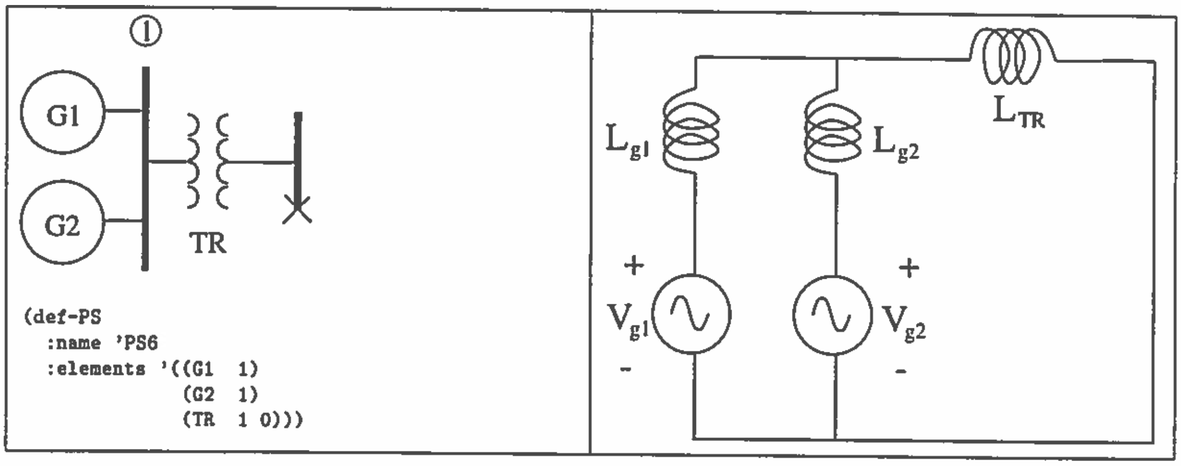
Power System #10

This last example is the largest we considered. We are not sure about the origin of this example; it was provided in personal communication with Alberto González Avalos. This power system contains 21 power systems elements, including three sources (see Figure 129); the circuit model, produced by PSAD had 26 elements,

```
Final values
Single VOLTAGE-SOURCE
name: VG1
nodes: (5, 0)
VVG1 = 1.0000
IVG1 = (5.5634 L 270.2258)
Single INDUCTOR
name: LG1
nodes: (5, 1)
LG1 = 0.0017
ZLG1 = (0.1000 L 90.0000)
VLG1 = (0.0209 L 70.4269)
ILG1 = (0.2094 L 340.4269)
Single VOLTAGE-SOURCE
name: VG2
nodes: (6, 0)
VVG2 = 1.0000
IVG2 = (5.5634 L 270.2258)
Single INDUCTOR
name: LG2
nodes: (6, 1)
LG2 = 0.0017
ZLG2 = (0.1000 L 90.0000)
VLG2 = (0.0209 L 70.4269)
ILG2 = (0.2094 L 340.4269)
Single VOLTAGE-SOURCE
name: VG3
nodes: (7, 0)
VVG3 = 1.0000
IVG3 = (2.8169 L 292.4141)
Single INDUCTOR
name: LG3
nodes: (7, 4)
LG3 = 0.0017
ZLG3 = (0.1000 L 90.0000)
VLG3 = (0.1466 L 70.4269)
ILG3 = (1.4657 L 340.4269)
SERIES cluster
name: L
nodes: (4, 0)
ZL = (0.5099 L 11.3099)
VL = (0.9609 L 351.7368)
IL = (1.8844 L 340.4269)
Component1:
   Single RESISTOR
   name: RL
   nodes: (4, 8)
   RL = 0.5000
   ZRL = 0.5000
   VRL = (0.9422 L 340.4269)
   IRL = (1.8844 L 340.4269)
```

```
Component2:
   Single INDUCTOR
   name: LL
   nodes: (8, 0)
   LL = 0.0017
   ZLL = (0.1000 L 90.0000)
   VLL = (0.1884 L 70.4269)
   ILL = (1.8844 L 340.4269)
SERIES cluster
name: S2
nodes: (1, 4)
ZS2 = (0.3000 L 90.0000)
VS2 = (0.1256 L 70.4269)
IS2 = (0.4188 L 340.4269)
Component1:
   SERIES cluster
   name: S1
   nodes: (1, 3)
   ZS1 = (0.2000 L 90.0000)
   VS1 = (0.0838 L 70.4269)
   IS1 = (0.4188 L 340.4269)
   Component1:
      Single INDUCTOR
      name: LTR1
      nodes: (1, 2)
      LTR1 = 0.0017
      ZLTR1 = (0.1000 L 90.0000)
      VLTR1 = (0.0419 L 70.4269)
      ILTR1 = (0.4188 L 340.4269)
   Component2:
      Single INDUCTOR
      name: LTL
      nodes: (2, 3)
      LTL = 0.0017
      ZLTL = (0.1000 L 90.0000)
      VLTL = (0.0419 L 70.4269)
      ILTL = (0.4188 L 340.4269)
Component2:
   Single INDUCTOR
   name: LTR2
   nodes: (3, 4)
   LTR2 = 0.0017
   ZLTR2 = (0.1000 L 90.0000)
   VLTR2 = (0.0419 L 70.4269)
   ILTR2 = (0.4188 L 340.4269)
```

FIGURE 118. Results of Power System #5

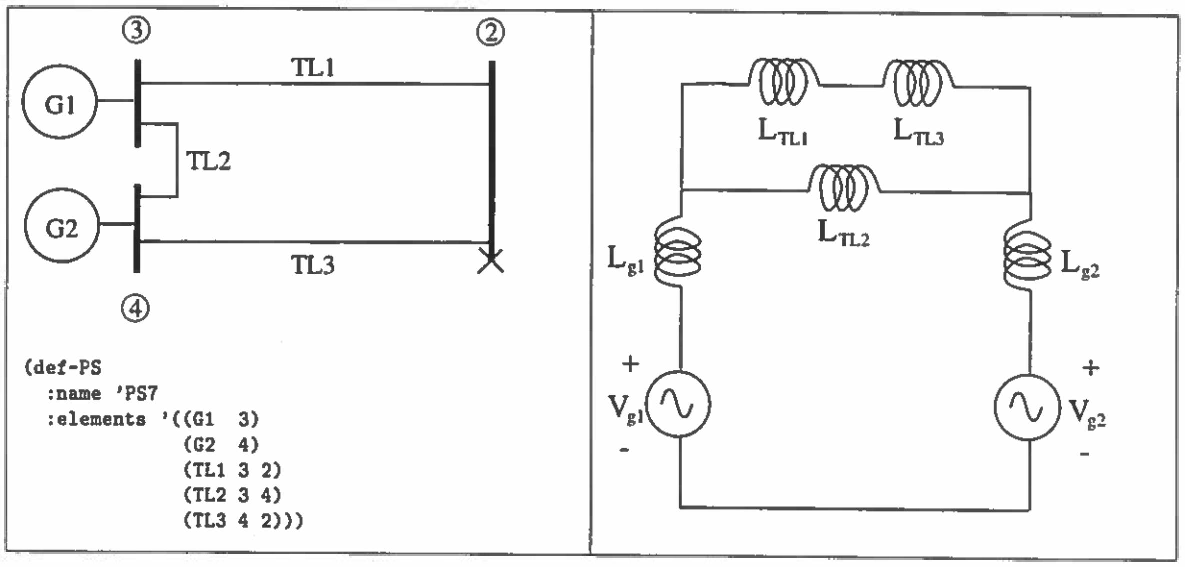


```
(def-PS
  :name 'PS6
  :elements '((G1  1)
              (G2  1)
              (TR  1 0)))
```

FIGURE 119. Power System #6, and Its Circuit Model

```
Circuit component: VG2
Polarities: ((LG2 VG2 +) (LG1 VG2 -) (LTR VG2 +))
Circuit component: VG1
Polarities: ((LG1 VG1 +) (LG2 VG1 -) (LTR VG1 +))

Final values
Single VOLTAGE-SOURCE
name: VG1
nodes: (2, 0)
VVG1 = 0.9570
IVG1 = (1.8241 L 270.0000)
Single INDUCTOR
name: LG1
nodes: (2, 1)
LG1 = 0.0063
ZLG1 = (0.3750 L 90.0000)
VLG1 = 0.6840
ILG1 = (1.8241 L 270.0000)
Single VOLTAGE-SOURCE
name: VG2
nodes: (3, 0)
VVG2 = 0.9520
IVG2 = (0.9054 L 270.0000)
Single INDUCTOR
name: LG2
nodes: (3, 1)
LG2 = 0.0125
ZLG2 = (0.7500 L 90.0000)
VLG2 = 0.6790
ILG2 = (0.9054 L 270.0000)
Single INDUCTOR
name: LTR
nodes: (1, 0)
LTR = 0.0017
ZLTR = (0.1000 L 90.0000)
VLTR = 0.2730
ILTR = (2.7295 L 270.0000)
```

FIGURE 120. Results of Power System #6



```
(def-PS
  :name 'PS7
  :elements '((G1  3)
              (G2  4)
              (TL1 3 2)
              (TL2 3 4)
              (TL3 4 2)))
```

FIGURE 121. Power System #7, and Its Circuit Model

including the voltage sources; its constraint-based model contains 2458 constraints. PSAD took 02:02:42.267 user time, of which 27,784 msec are cpu time, to do constraint propagation after the assertion of values to variables. The result of this example is not displayed in this appendix because of its length.

```
Final values
Single VOLTAGE-SOURCE
name: VG1
nodes: (5, 0)
VVG1 = (1.0000 L 270.0000)
IVG1 = (0.3609 L  42.8047)
Single VOLTAGE-SOURCE
name: VG2
nodes: (6, 0)
VVG2 = (0.6800 L 225.0000)
IVG2 = (0.3609 L  42.8047)
SERIES cluster
name: S3
nodes: (6, 5)
ZS3 = (1.9609 L  90.0000)
VS3 = (0.7076 L 132.8047)
IS3 = (0.3609 L  42.8047)
Component1:
   Single INDUCTOR
   name: LG2
   nodes: (6, 4)
   LG2 = 0.0133
   ZLG2 = (0.8000 L  90.0000)
   VLG2 = (0.2887 L 132.8047)
   ILG2 = (0.3609 L  42.8047)
Component2:
   SERIES cluster
   name: S2
   nodes: (5, 4)
   ZS2 = (1.1609 L  90.0000)
   VS2 = (0.4189 L 132.8047)
   IS2 = (0.3609 L  42.8047)
   Component1:
      Single INDUCTOR
      name: LG1
      nodes: (5, 3)
      LG1 = 0.0133
      ZLG1 = (0.8000 L  90.0000)
      VLG1 = (0.2887 L 132.8047)
      ILG1 = (0.3609 L  42.8047)
```

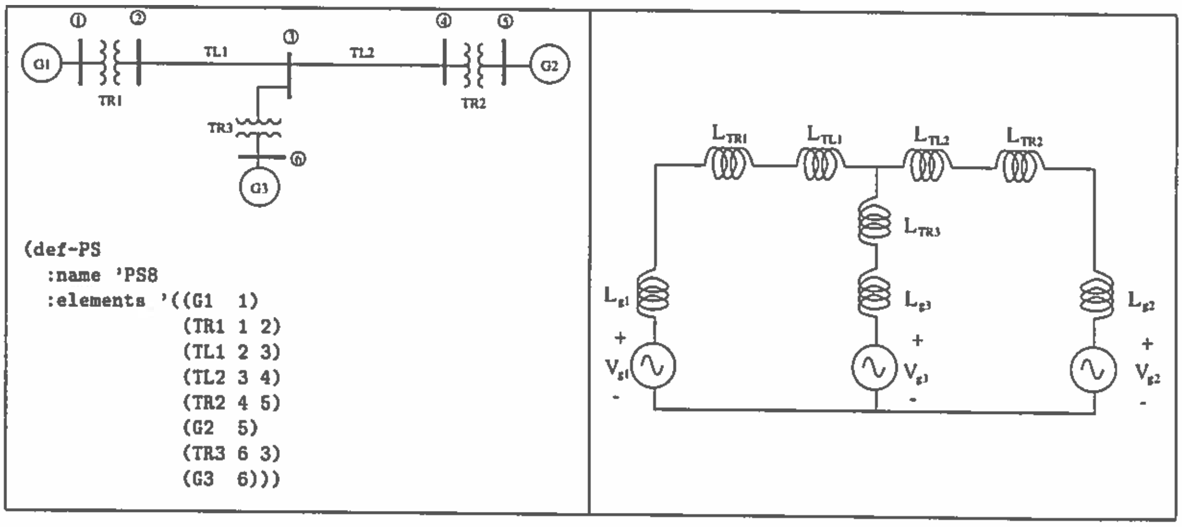
```
   Component2:
      PARALLEL cluster
      name: P1
      nodes: (3, 4)
      ZP1 = (0.3609 L  90.0000)
      VP1 = (0.1302 L 132.8047)
      IP1 = (0.3609 L  42.8047)
      Component1:
         Single INDUCTOR
         name: LTL2
         nodes: (3, 4)
         LTL2 = 0.0062
         ZLTL2 = (0.3731 L  90.0000)
         VLTL2 = (0.1302 L 132.8047)
         ILTL2 = (0.3490 L  42.8047)
      Component2:
         SERIES cluster
         name: S1
         nodes: (3, 4)
         ZS1 = (11.0075 L 90.0000)
         VS1 = (0.1302 L 132.8047)
         IS1 = (0.0118 L  42.8047)
         Component1:
            Single INDUCTOR
            name: LTL1
            nodes: (3, 2)
            LTL1 = 0.0709
            ZLTL1 = (4.2537 L  90.0000)
            VLTL1 = (0.0503 L 132.8047)
            ILTL1 = (0.0118 L  42.8047)
         Component2:
            Single INDUCTOR
            name: LTL3
            nodes: (4, 2)
            LTL3 = 0.1126
            ZLTL3 = (6.7537 L  90.0000)
            VLTL3 = (0.0799 L 132.8047)
            ILTL3 = (0.0118 L  42.8047)
```

FIGURE 122. Analysis of Power System #7 under Normal Conditions

```
Final values
Single VOLTAGE-SOURCE
name: VG1
nodes: (5, 0)
VVG1 = (1.0000 L 270.0000)
IVG1 = (0.6000 L 180.0000)
Single INDUCTOR
name: LG1
nodes: (5, 3)
LG1 = 0.0133
ZLG1 = (0.8000 L  90.0000)
VLG1 = (0.3559 L 297.4967)
ILG1 = (0.4449 L 207.4967)
Single VOLTAGE-SOURCE
name: VG2
nodes: (6, 0)
VVG2 = (0.6800 L 225.0000)
IVG2 = (0.3986 L 135.0000)
Single INDUCTOR
name: LG2
nodes: (6, 4)
LG2 = 0.0133
ZLG2 = (0.8000 L  90.0000)
VLG2 = (0.2537 L 152.7218)
ILG2 = (0.3171 L  62.7218)
```

```
Single INDUCTOR
name: LTL1
nodes: (3, 0)
LTL1 = 0.0709
ZLTL1 = (4.2537 L  90.0000)
VLTL1 = (0.7038 L 256.4979)
ILTL1 = (0.1654 L 166.4979)
Single INDUCTOR
name: LTL2
nodes: (3, 4)
LTL2 = 0.0062
ZLTL2 = (0.3731 L  90.0000)
VLTL2 = (0.1261 L 316.2330)
ILTL2 = (0.3379 L 226.2330)
Single INDUCTOR
name: LTL3
nodes: (4, 0)
LTL3 = 0.1126
ZLTL3 = (6.7537 L  90.0000)
VLTL3 = (0.6494 L 246.8445)
ILTL3 = (0.0962 L 156.8445)
```

FIGURE 123. Short-Circuit Analysis of Power System #7



```
(def-PS
  :name 'PS8
  :elements '((G1  1)
              (TR1 1 2)
              (TL1 2 3)
              (TL2 3 4)
              (TR2 4 5)
              (G2  5)
              (TR3 6 3)
              (G3  6)))
```

FIGURE 124. Power System #8, and Its Circuit Model

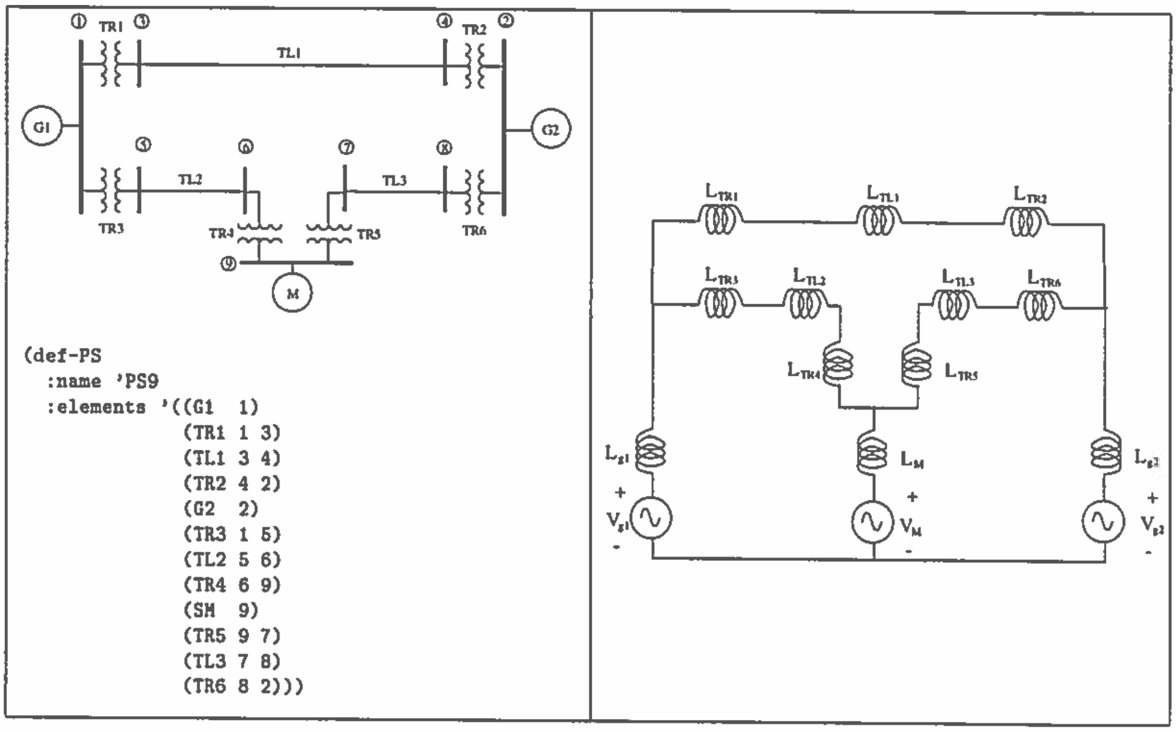
```
Final values
Single VOLTAGE-SOURCE
name: VG1
nodes: (7, 0)
VVG1 = 0.6900
IVG1 = (0.0465 L 90.0000)
Single VOLTAGE-SOURCE
name: VG2
nodes: (8, 0)
VVG2 = 0.9000
IVG2 = (0.0106 L 90.0000)
Single VOLTAGE-SOURCE
name: VG3
nodes: (9, 0)
VVG3 = 1.0000
IVG3 = (0.0571 L 270.0000)
SERIES cluster
name: S2
nodes: (7, 3)
ZS2 = (6.3000 L  90.0000)
VS2 = (0.2929 L 180.0000)
IS2 = (0.0465 L  90.0000)
Component1:
   SERIES cluster
   name: S1
   nodes: (7, 2)
   ZS1 = (0.3000 L  90.0000)
   VS1 = (0.0139 L 180.0000)
   IS1 = (0.0465 L  90.0000)
   Component1:
      Single INDUCTOR
      name: LG1
      nodes: (7, 1)
      ZLG1 = (0.2000 L  90.0000)
      VLG1 = (0.0093 L 180.0000)
      ILG1 = (0.0465 L  90.0000)
   Component2:
      Single INDUCTOR
      name: LTR1
      nodes: (1, 2)
      ZLTR1 = (0.1000 L  90.0000)
      VLTR1 = (0.0046 L 180.0000)
      ILTR1 = (0.0465 L  90.0000)
Component2:
   Single INDUCTOR
   name: LTL1
   nodes: (2, 3)
   ZLTL1 = (6.0000 L  90.0000)
   VLTL1 = (0.2789 L 180.0000)
   ILTL1 = (0.0465 L  90.0000)
```

```
SERIES cluster
name: S4
nodes: (8, 3)
ZS4 = (7.8000 L  90.0000)
VS4 = (0.0829 L 180.0000)
IS4 = (0.0106 L  90.0000)
Component1:
   Single INDUCTOR
   name: LG2
   nodes: (8, 5)
   ZLG2 = (0.2000 L  90.0000)
   VLG2 = (0.0021 L 180.0000)
   ILG2 = (0.0106 L  90.0000)
Component2:
   SERIES cluster
   name: S3
   nodes: (3, 5)
   ZS3 = (7.6000 L  90.0000)
   VS3 = (0.0807 L 180.0000)
   IS3 = (0.0106 L  90.0000)
   Component1:
      Single INDUCTOR
      name: LTL2
      nodes: (3, 4)
      ZLTL2 = (7.5000 L  90.0000)
      VLTL2 = (0.0797 L 180.0000)
      ILTL2 = (0.0106 L  90.0000)
   Component2:
      Single INDUCTOR
      name: LTR2
      nodes: (4, 5)
      ZLTR2 = (0.1000 L  90.0000)
      VLTR2 = (0.0011 L 180.0000)
      ILTR2 = (0.0106 L  90.0000)
SERIES cluster
name: S5
nodes: (9, 3)
ZS5 = (0.3000 L 90.0000)
VS5 = 0.0171
IS5 = (0.0571 L 270.0000)
Component1:
   Single INDUCTOR
   name: LG3
   nodes: (9, 6)
   ZLG3 = (0.2000 L 90.0000)
   VLG3 = 0.0114
   ILG3 = (0.0571 L 270.0000)
Component2:
   Single INDUCTOR
   name: LTR3
   nodes: (6, 3)
   ZLTR3 = (0.1000 L 90.0000)
   VLTR3 = 0.0057
   ILTR3 = (0.0571 L 270.0000)
```

FIGURE 125. Results of Power System #8

```
Single VOLTAGE-SOURCE
name: VG1
nodes: (3, 0)
VVG1 = 0.6900
IVG1 = (0.1095 L 270.0000)

Single VOLTAGE-SOURCE
name: VG2
nodes: (6, 0)
VVG2 = 0.9000
IVG2 = (0.1154 L 270.0000)

Single VOLTAGE-SOURCE
name: VG3
nodes: (7, 0)
VVG3 = 1.0000
IVG3 = (3.3333 L 270.0000)
```

FIGURE 126. Short-Circuit Values for Sources of Power System #8



```
(def-PS
  :name 'PS9
  :elements '((G1  1)
              (TR1 1 3)
              (TL1 3 4)
              (TR2 4 2)
              (G2  2)
              (TR3 1 5)
              (TL2 5 6)
              (TR4 6 9)
              (SM  9)
              (TR5 9 7)
              (TL3 7 8)
              (TR6 8 2)))
```

FIGURE 127. Power System #9, and Its Circuit Model

```
Final values
Single VOLTAGE-SOURCE
name: VG1
nodes: (9, 0)
VVG1 = 1.0000
IVG1 = (0.4715 L 270.0000)
Single INDUCTOR
name: LG1
nodes: (9, 1)
ZLG1 = (0.2000 L 90.0000)
VLG1 = 0.0943
ILG1 = (0.4715 L 270.0000)
Single VOLTAGE-SOURCE
name: VG2
nodes: (10, 0)
VVG2 = 2.0000
IVG2 = (1.3639 L 270.0000)
Single INDUCTOR
name: LG2
nodes: (10, 2)
ZLG2 = (0.2000 L 90.0000)
VLG2 = 0.2728
ILG2 = (1.3639 L 270.0000)
SERIES cluster
name: S2
nodes: (1, 2)
ZS2 = (5.1382 L  90.0000)
VS2 = (0.8215 L 180.0000)
IS2 = (0.1599 L  90.0000)
Component1:
   SERIES cluster
   name: S1
   nodes: (1, 4)
   ZS1 = (5.0382 L  90.0000)
   VS1 = (0.8055 L 180.0000)
   IS1 = (0.1599 L  90.0000)
   Component1:
      Single INDUCTOR
      name: LTR1
      nodes: (1, 3)
      ZLTR1 = (0.1000 L  90.0000)
      VLTR1 = (0.0160 L 180.0000)
      ILTR1 = (0.1599 L  90.0000)
   Component2:
      Single INDUCTOR
      name: LTL1
      nodes: (3, 4)
      ZLTL1 = (4.9382 L  90.0000)
      VLTL1 = (0.7895 L 180.0000)
      ILTL1 = (0.1599 L  90.0000)
Component2:
   Single INDUCTOR
   name: LTR2
   ...
```

```
SERIES cluster
name: S4
nodes: (1, 0)
ZS4 = (1.4345 L 90.0000)
VS4 = 0.9057
IS4 = (0.6314 L 270.0000)
Component1:
   SERIES cluster
   name: S3
   nodes: (1, 6)
   ZS3 = (1.3345 L 90.0000)
   VS3 = 0.8426
   IS3 = (0.6314 L 270.0000)
   Component1:
      Single INDUCTOR
      name: LTR3
      nodes: (1, 5)
      ZLTR3 = (0.1000 L 90.0000)
      VLTR3 = 0.0631
      ILTR3 = (0.6314 L 270.0000)
   Component2:
      Single INDUCTOR
      name: LTL2
      nodes: (5, 6)
      ZLTL2 = (1.2345 L 90.0000)
      VLTL2 = 0.7794
      ILTL2 = (0.6314 L 270.0000)
Component2:
   Single INDUCTOR
   name: LTR4
   nodes: (6, 0)
   ZLTR4 = (0.1000 L 90.0000)
   VLTR4 = 0.0631
   ILTR4 = (0.6314 L 270.0000)
SERIES cluster
name: S6
nodes: (0, 2)
ZS6 = (1.4345 L  90.0000)
VS6 = (1.7272 L 180.0000)
IS6 = (1.2041 L  90.0000)
Component1:
   SERIES cluster
   name: S5
   nodes: (0, 8)
   ZS5 = (1.3345 L  90.0000)
   VS5 = (1.6068 L 180.0000)
   IS5 = (1.2041 L  90.0000)
   Component1:
      Single INDUCTOR
      name: LTR5
      nodes: (0, 7)
      ZLTR5 = (0.1000 L  90.0000)
      VLTR5 = (0.1204 L 180.0000)
      ILTR5 = (1.2041 L  90.0000)
   Component2:
      Single INDUCTOR
      name: LTL3
      ...
```

FIGURE 128. Results of Power System #9

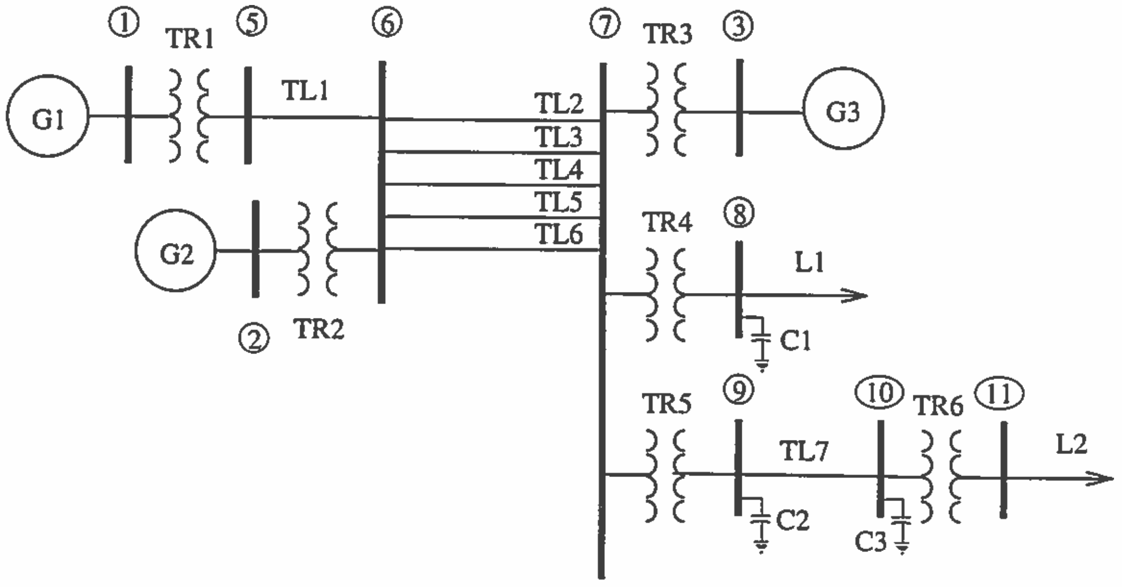


FIGURE 129. Power System #10

BIBLIOGRAPHY

[1] Gotz Alefeld and Jurgen Herzberger. *Introduction to Interval Computation.* Academic Press, New York, 1983.

[2] Hans Bandemer, ed. *Modelling Uncertain Data.* Akademie Verlag, Berlin, Germany, 1993.

[3] F. Benhamou, D. Mc Allester, and P. Van Hentenryck. CLP(Intervals) revisited. In *Logic Programming, Proceedings of the 1994 International Symposium*, 124–138, Ithaca, New York, November 1994.

[4] W. E. Boyce and R. C. DiPrima. *Elementary Differential Equations.* John Wiley, New York, second edition, 1969.

[5] Jacques Cohen. Constraint logic programming languages. *Comm. of the ACM*, 33:52–68, 1990.

[6] Thomas H. Cormen, Charles E. Leiserson, and Ronald L. Rivest. *Introduction to Algorithms.* McGraw-Hill, USA, 1989.

[7] E. Davis. Constraint propagation with interval labels. *Artificial Intelligence*, 24:347–410, 1987.

[8] E. Davis. A logical framework for solid object physics. In *Qualitative Reasoning Workshop Abstracts.* Qualitative Reasoning Group, University of Illinois at Urbana-Champaign, 1987.

[9] R. Davis. Diagnostic reasoning based on structure and behavior. *Artificial Intelligence*, 24:347–410, 1984.

[10] Johan de Kleer. How circuits work. *Artificial Intelligence*, 24:205–280, 1984.

[11] Johan de Kleer. An assumption-based tms. *Artificial Intelligence*, 28:127–162, 1986.

[12] Johan de Kleer and John Seely Brown. Qualitative physics based on confluences. *Artificial Intelligence*, 24:7–83, 1984. Also in *Readings in Knowledge Representation*, Brachman and Levesque, editors, Morgan Kaufmann, 1985, 88-126.

[13] T. L. Dean and D. V. McDermott. Temporal data base management. *Artificial Intelligence*, 32:1–55, 1987.

[14] Daniel Dvorak and Benjamin Kuipers. Model-based monitoring of dynamic systems. In *Proc. 11th Int. Joint Conf. on Artificial Intelligence (IJCAI-89)*, 1238–1243, San Mateo, CA, 1989. Morgan Kaufmann.

[15] A. E. Fitzgerald. *Basic Electrical Engineering*. McGraw-Hill, New York and London, 1945.

[16] Juan J. Flores. Hybrid representation constraint propagation. In *In Proceedings of the Nineth International Symposium on Artificial Intelligence*, 340–347, Cancun, Mexico, November 1996.

[17] Kenneth D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24:85–168, 1984.

[18] M. R. Genesereth. The use of design descriptions in automated diagnosis. *Artificial Intelligence*, 24:411–436, 1984.

[19] Turan Gönen. *Modern Power System Analysis*. John Wiley and Sons, New York, 1988.

[20] John J. Grainger and William D. Stevenson. *Power System Analysis*. McGraw-Hill, New York, 1994.

[21] Per Hagman. Using qualitative reasoning to solve voltage decay problems in a model-based automated diagnostic system. Master's thesis, Department of Computer Engineering, University of Central Florida, Orlando, Florida, 1996.

[22] W.C. Hamscher. Modeling digital circuits for troubleshooting. *Artificial Intelligence*, 51:223–271, 1991.

[23] Nevin Heintze and Spiro Michalylov. CLP(R) and some electrical engineering problems. *Journal of Automated Reasoning*, 9:321–260, 1992.

[24] Eero Hyvönen. Constraint reasoning based on interval arithmetic. In *Proc. 11th Int. Joint Conf. on Artificial Intelligence (IJCAI-89)*, 1193–1198, 1989.

[25] P Stuckey J Jaffar, S Michaylov and R H C Yap. The CLP(R) language and system: an overview. In *Digest of papers, Spring COMPCON 91, Thirty-sixth IEEE Computer Society International Conf*, 376–381, 1991.

[26] R. Baker Kearfott. Algorithm 763: Interval arithmetic: A FORTRAN 90 module for an interval data type. *Transcation on Mathematical Software*, 22(4):385–392, 1996.

[27] Robert B. Kerr. *Electrical Network Science.* Prentice-Hall, Englewood Cliffs, NJ, 1977.

[28] Benjamin J. Kuipers. The limits of qualitative simulation. In *Proc. 9th Int. Joint Conf. on Artificial Intelligence (IJCAI-85)*, 128–136, San Mateo, CA, 1985. Morgan Kaufmann.

[29] V. Kumar. Algorithms for constraint-satisfaction problems: A survey. *AI Magazine*, 13(1):32–44, 1992.

[30] Gordon Lancaster. *DC and AC Circuits.* Calendar Press, Oxford, 1974.

[31] Z. Liu and A. Farley. Shifting ontological perspectives in reasoning about physical systems. In *Proc. 8th National Conf. on Artificial Intelligence (AAAI-90)*, Menlo Park, Cambridge, London, 1990. AAAI Press/The MIT Press.

[32] Zheng-Yang Liu. Qualitative reasoning about physical systems with multiple perspectives. Technical Report CIS-TR-91-04, University of Oregon, 1991.

[33] Michael L. Mavrovouniotis and George Stephanopoulos. Order-of-magnitude reasoning with O[M]. *International Journal of Artificial Intelligence in Engineering*, 4(3):106–114, 1989.

[34] Drew McDermott. A general framework for reason maintenance. *Artificial Intelligence*, 50:289–329, 1991.

[35] Ramon E. Moore. *Interval Analysis.* Prentice-Hall, Inc., Englewood Cliffs, NJ, 1966.

[36] G. Pettersson. Impedance driven model-based diagnosis of electic power distribution system faults. Master's thesis, Department of Computer Engineering, University of Central Florida, Orlando, Florida, 1996.

[37] Jean-Francois Puget. A C++ implementation of CLP. In *Proceedings of SPICIS 94*, Singapore, November 1994.

[38] Archana Shankar, David Gilbert, and Michael Jampel. Transient analysis of linear circuits using constraint logic programming. In *Proceedings of the International Conference on Practical Applications of Constraint Technology*, 221–247, London, November 1996.

[39] W. Shepherd and P. Zand. *Energy Flow and Power Factor in Nonsinusoidal Circuits.* Cambridge University Press, Cambridge, MA, 1979.

[40] Jeffrey Mark Siskind and David Allen McAllester. Nondeterministic LISP as a substrate for constraint logic programming. In *Proc. 11th National Conf. on Artificial Intelligence (AAAI-93)*, 133–138, 1993.

[41] R. Stallman and G. J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9:135–196, 1977.

[42] Peter Struss. An application of model simplification and abstraction to fault localization in power transmission networks. In *Working notes of the workshop Approximaion and Abstraction of Computational Theories*, 205–212, 1992.

[43] Peter Struss. A theory of simplification and abstraction for relational models. In *Working notes of the workshop Approximaion and Abstraction of Computational Theories*, 213–226, 1992.

[44] Siddarth Subramanian and Raymond J. Mooney. Qualitative multiple-fault diagnosis of continuous dynamic systems using behavioral modes. In *Proc. 13th National Conf. on Artificial Intelligence (AAAI-96)*, 965–970, Menlo Park, Cambridge, London, 1996. AAAI Press/The MIT Press.

[45] G. J. Sussman and G. L. Steele. CONSTRAINTS: a language for expressing almost-hierarchical descriptions. *Artificial Intelligence*, 14:1–39, 1980.

[46] Earl W. Swokowski. *Fundamentals of Algebra and Trigonometry.* Prindle, Weber and Schmidt, Inc., Boston, Massachusetts, 1975.

[47] Edward Tsang. *Foundations of Constraint Satisfaction.* Academic Press Limited, London, 1993.

[48] Richard J. Wallace and Eugene C. Freuder. Anytime algorithms for constraint satisfaction and sat problems. *SIGART Bulletin*, 7(2), 1996.

[49] Alan K. Walton. *Network Analysis and Practice.* Cambridge University Press, Cambridge MA, 1987.

[50] David L. Waltz. Generating semantic descriptions from drawings of scenes with shadows. Technical Report AI-TR-271, MIT Artificial Intelligence Laboratory, Cambridge, MA, 1972.

[51] Allen C. Ward, Tomás Lozono-Pérez, and Seering Warren P. Extending the constraint propagation of intervals. In *Proc. 11th Int. Joint Conf. on Artificial Intelligence (IJCAI-89)*, 1453–1458, San Mateo, CA, 1989. Morgan Kaufmann.

[52] Brian Carter Williams. Qualitative analysis of MOS circuits. *Artificial Intelligence*, 24:281–346, 1984.