Schedules for Multiset Transformer Programs

Michel Chaudron
Rijksuniversiteit Leiden
Department of Mathematics and Computing Science
P.O. Box 9512
2300 RA Leiden, Netherlands
chaudron@cs.leidenuniv.nl

Abstract

The GAMMA formalism is a programming model based on the nondeterministic rewriting of multisets. Programs in this formalism are expressed with a minimum of control, which makes the model very well-suited for writing parallel programs at a high level of abstraction. In the implementation, however, the chaotic behaviour of GAMMA programs poses serious problems. To overcome these problems, we introduce a language that can be used to effectively control the nondeterministic behaviour. The control information is specified separately from the program text, which allows the logic of a program to be treated separately from control issues.
1 Introduction

The multiset transformer formalism (Gamma [1], [2]) provides a framework in which programs can be expressed with a minimum of (explicit) control. This absence of control is an advantage in the design phase because it allows programs to be expressed at a high level of abstraction where aspects of underlying hardware (sequential or parallel architecture, shared memory or message passing communication) do not have to be taken into account. This absence of control becomes a serious drawback in the implementation phase of a program. Without explicit control, the formalism allows many ways to execute a program. Not all of these many ways are efficient (with respect to the resources space and time). Ideally, efficient executions are obtained automatically. Alternatively, as is done for functional languages [6], explicit scheduling information may be added to a program.

In this paper we present a language that can be used to specify control-flow information for multiset transformer programs. By imposing some flow of control we schedule the actions of a program in time. For this reason we call the representation of control information a schedule. By introducing control-flow the programmer can guide the execution to obtain efficiency. We had several aims in mind when designing this language:

- With the language it should be possible to describe operational behaviour (in terms of execution orders) from a spectrum of possibilities. This spectrum ranges from the highly nondeterministic execution of multiset transformer programs to the completely deterministic behaviour of known algorithms.

- The language should be implementable.

- It should be possible to reason about the time complexity of a schedule (relative to other schedules).

A schedule is specified separately from the text of the multiset transformer program. This pushes the idea of treating complexity issues separately from correctness issues. The program text forms the logic component which is related to the correctness of an algorithm; the control component, given by a schedule, is responsible for the time complexity of the solution. The use of separate representations for logic and control is essential for combining portability and efficiency [7]. The logic component of a program is portable to any kind of machines. The operational behaviour of a program can be tuned to machine specific characteristics by adjusting the schedule.

In Section 2 we briefly introduce the multiset transformer formalism, and we present its semantics. Subsequently we introduce schedules in Section 3, and define their meaning. In Section 4 we relate scheduled programs to chaotic programs; i.e. programs without a schedule. To this end we introduce a generic schedule that describes the behaviour of the chaotic execution mechanism. This schedule provides the starting point for refinement into more efficient schedules.
2 Multiset Transformer Programs: Syntax and Semantics

A multiset transformer program consists of a number of conditional rewrite rules on multisets. Multisets can be formed over arbitrary domains of values. Execution of a program consists of repeatedly applying the rules of the program to a given multiset; when none of these rules apply, execution terminates. This execution method is nondeterministic in two ways:

• the order of the execution of rules is arbitrary, and
• the data to which a rule is applied, is chosen arbitrarily.

Because of this highly unpredictable behaviour multiset transformer programs are also referred to as chaotic programs. This in contrast to scheduled programs, which we will introduce in the next section.

Multiset transformation rules are written as conditional (multiset) rewrite rules \( \bar{x} \rightarrow m \leftarrow b \), where \( \bar{x} \) denotes a sequence of variables \( x_i \), \( m \) denotes a multiset expression and \( b \) a boolean expression. The free variables in \( m \) and \( b \) are taken from \( \bar{x} \). Consider, for example

\[
\text{add} \equiv x, y \rightarrow x + y \leftarrow \text{true}
\]

definition which defines a rule for replacing two numeric values by their sum.

Execution of a rule is as follows: if multiset \( M \) contains a sequence of values \( \bar{v} \) that matches \( \bar{x} \) such that evaluation of \( b \) with every occurrence of \( x \) replaced by \( v_i \), denoted \( b[\bar{x} := \bar{v}] \), yields \text{true}, then \( \bar{v} \) is removed from \( M \) and the multiset \( m[\bar{x} := \bar{v}] \) is inserted. A configuration of a program \( P \) and a multiset \( M \) is written \( \langle P, M \rangle \). Transitions take a system from one configuration to another: \( \langle P, M \rangle \xrightarrow{\sigma} \langle P', M' \rangle \), where the label \( \sigma \) stands for the multiset substitution that transforms \( M \) into \( M' \). The following semantic rule specifies the possible transitions a program can make by executing a single rule \( r = \bar{x} \rightarrow m \leftarrow b \).

\[
\text{if } (\bar{v} \subseteq M : b[\bar{x} := \bar{v}]) \text{ then } \langle r, M \rangle \xrightarrow{\sigma_1} \langle r, M[\sigma] \rangle \text{ where } \sigma = m[\bar{x} := \bar{v}]/\bar{v}
\]

Here we write \( M[\sigma] \) to denote the multiset that results from applying the substitution \( \sigma \) to \( M \). More formally, let \( M' = m[\bar{x} := \bar{v}] \), then \( M[M'/\bar{v}] = M \oplus (M' \ominus \bar{v}) \), where \( \oplus \) and \( \ominus \) denote multiset addition and subtraction respectively. Note that for ease of notation we identify the sequence \( \bar{v} \) with the multiset consisting of the same elements as \( \bar{v} \). The fact that a transition is made by executing a single rule once is emphasized by decorating the transition relation \( \xrightarrow{\sigma_1} \) with a subscript 1.

It is also possible to execute multiple transitions of the same rule concurrently. For instance, among the possible transitions for the configuration \( \langle \text{add}, \{1, 8, 6, 9\} \rangle \) are \( \langle \text{add}, \{1, 8, 6, 9\}\rangle^{(1,8,6,9)} \langle \text{add}, \{9, 6, 9\} \rangle \) and \( \langle \text{add}, \{1, 8, 6, 9\}\rangle^{(10,9,5,15)} \langle \text{add}, \{1, 8, 15\} \rangle \). From the labels of these transitions we can see that they are concerned with transformations of different parts of the multiset. When two transitions transform different parts of the available data, then these two transitions do not interfere with each other, hence they can also happen concurrently.

This notion of non-interference is formally captured by the following definition.

**Definition.** Given a multiset \( M \) and two multiset substitutions \( \sigma_1 = M_1/N_1 \) and \( \sigma_2 = M_2/N_2 \), we say that \( \sigma_1 \) and \( \sigma_2 \) are disjoint in \( M \) if \( N_1 \oplus N_2 \subseteq M \).
This notion corresponds to the features of the EREW machine model. The following more liberal definition of non-interference provides the possibility of concurrent reading of data, as is characteristic of the CREW machine model.

**Definition**

- Given a multiset $M$ and two multiset substitutions $\sigma_1 = M_1/N_1$ and $\sigma_2 = M_2/N_2$, we say that $\sigma_1$ is independent from $\sigma_2$ in $M$ if $N_1 \subseteq M \oplus N_2 \oplus M_2$. We write $M \models \sigma_1 \triangleleft \sigma_2$ if $\sigma_1$ is independent from $\sigma_2$ in $M$.

- Given a multiset $M$ and two multiset substitutions $\sigma_1 = M_1/N_1$ and $\sigma_2 = M_2/N_2$, we say that $\sigma_1$ and $\sigma_2$ are independent in $M$ if $M \models \sigma_1 \triangleleft \sigma_2$ and $M \models \sigma_2 \triangleleft \sigma_1$. We write $M \models \sigma_1 \triangleleft \sigma_2$ if $\sigma_1$ and $\sigma_2$ are independent in $M$.

To illustrate, consider a program that computes the primes of a set of numbers. The program consists of the rule:

$$\text{primes} \equiv x, y \rightarrow x \iff (y > x) \land (y \mod x = 0)$$

Consider the execution of the primes program with a multiset $M = \{2, 3, 4, 6, 8\}$. According to the CREW definition of independent, the substitutions $\{2\}/\{2, 4\}, \{2\}/\{2, 6\}$ and $\{2\}/\{2, 8\}$ can be executed concurrently, yielding a multiset $\{2, 3\}$. The EREW policy disjoint does not allow number 2 (which occurs in the multiset only once) to be involved in (read by) multiple multiset transformations. Because we are interested in a programming model with as much inherent parallelism as possible, we will use the CREW definition of independent in the definition of our semantics.

The label assigned to a transition of multiple concurrent executions is a combination of the individual substitutions.

**Definition** Given two multiset substitutions $\sigma_1 = M_1/N_1$ and $\sigma_2 = M_2/N_2$, the composition of $\sigma_1$ and $\sigma_2$ is defined as $\sigma_1 \oplus \sigma_2 = (M_1 \oplus M_2)/(N_1 \oplus N_2)$.

The transition $\langle \text{add}, \{1, 8, 6, 9\}\rangle \langle 1, 8, 6, 9\rangle \sim \langle \text{add}, \{8, 15\}\rangle$ is obtained by the concurrent execution of the two transitions mentioned in the previous example. Note that we have dropped the subscript 1 of the transition relation $\sim$.

A configuration $\langle r, M \rangle$ is terminal if no combination of the elements in $M$ satisfies the condition of $r$. We use the convention that terminal configurations are marked by the $\sqrt{\_}$ symbol: $\langle r, M \rangle \sqrt{\_}$. The following semantic rule defines terminal configurations.

$$\text{if } \neg (\exists \overline{v} \subseteq M : b[\overline{v} := \overline{v}]) \text{ then } \langle r, M \rangle \sqrt{\_}$$

More complex programs may be constructed using the combinators ‘o’ and ‘+’. Rules may be combined using the ‘+’ combinator into what are called simple programs [3]. A simple program is of the form $r_1 + \cdots + r_n$; any number of independent instances of the constituent rules may be executed concurrently. Simple programs may be composed sequentially using the ‘o’ operator; the final state of one program is used as initial state of the next. The program terms derivable in this way are ‘products of sums’; i.e. are of the form $(r_1 + \cdots + r_i) \circ \cdots \circ (r_j + \cdots + r_n)$. The purpose of limiting the syntax of program terms to this form is to exclude the parallel
composition of programs that contain sequential composition; e.g. \( P_1 + (P_2 \circ P_3) \). There are two reasons for excluding these forms: firstly, the syntax obtained in this way describes exactly the same set of programs that are definable by the original GAMMA model presented in [1] and [2]. Secondly, the excluded terms present difficulties with the compositionality of semantics [4].

The complete syntax and semantics is presented in Table 1 and 2.

### Table 1: Abstract Syntax of Multiset Transformer Programs

<table>
<thead>
<tr>
<th>Syntactic Categories</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r \in \text{Rule} )</td>
<td>( r ::= \overline{x} \to m \Leftarrow b )</td>
</tr>
<tr>
<td>( R \in \text{Simple} )</td>
<td>( R ::= r )</td>
</tr>
<tr>
<td>( P \in \text{Program} )</td>
<td>( P ::= R + R )</td>
</tr>
</tbody>
</table>

\[ \begin{align*}
\text{(C0)} & \quad \langle \overline{x} \to m \Leftarrow b, M \rangle \checkmark \quad \text{if } \lnot (\exists \overline{v} \subseteq M : b[\overline{x} := \overline{v}] ) \\
\text{(C1)} & \quad \langle \overline{x} \to m \Leftarrow b, M \rangle \sim_{\mathcal{S}_1} \langle \overline{x} \to m \Leftarrow b, M[\sigma] \rangle \quad \text{if } \sigma \sqsubseteq M \land b[\overline{x} := \overline{v}] \quad \text{where } \sigma = m[\overline{x} := \overline{v}] / \overline{v} \\
\text{(C2)} & \quad \frac{\langle R, M \rangle \sim_{\mathcal{S}_1} \langle R, M' \rangle}{\langle R, M \rangle \sim \langle R, M' \rangle} \\
\text{(C3)} & \quad \frac{\langle R_1 + R_2, M \rangle \sim_{\mathcal{S}_1} \langle R_1 + R_2, M' \rangle}{\langle R_2 + R_1, M \rangle \sim_{\mathcal{S}_1} \langle R_2 + R_1, M' \rangle} \\
\text{(C4)} & \quad \frac{\langle R, M \rangle \sim_{\mathcal{S}_1} \langle R, M[\sigma_1 \oplus \sigma_2] \rangle \quad \text{if } M \models \sigma_1 \land \sigma_2}{\langle R, M \rangle \sim_{\mathcal{S}_1} \langle R, M' \rangle} \\
\text{(C5)} & \quad \frac{\langle R_1, M \rangle \checkmark}{\langle R_1 + R_2, M \rangle \checkmark} \\
\text{(C6)} & \quad \frac{\langle P_1, M \rangle \checkmark}{\langle P_2, M \rangle \sim_{\mathcal{S}_1} \langle P_1', M' \rangle} \\
\text{(C7)} & \quad \frac{\langle P_1, M \rangle \sim_{\mathcal{S}_1} \langle P_2, M' \rangle}{\langle P_2 \circ P_1, M \rangle \sim_{\mathcal{S}_1} \langle P_2 \circ P_1', M' \rangle} \\
\text{(C8)} & \quad \frac{\langle P_1, M \rangle \checkmark}{\langle P_2, M \rangle \checkmark} \\
\end{align*} \]

### Table 2: Semantics of Multiset Transformer Programs
3 Schedules

We now introduce the schedule language. This language allows the programmer to control the execution of multiset transformer programs at the level of the execution of a single rule. We first present the syntactic constructs of the schedule language with an intuitive explanation of their meaning. Also, we give a rationale for this particular set of constructs. After that, we define the formal semantics of schedules. This semantics is expressed in terms of the transitions of individual rules as discussed in the previous section.

3.1 Syntax of the Schedule Language

A schedule for a multiset transformer program imposes a partial ordering on the execution of rules in time. The syntax of the schedule language is given in Table 3.

<table>
<thead>
<tr>
<th>Syntactic Categories</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c \in \text{Boolean-Expr} )</td>
<td>( s ::= \text{skip} )</td>
</tr>
<tr>
<td>( r \in \text{Rule} )</td>
<td>( r \rightarrow s )</td>
</tr>
<tr>
<td>( s \in \text{Schedule} )</td>
<td>( s; s )</td>
</tr>
<tr>
<td>( \overline{v} \in \text{Value-List} )</td>
<td>( s</td>
</tr>
<tr>
<td>( S \in \text{Schedule-Id} )</td>
<td>( c \triangleright s )</td>
</tr>
<tr>
<td>( !s )</td>
<td>( S(\overline{v}) )</td>
</tr>
</tbody>
</table>

Table 3: Abstract Syntax of Schedules

The symbol \( \text{skip} \) stands for the empty schedule. The \emph{rule-conditional} schedule \( r \rightarrow s \) starts with an attempt to execute \( r \). When this succeeds, then execution continues with \( s \). When execution of \( r \) fails, because data that satisfies \( r \)'s condition \( b \) is not available, then the schedule \( r \rightarrow s \) reduces to \( \text{skip} \).

The examples in Section 2 already showed that we identify individual rules of a program with names, e.g. \( \text{add} \triangleq x, y \rightarrow x + y \Leftarrow \text{true} \). In schedules we shall use the names of rules instead of their definitions; For instance, we write \( \text{add} \rightarrow \text{skip} \) rather than \( (x, y \rightarrow x + y \Leftarrow \text{true}) \rightarrow \text{skip} \). Furthermore, because \( r \rightarrow \text{skip} \) is a very common construction, we abbreviate it to \( r \).

Schedules can be combined using sequential and parallel composition. These combinators are denoted by \( ; \) and \( || \) respectively. Parallel composition is interpreted as: either one of the operand schedules may execute a step, or they may do a step concurrently. Whenever both of the operands of the \( || \) can do a step, then these steps may be executed concurrently only after the execution mechanism has ascertained that these steps are independent. These steps may always be executed in interleaved fashion. We emphasize that different symbols are used for the sequential composition of schedules \( ; \) and the intuitively similar sequential composition of chaotic programs \( \circ \). Also, concurrent composition of chaotic programs is written \( \triangleright \), whereas schedules are composed in parallel using the \( || \) operator.

The \emph{replication} operator \( ! \) is used in a similar way as it is used in the \( \pi \)-calculus [8]. We adapt the definition of the \( ! \) operator for our purposes: \( !s \) denotes an arbitrary, but finite\(^1\),

\(^1\) We need to be able to detect termination of a term.
number of copies of schedule \( s \) in parallel.

The construction \( \mathcal{S}(\overline{x}) \cong s \) is used to define (and possibly parameterize) a schedule term \( s \). The free variables that occur in \( s \) are taken from \( \overline{x} \). Whenever \( \mathcal{S}(\overline{x}) \) appears in a schedule then it gets replaced by the text \( s \) with \( v_i \) substituted for all free occurrences of \( x_i \).

Condition \( c \) in a conditional schedule \( \langle c \triangleright s \rangle \) is a boolean expression over the (control) variables \( \overline{x} \) that are introduced by a definition \( \mathcal{S}(\overline{x}) \equiv s' \). If \( c \) evaluates to true, then schedule \( s \) is executed, otherwise the schedule reduces to skip. The conditional is a key construction in the definition of common control structures. For instance, a schedule that \( n \) times consecutively repeats rule \( r \) can be defined as \( \text{Iter}(n) \equiv (n > 0) \triangleright (r ; \text{Iter}(n - 1)) \).

With the machinery introduced so far, it is possible to define an ordering according to which rules are selected for execution. The choice of data for the execution of a rule \( r \) is governed by \( r \)'s condition. When a more specific choice of data is required then this can be achieved by strengthening \( r \)'s condition.

**Definition**  
Let \( r = \overline{x} \rightarrow m \leftarrow b \) then a rule \( r' = \overline{x} \rightarrow m \leftarrow b' \), where \( b' \Rightarrow b \) is called a strengthening of \( r \), written \( r' \ll r \).

Rather than scheduling a rule \( r \) from a chaotic program directly, we usually schedule a strengthened rule \( r' \). To illustrate this, consider the multiset transformer program for sorting that is defined by the rule

\[
\text{swap} \equiv (x, i), (y, j) \rightarrow (x, j), (y, i) \leftarrow (x > y) \land (i < j)
\]

A schedule that, for instance, exchanges neighbouring values only, will use a rule \( \text{swap}' \) that strengthens the condition \( (i < j) \) to \( (i = j - 1) \) to get

\[
\text{swap}' \equiv (x, i), (y, j) \rightarrow (x, j), (y, i) \leftarrow (x > y) \land (i = j - 1)
\]

To facilitate this process we shall adopt the notational convention that rule definitions are parameterized in the variables that are used for data selection. For sorting this means that we define the rule

\[
\text{swap}(i, j) \equiv (x, i), (y, j) \rightarrow (x, j), (y, i) \leftarrow (x > y) \land (i < j)
\]

This notation is actually a shorthand for

\[
\text{swap}(a, b) \equiv (x, i), (y, j) \rightarrow (x, j), (y, i) \leftarrow (x > y) \land (i < j) \land (i = a) \land (j = b).
\]

For instance, to sort a sequence of elements (indexed from 0 to \( k \)) the following schedules for Bubble Sort and OddEven Sort should be invoked by \( \text{BubbleSort}_k \) and \( \text{OddEvenSort}_k \).

\[
\begin{align*}
\text{Bubble}(i, n) & \equiv (i < n) \triangleright (\text{swap}(i, i + 1) \parallel \text{Bubble}(i + 1, n)) \\
\text{BubbleSort}(m) & \equiv (m > 0) \triangleright (\text{Bubble}(0, m) \parallel \text{BubbleSort}(m - 1)) \\
\text{OddEven}(i, k) & \equiv (i < k) \triangleright (\text{swap}(i, i + 1) \parallel \text{OddEven}(i + 2, k)) \\
\text{OddEvenSort}(m) & \equiv (m > 0) \triangleright (\text{OddEven}(m \text{ mod } 2, k) \parallel \text{OddEvenSort}(m - 1))
\end{align*}
\]

More examples of schedules can be found in [7].
3.2 Rationale for the Schedule Language

Multiset transformer programs can display behaviour that ranges from highly nondeterministic chaotic execution to behaviour that corresponds to the execution of known algorithms. We want to be able to express all the possible behaviours of multiset transformer programs in the same formalism. To this end we designed the schedule language. The operators that are present in the scheduling language have been chosen for one of two reasons.

- With schedules we need to be able to describe any partial order of actions.
- For all practical purposes, we need the schedule-representation of any partial order to be finite. Some aspects of control strategies, like the number of iterations and the number of parallel actions cannot in general be defined a priori. Hence we need constructs that evolve dynamically as a function of the input (rather than of the size of the input only).

In order to describe a partial order of actions we need to describe two things:

- the precedes/succeeds relations between actions. This is traditionally represented by the ‘;’ symbol: ‘s₁; s₂’ means that before the actions of ‘s₂’ may be executed, all actions of ‘s₁’ must be finished.
- the fact that actions are unrelated. In our setting of schedules, the unrelatedness of actions means that they can be executed concurrently. We write ‘s₁∥s₂’ to indicate that independent actions of ‘s₁’ and ‘s₂’ may be executed concurrently.

Finite representations of potentially infinite schedules can only be obtained by operators that evolve dynamically:

- Generally the exact execution ordering of individual rules cannot be known in advance. Recursion is incorporated to describe iterations of arbitrary length. The unfolding of a recursive schedule typically depends on the given multiset. Choices based on the parameters of a schedule can be specified using the ‘c ⋄ s’ construct.
- We do not know in advance how many rules are being executed concurrently at any stage in the computation. The schedule ‘!s’ evolves dynamically into the number of copies of ‘s’ that is needed.

In Milner’s π-calculus replication can be used to simulate recursion and is therefore chosen, in place of recursion, as a primitive notion. In our setting replication and recursion are complementary notions. In a sense recursion is a generalization of sequential composition, and replication is a generalization of parallel composition. Using ‘;’ we can describe the sequential composition of a fixed number of schedules; e.g. ‘s; s; s; s’. Using recursion we can define a schedule that may unfold in any arbitrary number of copies of ‘s’. Similarly, ‘∥’ can be used to compose any fixed number of schedules in parallel. Using replication we can define a schedule ‘!s’ which denotes the parallel composition of an arbitrary number of copies of ‘s’.

Recursion cannot be simulated by ‘!s’ because the number of copies that is made by replication cannot be determined by the programmer (while this is possible using recursion). Also replication ‘!s’ cannot be simulated using a recursive definition, e.g. ‘S ≡ s∥S’. This is because the semantic rule (S6) can be used to reduce a schedule containing ‘!’ to skip. Repeatedly
unfolding ‘\(S\)’ by ‘\(s\parallel S\)’ will always retain the symbol ‘\(S\)’ in the schedule term, hence it can never be reduced to skip.

### 3.3 Semantics of the Schedule Language

The semantics of the schedule language is defined as a labelled transition system. A configuration of a schedule \(s\) and a multiset \(M\) is written \(\langle s, M \rangle\). The behaviour of a schedule is given by labelled transitions \(\langle s, M \rangle \xrightarrow{\lambda} \langle s', M' \rangle\). The label \(\lambda\) is either a multiset substitution \(\sigma\) or the special symbol \(\varepsilon\). The latter is used to label transitions that do not affect the multiset \(M\). The transitions for schedules are defined in terms of the transition relation for multiset transformer programs which was presented in Section 2.

To keep our definition of the semantic rules simple, we wish to identify several expressions. A typical case is that we want ‘\(||\)’ to be commutative. We therefore define a structural congruence ‘\(\equiv\)’ to be the smallest congruence relation over a set of terms such that a number of laws hold. Terms are thus grouped together on the basis of their syntax, allowing the semantic rules to focus on behavioural aspects of the terms. This method of separating structural from behavioural issues was inspired by [3]. In Table 4 and 5 we summarize the semantics of the schedule language.
Table 4: Structural Congruences

<table>
<thead>
<tr>
<th>(S0)</th>
<th>( \langle r, M \rangle \sqrt{\text{skip}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \langle r \rightarrow s, M \rangle \leftrightarrow \langle \text{skip}, M \rangle )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(S1)</th>
<th>( \langle r, M \rangle \sigma \downarrow \langle r', M' \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \langle r \rightarrow s, M \rangle \sigma \downarrow \langle s, M' \rangle )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(S2)</th>
<th>( \langle s_1, M \rangle \lambda \rightarrow \langle s'_1, M' \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \langle s_1; s_2, M \rangle \lambda \rightarrow \langle s'_1; s_2, M' \rangle )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(S3)</th>
<th>( \langle s_1, M \rangle \lambda \rightarrow \langle s'_1, M' \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \langle s_1 \parallel s_2, M \rangle \lambda \rightarrow \langle s'_1 \parallel s_2, M' \rangle )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(S4)</th>
<th>( \langle s_1, M \rangle \sigma_1 \rightarrow \langle s'_1, M_1 \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \langle s_2, M \rangle \sigma_2 \rightarrow \langle s'_2, M_2 \rangle )</td>
</tr>
</tbody>
</table>

(5) \[ \langle s_1 \parallel s_2, M \rangle \sigma_1 \parallel \sigma_2 \rightarrow \langle s'_1 \parallel s'_2, M[\sigma_1 \parallel \sigma_2] \rangle \]

<table>
<thead>
<tr>
<th>(S6)</th>
<th>( \langle s, M \rangle \lambda \rightarrow \langle s', M' \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \langle s_1, M \rangle \lambda \rightarrow \langle s', M' \rangle )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(S7)</th>
<th>( \langle s \parallel s, M \rangle \lambda \rightarrow \langle s', M' \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \langle s_1, M \rangle \lambda \rightarrow \langle s', M' \rangle )</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(S8)</th>
<th>( \langle s[\overline{x} := \overline{y}], M \rangle \lambda \rightarrow \langle s', M' \rangle )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \langle S(\overline{y}), M \rangle \lambda \rightarrow \langle s', M' \rangle )</td>
</tr>
</tbody>
</table>

\( s \equiv t \)

\[ \langle s, M \rangle \lambda \rightarrow \langle s', M' \rangle \]

\[ s' \equiv t' \]

\[ \langle t, M \rangle \lambda \rightarrow \langle t', M' \rangle \]

Table 5: Semantics of Schedules
4 Relating Schedules to Multiset Transformer Programs

Schedules are used to control the operational behaviour of programs. One of our aims is to be able to reason about the behaviour of scheduled programs. In particular we want to be able to prove that one schedule is better in some sense than another. We know that we improved on the starting point if we can prove that a schedule is better than the chaotic behaviour. To be able to reason about this chaotic behaviour, we need a schedule that describes it. In this section we show how to construct such schedules for chaotic programs. It can be shown that the schedules obtained by this method can do everything a chaotic program can do and vice versa.

We show how to define a generic schedule and prove that it has the same behaviour as the corresponding chaotic program. Because this schedule can display all possible behaviours, we call it the most general schedule. We write $\Gamma_P$ to denote the most general schedule of a chaotic program $P$. We define the most general schedule compositionally on the structure of chaotic programs (as given by the abstract syntax in Section 2).

**Definition** Let $R$ denote a simple program $r_1 + r_2 + \cdots + r_n$ and let $P_1$ and $P_2$ be two arbitrary chaotic programs.

$$
\Gamma_R \quad \triangleq \quad ! (r_1 \rightarrow \Gamma_R \parallel r_2 \rightarrow \Gamma_R \parallel \cdots \parallel r_n \rightarrow \Gamma_R)
$$

$$
\Gamma_{P_1 \circ P_2} \quad \triangleq \quad \Gamma_{P_2}; \Gamma_{P_1}
$$

The definition of $\Gamma_{P_1 \circ P_2}$ is straightforward. The replication that occurs in $\Gamma_R$ allows a schedule to exhibit an arbitrary degree of concurrency (like the corresponding chaotic program). The rule/conditional recursive invocation of $\Gamma_R$ guarantees proper termination.

For example, the most general schedule for the sorting program that consists of the single rule `swap` is: $\Gamma_{\text{swap}} \triangleq !(\text{swap} \rightarrow \Gamma_{\text{swap}})$.

In order to formally express the behavioural equivalence between a chaotic program and its most general schedule we introduce two new transition relations. The first relation, $\langle P, M \rangle \xrightarrow{\pi, s} \langle P', M' \rangle$, is the labelled reflexive transitive closure of the transition relation for chaotic programs. The label $\pi$ is a sequence of multiset substitutions. The second, denoted $\langle s, M \rangle \xrightarrow{\pi} \langle s', M' \rangle$, is the analogue for schedules, where $\pi$ is a sequence of multiset substitutions and $\varepsilon$’s. The definition of both relations is given in Table 6. We write $\pi_1 \cdot \pi_2$ to denote the concatenation of $\pi_1$ and $\pi_2$. A sequence of labels $\pi$ from which all occurrences of $\varepsilon$ have been removed, is denoted $\pi'$. 
The operational equivalence of a chaotic program and its most general schedule is expressed by the following theorem.

**Theorem** Let $P$ be a multiset transformer program and let $M$ be a multiset.

1. If $\langle P, M \rangle \sim^* \langle P', M' \rangle$, then $\exists \lambda : \lambda = \lambda \cdot : \langle P, M \rangle \rightarrow^* \langle \text{skip}, M' \rangle$.

2. If $\langle \Gamma P, M \rangle \rightarrow^* \langle \text{skip}, M' \rangle$, then $\langle P, M \rangle \sim^* \langle P', M' \rangle$.

Part 1 of the theorem states that every behaviour that a chaotic program can display, can also be displayed by the most general schedule. Conversely, part 2 states that if the most general schedule behaves in a certain way, then the chaotic program (without the schedule) can behave in the same way. In a sense this theorem can be seen as “soundness” and “completeness” of the deduction system for schedules with respect to the deduction system for chaotic programs. The proof of this theorem is presented in the next section.

## 5 Proofs

In this section we present the proofs of Theorem 1 and 2. These theorems relate the behaviour of chaotic programs to the behaviour of the most general schedule. A behaviour of such a system is, by our choice of semantics, represented by a sequence of transitions. The structure of both proofs is as follows: We start with showing that the first transition of one system can also be made by the other system. Next, induction over the length of the transition sequence achieves the generalization from one transition to sequences of transitions of arbitrary length.

For the sake of brevity, we use some auxiliary notation.

**Definition** Let $P = r_1 + \cdots + r_n$.

\[
\Pi P = (r_1 \rightarrow \Gamma P) \cdots (r_n \rightarrow \Gamma P)
\]

\[
\Delta_{p,i} = (r_1 \rightarrow \Gamma P) \cdots (r_{i-1} \rightarrow \Gamma P) \cdot r_i \rightarrow \Gamma P \cdot \cdots \cdot (r_{n} \rightarrow \Gamma P)
\]
A term $\Delta_{P,i}$ differs from $\Pi_P$ because it misses the $i^{th}$ term. From commutativity and associativity of $\parallel$ it follows that that $\Pi_P = (r_i \rightarrow \Gamma_P)\parallel \Delta_{P,i}$. Note that for simple programs $P$ the most general schedule $\Gamma_P$ can be written $\Pi_P$.

**Definition** $s^0 = \text{skip}, s^n = s\parallel (s^{n-1})$

The following definition introduces the concept of $\Gamma_P$-derived configuration. This notion comprises a syntactical and a semantical criterion. All configurations that a $\Gamma_P$-derived configuration may evolve into, according to our semantics, are also $\Gamma_P$-derived. If a schedule (rather than a whole configuration) satisfies the first property, then we say that this is a $\Gamma_P$-derived schedule. These will be useful in proving properties about the most general schedule $\Gamma_P$ which forms, by definition, a $\Gamma_P$-derived configuration with any multiset $M$.

**Definition** Let $P = r_1 + \ldots + r_n$. A configuration $\langle s, M \rangle$ is $\Gamma_P$-derived if

1. Schedule $s$ is of the form $(r_1 \rightarrow \Gamma_P)^{a_1}\parallel \ldots \parallel (r_n \rightarrow \Gamma_P)^{a_n}\parallel \Gamma_P^k$
2. $(k = 0) \Rightarrow (\forall i : 1 \leq i \leq n :: a_i = 0 \Rightarrow \langle r_i, M \rangle \sqrt{\big\rangle}$.

Next we prove that the property $\Gamma_P$-derived, is invariant with respect to our set of semantic rules; i.e. when a configuration is $\Gamma_P$-derived then any subsequent configuration that can be derived using the semantic rules is also $\Gamma_P$-derived.

**Lemma 1** Let $\langle s, M \rangle$ be a $\Gamma_P$-derived configuration. If $\langle s, M \rangle \xrightarrow{\lambda} \langle s', M' \rangle$ then $\langle s', M' \rangle$ is a $\Gamma_P$-derived configuration.

**Proof**

$\lambda \neq \varepsilon$:

Assume $\lambda \neq \varepsilon$. The only means to introduce a $\sigma$-transition is by (S1). In a $\Gamma_P$-derived schedule only terms of the form $(r \rightarrow \Gamma_P)$ may participate in a transition. Because $s$ is a schedule of a $\Gamma_P$-derived configuration this implies that there must be at least one term $(r \rightarrow \Gamma_P)$ that executed successfully; i.e. $\langle r_i \rightarrow \Gamma_P, M \rangle \xrightarrow{g_i} \langle \Gamma_P, M_i \rangle$. Let $f_i$ be the number of times that a term $(r_i \rightarrow \Gamma_P)$ is executed, but failed; let $g_i$ be the number of $(r_i \rightarrow \Gamma_P)$ terms that is executed successfully; let $G = \sum_{j=1}^{n} g_j$. The schedule resulting from $s$ after such a transition is of the form $(r_1 \rightarrow \Gamma_P)^{a_1-f_i-g_i}\parallel \ldots \parallel (r_n \rightarrow \Gamma_P)^{a_n-f_i-g_i}\parallel \Gamma_P^k$ where $k' \geq G \geq 1$; hence this schedule is in $\Gamma_P$-derived form. The resulting configuration is $\Gamma_P$-derived, because $k' > 0$.

$\lambda = \varepsilon$:

Then $M' = M$. The only means to introduce a $\varepsilon$-transition is by (S0). In a $\Gamma_P$-derived schedule only terms of the form $(r \rightarrow \Gamma_P)$ may participate in a transition. Let $s = (r_1 \rightarrow \Gamma_P)^{a_1}\parallel \ldots \parallel (r_n \rightarrow \Gamma_P)^{a_n}\parallel \Gamma_P^k$ be the schedule before the transition and let $s' = (r_1 \rightarrow \Gamma_P)^{a_1}\parallel \ldots \parallel (r_n \rightarrow \Gamma_P)^{a_n}\parallel \Gamma_P^k$ be the schedule after the transition. If $k' > 0$ then the configuration $\langle s', M \rangle$ is $\Gamma_P$-derived. If $k' = 0$ and $a'_i = 0$, we have to show that $\langle r_i, M \rangle \sqrt{\big\rangle}$. We consider two possibilities: If already $a_i = 0$, then by the assumption that $\langle s, M \rangle$ is $\Gamma_P$-derived, we have $\langle r_i, M \rangle \sqrt{\big\rangle}$. If $a_i > 0$ then the failing execution $\langle r_i \rightarrow \Gamma_P, M \rangle \xrightarrow{\varepsilon} \langle \text{skip}, M \rangle$ in (the derivation of) the last transition resulted in $a'_i = 0$. In this case we have by (S0) that $\langle r_i, M \rangle \sqrt{\big\rangle}$. 

$\square$
The next Lemma proves that any transition (from a given multiset) made by a simple chaotic program can also be made by the most general schedule (from the same multiset).

**Lemma 2**  Let \( P = r_1 + \ldots + r_n, n \geq 1 \)
\[
\langle P, M \rangle \xrightarrow{\sigma} \langle P, M' \rangle \quad \Rightarrow \quad \langle \Gamma_P, M \rangle \xrightarrow{\sigma} \langle s, M' \rangle
\]

**Proof**
by induction on the length of the proof of \( \langle P, M \rangle \xrightarrow{\sigma} \langle P, M' \rangle \).

The last step of the proof may have been made using either rule (C2) or rule (C4).

*Case (C2)*:
If the last deduction was made by (C2) then, by (C1) and (C3), \( P \) contains a single rule \( r_i \) such that \( \langle r_i, M \rangle \xrightarrow{\sigma_1} \langle r_i, M' \rangle \). By (S1) we infer \( \langle r_i \rightarrow \Gamma_P, M \rangle \xrightarrow{\sigma} \langle \Gamma_P, M' \rangle \).

Because \( (r_i \rightarrow \Gamma_P) = \Pi_P \) we get from (S3) that \( \langle \Pi_P, M \rangle \xrightarrow{\sigma} \langle \Gamma_P \| \Delta_{P, i}, M' \rangle \).

By (S6) and \( \Pi_P = \Gamma_P \) we derive the transition \( \langle \Gamma_P, M \rangle \xrightarrow{\sigma} \langle \Gamma_P \| \Delta_{P, i}, M' \rangle \).

*Case (C4)*:
The premisses of the last deduction step are
\[
\langle P, M \rangle \xrightarrow{\sigma_1} \langle P, M_1 \rangle \quad (1)
\]
\[
\langle P, M \rangle \xrightarrow{\sigma_2} \langle P, M_2 \rangle \quad (2)
\]
where \( \sigma = \sigma_1 \oplus \sigma_2 \) and \( M \models \sigma \| \sigma_2 \). We apply the induction hypothesis to (2) to get
\[
\langle \Gamma_P, M \rangle \xrightarrow{\sigma_2} \langle s', M_2 \rangle \quad (3)
\]
for some schedule \( s' \). By a derivation similar to the case (C2) we deduce from (1) that
\[
\langle \Pi_P, M \rangle \xrightarrow{\sigma_1} \langle \Gamma_P \| \Delta_{P, i}, M_1 \rangle \quad (4)
\]

Because \( M \models \sigma_1 \| \sigma_2 \) we may use (S5) to deduce from premisses (3) and (4) that
\[
\langle \Pi_P \| \Gamma_P, M \rangle \xrightarrow{\sigma} \langle \Gamma_P \| \Delta_{P, i} \| s', M[\sigma] \rangle \quad (5)
\]

Because \( \Pi_P = \Gamma_P \) we conclude using (S7) that \( \langle \Gamma_P, M \rangle \xrightarrow{\sigma} \langle \Gamma_P \| \Delta_{P, i} \| s', M' \rangle \) for some \( s' \).

\(\square\)

The interpretation of Theorem 1 is that whenever a chaotic program makes a sequence of transitions, then the most general schedule of that program can make a similar sequence of transitions. The multiset transformations of the chaotic behaviour occur in the same temporal order in the behaviour of the most general schedule, but in between the successful transformations \( \varepsilon \)-transitions may happen.
Theorem 1  For all multiset transformer programs \( P \), for all multisets \( M \) and sequences \( \overline{\sigma} \) of multiset substitutions
\[
\langle P, M \rangle \sim^* \langle P', M' \rangle \rightarrow \quad (\exists \overline{\lambda} : \overline{\lambda} = \overline{\sigma} : \langle \Gamma_P, M \rangle \xrightarrow{\overline{\lambda}} \langle \text{skip}, M' \rangle)
\]

The proof of Theorem 1 is constructed by double induction. The ‘outermost’ induction is on the structure of program \( P \). The base case of this needs another induction, the ‘inner’ one, which is on the length of sequence \( \overline{\sigma} \).

Proof

\( P = r_1 + \ldots + r_n \), where \( n > 0 \):

\( \overline{\sigma} = \langle \rangle \):

From \( \overline{\sigma} = \langle \rangle \), \( \langle P', M' \rangle \rightarrow \) and reflexivity of \( \sim^* \) we deduce \( \langle P, M \rangle \rightarrow \). From (C5) and (C0) follows \( \langle r_i, M \rangle \rightarrow \) for all \( 1 \leq i \leq n \). By (S0) we then also have \( \langle r_i \rightarrow \Gamma_P, M \rangle \xrightarrow{\varepsilon} \langle \text{skip}, M \rangle \) for all \( 1 \leq i \leq n \). Using (S5) we infer \( \langle r_1 \rightarrow \Gamma_P \| \cdots \| r_n \rightarrow \Gamma_P \| M \rangle \xrightarrow{\varepsilon} \langle \text{skip} \| \cdots \| \text{skip} \| M \rangle \). By structural congruence the right hand configuration of this transition equals \( \langle \text{skip}, M \rangle \). From (S6) follows \( (\langle r_1 \rightarrow \Gamma_P \| \cdots \| r_n \rightarrow \Gamma_P \| M \rangle) \xrightarrow{\varepsilon} \langle \text{skip}, M \rangle \). The transition sequence that has been constructed, consists of a single \( \varepsilon \)-labelled transition. This transition sequence meets the conditions of the theorem because \( \langle \varepsilon \rangle = \langle \rangle = \overline{\sigma} \).

\( \overline{\sigma} = \langle \sigma_1, \ldots, \sigma_l \rangle \), where \( l \geq 1 \):

A transition \( \langle P, M \rangle \sim^* \langle P', M' \rangle \) can be rewritten using transitivity of \( \sim^* \) as
\[
\langle P, M \rangle \overset{\overline{\sigma_1}}{\sim} \langle P'', M'' \rangle \sim^* \langle P', M' \rangle
\]
where \( \overline{\sigma_1} = \langle \sigma_2, \ldots, \sigma_l \rangle \). Because \( P \) is simple, it follows from the semantics for chaotic programs that \( P = P' = P'' \), hence we may write \( \langle P, M \rangle \overset{\overline{\sigma_1}}{\sim} \langle P, M'' \rangle \sim^* \langle P, M' \rangle \). The induction hypothesis gives a path that corresponds to the last sequence of transitions \( \langle P, M'' \rangle \sim^* \langle P, M' \rangle \rightarrow \) :
\[
\langle \Gamma_P, M'' \rangle \xrightarrow{\overline{\lambda}} \langle \text{skip}, M' \rangle
\]
for some \( \overline{\lambda} \) such that \( \overline{\lambda} = \overline{\sigma}_1 \). From \( \langle P, M \rangle \overset{\overline{\sigma}_1}{\sim} \langle P, M'' \rangle \) and Lemma 2 we infer
\[
\langle \Gamma_P, M \rangle \overset{\overline{\sigma}_1}{\sim} \langle s, M'' \rangle
\]
for some schedule \( s \). From Lemma 1 follows that \( \langle s, M'' \rangle \) is a \( \Gamma_P \)-derived configuration. Hence \( s \) is of the form \( (r_1 \rightarrow \Gamma_P)^{\sigma_1} \| \cdots \| (r_n \rightarrow \Gamma_P)^{\sigma_1} \| \Gamma_P^{k} \). Because \( \sigma_1 \neq \varepsilon \), we know that \( s \) contains at least one \( \Gamma_P \) term; i.e. \( k > 0 \). From (S3) and (6) follows
\[
\langle (r_1 \rightarrow \Gamma_P)^{\sigma_1} \| \cdots \| (r_n \rightarrow \Gamma_P)^{\sigma_1} \| \Gamma_P^{k-1} \| \Gamma_P, M'' \rangle
\]
\[
\langle (r_1 \rightarrow \Gamma_P)^{\sigma_1} \| \cdots \| (r_n \rightarrow \Gamma_P)^{\sigma_1} \| \Gamma_P^{k-1} \| \text{skip}, M' \rangle
\]
for all \( 1 \leq i \leq n \). From this it is easily seen that
\[
\langle (r_1 \rightarrow \Gamma_P)^{\sigma_1} \| \cdots \| (r_n \rightarrow \Gamma_P)^{\sigma_1} \| \Gamma_P^{k-1}, M' \rangle \xrightarrow{\varepsilon} \langle \text{skip}, M' \rangle
\]
Using transitivity of $\rightarrow^*$ we combine (7), (8) and (9) to conclude
$\langle \Gamma, M \rangle \xrightarrow{\lambda} \langle \text{skip}, M' \rangle$ where $\lambda = \langle \sigma_1 \rangle \lambda' \langle \varepsilon \rangle$. Note that $\lambda = \langle \sigma_1, \ldots, \sigma_t \rangle$ as required.

$P = P_2 \circ P_1$:

If $\langle P, M \rangle \rightarrow^* \langle P', M' \rangle$ we get using transitivity of $\sim^*$ that there exist $\overline{\sigma_1}$ and $\overline{\sigma_2}$
such that $\langle P_1, M \rangle \rightarrow^* \langle P_1, M'' \rangle$ and $\langle P_2, M'' \rangle \rightarrow^* \langle P_2, M' \rangle$ and $\overline{\sigma} = \overline{\sigma_1} \overline{\sigma_2}$. From
the induction hypothesis now follows $\langle \Gamma_{P_1}, M \rangle \rightarrow^* \langle \text{skip}, M'' \rangle$ such that $\overline{\lambda_1} = \overline{\sigma_1}$ and
$\langle \Gamma_{P_2}, M'' \rangle \rightarrow^* \langle \text{skip}, M' \rangle$ such that $\overline{\lambda_2} = \overline{\sigma_2}$. Let $\overline{\lambda} = \overline{\lambda_1} \overline{\lambda_2}$, hence $\lambda = \overline{\sigma_1} \overline{\sigma_2} = \overline{\sigma}$.

By (S2) we get $\langle \Gamma_{P_1}, \Gamma_{P_2}, M \rangle \rightarrow^* \langle \text{skip}, M' \rangle$.

We continue with two lemmas that are used to prove the conceptual converse of the previous theorem. Similar to the previous proof, we first show that any possible transition of a $\Gamma_0$-
derived schedule can also be made by the corresponding chaotic program.

**Lemma 3** Let $P = r_1 + \ldots + r_n$, where $n \geq 1$, let $M$ be a multiset, and let $s$ be a $\Gamma_0$-derived schedule.

$\langle s, M \rangle \xrightarrow{\sigma} \langle s', M' \rangle \Rightarrow \langle P, M \rangle \xrightarrow{\sigma} \langle P, M' \rangle$

**Proof**

Because $s$ is a $\Gamma_0$-derived schedule it is of the form $((r_1 \to \Gamma_0)^{a_1} \ldots (r_n \to \Gamma_0)^{a_n}) \| \Gamma_0^k$. The only possibility to introduce a $\sigma$-transition is by using (S1). Because $s$ is a $\Gamma_0$-derived schedule this implies that there must be at least one term $(r_i \to \Gamma_0)$ that executed successfully; i.e. the derivation of the $\sigma$-transition consists of transitions $\langle r_i \to \Gamma_0, M \rangle \xrightarrow{\sigma_i} \langle \Gamma_0, M_j \rangle$ with
$1 \leq j \leq m$ and $1 \leq i_j \leq n$. These transitions have as respective antecedents (from (S1))
$\langle r_i, M \rangle \xrightarrow{\sigma_i} \langle r_i, M_j \rangle$ for $1 \leq j \leq m$ and $1 \leq i_j \leq n$. The remainder of the proof is by induction
on $m$ (the number of transitions).

If $m = 1$ we have $\langle r_i \to \Gamma_0, M \rangle \xrightarrow{\sigma_i} \langle \Gamma_0, M' \rangle$ for some $i$ from $1 \leq i \leq n$. From rules (S1), (C2),
and possibly (C3), follows $\langle P, M \rangle \xrightarrow{\sigma} \langle P, M' \rangle$.

Next, we consider the case $m \geq 2$. By associativcy of $\|$ we may assume that $s = (r_i \to \Gamma_0) \| s'$. Note that $s'$ is a $\Gamma_0$-derived schedule. According to (S5) we have

$\langle r_{i_1} \to \Gamma_0, M \rangle \xrightarrow{\sigma} \langle \Gamma_0, M_1 \rangle$ (10)

and

$\langle s', M \rangle \xrightarrow{\sigma'} \langle s'', M'' \rangle$ (11)

where $\sigma' = \sigma_2 \| \ldots \| \sigma_m$, hence $\sigma = \sigma_1 \| \sigma'$, and $M \models \sigma_1 \mathord{\triangleright} \sigma'$. From (10) and (S1) we deduce

$\langle r_{i_1}, M \rangle \xrightarrow{\sigma_1} \langle r_{i_1}, M_1 \rangle$ (12)
From (12) and (C3) we infer
\[ \langle P, M \rangle \overset{\sigma_1}{\rightarrow} \langle P, M_1 \rangle \] (13)

By the induction hypothesis we get from (11) that
\[ \langle P, M \rangle \overset{\sigma}{\rightarrow} \langle P, M'' \rangle \] (14)

Because \( M \models \sigma_1 \& \sigma' \) we conclude by (C4) from (13) and (14) that
\[ \langle P, M \rangle \overset{\sigma}{\rightarrow} \langle P, M' \rangle \] (15)

□

Lemma 4 generalizes Lemma 3 in that it says that arbitrary sequences of transition made by a \( \Gamma_P \)-derived schedule can be mimicked by the corresponding chaotic program.

**Lemma 4** Let \( P = r_1 + \ldots + r_n \) where \( n \geq 1 \) and let \( \langle s, M \rangle \) be a \( \Gamma_P \)-derived configuration.

\[ \lambda \overset{s}{\rightarrow} \langle \text{skip}, M' \rangle \Rightarrow \langle P, M \rangle \overset{\lambda}{\rightarrow} \langle P', M' \rangle \]

**Proof**

by induction on the length of \( \lambda \)

\[ \lambda = (\_): \]

From reflexivity of \( \overset{s}{\rightarrow} \) follows that \( s = \text{skip} \) and \( M' = M \). By reflexivity of \( \overset{s}{\rightarrow} \) we have \( \langle P, M \rangle \overset{()}{\rightarrow} \langle P, M \rangle \). We still have to show \( \langle P, M \rangle \overset{\lambda}{\rightarrow} \).

By assumption \( \langle \text{skip}, M \rangle \) is \( \Gamma_P \)-derived. Then by the definition of \( \Gamma_P \)-derived, we have \( \forall i : 1 \leq i \leq n : \langle r_i, M \rangle \). By (C5) we then have \( \langle P, M \rangle \overset{\lambda}{\rightarrow} \).

\[ \lambda = (\lambda_1, \ldots, \lambda_n), n > 0 : \]

By transitivity of \( \overset{s}{\rightarrow} \) we can split the transition into

\[ \langle s, M \rangle \overset{\lambda_1}{\rightarrow} \langle s', M'' \rangle \] (16)

and

\[ \langle s', M'' \rangle \overset{\lambda'}{\rightarrow} \langle \text{skip}, M' \rangle \] (17)

where \( \lambda' = (\lambda_2, \ldots, \lambda_n) \). From Lemma 1 we know that \( \langle s', M'' \rangle \) is \( \Gamma_P \)-derived, hence applying the induction hypothesis to (17) gives \( \langle P, M'' \rangle \overset{\lambda'}{\rightarrow} \langle P', M' \rangle \).

For (16) we consider the cases \( \lambda_1 = \varepsilon \) and \( \lambda_1 \neq \varepsilon \):

*Case \( \lambda_1 = \varepsilon \):

Then \( M'' = M \) and we get, for any \( P \), by reflexivity of \( \overset{s}{\rightarrow} \) that \( \langle P, M \rangle \overset{()}{\rightarrow} \langle P, M \rangle \).

Transitivity of \( \overset{s}{\rightarrow} \) gives \( \langle P, M \rangle \overset{\lambda}{\rightarrow} \langle P, M' \rangle \).
Case $\lambda_1 \neq \varepsilon$:
By Lemma 3 we get $\lambda_3 \not\sim (P, M')$. Again, transitivity of $\sim^*$ gives $\lambda_1 \not\sim (P, M')$.

\[ \square \]

The previous lemmas take care of the case for simple chaotic programs. In Theorem 2 we finish things up by considering arbitrary programs.

**Theorem 2** Let $P$ be a multiset transformer program and $M$ a multiset
\[ (\Gamma_P, M) \rightarrow^* (\text{skip}, M') \Rightarrow (P, M) \sim^* (P', M') \]

**Proof**
by induction on the structure of the program.

Case $P = r_1 + \ldots + r_n$:
This follows immediately from Lemma 4 because $\Gamma_P$ is $\Gamma_P$-derived.

Case $P = P_2 \circ P_1$:
From $\Gamma_P, (\Gamma_P_2, M') \rightarrow^* (\text{skip}, M')$ follows by (S2) that there exists $\lambda_1$ and $\lambda_2$ such that $\lambda = \lambda_1 \cdot \lambda_2$ and $\Gamma_P, (\text{skip}, M') \rightarrow^* (\text{skip}, M')$. From the induction hypothesis now follows $\lambda_1 \not\sim^* (P', M'')$ and $\lambda_2 \not\sim^* (P', M'')$. By (C6), (C7) and transitivity of $\sim^*$ we arrive at $\not\sim^* (P', M')$.

\[ \square \]

6 **Concluding Remarks and Future Research**

We have presented a formal language that can be used to control the behaviour of multiset transformer programs. By nature these programs exhibit highly nondeterministic behaviour:

- in the selection of rewrite rules, and
- in the selection of elements from the multiset.

Though a powerful abstraction mechanism, this nondeterminism poses serious problems in the implementation of multiset transformer programs. By explicitly scheduling individual rules for execution, part of the nondeterminism can be effectively controlled. An important property of the language that we have proposed in this paper is that scheduling information is specified separately from the program text. Thus the logic of a program is treated separately from control issues.

For each multiset transformer program we have defined a generic schedule that mimics the chaotic behaviour of the program. This schedule can be used as a starting point in trying to improve on the chaotic behaviour. Currently research continues with the definition of a notion of refinement for schedules. This should be a notion that compares only the behaviour of schedules (assuming that correctness of the program is unaffected).
Acknowledgments

I would like to thank Edwin de Jong for many discussions and detailed comments on earlier drafts of this report. Part of the research presented here was done during a stay at the Theory and Formal Methods section of Imperial College in London. I am grateful to Chris Hankin of the ESPRIT COORDINATION-project for making this possible and for stimulating discussions with Mark Reynolds, Lindsey Errington and other people of the COORDINATION-project.

References


