Parsing in ISBES

Eerke A. Boiten*

Department of Mathematics and Computing Science
Eindhoven University of Technology
P.O Box 513, 5600 MB Eindhoven, The Netherlands
email: eerke@win.tue.nl

Abstract. It is shown how parsing can be described as a problem in the class ISBES, Intersections of Sets and Bags of Extended Substructures, defined in an earlier paper, by viewing parsing as a generalization of pattern matching in several ways. The resulting description is shown to be a good starting point for the transformational derivation of the Cocke-Kasami-Younger tabular parsing algorithm that follows. This derivation is carried out at the level of bag comprehensions.

Keywords. transformational programming, formal specification, substructures, bags, parsing, Cocke-Kasami-Younger

1 Introduction

Transformational programming is a methodology for the derivation of efficient programs from formal specifications by applying semantics preserving transformations, thus guaranteeing correctness of the final result. For a survey of the transformational method, cf. [Par90]. Current research is no longer concentrated on particular derivations or transformation steps, but on higher level knowledge: data types with their characteristic algorithms and properties (*theories of data*, e.g. [Bir87]); transformation strategies so well understood that they can be automated (*theories of programs*, e.g. [SL90]); derivations and their general shapes (*theories of derivations*, e.g. [PB91]).

The present paper falls in the first category: formal specifications in terms of particular data types, and the algorithms that can be derived from those.

An earlier paper [Boi91b] proposed a general problem, called ISBES (Intersection of a Set and a Bag of Extended Substructures), mainly as a generalization of pattern matching problems. Several instances of this problem were given, most of them pattern matching examples. The claim that *parsing* could also be treated in this way was removed from a preliminary version of the paper, since it was not clear how this could be substantiated.

* This research has been carried out at the University of Nijmegen, Department of Computing Science, sponsored by the Netherlands Organization for Scientific Research (NWO), under grant NF 63/62-518 (the STOP — Specification and Transformation Of Programs — project).

This paper describes how, from an almost trivial specification of parsing as an ISBES problem, by stepwise generalization an ISBES instance is obtained that naturally leads to a derivation of the Cocke-Kasami-Younger parsing algorithm.

Various characteristics of the final solution arise during *both* phases of the development: the formulation of parsing as an ISBES instance, and the derivation of the algorithm. This substantiates the claim that formal specification in ISBES-style has relevance for different kinds of problems besides pattern matching.

2 Types and Operations

Types are denoted as sets of introduction rules with laws on terms. The important type *Struct* of binary structures is defined by introduction rules

$$\frac{}{\varepsilon : \mathit{Struct}(\mathbf{t})} \qquad \frac{x : \mathbf{t}}{\tau(x) : \mathit{Struct}(\mathbf{t})} \qquad \frac{a, b : \mathit{Struct}(\mathbf{t})}{a \sqcup b : \mathit{Struct}(\mathbf{t})}$$

and law

$$x \sqcup \varepsilon = \varepsilon \sqcup x = x,$$

i.e. ε is the unit of $\sqcup$. When $\sqcup$ is associative, we have the type *List* of finite lists (usually with $+\!\!+$ for $\sqcup$). If it is also commutative, we get the type *Bag* of finite bags (with $\uplus$). When, additionally, $\sqcup$ is idempotent, the type *Set* of finite sets is obtained, i.e. $\sqcup$ can be denoted by $\cup$. This family of structured types is commonly named the Boom-hierarchy [Mee89].

A number of useful operators [Bir87] on these types, together colloquially called "Squiggol", are defined below (a, b are of type $\mathit{Struct}(\mathbf{t}_1)$ and c, d have type $\mathit{Struct}(\mathbf{t}_2)$):

- "filter" $\lhd$ that takes a predicate and a binary structure, and returns the structure containing those of its elements that satisfy the predicate:

$$\begin{aligned} p \lhd \varepsilon &= \varepsilon \\ p \lhd (\tau(x) \sqcup a) &= \mathbf{if}\ p(x)\ \mathbf{then}\ \tau(x) \sqcup (p \lhd a) \\ &\qquad \mathbf{else}\ p \lhd a\ \mathbf{fi} \end{aligned}$$

- "map" $*$ that takes a function and a binary structure, and applies the function elementwise:

$$\begin{aligned} f * \varepsilon &= \varepsilon \\ f * \tau(x) &= \tau(f\ x) \\ f * (a \sqcup b) &= (f * a) \sqcup (f * b) \end{aligned}$$

- "reduce" / that takes a binary operator $\oplus$ and a binary structure, that it reduces by putting $\oplus$ between all elements ($1_\oplus$ denotes the unit of $\oplus$):

$$\begin{aligned} \oplus/\varepsilon &= 1_\oplus \\ \oplus/\tau(y) &= y \\ \oplus/(a \sqcup b) &= (\oplus/a) \oplus (\oplus/b) \text{ for nonempty } a, b \end{aligned}$$

– "cross" X that takes two binary structures and a binary operation, generates all possible combinations of elements from one of each of the structures and combines them using the operation:

$$\begin{aligned} \mathsf{X}_{\oplus}\ \varepsilon &= \varepsilon \\ a\,\mathsf{X}_{\oplus}\ \tau(x) &= (\oplus x) * a \\ a\,\mathsf{X}_{\oplus}\ (c \sqcup d) &= (a\,\mathsf{X}_{\oplus}\ c) \sqcup (a\,\mathsf{X}_{\oplus}\ d) \end{aligned}$$

Tuples are denoted with <>. Let π_i denote the projection to the i-th component of a tuple; compositions of projections will be denoted with lists of indices, e.g. $\pi_1(\pi_2(\pi_2(x))) = \pi_{1,2,2}(x)$. Lists of two elements will occur frequently; $[a, b]$ will be used for the list $\tau(a) \# \tau(b)$. Partial parametrization of prefix functions is denoted using underscores, i.e. $f(a, \_, c)(b) = f(a, b, c)$.

3 ISBES Revisited

The problem class ISBES (Intersection of a Set and a Bag of Extended Substructures) can be specified as follows:

Given
- a structured type $S(\mathbf{t})$,
- an object O of the structured type $S(\mathbf{t})$,
- a set P (patterns) of objects of the type $S^+(\mathbf{t})$, where $S^+(\mathbf{t})$ is a type of tuples, the first component of which is in $S(\mathbf{t})$ and the second is in another type of "labels".
- a function D^+ (decompose) which yields the bag of extended substructures (of type $S^+(\mathbf{t})$) of an object of type $S(\mathbf{t})$,
- a function R (result) which takes a bag of elements of type $S^+(\mathbf{t})$ and returns the "answer" to the problem,

compute

$$R(P \cap D^+(O))$$

where $\cap$ is defined by

$$S \cap B = (\in S) \lhd B.$$

Formal definitions of the terms used in this description can be found in [Boi91b]; informally

- a *structured type* $S(\mathbf{t})$ is a type of objects that consist of *structure*, and *basic elements* of type $\mathbf{t}$. These can be seen as together forming the *information* of the structured object. Examples of structured types are lists, bags, sets, and trees of all possible kinds.
- a *substructure* of a structured object is an object of the same type, that contains a subset of the information present in that object. For each structured type, various useful notions of substructures exists – lists, e.g., have prefixes, suffixes, substrings, and segments.

– an *extended substructure* of a structured object is a substructure, labeled with information about the "position" of the substructure in the original object. An example form, for lists, segments with indices. The bag of all substructures of S that have label E is given by $\phi(S, E)$, where ϕ is a function characterizing the type of substructure chosen.

Some ideas on deriving programs from ISBES-style specifications are given in [Boi91b]. Important is also the following:

Observation 1 *When each label component of an extended substructure uniquely identifies the substructure within the original object, a data type transformation can take place, viz. the substructure can be represented by just its label when the original object is known (in context).*

In the case that a certain bag contains at most one occurrence of each value, it could be said that the bag "is actually a set". There is a subtle difference between the property mentioned above and the property that the bag of all substructures "is actually a set". The data type transformation can only be done when ϕ returns at most singleton bags (e.g. for lists with indices), whereas the bag of all substructures "is a set" in *all* cases where ϕ returns bags containing all different elements (e.g. in the case of segments with offsets).

4 Parsing in ISBES

The problem class ISBES describes generalizations of pattern matching; intuitively *parsing* appears to be a related problem and so maybe also something that could be described as an ISBES problem.

Pattern matching is the problem of deciding whether any element of a certain set of patterns occurs in a structure; parsing is the problem of determining if and how a certain sentence is generated by a certain grammar (to fix terminology for now). A borderline case of pattern matching occurs when the structure is one of the patterns. Parsing can be seen as a pattern matching generalization in at least two ways. In one view, the set of strings generated by the language is considered as the pattern set, and we ask whether the sentence is one of the patterns. This is worked out in detail in Section . An alternative view is to take a complicated (infinite) structure representing *all* possible sentences generated by the grammar, and as the pattern set (implicitly) only the structures that represent derivations of the sentence. This alternative view will be discussed in Section .

In order to be able to formally describe these ideas, we now first give a formal description of the parsing problem (what follows now is a Squiggol-ish description of traditional formal language theory).

Definition 2 *A* context free grammar *is a 4-tuple* $< S, V_N, V_T, PR >$ *such that* $S \in V_N$ *and* $PR : V_N \to Set(List(V))$ *where* $V = V_N + V_T$.

In such a grammar $< S, V_N, V_T, PR >$, S is called the *starting symbol*; elements of V_N are called *nonterminals* and elements of V_T *terminals*. If $r \in PR(X)$, then r is called an *alternative* or *right hand side* of (the *left hand side*) X.

Definition 3 *The* language *generated by a grammar* $< S, V_N, V_T, PR >$ *is the set* $\mathcal{L}(\tau(S))$*, where* $\mathcal{L} : List(V) \to Set(List(V_T))$ *is defined by:*

$$\begin{aligned} \mathcal{L}(\varepsilon) &= \{\varepsilon\} \\ \mathcal{L}(\tau(x)) &= \{\tau(x)\} \quad (x \in V_T) \\ \mathcal{L}(\tau(x)) &= \cup/\mathcal{L} * PR(x) \quad (x \in V_N) \\ \mathcal{L}(x \# y) &= \mathcal{L}(x) \mathsf{X}_{\#} \mathcal{L}(y) \end{aligned}$$

Definition 4 *A grammar* $< S, V_N, V_T, PR >$ *is in* Chomsky Normal Form *(CNF) iff all right hand sides consist of exactly one terminal or exactly two nonterminals, i.e.*

$$\forall\, X \in V_N : \forall\, s \in PR(X) : (\exists\, x \in V_T : s = \tau(x)) \vee (\exists\, Y, Z \in V_N : s = [Y, Z]).$$

Note that no nonterminal in a CNF grammar generates the empty string. This holds in particular for the starting symbol, so the empty string cannot be in the language generated.

The following subsections describe completely different ways of describing parsing as an ISBES problem. For completeness, we also note that in principle everything can be described as an ISBES problem: if one takes a trivial notion of substructure, viz. every object is only a substructure of itself, and one takes the pattern set to be some kind of universe, the ISBES problem reduces to computing $R(\{O\})$ for some object O and function R. This implies that "anything" is an ISBES problem, and thus so is parsing. An important point is, indeed, that ISBES is a *way of describing* problems, not a limited class of itself.

4.1 Substructures of Parse Forests

Consider the grammar (incidentally, one in CNF) in Van Wijngaarden notation

S:S, S;
  "a".

i.e. $< S, \{S\}, \{a\}, \lambda S.\{[S, S], [a]\} >$, then the type of parse trees for this grammar is

$$\frac{}{P_2(\text{“}a\text{”}) : S} \qquad\qquad \frac{x, y : S}{P_1(x, y) : S}$$

This gives a type describing all different parse trees. Sets of parse trees are also called parse *forests*. Parse forests can be represented in a condensed form as trees, where each node contains a (nonempty) *set* of children[1]. Although descriptions

[1] This may be called the *mixed type* of parse trees and sets; we intend to further study mixed types and their properties.

of types via introduction rules are usually understood to describe finite objects only, one can imagine infinite objects as well. The list of infinitely many a's can be viewed as a "supremum" of the type $List(\{a\})$, and likewise the "supremum" of all parse forests of the grammar above is the parse forest S^∞, characterized by

$$S^\infty = U(P_1(S^\infty, S^\infty), P_2(\text{“}a\text{”}))$$

where U ("union") is a constructor function with no additional properties. Now all parse trees of the grammar can be seen as (finite) subtrees of S^∞, viz. those where each node is a *singleton* set.

For arbitrary grammars a similar characterization of S^∞ can be given by simultaneously defining A^∞ for all $A \in V_N$ by $A^\infty = U_A(i * *PR(A))$, where $i(x) = x$ for $x \in V_T$, $i(X) = X^\infty$ for $X \in V_N$, and U_A is a unique constructor without additional properties.

For parse trees, define the *yield* to be the concatenated list of all its (terminal) leaves. Parsing is then an ISBES problem: the pattern set contains all parse trees that have the string to be parsed as their yield, the structured object is S^∞, and substructures are finite subforests with singleton nodes. The infinite parse forest S^∞ represents all possible choices to be made during parsing, whereas the pattern set represents in an implicit way all possible outcomes of the parsing process. The set of substructures of S^∞ can be decomposed in several ways. The decomposition that is chosen in a derivation that starts from this specification largely determines what parsing algorithm is obtained. (Note that $\cap\!\!\!\cap$ distributes through set union, and thus each subset of $D^+(S^\infty)$ can be independently intersected with the pattern set). Parsing algorithms derived from this specification have as an advantage that parse trees are already present in the solution, whereas usually the construction of parse trees has to be grafted upon a recognition algorithm [AU72].

4.2 Stepwise Generalization via the Pattern Set

Using the experience of describing string matching as an ISBES problem [Boi91b], we now give another description of parsing as an ISBES problem. From now on, we consider *non-empty* strings and (thus) non-empty segments only. (Formally, this is done by discarding the introduction rule for ε from the type *List*).

First, define the bag of all segments of a string by

$$segs\ (s) = \{\!|\, p |\ \exists\, x, y : s = x \# p \# y \,|\!\}$$

where $\{\!|\ |\!\}$ denotes bag comprehension (cf. the appendix). Since $w \in segs\ (w)$, a trivial ISBES description of parsing is given by

$$\begin{aligned} O &\triangleq w \\ P &\triangleq \mathcal{L}(S) \\ D^+ &\triangleq segs \\ R &\triangleq (w \in) \end{aligned}$$

i.e., compute

$$w \in (\mathcal{L}(S) \cap segs\ (w)).$$

Although this seems a trivial generalization, it suggests the (in the context of parsing relevant) computation of $\{w' \in segs\ (w) \mid w' \in \mathcal{L}(S)\}$.

The substructures used above are *segments*, and thus it is obvious that choosing *indexed* segments, defined by

$$segsi\ (s) = \{\!\![\ < p, < i, j >> \mid p = s[i..j]]\!\!\}$$

as *extended* substructures may be helpful in this case.[2] This leads to the following ISBES instance, where the pattern set has been extended to contain elements of the same type as *segsi*:

$$\begin{aligned} D^+ &\triangleq segsi \\ P &\triangleq \{< x, < m, m+|x| >> \mid x \in \mathcal{L}(S) \wedge m \leq m+|x| \leq |w|\ \} \\ R &\triangleq (< w, < 0, |w| >> \in). \end{aligned}$$

Note that the predicate for P allows an easy restriction to elements of $\mathcal{L}(S)$ not longer than $|w|$.

For $V_N = \{S\}$ and PR in CNF, this is already quite close to the Cocke-Kasami-Younger (CKY) parsing algorithm. In order to allow for multiple nonterminals in V_N, the pattern set and substructures are extended with nonterminals:

$$\begin{aligned} D^+(s) &\triangleq \{\!\![\ < p, < i, j, X >> \mid p = s[i..j] \wedge X \in V_N]\!\!\} \\ P &\triangleq \{< x, < m, m+|x|, X >> \mid x \in \mathcal{L}(X) \wedge m \leq m+|x| \leq |w|\ \} \\ R &\triangleq (< w, < 0, |w|, S >> \in). \end{aligned}$$

A recursive version of $P \cap D^+(O)$ can be seen as a parsing algorithm that corresponds to the CKY algorithm when the grammar is in CNF. This will be shown in a derivation below.

5 A Derivation of the CKY Parsing Algorithm

The goal is to compute $R(P \cap D^+(w))$. First the subexpression $P \cap D^+(w)$ is reduced to a union of bag comprehensions.

$P \cap D^+(w)$
$\quad =\{$ definition P, definition D^+ $\}$
$\{< x, < m, m+|x|, X >> \mid x \in \mathcal{L}(X) \wedge m \leq m+|x| \leq |w|\} \cap$
$\{\!\![< p, < i, j, X >> \mid p = w[i..j] \wedge X \in V_N]\!\!\}$

[2] Note that we use the convention of not numbering the elements themselves, but the "borders" between them. E.g. $w[1..2]$ denotes the list consisting of the second element of w, and $w[0..|w|] = w$.

$=$\{ definition $\cap\!\!\!\triangleright$, simplifications \}

$\{\![< w[i..i{+}k], < i, i{+}k, X >> \mid w[i..i{+}k] \in \mathcal{L}(X)]\!\}$

$=$\{ comprehension dummies made explicit \}

$\biguplus_{0 \le i < i+k \le |w|} \{\![< w[i..i{+}k], < i, i{+}k, X >> \mid w[i..i{+}k] \in \mathcal{L}(X)]\!\}$

This expression is abstracted by the following *embedding*. From this point on, the string w is assumed to be known in context. Let

$$E(i,k,X) = < w[i..i{+}k], < i, i{+}k, X >> .$$

Define a function cky by:

$$cky(i,k : 0 \le i < i{+}k \le |w|) = \{\![E(i,k,X) \mid w[i..i{+}k] \in \mathcal{L}(X)]\!\}.$$

Thus,

$$P{\cap\!\!\!\triangleright}D^+(w) = \biguplus_{0 \le i < i+k \le |w|} cky(i,k).$$

Also we have that

$$R(P{\cap\!\!\!\triangleright}D^+(w)) = (((= S) \circ \pi_{3,2}) \lhd cky(0,|w|) \neq \emptyset).$$

We derive a recursive definition for cky. Effective parsing algorithms are characterized by the fact that they are phrased in terms of the grammar only, without resorting to reference to the generated language. This should also be the driving force here: occurrences of $\mathcal{L}$ need to be replaced by occurrences of PR. This can be done easily when the grammar is in Chomsky Normal Form, as can be seen in the derivation below.

Some of the more complicated calculation rules used below are given in the appendix, as indicated by a number in the hint. The first step is to be a case introduction on k, viz. $k = 1 \vee k > 1$. In the case that $k = 1$ we have:

$cky(i,1)$

$=$ \{ definition cky \}

$\{\![E(i,1,X) \mid w[i..i{+}1] \in \mathcal{L}(X)]\!\}$

$=$ \{ CNF: strings of length 1 are produced in 1 step\}

$\{\![E(i,1,X) \mid w[i..i{+}1] \in PR(X)]\!\}$,

which is of the required form: it refers to PR but no longer to $\mathcal{L}$.

In the case that $k > 1$ we have:

$cky(i,k)$

$=$ \{ definition cky \}

$\{\![E(i,k,X) \mid w[i..i{+}k] \in \mathcal{L}(X)]\!\}$

$=$ \{ CNF: strings of length ≥ 2 are produced in more than 1 step \}

$\{\![E(i,k,X) \mid \exists\, m, Y, Z : 0 < m < k \wedge Y \in V_N \wedge Z \in V_N$

$\qquad\qquad \wedge w[i..i{+}m] \in \mathcal{L}(Y) \wedge w[i{+}m..i{+}k] \in \mathcal{L}(Z)$

$$\wedge [Y, Z] \in PR(X)\}\!\rangle$$

$= \{$ bag comprehension over linear index domain, Rule 10 $\}$

$$\biguplus_{m=1}^{k-1} \{\!\langle E(i,k,X) \mid \exists Y, Z : Y \in V_N \wedge Z \in V_N \wedge w[i..i{+}m] \in \mathcal{L}(Y) \wedge w[i{+}m..i{+}k] \in \mathcal{L}(Z) \wedge [Y, Z] \in PR(X)\}\!\rangle$$

$= \{$ generalization of domain of comprehension variable, Rule 12, twice$\}$

$$\biguplus_{m=1}^{k-1} \{\!\langle E(i,k,X) \mid \exists Y', Z' : Y' \in \{\!\langle E(i,m,Y) \mid w[i..i{+}m] \in \mathcal{L}(Y)\}\!\rangle \wedge Z' \in \{\!\langle E(i{+}m, k-m, Z) \mid w[i{+}m..i{+}k] \in \mathcal{L}(Z)\}\!\rangle \wedge [\pi_{3,2}(Y'), \pi_{3,2}(Z')] \in PR(X)\}\!\rangle$$

$= \{$ fold cky $\}$

$$\biguplus_{m=1}^{k-1} \{\!\langle E(i,k,X) \mid \exists Y', Z' : Y' \in cky(i,m) \wedge Z' \in cky(i{+}m, k{-}m) \wedge [\pi_{3,2}(Y'), \pi_{3,2}(Z')] \in PR(X)\}\!\rangle$$

$= \{$ bag comprehension over product domain, Rule 11 $\}$

$$\biguplus_{m=1}^{k-1} E(i,k,\_) * (\biguplus / (cky(i,m) \ \mathsf{X}_{\oplus} \ cky(i{+}m, k{-}m)))$$

where $Y' \oplus Z' = \{\!\langle\ X \mid [\pi_{3,2}(Y'), \pi_{3,2}(Z')] \in PR(X)\}\!\rangle$.

Altogether, we have

$cky(i, k : 0 \leq i < i{+}k \leq |w|)$
$=$ **if** $k = 1$
then $\{\!\langle E(i,1,X) \mid w[i..i{+}1] \in PR(X)\}\!\rangle$
else $\biguplus_{m=1}^{k-1} E(i,k,\_) * (\biguplus / (cky(i,m) \ \mathsf{X}_{\oplus} \ cky(i{+}m, k{-}m)))$
where $Y' \oplus Z' = \{\!\langle\ X \mid [\pi_{3,2}(Y'), \pi_{3,2}(Z')] \in PR(X)\}\!\rangle$ **fi**
where $E(i,k,X) =< w[i..i{+}k], < i, i{+}k, X >>$.

By Observation 1, this can be simplified by representing all intermediate results $E(i,k,X)$ by triples $< i,k,X >$. Formally this is done by a somewhat tedious unfold-fold derivation, starting from the definition $cky'(i,k) = E^{-1} * cky(i,j)$, resulting in

$cky(i, k : 0 \leq i < i{+}k \leq |w|)$
$= E * cky'(i,k)$ **where**
$cky'(i, k : 0 \leq i < i{+}k \leq |w|)$
$=$ **if** $k = 1$
then $\{\!\langle\ < i, 1, X > \ \mid w[i..i{+}1] \in PR(X)\}\!\rangle$
else $\biguplus_{m=1}^{k-1} < i, k, \_ > *(\biguplus / (cky'(i,m) \ \mathsf{X}_{\oplus} \ cky'(i{+}m, k{-}m)))$
where $Y' \oplus Z' = \{\!\langle\ X \mid [\pi_3(Y'), \pi_3(Z')] \in PR(X)\}\!\rangle$ **fi**,
$E(i,k,X) =< w[i..i{+}k], < i, i{+}k, X >>$.

By applying *tabulation* to this version of cky, computing $cky(i,k)$ for increasing values of k, the familiar tabular parsing algorithm is obtained. Effectively,

this can be seen as a transition from an inefficient top-down parsing algorithm (which computes $cky(i,k)$ several times for most values of i and k) to an efficient bottom-up parsing algorithm.

6 Final Remarks

Describing parsing as an ISBES style specification proved to be an interesting exercise. The resulting specification could still be called "descriptive" (as opposed to operational), but it did to some extent already "suggest" the Cocke-Kasami-Younger parsing algorithm. It contained already complete information about the intermediate values to be computed for obtaining the final result (which is not necessary for the specification as such), but not in which order they need to be computed (which is necessary for the operational version).

The derivation relies mostly on basic properties of bag comprehensions. It is somewhat unfortunate that the calculation does not proceed on the (higher) level of set-bag intersections, using rules like the ones given in [Boi91b]. This seems due to the fact that, for *efficient* parsing, some of the set-bag intersections that occur in the ISBES parsing specification have to be computed elementwise from previous intersections. The fact that *extended* substructures are carried around, however, makes the calculation seem more complicated than it actually is.

Acknowledgements

Helmut Partsch, Paul Frederiks and Johan Jeuring are thanked for many remarks that improved the clarity and readability of this note.

References

[AU72] A.V. Aho and J.D. Ullman. *The Theory of Parsing, Translation and Compiling, Vol. 1: Parsing*. Prentice-Hall, Englewood Cliffs, New Jersey, 1972.

[Bir87] R.S. Bird. An introduction to the theory of lists. In M. Broy, editor, *Logic of Programming and Calculi of Discrete Design. NATO ASI Series Vol. F36*, pages 5–42. Springer-Verlag, Berlin, 1987.

[Boi91a] E.A. Boiten. Can bag comprehension be used at all? Technical Report 91-21, Dept. of Informatics, K.U. Nijmegen, September 1991.

[Boi91b] E.A. Boiten. Intersections of bags and sets of extended substructures – a class of problems. In [Möl91], pages 33–48.

[Mee89] L.G.L.T. Meertens. Lecture notes on the generic theory of binary structures. In *STOP International Summer School on Constructive Algorithmics, Ameland*. September 1989.

[Möl91] B. Möller, editor. *Proceedings of the IFIP TC2 Working Conference on Constructing Programs from Specifications* North-Holland Publishing Company, Amsterdam, 1991.

[Par90] H. Partsch. *Specification and Transformation of Programs - a Formal Approach to Software Development*. Springer-Verlag, Berlin, 1990.

[PB91] H.A. Partsch and E.A. Boiten. A note on similarity of specifications and reusability of transformational developments. In [Möl91], pages 71–89.

[SL90] D.R. Smith and M.R. Lowry. Algorithm theories and design tactics. *Science of Computer Programming*, 14:305–321, 1990.

A Bag Comprehension and Calculation Rules

As described in [Boi91a], bag comprehension is an operation that can only be soundly used in a limited number of cases. For set comprehensions, one can safely write $\{E(x) \mid P(x)\}$; for list comprehensions one is forced to give a (list) domain for the comprehension variable (e.g., $[E(x) \mid x \leftarrow l \wedge P(x)]$); implicitly, such a domain must be present for bag comprehensions as well. "Sound" bag comprehensions are defined by the following rules (B and B_i denote bags):

$$\{\![\, x \mid x \in B \wedge P(x) \,]\!\} = P \lhd B \tag{5}$$

$$\{\![\, x \mid \mathbf{false} \,]\!\} = \emptyset \tag{6}$$

$$\begin{aligned} \{\![\, x \mid \exists y : Q(y,x) \,]\!\} = \biguplus_y (& \mathbf{if}\ \exists\, x : Q(y,x) \\ & \mathbf{then}\ \{\![\, \mathbf{that}\ x : Q(y,x) \,]\!\} \\ & \mathbf{else}\ \emptyset\ \mathbf{fi}) \\ \text{if} \quad & \forall x,y,z : (Q(y,x) \wedge Q(y,z)) \Rightarrow x = z \end{aligned} \tag{7}$$

$$\{\![\, f(x) \mid x \in B \,]\!\} = f * B \tag{8}$$

$$\{\![\, x \oplus y \mid x \in B_1 \wedge y \in B_2 \,]\!\} = B_1 \mathsf{X}_{\oplus}\ B_2. \tag{9}$$

Most of these rules will be used tacitly in derivations, in particular Rule 8 is often used in combination with another one. Some additional rules to be used are the following.

Bag comprehension over linear index domain. Provided m is of type $\mathbf{m}$, and $\mathbf{m}$ is linearly ordered by $\leq$, and $\{\![\, x \mid Q \,]\!\}$ is a sound bag comprehension, then

$$\{\![\, x \mid \exists\, m : a \leq m \leq b \wedge Q \,]\!\} = \biguplus_{m=a}^{b} \{\![\, x \mid Q \,]\!\}. \tag{10}$$

Bag comprehension over product domain. If $\{\![\, x \mid P(x,y,z) \,]\!\}$ is a sound bag comprehension for fixed y and z, then

$$\begin{aligned} & \{\![\, x \mid \exists\, y,z : y \in Y \wedge z \in Z \wedge P(x,y,z) \,]\!\} \\ & = \uplus/(Y\ \mathsf{X}_{\oplus}\ Z) \\ & \quad \mathbf{where}\ y \oplus z = \{\![\, x \mid P(x,y,z) \,]\!\}. \end{aligned} \tag{11}$$

Generalization of domain of comprehension variable. This rule is used for introducing bag comprehensions, in order to allow folding later on. For an injective

function f,

$$\begin{aligned} &\{\![\ E \mid \exists\, x : x \in X \wedge P(x) \wedge Q(x)]\!\} \\ &= \{\![\ E \mid \exists\, x' : x' \in \{\![\ f(x) \mid x \in X \wedge P(x)]\!\} \wedge Q(f^{-1}(x'))]\!\}. \end{aligned} \tag{12}$$