1981

Using design specifications for design

Neal M. Holtz
Carnegie Mellon University

Steven J(Steven Joseph) Fenves

Published In
.

USING DESIGN SPECIFICATIONS FOR DESIGN

by

Neal M. Holtz and Steven J. Fenves

DRC-12-03-81

August 1981

USING DESIGN SPECIFICATIONS FOR DESIGN

Neal M. Holtz[1] and Steven J. Fenves[2], M. ASCE

INTRODUCTION

This paper describes a means of representing a design specification in a computer-processable form so that it may be used to check a design or to assist in actually designing certain quantities. In a design mode, a computer program will automatically accept design variables that have firm values assigned to them, and will formulate concise, symbolic descriptions of the constraints on the remaining variables such that all pertinent specification criteria will be satisfied.

Note. – This is a report of research currently in progress at Carnegie-Mellon University. While it has been demonstrated by the development of a program that the ideas are workable, it is very likely that many modifications will be made to the implementation and to the representation, over the next few months as experience is gained with the system. Refer to the technical report by the same authors, to be released in the fall of 1980 [5],

SPECIFICATION REPRESENTATION

In order to allow the automatic reformulation of design specifications and constraints, two major and related issues must be considered. First, a means of representing the specification in computer-processable form must be developed. The representation must allow efficient processing of design data against the specification, in order to either Judge adequacy or to produce design constraints, which are algebraic expressions that express allowable values for those design variables that do not have fixed values. Then, the requisite algorithms must be developed that will enable a program to process a portion of a specification along with its associated data, in the checking or in the design mode. Ultimately, this processing will be initiated automatically (when a user attempts to change a data value in a data base), or in response to a users request to check a design.

This section deals with the issue of representation; the question of processing that representation is considered in the following section. The representation will be based on a network of decision tables to

^Graduate Student, Department of Civil Engineering, Carnegie-Mellon University, Pittsburgh, PA 15213.

[2]University Professor, Department of Civil Engineering, Carnegie-Mellon University, Pittsburgh, PA 15213.

describe the meaning of design specifications [2, 3? *t 6].

Network Representation. – It has long been recognized that the use of a system of decision tables is a convenient, precise way to represent the semantics of design specifications. Such a representation is shown in Figure 1 for a portion of a CSA specification [1] that deals with the required amount of longitudinal (flexural) reinforcing in a singly reinforced concrete beam.

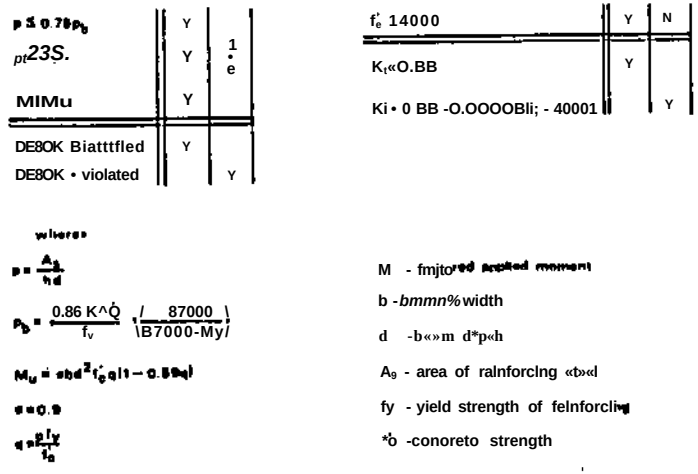

Figure 1: Decision Table Representation of a Specification

There is a hierarchical relation among the variables (hereafter called "data items") in a design specification. Before some data may be calculated, others (their "ingredients") have to have known values (the former are called "dependents" of the latter). This relationship was exploited in [**], when a set of algorithms was developed to process specifications for a design check. ""

All the data items in a specification (or a portion of one) were represented in network (or graph) form. Individual data were represented by nodes and the edges (arcs) in the graph represented direct data dependencies. That is, from any datum in the network, arcs were directed to all of its direct dependents. Thus, by following the arcs in a reverse direction from a node, all of its direct ingredients could be reached. This representation allows a computer program to compute only the values that are absolutely necessary when checking a design (more will be said about this in the section on specification processing). Figure 2 shows the dependency network for the portion of the CSA specification of Figure 1.

This network representation contains no indication of the logic necessary to perform any of the computations to evaluate a datum; it only shows the data dependencies. In order to allow a computer program

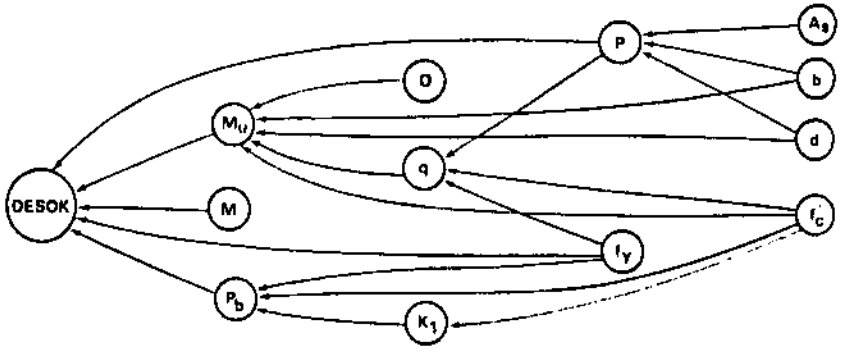

Figure 2: Dependency Network

to assist in the textual organization of specifications [6], the network model was extended to include more of the structure of the individual decision tables. Added to the network were nodes representing the individual rules and conditions. The requirement that a table be used to compute the value of only one datura allowed the assignment of a unique node in the network to the result of a table. The ingredients of that node were all of the rules of the table. Ingredient to each rule was the action to be taken if the rule governs, and the conditions that must be checked to see if the rule does govern. The ingredients of the conditions were all of the data contained in the algebraic expression of the condition.

Only minor additions to the above described network are necessary to create the new "fully expanded" representation that meets the two requirements mentioned earlier.

Final Representation. – The network representation will again derive directly from the decision table representation, and the graph will consist of the following five types of nodes:

1. "Basic Data" - these nodes represent the basic input values to the system; they represent data, the value of which are are normally not calculated from the specification but must be derived from the particular design (for example, beam depth). Note, however, that we are proposing a system whereby some of the basic data may be calculated from the specification, providing the values of enough of the other basic data are known.
2. "Derived Data" - these represent intermediate data in the system which are generally needed in some further calculation. The specification gives explicit rules for calculating these values. We may distinguish two types of derived data:

   - a "requirement" is a datum (always boolean) evaluating to "satisfied" or "violated".
   - a "determination" is any other derived datum (numeric or boolean).

   Some derived data are generated for internal use by the system, and are not explicitly mentioned in the specification. These are usually included for efficiency or for simplification and clarity. For instance, we will frequently label a particular condition with a datum name so that only one instance of the condition expression need be included in the network.
3. "Expressions" - These are algebraic expressions and are used to compute some value (numeric or boolean). These will be the expressions that are found in the specification.
4. "Decision Tables" - these represent the results of a computation performed be executing a decision table. They are essentially collection nodes, provided so that the datum that is computed via the table may have only one ingredient.
5. "Rules" - These nodes represent the rules of a decision table, and each is a collection node for the various conditions and the ~~one~~ action associated with each rule.

Directed arcs extend to each node from all of that node[1]s ingredients as follows:

1. "Basic data" have no ingredients, and because they have no arcs leading in to them, are referred to as "initial" nodes.
2. "Derived data" may have only <u>one</u> ingredient and that may be one of the following: the expression that is used to compute the value of that datum, or, the decision table generating the value of the node.
3. "Expressions" (which were called functions in [3]) have only basic or derived data as ingredients, and these are all of the data that explicitly appear in the expression.
4. "Decision tables" have only rules as ingredients.
5. "Rules" have <u>one</u> action, and zero or more conditions as ingredients. The arcs from the condition nodes to the rule are labelled with the value the condition must have for the rule to govern. If a rule has no condition ingredients, it is assumed to be an "else" rule.

In some cases, the left to right order of the arcs leading to a node is important. As there is generally an assumed left to right ordering when checking the rules of a decision table, the ordering of the arcs from rule nodes to table nodes is maintained and is significant.

There should be exactly <u>one</u> derived datum in the network (or subnetwork)^ that has no arcs leading from it. This node, called a "terminal" node, will represent the union of requirements from the

3the term "subnetwork" will usually be used to refer to a portion of the specification (i.e. a sub-requirement) that directly leads to a higher level requirement - for example, the portion dealing with "p^" (PBOK) in Figure 3.

specification to be satisfied.

For example, Figure 3 shows a portion of the "fully expanded" network representation of the specification shown in Figure 1. The derived datum node labelled "DESOK" is a terminal node which represents the requirement that states that the reinforcement is adequate if three sub-requirements are satisfied. The derived datura node labelled "PBOK" is a terminal node of the subnetwork which represents the sub-requirement dealing with "balanced" reinforcing.

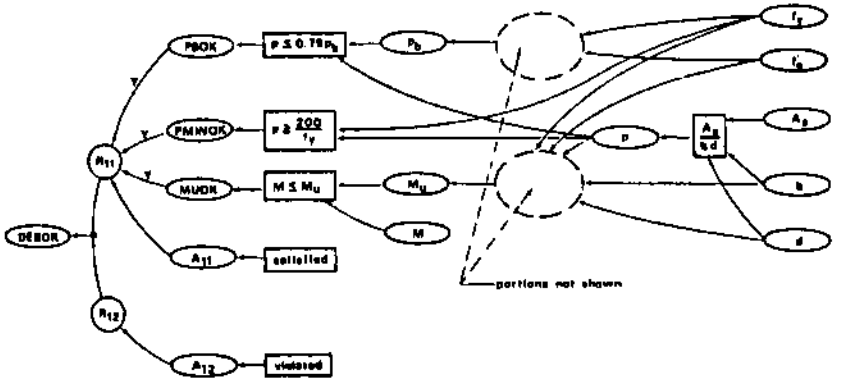

Figure 3: Fully Expanded Network

SPECIFICATION PROCESSING

The objective of specification checking is to determine the value of the topmost requirement; that of design is to make the values of one or more topmost requirements evaluate to "satisfied". Described in this section is a procedure to process the network in either a checking or design mode.

The checking procedure described in [4] was a recursive one. It involved starting with the terminal node of the network and attempting to compute a value for it. However, before computing the value for any datum in the network, the values of all of its direct ingredients must first have been computed. By maintaining a flag with each datura to indicate if it currently had a value, the above procedure resulted in evaluating the minimum possible amount of the network.

The new procedure for evaluation described here slightly extends the basic method of the earlier report. When the topmost node of the network has been determined, the program will attempt to compute a value for that node. The procedure will first recursively evaluate each of the node's ingredients. When all such values have been determined, the value of the node in question may be determined by applying one of a few rules (depending on the type of node) as explained below. The major difference from earlier work will be in the types of values allowed for each of the data. Where the previous work allowed only numeric or boolean values, the current program allows the following values for basic data:

- a simple numeric or boolean value.
- any arbitrary algebraic expression.
- no value defined. In this case, the program assumes that the data item is to be a designable quantity, and the variable name is carried symbolically through all computations.
- any number of single values of the above types, along with a boolean expression of the conditions under which each value is valid. The list of the constraint - value pairs required for this is termed a "production list".

Algebraic Expressions. — These, *r^f^rred* to here as simply expressions, are representations of algebraic expressions in a form that a computer program can evaluate. They may contain variables, constants, operators (addition, exponentiation, etc.) and references to funotions (SQRT, MIN, MAX, SIN, etc.).

Production Lists. — A production list is a means used to express a series of alternate values, and the conditions (or constraints) for which each of the values is meaningful. Formally, a production list is a list of "n" constraint-value pairs, $<C_i,V_i>$, where $n \pounds 1$ and $1 \; 1 \; i \; \mathbf{i} \; n$. C^, the constraint, is a boolean expression, and V_t, the value, is any algebraic expression. For any i, if C^ evaluates to "true", then Vi is the value of the production list. The constraints must be checked in order of increasing i, and the first one evaluating to "true" governs. C_n (the last constraint) will often be the constant "true" in order to supply V_n as the value of the production list if none of the other constraints is satisfied.

As an example, consider the problem of designing a concrete beam with three values of moment given along the span, say at U^1 and 8^1 from the left support, and at the center-line. The production list used to express the moment might be:

(<X 1 4, 287000> <X 1 8, 492000> <TRUE, 656000>}

where "X" is the distance from the left support. The production list states:

"if X is less than or equal'to 4, then the value is 287000,
else if X is less than or equal to 8, then the value is M92000,
else the value is 656000."

Note that production lists can be assigned to basic data. Thus, the system will handle any additional constraints on the input data that can be put in the form of a production list, and will handle "multi-valued" data such as moments along a span. The constraints that are thus put on the basic data are carried through the network in the evaluation process, and the final result is expressed in terms of the initial constraints.

Network Evaluation. — Now that the concepts of expressions and production lists have been introduced, we may address the issue of network evaluation. This is the process of computing the value of the terminal node of the network, given specific values for some or all of

the initial nodes. If all of the basic data have simple numeric values, then the the evaluation is a design check, and the resulting value of the terminal node will be the boolean constant "satisfactory" or "violated". If some of the basic data are undefined (have no value assigned), then the evaluation process is one of design, and the resulting value of the terminal node will be a constraint expression involving the undefined variables (and any additional variables the user may have supplied, such as "X" in the example above). The constraint will be a boolean expression that will specify the values of the basic data for which the topmost node will evaluate to "satisfied". In other words, the constraint expression indicates "allowable" values of the undefined basic data.

Then the processing of the network may be stated as follows. To evaluate any node, use that node's currently defined value if it has one. Otherwise, perform one of the following, depending on the type of the node:

1. Basic Datum - attempt to retreive the value from somewhere external to the system (the current system asks the user to type in the value). If unable to obtain a value (currently indicated by a null input), then return an expression consisting only of the data name.
2. Derived Datum - evaluate the datum's ingredient (it may have only one) and return that value.
3. Expression - evaluate all of the ingredients. As any of those values may be production lists, the value of the expression may be also. The value of the expression is formed by taking all possible combinations of values in the production lists of all the ingredients. Each combination so formed is one possible set of values for the variable ingredients, and these values are substituted into the expression to get one possible value for the expression. The constraints corresponding to each variable value are AND'ed together to form the constraint for the resulting expression value. For example, consider the expression "a + b". If the value of "a" is $\{\langle C_{a1},A_1\rangle\ \langle C_{a2},A_2\rangle\}$, and the value of "b" is $\{\langle C_{b1},B_1\rangle\ \langle C_{b2},A_2\rangle\}$, then the value of the expression is $\{\langle C_{a1}\ \text{AND}\ C_{b1},\ A_1+B_1\rangle\ \ \langle C_{a1}\ \text{AND}\ C_{b2},\ A_1+B_2\rangle\ \ \langle C_{a2}\ \text{AND}\ C_{b1},\ A_2+B_1\rangle\ \langle C_{a2}\ \text{AND}\ C_{b2},\ A_2+B_2\rangle\}$.
4. Rule - evaluate the condition ingredients in turn. If any evaluate to "false", then the value of the rule is "undefined". Otherwise, evaluate the action ingredient. All of the constraints and all of the values in the production lists of all of the conditions are "AND"'ed to form one expression. This in turn is "AND"'ed to each constraint in the action's production list, and the resulting list becomes the value of the rule.
5. Table - evaluate each rule ingredient in turn. If the value is "undefined", then this rule plays no part in the final value of the table. If the constraint portion of the value of the rule is true then the rule governs, then the value of the table is the value portion of the production list of the governing rule. Otherwise no particular rule governs, and the value of the table is simply the concatenation of the production lists of each of the rules.

The final value of the terminal node will be the production list {<C, "satisfied"> <true, "violated">}, where "C" is a constraint expression stating allowable values of the designable data.

Network Re-Evaluation. -- In normal design practice, it is quite common for the values of some of the basic data to change, requiring that the network be re-evaluated to either re-check the design, or to re-design some of the basic data. When this happens, the network processor should not have to re-evaluate the entire network, rather it should only have to re-do those portions affected by the changed data.

At present, we have a two-valued indication of the status of any given node: either the node has or does not have a value. When a datum node is evaluated, its presence flag is set to "valid" to indicate that it has a value. If the value of any node is ever changed from that used during network evaluation, then the presence flag of that node and of all its direct and indirect ingredients must be set to "void". Note that this is essentially identical to the procedure "WARN" given in [4].

EXAMPLE PROCESSING

An example will serve to demonstrate some of the capabilities of the system. Ultimately it is intended that the system accept specification input in a form as close textually to the decision tables of Figure 1 as most common input devices will allow. For the present system, the tables must be hand translated to the form shown in Figure 4. The translation from decision table form to the "internal" form shown is very easy, and will be simple and straight-forward to automate.

```
DESOK << TABLE(R11,R12);
  R11 << RULE(A11,PBOK,PMINOK,MUOK);
  R12 << RULE(A12);
    A11 << TRUE;
    A12 << FALSE;
    PBOK << P <= 0.75*PB;
    PMINOK << P > 200/FY;
    MUOK << M <= MU;
      P << AS/(B*D);
      PB << 0.85*K1*(FPC/FY)*(87000/(87000+FY));
        K1 << TABLE(R21,R22);
          R21 << RULE(A21,C21);
          R22 << RULE(A22,NOT C21);
            A21 << 0.85;
            A22 << 0.85-0.00005*(FPC-4000);
            C21 << FPC <= 4000;
      MU << PHI*B*D*D*FPC*Q*(1-0.59*Q);
        PHI << 0.90;
        Q << P*FY/FPC;
```

Figure 4: Example Network Input

The internal form is a set of standard algebraic expressions, with the addition of the "<<" symbol (which may be read "is computed by"). This is a binary infix operator (much like the value assignment ":=" in

PASCAL or "s" in FORTRAN), and is used to define the expression used to calculate a value for the datum. The expression to the right of the "<<" is stored as an ingredient of the datura to the left. The system automatically creates all the dependent and ingredient relationships from the input expressions. The indentation shown above is not significant to the system, it is only provided for clarity.

An example session using the above described network is shown in Figure 5. User input to the system is underlined, and all lines are numbered at the left for reference.

```
1          >EVALUATE(DESOK);
2    Enter the value of AS : >j
3    Enter the value of B : >JLL
4    Enter the value of D : >12.5;
5    Enter the value of FPC : M000;
6    Enter the value of FY : >60000;
7    Enter the value of M : >IF X < L/8 THEN 656000*7/16
     >                      ELSE IF X < L/4 THEN 656QQQ*V4
     >                           ELSE 656000;
8    Which variable of (L X AS) do you wish to solve for ? >A&
9    [N1]    (X < 0.125^L) AND (AS >= 0.44897103) AND (AS <= 1.6035076)
              OR
             (X < 0.25^L) AND (AS >= 0.80543933) AND (AS <= 1.6035076)
              OR
             (AS >= 1.1198247) AND (AS <= 1.6035076)
10         >ERASE(M);
11   [N2]    UNDEFINED
12         >EVALUATE(DESOK);
13   Enter the value of M : >±
14   Which variable of (M AS) do you wish to solve for ? >M
15   (N31    (AS > 0.33333333) AND (AS <= 1.6035076)
                AND (M <= 674999.99^AS»(-0.H800000»AS + 1.0))
```

Figure 5: Example Network Evaluation

Line 1 shows initiation of processing by a request to evaluate the topmost node, "DESOK". As the system attempted to process the specification, it eventually determined that a value was needed for "AS" (area of steel) and that "AS" was a basic datura. It thus requested that the user enter the value (line 2; future versions of the system may attempt to retreive the value from a data-base). The user typed nothing (except for the input terminating ";") to indicate that "AS" is to be a designable quantity. Lines 3 through 6 show numeric values being entered for other basic data.

Line 7 shows that the user wanted to design for three different values of applied moment. Rather than run the evaluation 3 different times, he chose to enter 3 values at once, together with constraints indicating when the values are valid. The variable names "X" and "L" have significance only to the user (in this case "X" represents distance along the span from the left support and "L" represents span length). The expression for "M" was entered over 3 lines to improve readability; this is of no significance to the system.

At line 8, the network processing has been completed. At this time, the system attempts to rewrite all resulting constraint expressions so

that only one variable appears on the left of any relational operator (<, <:, etc.). Here it found three variables in the result, so the user was asked which of the three he wants as his primary designable quantity.

Line 9 shows the resulting expression that gives allowable values for "AS" for the three different moments (specified in terms of "X" and "L"). The label "[N1]" is a system generated variable name, provided so that the output expression may easily be referred to later.

At line 10, the value of "M" is erased, and consequently the values of all data dependent on "M" are also erased. The topmost node is re-evaluated (line 12), and a value for "M" is requested (line 13) t but' not given. The values of the other data are retained from the previous evaluation. The result at line 15 gives upper and lower limits for the area of steel, "AS", and an upper limit for applied moment, "M", as a function of "AS".

CONCLUSIONS

It has been demonstrated that it is possible to automatically reformulate design checking expressions so that they may be used to directly design quantities of interest. The facility should be useful for designers who wish to use the specifications to compute values for design data. It will be of particular use to data-base management systems that implement the checking of design specifications as part of the consistency maintenance function.

REFERENCES

[1] Canadian Standards Association.
£2A Standard A21. 1-1970. Code for .tha 4fiAi£Il J2l £lain fl£ Reinforced Concrete structures.
CSA, 1970.

[2] Fenves, S. J.
Tabular Decision Logic for Structural Design.
Journal *Q£ £hS.* 5frructural Division. ASGE. 92(ST6) :473-1*90, December, 1966.

[3] Fenves, S. J., Gaylord, E. H., and Goel, S. K.
Decision Xabla Formulation *Q£* Uis. AI5£ Specifications.
Civil Engineering Studies, Structural Research Series No. 347, University of Illinois, August, 1969.

[4] Goel, S. K., and Fenves, S. J.
Computer-Aided Processing of Design Specifications.
jQUrpa; Pf thu Structural Division. ASCE 97(ST1):463-479, January, 1971.

[5] Holtz, N. M., and Fenves, S. J.
Using Design Specifications for Design.
Work in Progress , Carnegie-Mellon University, Department of Civil Engineering, Fall (projected), 1980.

[6] Nyraan, D. J., and Fenves, S. J.
Organizational Model for Design Specifications.
Journal of the Structural Division. A5££ 101(ST4):697-7i6, April, 1975.