REPRESENTATION AND ANALYSIS OF ELECTRICAL CIRCUITS IN A DEDUCTIVE SYSTEM

Takushi Tanaka

Artificial Intelligence Project
Department of Computer Science, Yale University
Box 2158 Yale Station New Haven, CT06520 U.S.A.

The National Language Research Institute
3-9-14 Nishigaoka Kita-ku, Tokyo 115 Japan*

ABSTRACT

We have developed representations and analysis methods for electrical circuits in a deductive system called DUCK.** Circuits are represented as conjunctions of logical predicates. Circuit analysis is done as an iteration of proofs which determine the basic structures in the circuit. Electrical constraints for the circuit are produced from the results of the proofs. The constraints are then solved using propagation methods.

1 INTRODUCTION

When an expert first looks at a circuit schematic, he tries to partition the circuit into familiar sub-circuits with known goals. He then tries to pursue the causality of electrical events through those sub-circuits to determine if and how it achieves the overall goal of the circuit. This is based on his assumption that every electronic circuit is designed as a goal oriented composition of basic circuits with known goals. Even in the analysis of an electrical circuit with an unknown goal, although we could perform a formal numerical analysis of the full circuit equations, we try to discover specific sub-structures, such as series or parallel circuits, in order to rewrite the circuit into a simpler one. In many cases we can analyze circuits without formal numerical analysis.

Using a deductive system, we have implemented the process which determines these specific structures in a circuit. Circuit elements and devices are represented as logical predicates which have element names and connecting nodes as terms. A circuit is represented as a conjunction of propositions which represent each electrical element or device. The process of determining a specific structure in the object circuit is realized as a proof of a predicate bound by existential quantifiers representing the existence of the specific structure. Since the transformation of a circuit into an equivalent one involves changing the axioms which represent the object circuit, it is executed as a meta procedure outside of the deductive system. If a specific structure is found in the object circuit, local electrical constraints for the specific structure are translated into global constraints on the object circuit using the results of the proof. The constraints are solved by means of propagation methods. [12]

* present address

** DUCK is a set of LISP routines which implement a relational database similar to those found in AI language such as PLANNER and Prolog. [8]

II REPRESENTATION OF CIRCUITS

A. Predicates for elements and circuits

The circuit CA12 of Figure 1 is represented as follows (1) in the DUCK system. [8] The proposition "(TERMINAL TI #1)" states that TI is a terminal at node #1. The proposition "(RESISTOR RI #1 #2)" states RI is a resistor connecting node #1 and node #2. The node order is important for elements or devices which have polarity, such as diodes. So we define the predicate "DIODE" such that "(DIODE DI #2 #3)" states DI is a diode with the cathode connected to node #2 and the anode to node #3. Now we can also define the predicate "TRANSISTOR" like "DIODE". "CA12" is a predicate without arguments. "(CA12)" as a propositional constant states that there is a circuit called CA12. "->" is the forward chaining symbol. It means logical implication, but procedurally, it means that when "(CA12)" is added to the database of axioms, then "(TERMINAL TI #1)", "(TERMINAL T2 #2)", ... , "(DIODE DI #2 #3)" are to be asserted (added to the database) immediately. [8], [9]

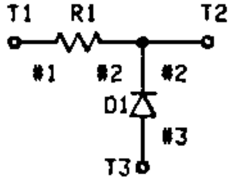


Figure 1: Circuit CA12

```
(RULE CIRCUIT-LIBRARY-CA12
  (CA12) -> (AND (TERMINAL T1 #1)
                 (TERMINAL T2 #2)
                 (TERMINAL T3 #3)
                 (RESISTOR R1 #1 #2)
                 (DIODE    D1 #2 #3))) -------(1)
```

B. Rules for non-polar elements

If the proposition "(CA12)" is asserted, we can prove the proposition "(RESISTOR R1 #1 #2)", but we cannot prove the proposition "(RESISTOR R1 #2 #1)". Since we want to have "(RESISTOR R1 #1 #2)" imply "(RESISTOR R1 #2 #1)", we will assert the following rule (2).

"?X", "?#A", and "?#B" means that variables "X", "#A", and "#B" are bound by universal quantifiers (skolemization). If "(RESISTOR R1 #1 #2)" is asserted, then "(RESISTOR R1 #2 #1)" will be asserted immediately. Similar rules are written for the predicates "CAPACITOR" and "INDUCTOR".

```
(RULE RESISTOR-IMAGE
  (RESISTOR ?X ?#A ?#B) ->
                      (RESISTOR ?X ?#B ?#A)) ---(2)
```

C. Predicates for abstract elements

We can define predicates for abstract elements from atomic formulae representing circuit elements and devices using logical connectives and quantifiers. These predicates correspond to the conceptual hierarchy of circuit elements and devices. First we will define the predicate "Z-ELEMENT", so that we can call resistors, capacitors, and inductors impedance elements (3). The backward chaining symbol "<-" is used insted of the forward chaining symbol. It also means logical implication, but differs proceduraly from forward chaining. The rule (3) is applied when the system tries to prove "(Z-ELEMENT DI #2 #3)" corresponding to the question "Does there exist DI an impedance element connecting node #2 and #3?". In order to prove the proposition, the system tries to prove either "(RESISTOR DI #2 #3)" or "(CAPACITOR DI #2 #3)" or "(INDUCTOR DI #2 #3)". We can also define predicates such as "ACTIVE-ELEMENT", "LINEAR-ELEMENT", ... , "ANY-ELEMENT" so that we can refer to these classes of elements and devices.

```
(RULE IMPEDANCE-ELEMENT
  (Z-ELEMENT ?X ?#A ?#B) <-
                (OR (RESISTOR  ?X ?#A ?#B)
                    (CAPACITOR ?X ?#A ?#B)
                    (INDUCTOR  ?X ?#A ?#B))) -(3)
```

D. Predicates for circuit identification

In order to decide which kind of analysis can apply to the object circuit, several predicates for circuit identification are defined. They identify the global properties of the object circuit. By proving the proposition "(EXIST(X #A)(TERMINAL X #A))", we can decide whether the object circuit is an open circuit with terminals which are connected to other circuits or not. So we can define the predicate "OPEN-CIRCUIT" as follows (4).

"IMPEDANCE-CIRCUIT" (section IV) is defined to prove that all the elements in the circuit are impedance elements. Predicates such as "TWO-TERMINAL-CIRCUIT" which count the number of terminals are defined using the built-in predicate "TOTAL" (section III).

```
(RULE  CT-IDENTIFIER-OPEN-CIRCUIT
  (OPEN-CIRCUIT)  <-  (TERMINAL  ?X  ?#A))        "(4)
```

III FINDING A SPECIFIC STRUCTURE IN A CIRCUIT

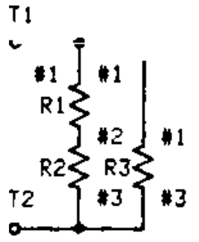
A. Representation of a specific structure

Let us consider the slightly more complicated circuit CA39 in Figure 2. In order to show the existence of a series circuit, we attempt to prove the following formula:

```
(EXIST(X Y #A #B #C)(AND(RESISTOR X #A #B)
                         (RESISTOR Y #B #C))) —(5)
```

When "(CA39)" is asserted, proposition (5) becomes true, because we can substitute constants (RI R2 #1 #2 #3) into variables (X Y #A #B #C) respectively, corresponding to the actual existence of a series circuit in CA39. But we can also substitute constants (RI RI #1 #2 #1) or (RI R3 #2 #1 #3) to the variables which do not correspond to series circuits. In order to reject the former incorrect substitution, the predicate "(NOT(= #A #C)" is added to (5).

Rejection of the latter incorrect substitution is rather difficult. A series circuit demands that the central node "#B" in (5) should not be connected to elements other than those in the formula. In order to represent this constraint the predicates "DEGREE" and "CONNECTED" are introduced.



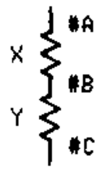


Figure 2: Circuit CA39   Figure 3: R-SERIES

B. Rules for node degree

We will use the backward chaining rule "CONNECTED" to enumerate the terminals, impedance elements and diodes X connected to node #A.

```
(RULE CT-STRUCTURE-CONNECTED
  (CONNECTED ?X ?#A) <-
            (OR (TERMINAL  ?X ?#A)
                (Z-ELEMENT ?X ?#A ?#B)
                (DIODE     ?X ?#A ?#B)
                (DIODE     ?X ?#B ?#A))) ---(6)
```

The predicate "DEGREE" has two arguments. It becomes true if the first one is a node and the second one is its degree in the object circuit represented in the database of axioms. "LEGREE" is defined as follows using "CONNECTED" and "TOTAL".

```
(RULE  CT-STRUCTURE-DEGREE
  (DEGREE  ?#A  ?#DEG)  <-
         (TOTAL  ?#DEG  1(CONNECTED  ?X  ?#A)))  -(7)
```

"TOTAL" is a special built-in predicate which states that the first term is a sum of the second term according to all known instances of the third term. Now we can define the predicate "R-SERIES" which states the existence of a series-resistor circuit in the object circuit as follows.

```
(RULE CT-STRUCTURE-R-SERIES
  (R-SERIES ?X ?Y ?#A ?#B ?#C)
          <- (AND (RESISTOR ?X ?#A ?#B)
                  (RESISTOR ?Y ?#B ?#C)
                  (NOT (= ?#A ?#C))
                  (DEGREE ?#B 2))) ---------(8)
```

Circuit structures for AC-analysis

We can define predicates for a schematic AC-analysis using series, parallel, star, and delta circuit transformations. The predicate "Z-SERIES" for a series circuit of impedance elements is defined by replacing "RESISTOR" with "Z-ELEMENT" in (8). Using the predicate "Z-ELEMENT", predicates "Z-PARALLEL", "Z-STAR", and "Z-DELTA" are also defined.

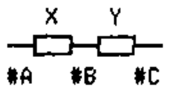


Figure 4: Z-SERIES

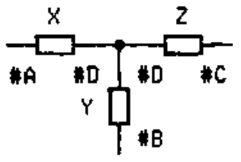


Figure 6: Z-STAR

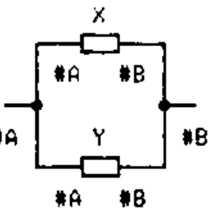


Figure 5: Z-PARALLEL

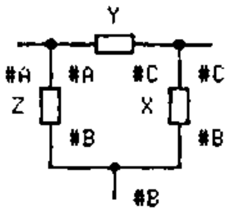


Figure 7: Z-DELTA

IV META PROCEDURES FOR THE DEDUCTIVE SYSTEM

A. Procedurally defined predicates

There are some circuit predicates which can not be adequately represented by the methods discussed so far. One such class is the procedurally defined circuits such as "SERIES-PARALLEL-CIRCUIT" or "MULTIPLE-RECTIFIER-CIRCUIT" which refer to an infinite, recursive set of circuits. Another class is composed of electronic circuits with specific goals, such as "FM-RECEIVER". In the definition of a predicate, the way in which an FM-receiver circuit is constructed from sub-circuits (i.e. RF-amplifier, ... , discriminator) to achieve the final goal of the circuit must be included. The former predicates will be developed in the following paragraphs, but the latter predicates have not yet been developed.

The procedure to determine whether a circuit is a series-parallel circuit of impedance elements or not is as follows. First the circuit is identified as a two terminal circuit and an impedance circuit. Then the circuit is rewritten iteratively into simpler equivalent forms each time a series or a parallel circuit is found in it. If the circuit can ultimately be rewritten into a single element, then we have shown that the original one was a series-parallel circuit. The first and last steps are realized using the predicates "TWO-TERMINAL-CIRCUIT", "IMPEDANCE-CIRCUIT" and "HAS-ONE-ELEMENT" in section HI. But the middle step is not so easy to implement in a deductive system, because rewriting a circuit into an equivalent one means changing the set of axioms representing the circuit. This can be done by introducing a state term into each predicate, [10] But it is easier to consider the procedure in a total system consisting of the deductive system and a meta procedure.

B. Rewriting a circuit

In order to control the deductive system procedurally, several mechanisms are provided for calling DUCK from LISP. If (9) is evaluated in LISP, the proposition "(CA39)" is added to the DUCK database. Then by the forward chaining rule of CIRCUIT-LIBRARY-CA39, "(TERMINAL TI #1)", ... , "(RESISTOR R3 #1 #3)" are added into the database. Then by the forward chaining rule of RESISTOR-IMAGE, "(RESISTOR RI #2 #1)", ... , "(RESISTOR R3 #3 #1)" are also added.

```
(ADD '(CA39))                                  (9)
```

Under this condition let us consider a procedure "R-SERIES-TRANSFORM" which finds a series-resistor circuit and then rewrites that part of the circuit into an equivalent element. At first, corresponding to a proof of the proposition "(EXIST (X Y #A #B #C) (R-SERIES X Y #A #B #C))", the following (10) is evaluated in LISP. If series circuits are found, it returns a stream of instances for the variables (X Y #A #B #C).

```
(FETCH '(R-SERIES ?X ?Y ?#A ?#B ?#C))         ■(10)
```

In order to rewrite a series part, we have to remove the set of axioms corresponding to the series circuit. If the first instance is (RI R2 #1 #2 #3), (11) and (12) are evaluated.

```
(ERASE '(RESISTOR RI #1 #2))                  (11)
(ERASE '(RESISTOR R2 #2 #3))                  (12)
```

When (11) is evaluated, not only "(RESISTOR RI #1 #2)" but also its deduced image "(RESISTOR RI #2 #1)" is erased from the database by a data dependency mechanism. [5]

```
(ERASE '(RESISTOR RI #2 #1))                  (13)
```

If (13) is evaluated insted of (11) when the first instance is (R2 RI #3 #2 #1), then "(RESISTOR RI #2 #1)" will be erased but its original "(RESISTOR RI #1 #2)" can not be erased. As we want to erase the original when the image is erased, we can set up "IF-ERASED-DEMON" in DUCK as a forward chaining from the erasure. The rule is written as follows (14). According to the rule, "(RESISTOR RI #1 #2)" is to be erased by (13).

```
(LISPRULE R-IMAGE-DELETE
  (RESISTOR ?X ?#B ?#A) ->
    (IF-ERASED '(RESISTOR ?X ?#B ?#A)
        (ERASE '(RESISTOR ?X ?#A ?#B)))) —(14)
```

The proposition representing an equivalent resistor is to be added as follows.

```
(ADD '(RESISTOR ERI #1 #3))                   ■(15)
```

C. LISP functions for AC-analysis

Now we can define the LISP function "Z-SP-CIRCUIT" which can determine whether the object circuit is a series-parallel circuit of impedance elements or not as follows.

```
(DE Z-SP-CIRCUIT (OBJ-CT)
  (AND (PROVE '(TWO-TERMINAL-CIRCUIT) OBJ-CT)
       (PROVE '(IMPEDANCE-CIRCUIT) OBJ-CT)
       (Z-SP-CIRCUIT1 OBJ-CT))) ------------(16)
(DE Z-SP-CIRCUIT1 (OBJ-CT)
  (AND OBJ-CT
   (OR(PROVE '(HAS-ONE-ELEMENT)OBJ-CT)
      (Z-SP-CIRCUIT1(Z-SERIES-TRANSFORM OBJ-CT))
      (Z-SP-CIRCUIT1(Z-PARALLEL-TRANSFORM OBJ-CT]
                                    ------------(17)
```

As we can define multiple databases of axiom sets (data pools) in DUCK, the original circuit and all the rewritten circuits for working are stored in separate data pools (see DUCK manual). The variable "OBJ-CT" in (16) and (17) is the name of the data pool which contains the circuit being processing. The LISP function "PROVE" corresponds to "FETCH" in a data pool. The LISP functions "Z-SERIES-TRANSFORM" and "Z-PARALLEL-TRANSFORM" return the name of the new data pool which contains the rewritten circuit, if they succeed in transforming the circuit, otherwise NIL. Finally we have defined the predicate "Z-SERIES-PARALLEL-CIRCUIT" by calling the LISP function "(Z-SP-CIRCUIT 'P00L1)" from DUCK.

The electrical constraints on the object circuit are produced as a side effect of "Z-SERIES-TRANSFORM" and "Z-PARALLEL-TRANSFORM".

We can define a more convenient predicate "Z-SERIES-PARALLEL-STAR-DELTA-ANALYTICAL-CIRCUIT" for actual AC-analysis adding "(Z-STAR-TRANSFORM OBJ-CT)" and "(Z-DELTA-TRANSFORM OBJ-CT)" to (17). It can analyze most two-terminal impedance circuits which electrical engineers meet daily.

V SCHEMATIC CIRCUIT ANALYSIS

If we assert "(CA55)" in Figure 8 as the object circuit, when we try to prove the proposition "(Z-SERIES-PARALLEL-STAR-DELTA-ANALYTICAL-CIRCUIT)", the LISP function "Z-SPYD-CIRCUIT" associated with the proposition is evaluated.

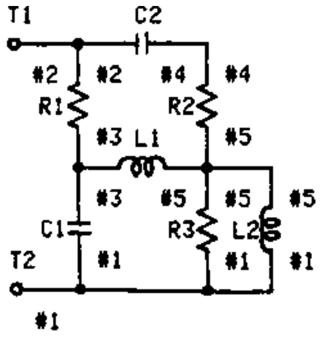


Figure 8: Circuit CA55

After identifing the circuit as a "TWO-TERMINAL-CIRCUIT" and an "IMPEDANCE-CIRCUIT", the LISP functions "Z-SERIES-TRANSFORM", ... , "Z-DELTA-TRANSFORM" are evaluated iteratively. Those functions try to prove the propositions "(Z-SERIES ?X ?Y ?#A ?#B ?#C)", ... , "(Z-DELTA ?X ?Y ?Z ?#A ...)" respectively. At the end of this procedure, the following list (18) is acquired.

```
*PREDICATE-VARIABLE-VALUE*
=((Z-SERIES (NX EZ1)(#C #2)(#B #4)(#A #5)(Y C2)(X R2))
  (Z-PARALLEL (NX EZ2)(#B #1)(#A #5)(Y L2)(X R3))
  (Z-STAR (NX EZ3)(NY EZ4)(NZ EZ5)(#D #5)(#C #3)
          (#B #2)(#A #1)(Z L1)(Y EZ1)(X EZ2))
  (Z-PARALLEL (NX EZ6)(#B #2)(#A #3)(Y R1)(X EZ3))
  (Z-PARALLEL (NX EZ7)(#B #3)(#A #1)(Y C1)(X EZ4))
  (Z-SERIES (NX EZ8)(#C #2)(#B #3)(#A #1)(Y EZ6)(X EZ7))
  (Z-PARALLEL (NX EZ9)(#B #2)(#A #1)(Y EZ5)(X EZ8)))
                                          -----(18)
```

The first line shows, at first the proposition "(Z-SERIES ?X ?Y ?#A ?#B ?#C)" was satisfied by the bindings in the following association list. "(NX EZI)" shows that the new name "EZI" was assigned to the equivalent element NX.

The following electrical constraints (19) are provided for a series-impedance circuit. "X" and "Y" represent impedances of elements X and Y respectively. "NX" represents the equivalent impedance of a series circuit. "#A", "#B", and "#C" represent electrical potentials of node #A, #B, and #C (see Figure 4). The symbols representing elements in the constraints mean their impedances. The symbols representing nodes mean their electrical potentials. "C+", "C-", "C*", and "C//" are defined as arithematic operators on complex numbers.

```
((C= NX (C+ X Y))
 (C= #B (C//(C+(C* #A Y)(C* #C X))(C+ X Y))))       (19)
```

According to the list (18), each local constraint provided for the basic structures are translated into global constraints on the object circuit, and the following list (20) is acquired.

```
*OBJECT-CIRCUIT-CONSTRAINTS*
= ((C= EZ1 (C+ R2 C2))
(C= #4  (C//(C+(C* #5 C2)(C* #2 R2))(C+ R2 C2)))
(C= EZ2 (C//(C* R3 L2)(C+ R3 L2)))
(C= EZ3 (C//(C+(C* EZ2 EZ1)(C* EZ1 L1)(C* L1 EZ2))EZ2))
(C= EZ4 (C//(C+(C* EZ2 EZ1)(C* EZ1 L1)(C* L1 EZ2))EZ1))
(C= EZ5 (C//(C+(C* EZ2 EZ1)(C* EZ1 L1)(C* L1 EZ2))L1))
(C= #5  (C//(C+(C* EZ1 L1 #1)(C* L1 EZ2 #2)
                                     (C* EZ2 EZ1 #3))
          (C+(C* EZ2 EZ1)(C* EZ1 L1)(C* L1 EZ2))))
(C= EZ6 (C//(C* EZ3 R1)(C+ EZ3 R1)))
(C= EZ7 (C//(C* EZ4 C1)(C+ EZ4 L1)))
(C= EZ8 (C+ EZ7 EZ6))
(C= #3  (C//(C+(C* #1 EZ6)(C* #2 EZ7))(C+ EZ7 EZ6)))
(C= EZ9 (C//(C* EZ8 EZ5)(C+ EZ8 EZ5)))) -----------(20)
```

The LISP function "PROPAGATE" solves these constraints. Here "C+" is analogus to "SETQ" in LISP. At each constraint, if all the variables in the right hand side become defined, it is evaluated and the left hand variable is assigned that value and the constraint is then removed from the list. This procedure is iterated until the right hand sides cease to fire. Suppose each element of CA55 has following value (21). Units are ohm, farad, and henry respectively. If 10 (volt) 60 (hertz) AC voltage is applied between terminal TI and T2, potentials of node #1 and #2 are set as (22), and impedance of each element at 60 (hertz) is also computed and set as (23).

After the evaluation of the LISP function "PROPAGATE", the following answers (24) are acquired.

```
*ELEMENT-VALUE*
= ((R1 10.0)(R2 20.0)(R3 30.0)(C1 0.0001)
   (C2 0.0002)(L1 1.0)(L2 2.0)) -----------------------(21)

*POTENTIALS* = ((#2 (10.0 0.0))(#1 (0.0 0.0))) ----(22)

*ELEMENT-IMPEDANCES*
= ((R1 (10.0 0.0))(R2 (20.0 0.0))(R3 (30.0 0.0))
   (C1 (0.0 -26.52576))(C2 (0.0 -13.26288))
   (L1 (0.0 376.922))(L2 (0.0 753.984))) -----------(23)

*COMPUTED-IMPEDANCES-POTENTIALS*
= ((EZ1 (20.0 -13.26288))(EZ2 (29.95250 1.191773))
   (EZ3 (196.6667 608.4257))(EZ4 (-245.6907 756.9014))
   (EZ5 (48.96205 -13.70207))(EZ6 (9.949947 .1473568))
   (EZ7 (-.2889358 -27.30829))(EZ8 (9.661011 -27.24093))
   (EZ9 (12.88425 -16.01124))(#3 (8.89739 -3.261531))
   (#5 (5.402148 1.524807))(#4 (9.297949 2.583477)))
                                             -----------(24)
```

VI <u>CONCLUSION</u>

We have developed a representation for electrical circuits using conjunctions of predicates, and realized the procedure of determining circuit structures as proofs. AC and multiple-rectifier analysis were done to confirm the effectiveness of this method. The structures to be found in the analysis were rather simple ones, and the object circuits were restricted to iterations of those structures.

We plan to develop this method to analyze more complicated electronic circuits with specific goals. For those circuits we have to introduce many predicates referring to circuit goals and device-element goals such as "RF-AMPLIFY", "FREQUENCY-CONVERT", "RF-BYPASS", "DC-BLOCK". The representation of a goal structure for the circuit is as important as the representation of the circuit itself. Predicates defining electrical potentials and currents and their changes will also be introduced, e.g. "CURRENT", "SIGNAL", "FLOW", "INCREASE", "DECREASE". Predicates such as "OPEN", "SHORT", "SATURATE", "CUTOFF" will be introduced to represent the electrical states of circuits and devices. Using those predicates, causality of electrical events in the circuit can be represented. This in turn permits the definition of functional predicates which can be used to classify circuits on the basis of their behavior.

The process for determining structures in a circuit must be improved. The approach illustrated in this paper is applicable only to simple circuits, for the time required to identify a circuit increases combinatorially with the number of components. To handle more complex circuits, the system must restrict the search space in an intelligent manner. For example, if it is known that the goal of the object circuit is "receive FM broadcast", then we have many hints to help in determining the structure. If a loud speaker is found in the circuit, the system can use the expectation that the adjacent circuit must be a power amplifier. These expectations will not only decide which structure should be looked for next, but will also decrease the search space for that structure in the circuit. The proof "(COMPLEMENTARY-POWER-AMPLIFIER ?X ?Y ?#A ?#B...)" is slower than proof of the "(COMPLEMENTARY-POWER-AMPLIFIER Q8 Q9 ?#A ?#B...)" as the number of transistors in the circuit increases. For this purpose we may introduce predicates such as "ADJACENT-ELEMENT", "NEAREST-TRANSISTOR". We are now developing various predicates for the qualitative analysis of electronic circuits.

ACKNOWLEDGMENTS

I would like to thank: Prof. Roger Schank for the opportunity to study at Yale AI-project, Prof. Drew McDermott for DUCK, and David Littleboy, Stanley Letovsky, and Jay Bolter for their helpful advice.

REFERENCES

[I] Brown,A. "Qualitative Knowledge, Causal Reasoning, and the Localization of Failures" M.I.T. AI-TR-362, 1977
[2] Charniak,E., Riesbeck,C., McDermott,D. "Artificial Intelligence Programming" Lawrence Erlbaum Association, 1980, NJ.
[3] DeKleer,J. "Causal and Teleological Reasoning in Circuit Recognition" M.I.T. AI-TR-529, 1979
[4] DeKleer,J. "The origin and Resolution of Ambiguities in Causal Arguments" IJCAI-79, 1979
[5] Doyle,J. "A Truth Maintenance System" Artificial Intelligence, vol.12, 1979
[6] McDermott,D. "Circuit Design as Problem solving" Artificial Intelligence and Pattern Recognition in Computer Aided Design, Latombe ed. IFIP, North-Holland, 1978
[7] McDermott,D. "Non Monotonic Logic II : Non-Monotonic Modal Theories" Research Report #174, Dept. of Computer Science, Yale Univ. , 1980
[8] McDermott,D. "DUCK: A Lisp-based Deductive System" Dept. of Computer Science, Yale Univ. 1982
[9] McDermott,D., Charniak,E. "Introduction to Artificial Intelligence" unpublished manuscript
[10] Nilsson,N.J. "Problem-solving Methods in Artificial Intelligence" McGRAW-Hill, 1971
[II] Nilsson,N.J. "Principles of Artificial Intelligence" Tioga Pub., Palo Alto, 1980
[12] Stallman,R.M.,Sussman,G.J. "Forward Reasoning and Dependency-Directed Back-tracking in a System for Computer Aided Circuit Analysis" Artificial Intelligence, vol.9, 1977
[13] Sussman,G.J., Stallman,R.M. "Heuristic Technique in Computer-Aided Circuit Analysis" IEEE Trans. Circuit and Systems, vol.CAS-22,No.II,1975
[14] Sussman,G.J. "SLICES At the Boundary Between Analysis and Synthesis" Artificial Intelligence and Pattern Recognition in Computer Aided Design, Latombe ed., IFIP, North-Holland, 1978
[15] Sussman,G.J.,Steele,G.Jr "Constraints: A Language for Expressing Almost-Hierachical Descriptions" Artificial Intelligence, vol.14, 1980
[16] Tanaka,T. "Simulation of Thinking Process on Electronic Circuits in Natural Language I,II,III", Technology Report of Kyushu Univ. vol.44,45, 1971, 1972 (Japanese)
[17] Tanaka,T. "Pattern Directed Circuit Recognition I,II" 23rd & 24th Covention of Information Processing Society in Japan, 1981,1982 (Japanese)