Formal Aspects of Computing (1995) 7: 683–703
© 1995 BCS

Conditional Composition

Rajit Manohar and K. Rustan M. Leino[1]

Department of Computer Science, Caltech, Pasadena, CA 91125, U.S.A.

Keywords: Function composition; Exceptions; Program semantics

Abstract. Generalizing the notion of function composition, we introduce the concept of *conditional* function composition and present a theory of such compositions. We use the theory to describe the semantics of a programming language with exceptions, and to relate exceptions to the *IF* statement.

1. INTRODUCTION

In this note, we generalize the concept of function composition. We introduce the notion of *conditional* composition of functions, and develop a theory of such compositions. We also introduce a conditional *replace* operator, which can be used to define conditional composition in terms of ordinary function application.

We show how these concepts can be used to define the semantics of a programming language with exceptions. In particular, we show how exceptions can be described in terms of the *IF* statement. This work generalizes the concepts introduced in [6].

2. THEORY OF CONDITIONAL COMPOSITION

We generalize the concept of function composition to *conditional* function composition. To enable us to do so, we use *tagged collections* (described next) which can be used to represent lists, arrays, or any other labeled collection of objects.

Our proof format is from [3]. We use [ ] for *everywhere* brackets, which denote

Correspondence and offprint requests to: Rajit Manohar, Department of Computer Science 256-80, Caltech, Pasadena, CA 91125, U.S.A. e-mail: `rajit@cs.caltech.edu`

[1] Supported in part by Air Force Office of Scientific Research grant number 91-0070. Now at DEC Systems Research Center, Palo Alto, CA.

universal quantification, and the infix dot to denote function application. The expression

$$\langle Q\ x \mid r.x \triangleright t.x \rangle$$

where x is an unordered list of identifier names (dummies), $r.x$ (the range) is a predicate, and $t.x$ (the term) is an expression of the appropriate type is used to denote quantification. When $r.x$ is true everywhere or understood, we omit the range and write the quantification as

$$\langle Q\ x \triangleright t.x \rangle \quad .$$

A common quantifier is the set constructor, written as

$$\{x \mid r.x \triangleright t.x\}$$

which denotes the set containing all the terms $t.x$ where x satisfies $r.x$.

We will describe tagged collections by example; a more precise, calculational treatment can be found in the appendix. An example one-element tagged collection is $(2\colon 4)$. It contains the number 4, *tagged* by 2. A tag and its associated element are delimited by a colon. Tagged collections are combined using operator $\sqcup$, called "tagged union." For example,

$$2\colon 4 \ \sqcup\ 3\colon 9 \ \sqcup\ 5\colon 25$$

shows the tags 2, 3, 5 associated with their respective squares. The set of tags is called the *index set*. Operator $\sqcup$ is associative and commutative, and is only defined when the index sets of its operands are disjoint. In this paper, we use (subsets of) a fixed index set, which we denote by $\mathcal{U}$.

Selection of an element in a tagged collection is denoted like function application. Given a tagged collection L, $L.i$ denotes the element whose tag is i. This operation is defined only when i is a member of the index set of L.

We use $Tag.D$ to denote the type of a tagged collection of elements from D with a fixed index set $\mathcal{U}$. Notice that the tagged collections in $Tag.D$ can be viewed as total functions from $\mathcal{U} \to D$.

We allow tagged collections to be written as quantified expressions. For example, a partial function p with range $R \subseteq \mathcal{U}$ can be written as

$$p = \langle \sqcup\, i \mid i \in R \triangleright i\colon p.i \rangle \quad .$$

2.1. FUNCTIONS AND COMPOSITION

Functions of type $Tag.D \to D$ for any domain D can be composed in different ways. We use f, g, and h to denote such functions. We use L to denote a tagged collection. Note that $L = \langle \sqcup\, i \triangleright i\colon L.i \rangle$.

We generalize the notion of function composition as follows. For any predicate p on $\mathcal{U}$, we define the conditional composition operator $\circ_p$, pronounced "compose p," as:

$$(f \circ_p g).L = f.(\langle \sqcup\, i \mid p.i \triangleright i\colon g.L \rangle \sqcup \langle \sqcup\, i \mid \neg p.i \triangleright i\colon L.i \rangle) \tag{1}$$

In words, $(f \circ_p g)$ applied to a tagged collection L applies f to the tagged collection obtained by replacing $L.k$ with $g.L$, for all k satisfying p.

Theorem (*Associativity*)

$$\text{If } [q \Rightarrow p], \text{ then } (f \circ_p g) \circ_q h = f \circ_p (g \circ_q h) \quad , \tag{2}$$

i.e., then $\circ_p$ and $\circ_q$ are mutually associative.

Proof.

$$\begin{array}{ll} & ((f \circ_p g) \circ_q h).L \\ = & \quad \{ \text{ (1): def. of } \circ_q \text{ } \} \\ & (f \circ_p g).(\langle \sqcup\, i \mid q.i \triangleright i\colon h.L\rangle \sqcup \langle \sqcup\, i \mid \neg q.i \triangleright i\colon L.i\rangle) \\ = & \quad \{ \text{ (1): def. of } \circ_p \text{ } \} \\ & f.(\langle \sqcup\, i \mid p.i \triangleright i\colon g.(\langle \sqcup\, i \mid q.i \triangleright i\colon h.L\rangle \sqcup \langle \sqcup\, i \mid \neg q.i \triangleright i\colon L.i\rangle)\rangle \sqcup \\ & \qquad \langle \sqcup\, i \mid \neg p.i \triangleright i\colon (\langle \sqcup\, i \mid q.i \triangleright i\colon h.L\rangle \sqcup \langle \sqcup\, i \mid \neg q.i \triangleright i\colon L.i\rangle).i\rangle) \\ = & \quad \{ \text{ (68): projection, since } \neg p.i \Rightarrow \neg q.i \text{ } \} \\ & f.(\langle \sqcup\, i \mid p.i \triangleright i\colon g.(\langle \sqcup\, i \mid q.i \triangleright i\colon h.L\rangle \sqcup \langle \sqcup\, i \mid \neg q.i \triangleright i\colon L.i\rangle)\rangle \sqcup \\ & \qquad \langle \sqcup\, i \mid \neg p.i \triangleright i\colon L.i\rangle) \\ = & \quad \{ \text{ (1): def. of } \circ_q \text{ } \} \\ & f.(\langle \sqcup\, i \mid p.i \triangleright i\colon (g \circ_q h).L\rangle \sqcup \langle \sqcup\, i \mid \neg p.i \triangleright i\colon L.i\rangle) \\ = & \quad \{ \text{ (1): def. of } \circ_p \text{ } \} \\ & (f \circ_p (g \circ_q h)).L \end{array}$$

□

Theorem (*Associativity*)

$$\circ_p \text{ is associative.} \tag{3}$$

Proof. Follows from (2) and $[p \Rightarrow p]$.

□

A function of special interest is the *projection* function, defined as

$$\restriction_k.L = L.k \tag{4}$$

and pronounced as "project k". This function can be used to extract a value associated with a particular tag from a tagged collection. The next few theorems show the effect of composing a function with a projection function.

Theorem (*Left identity*)

$$\restriction_k \text{ is left identity of } \circ_p, \text{ for all } k \text{ satisfying } p. \tag{5}$$

Proof.

$$\begin{array}{ll} & (\restriction_k \circ_p f).L \\ = & \quad \{ \text{ (1): def. of } \circ_p \text{ } \} \\ & \restriction_k.(\langle \sqcup\, i \mid p.i \triangleright i\colon f.L\rangle \sqcup \langle \sqcup\, i \mid \neg p.i \triangleright i\colon L.i\rangle) \\ = & \quad \{ \text{ (4): def. of } \restriction_k \text{ } \} \\ & (\langle \sqcup\, i \mid p.i \triangleright i\colon f.L\rangle \sqcup \langle \sqcup\, i \mid \neg p.i \triangleright i\colon L.i\rangle).k \\ = & \quad \{ \text{ (68): projection, since } p.k \text{ } \} \\ & f.L \end{array}$$

□

It follows that $\circ_p$ has no right identity if p is satisfied by more than one k since the projection functions are distinct.

Theorem (*Left zero*)

$$\restriction_k \text{ is the left zero of } \circ_p, \text{ for all } k \text{ satisfying } \neg p. \tag{6}$$

Proof.

$$\begin{array}{ll} & (\restriction_k \circ_p f).L \\ = & \quad \{\ (1)\text{: def. of } \circ_p\ \} \\ & \restriction_k.(\langle \sqcup\ i \mid p.i \triangleright i\text{:}\ f.L\rangle \sqcup \langle \sqcup\ i \mid \neg p.i \triangleright i\text{:}\ L.i\rangle) \\ = & \quad \{\ (4)\text{: def. of } \restriction_k\ \} \\ & (\langle \sqcup\ i \mid p.i \triangleright i\text{:}\ f.L\rangle \sqcup \langle \sqcup\ i \mid \neg p.i \triangleright i\text{:}\ L.i\rangle).k \\ = & \quad \{\ (68)\text{: projection, since } \neg p.k\ \} \\ & L.k \\ = & \quad \{\ (4)\text{: def. of } \restriction_k\ \} \\ & \restriction_k.L \end{array}$$

□

It follows that $\circ_p$ has no right zero if $\neg p$ holds for more than one k since the projection functions are distinct.

Theorem

$$(f \circ_p \restriction_k) \circ_q g = f \circ_{p \vee q} g, \text{ for all } k \text{ satisfying } q. \tag{7}$$

Proof.

$$\begin{array}{ll} & ((f \circ_p \restriction_k) \circ_q g).L \\ = & \quad \{\ (1)\text{: def. of } \circ_q\ \} \\ & (f \circ_p \restriction_k).(\langle \sqcup\ i \mid q.i \triangleright i\text{:}\ g.L\rangle \sqcup \langle \sqcup\ i \mid \neg q.i \triangleright i\text{:}\ L.i\rangle) \\ = & \quad \{\ (1)\text{: def. of } \circ_p\ \} \\ & f.(\langle \sqcup\ i \mid p.i \triangleright i\text{:} \restriction_k.(\langle \sqcup\ i \mid q.i \triangleright i\text{:}\ g.L\rangle \sqcup \langle \sqcup\ i \mid \neg q.i \triangleright i\text{:}\ L.i\rangle)\rangle \sqcup \\ & \qquad \langle \sqcup\ i \mid \neg p.i \triangleright i\text{:} (\langle \sqcup\ i \mid q.i \triangleright i\text{:}\ g.L\rangle \sqcup \langle \sqcup\ i \mid \neg q.i \triangleright i\text{:}\ L.i\rangle).i\rangle) \\ = & \quad \{\ (1)\text{: def. of } \circ_q\ \} \\ & f.(\langle \sqcup\ i \mid p.i \triangleright i\text{:} (\restriction_k \circ_q g).L\rangle \sqcup \\ & \qquad \langle \sqcup\ i \mid \neg p.i \triangleright i\text{:} (\langle \sqcup\ i \mid q.i \triangleright i\text{:}\ g.L\rangle \sqcup \langle \sqcup\ i \mid \neg q.i \triangleright i\text{:}\ L.i\rangle).i\rangle) \\ = & \quad \{\ (5)\text{: left identity, since } q.k\ \} \\ & f.(\langle \sqcup\ i \mid p.i \triangleright i\text{:}\ g.L\rangle \sqcup \\ & \qquad \langle \sqcup\ i \mid \neg p.i \triangleright i\text{:} (\langle \sqcup\ i \mid q.i \triangleright i\text{:}\ g.L\rangle \sqcup \langle \sqcup\ i \mid \neg q.i \triangleright i\text{:}\ L.i\rangle).i\rangle) \\ = & \quad \{\ (65)\text{: range split}\ \} \\ & f.(\langle \sqcup\ i \mid p.i \triangleright i\text{:}\ g.L\rangle \sqcup \\ & \qquad \langle \sqcup\ i \mid \neg p.i \wedge q.i \triangleright i\text{:} (\langle \sqcup\ i \mid q.i \triangleright i\text{:}\ g.L\rangle \sqcup \langle \sqcup\ i \mid \neg q.i \triangleright i\text{:}\ L.i\rangle).i\rangle \sqcup \\ & \qquad \langle \sqcup\ i \mid \neg p.i \wedge \neg q.i \triangleright i\text{:} (\langle \sqcup\ i \mid q.i \triangleright i\text{:}\ g.L\rangle \sqcup \langle \sqcup\ i \mid \neg q.i \triangleright i\text{:}\ L.i\rangle).i\rangle) \\ = & \quad \{\ (68)\text{: projection, twice}\ \} \\ & f.(\langle \sqcup\ i \mid p.i \triangleright i\text{:}\ g.L\rangle \sqcup \langle \sqcup\ i \mid \neg p.i \wedge q.i \triangleright i\text{:}\ g.L\rangle \sqcup \\ & \qquad \langle \sqcup\ i \mid \neg p.i \wedge \neg q.i \triangleright i\text{:}\ L.i\rangle) \\ = & \quad \{\ (65)\text{: range split}\ \} \\ & f.(\langle \sqcup\ i \mid p.i \vee (\neg p.i \wedge q.i) \triangleright i\text{:}\ g.L\rangle \sqcup \end{array}$$

$$\begin{array}{ll} & \langle \sqcup\ i \mid \neg p.i \wedge \neg q.i \triangleright i{:}\, L.i \rangle) \\ = & \quad \{ \text{ pred. calc. } \} \\ & f.(\langle \sqcup\ i \mid (p \vee q).i \triangleright i{:}\, g.L \rangle \sqcup \\ & \quad \langle \sqcup\ i \mid \neg(p \vee q).i \triangleright i{:}\, L.i \rangle) \\ = & \quad \{ \text{ (1): def. of } \circ_{p \vee q} \} \\ & (f \circ_{p \vee q} g).L \end{array}$$

□

2.1.1. POINT PREDICATES

Let $[\![k]\!]$ be the predicate that is true only when $x = k$, i.e., $[\![k]\!].x \equiv [x = k]$. $[\![k]\!]$ is a *point predicate* since it is true at exactly one point in $\mathcal{U}$.

Theorem

$$\begin{array}{l} f \circ_p \restriction_k = f \circ_{p \vee [\![k]\!]} \restriction_k \\ f \circ_p \restriction_k = f \circ_{p \wedge \neg [\![k]\!]} \restriction_k \end{array} \tag{8}$$

Proof. Note that the two lines given above express the same property (with different values for p). We prove the former.

$$\begin{array}{ll} & (f \circ_{p \vee [\![k]\!]} \restriction_k).L \\ = & \quad \{ \text{ (1): def. of } \circ_{p \vee [\![k]\!]}\text{; (4): def. of } \restriction_k \ \} \\ & f.(\langle \sqcup\ i \mid p.i \vee [\![k]\!].i \triangleright i{:}\, L.k \rangle \sqcup \langle \sqcup\ i \mid \neg p.i \wedge \neg [\![k]\!].i \triangleright i{:}\, L.i \rangle) \\ = & \quad \{ \text{ pred. calc.; (65): range split } \} \\ & f.(\langle \sqcup\ i \mid p.i \wedge \neg [\![k]\!].i \triangleright i{:}\, L.k \rangle \sqcup \langle \sqcup\ i \mid \neg p.i \wedge \neg [\![k]\!].i \triangleright i{:}\, L.i \rangle \sqcup \\ & \quad \langle \sqcup\ i \mid [\![k]\!].i \triangleright i{:}\, L.k \rangle) \\ = & \quad \{ \text{ (65): range split; (66): transform term;} \\ & \quad \text{ def. of point predicate; pred. calc. } \} \\ & f.(\langle \sqcup\ i \mid p.i \wedge \neg [\![k]\!].i \triangleright i{:}\, L.k \rangle \sqcup \langle \sqcup\ i \mid \neg p.i \wedge \neg [\![k]\!].i \triangleright i{:}\, L.i \rangle \sqcup \\ & \quad \langle \sqcup\ i \mid [\![k]\!].i \wedge p.i \triangleright i{:}\, L.k \rangle \sqcup \langle \sqcup\ i \mid [\![k]\!].i \wedge \neg p.i \triangleright i{:}\, L.i \rangle) \\ = & \quad \{ \text{ (65): range split, twice; pred. calc. } \} \\ & f.(\langle \sqcup\ i \mid p.i \triangleright i{:}\, L.k \rangle \sqcup \langle \sqcup\ i \mid \neg p.i \triangleright i{:}\, L.i \rangle) \\ = & \quad \{ \text{ (1): def. of } \circ_p\text{; (4): def. of } \restriction_k \ \} \\ & (f \circ_p \restriction_k).L \end{array}$$

□

We now have,

Theorem

$$f \circ_{false} g = f \tag{9}$$

Proof.

$$\begin{array}{ll} & (f \circ_{false} g).L \\ = & \quad \{ \text{ (1): def. of } \circ_{false} \} \\ & f.(\langle \sqcup\ i \mid false.i \triangleright i{:}\, g.L \rangle \sqcup \langle \sqcup\ i \mid \neg false.i \triangleright i{:}\, L.i \rangle) \end{array}$$

$$
\begin{array}{ll}
= & \quad \{ \text{ (63): empty range } \} \\
& f.(\emptyset \sqcup \langle \sqcup\, i \triangleright i \colon L.i \rangle) \\
= & \quad \{ \emptyset \text{ is the identity of } \sqcup \} \\
& f.L
\end{array}
$$

□

We mentioned earlier that $\circ_p$ had no right identity if p holds for more than one point. However, when p is a point predicate, we have that

Theorem (*Right identity*)

$$\lceil_k \text{ is the right identity of } \circ_{[\![k]\!]}. \tag{10}$$

Proof.

$$
\begin{array}{ll}
& f \circ_{[\![k]\!]} \lceil_k \\
= & \quad \{ \text{ (8), since } [\mathit{false} \vee [\![k]\!] \equiv [\![k]\!]] \ \} \\
& f \circ_{\mathit{false}} \lceil_k \\
= & \quad \{ \text{ (9) } \} \\
& f
\end{array}
$$

□

We introduce some special notation, $\lfloor\ \rfloor_k$, defined as follows

$$\lfloor f \rfloor_k = f \circ_{\mathit{true}} \lceil_k \tag{11}$$

Theorem

$$\lfloor f \rfloor_k = f \circ_{\neg [\![k]\!]} \lceil_k \tag{12}$$

Proof.

$$
\begin{array}{ll}
& f \circ_{\neg [\![k]\!]} \lceil_k \\
= & \quad \{ \text{ (8) } \} \\
& f \circ_{\neg [\![k]\!] \vee [\![k]\!]} \lceil_k \\
= & \quad \{ \text{ pred. calc. } \} \\
& f \circ_{\mathit{true}} \lceil_k \\
= & \quad \{ \text{ (11): def. of } \lfloor\ \rfloor_k \ \} \\
& \lfloor f \rfloor_k
\end{array}
$$

□

Theorem

$$\lfloor \lceil_k \rfloor_j = \lceil_j \tag{13}$$

Proof.

$$
\begin{array}{ll}
& \lfloor \lceil_k \rfloor_j \\
= & \quad \{ \text{ (11): def. of } \lfloor\ \rfloor_j \ \} \\
& \lceil_k \circ_{\mathit{true}} \lceil_j \\
= & \quad \{ \text{ (5): left identity, since } \mathit{true}.k, \text{ for any } k \ \} \\
& \lceil_j
\end{array}
$$

□

Theorem

$$\lfloor \lfloor f \rfloor_k \rfloor_j = \lfloor f \rfloor_j \tag{14}$$

Proof.

$$\begin{array}{ll} & \lfloor \lfloor f \rfloor_k \rfloor_j \\ = & \quad \{ \text{ (11): def. of } \lfloor\ \rfloor_j \text{ } \} \\ & \lfloor f \rfloor_k \circ_{true} \lceil_j \\ = & \quad \{ \text{ (11): def. of } \lfloor\ \rfloor_k \text{; (3): associativity } \} \\ & f \circ_{true} \lceil_k \circ_{true} \lceil_j \\ = & \quad \{ \text{ (5): left identity, since } true.k \text{, for any } k \text{ } \} \\ & f \circ_{true} \lceil_j \\ = & \quad \{ \text{ (11): def. of } \lfloor\ \rfloor_j \text{ } \} \\ & \lfloor f \rfloor_j \end{array}$$

□

Theorem

$$\lfloor\ \rfloor_k \text{ is idempotent} \tag{15}$$

Proof. Follows from (14), with $j := k$. □

Theorem

$$\lfloor f \circ_{true} g \rfloor_k = \lfloor f \rfloor_k \circ_{true} \lfloor g \rfloor_k \tag{16}$$

Proof.

$$\begin{array}{ll} & \lfloor f \circ_{true} g \rfloor_k \\ = & \quad \{ \text{ (11): def. of } \lfloor\ \rfloor_k \text{; (3): associativity } \} \\ & f \circ_{true} g \circ_{true} \lceil_k \\ = & \quad \{ \text{ (5): left identity, since } true.k \text{ for any } k \text{ } \} \\ & f \circ_{true} \lceil_k \circ_{true} g \circ_{true} \lceil_k \\ = & \quad \{ \text{ (11): def. of } \lfloor\ \rfloor_k \text{, twice; (3): associativity } \} \\ & \lfloor f \rfloor_k \circ_{true} \lfloor g \rfloor_k \end{array}$$

□

2.1.2. DUALITY

Theorem (7) shows how we can define $\circ_{p \vee q}$ in terms of $\circ_p$ and $\circ_q$ (for most q), but we cannot write $\circ_{p \wedge q}$ in terms of $\circ_p$ and $\circ_q$ and the operators we have introduced so far. However, if we had a way to define $\circ_{\neg p}$ in terms of $\circ_p$, then we could obtain $p \wedge q$ from $p \vee q$. We do not know how to do this, but instead introduce a new operator $\bullet_p$, defined by

$$\bullet_p = \circ_{\neg p} \quad . \tag{17}$$

Theorem (*dual of (7)*)

$$(f \bullet_p \restriction_k) \bullet_q g = f \bullet_{p \wedge q} g, \text{ for all } k \text{ satisfying } \neg q. \tag{18}$$

Proof.

$$\begin{array}{ll} & (f \bullet_p \restriction_k) \bullet_q g \\ = & \quad \{ \text{ (17): def. of } \bullet, \text{ twice } \} \\ & (f \circ_{\neg p} \restriction_k) \circ_{\neg q} g \\ = & \quad \{ \text{ (7), with } p, q := \neg p, \neg q, \text{ since } \neg q.k \ \} \\ & f \circ_{\neg p \vee \neg q} g \\ = & \quad \{ \text{ De Morgan } \} \\ & f \circ_{\neg(p \wedge q)} g \\ = & \quad \{ \text{ (17): def. of } \bullet_{p \wedge q} \ \} \\ & f \bullet_{p \wedge q} g \end{array}$$

□

We have additional properties of $\bullet_p$ which follow from those of $\circ_p$.

Theorem (*Left zero*)

$$\restriction_k \text{ is left zero of } \bullet_p \text{ for all } k \text{ satisfying } p. \tag{19}$$

Proof. Follows from (6), (17). □

Theorem (*Left identity*)

$$\restriction_k \text{ is left identity of } \bullet_p \text{ for all } k \text{ satisfying } \neg p. \tag{20}$$

Proof. Follows from (5), (17). □

Theorem (*Right identity*)

$$\restriction_k \text{ is the right identity of } \bullet_{\neg [\![k]\!]} \tag{21}$$

Proof. Follows from (10), (17). □

Theorem

$$\begin{array}{l} f \bullet_p \restriction_k = f \bullet_{p \vee [\![k]\!]} \restriction_k \\ f \bullet_p \restriction_k = f \bullet_{p \wedge \neg [\![k]\!]} \restriction_k \end{array} \tag{22}$$

Proof. Follow from (8), (17). □

Since $\restriction_k$ is the right identity of $\circ_{[\![k]\!]}$, we examine the effect of $\bullet_{[\![k]\!]} \restriction_k$.

Theorem

$$f \bullet_{[\![k]\!]} \restriction_k = f \circ_{true} \restriction_k \tag{23}$$

Proof. Follows from (11), (12), (17). □

2.1.3. MONOTONICITY

We explore additional properties of $\circ_p$ for monotonic functions. We assume that there exists some partial ordering $\sqsubseteq$ defined on the elements of D. We extend this order to $Tag.D$ by the following definition:

$$\langle \sqcup\, i \mid p.i \triangleright i\!:\! t.i \rangle \sqsubseteq \langle \sqcup\, i \mid p.i \triangleright i\!:\! u.i \rangle \equiv \langle \forall\, i \mid p.i \triangleright t.i \sqsubseteq u.i \rangle \tag{24}$$

This ordering is monotonic in each member of $Tag.D$. We now have the following theorems:

Theorem (*Monotonicity*)

$$f \circ_p g \text{ is monotonic in } f. \tag{25}$$

Proof.

$$\begin{array}{ll} & (f \circ_p g).L \sqsubseteq (f' \circ_p g).L \\ = & \quad \{\ (1)\text{: def. of } \circ_p\text{, twice }\} \\ & f.(\langle \sqcup\ i \mid p.i \triangleright i\colon g.L\rangle \sqcup \langle \sqcup\ i \mid \neg p.i \triangleright i\colon L.i\rangle) \sqsubseteq \\ & \qquad f'.(\langle \sqcup\ i \mid p.i \triangleright i\colon g.L\rangle \sqcup \langle \sqcup\ i \mid \neg p.i \triangleright i\colon L.i\rangle) \\ \Leftarrow & \\ & f \sqsubseteq f' \end{array}$$

□

Theorem (*Monotonicity*)

$f \circ_p g$ is monotonic in g, for monotonic f.

Proof.

$$\begin{array}{ll} & (f \circ_p g).L \sqsubseteq (f \circ_p g').L \\ = & \quad \{\ (1)\text{: def. of } \circ_p\text{, twice }\} \\ & f.(\langle \sqcup\ i \mid p.i \triangleright i\colon g.L\rangle \sqcup \langle \sqcup\ i \mid \neg p.i \triangleright i\colon L.i\rangle) \sqsubseteq \\ & \qquad f.(\langle \sqcup\ i \mid p.i \triangleright i\colon g'.L\rangle \sqcup \langle \sqcup\ i \mid \neg p.i \triangleright i\colon L.i\rangle) \\ \Leftarrow & \quad \{\ f \text{ is monotonic }\} \\ & (\langle \sqcup\ i \mid p.i \triangleright i\colon g.L\rangle \sqcup \langle \sqcup\ i \mid \neg p.i \triangleright i\colon L.i\rangle) \sqsubseteq \\ & \qquad (\langle \sqcup\ i \mid p.i \triangleright i\colon g'.L\rangle \sqcup \langle \sqcup\ i \mid \neg p.i \triangleright i\colon L.i\rangle) \\ = & \quad \{\ (24)\text{: } \sqsubseteq \text{ on } Tag.D\ \} \\ & \langle \sqcup\ i \mid p.i \triangleright i\colon g.L\rangle \sqsubseteq \langle \sqcup\ i \mid p.i \triangleright i\colon g'.L\rangle \\ = & \quad \{\ (24)\text{: } \sqsubseteq \text{ on } Tag.D\ \} \\ & \langle \forall\ i \mid p.i \triangleright g.L \sqsubseteq g'.L\rangle \\ \Leftarrow & \quad \{\ \text{calculus }\} \\ & g.L \sqsubseteq g'.L \\ \Leftarrow & \\ & g \sqsubseteq g' \end{array}$$

□

2.2. EXTENSION TO OTHER OPERATORS

So far we have examined the effect of extending $\circ$ to $\circ_p$. We now demonstrate that we can define $\circ_p$ in terms of function application given an additional operator. The new operator can then be used to extend other operators to tagged collections.

To be able to define $\circ_p$ in terms of application, notice that the $\circ_p$ operator

performs a selective substitution. Motivated by this, we define $\blacklozenge p \triangleright$ —a conditional *replace* operator— and its dual $\triangleleft p \blacklozenge$ as follows:

$$\langle \sqcup\, i \triangleright i\colon s.i \rangle \ \blacklozenge p \triangleright\ \langle \sqcup\, i \triangleright i\colon t.i \rangle = \langle \sqcup\, i \mid \neg p.i \triangleright i\colon s.i \rangle \sqcup \langle \sqcup\, i \mid p.i \triangleright i\colon t.i \rangle \quad (26)$$

$$\langle \sqcup\, i \triangleright i\colon s.i \rangle \ \triangleleft p \blacklozenge\ \langle \sqcup\, i \triangleright i\colon t.i \rangle = \langle \sqcup\, i \mid p.i \triangleright i\colon s.i \rangle \sqcup \langle \sqcup\, i \mid \neg p.i \triangleright i\colon t.i \rangle \quad (27)$$

We can define $\circ_p$ in terms of $\circ$ using the $\blacklozenge p \triangleright$ operator. To do so, we define $\mathcal{C}.u$ to be the tagged collection $\langle \sqcup\, i \triangleright i\colon u \rangle$. We now have:

$$(f \circ_p g).L = f.(L \ \blacklozenge p \triangleright\ \mathcal{C}.(g.L)) \quad (28)$$

$$(f \bullet_p g).L = f.(L \ \triangleleft p \blacklozenge\ \mathcal{C}.(g.L)) \quad (29)$$

(Expressions similar to, but different from, the right-hand sides of (28) and (29) can be applied in the relational database calculus. See [8] for details.)

3. APPLICATION TO PROGRAM SEMANTICS

We describe a program semantics with an additional conditional composition operator $\triangleleft_p$, where p is a predicate on $\mathcal{U}$. Imposing a structure on the predicates used for the weakest precondition semantics allows us to identify the composition operators presented earlier with the composition of statements. We follow the path of [6] and [7] which first describe an operational semantics in terms of traces and then derive a weakest precondition semantics from it.

3.1. TRACE SEMANTICS

We will use the semantics of statements presented in [7] and [4]. We use $\mathcal{X}$ to denote the (non-empty) state space. Variables that refer to the state are implicitly assumed to be from $\mathcal{X}$. Traces are nonempty sequences of states. *fin.t* is true just when t is a trace of finite length. *inf.t* is the negation of *fin.t*. Concatenation of traces is denoted by juxtaposition. For a trace t, *first.t* denotes the first state from the trace. When the trace is finite, *last.t* denotes the last state in the trace. Using these notational conventions, we have the following trace semantics for *skip* and assignment.

$$skip = \{x \triangleright x\} \quad (30)$$

$$(v := E) = \{x \triangleright x\, x[v := E]\} \quad (31)$$

For sequential composition, we have [7],

$$S;T = \{s, x, t \mid sx \in S \wedge xt \in T \wedge \mathit{fin}.s \triangleright sxt\} \cup \{s \mid s \in S \wedge \mathit{inf}.s \triangleright s\} \quad (32)$$

We assume that there exists a total function $idx\colon \mathcal{X} \to \mathcal{U}$. This function defines a partition on $\mathcal{X}$ as follows. Every state from $\mathcal{X}$ corresponds to a unique index from $\mathcal{U}$ which can be obtained by applying idx to the state. We introduce a conditional composition operator $\triangleleft_p$ (pronounce "try", as in triangle) —where p is a predicate on $\mathcal{U}$— that has the following operational meaning. We write $S \triangleleft_p T$ for the statement whose execution consists of: executing S; if execution thereof terminates in a state x satisfying $(p \circ idx)$, then executing T as well; if S does not terminate, or S terminates in a state x satisfying $\neg(p \circ idx)$, then not executing T. The trace semantics can be described by

$$S \lhd_p T = \{s, x, t \mid sx \in S \wedge xt \in T \wedge \mathit{fin}.s \wedge (p \circ \mathit{idx}).x \rhd sxt\} \cup \{s \mid s \in S \wedge (\mathit{inf}.s \vee \neg(p \circ \mathit{idx}).(\mathit{last}.s)) \rhd s\} \quad (33)$$

Analogous to $\bullet_p$, we define $\blacktriangleleft_p$ to be $\lhd_{\neg p}$.

Theorem

$$; \; = \lhd_{true} \quad (34)$$

Proof. Follows from the definition of $\lhd_{true}$. □

Theorem (*Identity of ;*)

skip is the left and right identity of ; (35)

Proof. Can be taken without modification from [7]. □

The following are some theorems about semicolon.

Theorem (From [4], [7].)

; is associative. (36)

; is universally $\cup$-distributive in its left argument. (37)

; is positively $\cup$-distributive in its right argument. (38)

We can use the definition of repetition and selection from [4] or [7] without modification since the trace semantics for semicolon has not been changed. We use the definition of the *IF* statement given in [4]. Since we have defined all our earlier semantics in terms of trace sets, we rewrite the definition presented in [4] in the following manner. For any predicate b, we define $b?$ to be

$$b? = \{x \mid x \in \mathcal{X} \wedge b.x \rhd x\} \quad . \quad (39)$$

Using this definition of $b?$, we define the *IF* statement from [4] by

$$\mathbf{if} \langle [\!] \; i \rhd b_i \rightarrow S_i \rangle \mathbf{fi} = \langle \cup \; i \rhd b_i?; S_i \rangle \cup \langle \forall \; i \rhd \neg b_i \rangle?; E \quad (40)$$

where E is defined as the set of all infinite traces (in [4] called "eternal" traces). If we know that $\langle \exists \; i \rhd b_i \rangle$ holds, then we can simplify (40) to

$$\mathbf{if} \langle [\!] \; i \rhd b_i \rightarrow S_i \rangle \mathbf{fi} = \langle \cup \; i \rhd b_i?; S_i \rangle \quad . \quad (41)$$

Using these trace semantics, we have the following theorem, which demonstrates how the semantics of the *IF* statement can be used to describe conditional composition.

Theorem (*Conditional Composition*)

$$S \lhd_p T = S; \mathbf{if} \; (p \circ \mathit{idx}) \rightarrow T \; [\!] \; \neg(p \circ \mathit{idx}) \rightarrow \mathit{skip} \; \mathbf{fi} \quad (42)$$

Proof. For this proof, we use $\mathit{IF} = \mathbf{if} \; (p \circ \mathit{idx}) \rightarrow T \; [\!] \; \neg(p \circ \mathit{idx}) \rightarrow \mathit{skip} \; \mathbf{fi}$

$S; \mathit{IF}$

$=$ { (32): def. of ; }

$\{s, x, t \mid sx \in S \wedge xt \in \mathit{IF} \wedge \mathit{fin}.s \rhd sxt\} \cup \{s \mid s \in S \wedge \mathit{inf}.s \rhd s\}$

$=$ { (41); (40): def. of *IF* }

$\{s, x, t \mid sx \in S \wedge xt \in ((p \circ \mathit{idx})?; T) \cup (\neg(p \circ \mathit{idx})?; \mathit{skip}) \wedge \mathit{fin}.s \rhd sxt\} \cup$
$\{s \mid s \in S \wedge \mathit{inf}.s \rhd s\}$

$$
\begin{array}{ll}
= & \{ \text{ range split } \} \\
& \{s, x, t \mid sx \in S \wedge xt \in ((p \circ idx)?; T) \wedge \mathit{fin}.s \triangleright sxt\} \cup \\
& \{s, x, t \mid sx \in S \wedge xt \in (\neg(p \circ idx)?; \mathit{skip}) \wedge \mathit{fin}.s \triangleright sxt\} \cup \\
& \{s \mid s \in S \wedge \mathit{inf}.s \triangleright s\} \\
= & \{ \text{ (39): def. of } b? \text{ twice } \} \\
& \{s, x, t \mid sx \in S \wedge xt \in T \wedge \mathit{fin}.s \wedge (p \circ idx).x \triangleright sxt\} \cup \\
& \{s, x, t \mid sx \in S \wedge xt \in \mathit{skip} \wedge \mathit{fin}.s \wedge \neg(p \circ idx).x \triangleright sxt\} \cup \\
& \{s \mid s \in S \wedge \mathit{inf}.s \triangleright s\} \\
= & \{ \text{ (30): def. of } \mathit{skip};\ sx := s \ \} \\
& \{s, x, t \mid sx \in S \wedge xt \in T \wedge \mathit{fin}.s \wedge (p \circ idx).x \triangleright sxt\} \cup \\
& \{s \mid s \in S \wedge \mathit{fin}.s \wedge \neg(p \circ idx).(\mathit{last}.s) \triangleright s\} \cup \{s \mid s \in S \wedge \mathit{inf}.s \triangleright s\} \\
= & \{ \text{ range split } \} \\
& \{s, x, t \mid sx \in S \wedge xt \in T \wedge \mathit{fin}.s \wedge (p \circ idx).x \triangleright sxt\} \cup \\
& \{s \mid s \in S \wedge (\mathit{inf}.s \vee (\mathit{fin}.s \wedge \neg(p \circ idx).(\mathit{last}.s))) \triangleright s\} \\
= & \{ \text{ absorption, since } \mathit{fin}.s \wedge \mathit{inf}.s = \mathit{false} \ \} \\
& \{s, x, t \mid sx \in S \wedge xt \in T \wedge \mathit{fin}.s \wedge (p \circ idx).x \triangleright sxt\} \cup \\
& \{s \mid s \in S \wedge (\mathit{inf}.s \vee \neg(p \circ idx).(\mathit{last}.s)) \triangleright s\} \\
= & \{ \text{ (33): def. of } \lhd_p \ \} \\
& S \lhd_p T
\end{array}
$$

□

Operator $\lhd_p$ also enjoys the following properties.

Theorem

$\lhd_p$ is associative (43)

Proof. Follows from (42), (41), and properties of ; . □

Theorem

$\lhd_p$ is universally $\cup$-distributive in its left argument. (44)

Proof. Follows from (42) and (37). □

Theorem

$\lhd_p$ is positively $\cup$-distributive in its right argument.

Proof. Follows from (42) and (36), (38), (41). □

3.2. WEAKEST PRECONDITIONS

We define function $wp.S.Q$ to be the weakest condition on the initial state such that: execution of S terminates; on termination Q holds [3]. We examine the weakest precondition semantics for conditional composition. We use the usual wp semantics from [3] using the definition of wp presented in [7] in terms of traces.

$$wp.S.Q.x = \langle \forall\ t \mid \mathit{first}.t = x \wedge t \in S \triangleright \mathit{fin}.t \wedge Q.(\mathit{last}.t) \rangle \tag{45}$$

We confine our attention to predicates Q that are *partitioned predicates*. Let $t = \langle \sqcup\ i \mid i \in \mathcal{U} \triangleright i{:}t.i \rangle$ be a tagged collection of predicates. We associate partition i with predicate $t.i$. We define the partitioned predicate $\mathcal{I}.t$ represented in terms of t to be

$$[\mathcal{I}.t \equiv \langle \forall\ i \triangleright (\llbracket i \rrbracket \circ idx) \Rightarrow t.i \rangle] \tag{46}$$

That is, $\mathcal{I}.t$ holds in every state x from some partition $i \in \mathcal{U}$ whenever $t.i$ holds of x. (This generalizes *pair predicates* in [6].) From now on, we use tagged collections and predicates interchangeably. Using this definition of a predicate, we can explore properties of wp and $\lhd_p$. We first examine some additional properties that relate $\llparenthesis p \rrparenthesis$, $\mathcal{C}$, and partitioned predicates.

Theorem
For a partitioned predicate Q,

$$[Q \equiv \mathcal{C}.Q] \tag{47}$$

Proof.

$\mathcal{C}.Q$
$=$ { def. of $\mathcal{C}$ }
$\langle \sqcup\ i \triangleright i{:}\,Q \rangle$
$=$ { (46), since $\langle \sqcup\ i \triangleright i{:}\,Q \rangle$ is a partitioned predicate }
$\langle \forall\ i \triangleright (\llbracket i \rrbracket \circ idx) \Rightarrow Q \rangle$
$=$ { (46), since Q is a partitioned predicate }
$\langle \forall\ i \triangleright (\llbracket i \rrbracket \circ idx) \Rightarrow \langle \forall\ j \triangleright (\llbracket j \rrbracket \circ idx) \Rightarrow Q.j \rangle \rangle$
$=$ { $\Rightarrow$ over $\forall$ }
$\langle \forall\ i \triangleright \langle \forall\ j \triangleright (\llbracket i \rrbracket \circ idx) \Rightarrow ((\llbracket j \rrbracket \circ idx) \Rightarrow Q.j) \rangle \rangle$
$=$ { interchange of dummies }
$\langle \forall\ j \triangleright \langle \forall\ i \triangleright (\llbracket i \rrbracket \circ idx) \Rightarrow ((\llbracket j \rrbracket \circ idx) \Rightarrow Q.j) \rangle \rangle$
$=$ { pred. calc. }
$\langle \forall\ j \triangleright \langle \exists\ i \triangleright \llbracket i \rrbracket \circ idx \rangle \Rightarrow ((\llbracket j \rrbracket \circ idx) \Rightarrow Q.j) \rangle$
$=$ { idx is total }
$\langle \forall\ j \triangleright (\llbracket j \rrbracket \circ idx) \Rightarrow Q.j \rangle$
$=$ { (46) }
Q

□

From (47), we can see that the set of partitioned predicates is the same as the set of ordinary predicates. Therefore, a partitioned predicate is simply a predicate that has some structure.

Theorem

$$[(p \circ idx) \equiv \langle \exists\ i \mid p.i \triangleright \llbracket i \rrbracket \circ idx \rangle] \tag{48}$$

Proof.

$\langle \exists\ i \mid p.i \triangleright (\llbracket i \rrbracket \circ idx).x \rangle$
$=$ { trading }

$$
\begin{array}{ll}
 & \langle \exists\ i \mid (\llbracket i \rrbracket \circ idx).x \triangleright p.i \rangle \\
= & \quad \{ \text{ def. of } \llbracket i \rrbracket \ \} \\
 & \langle \exists\ i \mid [idx.x = i] \triangleright p.i \rangle \\
= & \quad \{ \text{ one-point rule } \} \\
 & p.(idx.x) \\
= & \quad \{ \text{ def. of } \circ \ \} \\
 & (p \circ idx).x
\end{array}
$$

□

We now examine the effect of $\llparenthesis p \rrparenthesis$ on partitioned predicates.

Theorem

$$[(Q\ \llparenthesis p \rrparenthesis\ R) \equiv (\neg(p \circ idx) \Rightarrow Q) \wedge ((p \circ idx) \Rightarrow R)] \tag{49}$$

Proof.

$$
\begin{array}{ll}
 & (\neg(p \circ idx) \Rightarrow Q) \wedge ((p \circ idx) \Rightarrow R) \\
= & \quad \{ \text{ (48), twice } \} \\
 & (\langle \exists\ i \mid \neg p.i \triangleright \llbracket i \rrbracket \circ idx \rangle \Rightarrow Q) \wedge (\langle \exists\ i \mid p.i \triangleright \llbracket i \rrbracket \circ idx \rangle \Rightarrow R) \\
= & \quad \{ \text{ pred. calc. } \} \\
 & \langle \forall\ i \mid \neg p.i \triangleright (\llbracket i \rrbracket \circ idx) \Rightarrow Q \rangle \wedge \langle \forall\ i \mid p.i \triangleright (\llbracket i \rrbracket \circ idx) \Rightarrow R \rangle \\
= & \quad \{ \text{ (26): def. of } \llparenthesis p \rrparenthesis \text{; (46) } \} \\
 & \langle \sqcup\ i \triangleright i{:}Q \rangle\ \llparenthesis p \rrparenthesis\ \langle \sqcup\ i \triangleright i{:}R \rangle \\
= & \quad \{ \text{ def. of } \mathcal{C}, \text{ twice } \} \\
 & \mathcal{C}.Q\ \llparenthesis p \rrparenthesis\ \mathcal{C}.R \\
= & \quad \{ \text{ (47), twice } \} \\
 & Q\ \llparenthesis p \rrparenthesis\ R
\end{array}
$$

□

We now examine some properties of *wp*. Since the traces of semicolon, *skip*, and assignment are the same as those in [7], all the results still hold. In particular, we have

Theorem (From [7].)

$$[wp.skip.Q \equiv Q] \tag{50}$$

$$[wp.(v := E).Q \equiv Q[v := E]] \tag{51}$$

$$[wp.(S;T).Q \equiv wp.S.(wp.T.Q)] \tag{52}$$

$$[wp.(\textbf{if}\ b0 \rightarrow S \,[\!]\, b1 \rightarrow T\ \textbf{fi}).Q \equiv (b0 \vee b1) \wedge (b0 \Rightarrow wp.S.Q) \wedge (b1 \Rightarrow wp.T.Q)] \tag{53}$$

We now examine the weakest precondition semantics for conditional composition.

Theorem

$$[wp.(S \lhd_p T).Q \equiv wp.S.(Q\ \llparenthesis p \rrparenthesis\ wp.T.Q)] \tag{54}$$

Proof.

$$
\begin{array}{ll}
 & wp.(S \lhd_p T).Q \\
= & \quad \{\ (42)\text{: } \lhd_p \text{ as } IF\ \} \\
 & wp.(S;\ \textbf{if}\ (p \circ idx) \rightarrow T \,[\!]\, \neg(p \circ idx) \rightarrow skip\ \textbf{fi}).Q \\
= & \quad \{\ (52)\text{: } wp \text{ of } ;\ \} \\
 & wp.S.(wp.\ \textbf{if}\ (p \circ idx) \rightarrow T \,[\!]\, \neg(p \circ idx) \rightarrow skip\ \textbf{fi}\ .Q) \\
= & \quad \{\ (53)\text{: } wp \text{ of } IF\ \} \\
 & wp.S.(((p \circ idx) \vee \neg(p \circ idx)) \wedge ((p \circ idx) \Rightarrow wp.T.Q) \wedge \\
 & \qquad (\neg(p \circ idx) \Rightarrow wp.skip.Q)) \\
= & \quad \{\ [X \vee \neg X];\ (50)\ \} \\
 & wp.S.(((p \circ idx) \Rightarrow wp.T.Q) \wedge (\neg(p \circ idx) \Rightarrow Q)) \\
= & \quad \{\ (49)\ \} \\
 & wp.S.(Q \ \blacklozenge p \blacklozenge\ wp.T.Q)
\end{array}
$$

□

We now relate $\lhd_p$ to the composition operator $\circ_p$.

Theorem

$$[wp.(S \lhd_p T) \equiv wp.S \circ_p wp.T] \tag{55}$$

Proof.

$$
\begin{array}{ll}
 & wp.(S \lhd_p T).Q \\
= & \quad \{\ (54)\text{: } wp \text{ of } \lhd_p\ \} \\
 & wp.S.(Q \ \blacklozenge p \blacklozenge\ wp.T.Q) \\
= & \quad \{\ (47)\ \} \\
 & wp.S.(Q \ \blacklozenge p \blacklozenge\ \mathcal{C}.(wp.T.Q)) \\
= & \quad \{\ (28)\text{: def. of } \circ_p \text{ using } \blacklozenge p \blacklozenge\ \} \\
 & (wp.S \circ_p wp.T).Q
\end{array}
$$

□

We conclude this section by identifying a program with its weakest precondition. We have the following correspondences:

$$skip = id$$

$$S \lhd_p T = S \circ_p T, \quad \text{or, written more concisely,}$$

$$\lhd_p = \circ_p$$

$$\blacktriangleleft_p = \bullet_p$$

4. APPLICATION TO PROGRAMMING LANGUAGES

The $\lhd_p$ operator presented above can be used in various ways. In this section, we suggest a disciplined approach to the introduction of such an operator into a programming language by using partitioned state spaces. We mention the partitions one can use to describe the semantics of exceptions [6], loop exits, and structured jumps [1].

We describe how the semantics of conditional composition can be used to

describe the semantics of a program given a partition of the state space. We identify an element of the partition with an element from the index set $\mathcal{U}$.

> *Restriction.* Each element of the partition of the state space has the same cardinality. Furthermore, there exists a well-defined mapping from one element of the partition to another. (*End of Restriction.*)

4.1. PARTITIONS

We assume that there exist functions $\lfloor\ \rfloor_k$ for $k \in \mathcal{U}$, which map states to states. These functions are assumed to satisfy the following two properties:

$$\langle \forall\ x, k \triangleright idx.\lfloor x \rfloor_k = k \rangle \tag{56}$$

$$\langle \forall\ x \triangleright \langle \forall\ k, j \triangleright \lfloor \lfloor x \rfloor_k \rfloor_j = \lfloor x \rfloor_j \rangle \rangle \tag{57}$$

Properties (56) and (57) guarantee that the partitions are of equal size. As before, $[\![i]\!] \circ idx$ can be used to determine if the current state belongs to element i from the partition. The restriction given above implies that there exist functions $\lfloor\ \rfloor_k$ satisfying (56) and (57).

4.2. PROGRAM STATEMENTS

Notice that the functions $\lfloor\ \rfloor_k$ define a method by which one can switch from one element of the partition to another. To enable us to do this in the framework of a programming language, we introduce a new statement $switch_k$, defined by

$$switch_k = \{x \triangleright x\ \lfloor x \rfloor_k\} \tag{58}$$

> *Restriction.* For a programming language implementing such a semantics, *switch* is the only statement by which a transition can be made from one element of the state partition to another. (*End of Restriction.*)

Theorem (*Left zero*)

$$switch_k \text{ is the left zero of } \lhd_p, \text{ for all } k \text{ satisfying } \neg p. \tag{59}$$

Proof.

$$\begin{array}{ll} & switch_k \lhd_p T \\ = & \quad \{\ (33)\text{: def. of } \lhd_p\ \} \\ & \{s, x, t \mid sx \in switch_k \wedge xt \in T \wedge fin.s \wedge (p \circ idx).x \triangleright sxt\} \cup \\ & \{s \mid s \in switch_k \wedge (inf.s \vee \neg(p \circ idx).(last.s)) \triangleright s\} \\ = & \quad \{\ (58)\text{: def. of } switch\text{, since } \neg p.k\ \} \\ & \{s \mid s \in switch_k \triangleright s\} \\ = & \quad \{\ \text{def. of set}\ \} \\ & switch_k \end{array}$$

□

Theorem

$$wp.switch_k.Q.x = Q.\lfloor x \rfloor_k. \tag{60}$$

Proof.

$$\begin{array}{ll} & wp.switch_k.Q.x \\ = & \quad \{\ (45)\text{: def. of } wp\ \} \\ & \langle \forall\ t \mid first.t = x \wedge t \in switch_k \triangleright fin.t \wedge Q.(last.t) \rangle \\ = & \quad \{\ (58)\text{: def. of } switch_k\ \} \\ & \langle \forall\ t \mid t = x \triangleright Q.\lfloor t \rfloor_k \rangle \\ = & \quad \{\ \text{one-point rule}\ \} \\ & Q.\lfloor x \rfloor_k \end{array}$$

□

Restriction. Earlier, we had used tagged collections as predicates. We now impose one additional restriction on the predicates used in the tagged collection. If Q is the tagged collection $\langle \sqcup\ k \triangleright k{:}\, Q.k \rangle$, then

$$\langle \forall\ i, j, k, x \triangleright Q.k.\lfloor x \rfloor_i = Q.k.\lfloor x \rfloor_j \rangle \tag{61}$$

In other words, $Q.k.x$ does not depend on the element of the partition to which x belongs. (*End of Restriction.*)

Theorem

$$wp.switch_k = \restriction_k \tag{62}$$

Proof.

$$\begin{array}{ll} & wp.switch_k.Q.x \\ = & \quad \{\ (60)\ \} \\ & Q.\lfloor x \rfloor_k \\ = & \quad \{\ (46)\text{: tagged collection as a predicate, since (56): } idx.\lfloor x \rfloor_k = k\ \} \\ & Q.k.\lfloor x \rfloor_k \\ = & \quad \{\ (61)\ \} \\ & Q.k.x \\ = & \quad \{\ (4)\text{: def. of } \restriction_k\ \} \\ & (\restriction_k.Q).x \end{array}$$

□

Once again identifying a program with its wp, we have

$$switch_k = \restriction_k$$

Note that the properties of functions discussed in section 1 do not always carry over to the trace semantics, although they are maintained by the weakest precondition semantics. This can be expected, since trace semantics are more concrete than wp semantics which refer only to initial and final states of the computation. We can derive additional properties using the theorems from section 1. For instance, we can prove (from (5)) that

$$switch_k \lhd_p S = S, \quad \text{if } p.k$$

The semantics of partitioned state spaces can be used in various ways. Defining the partition of the state space in different ways can result in semantics for different language constructs. For instance:

1. For the semantics of exceptions presented in [6], a boolean coordinate *oc* is introduced into the state which is used to partition the state space into two subspaces.
2. To extend the semantics presented in [6] to handle more than one exception, we can let the *oc* coordinate be integer valued.
3. Structured jumps use a partition in which an extra coordinate *label* is used to identify the label of the block to which a branch is to be executed.

Each of these is discussed in detail in [8].

4.3. DISCUSSION

In this section, the semantics of partitions was introduced by imposing restrictions on programming language constructs, and introducing the *switch* statement. Notice that theorem (42) suggests an implementation of the semantics of partitions using the *IF* statement. By showing how this operator can be used to describe exceptions, we have also demonstrated that a programming language with exceptions would have to check if an exception had been raised after the completion of every statement. However, the suggested restrictions on programming languages—namely, that *switch* is the only statement that can change the current partition—enables a large number of these checks to be removed, since statements that raise or lower exceptions can be determined statically.

The results from [6] served as the foundation for the semantics presented here. Other early references to semantics, in particular semantics of exception handling, are [2] and the unpublished [1]. The formerly mentioned of these references provides a good discussion of how exception handling can simplify the structure of certain programs. It gives separate predicate transformers for normal and exceptional outcomes, whereas we do not distinguish between the two. In [1], an arbitrary number of outcomes is considered and a structured branching scheme is discussed. In [6], $wep.S$ is applied to a pair of predicates, whereas we only have one (partitioned) predicate.

5. CONCLUSION

In this note, tagged collections and their compositions were introduced. These collections can be viewed as quantified expressions and can be used to manipulate lists and functions using the existing properties of quantification and one additional axiom. These tagged collections were used to introduce the theory of conditional composition of functions, and the conditional replace operator.

These concepts were used to describe the semantics of programming languages, and to introduce new programming language constructs. We described the semantics of partitioned state spaces, which was then used to describe the semantics of exceptions, loop exits, and structured jumps.

6. ACKNOWLEDGMENTS

We would like to thank Mani Chandy for many helpful discussions and valuable suggestions, including that conditional composition be done in terms of a predicate instead of a subset. We are also grateful for the helpful comments provided by the members of our research group at Caltech. Finally we would like to thank the referees for their comments.

A. Tagged Collections

A quantifier Q is defined by a pair $(\star, u)$ where $\star$ is an associative and symmetric operator from $T \times T \to T$ and u is the unit element of $\star$ (cf. [5]). A quantified expression (or quantifier) is an expression of the form

$$\langle Q\ x \mid r.x \triangleright t.x \rangle \quad ,$$

where x is an unordered list of identifier names (dummies), $r.x$ (the range) is a predicate, and $t.x$ (the term) is an expression of some type T. When $r.x$ is true everywhere or understood, we omit the range and write the quantification as

$$\langle Q\ x \triangleright t.x \rangle \quad .$$

Informally, if the set of all values x for which $r.x$ holds is $\{x_0, x_1, \ldots, x_k\}$, then the value of the quantified expression is $u \star t.x_0 \star t.x_1 \star \cdots \star t.x_k$. For example, summation can be written as the quantifier $(+, 0)$.

We use quantification to denote a tagged collection. We represent an element in the tagged collection by $(i\colon x)$, where i is the index of the element, and x is the value tagged by i. We use $\emptyset$ to denote an empty collection, which has the property that $\emptyset \sqcup l = l$ for any l. Formally, we define $\bigsqcup$ to be the quantifier $(\sqcup, \emptyset)$. $\sqcup$ is an operator analogous to the disjoint set union operator, often written $\uplus$. An element $(i\colon x)$ is contained in $l0 \sqcup l1$ just when it is contained in either $l0$ or $l1$.

An indexed collection of objects has an additional property, namely that given an index, the element associated with that index is uniquely defined. Therefore, we restrict $\sqcup$ in the following manner: given $l0$ and $l1$, $l0 \sqcup l1$ is defined if and only if the index sets of $l0$ and $l1$ are disjoint.

We think of $(i\colon x)$ as function $(i \mapsto x)$ whose domain is the element i and whose range is the element x. Then, we have the property that $(i\colon x).i = x$ which is a consequence of function application. Notice that with this meaning, the restriction on $\sqcup$ follows from the fact that a function cannot be one-to-many.

Since $\bigsqcup$ is simply a quantifier, we can use the following properties which follow from the general properties of quantification. However, we must be careful since $\sqcup$ imposes certain restrictions on its arguments.

We present some of the axioms from [5]. These axioms are simply instances of a few of the axioms for general quantifiers.

The following two axioms can be used to eliminate quantification from an expression.

Axiom (*Empty Range*)

$$\langle \sqcup\ i \mid \mathit{false} \triangleright f.i\colon t.i \rangle = \emptyset \tag{63}$$

Axiom (*One-point rule*)

$$\langle \sqcup\ i \mid [i = E] \triangleright f.i : t.i \rangle = (f.E : t.E) \tag{64}$$

For more general ranges, the following axiom can be used.

Axiom (*Range split*)
For mutually exclusive r and s,

$$\langle \sqcup\ i \mid r.i \vee s.i \triangleright f.i : t.i \rangle = \langle \sqcup\ i \mid r.i \triangleright f.i : t.i \rangle \sqcup \langle \sqcup\ i \mid s.i \triangleright f.i : t.i \rangle \tag{65}$$

The following axiom can be used to rewrite the term.

Axiom (*Transform term*)
If, for all i, $r.i \Rightarrow (t.i = z.i) \wedge (f.i = g.i)$, then

$$\langle \sqcup\ i \mid r.i \triangleright f.i : t.i \rangle = \langle \sqcup\ i \mid r.i \triangleright g.i : z.i \rangle \tag{66}$$

Given a tagged collection, it is important to be able to determine the value associated with a particular tag in the collection. We propose the following axiom for tagged collections.

Axiom (*Projection*)
For injective f,

$$\langle \sqcup\ i \mid r.i \triangleright f.i : t.i \rangle.k = \langle \sqcup\ i \mid r.i \wedge [f.i = k] \triangleright f.i : t.i \rangle.k \tag{67}$$

We now prove,

Theorem (*Projection*)
If r and s are mutually exclusive and $r.k$ holds,

$$(\langle \sqcup\ i \mid r.i \triangleright i : t.i \rangle \sqcup \langle \sqcup\ i \mid s.i \triangleright i : z.i \rangle).k = t.k \tag{68}$$

Proof. For this proof, we define function $h.i$ in the following manner:

$$h.i = \begin{cases} t.i & \text{if } r.i \\ z.i & \text{if } s.i \end{cases}$$

Now,

$$\begin{array}{ll} & (\langle \sqcup\ i \mid r.i \triangleright i : t.i \rangle \sqcup \langle \sqcup\ i \mid s.i \triangleright i : z.i \rangle).k \\ = & \quad \{\ (66)\text{: transform term, twice }\} \\ & (\langle \sqcup\ i \mid r.i \triangleright i : h.i \rangle \sqcup \langle \sqcup\ i \mid s.i \triangleright i : h.i \rangle).k \\ = & \quad \{\ (65)\text{: range split, since } r,s \text{ are mutually exclusive }\} \\ & \langle \sqcup\ i \mid r.i \vee s.i \triangleright i : h.i \rangle.k \\ = & \quad \{\ (67)\text{: projection }\} \\ & \langle \sqcup\ i \mid (r.i \vee s.i) \wedge [i = k] \triangleright i : h.i \rangle.k \\ = & \quad \{\ r.k;\ (66)\text{: transform term, using def. of } h\ \} \\ & \langle \sqcup\ i \mid [i = k] \triangleright i : t.i \rangle.k \\ = & \quad \{\ (64)\text{: one-point rule }\} \\ & (k : t.k).k \\ = & \quad \{\ \text{definition }\} \\ & t.k \end{array}$$

□

Using these theorems and axioms, we can manipulate arbitrary tagged collections. For example, the tagged collection $a = \langle \sqcup\, i \mid 0 \leq i \wedge i < n \triangleright i{:}\, i^2 \rangle$ can be considered to be an array of size n that contains the squares of the first n natural numbers. $a.k$ would be the kth element of the array (for $0 \leq k < n$), since:

$$
\begin{array}{ll}
 & \langle \sqcup\, i \mid 0 \leq i \wedge i < n \triangleright i{:}\, i^2 \rangle.k \\
= & \quad \{\ \emptyset \text{ is the identity of } \sqcup\ \} \\
 & (\langle \sqcup\, i \mid 0 \leq i \wedge i < n \wedge [i = k] \triangleright i{:}\, i^2 \rangle \sqcup \emptyset).k \\
= & \quad \{\ (68), \text{ since } 0 \leq k \wedge k < n\ \} \\
 & k^2
\end{array}
$$

□

References

[1] R.-J.R. Back and M. Karttunen. A predicate transformer semantics for statements with multiple exits. University of Helsinki, unpublished MS, 1983.

[2] F. Cristian. Correct and robust programs. *IEEE Transactions on Software Engineering*, 10:163-174, 1984.

[3] E.W. Dijkstra and C.S. Scholten. *Predicate Calculus and Program Semantics*. Springer-Verlag, 1990.

[4] R.M. Dijkstra. Operational Semantics: Correction & embellishment. Internal note rutger 16, 1993.

[5] D. Gries and F. Schneider. *A Logical Approach to Discrete Math*. Springer-Verlag 1993.

[6] K.R.M. Leino and J.L.A. van de Snepscheut. Semantics of Exceptions. In E.-R. Olderog, editor, *Proceedings of the IFIP Working Conference on Programming Concepts, Methods, and Calculi*, pp. 447–466. June 1994.

[7] J.J. Lukkien. An operational semantics for the guarded command language. In R.S. Bird, C.C. Morgan, and J.C.P. Woodcock, editors, *Mathematics of Program Construction*, number 669 in Lecture Notes in Computer Science, pp. 233-249. Springer-Verlag, 1993.

[8] R. Manohar and K.R.M. Leino. Theory and Use of Conditional Composition. Caltech technical report CS-TR-94-12.