Automated Performance of Task-level Symbolic Computations in Engineering Design

Dhiraj K. Pathak, David M. Steier

EDRC 05-60-92

Automated Performance of Task-level Symbolic Computations in Engineering Design

Dhiraj K. Pathak and David M. Steier

05-6[illegible]-92

Robotics Institute and Engineering Design Research Center
Carnegie Mellon University
Pittsburgh, PA 15213

September, 1992

Abstract

In this paper we describe a prototype system, Soar/Mathematica, for automated performance of symbolic computations in engineering design. The input to the system comprises of a mathematical model of a design artifact and the queries or transformations desired on this model by the engineer. The system automatically does the mathematics required to achieve the input goals by intelligently using a computer algebra system. We demonstrate the system performance by an example from sensitivity analysis.

Keywords: Engineering Design, Artificial Intelligence, Symbolic Computing, Computer Algebra Systems, Soar.

This work has been supported by the Engineering Design Research Center, a National Science Foundation Engineering Research Center.

1. Introduction[1]

Engineering design activity typically involves building of a mathematical model of a real world artifact. This model is initially mostly symbolic and as the designer commits to specific design choices the symbolic parameters are replaced with numeric values till finally the model is entirely numeric. Along the way various questions are asked of the model - analysis, and various transformations performed on the model - synthesis. The insights obtained with respect to the mathematical model are related back to the physical world and the designer assesses the practicality of the design. The engineering design activity is thus seen as a mapping of the physical world constraints and objectives onto a mathematical model, computations with the mathematical model to arrive at concrete values for various model parameters, and the inverse mapping of the mathematical model to the physical world to assess realizibility and suitability of the designed artifact. Tools for numeric computation have a long history in engineering design and more recently the availability of powerful machines and sophisticated computer algebra systems (CASs) has opened the door on computer support of symbolic computations as well [3, 7, 13].

Automation of intellectually challenging tasks has been a long-standing goal of Artificial Intelligence. The framework for automatic performance of numeric computations has been persuasively argued by Abelson et al. [1]. Analogously, a framework for automating the symbolic computations in engineering design can be considered. In particular, we explore the use of a computer algebra system (CAS) by an intelligent agent. Actually, a CAS itself represents an automation of symbolic computations. However, given a task to be solved, an intelligent agent is required to mediate in the use of the CAS for the task. Two approaches are possible for achieving task-level automation of symbolic computations. One approach is to write programs in a CAS for specific tasks [5]. Here, the programmer is the intelligent agent and manually performs the mapping of pre-determined task specifications into CAS programs. The second approach is to build an agent that accepts task specifications from a user and automatically uses a CAS to perform these tasks as illustrated in figure 1. This approach can potentially support open-ended symbolic computations with the mathematics of the input task specifications governing what computations are actually performed. We have developed a prototype system, Soar/Mathematica, obtained by combining Soar [8, 14] with a CAS, Mathematica [16] as an exploration of the second approach. [2] We next describe the system design for Soar/Mathematica, followed by an example from sensitivity analysis, a discussion of related work, and conclusions.

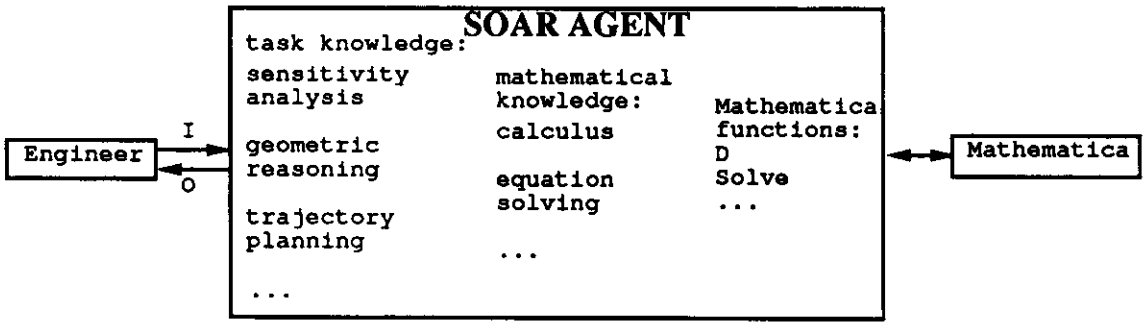

Figure 1: A mediating agent for symbolic computations using a CAS.

```
I: mathematical model, queries/transformations.
O: results of symbolic computations on models.
```

[1] We wish to acknowledge the foundational role played by Professor Allen Newell (d.) in shaping the development of this research.

[2] The central argument of this paper remains unchanged if Mathematica is replaced with any of the other readily available CASs.

2. System Design

Soar is a problem solving and learning architecture that is motivated both as a proposal for a unified theory of cognition [10] and as an integrated AI system. In Soar, all knowledge is encoded as productions and problem solving occurs in problem spaces. Each problem space has associated with it a state and a collection of operators contingent upon the problem space's state. Each problem space also has an associated collection of preference productions. Let us assume that Soar is in a problem space, P1, with state, S1. All applicable preference productions are executed in parallel and weighted preferences are added to Soar's preference memory. These are preferences for P1 operators to be applied in S1. Soar's decision procedure collects all preferences and selects an operator to apply. Operator application may add new elements to S1 and delete existing elements from S1. Consequently, the set of preference productions for P1 changes leading to a different operator choice by the decision procedure. Thus, Soar's problem solving model comprises of a sequential application of operators in a problem space. If a selected operator cannot be applied in P1, Soar generates an impasse, termed an operator no-change (ONC) impasse. [3] An impasse denotes a recognition by Soar that a new problem space is required to continue problem solving. A preference production suggests a new problem space, say P2, in response to the impasse. Processing similar to that described in P1 then occurs in P2 and along the way any number of other impasses may be generated. An impasse is resolved when the state contains elements that the operator's application were to have produced. Upon resolution of an impasse Soar resumes processing in the problem space where this impasse was generated. Soar learns new productions, termed chunks, by associating results of impasse processing with the elements in the impasse producing problem space's state that were tested in order to produce these results. Soar also has the capability to monitor any number of input data channels and to transmit data on any number of output data channels asynchronously with other problem solving.

To understand the requirements of a software program that automatically uses Mathematica for performing symbolic computations we have studied protocols of subjects using Mathematica for various engineering design tasks. An analysis of these protocols suggests that there are four primary performance processes: supervisory, formulation, interpretation, and external memory. Each of these processes is realized as a problem space in Soar as illustrated in Figure 2. In this figure, problem spaces are depicted by triangles and sp_i, f_i, em_i, and ip_i denote the various operators implementing each of the processes. There is a task space where the input task specification is represented. Each process is implemented as a set of operators in the task space contingent upon the task state representations. When a process operator is selected in the task space an ONC impasse ensues and the operator is implemented in a corresponding process space as shown in figure 2. At the outset, a bidirectional data channel is established between Soar and Mathematica using a client-server arrangement. Mathematica remains invoked throughout the interaction with it by Soar analogous to the situation that prevails when a person uses Mathematica directly. Data to be transmitted to Mathematica is represented in the task state from where it is automatically sent to Mathematica by a Soar output process and data produced by Mathematica is received by a Soar input process and represented in the task state. The organization of processes in Soar/Mathematica realizes a blackboard-style control [11, 12]. Each process is modular and can be developed independently. From a system design point of view, process modularity is a significant simplifying principle. Secondly, flexible control regimes can be obtained by simply changing the search control knowledge without modifying the individual processes. Thirdly, the representations built up by individual processes are visible to all other processes that can therefore respond to any relevant data opportunistically.

The knowledge of Mathematica used by Soar/Mathematica is of two kinds: procedural and declarative. For example, *(computes mma derivative D)*, represents the knowledge that Soar/Mathematica can use the *D* function to compute the derivative. Figure 3 illustrates the declarative representation of the *Equal* function in Mathematica. Mathematical knowledge is represented declaratively and a typical example is the fact, *(number 1)*, which asserts that *1* is a number. The performance processes of Soar/Mathematica are described as follows:

[3] Soar can respond to many other kinds of impasses as well.

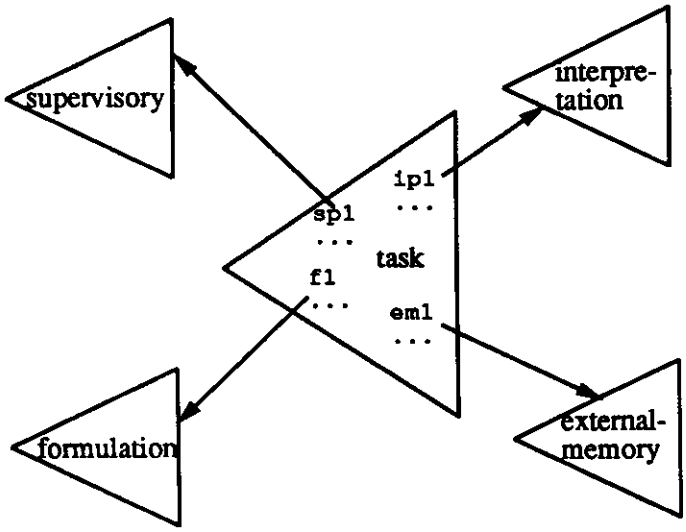

Figure 2: The problem space organization of Soar/Mathematica.

- The **supervisory** process is concerned with the mathematics of the input problem. An objective of the supervisory process is to transform the input computation into simpler computations that are directly performable. Sometimes the transformation will be *one-step*. For example, an input computation to find the derivative of a symbolic expression is one-step transformable to a computation that is simply the finding of a derivative of that symbolic expression. However, an input computation that requires finding the symbolic upper bound of a model variable subject to several symbolic inequalities may require an arbitrary number of transformations.

- The **formulation** process is concerned with the use of Mathematica for a symbolic computation. Sometimes the mapping of a symbolic computation will be direct in that a known Mathematica function will serve to perform the computation. For example, given that Soar/Mathematica knows that Mathematica computes a derivative using the D function allows for a direct mapping of a computation calling for the derivative of an expression into an expression executable in Mathematica. At other times the mapping of a computation onto suitable Mathematica input will require arbitrary problem solving. For example, if the computation is to compute the equality of two symbolic expressions and there is no direct knowledge of an appropriate Mathematica function that performs this computation then Soar/Mathematica might engage in substantial problem solving to arrive at some way of using Mathematica for the computation or finally reach a state where it accepts that the computation cannot be performed using Mathematica.

- The **interpretation** process is concerned with the mapping of the data received from Mathematica onto a meaning representation. For example, the character string, `a x^2 + b x + c == 2` is interpreted to the logical formula, *(and (equation) (lhs (and (name* `a x^2 + b x + c`*) (polynomial))) (rhs (name (2)) (constant) (number)))* which essentially says that the input from Mathematica is an equation whose left hand side is a polynomial and the right hand side is a constant.

- The **external memory** process is concerned with the management of indices into the external environment to allow for reuse of results produced previously.

3. Example

In this section we discuss an example of a symbolic computation performed by Soar/Mathematica in sensitivity analysis. Sensitivity analysis is concerned with the dependence of system behavior on system parame-

object:
id: m1
type: mma-fact
name: Equal
input: (equal x y)
output: boolean

Figure 3: The declarative specification of a Mathematica function.

ters [6]. Typical parameters are initial conditions, natural frequencies, and dead times; system behavior can be the time response, the transfer function, or any other quantity characterizing the system dynamics. This problem instance requires the calculation of system sensitivity with respect to the open-loop gain parameter, $S_K^T = \frac{\partial T}{\partial K}\frac{K}{T}$, for a system whose closed-loop transfer function is, $T(s) = \frac{K}{ts+(1+KK1)}$. [4]

The task is specified to Soar/Mathematica as illustrated in Figure 4 as an object network. The object, *o1*, is the goal whose purpose, *(S T K)*, denotes that the sensitivity is to be computed with respect to the open loop gain parameter of the system. The goal object is linked to the data objects, *d1*, *d2*, *d3*, *d4*, and *d5* that specify the various system parameters. This input task specification is added to the task space blackboard. Figure 5 illustrates the solution trace produced by Soar/Mathematica for the above input task specification. The decision cycles are numbered along the left margin [5] and the symbols G, P, S, and O stand for goal, problem space, state, and operator, respectively. An impasse is indicated by an arrow and indentation. Ellided processing prior to a decision cycle is indicated by an ellipsis in the left margin. In figure 5 we have shown only a few of the procedures of the supervisory, formulation, interpretation and external memory processes to demonstrate a working implementation of Soar/Mathematica. Soar begins processing in a problem-space called top-ps. In d3, a task operator is applied in top-ps to invoke the task problem space. In d7, there is a tie impasse as more than one operators are applicable in the task space. These are two supervisory process operators, *sp1* and *sp2*, that can be applied to the input task specification. The first operator is applicable because a definition for the input task exists and the second operator is applicable because it is a general heuristic that suggests an assessment of whether the computation can be performed by Mathematica. To resolve this impasse Soar invokes the selection space that has a search control heuristic that resolves the contention in favor of *sp1* since that is more specific than *sp2*. Therefore, in d10, Soar selects *sp1* whose application produces an impasse and leads to the invocation of the supervisory problem space. In this problem space the impasse is resolved by the application of the *use-definition* operator which is a rewrite rule for the input calculation, replacing the input specification with the definition of sensitivity. In d20, another supervisory operator is selected by Soar and the resulting impasse is resolved by applying the *split-calculation* operator in d25 that splits a compound calculation into atomic calculations. Here, the calculation, *(product (derivative T K) (quotient K T))* is split into *(derivative T K)*, *(quotient K T)*, and *(product (result (derivative T K)) (result (quotient K T)))*. In d38, Soar selects a formulation operator and the resulting impasse is resolved by the application of determine-mma-function operator that selects a Mathematica function for the calculation *(derivative T K)*. Between d42 and d181 Soar finds out appropriate arguments for the selected function by interpreting the task specification. In d181, Soar selects another formulation operator to build the input for Mathematica using the syntax for the selected Mathematica function and the names of the arguments to this function and completes this processing in d186-345. In d346, Soar selects an output operator to transmit formulated input to Mathematica by the application of the mma-transmit operator in the output problem space. I1 is the transmitted output to Mathematica and O1 is the input received from Mathematica in response. In d351, Soar applies an input operator that records the receipt of data from Mathematica. In d356, an interpretation operator is applied and the resulting impasse is resolved by the application of the build-meaning operator in the interpretation space and that maps the received data onto a meaning representation. Processing between d361 and d513

[4] The transfer function, T(s), is defined as the ratio of the Laplace transform of the output variable to the Laplace transform of the input variable under the assumption that all initial conditions are zero [9]. K, K1, and t are system parameters.

[5] In subsequent discussion we use the notation d<number> to refer to a particular decision cycle.

produces the second interaction with Mathematica and the output, I2, is sent out and data, O2, is received from Mathematica in response. In d565, Soar selects an external memory operator since the arguments of the selected Mathematica function refer to results already produced in previous interactions with Mathematica. In the external-memory space there is an operator, find-environment-index, that obtains the indices into the external environment (of Mathematica) corresponding to these previously computed results. Processing between d570 and d687 produces the third interaction with Mathematica and the output, I3, is sent out and the input, O3, is received from Mathematica in response. The entire episode takes 693 decision cycles to complete.

```
object:
    id: o1
    type: goal
    purpose: (S T K)
    data: d1 d2 d3 d4 d5

object:
        id: d1
        type: data
        description: (transfer-function)
                        (name (K/(1+K K1 + s t)))

object:
        id: d2
        type: data
        description: (parameter) (name t)

object:
        id: d3
        type: data
        description: (parameter) (name K)

object:
        id: d4
        type: data
        description: (parameter) (name K1)

object:
        id: d5
        type: data
        description: (open-loop-gain) (name K)
```

Figure 4: The input specification for the sensitivity analysis example.

4. Related Work

In this section we discuss the relationship of Soar/Mathematica to some work in automatic programming and intelligent user interfaces. Soar/Mathematica is an automatic programming system since it automatically devises the way in which a computational system, a CAS, must be used to implement a given task specification [2]. Of the many classes of programming, Soar/Mathematica has been explored in the realm of exploratory programming of the sort that occurs in the course of routine symbolic computations performed by engineers on mathematical models of design artifacts using a CAS. There are two key characteristics of this problem solving: an incremental algorithm design and immediate execution. The reason that the algorithm design is incremental is that typically the initial specification is not complete since all the implications of the model and computations specified on the model are not known to the problem solver. Therefore, the problem solver sets up sub-goals in accordance with the task mathematics and indulges in immediate perfor-

mance of any subgoal that seems direct enough. These characteristics are reflected by the Soar/Mathematica trace of Figure 5.

Soar/Mathematica can be viewed as an intelligent interface [15] to Mathematica since it allows a user to be oblivious of the knowledge of how to use Mathematica to perform symbolic computations. Soar/Mathematica elevates the level of interaction with Mathematica from that in Figure 5 to that in Figure 4. There is some work in providing expert help in the use of CASs [4]. However, Soar/Mathematica differs in its goal to undertake to perform automatically the necessary interaction with the CAS for the class of tasks considered.

5. Conclusions

In this paper we have described a system to automate the performance of symbolic computations in engineering design. The system is founded on an automatic use of a CAS. It separates the specification of computations from their implementation and can therefore enhance an engineer's efficacy. We have explored tasks that do not require treatment of traditional concerns of data structures and control flow. These tasks typically involve mathematical transformations that are directly available in CASs. Therefore, an agent requires a representation of general mathematical knowledge and a knowledge of a CAS's functionality in order to automatically carry out the input computations. We believe that Soar/Mathematica demonstrates a novel approach to task-level automation of symbolic computations in engineering design.

The development of Soar/Mathematica can proceed in two directions. The capabilities of the supervisory process can be enhanced to deal with more classes of mathematical computations. At the same time the formulation process can be enhanced to make more sophisticated use of Mathematica. An interesting twist is to use Soar's learning capability to automatically increase the capabilities of both these processes and is a direction we are currently pursuing. We note that once an input task has been interpreted by Soar/Mathematica in mathematical terms it is just another symbolic computation for it. Therefore, the core knowledge of mathematics and of Mathematica in the system will apply across tasks.

Currently, Soar/Mathematica is essentially an intelligent assistant in the performance of symbolic computations using a CAS. It can be developed into a more integrated engineering design system by the incorporation of additional task knowledge such as sensitivity of real systems. Then, it might be to possible to automate other portions of the engineering design process such as the formulation of appropriate mathematical models and assessment of the results of computations in task terms.

```
0   g: g1
1   p: p2 (top-ps)
2   s: s4 (top-state)
3   o: o10 (task)
4   ==>g: g11 (operator no-change)
5      p: p12 (task)
6      s: s13
7      ==>g: g148 (operator tie: sp1, sp2)
8         p: p149 (selection)
9         s: s150
10     o: o142 (supervisory-process: sp1)
11     ==>g: g160 (operator no-change)
12        p: p161 (supervisory-process)
13        s: s162
...
16        o: o167 (use-definition)
...
20     o: o188 (supervisory-process)
21     ==>g: g203 (operator no-change)
22        p: p204 (supervisory-process)
23        s: s205
...
25        o: o210 (split-calculation)
...
38     o: o270 (formulation-process)
39     ==>g: g273 (operator no-change)
40        p: p274 (formulation-process)
41        s: s275
42        o: o277 (determine-mma-function)
...
181    o: o286 (formulation)
182    ==>g: g825 (operator no-change)
183       p: p826 (formulation)
184       s: s827
185       o: o829 (mma-syntax)
...
346    o: o1271 (output)
347    ==>g: g1274 (operator no-change)
348       p: p1275 (output)
349       s: s1276
350       o: o1278 (mma-transmit)

I1:
D[Times[K,Power[Plus[1,Times[K,K1],Times[s,t]],-1]],K]

O1:
((Plus
  (Times -1
   (Times (Times K K1)
    (Power (Power (Plus 1 (Times K K1) (Times s t)) 2) -1)))
  (Power (Plus 1 (Times K K1) (Times s t)) (Times -1 1))))
```

Figure 5: The problem solving trace produced by Soar/Mathematica on the sensitivity analysis example.

```
351     o: o1287 (input)
352     ==>g: g1338 (operator no-change)
353        p: p1339 (input)
354        s: s1340
355        o: o1342 (result-received)
356     o: o1355 (interpretation)
357     ==>g: g1359 (operator no-change)
358        p: p1360 (interpretation)
359        s: s1361
360        o: o1363 (build-meaning)
...
512     o: o1965 (formulation)
513     o: o1979 (output)

I2:
Times[K,Power[Times[K,Power[Plus[1,Times[K,K1],Times[s,t]],-1]],-1]]

O2:
((Plus 1 (Times K K1) (Times s t)))

514     o: o1987 (input)
515     ==>g: g2003 (operator no-change)
516        p: p2004 (input)
517        s: s2005
518        o: o2007 (result-received)
519     o: o2020 (interpretation)
520     o: o2031 (supervisory-process)
521     o: o2037 (interpretation)
522     o: o2045 (supervisory-process)
...

565     o: o2204 (external-memory-process)
566     ==>g: g2248 (operator no-change)
567        p: p2249 (external-memory-process)
568        s: s2250
569        o: o2252 (find-environment-index)
...
686     o: o2748 (formulation)
687     o: o2762 (output)

I3:
Times[Out[1],Out[2]]

O3:
((Times (Plus 1 (Times K K1) (Times s t))
  (Plus
   (Times -1
    (Times (Times K K1)
     (Power (Power (Plus 1 (Times K K1) (Times s t)) 2) -1)))
   (Power (Plus 1 (Times K K1) (Times s t)) (Times -1 1)))))

688     o: o2770 (input)
689     ==>g: g2833 (operator no-change)
690        p: p2834 (input)
691        s: s2835
692        o: o2837 (result-received)
693     o: o2850 (interpretation)
```

Figure 5 continued.

References

[1] H. Abelson, M. Eisenberg, M. Halfant, J. Katzenelson, E. P. Sacks, G. J. Sussman, J. Wisdom, and K. Yip. Intelligence in scientific computing. *Communications of the ACM*, 32(5):546–562, May 1989.

[2] A. Barr and E. A. Feigenbaum. *The Handbook of Artificial Intelligence*, volume 2, chapter 10. Addison Wesley, 1982.

[3] T. Cline, H. Abelson, and W. Harris. Symbolic computing in engineering design. *AI EDAM*, 3(3):195–206, 1989.

[4] R. P. dos Santos and W. L. Roque. On the design of an expert help system for computer algebra systems. *SIGSAM Bulletin*, 24(4):22–5, October 1990.

[5] B. L. Evans and J. H. McClellan. Symbolic analysis of signals and systems. In A. V. Oppenheim and S. H. Nawab, editors, *Symbolic and Knowledge-Based Signal Processing*. Prentice Hall, Engelwood Cliffs, New Jersey, 1992.

[6] P. M. Frank. *Introduction to System Sensitivity Theory*. Academic Press, 1978.

[7] R. Grossman. *Symbolic Computation: Applications to Scientific Computing*. SIAM, 1989.

[8] J. E. Laird, A. Newell, and P. S. Rosenbloom. Soar: An architecture for general intelligence. *Artificial Intelligence*, 33:1–64, 1987.

[9] I. J. Nagrath and M. Gopal. *Control Systems Engineering*. Wiley Eastern, 1982.

[10] A. Newell. *Unified Theories of Cognition*. Harvard University Press, 1990.

[11] H. P. Nii. The blackboard model of problem solving. *AI Magazine*, 7(3):38–53, 1986.

[12] H. P. Nii. Blackboard systems part two: Blackboard application systems. *AI Magazine*, 7(3):82–106, 1986.

[13] R. Pavelle. *Applications of Computer Algebra*. Kluwer Academic Publishers, 1985.

[14] P. S. Rosenbloom, J. E. Laird, A. Newell, and R. McCarl. A preliminary analysis of the soar architecture as a basis for general intelligence. *Artificial Intelligence*, 47(1-3):289–325, 1991.

[15] J. W. Sullivan and S. W. Tyler. *Intelligent User Interfaces*. Addison-Wesley, Reading, Mass., 1991.

[16] S. Wolfram. *Mathematica*. Addison-Wesley, second edition, 1991.