University of Amsterdam
Programming Research Group

Tuplix Calculus

J.A. Bergstra
A. Ponse
M.B. van der Zwaag

Report PRG0713 December 2007

J.A. Bergstra

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ   Amsterdam
The Netherlands

tel. +31 20 525.7591
e-mail: janb@science.uva.nl

A. Ponse

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ   Amsterdam
The Netherlands

tel. +31 20 525.7592
e-mail: alban@science.uva.nl

M.B. van der Zwaag

Programming Research Group
Faculty of Science
University of Amsterdam

Kruislaan 403
1098 SJ   Amsterdam
The Netherlands

tel. +31 20 525.7584
e-mail: mbz@science.uva.nl

Programming Research Group Electronic Report Series

Tuplix Calculus[*]

Jan A. Bergstra Alban Ponse
Mark B. van der Zwaag

Section Software Engineering, Informatics Institute, University of Amsterdam
Email: `{janb,alban,mbz}@science.uva.nl`

Abstract

We introduce a calculus for tuplices, which are expressions that generalize matrices and vectors. Tuplices have an underlying data type for *quantities* that are taken from a zero-totalized field. We start with the core tuplix calculus CTC for entries and tests, which are combined using conjunctive composition. We define a standard model and prove that CTC is relatively complete with respect to it. The core calculus is extended with operators for choice, information hiding, scalar multiplication, clearing and encapsulation. We provide two examples of applications; one on incremental financial budgeting, and one on modular financial budget design.

1 Introduction

In this paper we propose *tuplix calculus*: a calculus for so-called *tuplices*, which are expressions that generalize matrices and vectors. Tuplices have an underlying data type called *quantities*. We shall require that this data type is modeled by a zero-totalized field, in the terminology of [8, 3], as will be explained in Section 2. A typical example of a tuplix is a budget, a compound of various attributes, each of which possesses a certain value (a quantity) and may refer to certain conditions and/or interdependencies. Another example of a tuplix is the modeling of *let*-expressions in functional programming. We provide a standard model for tuplix calculus and discuss some examples of its use.

A tuplix generalizes a vector or a matrix in that it collects a number of quantities from the same data type under a number of names (dimensions of the vector, matrix entries). What differs in the design of tuplix calculus from matrix or vector calculus is that other methods of compositional construction are envisaged. *Conjunctive composition* extends both vector (matrix) addition and set union into a novel compositional mechanism. In addition, 'information

[*]Thanks are due to Sanne Nolst Trenité (UvA, Faculty of Science) for many discussions and for helping to clarify the usefulness and necessity of formalization in budget calculations.

hiding' is provided which supports the use of auxiliary values whose name is made externally invisible. In order to provide a simple semantic model of this form of information hiding, *alternative composition* of tuplices is included as well. Alternative composition, written $x + y$ denotes a tuplix which is either x or y. Information hiding is modeled using *generalized alternative composition.* A further key feature of tuplix calculus is the inclusion of conditions as atomic tuplices; this is done via a *zero-test operator*. Finally an *encapsulation* mechanism is proposed. Encapsulation removes an entry and enforces that it will contain quantity zero only. Encapsulation and alternative composition work together exactly as in the process algebra ACP [4] from which the typescript has been borrowed (see [1] and [10] for more recent expositions of ACP-style process algebra).

Tuplices constitute a calculus rather than an algebra because information hiding introduces bound variables. The motivation for designing tuplix calculus came from a number of attempts to design financial budgets in a modular fashion. One might call a tuplix a budget but we prefer not to introduce that financial connotation by using a mathematically neutral term which can be viewed as describing a purely structural notion without any preferred application or even application area. We call tuplix calculus an abstract data type calculus. It is based on an algebraic abstract data type but this is augmented with operators involving bound variables, notably the generalized alternative composition operator.

The design of tuplix calculus is based on zero-totalized fields because this drastically simplifies type checking in general and equational logic in particular for fields. That zero-totalized fields are meadows, which in general may feature proper zero-divisors as a natural generalization, is of less importance to the design of tuplix calculus.

The paper is structured as follows. Section 2 discusses zero-totalized fields. Section 3 introduces the axiom system CTC (Core Tuplix Calculus). We define a standard model for the interpretation of tuplices and prove the relative completeness of CTC (relative: valid data identities are assumed in the proof theory) with respect to this standard model (Section 5). In the next sections, CTC is extended with several operators, starting with alternative composition in Section 6, leading to Basic Tuplix Calculus (BTC). Section 7 is an intermezzo containing some observations on the use of zero tests (*zero-test logic*). Section 8 introduces information hiding to BTC through the binding of data variables. Section 9 defines three auxiliary operations including encapsulation. Sections 10 and 11 present example applications. We end with some concluding remarks (Section 12).

2 Cancellation Meadows

Quantities will be taken from a *non-trivial cancellation meadow*, or, equivalently, from a *zero-totalized field*, in the terminology of [8, 3]. A zero-totalized field is the well-known algebraic structure 'field' with a total operator for division so that the result of division by zero is zero (and, for example, in a 47-totalized field one has chosen 47 to represent the result of all divisions by zero).

A *meadow* is a commutative ring with unit equipped with a total unary operation $(\_)^{-1}$ named inverse that satisfies the axioms

$$(u^{-1})^{-1} = u \quad \text{and} \quad u \cdot (u \cdot u^{-1}) = u,$$

and in which $0^{-1} = 0$. For quantities (and tuplix calculus) we also require the *cancellation axiom*

$$u \neq 0 \quad \& \quad u \cdot v = u \cdot w \quad \Rightarrow \quad v = w$$

to hold, thus obtaining *cancellation meadows*, which we take as the mathematical structure for quantities, requiring further that $0 \neq 1$ to exclude (trivial) one-point models. These axioms for cancellation meadows characterize exactly the equational theory of zero-totalized fields [3]. The property of cancellation meadows that is exploited in the tuplix calculus is that division by zero yields zero, while $u \cdot u^{-1} = 1$ for $u \neq 0$.

For the tuplix calculus, we define a *data type* (signature and axioms) for quantities which comprises the constants 0, 1, the binary operators $+$ and $\cdot$, and the unary operators $-$ and $(\_)^{-1}$. We often write $u - v$ instead of $u + (-v)$, u/v instead of $u \cdot v^{-1}$, and uv instead of $u \cdot v$, and we shall omit brackets if no confusion can arise following the usual binding conventions. Finally, we use numerals in the common way (2 abbreviates $1 + 1$, etc.). The axiomatization consists of the cancellation axiom

$$u \neq 0 \quad \& \quad u \cdot v = u \cdot w \quad \Rightarrow \quad v = w,$$

the *separation axiom*

$$0 \neq 1,$$

and the following 10 axioms for meadows (see [3]):

$$\begin{aligned}
(u + v) + w &= u + (v + w),\\
u + v &= v + u.\\
u + 0 &= u,\\
u + (-u) &= 0,\\
(u \cdot v) \cdot w &= u \cdot (v \cdot w),\\
u \cdot v &= v \cdot u,\\
1 \cdot u &= u,\\
u \cdot (v + w) &= u \cdot v + u \cdot w,\\
(u^{-1})^{-1} &= u,\\
u \cdot (u \cdot u^{-1}) &= u.
\end{aligned}$$

The following identities are derivable from the axioms for meadows.

$$\begin{aligned}
(0)^{-1} &= 0\\
(-u)^{-1} &= -(u^{-1})\\
(u \cdot v)^{-1} &= u^{-1} \cdot v^{-1}\\
0 \cdot u &= 0\\
u \cdot -v &= -(u \cdot v)\\
-(-u) &= u
\end{aligned}$$

Furthermore, the cancellation axiom and axiom $u \cdot (u \cdot u^{-1}) = u$ imply the *general inverse law*

$$u \neq 0 \quad \Rightarrow \quad u \cdot u^{-1} = 1$$

of zero-totalized fields.

3 Core Tuplix Calculus

Tuplix calculus builds on the data type defined in Section 2 which specifies non-trivial cancellation meadows. We use the letters u, v and w as data variables, and the letters p and q to range over (open) data terms.

We start with a core calculus which can be extended with several operators (as is done in later sections). The theory is parametrized with a nonempty set A of *attributes*, ranged over by a and b. We further assume given a countably-enumerable set of tuplix variables, ranged over by x, y and z. We introduce the signature for tuplices. We have constants ε (the empty tuplix) and δ (the null tuplix); the variables are tuplix terms; and there are two further kinds of atomic tuplices: *entries* (attribute-value pairs) of the form

$$a(p)$$

with $a \in A$, and p a data term, and, for any data term p, the *zero test*

$$\gamma(p)$$

($\gamma \notin A$). Finally, the core theory has one binary infix operator: the *conjunctive composition* operator $\between$. This operator is commutative and associative.

Axioms:

$$x \between y = y \between x \tag{T1}$$

$$(x \between y) \between z = x \between (y \between z) \tag{T2}$$

$$x \between \varepsilon = x \tag{T3}$$

$$x \between \delta = \delta \tag{T4}$$

$$a(u) \between a(v) = a(u + v) \tag{T5}$$

$$\gamma(u) = \gamma(u/u) \tag{T6}$$

$$\gamma(0) = \varepsilon \tag{T7}$$

$$\gamma(1) = \delta \tag{T8}$$

In this core calculus, a tuplix is a conjunctive composition of tests and entries, with ε representing an empty tuplix, and δ representing an erroneous situation which nullifies the entire composition. Entries with the same attribute can be combined to a single entry containing the sum of the quantities involved (axiom T5).

A zero test $\gamma(p)$ acts as a conditional: if the argument p equals zero, then the test is void and disappears from conjunctive compositions. If the argument is not equal to zero, the test nullifies any conjunctive composition containing it. Observe how we exploit the property of zero-totalized fields that p/p is always defined, and that the division p/p yields zero if p equals zero, and 1 otherwise. Further observe that an equality test $p = q$ can be expressed as $\gamma(p - q)$.

A tuplix term is *closed* if it is does not contain tuplix variables and also does not contain data variables. A tuplix term is *tuplix-closed* if it does not contain tuplix variables (but it may contain data variables). For reasoning about tuplices with open data terms, we add the following two axioms:

$$\gamma(u) \oplus \gamma(v) = \gamma(u/u + v/v) \tag{T9}$$

$$\gamma(u - v) \oplus a(u) = \gamma(u - v) \oplus a(v) \tag{T10}$$

The tuplix calculus is two-sorted. On the tuplix side we have the axioms T1–T10 and we use the proof rules of equational logic. On the data side, we refrain from giving a precise proof theory. We adopt the following rule to lift the valid data identities to the tuplix calculus: for all (open) data terms p and q,

$$\mathcal{D} \models p = q \quad \text{implies} \quad \gamma(p) = \gamma(q), \tag{De}$$

where $\mathcal{D}$ (a non-trivial cancellation meadow) is our model of the data type. This axiom system with axioms T1–T10 plus proof rule De is denoted by CTC (Core Tuplix Calculus).

4 Canonical Terms and Derived Proof Rules

A CTC canonical term is a term of the form

$$\gamma(p_0) \oplus a_1(p_1) \oplus \cdots \oplus a_k(p_k) \oplus x_1 \oplus \cdots \oplus x_l,$$

for some $k, l \geq 0$, and with distinct attributes a_i for $i = 1, \ldots, k$.

Lemma 1. *Every* CTC *term is derivably equal to a* CTC *canonical term.*

Proof. Easy: If it contains the constant δ, the term equals the canonical term $\gamma(1)$ (using axioms T4 and T8); conjunctive composition is commutative and associative; the ε constant disappears (axiom T3); entries with same attribute are combined using axiom T5; tests are combined using axiom T9 (and if there are no tests, we add a void $\gamma(0)$ test using axiom T7). □

The axiom system CTC is powerful enough for our purposes, as is witnessed by the completeness result in Section 5. Still, more general proof rules for the derivation of identities involving data equalities and substitution of data terms can be convenient. For example, we find that CTC derives the "obvious" identity

$$a(u+v) = a(v+u)$$

rather indirectly: because $(u+v)-(v+u)=0$ will be valid in our data model, we have

$$\gamma((u+v)-(v+u)) = \gamma(0)$$

by De. Then we derive

$$a(u+v) = a(u+v) \oplus \gamma((u+v)-(v+u)) = a(v+u)$$

using axioms T3, T7, and T10.

The following proof rule generalizes De:

$$\mathcal{D} \models p = q \quad \text{implies} \quad t[p/u] = t[q/u], \tag{De$^+$}$$

for tuplix terms t and with substitution $t[p/u]$ defined as usual for two-sorted equational logic (replacement of all data variables u in t by p). The following axiom scheme generalizes axiom T10:

$$t \oplus \gamma(u-p) = t[p/u] \oplus \gamma(u-p), \tag{T10$^+$}$$

where t ranges over tuplix terms.

These two rules follow from CTC as we shall now prove. We start with two lemmas.

Lemma 2. *For all data terms p and q,*

$$\mathcal{D} \models (1-p/p)\cdot q = 0 \quad \textit{implies} \quad \text{CTC} \vdash \gamma(p) = \gamma(p) \oplus \gamma(q).$$

Proof. Assume that $\mathcal{D} \models (1-p/p)\cdot q = 0$. Observe that it follows that

$$p/p = (p/p + q/q)/(p/p+q/q)$$

is a valid identity (check: distinguish cases $p=0$ and $p \neq 0$). From this, derive

$$\begin{aligned}\gamma(p) &= \gamma(p/p)\\ &= \gamma((p/p+q/q)/(p/p+q/q))\\ &= \gamma(p) \oplus \gamma(q)\end{aligned}$$

using De and axioms T6 and T9. □

Note that a test $\gamma((1-p/p)\cdot q)$ may be read as the logical implication '$p=0$ implies $q=0$', see also Section 7.

Lemma 3. *The following identity is derivable in* CTC.

$$\gamma(u) \oplus \gamma(u-v) = \gamma(v) \oplus \gamma(u-v)$$

Proof. Observe that

$$\left(1 - \frac{u/u + (u-v)/(u-v)}{u/u + (u-v)/(u-v)}\right) \cdot v = 0$$

('if $u = 0$ and $u = v$, then $v = 0$') is valid in any cancellation meadow. Derive

$$\begin{aligned}\gamma(u) \oplus \gamma(u-v) &= \gamma(u/u + (u-v)/(u-v)) \\ &= \gamma(u/u + (u-v)/(u-v)) \oplus \gamma(v) \\ &= \gamma(u) \oplus \gamma(u-v) \oplus \gamma(v)\end{aligned}$$

using Lemma 2 and axiom T9. The remaining part of the derivation is symmetrical. □

We are now ready to derive the two rules.

- Case De$^+$. Assume that $\mathcal{D} \models p = q$, and let t be a canonical term

  $$\gamma(p_0) \oplus a_1(p_1) \oplus \cdots \oplus a_k(p_k) \oplus x_1 \oplus \cdots \oplus x_l,$$

  for some $k, l \geq 0$. First observe that it follows from $\mathcal{D} \models p = q$ that

  $$\mathcal{D} \models p_i[p/u] = p_i[q/u] \quad \text{and} \quad \mathcal{D} \models p_i[p/u] - p_i[q/u] = 0$$

  for $i = 0, \ldots, k$. From this and De we derive that

  $$\gamma(p_0[p/u]) = \gamma(p_0[q/u])$$

  and

  $$\begin{aligned}a_i(p_i[p/u]) &= a_i(p_i[p/u]) \oplus \gamma(0) \\ &= a_i(p_i)[p/u] \oplus \gamma(p_i[p/u] - p_i[q/u]),\end{aligned}$$

  so we can apply the required substitutions in the entries using axiom T10.

- Case T10$^+$. Let t be a canonical term

  $$\gamma(p_0) \oplus a_1(p_1) \oplus \cdots \oplus a_k(p_k) \oplus x_1 \oplus \cdots \oplus x_l,$$

  for some $k, l \geq 0$. Observe that for $i = 0, \ldots, k$,

  $$\mathcal{D} \models (1 - (u-p)/(u-p)) \cdot (p_i - p_i[p/u]).$$

  Therefore we have by Lemma 2 that

  $$\gamma(u-p) = \gamma(u-p) \oplus \gamma(p_i - p_i[p/u])$$

  so that we can perform the substitutions in the entries using axiom T10, and in the test $\gamma(p_0)$ using Lemma 3.

5 Standard Model and Relative Completeness

We interpret tuplix terms in the *standard model* $\mathcal{M}(\mathcal{D}, A)$, where $\mathcal{D}$ is the model of the data type for quantities, and A is the set of attributes that are used. The data model $\mathcal{D}$ is required to be a non-trivial cancellation meadow. We write $\mathcal{D}$ for the domain of $\mathcal{D}$, and 0 for the element of $\mathcal{D}$ that is the interpretation of the data term 0.

The standard model is based on the set

$$F = A \xrightarrow{p} \mathcal{D}$$

of partial functions from A to $\mathcal{D}$ which are used to model the entries (the attribute-value pairs). The domain for the standard model is the power set

$$2^F$$

of the set of partial functions. An element of this power set stands for a number of *alternatives*: for CTC, the interpretation of tuplix terms yields either the empty set (the interpretation of δ; absence of alternatives) or a singleton set. When we add *choice* to the theory (see Section 6), the interpretation may yield sets with more than one element.

Some preliminaries:

1. For $a \in A$, $d \in \mathcal{D}$, let $f_{a,d}$ be the partial function with $f_{a,d}(a) = d$, and $f_{a,d}(b)$ undefined for $b \neq a$.
2. We denote by f_ε the function in F with $f_\varepsilon(a)$ undefined for all attributes $a \in A$; this function will be used in the interpretation of the term ε.
3. Define conjunctive composition $\oplus$ on elements of F as follows: for $a \in A$, if both f and g are undefined for a, then $(f \oplus g)$ is undefined for a; if $f(a)$ is defined and g is not defined for a, then $(f \oplus g)(a) = (g \oplus f)(a) = f(a)$; and if both f and g are defined for a, then $(f \oplus g)(a) = f(a) + g(a)$.

The closed terms of CTC are interpreted in the standard model as follows.

$$\begin{aligned}
[\![\delta]\!] &\overset{\text{def}}{=} \emptyset \\
[\![\varepsilon]\!] &\overset{\text{def}}{=} \{f_\varepsilon\} \\
[\![a(p)]\!] &\overset{\text{def}}{=} \{f_{a,[\![p]\!]}\} \\
[\![\gamma(p)]\!] &\overset{\text{def}}{=} \begin{cases} [\![\varepsilon]\!] & \text{if } [\![p]\!] = 0 \\ [\![\delta]\!] & \text{otherwise} \end{cases} \\
[\![s \oplus t]\!] &\overset{\text{def}}{=} \{f \oplus g \mid f \in [\![s]\!],\ g \in [\![t]\!]\}
\end{aligned}$$

We say that closed terms s and t are *equivalent* with respect to the standard model if $[\![s]\!] = [\![t]\!]$. Two open terms are equivalent, notation $s \sim t$, if all their closed instantiations are pair-wise equivalent. The axiom system CTC is sound with respect to the standard model, i.e., for all (open) tuplix terms s and t, CTC $\vdash s = t$ implies $s \sim t$.

Theorem 1. *The axiom system* CTC *is complete with respect to the standard model, i.e., for all (open) terms s and t, $s \sim t$ implies* $\mathrm{CTC} \vdash s = t$.

This completeness is *relative* in the sense that our proof theory assumes, by adoption of rule De, all valid data identities.

Proof. Suppose $s \sim t$. Using Lemma 1 we know that s and t are derivably equal to canonical terms

$$s' = \gamma(p_0) \oplus a_1(p_1) \oplus \cdots \oplus a_k(p_k) \oplus x_1 \oplus \cdots \oplus x_l$$

and

$$t' = \gamma(q_0) \oplus a_1(q_1) \oplus \cdots \oplus a_k(q_k) \oplus x_1 \oplus \cdots \oplus x_l$$

with $k, l \geq 0$. Observe that it follows from $s \sim t$ that we can find canonical terms having the same tuplix variables x_i and (mutually distinct) attributes a_j.

It also follows from $s \sim t$, that whenever the test $\gamma(p_0)$ succeeds, also the test $\gamma(q_0)$ succeeds, and vice versa. Therefore, the cancellation meadow identity $p_0/p_0 = q_0/q_0$ must be valid. It follows that

$$\gamma(p_0) = \gamma(q_0)$$

is derivable using axiom T6 and De.

It further follows from $s \sim t$, that whenever the test $\gamma(p_0)$, and hence also $\gamma(q_0)$, succeeds, then it must be that $p_i = q_i$ for $i = 1, \ldots, k$. A consequence is that the cancellation meadow identity

$$(1 - p_0/p_0)(p_i - q_i) = 0$$

is valid (check: straightforward case distinction on p_0). Using Lemma 2 we find that

$$\gamma(p_0) = \gamma(p_0) \oplus \gamma(p_i - q_i).$$

Because we also have $\gamma(p_0) = \gamma(q_0)$ it is easy to see that $s' = t'$ is derivable using axiom T10. □

6 Basic Tuplix Calculus

The axiom system CTC is extended to Basic Tuplix Calculus (BTC), by addition of the binary operator $+$ called *alternative composition* or *choice* to the signature, and by adoption of the following axioms.

$$x + y = y + x \tag{C1}$$

$$(x + y) + z = x + (y + z) \tag{C2}$$

$$x + x = x \tag{C3}$$

$$x + \delta = x \tag{C4}$$

$$x \oplus (y + z) = (x \oplus y) + (x \oplus z) \tag{C5}$$

$$\gamma(u) + \gamma(v) = \gamma(uv) \tag{C6}$$

Because choice is an associative operator, we shall often omit brackets in repeated applications.

The standard model $\mathcal{M}(\mathcal{D}, A)$ for CTC is extended to BTC by the following interpretation of alternative composition:

$$[\![s+t]\!] \overset{\text{def}}{=} [\![s]\!] \cup [\![t]\!].$$

So, the interpretation of a closed term yields a set of *alternatives*. Note that δ is a zero element for alternative composition: it stands for the absence of alternatives (recall that $[\![\delta]\!] = \emptyset$). The axioms C1–C6 are sound with respect to the standard model.

As for CTC, completeness results are *relative* because, by adoption of proof rule De, valid data identities may be used in derivations. The axiom system BTC is complete for closed (no data variables, no tuplix variables) terms. In the proof we use canonical terms: a BTC canonical term is an alternative composition

$$t_1 + \cdots + t_k$$

of CTC canonical terms for some $k \geq 0$ (in case $k = 0$, this term is defined as δ). Clearly, we can derive such a canonical term for every BTC term by pushing $+$ outward using axiom C5.

Theorem 2. *For closed terms,* BTC *is complete with respect to the standard model, i.e., for closed terms s and t, $s \sim t$ implies* BTC $\vdash s = t$*.*

Proof. Take closed terms s and t with $s \sim t$. We may assume for s and t that there are respective canonical terms

$$s_1 + \cdots + s_k \quad \text{and} \quad t_1 + \cdots + t_l$$

such that $[\![s_i]\!]$ and $[\![t_j]\!]$ are singleton sets for $i = 1, \ldots, k$ and $j = 1, \ldots, l$. Since $[\![s]\!] = [\![t]\!]$, it is clear that for every s_i there is a t_j such that $s_i \sim t_j$, and vice versa. By completeness of CTC these are derivably equal. Then, $s = t$ can be derived using axioms C1–C4. □

Completeness for open terms (which we did prove for CTC) appears to be more involved. We leave this open for future work.

7 Zero-Test Logic

We have seen how the zero-test operator $\gamma(p)$ tests the equality $p = 0$. Using axioms T6 and De, it is easy to derive the following identities, which we shall often use implicitly in derivations:

$$\gamma(u) = \gamma(-u),$$
$$\gamma(u) = \gamma(u/n),$$
$$\gamma(u) = \gamma(n \cdot u),$$

where n ranges over all non-zero numerals.

We present some observations on the use of the zero-test operator which lead to a simple logic.

First, the empty tuplix ε with $\varepsilon = \gamma(0)$ by axiom T7 may be read as 'true', and the null tuplix δ with $\delta = \gamma(1)$ by axiom T8 may be read as 'false'.

Negation. Define the test 'not $p = 0$' by

$$\widetilde{\gamma}(p) \overset{\text{def}}{=} \gamma(1 - p/p).$$

Conjunctive composition of tests may be read as logical conjunction:

$$\gamma(p) \between \gamma(q) \overset{(\text{T9})}{=} \gamma(p/p + q/q)$$

tests '$p = 0$ and $q = 0$'.

Alternative composition of tests may be read as logical disjunction:

$$\gamma(p) + \gamma(q) \overset{(\text{C6})}{=} \gamma(pq)$$

tests '$p = 0$ or $q = 0$'.

A *formula* would then be a tuplix-closed (no tuplix variables) BTC term without entries. Any formula can be expressed as a single test $\gamma(p)$ using axioms T7–T9 and C6, and the definition of negation. Let φ range over formulas, and write $\widetilde{\varphi}$ for the negation of φ.

We find that this logic has all the usual properties. Clearly, conjunction and disjunction are commutative, associative, and idempotent, and it is not difficult to derive distributivity, absorption, and double negation elimination. The following identities are easily derived as well:

$$\varphi + \widetilde{\varphi} = \varepsilon, \tag{1}$$
$$\varphi \between \widetilde{\varphi} = \delta, \tag{2}$$
$$\varphi + \varepsilon = \varepsilon, \tag{3}$$
$$\varphi \between \delta = \delta. \tag{4}$$

As usual, implication can be defined in terms of negation and disjunction:

$$\widetilde{\gamma}(p) + \gamma(q) = \gamma((1 - p/p) \cdot q)$$

tests '$p = 0$ implies $q = 0$'.

Note. If the absolute operator $|\_|$ (with $|p| = p$ if $p \geq 0$, and $|p| = -p$ otherwise) is added to the signature of the data type, we can also express inequalities:

$$\gamma(|q - p| = q - p)$$

expresses the test $p \leq q$.

8 Generalized Alternative Composition

The *generalized alternative composition (*or: *summation) operator* $\sum_u$ is a unary operator that *binds* data variable u and can be seen as a data-parametric generalization of the alternative composition operator $+$. We add this binder to the signature of BTC and write $FV(t)$ for the set of free data variables occurring in tuplix term t. We write $Var(p)$ for the set of data variables occurring in data term p (there is no variable binding within data terms). Define substitution $t[p/u]$ as: replace every free occurrence of data variable u in tuplix term t by the data term p, such that no variables of p become bound in these replacements. E.g., recall the proof rule T10$^+$:

$$t \oplus \gamma(u - p) = t[p/u] \oplus \gamma(u - p).$$

This rule remains sound in the setting with summation, but application of the rule may require the renaming of bound variables in t using axiom S2, see below, so that the substitution can be performed. When considering substitutions we shall implicitly assume that bound variables are renamed properly.

The axiom schemes for $\sum_u$ are as follows, where s and t range over tuplix terms and p ranges over data terms.

$$\textstyle\sum_u t = t \qquad \text{if } u \notin FV(t) \qquad \text{(S1)}$$
$$\textstyle\sum_u t = \sum_v t[v/u] \qquad \text{if } v \notin FV(t) \qquad \text{(S2)}$$
$$\textstyle\sum_u (s \oplus t) = s \oplus \sum_u t \qquad \text{if } u \notin FV(s) \qquad \text{(S3)}$$
$$\textstyle\sum_u (s + t) = \sum_u s + \sum_u t \qquad \text{(S4)}$$
$$\textstyle\sum_u \gamma(u - p) = \varepsilon \qquad \text{if } u \notin Var(p) \qquad \text{(S5)}$$
$$\textstyle\sum_u \widetilde{\gamma}(u - p) = \varepsilon \qquad \text{if } u \notin Var(p) \qquad \text{(S6)}$$

(Recall from Section 7 that $\widetilde{\gamma}(p)$ is defined as $\gamma(1 - p/p)$.)

The standard model for BTC is extended with the following interpretation of summation:

$$[\![\textstyle\sum_u t]\!] \overset{\text{def}}{=} \{[\![t[p/u]]\!] \mid p \text{ a closed data term}\}.$$

The axiom schemes S1–S6 are sound with respect to this model.

Note. A similar summation operator (binding of data variables that generalizes alternative composition) is part of the specification language μCRL [12], which combines the process algebra ACP [4] with equationally specified abstract data types. A detailed exposition of a semantics and proof theory for this 'choice quantification' in the setting of process algebra can be found in the work of Luttik [13]. A corresponding treatment is possible in our case.

Lemma 4. *The following identities are derivable for all data terms p with $u \notin Var(p)$.*

$$\textstyle\sum_u (t \oplus \gamma(u - p)) = t[p/u] \qquad (5)$$
$$\textstyle\sum_u t = t[p/u] + \sum_u t \qquad (6)$$
$$\textstyle\sum_u t = t[p/u] + \sum_u (t \oplus \widetilde{\gamma}(u - p)) \qquad (7)$$

Proof. Derivation of (5):

$$\begin{aligned}\textstyle\sum_u(t \oplus \gamma(u-p)) &= \textstyle\sum_u(t[p/u] \oplus \gamma(u-p))\\ &= t[p/u] \oplus \textstyle\sum_u \gamma(u-p)\\ &= t[p/u]\end{aligned}$$

using T10$^+$, S3 and S5. Derivation of (6):

$$\begin{aligned}\textstyle\sum_u t &= \textstyle\sum_u(t \oplus \gamma(u-p)) + \sum_u(t \oplus \widetilde{\gamma}(u-p))\\ &= t[p/u] + \textstyle\sum_u t.\end{aligned}$$

using T3, (1), S4, C3, and (5). Derivation of (7): similar. □

Example. We derive

$$\textstyle\sum_u(a(u) \oplus \gamma(u^2-1)) = a(-1) + a(1).$$

Proof: from

$$\gamma(u^2-1) = \gamma((u+1)(u-1)) = \gamma(u+1) + \gamma(u-1)$$

it follows that

$$\begin{aligned}\textstyle\sum_u(a(u) \oplus \gamma(u^2-1)) &= \textstyle\sum_u(a(u) \oplus \gamma(u+1)) + \sum_u(a(u) \oplus \gamma(u-1))\\ &= a(-1) + a(1)\end{aligned}$$

using (5).

Example. *Let-expressions* or *let-bindings* allow value declarations or partial bindings in expressions. The term

$$\textstyle\sum_u(t \oplus \gamma(u-p))$$

characterizes

$$\text{let } u = p \text{ in } t.$$

Of course, p may contain variables, as for instance 'let $u = 7v + 1$ in t' can simply be expressed as

$$\textstyle\sum_u(t \oplus \gamma(u - 7v - 1)).$$

9 Auxiliary Operators

For BTC with summation, we define three auxiliary operators: scalar multiplication, clearing, and encapsulation. In each case the axioms for choice and summation (numbered 6 and 7) can be omitted, for inclusion in axiom system CTC or BTC.

9.1 Scalar Multiplication

Scalar multiplication $p \cdot t$ multiplies the quantities contained in entries in tuplix term t by p. It is specified by means of the following axioms.

$$u \cdot \varepsilon = \varepsilon \tag{Sc1}$$
$$u \cdot \delta = \delta \tag{Sc2}$$
$$u \cdot \gamma(v) = \gamma(v) \tag{Sc3}$$
$$u \cdot a(v) = a(u \cdot v) \tag{Sc4}$$
$$u \cdot (x \between y) = u \cdot x \between u \cdot y \tag{Sc5}$$
$$u \cdot (x + y) = u \cdot x + u \cdot y \tag{Sc6}$$
$$p \cdot \textstyle\sum_v t = \sum_v (p \cdot t) \qquad \text{if } v \notin \mathit{Var}(p) \tag{Sc7}$$

Axiom Sc7 is an axiom scheme with p ranging over data terms and t ranging over tuplix terms. An example with scalar multiplication is given in Section 10.

Standard Model. Take the standard model $\mathcal{M}(\mathcal{D}, A)$ as before (see Section 5). For partial function $f \in F$ and value $d \in \mathcal{D}$, define the scalar multiplication $d \cdot f$ as expected: $(d \cdot f)(a) = d \cdot (f(a))$ if $f(a)$ is defined, and undefined otherwise. The interpretation of scalar multiplication is defined by

$$[\![p \cdot t]\!] \overset{\text{def}}{=} \{[\![p]\!] \cdot f \mid f \in [\![t]\!]\}.$$

9.2 Clearing

For set of attributes $I \subseteq A$, the operator

$$\varepsilon_I(x)$$

renames all entries of x with attribute in I to ε. It "clears" the attributes contained in I. Axioms:

$$\varepsilon_I(\varepsilon) = \varepsilon \tag{Cl1}$$
$$\varepsilon_I(\delta) = \delta \tag{Cl2}$$
$$\varepsilon_I(\gamma(u)) = \gamma(u) \tag{Cl3}$$
$$\varepsilon_I(a(u)) = \begin{cases} \varepsilon & \text{if } a \in I \\ a(u) & \text{otherwise} \end{cases} \tag{Cl4}$$
$$\varepsilon_I(x \between y) = \varepsilon_I(x) \between \varepsilon_I(y) \tag{Cl5}$$
$$\varepsilon_I(x + y) = \varepsilon_I(x) + \varepsilon_I(y) \tag{Cl6}$$
$$\varepsilon_I(\textstyle\sum_u t) = \sum_u (\varepsilon_I(t)) \tag{Cl7}$$

For a set of attributes $B \subseteq A$ one can think of a function

$$\mathit{Select}_B(x) \overset{\text{def}}{=} \varepsilon_{A \setminus B}(x).$$

This function allows to focus on those entries with attribute contained in B.

Standard Model. Take the standard model $\mathcal{M}(\mathcal{D}, A)$ as before (see Section 5). Define the function ε_I on elements of F as follows. For partial function $f \in F$ and attribute $a \in A$, if $f(a)$ is undefined or $a \in I$, then $\varepsilon_I(f)(a)$ is undefined, else $\varepsilon_I(f)(a) = f(a)$. The interpretation of clearing:

$$[\![\varepsilon_I(t)]\!] \stackrel{\text{def}}{=} \{\varepsilon_I(f) \mid f \in [\![t]\!]\}.$$

9.3 Encapsulation

Encapsulation can be seen as 'conditional clearing'. For set of attributes $H \subseteq A$, the operator $\partial_H(x)$ encapsulates all entries in x with attribute $a \in H$. That is, for $a \in H$, if the accumulation of quantities in entries with attribute a equals zero, the encapsulation on a is considered successful and the a-entries are *cleared* (become ε); if the accumulation is not equal to zero, they become null (δ). This accumulation of quantities is computed per alternative: the encapsulation operator distributes over alternative composition. Axioms:

$$\partial_H(\varepsilon) = \varepsilon \tag{E1}$$

$$\partial_H(\delta) = \delta \tag{E2}$$

$$\partial_H(\gamma(u)) = \gamma(u) \tag{E3}$$

$$\partial_H(a(u)) = \begin{cases} \gamma(u) & \text{if } a \in H \\ a(u) & \text{if } a \notin H \end{cases} \tag{E4}$$

$$\partial_H(x \oplus \partial_H(y)) = \partial_H(x) \oplus \partial_H(y) \tag{E5}$$

$$\partial_H(x + y) = \partial_H(x) + \partial_H(y) \tag{E6}$$

$$\partial_H(\textstyle\sum_u t) = \textstyle\sum_u(\partial_H(t)) \tag{E7}$$

We further define

$$\partial_{H \cup H'}(x) \stackrel{\text{def}}{=} \partial_H \circ \partial_{H'}(x).$$

Standard Model. Take the standard model $\mathcal{M}(\mathcal{D}, A)$ as before. We say that a partial function f in F is *neutral* on attribute a, if either $f(a)$ is undefined or $f(a) = 0$. We interpret encapsulation as follows.

$$[\![\partial_H(t)]\!] \stackrel{\text{def}}{=} \{\varepsilon_H(f) \mid f \in [\![t]\!],\ f \text{ neutral on all } a \in H\},$$

where ε_H is as defined in Section 9.2.

9.4 On the Derivation of Encapsulations

By the derivation of an encapsulation we mean the elimination of the encapsulation operator by application of its defining axioms from left to right. Of course, this elimination is in general only possible for tuplix-closed terms. We present some helpful identities and example derivations.

We start with a lemma.

Lemma 5. *For all tuplix-closed terms t, if no element of set of attributes H occurs in t, then*
$$\partial_H(t) = t$$
and, for any term s,
$$\partial_H(s \oplus t) = \partial_H(s) \oplus t$$
are derivable.

Proof. The first identity is easy, using structural induction on term t. Then, the second one follows using axiom E5:

$$\partial_H(s \oplus t) = \partial_H(s \oplus \partial_H(t)) = \partial_H(s) \oplus \partial_H(t) = \partial_H(s) \oplus t.$$

□

When deriving an encapsulation, we generally split the encapsulation up: for $a \in H$, we have by definition that

$$\partial_H(t) = \partial_{H \setminus \{a\}} \circ \partial_{\{a\}}(t),$$

and we start with $\partial_{\{a\}}(t)$. Observe that encapsulation distributes over (generalized) alternative composition, so we can push it inward until we reach an conjunctive composition in which we assume that the a-entries have been accumulated into a single entry using axiom T5. So this yields an application of the form
$$\partial_{\{a\}}(a(p) \oplus t')$$
where a does not occur in t', so using Lemma 5, this is equal to

$$\gamma(p) \oplus t'.$$

Example:

$$\begin{aligned}
\partial_{\{b\}}(a(-3) \oplus b(1) \oplus b(2) \oplus b(-3) \oplus c(3)) &= \partial_{\{b\}}(b(0) \oplus a(-3) \oplus c(3)) \\
&= \partial_{\{b\}}(b(0)) \oplus a(-3) \oplus c(3) \\
&= \gamma(0) \oplus a(-3) \oplus c(3) \\
&= a(-3) \oplus c(3).
\end{aligned}$$

Another example:

$$\partial_{\{a,b\}}(a(0) \oplus b(0)) = \partial_{\{a\}} \circ \partial_{\{b\}}(a(0) \oplus b(0)) = \partial_{\{a\}}(a(0)) = \varepsilon.$$

In applications that use information hiding (summation), we typically encounter encapsulations like this one:

$$\partial_{\{a\}}(a(2) \oplus \textstyle\sum_u (a(-u) \oplus b(u/2) \oplus c(u/2))) = b(1) \oplus c(1),$$

where instantiation of the hidden variable u is enforced by the encapsulation.

Let's see how to derive such encapsulations. First, we have, for data term p with $u \notin Var(p)$, and tuplix-closed term t that does not contain a,

$$a(p) \oplus \sum_u (a(q) \oplus t) = \sum_u (a(p+q) \oplus t).$$

Then we easily find that

$$\partial_{\{a\}}(a(p) \oplus \sum_u (a(q) \oplus t)) = \sum_u (\gamma(p+q) \oplus t)$$

using Lemma 5. In the particular case that $q = -u$ we find

$$\partial_{\{a\}}(a(p) \oplus \sum_u (a(-u) \oplus t)) = t[p/u]$$

using (5). Another example:

$$\begin{aligned}\partial_{\{a\}}(a(-6) \oplus \sum_u (a(2u) \oplus t)) &= \sum_u (\gamma(2u-6) \oplus t)\\ &= \sum_u (\gamma(u-3) \oplus t)\\ &= t[3/u].\end{aligned}$$

In the next example, the instantiation is determined within the summation:

$$\begin{aligned}\partial_{\{a\}}(\sum_u (a(u+1) \oplus b(-u/2))) &= \sum_u (\gamma(u+1) \oplus b(-u/2))\\ &= b(1/2).\end{aligned}$$

In a similar way, one can reduce

$$\partial_{\{a\}}(\sum_u (a(-u) \oplus b(u/2) \oplus c(-u/2) \oplus a(200) \oplus b(-50) \oplus c(-150)))$$

to

$$b(50) \oplus c(-250).$$

10 Example: Incremental Budgeting

A financial budget is modeled as a tuplix. We let an entry $a(p)$ represent a payment: the attribute a is used in the communication between payer and payee, and describes or identifies a transaction; we also refer to the attribute as the *channel* of the transaction, and say that the payment occurs *along* the channel. The term p represents the amount of money involved. An entry $a(p)$ with $p > 0$ stands for an obligation to pay amount p along channel a. If $p < 0$, the entry stands for the expected receipt of amount p along a.

In the following example we consider some annual budgets. In order to simplify descriptions it is assumed that various payments are due twice per year only, during periods A and B. Attributes of the form a_{A} and a_{B} are used in the specification of payments during these respective periods. Examples with monthly, weekly or daily payments can be given in a similar fashion.

Consider a budget B_{2006} containing the financial results of some entity in year 2006. E.g., take

$$B_{2006} = a_{\mathrm{A}}(30) \oplus a_{\mathrm{B}}(30) \oplus b_{\mathrm{A}}(20) \oplus b_{\mathrm{B}}(25) \oplus c_{\mathrm{A}}(-107).$$

On the basis of this realized budget, an allocated budget for 2007 covering corresponding entries could be specified as, e.g.,

$$B_{2007} = a_{\mathrm{A}}(32) \oplus a_{\mathrm{B}}(32) \oplus b_{\mathrm{A}}(21) \oplus b_{\mathrm{B}}(28) \oplus c_{\mathrm{A}}(-116).$$

Assuming that a 2008 budget is to be determined without having 2007 realization figures available, several ways to adapt the 2007 budget to a 2008 budget can be imagined. The widespread (and well-documented, see, e.g., [14]) strategy of *incremental budgeting* implies that the 2007 budget is taken as the point of departure for designing a 2008 budget. For 2008 one may consider two possible budgets: an ad hoc increase of each entry, leading to something like

$$B_{2008} = a_{\mathrm{A}}(33) \oplus a_{\mathrm{B}}(33) \oplus b_{\mathrm{A}}(22) \oplus b_{\mathrm{B}}(30) \oplus c_{\mathrm{A}}(-123),$$

or, alternatively,

$$B'_{2008} = (1 + (i/100)) \cdot B_{2007}$$

which adjusts each 2007 entry with the same inflation percentage i.

Yet another option for a 2008 budget is to adjust the 2006 realization B_{2006} with inflation twice. This yields

$$B''_{2008} = (1 + (i/100))^2 \cdot B_{2006}.$$

Still another option for the definition of a 2008 budget is the average

$$B'''_{2008} = (1/2) \cdot (B'_{2008} \oplus B''_{2008})$$

of the latter two budgets.

11 Example: Modular Budget Design

Modular financial budget design is a necessity in large organizations, assuming that budgets are at all used, i.e., that 'beyond budgeting' [2] is not (yet) the dominant strategy for financial planning. Financial budgets are probably the most complex budgets around which calls for modularity. Surprisingly, however, we have not been able to find any literature about the subject of formalized modular budget design. In the example of this section we will outline how tuplix calculus can support modular budget descriptions in a meaningful way. The example is presented in abstract terms but its origin is practical.

We consider an organization that consists of the following constituents:[1]

- Part S is a *financial source* which correlates with production figures.
- Part C is a *control group* that dispatches the incoming financial stream to the production units and the service center SC.

[1] In the practical case behind the example, S is a university division, C represents a graduate school, the PUs represent different master programs, SC provides various forms of support ranging from student counseling to timetabling, and CAP represents a department from which educational staff will be used.

- Parts PU_1 and PU_2 are *production units*. These units produce the same two types of products (type 1 and type 2).
- Part SC is a shared *service center* providing services needed by the production units.
- Part CAP is a *capacity group* from which both production units draw manpower.

Streams of money between these parts run as depicted in this figure:

$$\begin{tikzcd}
 & & \mathrm{PU}_1 \arrow[dr, "js_1{,}ss_1"] & \\
\mathrm{S} \arrow[r, "a_1{,}a_2"] & \mathrm{C} \arrow[ur, "b_1"] \arrow[dr, "b_2"'] \arrow[dd, "c"'] & & \mathrm{CAP} \\
 & & \mathrm{PU}_2 \arrow[ur, "js_2{,}ss_2"'] & \\
 & \mathrm{SC} & &
\end{tikzcd}$$

The labels (a_1, a_2, etc.) on the arrows in this picture are attribute names that will be used in the specification of payments.

The financial source rewards the production by the production units: for each product that is produced, a constant reward (depending on the type of the product) is allocated to the control group C. The control group will dispatch the rewards for both product types to the production units and to the service center. The production units receive money from C, and pay money to the capacity group in return of junior staff capacity as well as senior staff capacity.

We specify budgets for the parts S, C, PU_1 and PU_2, and we will examine how to compose one joint budget B from these. All budgets involved specify the same period of time (e.g., the calendar year 2008). We take a stepwise approach and specify the budgets in two phases taking increasingly more aspects into account. In the first stage, both production units obtain an equal reward, independent of their contribution to the total production. The budgets are defined as follows.

- The financial source S rewards production: for each product of type i that is produced ($i = 1, 2$), a constant reward reward_i is allocated to the control unit C. For production unit PU_i and product type j, the data variable

  $$n_{ij}$$

  stands for the number of products of type j produced by PU_i during the period that is covered.

  The budget:

  $$B_{\mathrm{S}} \stackrel{\mathrm{def}}{=} a_1(\mathit{reward}_1 \cdot (n_{11} + n_{21})) \obar a_2(\mathit{reward}_2 \cdot (n_{12} + n_{22}))$$

- The control unit C receives the rewards from the financial source. The amount paid to the service center SC is a fraction k (a value between 0 and 1) of the incoming stream, independently of the use that is made of it. It further pays each production unit half of the reward total. (Observe that, unless $k = 0$, this budget is not *balanced*: the expenses are higher than the income.)

$$\begin{aligned}B_{\mathrm{C}} \overset{\mathrm{def}}{=} \textstyle\sum_u \sum_v (&\\ &a_1(-u) \oplus a_2(-v) \oplus\\ &c(k \cdot (u+v)) \oplus\\ &b_1((u+v)/2) \oplus b_2((u+v)/2))\end{aligned}$$

- Budgets for the production units:

$$\begin{aligned}B_{\mathrm{PU}_1} &\overset{\mathrm{def}}{=} \textstyle\sum_u (b_1(-u) \oplus js_1(u/2) \oplus ss_1(u/2))\\ B_{\mathrm{PU}_2} &\overset{\mathrm{def}}{=} \textstyle\sum_u (b_2(-u) \oplus js_2(u/2) \oplus ss_2(u/2))\end{aligned}$$

  In this first version the production units obtain an equal amount of funding, which is spent in equal parts on senior staff (via ss_1 and ss_2) and on junior staff (via js_1 and js_2).

The combined budget:

$$B \overset{\mathrm{def}}{=} \partial_{\{a_1,a_2,b_1,b_2\}}(B_{\mathrm{S}} \oplus B_{\mathrm{C}} \oplus B_{\mathrm{PU}_1} \oplus B_{\mathrm{PU}_2}).$$

We find

$$\begin{aligned}B = \textstyle\sum_u (&\\ &\gamma(u - \mathit{reward}_1 \cdot (n_{11} + n_{21}) - \mathit{reward}_2 \cdot (n_{12} + n_{22})) \oplus\\ &c(k \cdot u) \oplus js_1(u/4) \oplus ss_1(u/4) \oplus js_2(u/4) \oplus ss_2(u/4)).\end{aligned}$$

A straightforward derivation of this identity leads to a closed term without summation; in the expression above we have introduced a 'let-binding' (see the example in Section 8) with variable u to improve the readability.

In the second stage of the budget, we take into account that the production units need funding proportional to their production volume, and may spend their resources on senior staff capacity and junior staff capacity in different proportions. Moreover, the control unit also charges the production units for the costs of the services provided by SC. This leads to the following refinement of the budgets:

$$\begin{aligned}B_{\mathrm{C}} \overset{\mathrm{def}}{=} \textstyle\sum_u \sum_v (&\\ &a_1(-u) \oplus a_2(-v) \oplus\\ &k \cdot c(u+v) \oplus\\ &(1-k) \cdot (b_1((n_{11}/(n_{11}+n_{21}))u + (n_{12}/(n_{12}+n_{22}))v) \oplus\\ &\qquad b_2((n_{21}/(n_{11}+n_{21}))u + (n_{22}/(n_{12}+n_{22}))v)))\\ B_{\mathrm{PU}_1} \overset{\mathrm{def}}{=} \textstyle\sum_u (&b_1(-u) \oplus js_1((1/4)u) \oplus ss_1((3/4)u))\end{aligned}$$

$B_{\mathrm{PU}_2} \overset{\mathrm{def}}{=} \sum_u (b_2(-u) \oplus js_2((1/3)u) \oplus ss_2((2/3)u))$

Additional phases that take more aspects into account can be easily imagined. For instance both production units may be given a fixed amount of funding independent of production volume and the remaining funding spread in proportion with production volume. That distribution strategy for C allows one unit to proceed when its production is low thus awaiting a next phase with better circumstances.

12 Conclusion

We have introduced a calculus for tuplices. It has an underlying data type called quantities which is required to be modeled by a zero-totalized field. We started with the core tuplix calculus CTC for entries and tests, which are combined using conjunctive composition. We defined a standard model and proved that CTC is relatively complete with respect to it. We further defined operators for choice, information hiding, scalar multiplication, clearing and encapsulation. We ended with two examples of applications; one on incremental financial budgeting, and one on modular financial budget design.

We refer to [6] for a discussion on the formalization of financial budgets. It also contains a more elaborate application of the tuplix calculus in the style of the example in Section 11.

Further related work seems to be scarce. We mention here the work of Elsas et al. [9, 11] on audit theory, and the work of Bergstra and Middelburg [5] on computational capital. Both are theoretical approaches that apply process theory in the analysis of organizations dealing with money streams: the former uses Petri nets, the latter process algebra. In this, they focus more on behavioral aspects than we do.

An immediate issue for future work is the completeness of BTC for open terms, and consequently the completeness of BTC with summation. We would further like to connect this theory to the formalization of interface groups and financial transfer architectures studied in [7].

References

[1] J.C.M. Baeten and W.P. Weijland. *Process Algebra.* Cambridge Tracts in Theoretical Computer Science 18, Cambidge University Press, 1990.

[2] BBRT, `http://www.bbrt.org/`.

[3] J.A. Bergstra, Y. Hirschfeld, and J.V. Tucker. Meadows. `http://www.science.uva.nl/~janb/FAM/meadowsVNR.pdf`, April 29, 2007.

[4] J.A. Bergstra and J.W. Klop. Process algebra for synchronous communication. *Information and Control* 60(1–3):109–137, 1984.

[5] J.A. Bergstra and C.A. Middelburg Parallel processes with implicit computational capital. Electronic report PRG0610, Section Software Engineering, University of Amsterdam, December 2006. Available at `http://www.science.uva.nl/research/prog/`.

[6] J.A. Bergstra, S. Nolst Trenité, and M.B. van der Zwaag. Towards a formalization of budgets. Electronic report PRG0712, Section Software Engineering, University of Amsterdam, December 2007. Available at `http://www.science.uva.nl/research/prog/`.

[7] J.A. Bergstra and A. Ponse. Interface groups and financial transfer architectures. arXiv:0707.1639v1 at `http://arxiv.org/`, July 2007.

[8] J.A. Bergstra and J.V. Tucker. The rational numbers as an abstract data type. *Journal of the ACM* 54(2), 2007.

[9] Ph.I. Elsas. *Computational Auditing.* PhD thesis, Vrije Universiteit, Amsterdam, 1996.

[10] W. Fokkink. *Introduction to Process Algebra.* Texts in Theoretical Computer Science, Springer-Verlag, 2000.

[11] P.R. Griffioen, Ph.I. Elsas, and R.P. van de Riet. Analyzing enterprises: the value cycle approach. In: *Database and Expert Systems Applications: 11th International Conference, DEXA 2000*, pages 685–697, Lecture Notes in Computer Science 1873, Springer-Verlag, 2000.

[12] J.F. Groote and A. Ponse. The syntax and semantics of μCRL. In: A. Ponse, C. Verhoef and S.F.M. van Vlijmen (editors), *Algebra of Communicating Processes '94*, pages 26–62, Workshops in Computing Series, Springer-Verlag, 1995.

[13] S.P. Luttik. On the expressiveness of choice quantification. *Ann. Pure Appl. Logic* 121(1):39–87, 2003.

[14] H.J. Tucker. Incremental budgeting: myth or model? *Western Political Quarterly* 35(3):327–338, 1982.

Electronic Reports Series of the Programming Research Group