CHOICE AS A FIRST CLASS CITIZEN

RICHARD BUCKLAND
Department of Computing, Macquarie University, Sydney
Distributed Research Group, Microsoft Institute, Sydney
AUSTRALIA
E-mail: richardb@macadam.mpce.mq.edu.au

ABSTRACT

The paper explicates a model of processes in which choices are given the same status as transitions, this allows them to be distinguished, compared and analysed algebraically. This treatment of choice is needed to support the development of process models which can represent a process throughout its entire development phase: from initial specification to implementation. We show that such a symmetric approach to choice is necessary to be able to correctly reason about the liveness and safety properties of a process. The process model developed uses higher dimensionality to code the extra information needed to represent choices.

1. Introduction

Of the established high level models of computation, including automata, CSP, CCS, Petri nets, Mazurkiewicz traces and event structures, none treats choices as individual entities. For each choice within a process the models express the fact that there is a choice and they record the alternative outcomes of the choice, but the actual choice — the decision procedure — is abstracted away.

This is entirely appropriate when describing processes at a high level of abstraction. However in the course of refining an abstract process into an actual program, by transforming a specification into incrementally more concrete specifications, this missing information needs to be incorporated at some point. In order to provide a rigorous treatment of the progression from the initial specification of a process to its implementation it is necessary that the same model of computation be used throughout.

The aims of this paper are twofold. Firstly to argue for an explicit representation of choice in process specifications, and secondly to begin to develop a model of processes which supports such a treatment of choice.

2. Making Choices Explicit

It is clear that it is necessary to incorporate choice information in process

\footnote{The author gratefully acknowledges a Microsoft Fellowship and a Commonwealth APA}
specifications at some stage — simply to be able to construct the process. It is also important to provide some information about choices in a process specification even when the specification is a highly abstract one. This is because the behavior of the process cannot be determined without knowing the interactions between its choices. In particular if all choices are left unspecified then the process must be regarded as capable of performing a large number of traces: given by all possible combinations of the outcomes of the choices. Typically however many of these combinations will not be possible. Indeed the nature of process specification typically leads to an abundance of such not-possible combinations.

For example consider a division routine in a CPU. Whenever called it makes a choice between crashing the system (if the divisor is zero) or returning the result. Suppose we have a process which checks the divisor first. If it is zero the process raises an exception and if not it calls the division routine. The combined system has only two possible traces and never crashes. However if we were unable to consider the interaction between the choices we would consider the system to have three traces.

Thus without some method of determining specific choices and their interactions we not only lose information about choices but may also lose information about the observable behavior (the trace structure) of the process — we retain "impossible" traces. This has important implications when attempting to determine liveness or safety properties; for example in the detection of deadlock.

There is therefore a need to be able to represent choices explicitly in process models, that is, to be able to treat them as objects.

3. Choices as Objects

Choice has traditionally been treated as a second class citizen in comparison to transition. We can label transitions or leave them unlabelled, we can compare them, describe their interactions, and record when they are instances of the same underlying action. With choices, on the other hand, we typically only note their existence and in some cases group them into broad categories (e.g. internal Vs external).

However in a process each choice is an individual entity, such as "if today is a weekday" or "if \( x = 0 \)". Processes may involve random choices about which nothing is known, probabilistic choices, choices determined by the environment, and so on. Treating choices in the same manner as transitions would allow us to leave them unspecified when modeling at high levels of abstraction, and to denote the actual choice or at least the class of choice when modeling at lower levels of abstraction.

To treat choice in this manner we must have elements denoting choices in the process representation. Consider a traditional transition diagram. It is
essentially a two-dimensional structure with states zero-dimensional elements, and transitions one-dimensional elements. The two-dimensional nature of the diagram arises from its branching. For each binary branch in the diagram there is a particular choice that mediates between the one-dimensional alternatives. However there is no actual element of the graph denoting this choice — the presence of the choice is simply indicated by the branch, or rather, by the presence of a two-dimensional region between the branches.

In contrast to the traditional two dimensional transition diagram higher dimensional transition systems (the constructive pasting schemes, or sometimes simply schemes\(^{1,5}\)) can be used to explicitly model choices. These are transition systems which include the regions between branches as explicit elements. Thus:

\[
\begin{array}{c}
\text{0} \quad \text{1} \quad \text{2} \\
\text{a} \quad \text{b} \\
\text{c} \\
\end{array}
\]

denotes the process which makes a choice \( \alpha \) between \( \langle a, b \rangle \) and \( \langle c \rangle \), whereas in a traditional transition system the corresponding diagram:

\[
\begin{array}{c}
\text{0} \quad \text{1} \quad \text{2} \\
\text{a} \quad \text{b} \\
\text{c} \\
\end{array}
\]

denotes the process which simply makes some choice between \( \langle a, b \rangle \) and \( \langle c \rangle \).

4. Higher Dimensional Schemes

In this section we introduce the algebraic constructs which will be used to model processes — higher dimensional schemes. Informally these can be viewed as directed graphs comprised of arrows (called cells) of various dimensions. Readers uninterested in the formal definition of schemes may wish to turn to \( \S 4.2 \) Examples of Schemes.

In the following subsection we provide the complete axiomatisation of schemes. We will introduce two binary operators which combine pairs of schemes to construct new schemes. Schemes will be defined in a recursive manner as those collections of cells which can be produced from zero dimensional cells using the operators in an interated manner.

4.1. Definitions

We first define the set \( \text{Cell} \) whose elements are called cells. A cell is a tuple \( (l, n, D, C) \) with \( n \in \mathbb{N} \), where \( D, C \) are subsets of \( \text{Cell} \), and where \( l \) is a label chosen from some alphabet. We will refer to \( n \) as the dimension of \( x \) and we will call the sets \( D \) and \( C \) the domain and codomain of \( x \) respectively. For
convenience we refer to the components of the tuple as $id(x), |x|, d(x)$ and $c(x)$ respectively. Two cells $a, b$ are then equal if, and only if, $id(a) = id(b), |a| = |b|, d(a) = d(b)$ and $c(a) = c(b)$.

We are concerned with certain sets of cells, the schemes. The dimension of a scheme will be the dimension of the highest dimensioned cell which occurs in the scheme. A scheme $P$ of dimension $n > 0$ has a domain and a codomain which are themselves schemes of dimension $n - 1$, given by the functions $\text{dom}(P)$ and $\text{cod}(P)$ respectively.

**Definition 0.1 (Point)** A singleton set consisting of a 0-dimensional cell $\{(l, 0, \{\}, \{\})\}$, with $l \in$, is a constructive pasting scheme of dimension 0, called a point.

**Definition 0.2 (Simple)** A simple constructive pasting scheme $S$ is a constructive pasting scheme of dimension $n$ for which all cells of dimension $i < n$ lie in the domain or codomain of some cell of dimension $i + 1$ of the scheme.

**Definition 0.3 (Plaster)** If $A$ and $B$ are distinct schemes of dimension $n$ which, when $n > 0$, satisfy:

$$\begin{align*}
\text{dom}(A) & = \text{dom}(B) \\
\text{cod}(A) & = \text{cod}(B) \\
A \cap B & = \text{dom}(A) \cup \text{cod}(B)
\end{align*}$$

then we say $A$ and $B$ are plasterable and the set $A \star_{x} B$ given by $A \cup B \cup \{(x, n + 1, A, B)\}$ is by definition a scheme of dimension $n + 1$ with domain $A$ and codomain $B$. The operation $\star$ is called a plaster.

**Definition 0.4 (Source, Target)** A scheme $P$ comes equipped with a dimension indexed collection of source functions $s_i$, one for each $i \in \mathbb{N}$, defined as follows:

$$s_i(P) = \begin{cases} 
P & \text{if } i \geq |P| \\
\text{dom}|P|^{-i}(P) & \text{otherwise}
\end{cases}$$

and a corresponding collection of target functions $t_i$ defined in terms of $\text{cod}(P)$ in the same manner.

Thus if $A$ is a three-dimensional scheme $s_2(A)$ is the domain of $A$, $s_1(A)$ is the domain of the domain of $A$, and $s_i(A)$ is $A$ itself for all $i \geq 3$.

**Definition 0.5 (Paste)** If $A$ and $B$ are distinct schemes of dimension $m$ and $n$ respectively which satisfy:

$$s_i(A) = t_i(B) = A \cap B$$

(2)
then the set $A \circ_i B$ given by $A \cup B$ is by definition a scheme of dimension $p = \max(m, n)$, with domain and codomain schemes defined as follows:

$$
\begin{align*}
\text{dom}(A \circ_i B) &= \begin{cases} 
\text{dom}(B) & \text{if } |A| \leq |B| = i + 1 \\
sp_{-1}(A) \circ_i sp_{-1}(B) & \text{otherwise}
\end{cases} \\
\text{cod}(A \circ_i B) &= \begin{cases} 
\text{cod}(A) & \text{if } |B| \leq |A| = i + 1 \\
t_{p-1}(A) \circ_i t_{p-1}(B) & \text{otherwise}
\end{cases}
\end{align*}
$$

The operation $\circ$ is called a \textit{paste}.

**Remark**  Space does not permit the proof that the domains and codomains above are well defined and uniquely determined. A proof of this result is given elsewhere\(^1\).

**Definition 0.6 (Scheme)** A \textit{Scheme} in a set $X$ of cells is a subset of $X$ which arises from points in $X$ by iterated paste and plaster.

### 4.2. Examples of Schemes

In this subsection we give an informal description of schemes and list some examples of them.

It is convenient to view a scheme as a topological object. A scheme is essentially a uniformly directed piece of space of some dimension, say $n$, bounded by its domain and codomain which are uniformly directed pieces of $(n-1)$-dimensional space. For example, when $n = 2$:

are schemes with domains $\{a, b, d, 0, 1, 2, 3\}$ and $\{u, v, w, x, 4, 6, 7, 8, 9\}$, and codomains $\{c, d, 0, 2, 3\}$ and $\{s, x, 4, 5, 9\}$ respectively. The 2-cell $\alpha$ has domain $\{a, b, 0, 1, 2\}$ and codomain $\{c, 0, 2\}$. The 1-cell $\nu$ has domain $\{6\}$ and codomain $\{7\}$. The 0-cells have no orientation.

An example of a three-dimensional scheme containing a single 3-cell is:

which is actually a tetrahedron which has been broken in half for ease of representation. The horizontal double arrow $\alpha$ represents the directed three-dimensional space inside the tetrahedron.
Schemes are constructed from smaller schemes by pasting and plastering. The plaster construction describes the encapsulation of an atomic \((n + 1)\)-dimensional piece of space by its \(n\)-dimensional boundaries. It essentially creates an \((n + 1)\)-dimensional pasting scheme from two \(n\) dimensional schemes. Thus:

\[
\begin{array}{c}
\bullet \rightarrow \bullet \\
\end{array}
\quad \text{and} \quad
\begin{array}{c}
\bullet \\
\rightarrow \\
\end{array}
\]

are the results of plastering two 0-dimensional schemes and two 1-dimensional schemes respectively.

Pasting is simply the concatenation of two schemes across a common \(i\)-dimensional face. For example:

\[
\begin{array}{c}
\bullet \downarrow \bullet \\
\end{array}
\quad \text{and} \quad
\begin{array}{c}
\bullet \\
\downarrow \\
\end{array}
\]

are the consequences of a 0 dimensional paste and a 1 dimensional paste respectively.

There are a number of pictures of schemes in Street\(^{10}\).

4.3. Properties of Schemes

Constructive pasting schemes satisfy the following properties:

**Lemma 0.7** For any scheme \(P\) of dimension \(n \geq 2\):

\[
\text{dom(dom}(P)\text{)) = \text{dom(cod}(P)\text{)), \quad \text{and dually}
\]
\[
\text{cod(cod}(P)\text{)) = \text{cod(dom}(P)\text{)).}
\]

**Lemma 0.8** A scheme is connected.

**Lemma 0.9** The top dimensional cells of a scheme \(P\) have compatible orientation; that is no scheme contains cells \(x, y, z\) such that:

\[
|x| + 1 = |y| = |z| = |P|, \quad y \neq z, \quad x \in \text{dom}(y), \quad x \in \text{dom}(z)
\]

and dually.

Lemmas 4.7–4.9 can be proved by a straightforward induction on the constructors paste and plaster\(^{1}\).

4.4. Examples of Non-Schemes

The following are not constructive pasting schemes:

\[
\begin{array}{c}
(1) \bullet \\
\end{array}
\quad
\begin{array}{c}
(2) \bullet \rightarrow \bullet \\
\end{array}
\quad
\begin{array}{c}
(3) \bullet \\
\end{array}
\]
5. Denoting Processes

In this section we develop the link between an abstract process and the scheme which denotes it.

5.1. States, Transitions, Sequential Composition

In the familiar manner 0-cells will denote states, and 1-cells the events which occur as transitions between them. Sequential composition is given by 0-dimensional pasting. For notational convenience where distinct events are in some sense different instances of the same action their labels will be distinguished by primes. Thus

\[
\begin{array}{c}
\text{hiccups} \\
0 \xrightarrow{} 1 \xrightarrow{} 2
\end{array}
\]

is a process which starts in state 0, hiccups twice, then finishes in state 2. Where no ambiguity arises the primes may be omitted.

5.2. Choice

In this model higher dimensional cells denote choice. Thus, as noted in §3:

\[
\begin{array}{c}
0 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{c} 2
\end{array}
\]

is a process with two possible traces: \( <a,b> \) and \( <c> \). The selection between these behaviors is given by \( x \). Note that this does not mean that \( x \) is constrained to be a deterministic choice, \( x \) may be wholly non-deterministic, probabilistic, or deterministic. At this stage we are only recording the fact that there is a choice and giving it a label to distinguish it from the infinite range of other choices that could also be used to select between \( <a,b> \) and \( <c> \). The nature of the actual choice denoted by \( x \) will determine the algebraic properties of the 2-cell and is of particular interest in deciding questions of process simulation and equivalence.

Compound choices are denoted in a straightforward manner. For example

\[
\begin{array}{c}
1 \xrightarrow{c} 2 \\
2 \xrightarrow{f} 3 \\
0 \xrightarrow{a} 3
\end{array}
\]

is the process which initially makes choice \( x \) between the process which does
<a>, and the process which first does b then makes a choice y between <o> and <e, f>. The 2-cells x, y jointly arbitrate between the three possible traces of the system.

In the above example it is clear from the orientation of the one-dimensional structure that choice x is evaluated first, then y. The situation is less clear in the case of:

Here the order of evaluation will be given by the orientation of the 2-cells; thus y then x. In general where there is ambiguity in evaluating order at dimension i it will be resolved recursively by the ordering on cells at dimension i + 1 (the existence of such a total order on i-cells is proved in Buckland 1).

A 3-cell will denote a meta-choice, that is a choice between two processes comprised of two-dimensional choices. Thus the tetrahedron of §4.2 above comprises a three-dimensional choice between two pairs of two-dimensional choices, and jointly these arbitrate over the four possible traces of the process. Similarly 4-cells are meta-meta choices, and so on.

6. Specification = Flattening

It is evident that a high dimensional meta-choice can always be "flattened" to a collection of two-dimensional choices in precisely the same way a collection of nested if's can be written as a single "flat" case statement. Similarly any higher dimensional n-scheme A_n can be flattened into some two-dimensional scheme A_2.

It might seem then that the two representations A_n and A_2 are equally useful and that the higher dimensional information of A_n is superfluous. However there is a sense in which A_n is a better representation for processes whose choices are not fully specified. This is because we have full information about the structure of A_n regardless of what its choices turn out to be. That is, we have all the cells of A_n, we know their dimension, their orientation, and we know how they fit together. This is precisely the information about a process which we need to know to be able to combine it with other processes to form larger systems. On other hand, as shown in §2, the structure of A_2 depends upon the as yet unspecified choices. As a result we cannot combine A_2 with other processes until its choices have been fully specified.

Higher dimensional schemes are fully abstract specifications of processes. Using the A_n form nothing need be known about the particular events or choices of a process for it to be sequentially composed with other processes, for it to be combined with other processes in a choice, or for it to be concurrently executed.
alongside another process. As choice cells are specified and the processes become more concretely specified their schemes can be flattened to progressively lower dimensions.

The need for higher dimensional cells therefore arises from the desire to be able to work with processes of partially specified choices. Dimensionality provides the necessary algebraic support for choice refinement, the process of incrementally adding more information about particular choices to a specification. This is an intuitively pleasing result as it mirrors the known observation\(^8\) that dimensionality supports transition refinement.

7. Nesting Choices

Processes are often specified with nested (meta) choice structure. We have seen above how higher dimensional cells denote individual choices — in this section we address the issue of how to construct the processes which contain them. The construction is trivial at dimension two, but the concept of a well defined nesting needs to be made more precise before we can proceed to higher dimensions.

In particular there is a requirement of nesting that is rarely made explicit: we require inner choices to be encapsulated, to be completed choices. An example illustrates this graphically. Consider the following fragment of pseudo-code:

```
...  
...  
Case C=1
   If P then
     ...
   ...
EndCase
   Else
     ...
   ...
EndIf
```

Whilst this code has some of the appearance of nesting it is not well defined. To have structured nesting inner choices cannot be intermingled with the structure of outer choices, that is, the inner choices must in some sense be encapsulated. In a programming language encapsulation is typically provided by begin–end pairs and schemes can be encapsulated in the same way.
For example:

\[
\begin{array}{c}
\text{\(b\)} \\
\Downarrow \text{always} \\
\text{\(\alpha\)} \\
\Downarrow \text{never} \\
\text{\(a\)}
\end{array}
\]

becomes

\[
\begin{array}{c}
\text{begin} \\
\downarrow \text{always} \\
\text{\(b\)} \\
\downarrow \alpha \\
\text{\(a\)} \\
\downarrow \text{never} \\
\text{end}
\end{array}
\]

where \textit{always} and \textit{never} are the evident choices. The two schemes will have the same behavior, the encapsulation is simply a marker to record that the choice is complete and available for nesting.

8. Concluding Remarks

The key motivation for this paper has been to develop an algebraic process model which can be used to represent processes at all stages of their development — from high level abstract specification to low level implementation.

To achieve this result it is necessary to have the facility to explicitly represent choices and their interactions. We have shown that using higher dimensional structure in the process model allows not-fully-specified choices and actions to be represented in a useful manner. In particular the representation we develop allows processes to be composed and combined in the usual ways independent of the extent to which their specifications remain open.

Incremental specification of a process typically results in two distinct types of refinement: choice refinement and action refinement. Choice refinement occurs when choices left open at higher levels of abstraction become fixed. In this paper we have shown that the higher dimensional model successfully provides for choice refinement and the change in the potential one-dimensional (observable) behaviors of the system which can arise as a consequence: via the mechanism of flattening. Elsewhere\textsuperscript{2} we show that the model also provides a \textit{true concurrency} treatment of action refinement.

At any level of abstraction the dimensionality of the model can be reduced to recover the familiar two-dimensional traces, although in flattening the schemes we throw away the information that would allow further refinement to be modeled.

A uniform treatment of choice is an attractive goal. A program\textsuperscript{3} to calculate with high dimensional schemes has been written to assist in the further classification of the choices and interactions of existing process models.

9. Acknowledgements

The author wishes to thank Mike Johnson and David Murphy for many
discussions and insightful comments. Thanks also to Robin Cockett, John Potter, John Power, Vaughan Pratt for fruitful discussions and suggestions at various stages throughout the work.

10. References