Acta Informatica 11, 1–30 (1978)

A Synthesis of Several Sorting Algorithms

John Darlington
Imperial College of Science and Technology, Department of Computing and Control,
180 Queen's Gate, London SW7 2BZ, United Kingdom

Summary. We synthesise versions of six well known sorting algorithms from a common specification using program transformation techniques. On the way to the sorting algorithms we synthesise three algorithms for generating permutations thus building up a family tree for the sorts exposing certain relationships between them.

1. Introduction

Where do algorithms come from? Existing techniques of algorithm analysis by and large treat algorithms as "pre-existing" immutable objects; in this paper we investigate an alternative approach concentrating more on the *origins* of algorithms. For a class of algorithms we study the nature of each algorithm and their relationships to one another by attempting to synthesise the algorithms systematically from a common high level definition of the task to be performed, constructing a family tree of algorithms.

In this paper we look at the sorting algorithms, starting with a common high level mathematical definition of what it means to sort an array or list we synthesise six well-known sorting algorithms: Quick Sort, Merge Sort, Insertion Sort, Selection Sort, Exchange Sort and Bubble Sort.

This approach has grown out of our work on automatic program transformation and synthesis, and while we admit that the mechanisation of the syntheses detailed here is impracticable at present, this approach has strongly influenced our methodology. The basis for each synthesis is a small set of formal program transformation rules and a set of reduction rules for set expressions. We aim to divide the syntheses into (i) the mathematical kernels, the key reductions that allow the synthesis to proceed and are the basic ideas behind each algorithm and (ii) a lot of trivial, obviously correct, applications of the program transformation and set reduction rules. Thus the syntheses could be machine checked if not machine generated, and we hope that our mechanical approach will highlight the decisions and mathematical facts that lie behind each algorithm.

This constructive approach to algorithm analysis and design has been advocated by Dijkstra, [4] and others. We differ mainly in adopting a more mathematical approach, in particular a more mathematical programming language, and in attempting to cover a whole class of algorithms. Manna and Waldinger in their work on automatic programming [6] have independently developed rules similar to our program transformation rules and we have benefited greatly from interactions with them, as we have with members of Cordell Green's automatic programming group who have developed a totally rule-driven system that is able to automatically generate several sorts [5].

In Sect. 2 we use a small example to illustrate our style of high level definition, target programming language, program transformation rules and set reduction rules. In Sect. 3, we describe the notation needed for the synthesis of the sorting algorithms. Sect. 4 gives an outline of the structure of the syntheses and Sects. 5 and 6 contain the detailed syntheses.

The style of language we use and the program transformation rules are described fully in Burstall and Darlington [1] and their application to program synthesis is outlined in Darlington [3].

2. Basic Rules and an Example of Synthesis

In this section we give the program transformation rules and the basic set reduction rules we will use. Our language is a simple equational one. At the top, definition, level the right hand sides will consist mainly of set and predicate logic constructs while at the target language level the right hand sides will be recursively defined expressions.

2.1. Program Transformation Rules

Given a set of equations we may add to them using the following inference rules.

(i) Definition. Introduce a new recursion equation whose left hand expression is not an instance of the left hand expression of any previous equation.

(ii) Instantiation. Introduce a substitution instance of an existing equation.

(iii) Unfolding. If $E \Leftarrow E'$ and $F \Leftarrow F'$ are equations and there is an occurrence in F' of an instance of E, replace it by the corresponding instance of E' obtaining F'' then add the equation $F \Leftarrow F''$.

(iv) Folding. If $E \Leftarrow E'$ and $F \Leftarrow F'$ are equations and there is some occurrence in F' of an instance of E', replace it by the corresponding instance of E obtaining F'', then add the equation $F \Leftarrow F''$.

(v) Laws. We may transform an equation by using on its right hand expression any laws we have about the primitives obtaining a new equation.

The new equations obtained by these rules may be taken as a definition of the function appearing on the left provided we take a disjoint and exhaustive set of them (the notion of disjointness and exhaustiveness depend on the data domain; we do not attempt an explicit definition but they are clear enough for integers, lists and sets).

In the above rules unfolding corresponds to the symbolic evaluation of recursively defined functions. In the case where we have set-constructs on the right hand side that have no immediate expansion, we use a set of reduction rules for such constructs which we give below. These are used via rule (v).

The novel rules is (iv), folding. This is the way that new recursions are introduced into a system of equations. We use this rule to replace non-executable set-expressions by recursively defined functions. A set-expression on the right hand side of an equation is reduced until an appropriate instance re-occurs and folding is then used to introduce a recursion.

2.2. Basic Set Reduction Rules for Set-Expressions

The following reduction rules are used repeatedly. Other reduction rules also used will be detailed when needed. The form of the rules will be set-expression$\Leftarrow$ set-expression indicating that the expression of the left hand side can be reduced to the expression on the right.

(i) Membership

RM1 $\{f(x) | x \in \Phi \textit{ and } P(x)\} \Leftarrow \Phi$

RM2 $\{f(x) | x \in s+S \textit{ and } P(x)\} \Leftarrow \{f(s)\} \cup \{f(x) | x \in S \textit{ and } P(x)\}$ **if** $P(s)$
$\{f(x) | x \in S \textit{ and } P(x)\}$ **otherwise**
$(s+S \textit{ is } \{s\} \cup S)$

RM3 $\{f(x) | x \in S1 \cup S2 \textit{ and } P(x)\} \Leftarrow \{f(x) | x \in S1 \textit{ and } P(x)\} \cup \{f(x) | x \in S2 \textit{ and } P(x)\}.$

(ii) Subset

RS1 $\{f(X) | X \subseteq \Phi \textit{ and } P(X)\} \Leftarrow \{\Phi\}$ **if** $P(\Phi)$
Φ **otherwise**

RS2 $\{f(X) | X \subseteq s+S \textit{ and } P(X)\} \Leftarrow \{f(X) | X \subseteq S \textit{ and } P(X)\} \cup \{f(s+X) | X \subseteq S \textit{ and } P(s+X)\}$

RS3 $\{f(X) | X \subseteq S1 \cup S2 \textit{ and } P(X)\} \Leftarrow \{f(X1 \cup X2) | X1 \subseteq S1 X2 \subseteq S2 \textit{ and } P(X1 \cup X2)\}.$

(iii) Cartesian Product

RC1 $\Phi \times T \Leftarrow \Phi$

RC2 $(S1 \cup S2) \times T \Leftarrow S1 \times T \cup S2 \times T.$

2.3. Synthesis

We will now give a preview of our syntheses by outlining the synthesis of a program to calculate the set of all functions between two sets. We define it thus

1. $Funcs(S, T) \Leftarrow \{f | f \subseteq S \times T \textit{ and } Isfunc(f)\}$
2. $Isfunc(f) \Leftarrow \forall \langle x1\ y1 \rangle \in f,\ \langle x2\ y2 \rangle \in f.\ x1 = x2 \Rightarrow y1 = y2.$

Notice the style of our definition. It is of the "generate and test" form. Even if this definition could be directly interpreted or compiled it would result in very inefficient computation as the large set $2^{S\times T}$ would be formed first and then many of its members filtered out by the *Isfunc* test. Our synthesis will use folding to promote this filter into the generation process producing a recursive definition for *Funcs* in which only acceptable candidates are produced at each level.

We start by using instantiation to derive from equation 1.

3. $Funcs(\Phi, T) \Leftarrow \{f \mid f \subseteq \Phi \times T \text{ and } Isfunc(f)\}$
 $\Leftarrow \{f \mid f \subseteq \Phi \text{ and } Isfunc(f)\}$ Using rule RC1
 $\Leftarrow \{\Phi\}$ **if** $Isfunc(\Phi)$
 Φ **otherwise** Using rule RS1
 $\Leftarrow \{\Phi\}$ Unfolding using definition of *Isfunc*.

We need to consider another base case, viz. $Funcs(\{s\}, T)$. This is fairly obviously $\{\{\langle s\ t\rangle\} \mid t \in T\} \cup \{\Phi\}$ but we will synthesise a recursion for this as it illustrates a technique we will use often later.

First we have

4. $Funcs(\{s\}, \Phi) \Leftarrow \{\Phi\}$ as above
5. $Funcs(\{s\}, t+T) \Leftarrow \{f \mid f \subseteq \{s\} \times t+T \text{ and } Isfunc(f)\}$
 $\Leftarrow \{f \mid f \subseteq \{s\} \times \{t\} \cup \{s\} \times T \text{ and } Isfunc(f)\}$ Rule RC2
 $\Leftarrow \{f1 \cup f2 \mid f1 \subseteq \{s\} \times \{t\}\ f2 \subseteq \{s\} \times T \text{ and } Isfunc(f1 \cup f2)\}$
 Rule RS3.

We now examine what values *f1* and *f2* can take. *f1* is either Φ or $\{\langle s\ t\rangle\}$. In the latter case unless *f2* is empty it will map s onto a $t' \in T$ such that $t \neq t'$ and we will have $\neg Isfunc(f1 \cup f2)$. Thus we can rewrite the above as

$\Leftarrow \{\{\langle s\ t\rangle\}\}$
$\cup \{f2 \mid f2 \subseteq \{s\} \times T \text{ and } Isfunc(f2)\}$
$\Leftarrow \{\{\langle s\ t\rangle\}\} \cup Funcs(\{s\}, T)$ Folding with 1.

Finally we synthesise the main recursion, which we get by letting S be $S1 \cup S2$ where $S1$ and $S2$ are disjoint.

6. $Funcs(S1 \cup S2, T) \Leftarrow \{f \mid f \subseteq (S1 \cup S2) \times T \text{ and } Isfunc(f)\}$ Instantiating 1
 $\Leftarrow \{f \mid f \subseteq S1 \times T \cup S2 \times T \text{ and } Isfunc(f)\}$ Using RC2
 $\Leftarrow \{f1 \cup f2 \mid f1 \subseteq S1 \times T$
 $f2 \subseteq S2 \times T$
 $\text{and } Isfunc(f1 \cup f2)\}$ Using RS3.

Our aim is to get a fold so we can introduce a recursion. We see that 6 is almost in a form that will fold twice with 1. To achieve this it is necessary to decompose the test $Isfunc(f1 \cup f2)$ into a form involving $Isfunc(f1)$ and $Isfunc(f2)$. In this case it is simple as we can see that *f1* and *f2* having disjoint domains $f1 \cup f2$ will be functional iff both *f1* and *f2* are functions i.e. $Isfunc(f1 \cup f2) \Leftrightarrow Isfunc(f1) \text{ and } Isfunc(f2)$. Thus we can rewrite the above as

$\Leftarrow \{f1 \cup f2 \mid f1 \subseteq S1 \times T \text{ and } Isfunc(f1)$
 $f2 \subseteq S2 \times T \text{ and } Isfunc(f2)\}$
$\Leftarrow \{f1 \cup f2 \mid f1 \in \{f3 \mid f3 \subseteq S1 \times T \text{ and } Isfunc(f1)\}$
 $f2 \in \{f4 \mid f4 \subseteq S2 \times T \text{ and } Isfunc(f2)\}\}$
$\Leftarrow \{f1 \cup f2 \mid f1 \in Funcs(S1, T)$
 $f2 \in Funcs(S2, T)\}$ Folding with 1.

Equations 3–6 define a recursion that computes *Funcs* ensuring that no unacceptable sets are produced at any level.

This process of applying set reductions, checking how the property of the constructed object depends on the properties of each of its components and then folding will re-occur repeatedly in our syntheses, and we give it the name "filter promotion".

3. Notation and Representation

Our basic notation will be that of informal set-theory, most of whose symbols have been used in the previous section. In addition we will use $\bigcup$ to denote union over a family of sets and $-$ to denote set subtraction, i.e. $X - Y = \{z \mid z \in X \textit{ and } z \notin Y\}$. Our other data types will be sequences and functions. We will often use sets and sequences interchangeably where the context makes it clear what is meant. In particular sets with a natural order over them, in our case always the non-negative integers will often be treated as sequences. Conversely certain set operation symbols ($\in$, $+$, Φ, etc.) will be used to denote the equivalent operation on sequences. Additional operations solely on sequences are.

first: *sequence* $\rightarrow$ *element*
:– takes the first element of the sequence
$\textit{first}(x+X) \equiv x$

rest: *sequence* $\rightarrow$ *sequence*
:– returns the rest of the sequence
$\textit{rest}(x+X) \equiv X$

[ ]: *set* $\rightarrow$ *sequence*
:– the initial sequence of integers up to the cardinality of the set
$[X] \equiv \langle 1, 2, \ldots, \textit{cardinality of } X \rangle$

N^k: *sequence* $\times$ *integer* $\rightarrow$ *sequence*
:– the initial segment of the sequence with length equal to the integer
$\langle n_1, n_2, \ldots, n_l \rangle^k \equiv \langle n_1, n_2, \ldots, n_k \rangle$

N_k: sequence $\times$ integer $\rightarrow$ sequence
:– the rest of the sequence
$\langle n_1, n_2, \ldots, n_l \rangle_k \equiv \langle n_{k+1}, \ldots, n_l \rangle$.

Functions are sets of ordered pairs, and we use the following operations on them.

Domain, Image: *function* $\rightarrow$ *set*
:– $\textit{Domain}(f) \equiv \{x \mid \langle x\ y \rangle \in f\}$
:– $\textit{Image}(f) \equiv \{y \mid \langle x\ y \rangle \in f\}$

$-_{Im}$: *function* $\times$ *element* $\rightarrow$ *function*
:– $f -_{Im} y1 \equiv \{\langle x\ y \rangle \mid \langle x\ y \rangle \in f \textit{ and } y1 \neq y\}$.

The objects we will sort will be functions from a sequence of the integers into a set of atoms with some total order, $\leqq$, over them. We will often treat these sets of ordered pairs as sequences of ordered pairs, the order determined by the natural order over the domain (the integers). These functions can be thought of as representing the more usual computer arrays or lists, thus the function $\{\langle 1\ b\rangle, \langle 2\ a\rangle, \langle 3\ c\rangle\}$ represents the array $[b\ a\ c]$.

The use of such a "mathematical" representation for the more normal data structures makes our algorithms much more amenable to formal manipulation. The final algorithms could easily be translated into a more familiar form by the use of a "representation relationship" (see Burstall and Darlington [1]) mapping sets of ordered pairs into lists or arrays.

4. Structure and Presentation

4.1. Structure of the Syntheses

We first define Perm, the set of all permutations of a set. Using functions to represent arrays enables us to define *Perm*(X) simply as the set of all bijective (one to one and onto) functions from $[X]$ to X. As we saw earlier, functions are easily defined as a filter on the set of all subsets of the Cartesian product, thus we have

$$Perm(X) \Leftarrow \{f \mid f \subseteq [X] \times X \text{ and } Bijective(f, [X], X)\}$$

where

$$Bijective(f, Y, X) \Leftarrow f \text{ is a total one to one function from } Y \text{ onto } X.$$

Notice that this definition is in the generate and test form. From this high level definition, which we call P, we can synthesise recursive algorithms that compute Perm much more efficiently. Three different ways of proceeding with these syntheses give us three different algorithms for generating permutations which we call *P1*, *P2*, *P3*.

We then define *Sort* which takes a set and returns the function representing the ordered array. We do this by filtering out of *Perm* all the unordered functions i.e. those functions whose range elements are not in the order established by the natural order on the domain.

Thus

$$Sort(X) \Leftarrow Ordered(Perm(X))$$

where

$$Ordered(X) \Leftarrow \{f \mid f \in X \text{ and } Ord(f)\}$$

and

$$Ord(f) \Leftarrow \forall \langle n1\ x1\rangle \in f, \langle n2\ x2\rangle \in f. \quad n1 < n2 \Leftrightarrow x1 < x2.$$

We again take this generate and test definition and seek to promote the filtering. For each of the 3 permutation algorithms *P1*, *P2* and *P3* we synthesise two sorting algorithms deriving versions of Quick Sort, Selection Sort, Merge

Sort, Insertion Sort, Bubble Sort and Exchange Sort respectively. The structure of our syntheses is thus

$$\begin{tikzcd}
 & & P \arrow[dl] \arrow[d] \arrow[drr] & & & \\
 & P1 \arrow[dl] \arrow[d] & P2 \arrow[d] \arrow[dr] & & P3 \arrow[d] \arrow[dr] & \\
\textit{Quick Sort} & \textit{Selection Sort} & \textit{Merge Sort} & \textit{Insertion Sort} & \textit{Exchange Sort} & \textit{Bubble Sort}
\end{tikzcd}$$

The derivation of the first four sorts shows a pleasing symmetry, with the derivation of *P3* and *Exchange Sort* and *Bubble Sort* being different in character. The difference in the synthesis for *P1* and *P2* is that in *P1* the "work" necessary to ensure that the bijective filter is satisfied is done by decomposition, going down the recursion, while in *P2* it is done on reconstruction, coming up the recursion. The difference between *Quick Sort* and *Selection Sort* and *Merge Sort* and *Insertion Sort* is the same viz. for *Quick Sort* and *Selection Sort* we recurse by decomposing a set *S* into two disjoint sets *S1*, *S2* while for *Selection Sort* and *Insertion Sort* is the same viz. for *Quick Sort* and *Merge Sort* we recurse and *Bubble Sort* are very similar. *P3* is intrinsically an infinite algorithm that cycles through the space of all permutations. *Exchange Sort* and *Bubble Sort* differ in the way they move through this space towards the ordered permutation.

The idea of defining sorting as selecting the ordered permutation has been used previously by Bob Kowalski and Maarten van Emden in their work on Predicate Logic programming.

4.2. Presentation of the Syntheses

This paper has two, perhaps conflicting, aims. Firstly, we would like to show that program transformation methods can be used for the complete derivation of non-trivial algorithms and, secondly, we would like to present the syntheses in such a way that the structure of the derivations is easily assimilated and the relationships between the various algorithms becomes apparent. The first approach leans to the eventual mechanisation of this process, while the second is more inclined towards using program transformation for manual program development or analysis. Thus we would like to base our deductions on as small a set of transformation rules as possible, but present them in a high level or structured manner. We are, therefore, building up a catalogue of higher level transformation types which can be used to discover and express the transformations in, and which, if needed, can be further expanded to a series of applications of the fundamental transformation rules. Filter promotion is one such high level transformation. Thus we have adopted a two level approach to the presentation of the syntheses. Section 5 gives a top level view of the syntheses. Each derivation is presented as an initial "inspiration step" or lemma which hopefully gives the basic fact underlying the particular algorithm and then a series of high level transformations that improve efficiency. These high level transformations are almost all filter promotions. In Sect. 6 each of these high level transformations is expanded out into more fundamental transformations.

5. The Syntheses

5.1.

Our top level definition defines the set of all permutations of a set.

P $Perm(X) \Leftarrow \{f \mid f \subseteq [X] \times X \text{ and } Bijective(f, [X], X)\}$

where

$Bijective(f, Y, X) \Leftarrow f$ *is a total one to one function from Y onto X.*

5.2. Synthesis of *P1*, *Quick Sort* and *Selection Sort* from *P*

5.2.1. *P1* from *P*

Repeating P we have

1. $Perm(X) \Leftarrow \{f \mid f \subseteq [X] \times X \text{ and } Bijective(f, [X], X)\}$

which we generalise to

2. $Perm1(N, X) \Leftarrow \{f \mid f \subseteq N \times X \text{ and } Bijective(f, N, X)\}$.

The first lemma we need for this synthesis is that $X = \bigcup_{Y \subset X} Y$. In fact we do not need to take all subsets of X, all those of equal size will do. We will denote this by writing $X = \bigcup_{Y \subset_k X} Y$ meaning all subsets Y of X of cardinality k, $1 \leqq k < Card(X)$. Given this it is easily shown that $N \times X = \bigcup_{Y \subset_k X} (N^k \times Y) \cup (N_k \times (X - Y))$, thus we rewrite 2 as

3. $Perm(N, X) \Leftarrow \{f \mid f \subseteq \bigcup_{Y \subset_k X} (N^k \times Y) \cup (N_k \times (X - Y)) \text{ and } Bijective(f, N, X)\}$.

In 6.1 we will show how the Bijective filter can be promoted before the union. We get

4. $$Perm1(N, X) \Leftarrow \bigcup_{Y \subset_k X} \{f1 \cup f2 \mid f1 \subseteq N^k \times Y \text{ and } Bijective(f1, N^k, Y)$$
$$f2 \subseteq N_k \times (X - Y) \text{ and } Bijective(f2, N_k, X - Y)\}$$
$$\Leftarrow \bigcup_{Y \subset_k X} \{f1 \cup f2 \mid f1 \in Perm1(N^k, Y), f2 \in Perm1(N_k, X - Y)\}$$
Folding with 2.

As for any call of $Perm1(N, Y)$, $Card(N) = Card(X)$ the base cases we need to consider are

5. $Perm(\Phi, \Phi) \Leftarrow \{\Phi\}$ Instantiating 1 and rule RS1.

6. $Perm(\{n\}, \{n\}) \Leftarrow \{\{\langle n\ x\rangle\}\}$ Instantiating 1 and evaluating.

Thus we have algorithm *P1*.

P1 $Perm(X) \Leftarrow Perm1([X], X)$
$Perm1(\Phi, \Phi) \Leftarrow \{\Phi\}$
$Perm1(\{n\}, \{x\}) \Leftarrow \{\{\langle n\ x\rangle\}\}$
$$Perm1(N, X) \Leftarrow \bigcup_{Y \subset_k X} \{f_1 \cup f_2 \mid f_1 \in Perm1(N^k, Y),\ f_2 \in Perm1(N_k, X-Y)\}.$$

5.2.2. *Quick Sort* from *P1*. We define *Sort* by

1. $Sort(X) \Leftarrow Ordered(Perm(X))$

where $Ordered: 2^{X \to Y} \to 2^{X \to Y}$ (filters out unordered permutations).

2. $Ordered(X) \Leftarrow \{f \mid f \in X \text{ and } Ord(f)\}$
3. $Ord(f) \Leftarrow \forall \langle n1\ x1\rangle \in f, \langle n2\ x2\rangle \in f$
$\qquad n1 < n2 \Leftrightarrow x1 < x2.$

Thus

4. $Sort(X) \Leftarrow Ordered(Perm(X))$
$\qquad \Leftarrow Ordered(Perm1([X], X))$ Unfolding using *P1*

and we define

5. $Sort1(N, X) \Leftarrow Ordered(Perm1(N, X)).$

Thus

6. $Sort1(\Phi, \Phi) \Leftarrow \{\Phi\}$ Instantiating 5 and unfolding using *P1* and 2.
7. $Sort1(\{n\}, \{x\}) \Leftarrow \{\{\langle n\ x\rangle\}\}$ Instantiating 5 and unfolding using *P1*, 2 and 3.
8. $Sort1(N, X) \Leftarrow Ordered(\bigcup_{Y \subset_k X} \{f1 \cup f2 \mid f1 \in Perm1(N^k, Y)\ f2 \in Perm1(N_k, X-Y)\})$
Instantiating 5 and unfolding *P1*.

A straightforward promotion of the *Ordered* filter (6.2) gets us 9 below.

$(Y < X - Y \text{ means } \forall y \in Y, x \in X - Y \cdot y < x).$

9. $Sort1(N, X)$
$$\Leftarrow \bigcup_{\substack{Y \subset_k X \\ Y < X - Y}} \{f1 \cup f2 \mid f1 \in Perm1(N^k, Y) \text{ and } Ord(f1)\ f2 \in Perm1(N_k, X-Y) \text{ and } Ord(f2)\}$$
$$\Leftarrow \bigcup_{\substack{Y \subset_k X \\ Y < X - Y}} \{f1 \cup f2 \mid f1 \in Sort1(N^k, Y),\ f2 \in Sort1(N_k, X-Y)\}$$ Folding with 2 and 4.

The conditions on Y, $Y \subset_k X$ and $Y < X - Y$, mean that for any k there is just one such Y. Thus we can rewrite the above as

$\Leftarrow \{f1 \cup f2 \mid f1 \in Sort1(N^k, Y)$
$\qquad f2 \in Sort1(N_k, X-Y)\}$
$\qquad$ **where** $k = Card(Y)$
$\qquad\quad$ **for some** $Y \subset X$ **such that** $Y < X - Y$.

In the full *Quick Sort* algorithm Y is chosen by selecting an element from X and then dividing X into two sets one all less than the chosen element and one all greater than the chosen element. We can synthesise this by defining.

10. $Filter(X) \Leftarrow Y$ **such that** $Y \subset X$ *and* $Y < (X - Y)$.
11. $Filter1(x, X) \Leftarrow Y$ **such that** $Y \subset X$ *and* $\forall y \in Y \cdot y < x$.

If we consider the case $x + X$ where $x \notin X$ we have

12. $Filter(x + X) \Leftarrow Y$ **such that** $Y \subset x + X$ *and* $Y < (x + X - Y)$.

Now as $x \in X$, $Y \subset X$ and $\forall y \in Y \cdot y < x$ implies $Y < X - Y$ we can rewrite this as

$\Leftarrow Y$ **such that** $Y \subset X$ *and* $\forall y \in Y \cdot y < x$
$\Leftarrow Filter1(x, X)$ Folding with 11.

The other cases needed are

13. $Filter(\Phi) \Leftarrow \Phi$ Instantiating 10 and rule RS1
14. $Filter1(x, \Phi) \Leftarrow \Phi$ Instantiating 11 and rule RS1
15. $Filter1(x, x1 + X) \Leftarrow x1 + Filter1(x, X)$ **if** $x1 < x$
$Filter1(x, X)$ **otherwise** Rule RS2 and folding with 11.

Thus we have algorithm *S1* (*Quick Sort*)

```
S1  $Sort(X) \Leftarrow Sort1([X], X)$
    $Sort1(\Phi, \Phi) \Leftarrow \{\Phi\}$
    $Sort1(\{n\}, \{n\}) \Leftarrow \{\{\langle n\ x \rangle\}\}$
    $Sort1(N, X) \Leftarrow \{f1 \cup f2 \mid f1 \in Sort1(N^k, Y)$
                                  $f2 \in Sort1(N_k, X - Y)\}$
                         where $k = Card(Y)$
                            where $Y = Filter(X)$
    $Filter(\Phi) \Leftarrow \Phi$
    $Filter(x + X) \Leftarrow Filter1(x, X)$
    $Filter1(x, \Phi) \Leftarrow \Phi$
    $Filter1(x, x1 + X) \Leftarrow x1 + Filter1(x, X)$ if $x1 < x$
                               $Filter1(x, X)$ otherwise
```

Of course there are many more improvements that can be made to this algorithm, for instance the computation of $X - Y$ can be interwoven with the computation of $Y = Filter(X)$ but we hope that we have at least captured the "essence" of the algorithm.

5.2.3. *Selection Sort* from *P1*. The derivation of *Selection Sort* is straightforward. It can be derived "horizontally" from *S1* by always choosing Y to be a singleton. We choose to derive it from *P1* to push the decision points as high in the synthesis as possible. Thus we have

1. $$Perm1(N, X) \Leftarrow \bigcup_{Y \subset_1 X} \{f1 \cup f2 \mid f1 \in Perm1(N^1, Y),\ f2 \in Perm1(N_1, X - Y)\}$$
$$\Leftarrow \bigcup_{y \in X} \{f1 \cup f2 \mid f1 \in Perm1(\{first(N)\}, \{y\}),\ f2 \in Perm1(rest(N), X - \{y\})\}$$
$$\Leftarrow \bigcup_{y \in X} \{\langle first(N), y \rangle + f \mid f \in Perm1(rest(N), X - \{y\})\}.$$

Thus we have a permutation algorithm *P1'*

P1′ $Perm(X) \Leftarrow Perm1([X], X)$
$Perm1(\Phi, \Phi) \Leftarrow \{\Phi\}$
$Perm1(\{n\}, \{x\}) \Leftarrow \{\{\langle n\ x \rangle\}\}$
$Perm1(N, X) \Leftarrow \bigcup_{y \in X} \{\langle first(N), y \rangle + f \mid f \in Perm1(rest(N), X - \{y\})\}$

Again we define

2. $Sort(X) \Leftarrow Ordered(Perm(X))$
   $\Leftarrow Ordered(Perm1([X], X))$

and let

3. $Sort1(N, X) \Leftarrow Ordered\,(Perm1(N, X))$
4. $Sort1(\Phi, \Phi) \Leftarrow \{\Phi\}$ Unfolding using *P1′* and 5.2.2, 2.
5. $Sort(N, X) \Leftarrow Ordered(Perm(N, X))$.

Pushing the *Ordered* filter inside (6.3) gives us,

6. $Sort1(N, X) \Leftarrow \{\langle first(N), y \rangle + f \mid f \in Perm1(rest(N), X - \{y\})$
   $and\ Ord(f)\}$
   for some y **such that** $y \in X$ *and* $\forall x \in X - \{y\} \cdot y < x$
   $\Leftarrow \{\langle first(N), y \rangle + f \mid f \in Sort1(rest(N), X - \{y\})\}$
   for some y **such that** $y \in X$ *and* $\forall x \in X - \{y\} \cdot y < x$
   Folding with 5.2.2, 2 and 3.

We now define

7. $Least(X) \Leftarrow y$ **such that** $y \in X$ *and* $\forall x \in X - \{y\} \cdot y < x$.

Thus

8. $Sort1(N, X) \Leftarrow \{\langle first(N), y \rangle + f \mid f \in Sort1(rest(N), X - \{y\})\}$
   where $y = Least(X)$ Folding 6 with 7.
9. $Least(\{x\}) \Leftarrow x$ Instantiating 7 and reducing.
10. $Least(x1 + X)$
    $\Leftarrow x1$ **if** $\forall x \in X \cdot x1 < x$
    y **such that** $y \in X$ *and* $\forall x \in X - y \cdot y < x$
    otherwise
    Instantiating 7 and rule RM1.

But as $x1 < y1$ where $y1 \in X$ and $\forall x \in X - y1 \cdot y1 < x$ implies $\forall x \in X \cdot x1 < x$ we can rewrite this as

$\Leftarrow x1$ **if** $x1 < y$ **where** $y \in X$ *and* $\forall x \in X - y \cdot y < x$
y **such that** $y \in X$ *and* $\forall x \in X - y \cdot y < x$
otherwise

$\Leftarrow x1$ **if** $x1 < Least(X)$
$Least(X)$ **otherwise**
Folding with 7.

Thus we have algorithm *S2* (*Selection Sort*)

S2 $Sort(X) \Leftarrow Sort1([X], X)$
$Sort1(\Phi, \Phi) \Leftarrow \{\Phi\}$
$Sort1(N, X) \Leftarrow \{\langle first(N), y\rangle + f \mid f \in Sort1(rest(N), X - \{y\})\}$
where $y = Least(X)$
$Least(\{x\}) \Leftarrow x$
$Least(x1 + X) \Leftarrow x1$ **if** $x1 < Least(X)$
$Least(X)$ **otherwise**

5.3. *P2*, *Merge Sort* and *Insertion Sort* from *P*

5.3.1. *P2* from *P*. Repeating our definition from *P* we have

1. $Perm(X) \Leftarrow \{f \mid f \subseteq [X] \times X \text{ and } Bijective(f, [X], X)\}$

and we can immediately derive

2. $Perm(\Phi) \Leftarrow \{\Phi\}$ Instantiating 1 rules RC1, RS1.

and

3. $Perm(\{x\}) \Leftarrow \{\{\langle 1\ x\rangle\}\}$ Unfolding.

To set ourselves on the path to *Merge Sort* we choose to decompose *X* into two distinct non-empty sets getting

4. $Perm(X1 \cup X2) \Leftarrow \{f \mid f \subseteq [X1 \cup X2] \times (X1 \cup X2) \text{ and } Bijective(f, [X1 \cup X2], X1 \cup X2)\}.$

The insight required is that from the definition of *Perm* and *Bijective* we know that for any $f1 \in Perm(X1)$ and any $f2 \in Perm(X2)$ $Image(f1) = X1$ and $Image(f2) = X2$ thus we have $Perm(X1 \cup X2) = Perm(Image(f1) \cup Image(f2))$. Thus we could re-express the above and produce an algorithm of the form $Perm(X1, X2) = f(f1, f2)$ for any $f1 \in Perm(X1)$, $f2 \in Perm(X2)$ where *f* forgets all the structure built into *f1* and *f2* and computes *Perm* all over again from $Image(f1)$ and $Image(f2)$. To get an algorithm closer to the familiar one we define an *f* that computes a subset of *Perm*, viz. those permutations are attainable from *f1* and *f2* without disturbing the internal order of *f1* and *f2*. We will call this *f Merge*!

More exactly

5. $Merge(f1, f2) \Leftarrow \{f \mid f \subseteq [Image(f1) \cup Image(f2)] \times Image(f1) \cup Image(f2) \text{ and } Bijective(f, [Image(f1) \cup Image(f2)], Image(f1) \cup Image(f2)) \text{ and } Regular(f, f1) \text{ and } Regular(f, f2)\}$

where

6. $Regular(f, f1) \Leftarrow \forall x1, x2 \in Image(f1) \cdot f1^{-1}(x1) < f1^{-1}(x2) \Rightarrow f^{-1}(x1) < f^{-1}(x2).$

We shall see later in 6.4 where this restriction of *Perm* arises from.

To get $Perm(X1 \cup X2)$ it is now no longer sufficient to take any $f1 \in Perm(X1)$, $f2 \in Perm(X2)$ we need to take all such $f1$ and $f2$, i.e.

7. $$Perm(X1 \cup X2) \Leftarrow \bigcup_{\substack{f1 \in Perm(X1)\\ f2 \in Perm(X2)}} Merge(f1, f2).$$

We, of course, need to justify this equation but we will postpone its (informal) justification until we see where the *Regular* filter arises from.

We have to produce a recursion for *Merge*. We first generalise 5 defining

8. $$\begin{aligned} Merge1(f1, f2, N) \Leftarrow \{f \mid f \subseteq N \times Image(f1) \cup Image(f2) \\ \textit{and } Bijective(f, N, Image(f1) \cup Image(f2)) \\ \textit{and } Regular(f, f1) \\ \textit{and } Regular(f, f2)\}. \end{aligned}$$

Thus

9. $$\begin{aligned} Merge(f1, f2) &\Leftarrow Merge1(f1, f2, [Image(f1) \cup Image(f2)]) && \text{Folding 5.3.1, 5 with 5.3.1, 8} \\ &\Leftarrow Merge1(f1, f2, [f1 \cup f2]). \end{aligned}$$

We first synthesise the main recursion for *Merge1*.

10. $$\begin{aligned} & Merge1(\langle n1\ x1\rangle + f1, \langle n2\ x2\rangle + f2, n+N) \\ & \quad \Leftarrow \{f \mid f \subseteq n+N \times Image(\langle n1\ x1\rangle + f1) \cup Image(\langle n2\ x2\rangle + f2) \\ & \qquad \textit{and } Bijective(f, n+N, Image(\langle n1\ x1\rangle + f1) \\ & \qquad\qquad \cup Image(\langle n2\ x2\rangle + f2)) \\ & \qquad \textit{and } Regular(f, \langle n1\ x1\rangle + f1) \\ & \qquad \textit{and } Regular(f, \langle n2\ x2\rangle + f2)\} \quad \text{Instantiating 8.} \end{aligned}$$

The filter promotion in this case is more complicated as we have two filters, for details see 6.4, but eventually we can rewrite 10 as

$$\begin{aligned} & \Leftarrow \{\langle n\ x1\rangle + f1' \mid f1' \subseteq N \times Image(f1) \cup Image(\langle n2\ x2\rangle + f2) \\ & \qquad \textit{and } Bijective(f1', N, Image(f1) \cup Image(\langle n2\ x2\rangle + f2)) \\ & \qquad \textit{and } Regular(f1', f1) \\ & \qquad \textit{and } Regular(f1', \langle n2\ x2\rangle + f2)\} \\ & \cup \{\langle n\ x2\rangle + f2' \mid f2' \subseteq N \times Image(\langle n1\ x1\rangle + f1) \cup Image(f2) \\ & \qquad \textit{and } Bijective(f2', N, Image(\langle n1\ x1\rangle + f1) \cup Image(f2)) \\ & \qquad \textit{and } Regular(f2', \langle n1\ x1\rangle + f1) \\ & \qquad \textit{and } Regular(f2', f2)\} \\ & \Leftarrow \{\langle n\ x1\rangle + f1' \mid f1' \in Merge1(f1, \langle n2\ x2\rangle + f2, N)\} \\ & \cup \{\langle n\ x2\rangle + f2' \mid f2' \in Merge1(\langle n1\ x1\rangle + f1, f2, N)\} \quad \text{Folding with 8.} \end{aligned}$$

Now we have only the base cases of *Merge1* to do. We notice from equation 9 that when $Merge1(f1, f2, N)$ is first called we have $Card(N) = Card(f1) + Card(f2)$ and from equation 11 we see that at each recursive call one element is removed from N and one from either $f1$ or $f2$ so this relationship is maintained. So the base cases we need to examine are

12. $Merge1(\Phi, \Phi, \Phi) \Leftarrow \{\Phi\}$ Instantiating 8 and unfolding

and

$$\begin{array}{ll} 13.\ \mathit{Merge1}(\langle n1\ x1\rangle + f1, \Phi, n+N) \Leftarrow \{f \mid f \subseteq n+N \times \mathit{Image}(f1) \cup \mathit{Image}(\Phi) & \\ \qquad \text{and } \mathit{Bijective}(f, N, \mathit{Image}(f1) \cup \mathit{Image}(\Phi)) & \\ \qquad \text{and } \mathit{Regular}(f, \langle n1\ x1\rangle + f1) & \\ \qquad \text{and } \mathit{Regular}(f, \Phi)\} & \text{Instantiating 5.3.1, 8.} \end{array}$$

There is a corresponding case with $f1\ \Phi$. We are again able to produce a recursion for those cases but as the synthesis is quite straightforward we will not bother to present it. We get

$$\begin{array}{l} 14.\ \mathit{Merge1}(\langle n1\ x1\rangle + f1, \Phi, n+N) \\ \qquad \Leftarrow \{\langle n\ x1\rangle + f \mid f \in \mathit{Merge1}(f1, \Phi, N)\} \end{array}$$

and

$$\begin{array}{l} 15.\ \mathit{Merge1}(\Phi, \langle n2\ x2\rangle + f2, n+N) \\ \qquad \Leftarrow \{\langle n\ x2\rangle + f \mid f \in \mathit{Merge1}(\Phi, f2, N)\}. \end{array}$$

These two base cases just return the original "array" but shifted along so that it will fit on the end of the arrays produced by the main recursion.

Thus we have algorithm *P2*

$$\begin{array}{ll} \textbf{P2}\ \ \mathit{Perm}(\Phi) & \Leftarrow \{\Phi\} \\ \mathit{Perm}(\{x\}) & \Leftarrow \{\{\langle 1\ x\rangle\}\} \\ \mathit{Perm}(X1 \cup X2) \Leftarrow & \bigcup_{\substack{f1 \in Perm(X1) \\ f2 \in Perm(X2)}} \mathit{Merge}(f1, f2) \\ \mathit{Merge}(f1, f2) & \Leftarrow \mathit{Merge1}(f1, f2, [f1 \cup f2]) \\ \mathit{Merge1}(\Phi, \Phi, \Phi) \Leftarrow \{\Phi\} & \\ \mathit{Merge1}(\langle n1\ x1\rangle + f1, \Phi, n+N) & \\ \qquad \Leftarrow \{\langle n\ x1\rangle + f \mid f \in \mathit{Merge1}(f1, \Phi, N)\} & \\ \mathit{Merge1}(\Phi, \langle n2\ x2\rangle + f2, n+N) & \\ \qquad \Leftarrow \{\langle n\ x2\rangle + f \mid f \in \mathit{Merge1}(\Phi, f2, N)\} & \\ \mathit{Merge1}(\langle n1\ x1\rangle + f1, \langle n2\ x2\rangle + f2, n+N) & \\ \qquad \Leftarrow \{\langle n\ x1\rangle + f1' \mid f1' \in \mathit{Merge1}(f1, \langle n2\ x2\rangle + f2, N)\} & \\ \qquad \cup \{\langle n\ x2\rangle + f2' \mid f2' \in \mathit{Merge1}(\langle n1\ x1\rangle + f1, f2, N)\}. & \end{array}$$

5.3.2. *Merge Sort* from *P2*. Again we define

1. $\mathit{Sort}(X) \Leftarrow \mathit{Ordered}(\mathit{Perm}(X))$

where *Perm* is given by *P2* and *Ordered* by 5.2.2, 2. Thus we have immediately

$$\begin{array}{lll} 2.\ \mathit{Sort}(\Phi) & \Leftarrow \{\Phi\} & \text{Instantiating 1 and unfolding} \\ 3.\ \mathit{Sort}(\{x\}) & \Leftarrow \{\{\langle 1\ x\rangle\}\} & \text{Instantiating 1 and unfolding} \\ 4.\ \mathit{Sort}(X1 \cup X2) \Leftarrow & \bigcup_{\substack{f1 \in Perm(X1) \\ f2 \in Perm(X2)}} \mathit{Ordered}(\mathit{Merge}(f1, f2)) & \text{Instantiating 1, unfolding using 5.3.1, 7 and taking } \mathit{Ordered} \text{ inside the union.} \end{array}$$

Examination of the filters (6.5) enables us to rewrite this as

$$\Leftarrow \bigcup_{\substack{f1 \in Ordered(Perm(X1)) \\ f2 \in Ordered(Perm(X2))}} \mathit{Ordered}(\mathit{Merge}(f1, f2))$$

Thus we have

5. $Sort(X1 \cup X2) \Leftarrow \bigcup_{\substack{f1 \in Sort(X1)\\ f2 \in Sort(X2)}} Ordered(Merge(f1, f2))$ Folding with 1.

We now look at $Ordered(Merge(f1, f2))$, which is $Ordered(Merge1(f1, f2, [f1 \cup f2]))$ so we define

6. $MergeS(f1, f2, N) \Leftarrow Ordered(Merge1(f1, f2, N))$

and rewrite 5 as

7. $Sort(X1 \cup X2) \Leftarrow \bigcup_{\substack{f1 \in Sort(X1)\\ f2 \in Sort(X2)}} MergeS(f1, f2, [f1 \cup f2])$ Unfolding 5 with 5.3.1, 9 Folding with 6.

Immediately we have

8. $MergeS(\Phi, \Phi, \Phi) \Leftarrow \{\Phi\}$ Instantiating 6 and unfolding using 5.3.1, 2 and 5.3.2, 2.

We now look at

9. $MergeS(\langle n1\ x1\rangle + f1, \langle n2\ x2\rangle + f2, n + N)$
$\Leftarrow Ordered(Merge1(\langle n1\ x1\rangle + f1, \langle n2\ x2\rangle + f2, n + N))$
Instantiating 6.

Promoting filters (6.6) gets us to

10. $MergeS(\langle n1\ x1\rangle + f1, \langle n2\ x2\rangle + f2, n + N)$

$$\begin{aligned}
&\Leftarrow \{\langle n\ x1\rangle + f1' \mid f1' \in Merge1(f1, \langle n2\ x2\rangle + f2, N) \text{ and } Ord(f1')\}\\
&\qquad \textbf{if } x1 < x2\\
&\quad\ \{\langle n\ x2\rangle + f2' \mid f2' \in Merge1(\langle n1\ x1\rangle + f1, f2, N) \text{ and } Ord(f2')\}\\
&\qquad \textbf{otherwise}\\
&\Leftarrow \{\langle n\ x1\rangle + f1' \mid f1' \in MergeS(f1, \langle n2\ x2\rangle + f2, N)\}\\
&\qquad \textbf{if } x1 < x2\\
&\quad\ \{\langle n\ x2\rangle + f2' \mid f2' \in MergeS(\langle n1\ x1\rangle + f1, f2, N)\}\\
&\qquad \textbf{otherwise}
\end{aligned}$$

Folding with 5.2.2, 2 and 6.

Finally we have the other base cases to do. The synthesis of the recursions are straightforward and we omit details. In fact as $f1$ and $f2$ are ordered and $Merge1(f1, \Phi, N)$, say, does not rearrange $f1$ we could just continue to use $Merge1$ for these base cases, but we find it neater to write

11. $MergeS(\langle n1\ x1\rangle + f1, \Phi, n + N)$
$\Leftarrow \{\langle n\ x1\rangle + f1' \mid f1' \in MergeS(f1, \Phi, N)\}$

and

12. $MergeS(\Phi, \langle n2\ x2\rangle + f2, n + N)$
$\Leftarrow \{\langle n\ x2\rangle + f2' \mid f2' \in MergeS(\Phi, f2, N)\}$.

Thus we have algorithm *S3* (*Merge Sort*)

S3 $Sort(\Phi) \Leftarrow \{\Phi\}$

$Sort(\{x\}) \Leftarrow \{\{\langle 1\ x\rangle\}\}$

$Sort(X1 \cup X2) \Leftarrow \bigcup_{\substack{f1 \in Sort(X1)\\ f2 \in Sort(X2)}} MergeS(f1, f2, [f1 \cup f2])$

$MergeS(\Phi, \Phi, \Phi) \Leftarrow \{\Phi\}$

$MergeS(\langle n1\ x1\rangle + f1, \Phi, n+N)$
$\quad \Leftarrow \{\langle n\ x1\rangle + f1' \mid f1' \in MergeS(f1, \Phi, N)\}$

$MergeS(\Phi, \langle n2\ x2\rangle + f2, n+N)$
$\quad \Leftarrow \{\langle n\ x2\rangle + f2' \mid f2' \in MergeS(\Phi, f2, N)\}$

$MergeS(\langle n1\ x1\rangle + f1, \langle n2\ x2\rangle + f2, n+N)$
$\quad \Leftarrow \{\langle n\ x1\rangle + f1' \mid f1' \in MergeS(f1, \langle n2\ x2\rangle + f2, N)\}$ **if** $x1 < x2$
$\quad \{\langle n\ x2\rangle + f2' \mid f2' \in MergeS(\langle n1\ x1\rangle + f1, f2, N)\}$ **otherwise**.

5.3.3. *Insertion Sort* from *P2*. If in algorithm *P2* instead of decomposing X to $X1 \cup X2$ we decompose it to $x + X$ the equation for the main recursion becomes

1. $Perm(x+X) \Leftarrow \bigcup_{f2 \in Perm(X)} Mergel(\{\langle 1\ x\rangle\}, f2, [\{\langle 1\ x\rangle\} \cup f2])$ Instantiating 5.3.1, 11 and unfolding using 5.3.1, 9.

Thus we specialise *Mergel* getting new equations

2. $Mergel(\{\langle 1\ x\rangle\}, \Phi, n+\Phi) \Leftarrow \{\{n\ x\}\}$ Unfolding using 5.3.1, 14 and 5.3.1, 12

and

3. $Mergel(\{\langle 1\ x\rangle\}, \langle n2\ x2\rangle + f2, n+N)$
$\quad \Leftarrow \{\langle n\ x\rangle + f1' \mid f1' \in Mergel(\Phi, \langle n2\ x2\rangle + f2, N)\}$
$\quad \cup \{\langle n\ x2\rangle + f2' \mid f2' \in Mergel(\{\langle 1\ x\rangle\}, f2, N)\}$
Unfolding using 5.3.1, 11.

We call this modified Perm algorithm *P2′*, note that equation 5.3.1, 3 is no longer required.

P2′ $Perm(\Phi) \Leftarrow \{\Phi\}$

$Perm(x+X) \Leftarrow \bigcup_{f \in Perm(X)} Mergel(\{\langle 1\ x\rangle\}, f, [\{\langle 1\ x\rangle\} \cup f])$

$Mergel(\Phi, \Phi, \Phi) \Leftarrow \{\Phi\}$

$Mergel(\{\langle 1\ x\rangle\}, \Phi, n+\Phi)$
$\quad \Leftarrow \{\{\langle n\ x\rangle\}\}$

$Mergel(\Phi, \langle n2\ x2\rangle + f2, n+N)$
$\quad \Leftarrow \{\langle n\ x2\rangle + f \mid f \in Mergel(\Phi, f2, N)\}$

$Mergel(\{\langle 1\ x\rangle\}, \langle n2\ x2\rangle + f2, n+N)$
$\quad \Leftarrow \{\langle n\ x\rangle + f1' \mid f1' \in Mergel(\Phi, \langle n2\ x2\rangle + f2, N)\}$
$\quad \cup \{\langle n\ x2\rangle + f2' \mid f2' \in Mergel(\{\langle 1\ x\rangle\}, f2, N)\}$.

To get *Insertion Sort* we as usual define

4. $Sort(X) \Leftarrow Ordered(Perm(X))$

where *Perm* is now defined by *P2'*. We will restrict ourselves to outlining the synthesis of the main recursion, thus instantiating 4 we get

5. $Sort(x+X) \Leftarrow \bigcup_{f \in Perm(X)} Ordered(Merge1(\{\langle 1\ x\rangle\}, f, [\{\langle 1\ x\rangle\} \cup f]))$ Unfolding using 1 and taking *Ordered* inside the union.

As with *Merge Sort* we can force a fold with 4 because examining the filters allows us to rewrite 5 as

6. $Sort(x+X) \Leftarrow \bigcup_{f \in Ordered(Perm(X))} Ordered(Merge1(\{\langle 1\ x\rangle\}, f, [\{\langle 1\ x\rangle\} \cup f]))$
$\Leftarrow \bigcup_{f \in Sort(X)} Ordered(Merge1(\{\langle 1\ x\rangle\}, f, [\{\langle 1\ x\rangle\} \cup f]))$ Folding with 4.

We now look at

7. $MergeS(\{\langle 1\ x\rangle\}, f, N) \Leftarrow Ordered(Merge1(\{\langle 1\ x\rangle\}, f, N))$

and in particular

8. $MergeS(\{\langle 1\ x\rangle\}, \langle n2\ x2\rangle + f2, n+N)$
$\Leftarrow Ordered(Merge1(\{\langle 1\ x\rangle\}, \langle n2\ x2\rangle + f2, n+N)$

Promoting (6.7) gets us to

9. $MergeS(\{\langle 1\ x\rangle\}, \langle n2\ x2\rangle + f2, n+N)$
$\Leftarrow \{\langle n\ x\rangle + f1' \mid f1' \in Merge1(\Phi, \langle n2\ x2\rangle + f2, N)$ *and* $Ord(f1')\}$ **if** $x < x2$
$\{\langle n\ x2\rangle + f2' \mid f2' \in Merge1(\{\langle 1\ x\rangle\}, f2, N)$ *and* $Ord(f2')\}$ **otherwise**
$\Leftarrow \{\langle n\ x\rangle + f1' \mid f1' \in MergeS(\Phi, \langle n2\ x2\rangle + f2, N)\}$ **if** $x < x2$
$\{\langle n\ x2\rangle + f2' \mid f2' \in MergeS(\{\langle 1\ x\rangle\}, f2, N)\}$ **otherwise**
Folding with 5.2.2, 2 and 7.

The base cases go through smoothly and we have algorithm *S4*, *Insertion Sort*

```
S4  $Sort(\Phi) \Leftarrow \{\Phi\}$
    $Sort(x+X) \Leftarrow \bigcup_{f \in Sort(X)} MergeS(\{\langle 1\ x\rangle\}, f, [\{\langle 1\ x\rangle\} \cup f])$
    $MergeS(\Phi, \Phi, \Phi) \Leftarrow \{\Phi\}$
    $MergeS(\{\langle 1\ x\rangle\}, \Phi, n+\Phi)$
            $\Leftarrow \{\{\langle n\ x\rangle\}\}$
    $MergeS(\Phi, \langle n2\ x2\rangle + f2, n+N)$
            $\Leftarrow \{\langle n\ x2\rangle + f2' \mid f2' \in MergeS(\Phi, f2, N)\}$
    $MergeS(\{\langle 1\ x\rangle\}, \langle n2\ x2\rangle + f2, n+N)$
            $\Leftarrow \{\langle n\ x\rangle + f1' \mid f1' \in MergeS(\Phi, \langle n2\ x2\rangle + f2, N)\}$
                    if $x < x2$
             $\{\langle n\ x2\rangle + f2' \mid f2' \in MergeS(\{\langle 1\ x\rangle\}, f2, N)\}$
                    otherwise.
```

5.4. Synthesis of *P3*, *Exchange Sort* and *Bubble Sort* from *P*

In this section we consider the class of sorts that operate by swapping adjacent elements of the array whenever local inversions occur. We synthesise versions of the two best known algorithms in this class, *Exchange Sort* and *Bubble Sort*.

5.4.1. *P3* from *P*. In the previous two sections we took our definition of the set of all permutations of a set to be

1. $Perm(X) \Leftarrow \{f \mid f \subseteq [X] \times X \text{ and } Bijective(f, [X], X)\}$
2. $Bijective(f, Y, X) \Leftarrow f$ *is a total one to one function from Y onto X.*

In this section however we take a slightly different starting point. *Perm* now takes a function, representing an array, as argument and again produces the set of all permutations i.e.

3. $Perm(f) \Leftarrow \{f_1 \mid f_1 \subseteq Domain(f) \times Image(f)$
   $\quad\quad and\ Bijective(f_1, Domain(f), Image(f))\}$

where *Bijective* is as before.

The key to producing a recursion to compute *Perm*(f) lies in first synthesising another function *Perm1*(f) which will compute a subset of *Perm*(f). Informally *Perm1*(f) computes all those permutations of f that can be achieved by going along f and either swapping or not swapping adjacent elements. Elements can therefore move to the right freely but at most one place to the left. Thus we can characterise *Perm1*(f) exactly.

4. $Perm1(f) \Leftarrow \{f_1 \mid f_1 \subseteq Domain(f) \times Image(f)$
   $\quad\quad and\ Bijective(f_1, Domain(f), Image(f))$
   $\quad\quad and\ Close(f_1, f)\}$
5. $Close(f_1, f) \Leftarrow \forall \langle n\ x \rangle \in f_1 \cdot 1 \leq f^{-1}(x) \leq n+1.$

We will see in (6.8) where this definition of *Perm1* arises.
We have

6. $Perm1(\Phi) \Leftarrow \{\Phi\}$ Instantiating 4 and unfolding.
7. $Perm1(\langle n\ x \rangle + \Phi) \Leftarrow \{\langle n\ x \rangle + \Phi\}$ Instantiating 4 and unfolding.
8. $Perm1(\langle n_1\ x_1 \rangle + \langle n_2\ x_2 \rangle + f)$

$$\begin{aligned}
&\Leftarrow \{f_1 \mid f_1 \subseteq Domain(\langle n_1\ x_1 \rangle + \langle n_2\ x_2 \rangle + f) \times Image(\langle n_1\ x_1 \rangle + \langle n_2\ x_2 \rangle + f)\\
&\quad and\ Bijective(f_1, Domain(\langle n_1\ x_1 \rangle + \langle n_2\ x_2 \rangle + f),\\
&\quad\quad Image(\langle n_1\ x_1 \rangle + \langle n_2\ x_2 \rangle + f))\\
&\quad and\ Close(f_1, \langle n_1\ x_1 \rangle + \langle n_2\ x_2 \rangle + f)\}.
\end{aligned}$$

Promoting these two filters (6.8) gets us to ($N = n_2 + Domain(f)$).

10. $Perm1(\langle n_1\ x_1 \rangle + \langle n_2\ x_2 \rangle + f)$

$$\begin{aligned}
&\Leftarrow \{\langle n_1\ x_1 \rangle + f_1 \mid f_1 \subseteq N \times x_2 + X\\
&\quad\quad and\ Bijective(f_1, N, x_2 + X)\\
&\quad\quad and\ Close(f_1, \langle n_2\ x_2 \rangle + f)\}\\
&\cup \{\langle n_1\ x_2 \rangle + f_2 \mid f_2 \subseteq N \times x_1 + X\\
&\quad\quad and\ Bijective(f_2, N, x_1 + X)\\
&\quad\quad and\ Close(f_2, \langle n_2\ x_1 \rangle + f)\}
\end{aligned}$$

$$\begin{aligned}
&\Leftarrow \{\langle n_1\ x_1\rangle + f_1 \mid f_1 \subseteq \mathit{Domain}(\langle n_2\ x_2\rangle + f) \times \mathit{Image}(\langle n_2\ x_2\rangle + f)\\
&\qquad \mathit{and}\ \mathit{Bijective}(f_1, \mathit{Domain}(\langle n_2\ x_2\rangle + f),\\
&\qquad\qquad \mathit{Image}(\langle n_1\ x_2\rangle + f))\\
&\qquad \mathit{and}\ \mathit{Close}(f_1, \langle n_2\ x_2\rangle + f)\}\\
&\cup \{\langle n_1\ x_2\rangle + f_2 \mid f_2 \subseteq \mathit{Domain}(\langle n_2\ x_1\rangle + f) \times \mathit{Image}(\langle n_2\ x_1\rangle + f)\\
&\qquad \mathit{and}\ \mathit{Bijective}(f_2, \mathit{Domain}(\langle n_2\ x_1\rangle + f),\\
&\qquad\qquad \mathit{Image}(\langle n_2\ x_1\rangle + f))\\
&\qquad \mathit{and}\ \mathit{Close}(f_1, \langle n_2\ x_1\rangle + f)\}\\
&\qquad \text{Using the definitions of } \mathit{Domain} \text{ and } \mathit{Image}\\
&\Leftarrow \{\langle n_1\ x_1\rangle + f_1 \mid f_1 \in \mathit{Perm1}(\langle n_2\ x_2\rangle + f)\}\\
&\cup \{\langle n_1\ x_2\rangle + f_2 \mid f_2 \in \mathit{Perm1}(\langle n_2\ x_1\rangle + f)\}\\
&\qquad \text{Folding with 4.}
\end{aligned}$$

How do we get *Perm* from *Perm1*? A moment spent considering the target sorting algorithms gives us a clue. They both operate by closure repeatedly calling the swapping subroutine until the ordered list is achieved.

Thus we first define *Perm2* the extension of *Perm1* over a set.

11. $\mathit{Perm2}(S) \Leftarrow \bigcup_{f\in S} \mathit{Perm1}(f).$

The crucial lemma we need to establish is that the fixed point of *Perm2* is equal to *Perm*, that is $\mathit{Perm2}(S) = S \Rightarrow S = \mathit{Perm}(f)$ for any $f \in S$, but this is clear from the definitions of *Perm*, *Perm1* and *Perm2*, 3, 4, 5 and 11. Thus we can modify 11 to

```
12. $Perm2(S) \Leftarrow$ if $\bigcup_{f\in S} Perm1(f) = S$
                        then $S$
                        else $Perm2(\bigcup_{f\in S} Perm1(f))$
```

and write

13. $\mathit{Perm}(f) \Leftarrow \mathit{Perm2}(\{f\})$

and we have algorithm *P3*

```
P3  $Perm(f) \Leftarrow Perm2(\{f\})$
    $Perm2(S) \Leftarrow$ if $u = S$ then $S$
                        else $Perm2(u)$
                          where $u = \bigcup_{f\in S} Perm1(f)$
    $Perm1(\Phi) \Leftarrow \{\Phi\}$
    $Perm1(\{\langle n\ x\rangle\}) \Leftarrow \{\{\langle n\ x\rangle\}\}$
    $Perm1(\langle n_1\ x_1\rangle + \langle n_2\ x_2\rangle + f)$
              $\Leftarrow \{\langle n_1\ x_1\rangle + f_1 \mid f_1 \in Perm1(\langle n_2\ x_2\rangle + f)\}$
              $\cup \{\langle n_1\ x_2\rangle + f_2 \mid f_2 \in Perm1(\langle n_2\ x_1\rangle + f)\}.$
```

5.4.2. *Exchange Sort* from *P3*. We define *Sort* as usual (only this time it takes an array as argument)

1. $\mathit{Sort}(f) \Leftarrow \mathit{Ordered}(\mathit{Perm}(f))$

where ordered is given by 5.2.2, 2 and *Perm* by *P3*.

To produce *Exchange Sort* and *Bubble Sort* we constrain *Perm1* so that only swaps that move towards the ordered list are made. For *Exchange Sort* we constrain *Perm1* so that it removes as many inversions as possible per call, for *Bubble Sort* we constrain it so that one inversion is removed per call. More exactly for *Exchange Sort* we have

2. $Sort11(f) \Leftarrow Best(Perm1(f))$
3. $Best(S) \Leftarrow f$ **such that** $f \in S$ $\quad and\ \forall f1 \in S - \{f\} \cdot Noofinversions(f) \leqq Noofinversions(f1)$
4. $Noofinversions(f) \Leftarrow Card(\{x1 \mid \langle n1\ x1\rangle \in f, \langle n2\ x2\rangle \in f,\ and\ n1 < n2\ and\ x1 > x2\})$.

Thus we have

5. $Sort11(\langle n1\ x1\rangle + \langle n2\ x2\rangle + f)$

$$\begin{aligned}
&\Leftarrow Best(\{\langle n1\ x1\rangle + f1 \mid f1 \in Perm1(\langle n2\ x2\rangle + f)\} \\
&\qquad \cup \{\langle n1\ x2\rangle + f2 \mid f2 \in Perm1(\langle n2\ x1\rangle + f)\}) \\
&\Leftarrow Best(\{\langle n1\ x1\rangle + f1 \mid f1 \in Perm1(\langle n2\ x2\rangle + f)\}) \\
&\qquad\qquad \textbf{if } x1 < x2 \\
&\quad Best(\{\langle n1\ x2\rangle + f2 \mid f2 \in Perm1(\langle n2\ x1\rangle + f)\}) \\
&\qquad\qquad \textbf{otherwise} \\
&\Leftarrow \langle n1\ x1\rangle + Best(Perm1(\langle n2\ x2\rangle + f)) \textbf{ if } x1 < x2 \\
&\quad \langle n1\ x2\rangle + Best(Perm1(\langle n2\ x1\rangle + f)) \textbf{ otherwise} \\
&\Leftarrow \langle n1\ x1\rangle + Sort11(\langle n2\ x2\rangle + f) \textbf{ if } x1 < x2 \\
&\quad \langle n1\ x2\rangle + Sort11(\langle n2\ x1\rangle + f) \textbf{ otherwise}.
\end{aligned}$$

When $Sort11(f) = f$, we have $Ord(f)$ thus we can write

```
6. $Sort(f) \Leftarrow$ if $u = f$
        then $f$
        else $Sort(u)$
            where $u = Sort11(f)$.
```

We now have a recursive sort program but we are not quite finished. In an *Exchange Sort* the termination test is interwoven with the main recursion. Here we still have it separate as the rest $u = f$ requires an iteration along the array. We will only outline this interweaving process and assume that equality between arrays or functions is computed by $=_f$ whose main recursion is given by

7. $\langle n\ x\rangle + f1 =_f \langle m\ y\rangle + f2$
$$\Leftarrow n = m\ and\ y = x\ and\ f1 =_f f2.$$

Thus we define

8. $Exchange(f) \Leftarrow \langle f =_f Sort11(f), Sort11(f)\rangle$

and for the main recursion we have

9. $Exchange(\langle n\ x\rangle + \langle m\ y\rangle + f)$

$$\begin{aligned}
&\Leftarrow \langle n = n\ and\ x = x\ and\ \langle m\ y\rangle + f =_f Sort11(\langle m\ y\rangle + f), \\
&\qquad n\ x\rangle + Sort11(\langle m\ y\rangle + f)\rangle \quad \textbf{if } x < y \\
&\quad \langle n = n\ and\ x = y\ and\ \langle m\ y\rangle + f =_f Sort11(\langle m\ x\rangle + f), \\
&\qquad \langle n\ y\rangle + Sort11(\langle m\ x\rangle + f)\rangle \quad \textbf{otherwise}.
\end{aligned}$$

Unfolding using 5, rearranging the conditional, and unfolding using 7

```
⇐ $\langle \mathit{True}\ and\ \langle m\ y\rangle + f =_f \mathit{Sort11}(\langle m\ y\rangle + f),$
        $\langle n\ x\rangle + \mathit{Sort11}(\langle m\ y\rangle + f)\rangle$  if $x<y$
  $\langle \mathit{False}\ and\ \langle m\ y\rangle + f =_f \mathit{Sort11}(\langle m\ x\rangle + f),$
        $\langle n\ y\rangle + \mathit{Sort11}(\langle m x\rangle + f)\rangle$  otherwise
⇐ $\mathit{True}\ and\ u, \langle n\ x\rangle + v\rangle$
      where $\langle u, v\rangle = \langle\langle m\ y\rangle + f =_f \mathit{Sort11}(\langle m\ y\rangle + f),$
            $\mathit{Sort11}(\langle m\ y\rangle + f)\rangle$  if $x<y$
  $\langle \mathit{False}, \langle n\ y\rangle + v\rangle\rangle$
      where $\langle u, v\rangle = \langle\langle m\ x\rangle + f =_f \mathit{Sort11}(\langle m\ x\rangle + f),$
            $\mathit{Sort11}(\langle m x\rangle + f)\rangle$  otherwise.
```

Abstracting. Notice that *u* is not used in the second abstraction and we use this to achieve a form that will fold

```
⇐ $\langle u, \langle n\ x\rangle + v\rangle$
      where $\langle u, v\rangle = \mathit{Exchange}(\langle m\ y\rangle + f)$ if $x<y$
  $\langle \mathit{False}, \langle n\ y\rangle + v\rangle\rangle$
      where $\langle u, v\rangle = \mathit{Exchange}(\langle m\ x\rangle + f)$ otherwise
        Folding with 8.
```

Finally we tie *Sort* up with *Exchange*

```
10. $\mathit{Sort}(f)$ ⇐ if $u$ then $f$
                 else $\mathit{Sort}(v)$
                     where $\langle u, v\rangle = \langle f =_f \mathit{Sort11}(f), \mathit{Sort11}(f)\rangle$
                           Abstracting 6
    ⇐ if $u$ then $f$
         else $\mathit{Sort}(v)$
             where $\langle u, v\rangle = \mathit{Exchange}(f)$
                   Folding with 8.
```

We will not bother detailing the synthesis of the base cases and present algorithm *S5* (*Exchange Sort*)

```
S5  $\mathit{Sort}(f)$ ⇐ if $u$ then $f$
                 else $\mathit{Sort}(v)$
                     where $\langle u, v\rangle = \mathit{Exchange}(f)$
    $\mathit{Exchange}(\Phi) \Leftarrow \langle \mathit{True}, \Phi\rangle$
    $\mathit{Exchange}(\{\langle n\ x\rangle\}) \Leftarrow \langle \mathit{True}, \{\langle n\ x\rangle\}\rangle$
    $\mathit{Exchange}(\langle n\ x\rangle + \langle m\ y\rangle + f)$
                 ⇐ $\langle u, \langle n\ x\rangle + v\rangle$
                       where $\langle u, v\rangle = \mathit{Exchange}(\langle m\ y\rangle + f)$  if $x<y$
                   $\langle \mathit{False}, \langle n\ y\rangle + v\rangle$
                       where $\langle u, v\rangle = \mathit{Exchange}(\langle m\ x\rangle + f)$  otherwise.
```

5.4.3. *Bubble Sort* from P3. As mentioned in the previous section *Bubble Sort* is derived by choosing from *Perm1* the permutation with just one inversion removed (if there is one).

```
1. $Sort12(f) \Leftarrow Oneremoved(Perm1(f), f)$
2. $Oneremoved(S, f) \Leftarrow f1 \cdot$ such that $f1 \in S$
                          $and\ Noofinversions(f1) + 1 = Noofinversions(f)$
                          if $such\ an\ f1\ exists$
                       $f$ otherwise.
```

This produces a recursion

```
3. $Sort12(\langle n1\ x1\rangle + \langle n2\ x2\rangle + f)$
          $\Leftarrow \langle n1\ x1\rangle + Sort12(\langle n2\ x2\rangle + f)$ if $x1 < x2$
           $\langle n1\ x2\rangle + \langle n2\ x1\rangle + f$        otherwise.
```

Similarly we define

4. $Bubble(f) \Leftarrow \langle f =_f Sort12(f), Sort12(f)\rangle$

and the main recursion is

```
5. $Bubble(\langle n\ x\rangle + \langle m\ y\rangle + f)$
          $\Leftarrow \langle u, \langle n\ x\rangle + v\rangle$
             where $\langle u, v\rangle = \langle\langle m\ y\rangle + f =_f Sort12(\langle m\ y\rangle + f),$
                                          $Sort12(\langle m\ y\rangle + f)\rangle$ if $x < y$
           $\langle False, \langle n\ y\rangle + \langle m\ x\rangle + f\rangle\rangle$                  otherwise
                 Unfolding using 3, rearranging the conditional,
                 unfolding using 5.4.2, 7, simplifying the conjunction
                 and abstracting.
          $\Leftarrow \langle u, \langle n\ x\rangle + v\rangle$
             where $\langle u, v\rangle = Bubble(\langle m\ y\rangle + f)$ if $x < y$
           $\langle False, \langle n\ y\rangle + \langle m\ x\rangle + f\rangle$        otherwise
                 Folding with 4
```

and our version of *Bubble Sort* is algorithm *S6*

```
S6  $Sort(f) \Leftarrow$ if $u$ then $f$
                  else $Sort(v)$
                     where $\langle u, v\rangle = Bubble(f)$
    $Bubble(\Phi) \Leftarrow \langle True, \Phi\rangle$
    $Bubble(\{\langle n\ x\rangle\}) \Leftarrow \langle True, \{\langle n\ x\rangle\}\rangle$
    $Bubble(\langle n\ x\rangle + \langle m\ y\rangle + f)$
                  $\Leftarrow \langle u, \langle n\ x\rangle + v\rangle$
                     where $\langle u, v\rangle = Bubble(\langle m\ y\rangle + f)$ if $x < y$
                     $\langle False, \langle n\ y\rangle + \langle m\ x\rangle + f\rangle$ otherwise.
```

6. Lower Level Details

6.1. We have

$$Perm1(N, X) \Leftarrow \{f \mid f \subseteq \bigcup_{Y \subset_k X} (N^k \times Y) \cup (N_k \times (X - Y)) \ and\ Bijective(f, N, X)\}$$

$$\Leftarrow \{\cup f \mid f \subseteq (N^k \times Y) \cup (N_k \times (X - Y)), Y \subset_k X \ and\ Bijective(\cup f, N, X)\} \quad \text{Rule RS3.}$$

Consider any two subsets $Y1, Y2 \subset_k X$ such that $Y1 \neq Y2$ and any $f1, f2$ $f1 \subseteq (N^k \times Y1) \cup (N_k \times (X - Y1))$ and $f2 \subseteq (N^k \times Y2) \cup (N_k \times (X - Y2))$. An element $x1$ such that $x1 \in Y1$, $x1 \notin Y2$ is going to be mapped onto by an element of N^k in $f1$ and by an element of N_k in $f2$, therefore $f1 \cup f2$ cannot pass the bijective filter. Thus we can rewrite the above as

$$\begin{aligned}
&\Leftarrow \bigcup_{Y \subset_k X} \{f \mid f \subseteq (N^k \times Y) \cup (N_k \times (X - Y)) \\
&\qquad\qquad\qquad \textit{and } Bijective(f, N, X)\} \\
&\Leftarrow \bigcup_{Y \subset_k X} \{f1 \cup f2 \mid f1 \subseteq (N^k \times Y) \\
&\qquad\qquad f2 \subseteq (N_k \times (X - Y)) \\
&\qquad\qquad\qquad \textit{and } Bijective(f1 \cup f2, N, X)\} \quad \text{Rule RS3.}
\end{aligned}$$

To achieve a fold and produce a recursion for *Perm1* we need to decompose $Bijective(f1 \cup f2, N, X)$. But we know that the union of two functions on disjoint domains and ranges will be bijective if and only if the two functions are bijective ($N = N^k \cup N_k$). Thus we have

$$\begin{aligned}
Bijective(f1 \cup f2, N, X) \Leftrightarrow\ & Bijective(f1, N^k, Y) \\
& \textit{and } Bijective(f2, N_k, X - Y) \\
& \textit{for any } Y \subseteq X.
\end{aligned}$$

This allows us to write the above as

$$\begin{aligned}
Perm1(N, X) \Leftarrow \bigcup_{Y \subset_k X} \{f1 \cup f2 \mid\ & f1 \subseteq N^k \times Y \textit{ and } Bijective(f1, N^k, Y) \\
& f2 \subseteq N_k \times (X - Y) \textit{ and } Bijective(f2, N_k, X - Y)\}.
\end{aligned}$$

6.2. We have

$$\begin{aligned}
Sort1(N, X) \Leftarrow\ & Ordered\Big(\bigcup_{Y \subset_k X} \{f1 \cup f2 \mid f1 \in Perm1(N^k, Y) \\
&\qquad\qquad f2 \in Perm1(N_k, X - Y)\}\Big) \\
\Leftarrow\ & \bigcup_{Y \subset_k X} \{f1 \cup f2 \mid f1 \in Perm1(N^k, Y) \\
&\qquad\qquad f2 \in Perm1(N_k, X - Y) \\
&\qquad\qquad\quad \textit{and } Ord(f1 \cup f2)\}
\end{aligned}$$

Using the fact that $Ordered(X1 \cup X2) = Ordered(X1) \cup Ordered(X2)$ and unfolding.

In order to force a fold with 5.2.2, 5 we need to decompose $Ord(f1 \cup f2)$. We note that

(i) $\forall n1 \in N^k,\ n2 \in N_k \cdot n1 < n2$

(ii) $\forall f1 \in Perm1(N^k, Y),\ f2 \in Perm1(N_k, X - Y).$

$$\begin{aligned}
&Image(f1) = Y,\ Image(f2) = X - Y \\
&Domain(f1) = N^k,\ Domain(f2) = N_k.
\end{aligned}$$

Therefore unless $\forall y \in Y, x \in X - Y \cdot y < x$ there will be a $\langle n1\ y1 \rangle \in f1$, $\langle n2\ y2 \rangle \in f2$ such that $n1 < n2$ and $y1 > y2$ i.e. *not* $Ord(f1 \cup f2)$. Given such a Y we have $Ord(f1 \cup f2) \Leftrightarrow Ord(f1)$ *and* $Ord(f2)$. We will write the condition $\forall y \in Y, x \in X - Y \cdot y < x$ *as* $Y < X - Y$ and rewrite the above as

$$\begin{aligned}
\Leftarrow \bigcup_{\substack{Y \subset_k X \\ Y < X - Y}} \{f1 \cup f2 \mid\ & f1 \in Perm1(N^k, Y) \textit{ and } Ord(f1) \\
& f2 \in Perm1(N_k, X - Y) \textit{ and } Ord(f2)\}.
\end{aligned}$$

6.3. We start with

$$\begin{aligned}
&Sort(N, X) \Leftarrow Ordered(Perm(N, X))\\
&\qquad \Leftarrow \bigcup_{y \in X} \{\langle first(N), y\rangle + f \mid f \in Perm1(rest(N), X - \{y\})\\
&\qquad\qquad\qquad \text{and } Ord(\langle first(N), y\rangle + f)\}
\end{aligned}$$

Unfolding using *P1* and 5.2.2, 2.

As we know that $\forall n \in rest(N), first(N) < n$, by an argument similar to the one used in section 6.2 we can see that unless y is chosen such that $\forall x \in X - \{y\} \cdot y < x$ all components of the union will be Φ. For such a y we also have

$$Ord(\langle first(N), y\rangle + f) \Leftrightarrow Ord(f) \qquad (\text{as } f \in Perm1(rest(N), X - \{y\})$$

so we have

$$\begin{aligned}
&Sort1(N, X) \Leftarrow \{\langle first(N), y\rangle + f \mid f \in Perm1(rest(N), X - \{y\}) \text{ and } Ord(f)\}\\
&\qquad \textbf{for some } y \textbf{ such that } y \in X \text{ and } \forall x \in X - \{y\} \cdot y < x.
\end{aligned}$$

6.4. We have

$$\begin{aligned}
&Merge1(\langle n1\ x1\rangle + f1, \langle n2\ x2\rangle + f2, n + N)\\
&\qquad \Leftarrow \{f \mid f \subseteq n + N \times Image(\langle n1\ x1\rangle + f1) \cup Image(\langle n2\ x2\rangle + f2)\\
&\qquad\qquad \text{and } Bijective(f, n + N, Image(\langle n1\ x1\rangle + f1)\\
&\qquad\qquad\qquad \cup Image(\langle n2\ x2\rangle + f2))\\
&\qquad\qquad \text{and } Regular(f, \langle n1\ x1\rangle + f1)\\
&\qquad\qquad \text{and } Regular(f, \langle n2\ x2\rangle + f2)\}.
\end{aligned}$$

Letting $Image(f1) = X1$ and $Image(f2) = X2$ we can rewrite the above as

$$\begin{aligned}
&\Leftarrow \{f \mid f \subseteq n + N \times x1 + X1 \cup x2 + X2\\
&\qquad \text{and } Bijective(f, n + N, x1 + X1 \cup x2 + X2)\\
&\qquad \text{and } Regular(f, \langle n1\ x1\rangle + f1)\\
&\qquad \text{and } Regular(f, \langle n2\ x2\rangle + f2)\}.
\end{aligned}$$

We are now faced with the task of reducing the Cartesian Product. The above reduces to

$$\begin{aligned}
&\Leftarrow \{f \mid f \subseteq \langle n\ x1\rangle + (\{n\} \times X1) \cup \langle n\ x2\rangle + (\{n\} \times X2)\\
&\qquad\qquad \cup N \times \{x1\} \cup N \times X1 \cup N \times \{x2\} \cup N \times X2\\
&\qquad\qquad \text{and } Bijective(f, n + N, x1 + X1 \cup x2 + X2)\\
&\qquad\qquad \text{and } Regular(f, \langle n1\ x1\rangle + f1)\\
&\qquad\qquad \text{and } Regular(f, \langle n2\ x2\rangle + f2)\}
\end{aligned}$$

Using rule RC3.

$$\begin{aligned}
&\Leftarrow \{f1' \cup f2' \cup f3' \cup f4' \cup f5' \cup f6' \cup f7' \cup f8' \mid\\
&\quad f1' \subseteq \{\langle n\ x1\rangle\}, f2' \subseteq \{n\} \times X1, f3' \subseteq \{\langle n\ x2\rangle\}, f4' \subseteq \{n\} \times X2\\
&\quad f5' \subseteq N \times \{x1\}, f6' \subseteq N \times X1, f7' \subseteq N \times \{x2\}, f8' \subseteq N \times X2\\
&\quad \text{and } Bijective(f1' \cup f2' \cup f3' \cup f4' \cup f5' \cup f6' \cup f7' \cup f8',\\
&\qquad\qquad\qquad n + N, x1 + X1 \cup x2 + X2)\\
&\quad \text{and } Regular(f1' \cup f2' \cup f3' \cup f4' \cup f5' \cup f6' \cup f7' \cup f8', \langle n1\ x1\rangle + f1)\\
&\quad \text{and } Regular(f1' \cup f2' \cup f3' \cup f4' \cup f5' \cup f6' \cup f7' \cup f8', \langle n2\ x2\rangle + f2)\}
\end{aligned}$$

Using rule RS3.

We must now examine which combinations of $f'i's$ will survive the three filters. It is easier if we present this in tabular form. In the following table a row with ticks in the $f'i$'th column means that all the other $f'i's$ must be Φ for the total combination $\bigcup_{1 \leq i \leq 8} fi'$ to pass the filter in question.

We first consider the *Bijective* filter, this ensures that the compound relation is total on its *Domain* and *Image* and functional. These considerations ensure that only the following combinations pass the *Bijective* filter.

Combination	Component							
	$f1'$	$f2'$	$f3'$	$f4'$	$f5'$	$f6'$	$f7'$	$f8'$
1	∨					∨	∨	∨
2			∨		∨	∨		∨
3		∨			∨		∨	
4				∨	∨	∨	∨	

Looking at the first *Regular* filter we see that this disqualifies combination 3 because $f2'$ will ensure that n maps onto some $x1'$ from $X1 = Image(f1)$ while $f5'$ means $x1$ will be mapped onto by some n' from N and $n < n'$ whereas $f1^{-1}(x1) < f1^{-1}(x1')$. Similarly the second *Regular* filter knocks out combination 4 and we are left with 1 and 2.

Here we see the origin of the new definition introduced for *Merge* at equations 5 and 6 in section 5.3, 1. Adding the extra, *Regular*, filters to the definition of *Perm* is exactly equivalent to omitting these combinations of the Cartesian Product. Purely for the purpose of the syntheses we need not have explicitly given the definition of *Regular* that we did in equation 5.3.1, 6. Instead we could have decided arbitrarily to discard these combinations and define *Regular* as the originally undefined extra filter established by these omissions. We shall encounter a very similar pheonomena in our syntheses of *Bubble Sort* and *Exchange Sort* in section 5.4, 1.

We can also now give some justification for the introduction of equation 5. $Perm(X)$ requires that every element of X appears in the first place of some "array". The *Regular* filters ensure that only $x1$ and $x2$ can appear in first positions in $Merge1(\langle n1\ x1\rangle + f1, \langle n2\ x2\rangle + f2, n + N)$ so for $Perm(X1 \cup X2)$ we have to take all $f1 \in Perm(X1)$ and all $f2 \in Perm(X2)$ to ensure that every member of $X1 \cup X2$ gets a chance to be in first place.

Thus the two expressions that survive are

$$C1 = \langle n\ x1\rangle + f6' \cup f7' \cup f8'$$
$$\text{where } f6' \subseteq N \times X1, f7' \subseteq N \times \{x2\}, f8' \subseteq N \times X2$$

and

$$C2 = \langle n\ x2\rangle + f5' \cup f6' \cup f8'$$
$$\text{where } f5' \subseteq N \times \{x1\}, f6' \subseteq N \times X1, f8' \subseteq N \times X2$$

and we can recombine the Cartesian Product thus

$$C1 = \langle n\ x1\rangle + f1' \text{ where } f1' \subseteq N \times (x2 + X1 \cup X2)$$
$$C2 = \langle n\ x2\rangle + f2' \text{ where } f2' \subseteq N \times (x1 + X1 \cup X2).$$

Without going into tedious detail it should be clear to the reader that for these combinations the filters decompose smoothly, i.e.

$$\begin{aligned}
&\mathit{Bijective}(\langle n\ x1\rangle + f1', n+N, x1+X1 \cup x2+X2)\\
&\qquad\qquad \Leftrightarrow \mathit{Bijective}(f1', N, X1 \cup x2+X2)\\
&\mathit{Regular}(\langle n\ x1\rangle + f1', \langle n1\ x1\rangle + f1)\\
&\qquad\qquad \Leftrightarrow \mathit{Regular}(f1', f1)\\
&\mathit{Regular}(\langle n\ x1\rangle + f1', \langle n2\ x2\rangle + f2)\\
&\qquad\qquad \Leftrightarrow \mathit{Regular}(f1', \langle n2\ x2\rangle + f2)\\
&\mathit{Bijective}(\langle n\ x2\rangle + f2', n+N, x1+X1 \cup x2+X2)\\
&\qquad\qquad \Leftrightarrow \mathit{Bijective}(f2', N, x1+X1 \cup X2)\\
&\mathit{Regular}(\langle n\ x2\rangle + f2', \langle n1\ x1\rangle + f1)\\
&\qquad\qquad \Leftrightarrow \mathit{Regular}(f2', \langle n1\ x1\rangle + f1)\\
&\mathit{Regular}(\langle n\ x2\rangle + f2', \langle n2\ x2\rangle + f2)\\
&\qquad\qquad \Leftrightarrow \mathit{Regular}(f2', f2).
\end{aligned}$$

Thus we rewrite the above as

$$\begin{aligned}
&\mathit{Merge}(\langle n1\ x1\rangle + f1, \langle n2\ x2\rangle + f2, n+N)\\
&\qquad \Leftarrow \{\langle n\ x1\rangle + f1' \mid f1' \subseteq N \times x2+X1 \cup X2\\
&\qquad\qquad\qquad \mathit{and\ Bijective}(f1', N, x2+X1 \cup X2)\\
&\qquad\qquad\qquad \mathit{and\ Regular}(f1', f1)\\
&\qquad\qquad\qquad \mathit{and\ Regular}(f1', \langle n2\ x2\rangle + f2)\}\\
&\qquad \cup \{\langle n\ x2\rangle + f2' \mid f2' \subseteq N \times x1+X1 \cup X2\\
&\qquad\qquad\qquad \mathit{and\ Bijective}(f2', N, x1+X1 \cup X2)\\
&\qquad\qquad\qquad \mathit{and\ Regular}(f2', \langle n1\ x1\rangle + f1)\\
&\qquad\qquad\qquad \mathit{and\ Regular}(f2', f2)\}
\end{aligned}$$

Recalling that $X1 = \mathit{Image}(f1)$ and $X2 = \mathit{Image}(f2)$ we can rewrite the above as

$$\begin{aligned}
&\Leftarrow \{\langle n\ x1\rangle + f1' \mid f1' \subseteq N \times \mathit{Image}(f1) \cup \mathit{Image}(\langle n2\ x2\rangle + f2)\\
&\qquad\qquad \mathit{and\ Bijective}(f1', N, \mathit{Image}(f1) \cup \mathit{Image}(\langle n2\ x2\rangle + f2))\\
&\qquad\qquad \mathit{and\ Regular}(f1', f1)\\
&\qquad\qquad \mathit{and\ Regular}(f1', \langle n2\ x2\rangle + f2)\}\\
&\cup \{\langle n\ x2\rangle + f2' \mid f2' \subseteq N \times \mathit{Image}(\langle n1\ x1\rangle + f1) \cup \mathit{Image}(f2)\\
&\qquad\qquad \mathit{and\ Bijective}(f2', N, \mathit{Image}(\langle n1\ x1\rangle + f1) \cup \mathit{Image}(f2))\\
&\qquad\qquad \mathit{and\ Regular}(f2', \langle n1\ x1\rangle + f1)\\
&\qquad\qquad \mathit{and\ Regular}(f2', f2)\}
\end{aligned}$$

6.5. We have

$$\mathit{Sort}(X1 \cup X2) \Leftarrow \bigcup_{\substack{f1 \in \mathit{Perm}(X1)\\ f2 \in \mathit{Perm}(X2)}} \mathit{Ordered}(\mathit{Merge}(f1, f2))$$

To force a fold with 5.3.2, 1 we rewrite this as

$$\Leftarrow \bigcup_{\substack{f1 \in \mathit{Ordered}(\mathit{Perm}(X1))\\ f2 \in \mathit{Ordered}(\mathit{Perm}(X2))}} \mathit{Ordered}(\mathit{Merge}(f1, f2)).$$

It is fairly easy to see that this rewrite is legitimate. We have to show that for any $f1 \in \mathit{Perm}(X1)$ and $f1 \notin \mathit{Ordered}(\mathit{Perm}(X1))$ or

$$f2 \in Perm(X2) \text{ and } \quad f2 \notin Ordered(Perm(X2)),\ Ordered(Merge(f1, f2)) = \Phi.$$

But this is straightforward if we remember that the *Regular* filter ensures that *Merge* does not disturb the internal order of *f1* or *f2* so if either *f1* or *f2* are not ordered *Merge*(*f1*, *f2*) cannot be ordered. More precisely we have

① $not\ Ord(f1) \Rightarrow \exists \langle n1\ x1\rangle, \langle n2\ x2\rangle \in f1 \cdot$
$\qquad n1 < n2\ and\ x2 < x1$

② $\forall f \in Merge(f1, f2) \cdot$
$\qquad \Rightarrow (Regular(f, f1))\ i.e.\ \forall x3, x4 \in Image(f1) \cdot$
$\qquad\qquad f1^{-1}(x3) < f1^{-1}(x4)$
$\qquad\qquad \Rightarrow f^{-1}(x3) < f^{-1}(x4)$

① *and* ② $\Rightarrow \forall f \in Merge(f1, f2) \cdot \exists \langle n1\ x1\rangle, \langle n2\ x2\rangle \in f \cdot n1 < n2\ and\ x2 < x1$
$\qquad \Rightarrow not\ Ord(f)$

i.e. $Ordered(Merge(f1, f2)) = \Phi$ □

6.6. $MergeS(\langle n1\ x1\rangle + f1, \langle n2\ x2\rangle + f2, n+N)$

$\Leftarrow Ordered(Merge1(\langle n1\ x1\rangle + f1, \langle n2\ x2\rangle + f2, n+N))$

Instantiating 5.3.2.6

$\Leftarrow Ordered(\{\langle n\ x1\rangle + f1' \mid f1' \in Merge1(f1, \langle n2\ x2\rangle + f2, N)\}$
$\quad \cup \{\langle n\ x2\rangle + f2' \mid f2' \in Merge1(\langle n1\ x1\rangle + f1, f2, N)\})$

Unfolding using 5.3.1.11

$\Leftarrow Ordered(\{\langle n\ x1\rangle + f1' \mid f1' \in Merge1(f1, \langle n2\ x2\rangle + f2, N)\})$
$\cup\ Ordered(\{\langle n\ x2\rangle + f2' \mid f2' \in Merge1(\langle n1\ x1\rangle + f1, f2, N)\})$

Taking *Ordered* inside the union

$\Leftarrow \{\langle n\ x1\rangle + f1' \mid f1' \in Merge1(f1, \langle n2\ x2\rangle + f2, N)$
$\qquad and\ Ord(\langle n\ x1\rangle + f1')\}$
$\cup \{\langle n\ x2\rangle + f2' \mid f2' \in Merge1(\langle n1\ x1\rangle + f1, f2, N)$
$\qquad and\ Ord(\langle n\ x2\rangle + f2')\}$

Unfolding using 5.2.2.2.

Now assume that $x1 < x2$, looking at the second component of the union and remembering that $\forall n' \in N \cdot n < n'$, it is easy to show that

$$\forall f2' \in Merge1(\langle n1\ x1\rangle + f1, f2, N), \exists \langle n'\ x1\rangle \in f2' \cdot n < n'$$

i.e. $not\ Ord(\langle n\ x2\rangle + f2')$ for any $f2' \in Merge1(\langle n1\ x1\rangle + f1, f2, N)$.

Thus in this case the second component is Φ, similarly the first component is Φ if $x2 < x1$. Furthermore we have

$Ord(\langle n\ x1\rangle + f1') \Leftrightarrow Ord(f1')$ (if $x1 < x2$)

and

$Ord(\langle n\ x2\rangle + f2') \Leftrightarrow Ord(f2')$ (if $x2 < x1$)

and we can rewrite the above as

$MergeS(\langle n1\ x1\rangle + f1, \langle n2\ x2\rangle + f2, n+N)$
$\quad \Leftarrow \{\langle n\ x1\rangle + f1' \mid f1' \in Merge1(f1, \langle n2\ x2\rangle + f2, N)\ and\ Ord(f1')\}$
$\qquad\qquad$ **if** $x1 < x2$
$\quad\ \ \{\langle n\ x2\rangle + f2' \mid f2' \in Merge1(\langle n1\ x1\rangle + f1, f2, N)\ and\ Ord(f2')\}$
$\qquad\qquad$ **otherwise**.

6.7. We have

$$\begin{aligned}&\mathit{MergeS}(\{\langle 1\ x\rangle\}, \langle n2\ x2\rangle + f2, n+N)\\ &\quad \Leftarrow \{\langle n\ x\rangle + f1' \mid f1' \in \mathit{Merge1}(\Phi, \langle n2\ x2\rangle + f2, N)\\ &\qquad\qquad \mathit{and}\ \mathit{Ord}(\langle n\ x\rangle + f1')\}\\ &\quad \cup \{\langle n\ x2\rangle + f2' \mid f2' \in \mathit{Merge1}(\{\langle 1\ x\rangle\}, f2, N)\\ &\qquad\qquad \mathit{and}\ \mathit{Ord}(\langle n\ x2\rangle + f2')\}\\ &\qquad\qquad \text{Unfolding 5.3.3, 8 using 5.2.2, 2.}\end{aligned}$$

If $x<x2$ for any $f2' \in \mathit{Merge1}(\{\langle 1\ x\rangle\}, f2, N)$ there will be a $\langle n'\ x\rangle \in f2'$ so that $n<n'$ and $x<x2$ i.e. *not* $\mathit{Ord}(\langle n\ x2\rangle + f2')$, thus the lower component of the union will be Φ. Similarly if $x2<x$ we have *not* $\mathit{Ord}(\langle n\ x\rangle + f1')$ for any $f1' \in \mathit{Merge1}(\Phi, \langle n2\ x2\rangle + f2, N)$ so the upper component is Φ. Furthermore in the case $x\langle x2$ we have

$\mathit{Ord}(\langle n\ x\rangle + f1') \Leftrightarrow \mathit{Ord}(f1')$ and in the case $x2<x$ we have
$\mathit{Ord}(\langle n\ x2\rangle + f2') \Leftrightarrow \mathit{Ord}(f2')$. Thus we have

$$\begin{aligned}&\mathit{MergeS}(\{\langle 1\ x\rangle\}, \langle n2\ x2\rangle + f2, n+N)\\ &\quad \Leftarrow \{\langle n\ x\rangle + f1' \mid f1' \in \mathit{Merge1}(\Phi, \langle n2\ x2\rangle + f2, N)\\ &\qquad\qquad \mathit{and}\ \mathit{Ord}(f1')\}\\ &\qquad\qquad \textbf{if}\ x<x2\\ &\quad\ \{\langle n\ x2\rangle + f2' \mid f2' \in \mathit{Merge1}(\{\langle 1\ x\rangle\}, f2, N)\\ &\qquad\qquad \mathit{and}\ \mathit{Ord}(f2')\}\\ &\qquad\qquad \textbf{otherwise}.\end{aligned}$$

6.8. We have

$$\begin{aligned}&\mathit{Perm1}(\langle n_1\ x_1\rangle + \langle n_2\ x_2\rangle + f)\\ &\quad \Leftarrow \{f_1 \mid f_1 \subseteq \mathit{Domain}(\langle n_1\ x_1\rangle + \langle n_2\ x_2\rangle + f) \times \mathit{Image}(\langle n_1\ x_1\rangle\\ &\qquad\qquad\qquad + \langle n_2\ x_2\rangle + f)\\ &\qquad \mathit{and}\ \mathit{Bijective}(f_1, \mathit{Domain}(\langle n_1\ x_1\rangle + \langle n_2\ x_2\rangle + f),\\ &\qquad\qquad \mathit{Image}(\langle n_1\ x_1\rangle + \langle n_2\ x_2\rangle + f))\\ &\qquad \mathit{and}\ \mathit{Close}(f_1, \langle n_1\ x_1\rangle + \langle n_2\ x_2\rangle + f)\}\end{aligned}$$

We let $\mathit{Domain}(f) = N'$ and $n_2 + N' = N$ and $\mathit{Image}(f) = X$. Thus $\mathit{Domain}(\langle n_1\ x_1\rangle + \langle n_2\ x_2\rangle + f) \times \mathit{Image}(\langle n_1\ x_1\rangle + \langle n_2\ x_2\rangle + f)$ is $(n_1 + N) \times (x_1 + x_2 + X)$.
Expanding this out we get

$$\begin{aligned}&\langle n_1\ x_1\rangle + \langle n_1\ x_2\rangle \cup \{n_1\} \times X\\ &\cup N \times \{x_1\} \cup N \times \{x_2\} \cup N \times X\end{aligned}$$

which we rewrite as

$$\begin{aligned}&\langle n_1\ x_1\rangle + \langle n_1\ x_2\rangle \cup \{n_1\} \times X\\ &\cup N \times (x_1 + X) \cup N \times (x_2 + X).\end{aligned}$$

The definition of *Perm1* given by 5.4.1, 4 and 5.4.1, 5 arises from omitting $\{n_1\} \times X$ from this expression, meaning that n_1 is allowed to map onto only x_1 or x_2. Thus as with *Regular* in the earlier section we need not have interpreted *Close* by giving 5.4.1, 5. Rewriting 5.4.1, 8 with this restricted expression we get

$$\begin{aligned}
&\mathit{Perm1}(\langle n_1\, x_1\rangle + \langle n_2\, x_2\rangle + f)\\
&\quad \Leftarrow \{f_1 \mid f_1 \subseteq \langle n_1\, x_1\rangle + \langle n_1\, x_2\rangle\\
&\qquad\qquad + N \times (x_1 + X) \cup N \times (x_2 + X)\\
&\qquad \textit{and } \mathit{Bijective}(f_1, n_1 + N, x_1 + x_2 + X)\\
&\qquad \textit{and } \mathit{Close}(f_1, \langle n_1\, x_1\rangle + \langle n_2\, x_2\rangle + f)\}.
\end{aligned}$$

We are again in the familiar position of forcing a fold with 5.4.1, 5. The reader should be aware of the technique now so we will be brief. Both *Bijective* and *Close* (having omitted $\{n_1\} \times X$) decompose nicely and the various combinations of the subsets (using rule RS3) mean that n_1 goes onto either x_1 or x_2 which are combined with subsets from $N \times x_2 + X$ and $N \times x_1 + X$ respectively. Thus we have

$$\begin{aligned}
&\mathit{Perm1}(\langle n_1\, x_1\rangle + \langle n_2\, x_2\rangle + f)\\
&\quad \Leftarrow \{\langle n_1\, x_1\rangle + f_1 \mid f_1 \subseteq N \times x_2 + X\\
&\qquad\qquad \textit{and } \mathit{Bijective}(f_1, N, x_2 + X)\\
&\qquad\qquad \textit{and } \mathit{Close}(f_1, \langle n_2\, x_2\rangle + f)\}\\
&\quad \cup \{\langle n_1\, x_2\rangle + f_2 \mid f_2 \subseteq N \times x_1 + X\\
&\qquad\qquad \textit{and } \mathit{Bijective}(f_2, N, x_1 + X)\\
&\qquad\qquad \textit{and } \mathit{Close}(f_2, \langle n_2\, x_1\rangle + f)\}.
\end{aligned}$$

7. Conclusion

We hope we have demonstrated the practically and usefulness of the synthesis approach and thrown some further light on the structure of the sorting algorithms. However, we see this as only a preliminary attempt at a very large task and feel that further attempts at regularising or even partly mechanising the syntheses would prove useful. Also we make no claim that we have exposed *the* structure for these sorting algorithms. The work reported in Green et al. [5] reveals interesting similarities and differences. In many ways we have made life more difficult by attempting to synthesise the permutation algorithms on the way. A subsequent smaller study attempting to synthesis only 4 of the sorts and using slightly richer syntax proved much simpler, Clark and Darlington [2].

This work has been supported by SRC grants to the Department of Artificial Intelligence at the University of Edinburgh, and the Department of Computing at Imperial College. We would like to thank all our colleagues there for their help, in particular Rod Burstall and Jerry Schwarz for reading original drafts and making very valuable suggestions.

References

1. Burstall, R.M., Darlington, J.: A transformation system for developing recursive programs. J. Assoc. Comput. Mach. **24**, 44–67 (1977)
2. Clark, K., Darlington, J.: Algorithm classification through synthesis. Internal Report Department of Computing and Control, Imperial College, London, 1978. Comput. J. (to appear)
3. Darlington, J.: Applications of program transformation to program synthesis. Proceedings of Symposium on Proving and Improving Programs, pp. 133–144, Arc-et-Senans, France, 1975
4. Dijkstra, E.W.: A discipline of programming. Englewood Cliffs, N.J.: Prentice-Hall 1976
5. Green, C.C., et al.: Progress report on program-understanding systems. Stanford Memo AIM 240, Stanford University, Computer Science Dept., 1974
6. Manna, Z., Waldinger, R.: Knowledge and reasoning in program synthesis. Artificial Intelligence **6**, No. 2 (1975)

Received May 17, 1977