Forward Reasoning and Dependency-Directed Backtracking
In a System for Computer-Aided Circuit Analysis

by Richard M. Stallman and Gerald Jay Sussman

Abstract:

We present a rule-based system for computer-aided circuit analysis. The set of rules, called EL, is written in a rule language called ARS. Rules are implemented by ARS as pattern-directed invocation demons monitoring an associative data base. Deductions are performed in an antecedent manner, giving EL's analysis a catch-as-catch-can flavor suggestive of the behavior of expert circuit analyzers. We call this style of circuit analysis propagation of constraints. The system threads deduced facts with justifications which mention the antecedent facts and the rule used. These justifications may be examined by the user to gain insight into the operation of the set of rules as they apply to a problem. The same justifications are used by the system to determine the currently active data-base context for reasoning in hypothetical situations. They are also used by the system in the analysis failures to reduce the search space. This leads to effective control of combinatorial search which we call dependency-directed backtracking.

Work reported herein was conducted at the Artificial Intelligence Laboratory, a Massachusetts Institute of Technology research program supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored by the Office of Naval Research under contract number N00014-75-C-0643.

Steele, Charles Rich and Ben Kuipers gave us some important editorial help. John Allen, David Marr, Pat Winston and Paul Penfield also provided good advice.

Contents:

A second major problem is the difficulty of debugging programs containing large amounts of knowledge. The complexity of the interactions between the "chunks" of knowledge makes it difficult to ascertain what is to blame when a bug manifests itself.[Complexity] One approach to this problem is to build systems which remember and explain their reasoning.[Explainers] Such programs are more convincing when right, and easier to debug when wrong.

We have designed and implemented[LISP] a problem-solving language called ARS[ARS] in which problem-solving rules are represented as demons with multiple patterns of invocation[Pattern-directed invocation] monitoring an associative data base.[Data bases] It performs all deductions in an antecedent manner, threading the deduced facts with justifications which mention the antecedent facts used and the rule of inference applied. These justifications may be examined by the user to gain insight into the operation of the system of rules as they apply to a problem. The same justifications are employed by the system to determine the currently active data-base context for reasoning in hypothetical situations.[Context] Justifications are also used in the analysis of blind alleys to extract information which will limit future search.

We have used ARS to implement a set of rules for electronic circuit analysis. This set of rules, a version of EL,[EL] encodes familiar approximations to physical laws such as Kirchoff's laws and Ohm's law as well as models for more complex devices such as transistors. Facts, which may be given or deduced, represent data such as the circuit topology, device parameters, and voltages and currents. The antecedent reasoning of ARS gives analysis by EL a "catch-as-catch-can" flavor suggestive of the behavior of a circuit expert. The justifications prepared by ARS allow an EL user to examine the basis of its conclusions. This is useful in understanding the operation of the circuit as well as in debugging the EL rules. For example, a device parameter not mentioned in the derivation of a voltage value has no part in determining that value. If a user changes some part of the circuit specification (a device parameter or an imposed voltage or current), only those facts depending on the changed fact need be "forgotten" and re-deduced, so small changes in the circuit may need only a small amount of new analysis. Finally, the search-limiting combinatorial methods supplied by ARS lead to efficient analysis of circuits with piecewise-linear models.

The application of a rule in ARS implements a <u>one-step deduction</u>. A few examples of one-step deductions, resulting from the application of some EL rules in the domain of resistive network analysis, are:

1: If the voltage on one terminal of a voltage source is given, one can assign the voltage on the other terminal.

2: If the voltage on both terminals of a resistor are given, and the resistance is known, then the current through it can be assigned.

3: If the current through a resistor, and the voltage on one of its terminals, is known, along cannot be used symbolically; they can be applied only after one has guessed[Advice] a particular operating region for each nonlinear device in the circuit. Trial and error can find the right regions but this method of assumed states is potentially combinatorially explosive. ARS supplies dependency-directed backtracking, a scheme which limits the search as follows: The system notes a contradiction when it attempts to solve an impossible algebraic relationship, or when discovers that a transistor's operating point is not within the possible range for its assumed region. The antecedents of the contradictory facts are scanned to find which nonlinear device state guesses (more generally, the backtrackable choicepoints) are relevant; ARS never tries that combination of guesses again. A short list of relevant choicepoints eliminates from consideration a large number of combinations of answers to all the other (irrelevant) choices. This is how the justifications (or dependency records) are used to extract and retain more information from each contradiction than a chronological backtracking system.[Backtracking] A chronological backtracking system would often have to try many more combinations, each time wasting much labor rediscovering the original contradiction.

How it works:

In EL all circuit-specific knowledge is represented as assertions in a relational data base. General knowledge about circuits is represented by laws, which are demons subject to pattern-directed invocation. Some laws represent knowledge as equalities. For example, there is one demo for Ohm's law for resistors, one demon that knows that the current going into one terminal of a resistor must come out of the other, one demon that knows that the currents on the wires coming into a node must sum to zero, etc. Other laws, called Monitors handle knowledge in the form of inequalities: For example, I-MONITOR-DIODE knows that a diode can have a forward current if and only if it is ON, and can never have a backward current.

When an assertion (for example, (= (VOLTAGE (C Q1)) 3.4), which says that the voltage on Ql's collector has the value 3.4 volts) is added to the data base, several demons will in general match it and be triggered. (In this example, they will include DC-KVL, which makes sure that all other elements' terminals connected to Ql's collector are also known to have that voltage, and VCE-MONITOR-BJT, which checks that Ql is correctly biased for its assumed operating region. The names of the triggered laws are put on a queue, together with arguments such as the place in the circuit that the law is to operate. Eventually they will be taken off the queue and processed, perhaps making new deductions and starting the cycle over again.

When a law is finally processed, it can do two useful things: make a new assertion (or several), or detect a contradiction. A new assertion is entered in the data base and has its antecedents recorded; they are the asserting demon itself, and all the assertions which invoked it c were used by it. This complete memory of how every datum was deduced becomes useful when a and D5 were connected in series. Next, one of the conspiring choices is arbitrarily called the "culprit" ("scape-goat" might be a better term) and re-chosen differently. This is not mere undirected trial and error search as occurs when chronological backtracking with a sequential control structure is used, since it is guaranteed not to waste time trying alternative answers to an irrelevant question. The NOGOOD assertion is a further innovation that saves even more computation by reducing the size of the search space, since it contains not *all* the choices in effect, but only those that were *specifically used* in deducing the contradiction. Frequently some of the circuit's transistors will not be mentioned at all. Then, the NOGOOD applies regardless of the states assumed for those irrelevant transistors. If there are ten transistors in the circuit not mentioned in the NOGOOD, then since every transistor has three states (in the EL model) the single NOGOOD has ruled out $3^{10}=59049$ different states of the whole circuit.

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS

Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS

Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS

Sync your system to PACER to automate legal marketing.

WHAT WILL YOU BUILD? | sales@docketalarm.com | 1-866-77-FASTCASE