Purposive Discovery of Operations

Michael Sims

John Bresina

Artificial Intelligence Research Branch

MS 269-2

NASA Ames Research Center

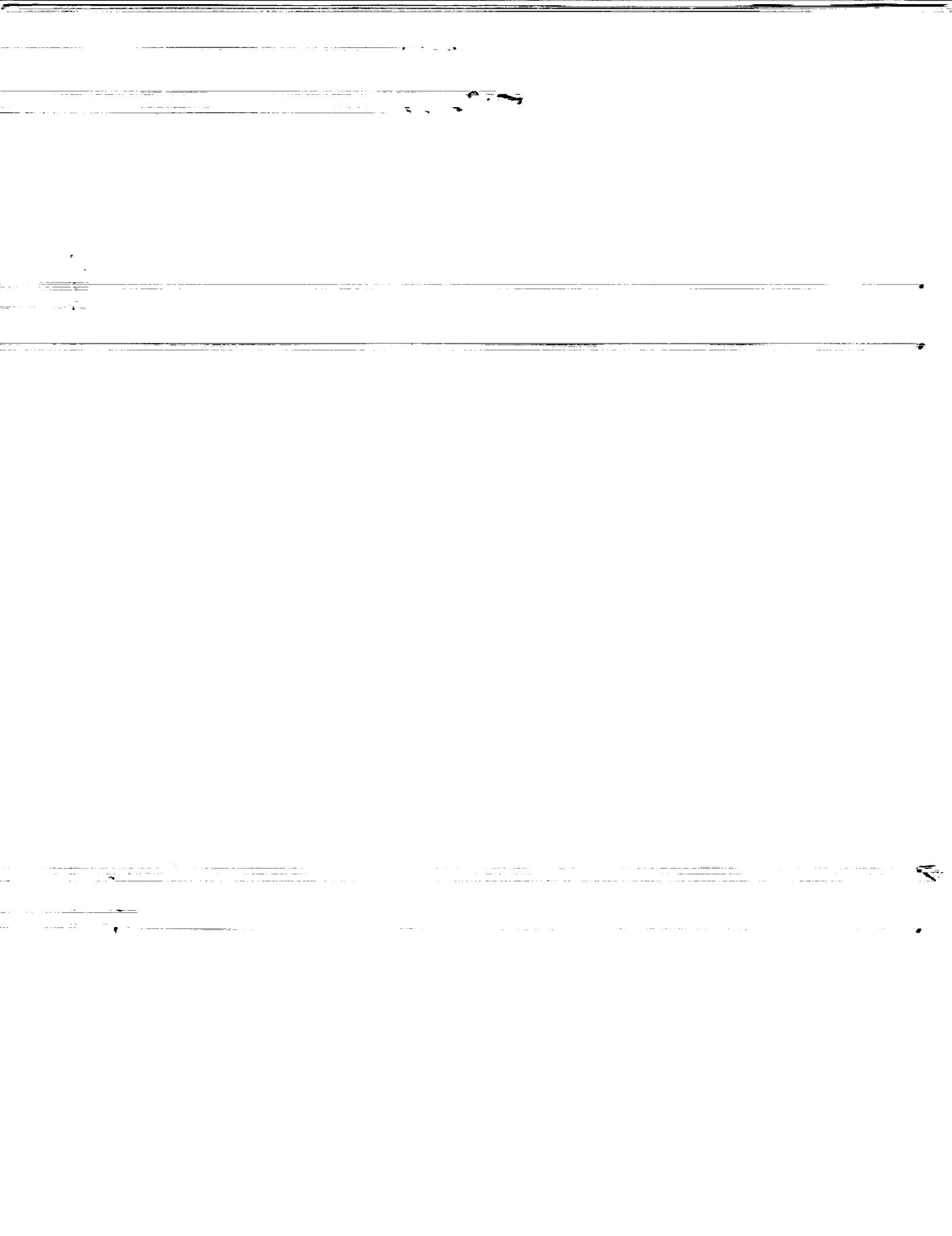
Moffett Field, CA 94035-1000

NASA Ames Research Center

Artificial Intelligence Research Branch

Technical Report FIA-92-27

July, 1992



Purposive Discovery of Operators

Michael H. Sims
NASA Ames Research Center
Mail Stop 269-3, Moffett Field, CA 94035-1000, USA
email: Sims@ptolemy.arc.nasa.gov

John L. Bresina
Sterling Software
NASA Ames Research Center
Mail Stop 269-2, Moffett Field, CA 94035-1000, USA

July 1992

Abstract

This paper introduces the *Generate, Prune & Prove (GPP)* methodology for discovering definitions of mathematical operators. GPP is a task within the IL exploration discovery system. We developed GPP for use in the discovery of mathematical operators with a wider class of representations than was possible with the previous methods by Lenat and by Shen. GPP utilizes the *purpose* for which an operator is created to prune the possible definitions. The relevant search spaces are immense and there exists insufficient information for a complete evaluation of the purpose constraint, so it is necessary to perform a partial evaluation of the purpose (i.e., pruning) constraint. The constraint is first transformed so that it is operational with respect to the partial information, and then it is applied to examples in order to test the generated candidates for an operator's definition. In the GPP process, once a candidate definition survives this empirical prune, it is passed on to a theorem prover for formal verification. In this paper, we describe the application of this methodology to the (re)discovery of the definition of multiplication for Conway numbers, a discovery which is difficult for human mathematicians. We successfully model this discovery process utilizing information which was reasonably available at the time of Conway's original discovery. As part of this discovery process, we reduce the size of the search space from a computationally intractable size to 3468 elements.

1. Introduction

This paper addresses a method for the automatic discovery of operators in mathematical domains. Our research indicates that the use of often implicit information, such as an operator's purpose, can be crucial in controlling such a discovery. When we invent concepts, we often have a specified purpose which the concept is intended to satisfy. In this paper, we examine the creation of mathematical operators which satisfy their intended purposes. In particular, we have undertaken the investigation of the computer discovery of elementary operations of number systems (e.g., addition of integers).

Purposive, structural (e.g., complex numbers having real and imaginary parts), and problem decomposition information seem crucial to the realistic computer modeling of much of mathematical reasoning. Further, the ability to reason with and about partial information regarding the mathematical objects at hand is often required. It is only by such reasoning that we can constrain the size of the relevant search spaces in many problems. This work is a case study in the realistic modeling of the discovery of Conway multiplication, with implications to the information and processes required for the modeling of a broad spectrum of mathematics.

A generation ago, the results of symbolic manipulation of mathematical formulas was more promise than utility, and today we find wide use of systems such as Macsyma and Mathematica. Similarly, today there is great promise for the future use of sophisticated software to aid in the *discovery* of mathematics. The research presented in this paper is one task (i.e., subprogram) in an ambitious project (the IL project [Sims 90]) to push the limits of automatic discovery of mathematics in the domain of Conway numbers. The presented task was developed and subsequently implemented to enable the discovery of an important class of operators.

In this paper, we present *Generate, Prune, & Prove (GPP)*, a specialization of generate and test for the discovery of mathematical operator definitions. In the Generate, Prune and Prove process (see figure 1):

- a conjecture is generated in some language using heuristic search (*generate phase*);

- that conjecture is evaluated with respect to applicable constraints on examples (*prune phase*);
- if the conjecture successfully passes the prune phase then a theorem prover is used to formally verify the conjecture (*prove phase*).

In this paper's example of GPP we search over the space of operator definitions; the search is drastically reduced by using problem decomposition to search over partial definitions of these operators. In GPP, to discover a new operator definition one needs: (i) the purpose of the operator, (ii) a language for specifying a definition, (iii) methods for testing a candidate definition, and (iv) ways to control these activities. We next summarize the GPP method; the details of GPP are illustrated in later sections with the example of the discovery of a multiplicative operator for the domain of Conway numbers [Conway 76]. A description of the application of this methodology to the simpler case of multiplication of complex numbers can be found in [Sims & Bresina 89]. The prove phase will not be described in detail in this paper.

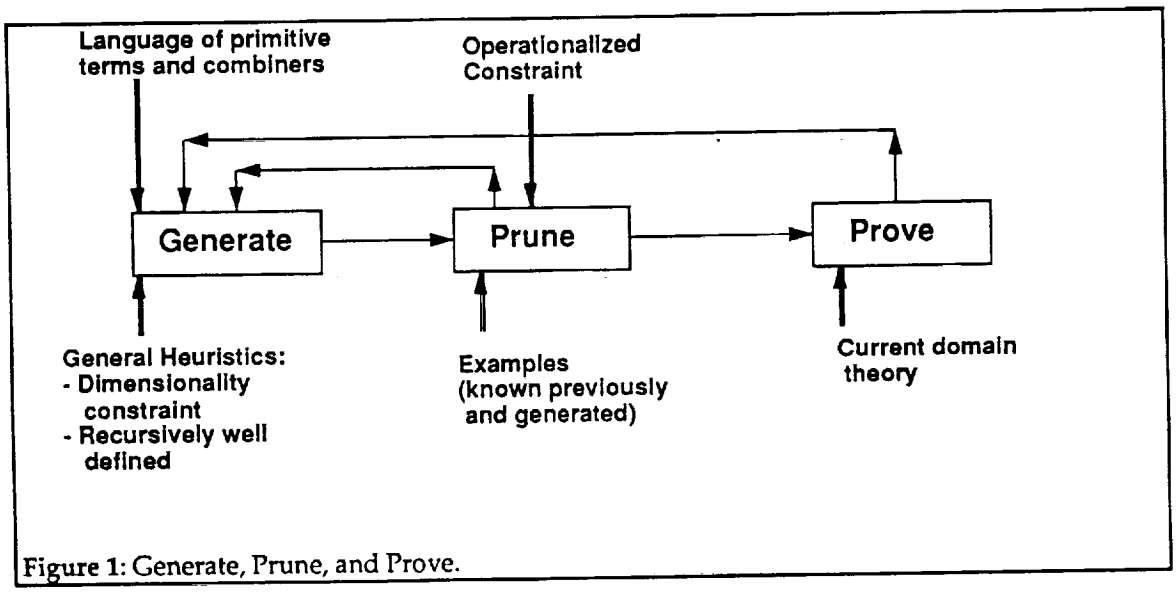


Figure 1: Generate, Prune, and Prove.

The GPP process consists of the generation of candidates, the pruning of those candidates on examples, and finally a formal validation using a theorem prover (see figure 1). During the generate phase, candidate operator definitions (or partial definitions) are generated in increasing order of complexity. The generation process is biased by heuristics regarding what operator definitions might look like. The intended purpose of the operator is

used to specify a pruning constraint that is made operational for the current problem and to specify a set of pruning examples. During the prune phase, a candidate definition is tested using the pruning constraint and examples; if, on any example, the constraint (applied to the candidate) is not satisfied, then the candidate is pruned from the search space. During the prove phase, a theorem prover attempts to prove the conjecture that a candidate definition satisfies its intended purpose. This analytic prune (i.e., prove phase) is applied after the empirical prune (i.e., prune phase) has reduced possible candidates to a small number. The rationale for the control flow between the three phases of GPP is based on their relative expected computational costs. Usually, the prove phase is computationally the most expensive phase; hence, we try to get as much pruning as possible from the empirical prune phase — which is why our heuristics call for expending effort strengthening the pruning constraint if it appears that it is satisfied by too many candidates.[1]

In the remainder of this paper, we discuss the use of the generate and prune phases of GPP to discover the definition of the multiplicative operator for Conway numbers. This discovery is subtle even for human mathematicians. The knowledge used in this computer discovery would have been realistically available to John Conway at the time of his original discovery. This knowledge includes that a purpose of the definition for Conway multiplication is to enable Conway numbers to be a field, and for the natural mapping from the reals to the Conway numbers to be a homomorphism under this definition of multiplication.

This research was stimulated by previous work [Lenat 77; Shen 90] on the discovery of mathematical operators. Lenat's original work and Shen's refinement of it into an elegant, functional transformation framework depend upon a special representation of operator definitions, which typically is not appropriate for mathematical operators. We extend their pioneering work by developing a methodology applicable to a wider class of representations.

---

[1] These strengthening of constraint heuristics were designed but are not yet implemented.

Our approach to this work is in the spirit of Macsyma or Dendral [Barr & Feigenbaum 82], in which we accept a hard, real world problem and attempt to engineer a solution to that problem using the best tools available, which often come from a knowledge engineering of how humans solve those problems. Our concern is not so much as to whether the mathematics has the best representation, since as mathematicians we often begin with suboptimal representations. Rather our emphasis is on trying to adequately and realistically capture the mathematical reasoning. Our attention on the quality of handling our single problem is in the spirit of those earlier pioneering works; however, it is somewhat in variance with the current trend in machine learning which emphasizes generality, often in the form of shallow analysis on a large number of data sets. Our opinion is that generality is of little use if we cannot handle the single problem in realistic detail, and there are no computer systems in existence today which are capable of adequately handling the class of problems we describe here. However, we strongly believe that a Dendral or Macsyma level of effort will make this reasoning accessible and will make available an immensely useful tool to the mathematics community. In this paper, we describe our first efforts in that direction.

We next present some background material to help understand the discovery case study. We have investigated GPP in the context of a general exploration discovery system, called IL. We begin by briefly describing the IL discovery system and then the problem domain of Conway numbers.

1.1 The IL Discovery System

We have developed an exploration discovery system called IL[2] [Sims & Bresina 89; Sims 90]. In the spirit of Lenat's AM [Lenat 77; Bundy 83][3], IL begins with a core of domain knowledge and control heuristics and then opportunistically expands its knowledge. In addition to reasoning via the

---

[2] IL is named for Imre Lakatos, a philosopher of mathematics [Lakatos 76].

[3] Lenat's follow up system, Eurisko [Lenat 83], extended our understanding of representational issues, but didn't represent an extension of AM in the realm of mathematical reasoning.

empirical techniques utilized by AM, IL can reason via analytic techniques; e.g., theorem proving and explanation-based learning. Similar to AM, IL has an agenda-based control structure with tasks and heuristics. Processes, such as GPP, are implemented within IL as a blend of specific tasks and heuristics. The version of GPP described in this paper has never been fully integrated with the rest of IL, largely due to the computational overhead of running the entire system together. The version of GPP here described ran as a stand alone program, and utilized some of the subcomponents of the full system such as the theorem prover. However, the representations used by this GPP are compatible with the rest of IL, and we have generated heuristics for enabling such an integration. Previous versions of GPP were integrated as IL tasks [Sims 90].

IL's theorem proving uses a heuristically-controlled, depth-first, natural-deduction theorem prover which is depth limited. Although the theorem prover can prove only simple conjectures, its reasoning and representation are available to the rest of IL's components. IL's theorem prover was not used in this paper's example to verify that the operator definition that survived the prune phase was, in fact, a valid Conway multiplication operator. The reason is that the theorem prover is not sufficiently powerful to verify expressions of that complexity on our computational hardware. It was, however, used as an integral part of the GPP reasoning process to verify a multitude of Conway expressions (e.g., $\overline{-1} \leq \overline{1}$) which do require formal proofs.

1.2 The Problem Domain of Conway Numbers

Conway numbers[4] can be viewed as a super-class of the real numbers which also contains infinite and infinitesimal numbers. As a domain of mathematics for computer discovery research, Conway numbers have the following advantages: (i) the formal domain description is sufficiently parsimonious to allow a realistic representation of much of the underlying

[4] Conway numbers are also referred to as *Surreal numbers* .

mathematics in IL, and (ii) Conway numbers are a new area of mathematics, so some of the interesting discoveries yet to be made may be simple enough to permit IL to make a real contribution to the mathematics. In this section, we briefly present the intuition behind Conway numbers.

Conway numbers can be viewed of as a generalization of both the Dedekind cut generation of the reals and Cantor and von Neumann's generation of the ordinals. A Conway number is a recursively defined object with a left set, L, and a right set, R, which we write as <L | R>. Each of the left and right sets is either empty or contains (possibly uncountably many) Conway numbers. By definition, no member of the left set is greater than or equal to any member of the right set. Hence, a Conway number can be thought of as sitting (somewhere) between its bounding left and right sets. By the notation $x = \langle\{...,x^L,...\} \mid \{...,x^R,...\}\rangle$, we mean that the Conway number x has x^L as a typical element of the left set and x^R as a typical element of the right set.

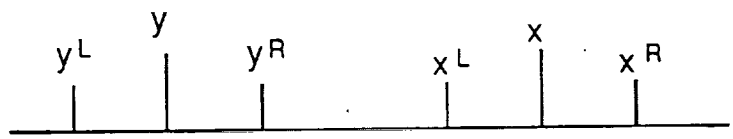


Figure 2 Conway numbers x and y with typical elements of their left and right sets.

Since Conway numbers are recursively defined in terms of left and right sets of Conway numbers, one begins the recursive generation by initially (on "day 0") defining a Conway number with an empty left set and an empty right set. We will call this Conway number $\overline{0}$, so $\overline{0} = \langle\{\} \mid \{\}\rangle$. Now we have one Conway number, so we can define two new Conway numbers on the next "day" (i.e., day 1). They are: $\overline{1} = \langle\{\overline{0}\} \mid \{\}\rangle$ and $\overline{-1} = \langle\{\} \mid \{\overline{0}\}\rangle$. This means, for example, that the left set of $\overline{-1}$ is the empty set and the right set contains the Conway number $\overline{0}$. On day 2, we can use the results of day 1 and define the new Conway numbers: $\overline{-2} = \langle\{\} \mid \{\overline{-1}\}\rangle$, $\overline{-\frac{1}{2}} = \langle\{\overline{-1}\} \mid \{\overline{0}\}\rangle$, $\overline{\frac{1}{2}} = \langle\{\overline{0}\} \mid \{\overline{1}\}\rangle$ and $\overline{2} = \langle\{\overline{1}\} \mid \{\}\rangle$. We continue this definition process on successive "days" forever. Hence, there is a natural notion of the *age* of a Conway number (i.e.,

the objects with the smallest day of birth being the oldest), and most definitions of Conway operators are defined via recursion over this age. There is a homomorphism of the reals into the Conway numbers, and under this mapping the real numbers 0, 1, -1 will map into $\overline{0}$, $\overline{1}$, and $\overline{-1}$, respectively.

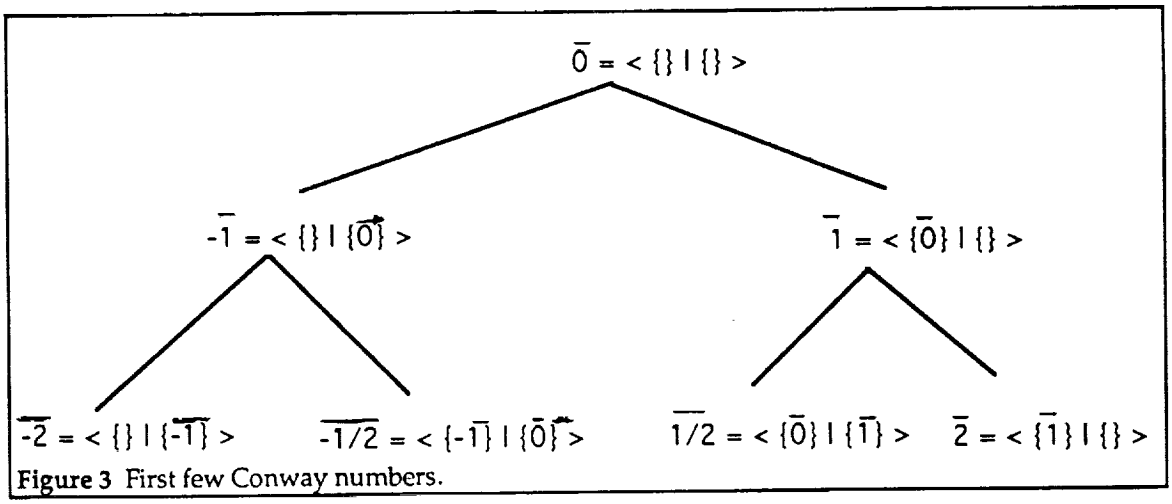


Figure 3 First few Conway numbers.

Equality and inequality are defined relations on Conway numbers with the following definitions:

- $x \geq y$ is defined by (no $x^R \leq y$) and ($x \leq$ no y^L)
- $x \leq y$ is defined by $y \geq x$
- $x \equiv y$ is defined by ($x \geq y$) and ($x \leq y$)

In the above, "no $x^R \leq y$" means that there is no element of the right set of x (i.e., no x^R) which is less than or equal to y. Note, these are recursively defined over the *age* of the Conway numbers.

Under Conway equality, $\equiv$, two objects may be equal and yet have different left and right sets (i.e., different representations). For example, $\langle \{\} \mid \{\} \rangle \equiv \langle \{\overline{-1}\} \mid \{\overline{2}\} \rangle$. Conway equality defines an equivalence class over these different representations, and it is these equivalence classes which we refer to as Conway numbers and which contain the subclass which is isomorphic to the reals and which has the property of being a totally ordered field. Certain properties, such as associativity, work only for the equivalence classes and do not necessarily imply a strict identity of the underlying elements. The above recursive-day procedure for generating Conway numbers gives a canonical

representation for a Conway number, and we will often assume that a Conway numbers representation is that canonical one. One characteristic of the canonical representation is that the elements of the left and right sets are older than the number represented (i.e., $\overline{1} = <\{\overline{0}\} \mid \{\}>$ and $\overline{0}$ is older than $\overline{1}$).

To define an operator which has Conway numbers as its range, we need to specify how to generate its left and right sets. In general, there can be more than one set generator for a left side and more than one set generator for the right side, in which case, the resulting left (right) side is the union of the sets generated by each left (right) set generator. Consider the standard operator definition for Conway addition:

- $x + y$ is defined by $<\{x^L + y, x + y^L\} \mid \{x^R + y, x + y^R\}>$

The left and right sets of the definition each contain two arithmetic expressions involving typical elements of the arguments (x and y). We refer to an expression such as $x^L + y$ as a *set generator* because computationally each typical element in the expression acts like an iteration variable over its corresponding set. The sets of Conway numbers generated by the two set generators in the left set of the definition are unioned together to form the left set of the answer (i.e., the sum), and analogously for the right set.

The generator $x^L + y$ generates an element of the left hand side for each element in the left set of x; hence, if x has an empty left set, then no elements are generated. If $x = y = \overline{1} = <\{\overline{0}\} \mid \{\}>$, the only x^L is $\overline{0}$, the only y^L is $\overline{0}$, and there are no y^R's nor x^R's. Hence, on the example $\overline{1} + \overline{1}$ the generator $x^L + y$ contributes $\{\overline{0} + \overline{1}, \overline{1} + \overline{0}\}$ to the left set. We then recursively evaluate $\overline{0} + \overline{1}$ and $\overline{1} + \overline{0}$ (which both yeild $\overline{1}$), and compute the union of the left sets.

Conway gives the following four set generators as comprising the standard (recursive) definition of the multiplicative operator for Conway numbers (i.e., x*y). These expressions can be viewed as algebraic combinations of terms

such as x, y^R (i.e., a typical element of the right set of y), etc. This standard definition is written as[5]:

- $x*y = <\{ x^L*y + x*y^L + (-x^L*y^L),\ x^R*y + x*y^R + (-x^R*y^R)\} \mid$
  $\{ x^L*y + x*y^R + (-x^L*y^R),\ x^R*y + x*y^L + (-x^R*y^L)\}>$

Breaking this expression apart we have four set generators:

- Left set generators:
  $x^L*y + x*y^L + (-x^L*y^L)$
  $x^R*y + x*y^R + (-x^R*y^R)$
- Right set generators:
  $x^L*y + x*y^R + (-x^L*y^R)$
  $x^R*y + x*y^L + (-x^R*y^L)$

We refer to this operator for Conway multiplication as *conway_mult*, or *. To illustrate the meaning of this definition, consider the computation of $\overline{2}$ * $\overline{-2}$, where $\overline{2} = <\{\overline{0}, \overline{1}\} \mid \{\}>$ and $\overline{-2} = <\{\} \mid \{\overline{-1}, \overline{0}\}>$. The left set of the result is empty because in the first left set generator, y^L, iterates over the empty left set of $\overline{-2}$ and in the second left set generator, x^R, iterates over the empty right set of $\overline{2}$. The right set of the result is the union of $\{\overline{-3}, \overline{-2}, \overline{0}\}$ (from the first right set generator) and the empty set (from the second right set generator). Hence, the result is $<\{\} \mid \{\overline{-3}, \overline{-2}, \overline{0}\}>$.

For more details on Conway numbers, see [Conway 76]. Gonshor's book [Gonshor 86] also gives a good presentation of the theory of Conway numbers, but the approach and notation are different from that presented here. There is a light hearted novelette by Knuth [Knuth 74] which lets the reader discover the fundamental ideas of Conway numbers from the perspective of two people on a beach who find tablets with intriguing hints.

---

[5] In IL an expression such as $x^l*y + x*y^r + (-x^l*y^r)$ is represented as the following typical element expression (te_exp):

```
[te_exp,[Generated_Set,Ans6,[[te,[XLS,XL]],[te,[YRS,YR]]],
      [and,[[cmult,[XL,Y,Ans1]],[cmult,[X,YR,Ans2]]
            [cadd,[Ans1,Ans2,Ans3]],[cmult,[XL,YR,Ans4]],
            [cnegate,[Ans4,Ans5]],[cadd,[Ans3,Ans5,Ans6]]]]]].
```

By [te,[XLS,XL]], we mean that XL is a typical element of the set XLS which is the left set of X.

1.3 Problem Statement

Discovering a Conway operator definition thus involves finding some number of left set generators and some number of right set generators; ideally, the definition should not contain redundant generators.

Although the definitions of Conway addition and Conway negation (i.e., $-x = \langle\{-x^R\} \mid \{-x^L\}\rangle$) are relatively straightforward, the definition of Conway multiplication is not. Conway does give motivation of the definition for Conway multiplication, however, it is clear that finding the definition requires considerable search for humans. The (re)discovery of this multiplication operator, *conway_mult*, is the case study used in this paper to illustrate the GPP method. Note that *a priori* knowledge of the above definition for Conway multiplication was *not* used by IL in the discovery process.

Although it is not our intention to do a protocol analysis of this discovery, it is interesting to note the comments on this discovery by Conway. He states *"It takes a great deal of thought to find the correct definition [for Conway multiplication.]"* [Conway 76, p. 6]. He also stated that after he had developed the basic theory of Conway numbers *"Only several weeks' hard thought, sustained by the conviction that there must be a 'generic' definition, finally led to the 'correct' formula [for Conway multiplication]."* [Conway 76, p. 27]. So even for one of the outstanding mathematicians of our era this discovery is not a simple exercise.

This definition of Conway multiplication is the simplest definition which survived our implementation's GPP's empirical prune given the intended purpose for which this operator is created. Conway had similarly described his discovery, *"We can summarize by saying that the definitions of the various operators and relations are just the simplest possible definitions which are consistent with their intended properties [i.e., the purpose]."* [Conway 76, p. 6].

In table 1 we summarize the problem being addressed. Since it was our intention to push the limits of autonomous discovery in mathematics, our interests lie in the automatic generation of the problem statement in table 1. The discussion of that generation is outside the scope of this paper. On the other hand, in an implementation of a mathematician aid or apprentice, it could well be that the mathematician would explicitly supply the problem statement.

Table 1: Problem specification for discovery of Conway Multiplication

Case Study Specification:

Given:

- Reals (objects and defining operators)
- Reals are a field
- Knowledge that there exists an isomorphism from the reals to a subset of the Conway numbers
- The form of that homomorphism map on the ordinals (integers) into the Conway numbers[6]; e.g., {0,1} -> $<\{\overline{0}, \overline{1}\} \mid \{\}>$. That is, the only part of the homomorphism that we explicitly know is the map on the integers.
- Conway number objects
- The operator definitions of Conway addition and negation

Find:

- Conway multiplication operator's definition such that:
  - the above homomorphism holds
  - Conway numbers are a field
  - this definition of multiplication preserves the ordinal map

(i.e., such that it satisfies the *purpose* for Conway multiplication)

---

[6] In part, the justification for this map can be found in the Cantor/von Neumann description of ordinal numbers. For example, the ordinal number 2 is associated with the set {0, 1}. The specified mapping maps the real ordinal 2 onto $<\{\overline{0}, \overline{1}\} \mid \{\}>$, which we label as $\overline{2}$; similarly for the other ordinals.

According to table 1, we want to discover the definition of Conway multiplication given its *purpose*. The purpose is:

(a) the homomorphism holds between the reals and Conway numbers
(b) the homomorphism preserves the natural map of ordinals
(c) Conway numbers are a field

Each of these objects (e.g., reals, field) have definitions which can be further expanded.

In IL, this purpose is represented as a conjunction of three predicates that correspond to these properties (a, b, and c). This full purpose is a constraint on the definition of Conway multiplication. However, for several reasons, this purpose constraint is not evaluable as written, since, for example, we do not know the full definitions of the homomorphism or Conway multiplication. We wish to transform this full definition until it is operationalized (i.e., consistently evaluable - see section 2.2.1) for the prune phase of GPP. Alternatively, we may view this full purpose constraint as the goal to be satisfied by a design synthesis process, where we are designing a new operator [Amarel 86].

We next describe the three phases of GPP: *generate* – constrained generation of candidate definitions, *prune* – pruning of candidate definitions via examples, and *prove* – analytic pruning with IL's theorem prover.

2. Generate, Prune and Prove

2.1 The Generate Phase

The first phase of GPP generates expressions in a language of operator definitions which are later empirically tested. The space of expressions that is searched consists of candidate set generators of the left and right sets. The space of set generators is searched in order of increasing complexity (i.e., the number of combiners) in a breadth-first fashion. This search space is built out of a set of objects and combiners related to the to-be-defined operator (in this case *conway_mult*).

We begin our recursive generation of the set generators at the 0^{th} complexity level. For Conway operators, this level contains the variables[7] and typical elements of their left and right sets, as well as their negations[8]; i.e., the set {x, x^L, x^R, y, y^L, y^R, -x, $-x^L$, $-x^R$, -y, $-y^L$, $-y^R$}.[9] The n^{th} level of complexity is generated by n applications of combiners to the 0^{th} level objects. The determination of which combiners to use is discussed next.

Notice that for the reals, integer multiplication can be defined in terms of addition. This same relation holds between exponentiation (over the integers) and multiplication. Hence, for the reals, there is a natural complexity order (low to high) of addition, multiplication, and exponentiation. We use the mapping between reals and Conway numbers to impose the same complexity ordering for Conway operators. Then we can define the set of combiners as the to-be-defined operator and all lower complexity operators. For *conway_mult*, the set contains *conway_mult* (i.e., *) and *conway_add* (i.e., +). Hence, for example, the 1^{st} level includes (-x)+y, $x+x^L$, $x^L+(-x^R)$, x^L+x^L, $x^R+(-y)$, x^L+y^R, (-y)*y, $x*x^L$, $x^L*(-x^R)$, x^R*x^R, $x^R*(-y)$, x^L*y^R. If we were trying to discover exponentiation for Conway Numbers, then the set of combiners would contain *conway_exp*, *conway_mult*, and *conway_add*.

Looking for *conway_mult*'s definition in such an immense search space, without strong guidance, is not currently computationally feasible in the context of a general purpose discovery system, such as IL. We incorporated (into the candidate generator) the following additional constraints, in the form of heuristics, which acts to sufficiently reduce this space to enable IL's discovery of the standard definition of Conway addition, Conway multiplication, complex addition and complex multiplication. These

---

[7] Since *conway_mult* is a binary operator we use two variables (i.e., x and y) at this level.

[8] More generally, the 0^{th} complexity level of an operator, consists of the appropriate variables and their inverses under operators of lower complexity. See the paragraph that follows.

[9] For operators over complex numbers, the 0^{th} complexity level contains the variables and their real parts and imaginary parts, as well as their negations [Sims & Bresina 89].

heuristics capture "rules of thumb" which, although not necessarily always true, are useful in focusing the system's strategy and attention.

- **Recursive Heuristic**: In recursive Conway definitions, the age of any set generator must be greater than the age of the object being defined (which is, in this case, x*y).
- **Dimensionality Heuristic**: The set generators in an operator definition should have similar dimensional characteristics to that of the operator.[10]

The recursive heuristic is intended to ensure that the definition is well-formed with respect to recursion. In the case of *conway_mult*, this constraint means that x*y should not appear in its own definition, but $x*x^L$, x^L*x^R, x^L*x^L, x^R*y may all appear (because the typical elements of a Conway number are *older* than that Conway number). To motivate the idea behind the dimensionality heuristic, consider the following example. When defining the area of a geometric object, one expects the definition to contain terms with dimension of length-squared but, for example, not terms which have dimensions of length-cubed or higher powers. Analogously to this example, we expect the definition of *conway_mult*(x,y) to contain expressions involving the product of two terms (e.g., x^L*y^R or x^L*x^R), but not expressions involving the product of three (or more) terms (e.g., $x^L*x^R*y^L$), nor expressions which do not involve products at all (e.g., $x^L + y^R$).

These heuristic constraints substantially reduce the number of candidate set generators considered in finding *conway_mult*'s definition. Any subset of these set generators can appear in the left set, and any subset can appear in the right set of a candidate definition. Hence, the space of complete candidate *conway_mult* definitions is the cross product of the power sets of the set of these candidate set generators. If the number of set generators considered is N, then this space contains 2^{2N} candidate complete definitions. In the discovery of *conway_mult*, 1734 set generators were considered before finding the correct definition, and in the next section, we address the issue of how to efficiently search this space of 2^{3468}complete candidate definitions. It's worth

---

[10] For another use of dimensionality in discovery pruning applied to physical laws, see [Kokar 86]. Also, see [Bhaskar & Nigam 90].

noting that this space really is huge and no real system could exhaustively search it.[11] In order to address such a problem you have to be cleverer and the rest of the paper discusses the methods that we used to turn this into a tractable problem.

The above heuristics may eliminate certain operator definitions that are desired. If we were to re-implement the system anew, we would further investigate a better set of heuristics to make manageable the creation of the candidate generators. Our point with respect to these heuristics is that there exists useful heuristics which can be moved directly into the generate phase to strongly control the generation process. We have used two heuristics that make sense for recursively defined arithmetic operators, and work for the operators we have tested, but they may well not work for other operators.

2.2 The Prune Phase

In the second phase of GPP, the candidate definitions are subjected to an empirical prune over a set of examples. If one imagines drawing a box around the generate and prune phases, we can view that new box as the generator for the prove phase (see figure 1). Then the overall effect of having the empirical prune phase is to have moved an efficient (pre)test for the conjecture into the generator for this prove phase. As should become clear in sections 2.2.1 and 2.2.2, the constraints that we use in this prune phase are provably correct, in the sense that any candidate which is pruned is provably not a valid candidate.

The pruning constraint is derived by IL from the purpose of the operator. As an example, the purpose for *conway_mult* is, in part, that there be a *field homomorphism* between the reals and Conway numbers. The following expression is part of the definition of this field homomorphism:

- $(\forall\ r_1 \in \text{reals})\ (\forall\ r_2 \in \text{reals})\ [\phi(r_1 r_2) \cong \phi(r_1) * \phi(r_2)]$ (1)

---

11 To get a rough measure of the size of this search space, note that if you computed in parallel on every electron, proton, and neutron within 10 billion light years of us then you would have to evaluate more than 2^{3000} complete definitions on each of those particles each picosecond since the Big Bang. Neither faster computers nor parallelism will ever be of any help.

Here, ϕ is a one-to-one homomorphism of the reals into the Conway numbers, the multiplication of reals is represented by concatenation (i.e., $r_1 r_2$), *conway_mult* is represented by $*$, and $\equiv$ is the equality of Conway numbers.

Such a homomorphism constraint is mathematically natural, and it here proves to be powerful as an empirical pruning constraint. In the *conway_mult* discovery, this constraint allowed IL to effectively prune all generated incorrect candidate definitions. In the case of other operator discoveries (e.g., the discovery of the multiplication operator for complex numbers) other elements of the purpose (e.g., multiplicative identity property) may prove valuable for the empirical pruning.

We assume here that IL has only partial knowledge of ϕ. In particular, IL only knows the restriction of ϕ to integers. Hence, if the candidates we test are *complete* definitions of *conway_mult*, then the homomorphism constraint is evaluable when r_1 and r_2 are integers. We next describe how partial knowledge is handled.

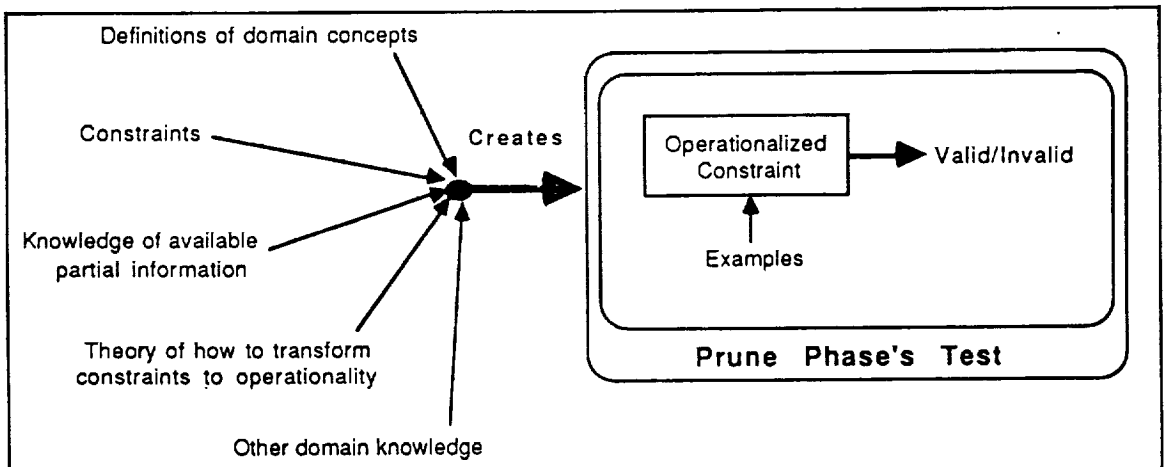


Figure 4 Constraint operationalization process.

2.2.1 Partial candidate definitions

Consider again the full definition of Conway multiplication:

$$x*y = <\{ x^L*y + x*y^L + (-x^L*y^L),\ x^R*y + x*y^R + (-x^R*y^R)\} \mid \{ x^L*y + x*y^R + (-x^L*y^R),\ x^R*y + x*y^L + (-x^R*y^L)\}>.$$

In the generate phase of GPP, IL generates N set generators, such as x^L*y or $x^L*y + x*y^L$. Since we can have an arbitrary subset of these set generators on the left and an arbitrary subset on the right, we have 2^{2N} complete candidate definitions. We want to avoid *explicitly* testing the 2^{2N} complete candidate definitions, so we attempt to make this task more efficient by testing *partial* definitions. For example, we might test the partial definition:

$<\{..., x^L*y, ...\} \mid \{...\}>$

where the "..." indicates that we do not have information concerning those entries. We can gain efficiency if whenever a partial definition is pruned, then all complete definitions that are extensions of the pruned candidate are (implicitly) pruned as well.

In order for this to work, it must be possible to transform the pruning constraint into a constraint which is consistently evaluable. By a *consistently evaluable constraint* we mean one which satisfies the following: (i) the constraint is *evaluable* given only the partial definition of a candidate *conway_mult*, and (ii) the constraint is *consistent* with the original constraint. By *consistent* we mean that if the transformed constraint prunes a partial definition, then the original constraint would prune all complete definitions which extend the pruned partial definition. However, we do *not* require the transformed constraint to be equivalent to the original constraint.

One obvious decomposition for Conway operators is to independently test the left and right sides of a definition[12], which would reduce the number of candidates explicitly tested to 2^{N+1}. A second decomposition is to independently test a single (left or right) set generator, which further reduces the number of candidates explicitly tested to 2N.

This decomposition of the evaluation of generated candidates into subparts is essential for the discovery of Conway multiplication. However, the problem decomposition complicates the testing of the individual parts, as will become clear shortly. Next, we describe the above sequence of two decompositions

---

[12] This decomposition also applies to operators over complex numbers, where left and right sides is analogous to real and imaginary parts (see [Sims & Bresina 89]).

applied to the *conway_mult* operator and the associated transformations of the pruning constraint.

In order to independently test the left and right sides of a *conway_mult* candidate, it must be possible to likewise decompose the homomorphism constraint into a constraint that applies to a candidate left side and another constraint that applies to a candidate right side, such that both new constraints are consistently evaluable.

The part (i.e., conjunct) of the homomorphism constraint (equation (1)) that is affected by testing a partial candidate is Conway equality. For Conway numbers, $A \equiv B$ is defined as ($A \geq B$ and $A \leq B$), and we can expand the definitions of $\geq$ and $\leq$ yielding:

- (no $A^R \leq B$) and ($A \leq$ no B^L) and (no $B^R \leq A$) and ($B \leq$ no A^L).

Note that the homomorphism constraint is of the form $A \equiv B$, where A is $\phi(r_1 r_2)$ and B is $\phi(r_1) * \phi(r_2)$. This expression becomes:

$$(\text{no } \phi(r_1 r_2)^R \leq \phi(r_1)*\phi(r_2)) \text{ and } (\phi(r_1 r_2) \leq \text{no } \phi(r_1)*\phi(r_2)^L) \text{ and}$$
$$(\text{no } \phi(r_1)*\phi(r_2)^R \leq \phi(r_1 r_2)) \text{ and } (\phi(r_1)*\phi(r_2) \leq \text{no } \phi(r_1 r_2)^L) \qquad (2)$$

Table 2: Minimum information necessary to evaluate the subexpressions of the homomorphism constraint.

Evaluable Expression	Needed Information
$(\phi(r_1 r_2) \leq \text{no } \phi(r_1)*\phi(r_2)^L)$	Left set's definition
$(\text{no } \phi(r_1)*\phi(r_2)^R \leq \phi(r_1 r_2))$	Right set's definition
$(\text{no } \phi(r_1 r_2)^R \leq \phi(r_1)*\phi(r_2))$	Complete definition
$(\phi(r_1)*\phi(r_2) \leq \text{no } \phi(r_1 r_2)^L)$	Complete definition

Table 2 shows the needed information to evaluate each of the above four homomorphism conjuncts. As long as r_1 and r_2 are integers (whether or not the candidate is a complete definition of *conway_mult*), then we can compute $\phi(r_1 r_2)$; in the conjunction above, this means A, A^R, and A^L are all evaluable. The *conway_mult* candidate is used to compute $\phi(r_1) * \phi(r_2)$; hence, if the candidate is a partial definition, then some part of B (in the above

conjunction) may not be evaluable. If our partial candidate definition is just the left set generators, then we can only compute B^L – we can not compute B nor B^R; hence, only one of the above four conjuncts is evaluable, (i.e., $A \leq$ no B^L). Similarly, if our partial candidate definition is just the right set generators, then the only evaluable conjunct is (no $B^R \leq A$). Thus, we can decompose the original constraint into the following consistently evaluable constraints:

- *left-constraint:*
  $(\forall\ r1 \in \text{reals})\ (\forall\ r_2 \in \text{reals})\ [\phi(r_1 r_2) \leq \text{no}\ [\phi(r_1) * \phi(r_2)]^L]$
- *right-constraint:*
  $(\forall\ r_1 \in \text{reals})\ (\forall\ r_2 \in \text{reals})\ [\text{no}\ [\phi(r_1) * \phi(r_2)]^R \leq \phi(r_1 r_2)]$ (3)

The left-constraint applies to the set generators for the candidate expression's left set, and similarly the right-constraint applies to those of the right set. Together, these two constraints are less expensive to evaluate than the original constraint; on the other hand, they form a less stringent prune than the original constraint.

Consider the second Conway operator definition decomposition, mentioned earlier, of testing a single generator from the set of left set generators or from the set of right set generators. How must the above left and right constraints be transformed to be consistently evaluable when the candidate is a single set generator? A left set generator's contribution to the product is a subset of the product's left side set. Because of the "no" quantifier in expression (2), if the left constraint derived from (2) is not satisfied for even one left set generator, it will not be satisfied for any left side definition which includes that set generator. Therefore, for this example, no additional transformation is necessary; we can simply generate single set generators and test each one individually as a candidate right set generator or as a candidate left set generator using the above two constraints.

2.2.2 Recursive partial definitions

There is an additional complication, resulting from Conway operators being recursive, that is ignored in the preceding discussion. If the candidate is only a partial definition of *conway_mult*, then recursive calls to *conway_mult*

only return part of the product; hence, the above two constraints are not (in general) evaluable. A solution to this problem is to use mathematical induction. That is, in order to determine whether a single set generator satisfies the pruning constraint on a specific example, IL could perform an inductive proof where the induction is over the age of Conway numbers. The inductive hypothesis is that the homomorphism constraint (instantiated with the specific example) holds on *conway_mult* computations when the arguments are *older*. Recall that the recursive heuristic used in the generate phase insures that the recursive *conway_mult* calls have arguments that are *older* than the arguments to the top level *conway_mult*. Since $\overline{0}$ is the oldest number, the base case is $\overline{0} * \overline{0}$. Since $\overline{0}$ has empty right and left sets, the pruning constraint is vacuously satisfied. So, IL must prove that the top-level *conway_mult* computation satisfies the pruning constraint (on the specific example) under the inductive assumption that the recursive *conway_mult* computations do.

If IL has to perform a formal inductive proof for each pruning example, we would lose much of the efficiency gained by testing single set generators. Therefore, instead of this costly process, the (original) homomorphism constraint is used to create an "ideal" *conway_mult* function, called *ideal_conway_mult*, that, by definition, satisfies the constraint. The recursive calls in the *conway_mult* candidate are replaced with calls to this *ideal_conway_mult*. Using this function to compute the recursive calls to *conway_mult* is valid under the inductive hypothesis.

Creating the *ideal_conway_mult* involves transforming the constraint into a function that, given two Conway numbers, returns the product without using *conway_mult*. This transformation results in (see figure 5): $c_1 \bullet c_2 =_{def} \phi(\phi^{-1}(c_1)\phi^{-1}(c_2))$, where *ideal_conway_mult* is represented by $\bullet$. That is, to compute the product of two Conway numbers c1 and c2, we first map c_1 and c_2 into real numbers, then we multiply those real numbers, and then map that (real) product to a Conway number.

This function obviously satisfies the homomorphism constraint. As we stated previously, IL has only partial knowledge of ϕ; furthermore, IL does

not have a way to directly compute ϕ^{-1}. However, IL saves examples of ϕ; i.e., it caches input-output associations (i.e., <real #, Conway #> pairs) that it has computed. So, *ideal_conway_mult* can only be evaluated on Conway numbers that were previous outputs of ϕ.

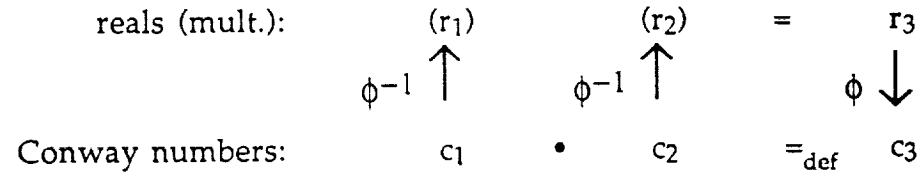


Figure 5: Defining ideal Conway multiplication which we denote as *ideal_conway_mult* or •. Since $\phi(r)$ is also written as $\overline{r}$, this definition can be written as: $\overline{r_1} \bullet \overline{r_2} = \overline{r_1 r_2}$

2.2.3 Implementation of constraint transformation

In this subsection, we briefly describe our implementation of the operationalization of a pruning constraint given a partial candidate and the creation of the ideal operator for evaluation of recursive calls. Recall that the objective of this transformation is a pruning constraint that is consistently evaluable with respect to the given partial candidate definition. The overall process can be summarized as follows. The pruning constraint is transformed using the rewrite rules and IL's definitions until it contains evaluable subexpressions that refer to the partial candidate definition. This process is carried out in a depth-first order. At each step, the current subexpression is tested symbolically for evaluability; if it is not evaluable, then rewrite rules are applied if available, else, the subexpression is expanded and partially compiled if possible.

We now illustrate this process with some example transformation steps from the example of this paper. In this case, the partial candidate definition is a single (right or left) set generator, and the original pruning constraint in IL looks like the following expression. Recall, this is part of the purpose of Conway multiplication. The notation is a list of a predicate name followed by its arguments, with variable names capitalized. Except for minor syntactic

changes to improve the readability, the following expressions use IL's representation.

```
(group_homomorphism
      real_to_conway_map
      [reals real_mult]
      [conways conway_mult])
```

This expression is a predicate that states that *real_to_conway_map* is a homomorphism from the multiplicative group of reals to the multiplicative group of *conways*. Expanding the definition of *group_homomorphism* results in the following.

```
(and
  (get_value conways equality Equal)
  (forall Real1 ∈ reals
    (forall Real2 ∈ reals
     (and
          (real_to_conway_map Real1  Conway1)
          (real_to_conway_map Real2  Conway2)
          (real_mult Real1 Real2 Real_product)
          (real_to_conway_map Real_product Conway3)
          (conway_mult Conway1 Conway2 Conway_product)
          (Equal Conway3 Conway_product)))))
```

Partial compilation is used whenever possible to simplify the expression during the transformation process. Any subexpression whose input arguments are constants and whose relation's definition is executable, such as the above *get_value*, is evaluated. The resulting values assigned to the subexpression's output variables are substituted throughout the rest of the expression. Applied to the above expression, partial compilation results in the following.

```
  (forall Real1 ∈ reals
    (forall Real2 ∈ reals
     (and
          (real_to_conway_map Real1  Conway1)
          (real_to_conway_map Real2  Conway2)
          (real_mult Real1 Real2 Real_product)
```

(*real_to_conway_map* Real_product Conway3)
(*conway_mult* Conway1 Conway2 Conway_product)
(≡ Conway3 Conway_product))))

Since we have only a partial definition for *conway_mult*, the *conway_mult* subexpression (above) is not evaluable and would be expanded (after a few steps) into an expression that described how the sets of left and right generators are used to produce the value of Conway_product. At this point a partial evaluation rewrite rule can be applied.

The partial evaluation rewrite rules expand an expression such that an evaluable subpart can be isolated. One of these rewrite rules regards how a right (left) set is computed from the set of right (left) generators. The rule specifies that such a computation can be rewritten as the union of the set produced by the first generator and the set produced by the rest of the generators. This rewrite allows the application of a generic rule that notes that the set produced by the single candidate generator is a subset of the set produced by any complete definition that includes the candidate.

This conclusion enables application of the following consistency preserving rewrite rule (where C is some constraint):

If $S \subset T$, then can rewrite $\neg\forall x \in T\ C(x)$ as $\neg\forall x \in S\ C(x)$.

This rule is used, for example, to transform the left set constraint (listed as the first entry in table 2 above) to apply to the subset of the left set produced by a single candidate left generator. The objective of the consistency preserving rewrite rules is to transform the constraint so that it is evaluable with the partial information while maintaining that the transformed constraint is consistent with the original constraint.

The transformation process terminates when the partial candidate definition occurs as a subexpression in the overall constraint and the conjuncts of the constraint which refer to this partial definition are evaluable. At this point

the subexpressions of the conjunction that are not evaluable are identified and deleted from the expression.[13]

The symbolic test of evaluability is defined as follows. An expression is considered evaluable if every subexpression can be evaluated in sequence. For a subexpression to be evaluable, its predicate relation must be well-defined within IL and each of its input arguments must either be a constant or be considered bound. An input variable is considered bound if either the variable is quantified or if the variable appears as an output variable in some preceding subexpression that is evaluable.

The result of the transformation process is a constraint that is evaluable on the partial candidate definition and that is consistent with the original constraint. The implementation is general in that it can be applied to any IL constraint expression; however, it does depend on the availability of domain-specific rewrite rules (i.e., partial evaluation rules and consistency rules).

Recall that part of the operationalization of the pruning constraint involved the creation of an ideal operator (e.g., *ideal_conway_mult*) for the evaluation of recursive calls (see figure 5). The implementation of this automatic creation uses similar techniques as the above described transformation process, and depends on domain-specific rewrite rules to transform unevaluable subexpressions to ones that can be evaluated with the partial information available.

2.2.4 Examples used in pruning

By a *valid pruning example* for testing a candidate definition, we mean an input vector for which the pruning constraint is evaluable. For the left and right constraints in (3), the components of the input vector are the variables that are universally quantified; hence, a pruning example consists of values for r_1 and r_2. The partial knowledge in the pruning constraint influences the

---

[13] IL's Prolog-like predicate representation of relations does not usually allow explicit nesting. Hence, this deletion of a subexpression amounts to making a conjunct true, thus generalizing the expression.

examples on which the constraint is evaluable. The partial knowledge about ϕ restricts r_1 and r_2 to be integers which each satisfy the following: (i) IL knows $\phi(r_i) = c_i$, (ii) for each right typical element of c_i, c_i^R, IL knows a real number s_i such that $\phi(s_i) = c_i^R$, and (iii) for each typical left element of c_i, c_i^L, IL knows a real number t_i such that $\phi(t_i) = c_i^L$. These last two properties ensure that *ideal_conway_mult* is evaluable. Examples of evaluating the pruning constraints are presented in the next subsection.

The generation (or selection) of good pruning examples can be expensive and depends on a knowledge-driven analysis of the problem domain. Ideally, the pruning examples would be as independent as possible, so that the sets of candidates that each would reject would have little overlap. There is a trade off between the expense of pruning with a large number of lower quality examples (i.e., not very stringent filters) versus the expense of finding higher quality examples. For example, one (knowledge-rich) heuristic for multiplication would be to use examples in each of the three cases: (i) positive number times positive number, (ii) positive number times negative number, (iii) negative number times negative number. This selection of good pruning examples was not implemented; rather, training examples were randomly selected.

2.2.5 Example of prune evaluation[14]

To summarize the prune phase, over a set of examples (valid values for r_1 and r_2) each candidate set generator that the system generates is empirically tested as a possible left set generator and as a possible right set generator for the definition of *conway_mult* using the two pruning constraints of (3):

- *left-constraint:* $\phi(r_1 r_2) \leq$ no $[\phi(r_1) * \phi(r_2)]^L$
- *right-constraint:* no $[\phi(r_1) * \phi(r_2)]^R \leq \phi(r_1 r_2)$

We now illustrate the pruning process on a particular candidate right set generator, say $x^R * y^R$, over the following two pruning examples: (2, 3) and (-2, -3). Recall, that $x^R * y^R$ being a right set generator means that the full

---

[14] The activities described in this section were fully implemented except as noted.

definition of $x * y$ is of the form: $x * y = \langle\{...\} \mid \{...,x^R * y^R,...\}\rangle$; or in this case: $\phi(r_1) * \phi(r_2) = \langle\{...\} \mid \{...,\phi(r_1)^R * \phi(r_2)^R,...\}\rangle$. That is, $\phi(r_1)^R * \phi(r_2)^R$ is one particular member of the set of right set generators $[\phi(r_1) * \phi(r_2)]^R$. Hence, for this candidate, the right-constraint becomes: $[\text{no } [\phi(r_1)^R * \phi(r_2)^R] \le \phi(r_1 r_2)]$. Since with this candidate, $x * y$ is recursive, the recursive call of $*$ in $x^R * y^R$ is replaced by *ideal_conway_mult*, $\bullet$, yielding: $\text{no } [\phi(r_1)^R \bullet \phi(r_2)^R] \le \phi(r_1 r_2)$, which is equivalent to: $\text{no } [\phi(\phi^{-1}(\phi(r_1)^R)\phi^{-1}(\phi(r_2)^R))] \le \phi(r_1 r_2)$, by the definition of *ideal_conway_mult*.

For a first instance, consider the case of the real multiplication (2)(3) = 6. Our example is the input vector (2,3). For our right set generator, $x^R * y^R$, the right-constraint becomes: $\text{no } [\phi(2)^R \bullet \phi(3)^R] \le \phi(6) = \overline{6}$. The mapping for integers is assumed known and includes the following: $\phi(2) = \overline{2} = \langle\{\overline{1}, \overline{0}\} \mid \{\}\rangle$, $\phi(3) = \overline{3} = \langle\{\overline{2}, \overline{1}, \overline{0}\} \mid \{\}\rangle$, and $\phi(6) = \overline{6} = \langle\{\overline{5}, \overline{4}, \overline{3}, \overline{2}, \overline{1}, \overline{0}\} \mid \{\}\rangle$. So, there are no elements $\phi(2)^R$ nor $\phi(3)^R$. Hence, the right-constraint is satisfied. In other words, $x^R * y^R$ passes our right-constraint on the example (2,3) and is still in the running to be a generator for the right set of multiplication.

For the second example, consider the case of the real multiplication, (-2)(-3) = 6, where our example will be the input vector, (-2,-3). Let us consider the same right set generator as above, $x^R * y^R$. The right-constraint is: $\text{no } [\phi(-2)^R \bullet \phi(-3)^R] \le \phi(6)$, which can be rewritten as: $\text{no } [\overline{-2}^R \bullet \overline{-3}^R] \le \overline{6}$. The assumed integer mapping gives: $\overline{-2} = \langle\{\} \mid \{\overline{-1}, \overline{0}\}\rangle$, and $\overline{-3} = \langle\{\} \mid \{\overline{-2}, \overline{-1}, \overline{0}\}\rangle$. Picking the first elements of $\overline{-2}^R$ and $\overline{-3}^R$ the constraint becomes: $\text{not } (\overline{-1} \bullet \overline{-2} \le \overline{6})$. From the definition of *ideal_conway_mult*, $\bullet$, (see figure 5) we know $\overline{-1} \bullet \overline{-2} = \overline{(-1)(-2)} = \overline{2}$, and so this constraint becomes $\text{not}(\overline{2} \le \overline{6})$. However, it turns out that an application of the definition of Conway inequality, $\le$, to the definitions of $\overline{2}$ and $\overline{6}$ produces the expected result that indeed $\overline{2} \le \overline{6}$. Hence, $\text{not}(\overline{2} \le \overline{6})$ is false and the candidate right set generator, $x^R * y^R$, fails the right-constraint on this example. Since a valid right set generator candidate must satisfy the right-constraint on all examples, we can prune this candidate. IL would then continue to search for other valid candidates.

If all candidate (right or left) set generators are pruned, then IL reinvokes the generate phase to generate the next higher complexity level of candidate set generators. If there remains a small number of set generators, then the processing would pass on to the prove phase of GPP. If there remain too many set generators, then the pruning constraint would be strengthened and the prune phase would continue.[15] The pruning constraint can be strengthened by (i) increasing (or improving) the set of pruning examples, or (ii) using more of the operator's intended purpose to derive the pruning constraints; in this case, this would mean incorporating, in the pruning constraint, some of the other properties (in addition to the homomorphism) in the definition of a field. The decision of how much of the purpose to use in deriving the pruning constraint is another efficiency tradeoff. Ideally, one wants to use those parts of the "purpose" which will produce the strongest pruning constraint per (computational) cost.

2.3 The Prove Phase

In the third phase of GPP, a single complete candidate definition (for *conway_mult*) is formed from all left and right set generators that have not been pruned. IL would then conjecture that *conway_mult* with this definition satisfies *conway_mult*'s specified purpose. In the example of this paper, it would have then conjectured that Conway numbers with *conway_add* and this candidate *conway_mult* form a field. During the process of proving this conjecture, IL may opportunistically prove a number of other properties (e.g., that *conway_mult* and Conway numbers without Conway zero form a group).

IL's theorem prover might prove the conjecture false or it might be unable to prove it either true or false due to limitations of the theorem proving process. Various strategies for proceeding can be appropriate; for example: (i) generating and pruning more candidate *conway_mult* definitions, (ii) trying to conjecture and prove needed lemmas, (iii) changing *conway_add*'s

---

[15] Although we designed this constraint strengthening process, it has not been implemented.

definition, (iv) changing the definition of Conway numbers, etc. Further discussion of the prove phase is contained in [Sims 90].

3. Concluding Remarks

3.1 Historical Context and Related Work

We realized for some time that IL's discovery of the definition for Conway multiplication would be an interesting accomplishment. However, since it was also clearly a challenging problem even for humans, we avoided it until work by Wei-Min Shen [Shen 90] raised the prospect of a reasonable methodology for modeling the discovery. In order to understand why ultimately Shen's approach is inappropriate for our problem, it is necessary to go back and address a similar problem in an earlier system, Douglas Lenat's AM program [Lenat 77].

AM had a large body of heuristics which controlled its reasoning and consequently its discoveries. AM began with a core of knowledge about set theory and those relatively general heuristics, and was able to discover some concepts of real numbers including addition, multiplication, and prime numbers. Among the discoveries made by the AM program were the mathematical operators of addition and multiplication for nonnegative integers. At first appearances, this methodology should be applicable for the problem we are addressing of discovering Conway multiplication. What Shen was able to do was to give a very elegant and parsimonious representation of AM's operator discovery process by describing the process in terms of Backus' functional transformations [Backus 78]. The work of this paper can be seen as follow up work to Shen's.

Shen's process appears very general, and it seemed a natural way to encode operator discovery in IL. Consequently, we considered how we might implement these functional transformations in IL. After a fair amount of consideration, we came to the opinion (although by no means a proof) that to make the described discoveries with functional transformations required a very special representation for the mathematical objects (and this applies to

AM as well).[16] These objects needed to be represented literally as a bag of "t's". Although this representation is somewhat intuitive, it only works for integers – and small integers at that. It is usually unreasonable to represent 1 billion as 1 billion "t's" inside of the computer. So we need alternative representations for numbers. However, the functional transformations used in the Lenat/Shen work do not seem to work on these alternative representations. For example, the definition of the operation of addition in AM is literally just the concatenation of these bags of "t's", corresponding to the numbers added. What this concatenation operator translates into for other representations is not obvious, and the transformations that create these operators used special properties of the bag of "t's" in a strong way (e.g., that they are associative and commutative under concatenation).

We then looked for alternative methods for discovery of Conway multiplication. As has often been the case with our other work with IL, the guidance for how to do this came from the mathematicians. To discovery such an operator, a human mathematician uses a great deal more domain knowledge, for example, the purpose for which the operator is being created.

In related research on the discovery of graph theory properties, by the use of an elegant representation Epstein is able to generate only provably correct properties [Epstein 87]. Thereby she was able to move a powerful prune into the generation process in an effective way.

Kedar [Kedar 88] has investigated the use of purpose in the discovery of definitions of day-to-day objects (such as a cup), and our work can be seen, in part, as an extension of that work to mathematical domains. We search for an operator definition consistent with a specified purpose.

---

[16] Lenat and Brown may have understood this as a result of their work on the representations of AM and Eurisko [Lenat and Brown 84]. They refer in a general way to the importance of representations in AM's discoveries. However, it was not obvious to us that their evaluation suggested such a strong connection between the operator discoveries and the representation of numbers as bag of "t's". Rather Lenat and Brown emphasize the important contribution of their Lisp representation for the discovery of number theory expressions. Shen's functional transformation work is of importance, in part, because it gives us the clarity to better understand AM's discoveries.

Kokar [Kokar 86] has investigated the discovery of concepts consistent with a constraint of invariance (such as under change of units), and Lowry [Lowry 92] has modeled the discovery of special relativity given the requirement of invariance of the equations under a symmetry group. Both of these uses of invariance are analogous to our use of the homomorphism constraint.

Operationalization has been utilized in machine learning [Mostow 87], especially in the context of explanation-based generalization [Keller 87; Hirsh 90]. We have borrowed the notion of operationalizing and the flavor of the process from this work, as well as incorporating techniques of partial evaluation from the programming literature [Bjorner, *et al.* 87]. We have coupled the operationalization techniques together with techniques for handling partial information concerning operators, etc. A somewhat related approach was used by Carbonell and Gil [Carbonell & Gil 90] in the refinement of partially specified Strips operators for grinding telescopes. However, Strips operators and this refinement process differ in nature from those needed in mathematical domains.

Our method of problem decomposition is a specialization of more generic methods that have been used in the planning community (e.g., [Nilsson 71]).

The use of examples to control the size of the spaces searched is well understood in mathematics and has a long history in artificial intelligence as well [Gelernter 63]. Bledsoe gives the perspective *"we cannot over emphasize the importance of being able to calculate properties about a particular [instance] rather than prove the same properties about the uninstantiated variable."* [Bledsoe 83]. Explanation based learning typically uses a single example to guide the operationalization process. In contrast, our constraint operationalization does not use examples. In GPP, examples are used in combination with the operationalized constraint to prune candidate partial definitions.

3.2 Experimental Results

This work is implemented in Prolog. We have implemented the generate and prune phases of GPP, including the automatic transformations of the pruning constraint to apply to a partial candidate and the construction of an ideal function to use for recursive calls to a partial candidate. The prove phase is a simple call to IL's theorem prover, VERIFY. VERIFY is widely used in GPP and other activities of IL for the proofs of simple results. We did not however call VERIFY to prove that the Conway multiplication definition that survived the prune phase, because such a proof is far beyond the limited current capabilities of VERIFY. We have successfully tested the generate and prove phase implementation on the (re)discovery of the Conway and complex multiplication operators, as well as the Conway addition operator.

The results on the *conway_mult* case study were: (i) the process discovered the correct definition in generation level 5, (ii) the number of candidates set generators empirically tested was 3468 (including both right and left set generators), and (iii) the generate and prune phases took on the order of an hour on a dedicated TI Explorer II.

3.3 Summary

GPP was tested on the discovery of the operator definition for Conway multiplication (described herein) as well as the similar and simpler examples of the discovery of Conway addition and complex multiplication [Sims and Bresina 89]. This methodology trivially applies to simpler cases of Conway negation and complex addition and negation.

The purpose constraint is, in general, not operational for application to a given set generator candidate and example. There is only partial information concerning the mapping from the reals to the Conway numbers and we only know part of the definition for Conway multiplication when we evaluate it. Having only partial knowledge available is a major source of complexity. Due to the partial knowledge, the pruning constraint requires significant transformation in order to operationalize it.

A mathematician would probably use less search and perform a deeper analysis in the discovery of Conway multiplication than IL does. However, even given IL's bias toward trading search for knowledge, doing this discovery in a general way required mathematical reasoning that was both surprisingly subtle and tantalizingly close to our modeling capabilities. Under favorable circumstances the methods we have demonstrated may lead to new discoveries, but the techniques are not yet ready for widespread use. We believe that with only minor changes our implementation should be applicable to similarly structured mathematical objects, such as matrices, quaterions, tensors, etc.

However, we believe that for techniques in machine learning, such as those of machine discovery, to be relevant to the work of doing actual mathematics, the issue of concern is not generality but depth. We have presented what we feel is a fairly deep and realistic rendering of a piece of nontrivial mathematics. Our concern is not so much where else it works but rather what important subtleties are we not capturing. In that spirit, we end by pointing out issues which are not adequately handled in this piece of research.

One of the most serious limitation of our overall effort to model this kind of theory formation in mathematics is the effort required for detailed knowledge engineering.

A specific limitation of our implementation is the assumption encoded in the problem decomposition method employed. In the case study, we used a decomposition of the problem of finding a complete operator definition into the independent problems of finding a left side generator and finding a right side generator. Unfortunately, the generators for some operators, such as the Conway square root, interleave recursively between the left and right sets; hence, the described problem decomposition is not appropriate for this case. A more general solution would necessitate treating the problem decomposition process as a search through alternative decompositions; as is done, for example, in the problem reduction system REAPPR [Bresina 88].

The following are aspects of the generate and prune process that may prove to be limitations for the discovery of certain types of operators.

- The search for candidate generators is terminated when a small, nonzero number of generators in a given level of complexity is found.
- The system assumes that all the left set generators and all the right set generators have the same complexity.
- At termination of the generate and prune phases, the proposed definition is comprised of all the surviving left set generators and all the surviving right set generators. Instead, the system could propose definitions using subsets of these generators.

The contribution of this work to mathematics is a glimpse of what the not too distant future may offer in the way of automated research aids. The contributions to AI include the following.

- the in-depth treatment of a difficult mathematical problem utilizing a multitude of AI techniques
- the handling of partial information
  This partial information was a major contributor to the complexity of the system design and at the same time was essential to handling the immense *a priori* search space.
- we introduced purpose-directed generate, prune and prove (GPP) as a method for operator creation and verification

We believe, at this time, that great insights and subsequent progress will come from adequately handling a small number of real problems in depth, rather than following the more common pursuit for generality via the treatment of many simplified cases.

Acknowledgements: Thanks to S. Amarel, P. Friedland, and the many reviewers of previous drafts who provided many helpful suggestions.

References

[Amarel 86] S. Amarel. Program synthesis as a theory formation task - problem representations and solution methods. In R.S. Michalski, J.G. Carbonell and T.M. Mitchell (eds.). *Machine Learning: An Artificial Intelligence Approach - Vol. II.* pp. 499-569. Morgan Kaufmann, San Mateo, CA, 1986.

[Backus 78] J. Backus. Can programming be liberated from the von Neumann style: A functional style and its algebra of programs. *Communications of ACM*, August 1978.

[Barr & Feigenbaum 82] A. Barr and E.A. Feigenbaum. *The Handbook of Artificial Intelligence*, Vol. II, William Kaufmann, Los Altos, CA. 1982.

[Bhaskar & Nigam 90] R. Bhaskar and N. Nigam. Qualitative physics using dimensional analysis. *Artificial Intelligence Journal*, 45, pp. 73-111, 1990.

[Bjorner, *et al.* 87] D. Bjorner, A.P. Ershov, and N.D. Jones (eds.) *Workshop on Partial Evaluation and Mixed Computation.* Lecture Notes in Computer Science, Springer-Verlag, 1987.

[Bledsoe 83] W.W. Bledsoe. Using examples to generate instantiations of set variables. In *Proceedings of IJCAI-83*, pp. 892-901. Karlsruhe, West Germany, 1983.

[Bresina 88] J.L. Bresina. *REAPPR - An expert system shell for planning.* Technical Report LCSR-TR-119, Laboratory for Computer Science Research, Rutgers University, February, 1988.

[Bundy 83] A. Bundy. *The Computer Modelling of Mathematical Reasoning.* Academic Press, Inc., Orlando FL. 1983.

[Carbonell & Gil 90] J.G. Carbonell and Y. Gil. Learning by experimentation: the operator refinement method. In Y. Kodratoff and R.S. Michalski (eds.). *Machine Learning: An Artificial*

Intelligence Approach - Vol. III. pp. 191-213. Morgan Kaufmann, San Mateo, CA, 1990.

[Conway 76] J.H. Conway. *On Numbers and Games*. Academic Press, Orlando FL. 1976.

[Epstein 87] S.L. Epstein. On the discovery of mathematical theorems. In *Proceedings of IJCAI-87*, Milan, pp. 194-197, Morgan Kaufmann, 1987.

[Gonshor 86] H. Gonshor. *An Introduction to the Theory of Surreal Numbers*. Cambridge University Press. NY. 1986.

[Hirsh 90] H. Hirsh. Conditional operationality and explanation-based generalization. In Y. Kodratoff and R.S. Michalski (eds.). *Machine Learning: An Artificial Intelligence Approach - Vol. III.* pp. 383-395. Morgan Kaufmann, San Mateo, CA, 1990.

[Kedar 88] S.T. Kedar-Cabelli. *Formulating Concepts and Analogies According to Purpose*. Ph.D. Thesis, Rutgers University, May 1988.

[Keller 87] R.M. Keller. Defining operationalization for explanation-based learning. In *Proceedings of AAAI-87*, pp. 482-487, Seattle, WA, Morgan Kaufmann, 1987.

[Kokar 86] M.M. Kokar. Determining arguments of invariant functional descriptions, *Machine Learning*, 1, pp. 403-422. 1986.

[Knuth 74] D. E. Knuth. *Surreal Numbers*. Addison-Wesley, 1974.

[Lakatos 76] I. Lakatos. *Proofs and Refutations. - The Logic of Mathematical Discovery*. Worrall and E. Zahar (Eds.). Cambridge University Press, Cambridge, 1976.

[Lenat & Brown 84] D. Lenat and J.S. Brown. Why AM and Eurisko appear to work. *Artificial Intelligence Journal* 23, pp. 269-294, 1984.

[Lenat 77] D. Lenat. Automatic theory formation in mathematics. In *Proceedings of IJCAI-77*, pp. 833-842. Cambridge MA, 1977.

[Lenat 83] D. Lenat. EURISKO: A program that learns new heuristics and domain concepts. The nature of heuristics III: Program

design and results. *Artificial Intelligence Journal* 21(2), March, 1983.

[Lowry 92] M.R. Lowry. Symmetry as bias: Rediscovering special relativity. In *Proceedings of AAAI-92*, San Jose, CA, AAAI Press, pp.56-62, 1992.

[Minton, *et al.* 89] S.N. Minton, J.G. Carbonell, C.A. Knoblock, D.R. Kuokka, O. Etzioni, and Y. Gil. Explanation-based learning: A problem solving perspective. *Artificial Intelligence Journal* 40, pp. 63-118, 1989.

[Mitchell, *et al.* 86] T.M. Mitchell, R.M. Keller, and S.T. Kedar-Cabelli. Explanation-based generalization: a unifying view. *Machine Learning*, 1, pp. 47-80, 1986.

[Mostow 87] J. Mostow. Searching for operational concept descriptions in BAR, MetaLEX, and EBG. In *Proceedings of the International Machine Learning Workshop-87*, University of California at Irvine, Irvine, CA, pp. 376-382, Morgan Kaufmann, June 1987.[

[Nilsson 71] N.J. Nilsson. *Problem-solving Methods in Artificial Intelligence*. McGraw-Hill, New York, 1971.

[Shen 90] W. Shen. Functional transformations in AI discovery systems. *Artificial Intelligence Journal* 41, pp. 257-272, 1990.

[Sims & Bresina 89] M.H. Sims and J.L. Bresina. Discovering mathematical operator definitions. *Proceedings of the Sixth International Workshop on Machine Learning*. pp. 308-313. Cornell University, 1989.

[Sims 90] M.H. Sims. *IL: An Artificial Intelligence Approach to Theory Formation in Mathematics*, Ph.D. Thesis, Rutgers University, May 1990.