Proceedings of 1990 North American Conf. on Logic Programming

Constraint Logic Programming Applied to Hypothetical Reasoning in Chemistry

Jean Jourdan and Raúl E. Valdés-Pérez
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 - USA
{jjourdan,valdes}@cs.cmu.edu

Abstract

We have applied the new constraint-logic-programming (CLP) language PrologIII to the real, complex problem of devising chemical reaction-pathway hypotheses.
The problem statement is organized as a set of informal premises that acceptable hypotheses must fulfill, including two key premises that specify the numbers of reaction steps and chemical species to be allowed in a pathway. Our approach to the problem involves formalizing these diverse premises as boolean, numerical, and other PrologIII constraint types.
The resulting program is a clear improvement in terms of scope and flexibility on its immediate predecessor developed in a procedural language. The program is part of ongoing research, since automation of the chemistry problem has not previously been realized.
Our application has involved in a natural way constraints of both boolean and numerical character; previous reported CLP applications have exploited only one or the other language ability. The current program illustrates how the combination of such constraints can be achieved. Our result also constitutes evidence that the wholly separate processing of boolean and numerical constraints in PrologIII is not an important obstacle to the solution of tasks presenting both aspects.
From our experience we extract some observations on PrologIII and CLP applications programming, with emphasis on factors that increase program efficiency.

1 Introduction

A constraint logic programming (CLP) language [1] integrates logic programming and constraint handling. This integration allows problems having a numerical and/or logical character to be tackled in a declarative way with non-deterministic computation. Thus it extends the scope of logic programming and enables finer control over a logic program.
Several existing CLP languages are CLP(R) [2], CHIP [3], and PrologIII [4, 5, 6]. These new languages are Prolog-like logic programming languages extended by constraint-handling techniques. Aside from the Herbrand domain, which they share, they differ by the remaining domains on which constraints can be expressed. For example, CLP(R) only permits constraints on real terms, PrologIII on linear rational and on boolean terms, and CHIP on linear rational, boolean, and finite-domain terms.

The published applications have not exploited within one program the ability offered by CLP languages to express and combine constraints on several computational domains including boolean and numeric. Many reported applications concern diagnosis, testing, verification, and design of digital circuits. Two CHIP applications treat the problems of generating focused tests to detect faults [7] and of diagnosing faults in digital circuits [8] using boolean constraints. A CLP(R) application to several electrical engineering problems [9, 10], including circuit design, uses only numerical constraints. Other application domains are as follows. Two CHIP papers in scheduling, allocation, and planning [11, 12] have involved numerical and "symbolic" (finite domain) constraints. The financial options-trading O.T.A.S system [13], written in CLP(R), automatically generates and analyzes investment strategies using numerical constraints. None of these applications combines boolean with numerical constraints in a same program.

A contribution of this paper is to demonstrate that the combination of several types of constraint can arise naturally on real problems, and to illustrate the programming involved. Our approach exploits most of the expressible constraint types in PrologIII, and shows how numerical and boolean constraints work together to narrow the search for solutions.

Our application domain is chemistry, which has not yet been explored by CLP. Our PrologIII implementation greatly extends the scope of an earlier Lisp-based implementation, thanks to the flexibility of declarative programming. The increased scope enables the study of chemical reactions which were beyond the scope of the earlier program. Also, the implementation is quite short and transparent, involving about three pages of code. We add that this application is ongoing research; automation of the task carried out by these two programs has not been done before.

The rest of this paper is organized as follows. Section 2 surveys briefly the PrologIII language. Section 3 describes the task and our earlier experiences with a Lisp-based implementation, and poses a practical question which is addressed by the PrologIII re-implementation. Section 4 explains in detail how the problem is handled in PrologIII. Finally, section 5 evaluates our result and discusses some insights gained into CLP application.

2 PrologIII - A CLP Language

A CLP language combines the declarativeness of logic programming with the efficiency of constraint-handling techniques. The unification concept of logic programming is replaced by the concept of constraint-handling, which treats unification as just another constraint.

CLP advances *logic programming* by extending the domains of expression to include subsets of ordinary algebra, as well as propositional logic. CLP also permits better control over a logic program by constraining the possible unifications through explicit constraints. CLP improves on *constraint programming* because it supports the dynamic generation of constraints, e.g., sets of rules can devise appropriate constraints by examining the problem instance.

PrologIII is a CLP language that enables expression of linear numerical constraints, boolean constraints, and a generalization of the inequality relation to include all PrologIII object types.

PrologIII terms are formed using constants, variables, and operators. The constants are those of propositional logic [$1', 0'$], the non-negative integers, and the empty list $\langle\rangle$. There are three types of operator on these terms: boolean operators [$\wedge, \vee, \neg$],

numerical operators [*, /, +, -], several constructive operators [$\langle\rangle$, .], and others. These operators are typed, so only certain operands are legal. An important point is that any variable appearing in a boolean expression is typed **boolean**, so the variable cannot be unified with a non-boolean expression. Two examples of PrologIII terms are $2 + 3/5$, $\langle 1, 2, 3\rangle$, and $(\neg a \vee b)$.

Constraints in PrologIII are formulated using typed relations. Examples of the numerical relations are $<$, $>$, $\geq$, $\leq$. A numerical constraint using these relations is $2 * x < 4 * 7 + 2$. An example of a boolean relation is the logical implication $\Rightarrow$. A boolean constraint using implication is $(a \wedge b) \Rightarrow c$. PrologIII also allows constraints on the composite object **tree** using the equality ($=$) and inequality ($\neq$) relations. Finally, there are unary relations to restrict the type of a variable, for example, $l : 5$ restricts the type of variable l to a list of length 5. One can specify a set of constraints to be satisfied simultaneously. An example of such a set is: $\{x \geq y,\ y > 10,\ a \Rightarrow (c \vee d),\ v = tree(u),\ x \neq 5/3\}$.[1] Within a rule, the predicates and constraints are separated by a comma; if a rule has constraints but not predicates, then $\rightarrow$, appears.

It is important to note some representational limitations. First, PrologIII cannot treat an expression like $x < y$ as boolean. Hence, a boolean variable cannot unify with such an expression. This inability is due to the wholly separate processing of boolean and numerical constraints. A second limitation is that numerical constraints refer to the domain of rational numbers; a variable's domain cannot be restricted to the integers. Forcing a variable to be integral is achieved indirectly, for example by using the system predicate **integer(x)** which assigns an integer value to **x**. Numerical constraints are required to be linear.

A PrologIII program is a finite collection of rules of the form: $Q_0 \rightarrow Q_1 Q_2 \ldots Q_p, S$ where $Q_0 Q_1 Q_2 \ldots Q_p$ are terms as defined above, and S is a set of constraints to be satisfied. Like Prolog, PrologIII can be regarded as a non-deterministic machine. One state of this machine is represented by the triplet $(W, T_0 T_1 T_2 \ldots T_q, S)$ where $T_0 T_1 T_2 \ldots T_q$ is a suite of terms called goals, S is a set of satisfiable constraints, and W is a set of variables. The state change is represented as follows:

$$\begin{aligned} state_{\,n} &: (W,\ T_0\, T_1\ \ldots\ T_q,\ S) \\ chosen\ rule &: Q_0 \rightarrow Q_1\ Q_2\ \ldots\ Q_p, S' \\ state_{\,n+1} &: (W,\ Q_1\ Q_2\ \ldots\ Q_p\ T_1\ T_2\ \ldots\ T_q,\ S \cup S' \cup \{Q_0 = T_0\}) \end{aligned}$$

The PrologIII machine will go from $state_{\,n}$ to $state_{\,n+1}$ if and only if the set of constraints $S \cup S' \cup \{Q_0 = T_0\}$ can be satisfied. Within this state change the PrologIII machine will simplify the set of constraints. When the list of goals is exhausted, PrologIII succeeds, and outputs the set of variables W together with a collection of simplified constraints holding on the variables.

3 The Task

The CLP application described here is part of an effort directed at partially automating the elucidation of chemical-reaction pathways [14, 15] The elucidation problem is the following: Given an *instantiation* of the (two-reagent) reaction schema $A + B \rightsquigarrow T + P_1 + \cdots + P_{k \geq 0}$, ascertain the pathway, i.e., the set of constituent reaction steps, as well as the other species[2] $P_{k+1}, \ldots, P_n$ (possibly none)

[1] In this paper we use standard mathematical notation rather than PrologIII notation for these symbols: $\rightarrow, \Rightarrow, \wedge, \vee, \neg, \neq, \geq$. Some differences between the Edinburgh notation and ours in this paper are: '$\rightarrow$' vs. '$:-$', '$\langle x, y, z, t\rangle$' vs. '$[x, y, z, t]$', '$\langle x\rangle.y$' vs. '$[x|y]$'.

[2] For our purposes, a species is a distinct molecular formula, e.g., CH_2O.

formed during the reaction, but not yet identified. For example, an (unbalanced) overall reaction such as $C_7H_9N + CH_2O \rightsquigarrow C_{17}H_{18}N_2 + H_2O$ may consist of this three-step pathway:

$$\begin{array}{rcl} C_7H_9N + CH_2O & \rightarrow & H_2O + \lfloor C_8H_9N \rfloor \\ \lfloor C_8H_9N \rfloor + \lfloor C_8H_9N \rfloor & \rightarrow & \lfloor C_{16}H_{18}N_2 \rfloor \\ \lfloor C_8H_9N \rfloor + \lfloor C_{16}H_{18}N_2 \rfloor & \rightarrow & C_{17}H_{18}N_2 + C_7H_9N \end{array}$$

in which there appear the two previously unidentified species C_8H_9N and $C_{16}H_{18}N_2$, shown within $\lfloor\ \rfloor$. Supporting such an hypothesis with evidence is partly an experimental process, outside the scope of this paper. Here we examine only the problem of forming initial pathway hypotheses for a particular reaction.

3.1 Task knowledge

Our approach has organized the task of forming pathway hypotheses as a sound deduction from premises. These premises refer to necessary properties at the level of species, steps, pathways, and sets of pathways. We shall discuss these premises in that order; chemical justification of them is given in [14], but is omitted here. (Hereafter, formula shall be synonymous with 'molecular formula.')

Premise referring to individual species.

- A formula consists of non-negative integers, not all zero (such a formula is called legal).

Premises referring to a pathway step.

- For each chemical element, the number of its reactant atoms in any step equals the number of its product atoms in the same step. That is, steps must be (chemically) balanced.
- All reaction steps have either one or two reactant molecules.
- All reaction steps have either one or two product molecules.
- No formula appears both among the reactants and among the products of a single reaction step.

Premises referring to a pathway (set of steps).

- The pathway is an instantiation of the reaction schema $A+B \rightsquigarrow T+(products)$. That is, the reagents A, B are the starting materials, forming the target product T.
- Neither reagent alone suffices to form the target product T.
- Any pathway has present exactly $\mathcal{S}$ distinct species formulas, which must include the starting materials and known products ($\mathcal{S}$ is a fixed integer parameter).
- There are exactly $\mathcal{R}$ reaction steps, all distinct ($\mathcal{R}$ is a fixed integer parameter).
- All pathway species must be formable from the two reagents A, B.

- For any pathway containing variables (i.e., unknown species), there must exist an assignment of legal formulas to the variables such that the steps are balanced.

Premise referring to a set of pathways.

- No two pathways are identical, nor become identical under any of the following transformations: 1. permuting the reactants (or products) within a step, 2. switching the reactants and products within a step, and 3. permuting two steps.

3.2 Experience with a previous implementation

We wrote previously a Common Lisp program to enumerate all distinct, *schematic*, reaction pathways below a certain complexity. These were collected into a catalogue; for example, one entry having six species ($\mathcal{S}=6$) and two reaction steps ($\mathcal{R}=2$) is this:

$$\begin{array}{rcl} A+B & \rightarrow & Y+Z \\ Z & \rightarrow & T+X \end{array}$$

Over the total program scope of $\mathcal{R} \leq 3, \mathcal{S} \leq 8$, there were 15 494 catalogue entries. This catalogue permits forming pathway hypotheses in the following manner. The elucidation task presupposes that the reagents, and at least one product, are known, implying that their formulas are available. Some of a pathway's variables could be instantiated with these formulas, and the remaining unknowns inferred if possible. If the resulting instantiated pathway were legal, then the pathway would be a legitimate hypothesis, later to be confirmed or not with other evidence.
We illustrate this procedure with an instantiation resulting in an *illegal* pathway. Assume again that the (unbalanced) reaction scheme is: $C_7H_9N + CH_2O \rightsquigarrow C_{17}H_{18}N_2 + H_2O$, so two products of the reaction are already known. The variables of the schematic pathway shown next are bound to the reagents and products:

$$\begin{array}{rcl|lcl|lcl} A+B & \rightarrow & X+Y & A & = & C_7H_9N & T & = & C_{17}H_{18}N_2 \\ X+X & \rightarrow & Z & B & = & CH_2O & W & = & H_2O \\ Z+Z & \rightarrow & T+W & & & & & & \end{array}$$

After binding, there remain the three undetermined species X, Y, Z, and three constraint equations for each chemical element, one equation arising from each step. Solving these equations, we obtain the following illegal pathway:

$$\begin{array}{rcl} C_7H_9N + CH_2O & \rightarrow & C_{17/4}H_5N_{1/2}O_{1/4} + C_{15/4}H_6N_{1/2}O_{3/4} \\ C_{17/4}H_5N_{1/2}O_{1/4} + C_{17/4}H_5N_{1/2}O_{1/4} & \rightarrow & C_{17/2}H_{10}NO_{1/2} \\ C_{17/2}H_{10}NO_{1/2} + C_{17/2}H_{10}NO_{1/2} & \rightarrow & C_{17}H_{18}N_2 + H_2O \end{array}$$

The pathway conjectures illegal formulas having a fractional number of elemental atoms.[3]
Experience with the above procedure soon showed that on many reaction schemes, the number of *simplest* pathways accounting for the transformation of reagents into products were quite few. For example, on the current reaction scheme, no legal hypotheses having fewer than six species and three reaction steps (6s3r) were found. In the class 6s3r, only five pathways were output, of the 6 773 stored. It became clear that the formulas of reagents/products, together with elemental conservation (i.e., that the total number of atoms of each element is constant), constituted an extremely powerful constraint.

[3] "Multiplying through" by 4 results in eight reactant molecules for the first step, exceeding the ceiling of two molecules stipulated by a task premise.

At that point, the goal became to design an algorithm to exploit the formulas and elemental conservation *earlier in the generation phase*. Doing so would expand the scope of the reactions handled, which because of the catalogue approach seemed limited practically to three reaction steps, even with dedicated use of an engineering workstation. The reason was this: even though the catalogue had only 15 494 entries, many more were constructed, then rejected, in the course of generating the catalogue, and the number rejected would explode combinatorially if the catalogue scope were increased.
After moderate effort, no satisfactory algorithm was found. Finally, we perceived that constraint logic programming was an advantageous approach to the problem, in brief: How to express the constraints and apply them as early as possible in the generation of legal, pathway hypotheses?

4 A PrologIII Solution

This section describes our PrologIII answer to the question of how to express the task premises of section 3.1 and apply them to achieve an efficient search for reaction-pathway hypotheses.
A pathway-generation problem consists of the number $\mathcal{S}$ (of pathway species), the number $\mathcal{R}$ (of pathway steps), and a list of species of length $\mathcal{S}$, some of which are instantiated.[4] The remaining task premises do not vary from one problem to the next. $\mathcal{S}$ and $\mathcal{R}$ shall turn out to be pervasive parameters in the sense that their magnitudes determine the sizes of various program data structures.
The core idea underlying the program is to build a pathway template from the parameters $\mathcal{S}, \mathcal{R}$. To build a template means to generate a pathway data structure containing only variables, while attaching sets of constraints on those variables. The template is built deterministically, in order not to permit any backtracking on constraint generation.
Then the values of certain pathway variables are enumerated non-deterministically to find all consistent instantiated pathways, i.e., pathways in terms of formulas. All backtracking is confined to this enumeration stage.
First we describe in subsection 4.1 how the domain objects (species, formula, step, and pathway) are represented. Then subsection 4.2 shows how the pathway template is built and constrained. Finally, subsection 4.3 discusses how the pathways are enumerated.

4.1 Representation of domain objects

A formula is a vector of integers, one integer for each chemical element, e.g., $C_{13}H_{18}NI$ is represented by $\langle 13, 18, 1, 1\rangle$. An unknown formula is a vector of numeric variables.
A species is a quintuplet $\langle n, x, a, b, f\rangle$. The following gives some notion of their types and meaning; we shall adhere to this naming convention below.

- **n** An identifier between 1 (or 0) and $\mathcal{S}$.
- **x** A numeric variable.
- **a,b** Two boolean variables.
- **f** The species formula.

[4] The number of instantiated species (reagents and known products) need not equal $\mathcal{S}$, since not all the species formulas need be known beforehand.

A step is a list of integer species-identifiers; an unknown step is a list of unknown identifiers. The main purpose of the identifiers is to serve as convenient species place-holders; however, their arithmetic properties are used to express the constraint derived from Proposition 1 below. Each identifier corresponds to a distinct species, hence also to a distinct formula. According to the premises of section 3.1, a step has at most two reactant and two product molecules, so we represent the general step $R_1 + R_2 \rightarrow P_1 + P_2$ by the identifier quadruplet $\langle r_1, r_2, p_1, p_2 \rangle$. A pathway is a list of steps of length $\mathcal{R}$.
The top-level goal is the following PrologIII rule.

$$\begin{aligned} &Pathway(\mathcal{R}, \mathcal{S}, p, q) \rightarrow \\ &\quad All_formulas_are_distinct(p) \\ &\quad Pathway_template(\mathcal{S}, \mathcal{R}, q, v, u) \\ &\quad All_distinct_steps(q) \\ &\quad Need_both_reagents(p) \\ &\quad Enumerate_the_species(v, p, u); \end{aligned}$$

The rule's first four subordinate predicates build the pathway template while constraining its variables. The last predicate **Enumerate_the_species** performs the enumeration. The four parameters to **Pathway** are these:

$\mathcal{R}$	The number of reaction steps.
$\mathcal{S}$	The number of species.
p	A list (of length $\mathcal{S}$) of species, including the initially unknown species.
q	A list (of length $\mathcal{R}$) of steps.

An $\mathcal{S}$=4 example of **p** on the scheme $O + H_2 \rightsquigarrow OH$ is:

$$\langle\langle 1, x1, a1, b1, \langle 1, 0\rangle\rangle, \langle 2, x2, a2, b2, \langle 0, 2\rangle\rangle, \langle 3, x3, a3, b3, \langle 1, 1\rangle\rangle, \langle 4, x4, a4, b4, f4\rangle\rangle$$

An $\mathcal{R}$=3 example of **q** is $\langle\langle r_1, r_2, p_1, p_2\rangle, \langle r_3, r_4, p_3, p_4\rangle, \langle r_5, r_6, p_5, p_6\rangle\rangle$.

4.2 Creating and constraining the pathway template

The description below of how task premises are expressed as constraints follows this pattern: one or more premises (or parts of a premise) from section 3.1 appear, preceded by a ♣. Then appear PrologIII fragments expressing the premises, justified more formally if the correctness of the fragment seems unclear.

4.2.1 Constraints on formulas

♣ *Any pathway has present exactly $\mathcal{S}$ distinct species (formulas) which must include the starting materials and known products.*

The first predicate in the *Pathway* rule above serves as our first example of a constraint expression within a PrologIII rule. The recursive rules below require all $\mathcal{S}$ species formulas to be distinct ($f1 \neq f2$). (The other aspects of the premise are expressed later, not here).
If all $\mathcal{S}$ formulas are input to the program, then the rules have little effect. The purpose is really to ensure that when $\mathcal{S}$ is greater than the number of known formulas,

the newly conjectured formulas are unique.

$All_formulas_are_distinct(\langle\rangle) \rightarrow;$
$All_formulas_are_distinct(\langle e \rangle.l) \rightarrow$
$\quad Pairwise_distinct_formulas(e, l)$
$\quad All_formulas_are_distinct(l);$

$Pairwise_distinct_formulas(e, \langle\rangle) \rightarrow;$
$Pairwise_distinct_formulas(\langle n1, x1, a1, b1, f1\rangle, \langle\langle n2, x2, a2, b2, f2\rangle\rangle.l) \rightarrow$
$\quad Pairwise_distinct_formulas(\langle n1, x1, a1, b1, f1\rangle, l),$
$\quad \{f1 \neq f2\};$

These simple rules illustrate the ability of the language to pose constraints using recursion.

4.2.2 Constraints on species identifiers

♣ *Any pathway has present exactly $\mathcal{S}$ distinct species formulas.*

This pathway premise can be re-directed as a constraint on species identifiers, since each identifier corresponds to a unique formula. Actually, the *exactly* $\mathcal{S}$ is not captured; the following constraint expresses instead *at most* $\mathcal{S}$.

$$r_1, r_2, p_1, p_2 \leq \mathcal{S}$$

♣ *All reaction steps have either one or two reactant molecules. All reaction steps have either one or two product molecules.*

These premises are taken into account by our representation of a general step as the identifier quadruplet $\langle r_1, r_2, p_1, p_2 \rangle$. The species identifier '0' represents the absence of a species, e.g., if $r_1 = 0$, then the step has only one reactant molecule.

$$r_1, p_1 \geq 0, \quad r_2, p_2 \geq 1$$

r_2 and p_2 are non-zero, so every step has at least one reactant and product.

♣ *No reaction step involves a same molecular formula among both its reactants and products.*

This is easily rendered as a numerical constraint at the reaction step level, in terms of species identifiers:

$$r_1 \neq p_1, \quad r_1 \neq p_2, \quad r_2 \neq p_1, \quad r_2 \neq p_2$$

We emphasize that these inequalities are over the domain of species identifiers, not of formulas, although the effect is the same, since the formulas differ if the identifiers do. This constraint prohibits steps of form $R \rightarrow P$.
The above constraints wholly capture the effect of the premise. However, the goal of *highly constrained* enumeration is served by including another constraint *entailed by* the premise. For example, a step of form $R + R \rightarrow P + P$ must result eventually in a violation of the premise, because the formula of R must equal that of P for the step to balance. This form is prohibited by making use of a special-purpose proposition:

Proposition 1 *A step is of the form $R + R \rightarrow P + P$ if and only if $(r_1 + r_2) + (p_1 + p_2) * 1000 \;=\; 2r_2 + 2p_2 * 1000$ and $\mathcal{S} < 1000$.*

Proof. If the step is $R+R \rightarrow P+P$, then in our representation we have $r_1 = r_2$ and $p_1 = p_2$, and the equation follows immediately. Conversely, if we assume the equation and that $\mathcal{S} < 1000$ (in practice, a safe assumption), then we see that $r_1 + r_2 = 2r_2$ and $p_1 + p_2 = 2p_2$. Therefore, $r_1 = r_2$ and $p_1 = p_2$, and the proof is complete.

♣ *No two pathways are identical, nor become identical under any of the following transformations: 1. permuting the reactants (or products) within a step.*

The following numerical constraint enforces a unique ordering on the reactants and products via the species identifiers.

$$r_1 \leq r_2, \quad p_1 \leq p_2$$

This ordering prevents any permutation of reactants or products.
We can collect all the identifier constraints so far in a PrologIII rule:

$$
\begin{array}{l}
Step_template(\langle r_1, r_2, p_1, p_2\rangle, \mathcal{S}) \rightarrow, \\
\quad \{ \\
\quad 0 \leq r_1 \leq \mathcal{S}, \\
\quad 1 \leq r_2 \leq \mathcal{S}, \\
\quad 0 \leq p_1 \leq \mathcal{S}, \\
\quad 1 \leq p_2 \leq \mathcal{S}, \\
\quad (r_1 + r_2) + (p_1 + p_2) * 1000 \neq 2r_2 + 2p_2 * 1000, \ /* \ prohibit\ R+R \rightarrow P+P \ */ \\
\quad r_1 \neq p_1, \ r_1 \neq p_2, \\
\quad r_2 \neq p_1, \ r_2 \neq p_2, \ \ /* \ reactants\ and\ products\ are\ distinct \ */ \\
\quad r_1 \leq r_2, \ p_1 \leq p_2 \ \ /* \ ordering\ on\ the\ identifiers \ */ \\
\quad \};
\end{array}
$$

4.2.3 Constraints on steps

♣ *For each chemical element, the number of its reactant atoms in any step equals the number of its product atoms in the same step. That is, steps must be (chemically) balanced.*

The premise that the number of elemental atoms must balance is handled easily. Each chemical element gives rise to a separate balance constraint, expressed by the following PrologIII rule:

$$
\begin{array}{l}
Balanced_step(\langle\langle\rangle, \langle\rangle, \langle\rangle, \langle\rangle\rangle) \rightarrow; \\
Balanced_step(\langle\langle a\rangle.f_r1, \langle b\rangle.f_r2, \langle c\rangle.f_p1, \langle d\rangle.f_p2\rangle) \rightarrow \\
\quad freeze(a, integer(a)) \\
\quad freeze(b, integer(b)) \\
\quad freeze(c, integer(c)) \\
\quad freeze(d, integer(d)) \\
\quad Balanced_step(\langle f_r1, f_r2, f_p1, f_p2\rangle), \\
\quad \{a + b = c + d, a \geq 0, b \geq 0, c \geq 0, d \geq 0\};
\end{array}
$$

The restriction to integral formulas is accomplished using the well-known **freeze** primitive generally found in versions of Prolog.

4.2.4 Constraints on pathways

♣ *For any pathway containing unknown species (i.e., variables), there must exist an assignment of legal formulas to the unknowns such that the steps are balanced.*

The existence of an assignment is testable as shown by the following proposition.

Proposition 2 *A pathway containing variables is consistent if and only if there exists a satisfying positive-integer assignment to the variables of the corresponding numerical equations.*[5]

The numerical equations corresponding to a pathway are those obtained by changing the domain to that of the integers, and replacing the '$\rightarrow$' symbols by '=.' So, each pathway will have PrologIII constraints of numerical equations, one equation for each $\langle r1, r2, p1, p2\rangle$ step.

$$Attached_equation_system(\langle x_r1, x_r2, x_p1, x_p2\rangle) \rightarrow,$$
$$\{x_r1 + x_r2 = x_p1 + x_p2\};$$

The rule's parameter is not a step, rather a list of numeric variables attached to the species of a step in the quintuplet representation, reached via the identifier.
The numerical-constraint processing algorithm will test whether these constraints are soluble in the rationals. If these equations have a rational solution, then they have also an integer solution, since they are homogeneous. Positive solutions are enforced by adding the constraints $x_i \geq 1$.[6]

♣ *There are exactly $\mathcal{R}$ reaction steps, all distinct.*

For our purpose, two steps are distinct if their species formulas are both: 1. not identical, symbol for symbol, and 2. not identical after switching the roles of reactant/product in the step. For example, the steps $C_7H_9N + CH_2O \rightarrow H_2O + C_8H_9N$ and $H_2O + C_8H_9N \rightarrow C_7H_9N + CH_2O$ are not distinct because they fail the second requirement. The requirements are expressed on the identifiers domain by:

$$(\forall\, step_i, step_j \in pathway)\ (step_i \neq step_j) \wedge (step_i \neq switch_roles(step_j))$$

rendered thus in PrologIII:

$$All_distinct_steps(\langle\rangle) \rightarrow;$$
$$All_distinct_steps(\langle\langle r1, r2, p1, p2\rangle\rangle.l) \rightarrow$$
$$\quad Pairwise_distinct_steps(\langle r1, r2, p1, p2\rangle, l)$$
$$\quad All_distinct_steps(l);$$

$$Pairwise_distinct_steps(e, \langle\rangle) \rightarrow;$$
$$Pairwise_distinct_steps(\langle r1, r2, p1, p2\rangle, \langle\langle r3, r4, p3, p4\rangle\rangle.l) \rightarrow$$
$$\quad Pairwise_distinct_steps(\langle r1, r2, p1, p2\rangle, l),$$
$$\quad \{\langle r1, r2, p1, p2\rangle \neq \langle r3, r4, p3, p4\rangle,\ \langle r1, r2, p1, p2\rangle \neq \langle p3, p4, r3, r4\rangle\};$$

4.2.5 Path-oriented constraints on pathways

This section treats premises that refer to paths from reagents to various species; certain paths must or must not exist in an acceptable pathway. This section provides our first use of *boolean constraints*; all previous constraints have been numerical.

♣ *Neither reagent alone suffices to form the target product T.*

A key to formulating the premise is perceiving the analogy between a set of chemical reactions and a propositional proof. The reaction step $R_1 + R_2 \rightarrow P_1 + P_2$ corresponds to the logical implication $r_1 \wedge r_2 \Rightarrow p_1 \wedge p_2$. Here is a full pathway example:

$A + B$	$\rightarrow$	$X + Y$	$a \wedge b$	$\Rightarrow$	$x \wedge y$
$X + X$	$\rightarrow$	Z	x	$\Rightarrow$	z
$Z + Z$	$\rightarrow$	$T + W$	z	$\Rightarrow$	$t \wedge w$

[5] A proof of the result is in [14].

[6] These constraints are actually placed in the predicate **belong_to**, described in section 4.3, because here a null species is possible, and its attached x would need to equal zero.

The clause-form implications associated with a pathway will be referred to as $\mathcal{C}$. We proceed to re-express the above premise as a statement about propositional proofs.

Proposition 3 *In a pathway, species X_1 by itself does not form species X_2 if and only if the corresponding set of clauses $\mathcal{C}$ does not derive $x_1 \Rightarrow x_2$. That is, if and only if $\mathcal{C} \nvdash \{\neg x_1 \vee x_2\}$.*

We combine this statement with the refutation theorem $\mathcal{C} \vdash f \Leftrightarrow \mathcal{C} \cup \{\neg f\} \vdash \emptyset$ to express the desired conditions. A pathway satisfies our formability premise if and only if both these statements are true:
1. $\mathcal{C} \cup \{\neg(\neg reagent_1 \vee target)\} \nvdash \emptyset$ (or, $\mathcal{C} \cup \{reagent_1\} \cup \{\neg target\} \nvdash \emptyset$)
2. $\mathcal{C} \cup \{\neg(\neg reagent_2 \vee target)\} \nvdash \emptyset$ (or, $\mathcal{C} \cup \{reagent_2\} \cup \{\neg target\} \nvdash \emptyset$)

Since the two sets of clauses are not independent, they must be implemented separately, otherwise the theorems of one set can - incorrectly - be used by the other set. Therefore, we shall use two independent sets of clauses: a and b sets; each receives its own version of $\mathcal{C}$, $\neg target$, and $reagent_1/reagent_2$.
The following rule attaches independent boolean constraints from a single pathway step to the a and b clause-sets.

$$Attached_clause_system(\langle a_r1,\ a_r2,\ a_p1,\ a_p2\rangle,\ \langle b_r1,\ b_r2,\ b_p1,\ b_p2\rangle) \rightarrow,$$
$$\{a_r1 \wedge a_r2 \Rightarrow a_p1,\quad a_r1 \wedge a_r2 \Rightarrow a_p2,$$
$$b_r1 \wedge b_r2 \Rightarrow b_p1,\quad b_r1 \wedge b_r2 \Rightarrow b_p2\};$$

The expression of the conditions is completed by adding to the a-set its copies $\{a_1\}, \{\neg a_3\}$ of the clauses $\{reagent_1\}$, $\{\neg target\}$, and to the b-set its copies $\{b_1\}$, $\{\neg b_3\}$ of the clauses $\{reagent_2\}$, $\{\neg target\}$, via this rule:

$$Need_both_reagents(\langle\langle 1, x1, a1, b1, f1\rangle.\langle 2, x2, a2, b2, f2\rangle.\langle 3, x3, a3, b3, f3\rangle\rangle.l) \rightarrow,$$
$$\{a1 = 1',\ a3 = 0',\ b2 = 1',\ b3 = 0'\};$$

$\langle 1, x1, a1, b1, f1\rangle$ represents the first reagent, $\langle 2, x2, a2, b2, f2\rangle$ the second reagent, and $\langle 3, x3, a3, b3, f3\rangle$ the target product; by convention they head the list of species. The third and fourth entries of a quintuplet serve to attach boolean clauses to the species, to enforce respectively that the first and second reagents alone not form the target product.
If instead we considered three-reagent reactions, we would need six sets of independent clauses instead of two to express the following conditions:

$$(\forall\, i, j, k \in (1,2,3))\ \ reagent_i \not\Rightarrow target,\ reagent_j \wedge reagent_k \not\Rightarrow target$$

♣ *All pathway species must be formable from the two reagents A, B.*

The intent of this second path-oriented premise is to exclude spurious pathways such as $A + B \rightarrow T$, $Y \rightarrow X$. We take account of this premise *as a by-product of* a quite powerful constraint on pathway generation, which is motivated next. Informally, a next pathway step is "executable" only if its reactants are available, either as starting materials or as products of "previous" steps. For example, at a reaction's start, only steps involving the two reagents can react. This availability criterion induces a partial order on the steps of a pathway. We shall restrict pathway generation to those pathways that are a legitimate linear path through the partial order. This restriction, accomplished with the **freeze** primitive, turns out to have a powerful beneficial impact on total generation time. A binding to the reactants is not propagated until it is verified that the binding is available from the pool

of species created by the previous reaction steps. The constraint is expressed by modifying the *step_template* rule of section 4.2.2 thus:

$$Step_template(\langle r_1, r_2, p_1, p_2\rangle, \mathcal{S}, u) \rightarrow$$
$$available(r_1, u)$$
$$available(r_2, u),$$
$$\{[constraints \ldots]\};$$

$$available(e, l) \rightarrow freeze(e, member(e, l));$$

where u is the list of available species identifiers, which is initialized to $\langle 0, 1, 2\rangle$, meaning that the two reagents and the null species are the only ones initially available.
All of the constraints described above are used in the two rules below, which build the pathway template via recursion on the list of step templates. All the numerical and boolean systems that constrain the pathway template are formulated by these rules.

$$Pathway_template(\mathcal{S}, 0, \langle\rangle, \langle\rangle, u) \rightarrow;$$

$$Pathway_template(\mathcal{S}, r, \langle\langle r1, r2, p1, p2\rangle\rangle.l, \langle\langle r1, x1, a1, b1, f1\rangle, \langle r2, x2, a2, b2, f2\rangle,$$
$$\langle p1, x3, a3, b3, f3\rangle, \langle p2, x4, a4, b4, f4\rangle\rangle.v, u) \rightarrow$$
$$Step_template(\langle r1, r2, p1, p2\rangle, \mathcal{S}, u)$$
$$Attached_equation_system(\langle x1, x2, x3, x4\rangle)$$
$$Balanced_step(\langle f1, f2, f3, f4\rangle)$$
$$Attached_clause_system(\langle a1, a2, a3, a4\rangle, \langle b1, b2, b3, b4\rangle)$$
$$Pathway_template(\mathcal{S}, r-1, l, v, u);$$

4.2.6 Unexpressed premises

♣ *Any pathway has present exactly $\mathcal{S}$ distinct species formulas.*

By itself, this premise makes a statement about acceptable *completed* pathways; hence, it has little potential to reject partially generated pathways. We were content to express the premise as a PrologIII verification, rather than constraint.

♣ *No two pathways are identical, nor become identical under any of the following transformations: 3. permuting two steps.*

One attempt to express the premise involves defining an order on the steps. We formulated such constraints by placing an order on steps, but this order interfered with the partial order of section 4.2.5, so we did not express the premise.

4.3 Species enumeration

The rules of the previous sections created a pathway template of a fixed number of steps $\mathcal{R}$ having constraints on the pathway species and their attached variables (numeric, boolean etc.). At this point the template contains no constants. To obtain our goal of a pathway in terms of known formulas, we must instantiate some of the template variables while taking into account the $\mathcal{S}$ available species, some of whose formulas are already known. The obvious course is to choose a set of variables and enumerate all of their legal values, in order to find all consistent instantiated pathways.
Enumeration is better understood with reference to two separate lists: 1. the assignable species, i.e., the list of $\mathcal{S}$ species, some of which come initially instantiated (e.g. see the input parameter to rule **Pathway** in section 4.1), and 2. the pathway

species, i.e., the list (of length $4\mathcal{R}$) of species quintuplets in the pathway template, four quintuplets per step.
We recall that each *assignable* species has as unique identifier an integer from 1 to $\mathcal{S}$. Then, enumeration proceeds by assigning an *assignable* species to a *pathway* species via the identifiers, thus unifying their attached constraint variables in the quintuplet representation. Unification is done by the predicate **belong_to**, which is like a typical **member** predicate. When all the pathway species have been unified with the identifiers, and no constraint is violated, then this good pathway is printed, and the remaining solutions are found by continuing the enumeration using backtracking.
Pathway-species enumeration uses the predefined PrologIII predicate **integer(n)**, which serves to enumerate all integer assignments to **n** that satisfy accompanying constraints. Backtracking is done at this predicate; each backtrack on **integer(n)** yields another value for **n**, to be checked as always against the constraints. It is important to note that the program is deterministic up to when assignments are made to pathway species via the **integer(n)** predicate, within the predicate **Instantiate_species** below. That is, the program was designed so that the entire process of placing constraints on the pathway template involves no choice points, therefore no backtracking to place different constraints is possible. This localized backtracking proved an important contributor to efficiency, given our experience with alternate designs that did involve backtracking to template generation.
A general rule is that enumeration is more efficient when the variable of enumeration is more constrained. The pathway species are considerably constrained, as shown in previous sections. Hence, this aspect also contributes to overall efficiency.
The enumeration process is summarized by the following two PrologIII rules, where **l** is the list of $\mathcal{S}$ *assignable* species, **v** is the list of *pathway* species, and **u** is a list of available identifiers, i.e., reagents or previously formed products.

$$
\begin{aligned}
&Enumerate_the_species(v, p, \langle 0, 1, 2\rangle.u) \rightarrow \\
&\quad Instantiate_species(v, p, u);
\end{aligned}
$$

$$
\begin{aligned}
&Instantiate_species(\langle\rangle, p, k) \rightarrow; \\
&Instantiate_species(\langle\langle n, x, a, b, f\rangle\rangle.v, p, u) \rightarrow \\
&\quad integer(n) \\
&\quad belong_to(\langle n, x, a, b, f\rangle, p) \\
&\quad eq(u, \langle n\rangle.k) \\
&\quad Instantiate_species(v, p, k);
\end{aligned}
$$

To better understand the role of the constraints during enumeration, we should look at the predicate **belong_to**. At the start of enumeration, the only active constraints are those on the variables being enumerated; these constraints are explicitly taken into account by the predicate **integer(n)**. The remaining constraints become active only when a pathway species is unified with an assignable species.
The binding between the pathway species and the species identifiers is achieved by the predicate **belong_to**.

$$
\begin{aligned}
&belong_to(\langle 0, 0, 1', 1', f\rangle, l) \rightarrow \\
&\quad coefficients_all_zero(f);
\end{aligned}
$$

$$
\begin{aligned}
&belong_to(\langle n, x, a, b, f\rangle, \langle\langle n, x, a, b, f\rangle\rangle.l) \rightarrow, \\
&\quad \{n \neq 0,\ x \geq 1\}; \\
&belong_to(\langle n, x, a, b, f\rangle, \langle\langle n', x', a', b', f'\rangle\rangle.l) \rightarrow \\
&\quad belong_to(\langle n, x, a, b, f\rangle, l), \\
&\quad \{n \neq 0, n \neq n'\};
\end{aligned}
$$

The first rule below is used on species having the zero identifier, i.e., null species; their attached variables are bound to appropriate dummy values.

We shall illustrate how constraints become active with a two-step pathway template. Four independent sets of pathway constraints are shown below (these are only a subset of the actual constraints, omitting for example the **Need_both_reagents** constraints, to keep the example small).

species identifiers

n_r11 + n_r12	→	n_p11 + n_p12
n_r21 + n_r22	→	n_p21 + n_p22

numerical equations

x_r11 + x_r12	=	x_p11 + x_p12
x_r21 + x_r22	=	x_p21 + x_p22

set of a-clauses

a_r11 ∧ a_r12	⇒	a_p11
a_r11 ∧ a_r12	⇒	a_p12
a_r21 ∧ a_r22	⇒	a_p21
a_r21 ∧ a_r22	⇒	a_p22

elemental balance

f_r11 + f_r12	=	f_p11 + f_p12
f_r21 + f_r22	=	f_p21 + f_p22

These constraints hold over the following domains: **n** integers (species identifiers); **a** boolean values; **x** rationals; **f** vector of integers (formulas).

Suppose that an enumeration step unifies n_r11 with 1, i.e., n_r11 is bound to the first-reagent identifier. The only immediate affect on the constraints is to unify f_r11 with the formula of the first reagent. Suppose the next step also unifies n_r12 with 1. Now the affect on the constraints is broader: we have the unification $a_r11 = a_r12$, which simplifies the set of boolean constraints; we have $x_r11 = x_r12$, transforming the first numerical equation into $2 * x_r11 = x_p11 + x_p12$, and similarly for the elemental balance equations. As illustrated, the constraints become stronger as enumeration proceeds.

Occasionally during enumeration the constraints become inconsistent. Proceeding with the example, suppose the next enumeration steps unify n_p11 with 0 (the null species) and n_p12 with 3 (the target product). The first a-clause becomes $\{a_1 \Rightarrow a_3\}$ where a_1 is a boolean variable attached to the first reagent and a_3 is attached to the target product. The conjunction of this clause with constraints introduced by **Need_both_reagents** is inconsistent: $\{a_1 \Rightarrow a_3,\ a_1 = 1',\ a_3 = 0'\}$.

After each unification, PrologIII checks that the resulting constraints sets are consistent; if not, backtracking occurs, resulting in unifying a pathway species with a different species identifier.

5 Evaluation

Our purpose in this paper has been to demonstrate the practical power of PrologIII, and to illustrate its programming with special reference to combining constraints over different computational domains. The next section extracts some general observations from this experience. First we will justify the assertion that our program is a practical success.

The previous Lisp-based program, called STOICH, is limited to reactions of at most three steps ($\mathcal{R} \leq 3$), for the reasons given in section 3.2. In contrast, increasing $\mathcal{R}$ in our PrologIII program (let's call it STOICH/CLP) from $\mathcal{R}_1$ to $\mathcal{R}_2$ involves a trivial change within the initial goal from a list $\langle w_1, w_2, \ldots, w_{\mathcal{R}_1} \rangle$ to a list $\langle w_1, w_2, \ldots, w_{\mathcal{R}_2} \rangle$. Increasing the number of pathway species $\mathcal{S}$ is also only a trivial change to the sizes of lists in the initial goal.

A second limitation of STOICH is its pervasive assumption of a two-reagent reaction. Changing this assumption so it can handle one-reagent or three-reagent reactions could involve weeks of work, and hasn't been attempted. In contrast, STOICH/CLP

was designed with the same assumption, but has been changed to handle a three-reagent reaction currently under study; this effort took only one hour.
The correctness of STOICH/CLP has been confirmed by comparing its output with that of STOICH. Finally, the speeds of the two programs are roughly comparable within the $\mathcal{R} \leq 3$ scope of STOICH, although STOICH/CLP fares better. Outside that scope, there is no other program to compare STOICH/CLP against, since we are the first to automate this task.
The execution time of STOICH/CLP is found to be modest when the number of unknown species is 0 or 1. For constant $\mathcal{S}$, increasing the number of steps does not appreciably change the time. However, the Macintosh II execution time of STOICH/CLP increases (from $\approx$ 10 min to $\approx$ 10 hours) when the number of initially unknown species (equal to $\mathcal{S}$ minus the number of known formulas) goes from one to two.[7] The reason seems to be that many promising partial pathways have to be considered, i.e., there are fewer early search cutoffs. It is unclear yet whether the same jump in execution time will be observed when going from two to three unknown species.

5.1 Observations on use of PrologIII

PrologIII, as a logic programming language, has shown that its declarativeness contributes to flexibility, for the reasons of the previous section. Also, only about twenty PrologIII rules were needed to express the task premises efficiently. As also seen above, the PrologIII implementation improved the scope and speed of our approach to hypothetical reasoning in reaction chemistry. We have proved the ability of PrologIII to handle naturally a complex, non-academic, research problem.
We can point out some general ideas about programming in PrologIII, and CLP in general. First, there exists a programming tradeoff illustrated by the following two extremes. In one extreme, all task knowledge is formulated as *verifications*, causing inefficiency because no early search pruning is possible. In the other extreme, all knowledge is formulated as *constraints*, causing inefficiency because the resulting large constraint expressions are handled poorly by exponential algorithms. Hence, there is a tradeoff between these two factors, although the ideas of the next paragraph can shift the trade in favor of constraints.
The execution time of a CLP program can be large, due to the ability to express constraints whose internal manipulation involves exponential algorithms. This problem can be greatly lessened by taking care to formulate the constraints as several independent sets, i.e., that share no variables, in order to manipulate them without amassing one giant constraint. The general inspiration is the classic one of dividing an exponential problem into $\alpha > 1$ sub-problems, so that an x^n problem reduces to an $\alpha x^{n/\alpha}$ problem, where $\alpha x^{n/\alpha} \ll x^n$. We add that forming independent constraints seems easier when the language supports expression over separate computational domains, as PrologIII does.
We have observed that often it is beneficial to incorporate redundant constraints into one's program. To illustrate, we consider a premise P that states a property required of an acceptable, completed object (e.g., a reaction pathway must have $\mathcal{S}$ species). Expressing P directly may yield little ability to prune unfruitful partial objects (e.g., a partial pathway of less than $\mathcal{R}$ steps), since it expresses a property of completed objects. However, P together with another premise P_2 may entail a property that does directly constrain partial objects, enabling earlier search cutoffs.

[7] This result is from an 8-species/5-step problem for which 6 of the species formulas are known. Simpler examples take less time.

Continuing with our example, P_2 states that a step has at most two products. Hence, if s, r are respectively the number of species and steps of a partial pathway, then the property $s + 2(\mathcal{R} - r) \geq S$ must hold for any partial pathway. In this way we derive a constraint usable earlier in the generation process.
Finally, our work strongly suggests that the wholly separate processing of numerical and boolean constraints in PrologIII, illustrated by the inability of $x < y$ to unify with a boolean variable, is not a large obstacle to the solution of tasks presenting both aspects.

Acknowledgment. We thank Profs. Herbert Simon, Tom Mitchell, and Bruce Buchanan for guiding this research on automated chemistry. We thank Prof. Mitchell for hosting the first author at CMU. PrologIA graciously allowed us to use PrologIII before its release. Nevin Heintze discussed with us our ideas on CLP, and commented in detail on a draft of this paper.

The first author acknowledges research support from the French Conseil Regional PACA, and from G.I.A Marseille Luminy. The second author is supported by a Graduate Fellowship from the Avionics Laboratory at Wright-Patterson Air Force Base, awarded by Universal Energy Systems.

References

[1] J. Jaffar and J.-L. Lassez, "Constraint logic programming," in *Proc. of Conf. on Principles of Programming Languages*, January 1987.

[2] J. Jaffar and S. Michaylov, "Methodology and implementation of a CLP system," in *Proc. of 4th Int. Conf. on Logic Programming*, May 1987.

[3] M. Dincbas, H. Simonis, and P. van Hentenryck, "Extending equation solving and constraint handling in logic programming," in *Colloquium on Resolution of Equations in Algebraic Structures* (MCC, ed.), May 1987.

[4] A. Colmerauer, "Opening the Prolog III universe," *Byte*, pp. 177–182, August 1987.

[5] A. Colmerauer, "Une introduction a Prolog III," rapport technique, groupe d'intelligence artificielle, Faculte des Sciences de Luminy, Marseille, France, 1989.

[6] A. Colmerauer, "Notes sur Prolog III," actes du seminair: programmation en logique, CNET Tregastel, France, May 1986.

[7] H. Simonis, "Test generation using the constraint logic programming language CHIP," in *Proc. of 6th Int. Conf. on Logic Programming*, 1989.

[8] H. Simonis and M. Dincbas, "Using logic programming for fault diagnosis in digital circuits," in *German Workshop on Artificial Intelligence*, pp. 139–148, September 1987.

[9] N. Heintze, S. Michaylov, and P. Stuckey, "CLP(R) and some electrical engineering problems," in *Proc. of 4th Int. Conf. on Logic Programming*, pp. 675–703, 1987. To appear in J. of Logic Programming.

[10] N. Heintze, S. Michaylov, and P. Stuckey, "CLP(R) and some electrical engineering problems," Technical Report CMU-CS-89-139, Carnegie Mellon University, 1989.

[11] M. Dincbas, H. Simonis, and P. van Hentenryck, "Solving the car-sequencing problem in constraint logic programming," in *Proc. of European Conf. on Artificial Intelligence*, 1988.

[12] M. Dincbas, P. van Hentenryck, H. Simonis, A. Aggous, and T. Graf, "Applications of CHIP to industrial and engineering problems," in *1st Int. Conf. on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems*, June 1988.

[13] C. Lassez, "An application of constraint logic programming in finance." To appear in Systems and Control Encyclopedia.

[14] R. E. Valdes-Perez, "Deductive assistance for elucidation of chemical reaction pathways," Technical Report CMU-CS-90-104, Carnegie Mellon University, 1990.

[15] R. E. Valdes-Perez, "Deductive assistance for elucidation of reaction pathways." Annual Meeting of American Institute of Chemical Engineers, 1990.

A Example

Our example is the following two-reagent reaction,

$$C_{13}H_{18}NX + C_{17}H_{17}N_2X \rightsquigarrow C_{31}H_{37}N_2X + C_{24}H_{27}N_2X + C_6H_7N + HX$$

which specifies four products that have been identified (X is an alias for either of the halogens I or Cl). The target is the first product, a blue dye of much commercial value. The program outputs these ten simplest pathway hypotheses, all having seven species and three steps (schematic versions are shown at the right).

1.

$$\begin{array}{rcl} C_{17}H_{17}N_2X & \rightarrow & C_6H_7N + C_{11}H_{10}NX \\ C_{13}H_{18}NX + C_{11}H_{10}NX & \rightarrow & HX + C_{24}H_{27}N_2X \\ 2C_{24}H_{27}N_2X & \rightarrow & C_{17}H_{17}N_2X + C_{31}H_{37}N_2X \end{array} \qquad \begin{array}{rcl} B & \rightarrow & W + Y \\ A + W & \rightarrow & X + Z \\ 2X & \rightarrow & T + B \end{array}$$

2.

$$\begin{array}{rcl} C_{17}H_{17}N_2X & \rightarrow & HX + C_{17}H_{16}N_2 \\ C_{13}H_{18}NX + C_{17}H_{16}N_2 & \rightarrow & C_6H_7N + C_{24}H_{27}N_2X \\ 2C_{24}H_{27}N_2X & \rightarrow & C_{17}H_{17}N_2X + C_{31}H_{37}N_2X \end{array} \qquad \begin{array}{rcl} B & \rightarrow & W + Z \\ A + W & \rightarrow & X + Y \\ 2X & \rightarrow & T + B \end{array}$$

3.

$$\begin{array}{rcl} C_{13}H_{18}NX & \rightarrow & HX + C_{13}H_{17}N \\ 2C_{24}H_{27}N_2X & \rightarrow & C_{17}H_{17}N_2X + C_{31}H_{37}N_2X \\ C_{17}H_{17}N_2X + C_{13}H_{17}N & \rightarrow & C_6H_7N + C_{24}H_{27}N_2X \end{array} \qquad \begin{array}{rcl} A & \rightarrow & W + Z \\ B + W & \rightarrow & X + Y \\ 2X & \rightarrow & T + B \end{array}$$

4.

$$\begin{array}{rcl} C_{13}H_{18}NX & \rightarrow & C_6H_7N + C_7H_{11}X \\ C_{17}H_{17}N_2X + C_7H_{11}X & \rightarrow & HX + C_{24}H_{27}N_2X \\ 2C_{24}H_{27}N_2X & \rightarrow & C_{17}H_{17}N_2X + C_{31}H_{37}N_2X \end{array} \qquad \begin{array}{rcl} A & \rightarrow & W + Y \\ B + W & \rightarrow & X + Z \\ 2X & \rightarrow & T + B \end{array}$$

5.

$$\begin{array}{rcl} C_{13}H_{18}NX & \rightarrow & HX + C_{13}H_{17}N \\ C_{17}H_{17}N_2X + C_{13}H_{17}N & \rightarrow & C_6H_7N + C_{24}H_{27}N_2X \\ C_{13}H_{17}N + C_{24}H_{27}N_2X & \rightarrow & C_6H_7N + C_{31}H_{37}N_2X \end{array} \qquad \begin{array}{rcl} A & \rightarrow & W + Z \\ B + W & \rightarrow & X + Y \\ W + X & \rightarrow & T + Y \end{array}$$

6.

$$\begin{array}{rcl} C_{13}H_{18}NX & \rightarrow & C_6H_7N + C_7H_{11}X \\ C_{17}H_{17}N_2X + C_7H_{11}X & \rightarrow & HX + C_{24}H_{27}N_2X \\ C_7H_{11}X + C_{24}H_{27}N_2X & \rightarrow & HX + C_{31}H_{37}N_2X \end{array} \qquad \begin{array}{rcl} A & \rightarrow & W + Y \\ B + W & \rightarrow & X + Z \\ W + X & \rightarrow & T + Z \end{array}$$

7.

$$\begin{array}{rcl} C_{13}H_{18}NX + C_{17}H_{17}N_2X & \rightarrow & C_6H_8NX + C_{24}H_{27}N_2X \\ 2C_{24}H_{27}N_2X & \rightarrow & C_{17}H_{17}N_2X + C_{31}H_{37}N_2X \\ C_6H_8NX & \rightarrow & HX + C_6H_7N \end{array} \qquad \begin{array}{rcl} A + B & \rightarrow & W + X \\ 2X & \rightarrow & T + B \\ W & \rightarrow & Y + Z \end{array}$$

8.

$$\begin{array}{rcl} C_{13}H_{18}NX + C_{17}H_{17}N_2X & \rightarrow & HX + C_{30}H_{34}N_3X \\ 2C_{24}H_{27}N_2X & \rightarrow & C_{17}H_{17}N_2X + C_{31}H_{37}N_2X \\ C_{30}H_{34}N_3X & \rightarrow & C_6H_7N + C_{24}H_{27}N_2X \end{array} \qquad \begin{array}{rcl} A + B & \rightarrow & W + Z \\ 2X & \rightarrow & T + B \\ W & \rightarrow & X + Y \end{array}$$

9.

$$\begin{array}{rcl} C_{13}H_{18}NX + C_{17}H_{17}N_2X & \rightarrow & C_6H_7N + C_{24}H_{28}N_2X_2 \\ 2C_{24}H_{27}N_2X & \rightarrow & C_{17}H_{17}N_2X + C_{31}H_{37}N_2X \\ C_{24}H_{28}N_2X_2 & \rightarrow & HX + C_{24}H_{27}N_2X \end{array} \qquad \begin{array}{rcl} A + B & \rightarrow & W + Y \\ 2X & \rightarrow & T + B \\ W & \rightarrow & X + Z \end{array}$$

10.

$$\begin{array}{rcl} C_{13}H_{18}NX + C_{17}H_{17}N_2X & \rightarrow & C_6H_8NX + C_{24}H_{27}N_2X \\ C_{13}H_{18}NX + C_{24}H_{27}N_2X & \rightarrow & C_6H_8NX + C_{31}H_{37}N_2X \\ C_6H_8NX & \rightarrow & HX + C_6H_7N \end{array} \qquad \begin{array}{rcl} A + B & \rightarrow & W + X \\ A + X & \rightarrow & T + W \\ W & \rightarrow & Y + Z \end{array}$$

B Program listing

This is a complete listing of the PrologIII fragments in the body of the paper, presented here in depth-first rule order. The program has been simplified somewhat for this paper, but it is very close to the actual program.

$$
\begin{array}{l}
Pathway(\mathcal{R},\mathcal{S},p,q) \rightarrow \\
\quad All_formulas_are_distinct(p) \\
\quad Pathway_template(\mathcal{S},\mathcal{R},q,v,u) \\
\quad All_distinct_steps(q) \\
\quad Need_both_reagents(p) \\
\quad Enumerate_the_species(v,p,u);
\end{array}
$$

$$
\begin{array}{l}
All_formulas_are_distinct(\langle\rangle) \rightarrow; \\
All_formulas_are_distinct(\langle e\rangle.l) \rightarrow \\
\quad Pairwise_distinct_formulas(e,l) \\
\quad All_formulas_are_distinct(l);
\end{array}
$$

$$
\begin{array}{l}
Pairwise_distinct_formulas(e,\langle\rangle) \rightarrow; \\
Pairwise_distinct_formulas(\langle n1,x1,a1,b1,f1\rangle,\langle\langle n2,x2,a2,b2,f2\rangle\rangle.l) \rightarrow \\
\quad Pairwise_distinct_formulas(\langle n1,x1,a1,b1,f1\rangle,l), \\
\quad \{f1 \neq f2\};
\end{array}
$$

$$
Pathway_template(\mathcal{S},0,\langle\rangle,\langle\rangle,u) \rightarrow;
$$

$$
\begin{array}{l}
Pathway_template(\mathcal{S},r,\langle\langle r1,r2,p1,p2\rangle\rangle.l,\langle\langle r1,x1,a1,b1,f1\rangle, \\
\langle r2,x2,a2,b2,f2\rangle,\langle p1,x3,a3,b3,f3\rangle,\langle p2,x4,a4,b4,f4\rangle\rangle.v,u) \rightarrow \\
\quad Step_template(\langle r1,r2,p1,p2\rangle,\mathcal{S},u) \\
\quad Attached_equation_system(\langle x1,x2,x3,x4\rangle) \\
\quad Balanced_step(\langle f1,f2,f3,f4\rangle) \\
\quad Attached_clause_system(\langle a1,a2,a3,a4\rangle,\langle b1,b2,b3,b4\rangle) \\
\quad Pathway_template(\mathcal{S},r-1,l,v,u);
\end{array}
$$

$$
\begin{array}{l}
Step_template(\langle r_1,r_2,p_1,p_2\rangle,\mathcal{S},u) \rightarrow, \\
\quad available(r_1,u) \\
\quad available(r_2,u), \\
\quad \{ \\
\quad 0 \leq r_1 \leq \mathcal{S}, \\
\quad 1 \leq r_2 \leq \mathcal{S}, \\
\quad 0 \leq p_1 \leq \mathcal{S}, \\
\quad 1 \leq p_2 \leq \mathcal{S}, \\
\quad /*\, next,\ prohibit\ R+R \rightarrow P+P\ */ \\
\quad (r_1+r_2)+(p_1+p_2)*1000 \neq 2r_2+2p_2*1000, \\
\quad r_1 \neq p_1, r_1 \neq p_2, \\
\quad r_2 \neq p_1, r_2 \neq p_2, \ /*\, reactants\ and\ products\ are\ distinct\ */ \\
\quad r_1 \leq r_2, p_1 \leq p_2 \ /*\, ordering\ on\ the\ identifiers\ */ \\
\quad \};
\end{array}
$$

$available(e,l) \rightarrow freeze(e, member(e,l));$

$Attached_equation_system(\langle x_r1, x_r2, x_p1, x_p2\rangle) \rightarrow,$
$\{x_r1 + x_r2 = x_p1 + x_p2\};$

$Balanced_step(\langle\langle\rangle, \langle\rangle, \langle\rangle, \langle\rangle\rangle) \rightarrow;$
$Balanced_step(\langle\langle a\rangle.f_r1, \langle b\rangle.f_r2, \langle c\rangle.f_p1, \langle d\rangle.f_p2\rangle) \rightarrow$
$Balanced_step(\langle f_r1, f_r2, f_p1, f_p2\rangle),$
$freeze(a, integer(a))\ freeze(b, integer(b))$
$freeze(c, integer(c))\ freeze(d, integer(d))$
$\{a + b = c + d, a \geq 0, b \geq 0, c \geq 0, d \geq 0\};$

$Attached_clause_system(\langle a_r1, a_r2, a_p1, a_p2\rangle, \langle b_r1, b_r2, b_p1, b_p2\rangle) \rightarrow,$
$\{a_r1 \wedge a_r2 \Rightarrow a_p1, \quad a_r1 \wedge a_r2 \Rightarrow a_p2,$
$b_r1 \wedge b_r2 \Rightarrow b_p1, \quad b_r1 \wedge b_r2 \Rightarrow b_p2\};$

$All_distinct_steps(\langle\rangle) \rightarrow;$
$All_distinct_steps(\langle\langle r1, r2, p1, p2\rangle\rangle.l) \rightarrow$
$Pairwise_distinct_steps(\langle r1, r2, p1, p2\rangle, l)$
$All_distinct_steps(l);$

$Pairwise_distinct_steps(e, \langle\rangle) \rightarrow;$
$Pairwise_distinct_steps(\langle r1, r2, p1, p2\rangle, \langle\langle r3, r4, p3, p4\rangle\rangle.l) \rightarrow$
$Pairwise_distinct_steps(\langle r1, r2, p1, p2\rangle, l),$
$\{\langle r1, r2, p1, p2\rangle \neq \langle r3, r4, p3, p4\rangle, \langle r1, r2, p1, p2\rangle \neq \langle p3, p4, r3, r4\rangle\};$

$Need_both_reagents(\langle\langle 1, x1, a1, b1, f1\rangle.\langle 2, x2, a2, b2, f2\rangle.$
$\langle 3, x3, a3, b3, f3\rangle\rangle.l) \rightarrow,$
$\{a1 = 1', a3 = 0', b2 = 1', b3 = 0'\};$

$Enumerate_the_species(v, p, \langle 0, 1, 2\rangle.u) \rightarrow$
$Instantiate_species(v, p, u);$

$Instantiate_species(\langle\rangle, p, k) \rightarrow;$
$Instantiate_species(\langle\langle n, x, a, b, f\rangle\rangle.v, p, u) \rightarrow$
$integer(n)$
$belong_to(\langle n, x, a, b, f\rangle, p)$
$eq(u, \langle n\rangle.k)$
$Instantiate_species(v, p, k);$

$belong_to(\langle 0, 0, 1', 1', f\rangle, l) \rightarrow$
$coefficients_all_zero(f);$

$belong_to(\langle n, x, a, b, f\rangle, \langle\langle n, x, a, b, f\rangle\rangle.l) \rightarrow,$
$\{n \neq 0,\ x \geq 1\};$

$belong_to(\langle n, x, a, b, f\rangle, \langle\langle n', x', a', b', f'\rangle\rangle.l) \rightarrow$
$belong_to(\langle n, x, a, b, f\rangle, l),$
$\{n \neq 0, n \neq n'\};$

$coefficients_all_zero(\langle\rangle) \rightarrow;$
$coefficients_all_zero(\langle 0\rangle.l) \rightarrow$
$coefficients_all_zero(l);$