Towards a Compositional Method for Coordinating Gamma Programs *

Michel Chaudron

May 24, 1996

Abstract

With the growing complexity of software, incurred by the widespread acceptance of parallel and distributed computer systems and networks, program design would benefit from clearly separating the correctness issues (the *computation*) from efficiency issues (the *coordination*). Gamma has shown to be a powerful and expressive programming model that allows the basic computations of a program to be expressed with a minimum of control. This enables the programmer to defer efficiency related decisions until a second stage in the design process. In support of this second activity we introduce in this paper a coordination language that exploits the highly nondeterministic behaviour of Gamma to impose additional control. Furthermore, we propose a compositional notion of refinement that can be used to reason about coordination of Gamma programs. This notion induces a number of refinement laws that can be used in an algebraic style of reasoning. Some examples are presented to illustrate application of these laws.

*A short version of this report appears as [10].

Contents

1 Introduction

The first and foremost requirement of a program is that it yields the correct answer. In many cases, the second requirement is that the answer is computed as fast as possible, or at least within reasonable time. With the growing complexity of software, incurred by the widespread acceptance of parallel and distributed computer systems and networks, program design would benefit from clearly separating the correctness issues (the *computation*) from efficiency issues (the *coordination*). The significance of treating the computational part of a program separately from coordination issues is discussed in [13]. An approach based on this separation of concerns consists of the following phases:

1. First, a program is constructed that performs the required computation, but does not impose premature constraints on the behaviour of the program.

2. Secondly, the computations of the program are coordinated into an efficient execution strategy. This is often achieved by exploiting some property of a particular order of execution.

In order to realize this approach, we need a programming model that supports the separation between computation and coordination. Existing programming models usually stress only one of these aspects. For instance, functional and logical programming languages emphasize their declarative nature and the advantages it has for proving correctness. In order to improve their efficiency, functional- and logic-programs have to be tailored to capitalize on the fixed execution strategies provided by the respective execution mechanisms. With imperative languages, the programmer has complete control over the operational behaviour. Here, the control-flow is an integral part of the program, which makes it very difficult to focus on the correctness while abstracting from operational details.

Programming by multiset transformation, as exemplified by Gamma [5] has shown to be well suited to express the computations of a program without imposing premature constraints on the mode of execution. The minimal level of control yields some important advantages. First, Gamma programs are inherently parallel; in fact, it needs additional effort from the programmer to write sequential programs. Furthermore, the semantics of Gamma can be stated in a very clean and concise way, thus facilitating formal reasoning about programs. For this reason, Gamma is often put forward as an intermediate language in the program derivation process: first a Gamma program is specified

which abstracts from operational details and which is therefore easier to prove correct. Subsequently, a specialized version of this program can be constructed by introducing additional control. A method for the derivation of Gamma programs was proposed in [4].

Though in [3] it has been demonstrated that the Gamma model can be implemented, the absence of control makes it very difficult to do this efficiently. Several efforts aimed at achieving more efficient executions of Gamma programs have been proposed.

In [14], Hankin et al. derive a number of refinement and equivalence laws by considering the input-output behaviour of Gamma programs induced by an operational semantics. The theory developed is used to design a method that imposes a "pipelining" execution order on programs. The main shortcoming of the calculus reported in [14] is that the laws described are not compositional; i.e. refining part of a program does not necessarily result in a refinement of the program as a whole. This limits the applicability of the laws. The following remedies have been put forward:

- In [19] a compositional semantics, based on transition traces, is given for some combining forms of Gamma programs. A more detailed study of the laws resulting from this semantics is given in [20].
- Another approach, presented in [11], uses a notion of equivalence based on bisimulation [17] to obtain compositional semantics.
- More general laws are obtained by focussing on a set of higher-level combining forms for Gamma, called *TROPES* [15]. The *TROPES* encapsulate typical forms of Gamma programs that have emerged from programming experience (e.g. [5]).

These approaches have in common that they try to improve the efficiency of a program by modifying the program itself. An alternative approach, that is adopted in this paper, is based on the idea of separating computation from coordination. The basic computation that is required to solve the problem at hand, is specified as a Gamma program which is proven correct. Then we exploit the highly nondeterministic behaviour of Gamma to improve the efficiency. This is done by using the coordination language presented in [9] that enables the programmer to control the otherwise chaotic execution of Gamma programs to the level of fully deterministic behaviour. The coordination component is specified separately from the Gamma program allowing more efficient versions of a pro-

gram to be constructed while leaving the computational part unaffected.

Using this approach it is essential that the coordination component does not affect the established correctness of the original Gamma program. To meet this requirement, we present in this paper a method that opens up the opportunity for coordinating Gamma programs in a provably correct way. The method that we propose uses a transformational approach: Based on the semantics of schedules and a notion of refinement we derive laws that can be used to refine schedules while preserving correctness.

The paper is organized as follows. First we successively introduce the Gamma model and its coordination language. In Section 4.2 we then propose a compositional notion of refinement that can be used in reasoning about the coordination component of a program. In Section 4.4 we derive a number of laws of refinement that support an algebraic style of reasoning. We illustrate these laws with some example applications in Section 5. We conclude the paper in Section 6 with some final remarks and directions for future research.

2 The Gamma Model

We start with a brief introduction to Gamma. For more details the reader is referred to [5] which includes a broad spectrum of example programs.

The uniform data structure in Gamma is the multiset. Multisets can be formed over arbitrary domains of values, including integers, reals, booleans and tuples. The simplest Gamma program is a conditional multiset rewrite-rule, written as $\overline{x} \rightarrow m \Leftarrow b$. Here $\overline{x}$ denotes a sequence of variables $x_1, \ldots, x_n$, m is a multiset expression, and b is a boolean expression. The free variables that occur in m and b are taken from $x_1, \ldots, x_n$.

Application of the rule $\overline{x} \rightarrow m \Leftarrow b$ to a multiset proceeds by replacing elements in the multiset satisfying the condition b by the elements that result from evaluating the multiset expression m. This step is repeated until no more elements are present that satisfy b.

For example, a Gamma program for sorting a sequence of numbers into ascending order

is given by the following rule.

$$swap \mathrel{\widehat{=}} (x,i),(y,j) \rightarrow (y,i),(x,j) \Leftarrow x > y \ \wedge \ i < j$$

The sequence is represented by a multiset consisting of value-index pairs. The program executes by exchanging ill-ordered values until there are no more pairs that satisfy this condition. At that point the resulting multiset represents a well-ordered sequence. It is important to note that the Gamma program does not specify in which order pairs of values are compared and exchanged. Disjoint pairs can be compared and exchanged in parallel, but this need not necessarily be the case. The Gamma program can be seen as the specification of a wide spectrum of more deterministic sorting strategies.

More complex Gamma programs can be built using two basic combinators. Individual rules can be composed into so-called *simple* programs [14] using the parallel combinator, denoted "+". The constituent rules in parallel composition are executed in any order (possibly in parallel) until none of the rules can be successfully applied. Simple programs can in turn be composed using the sequential combinator, denoted " $\circ$ ". If P_1 and P_2 are simple programs, then $P_1 \circ P_2$ first executes P_2 until its rules can no longer be applied, after which P_1 is executed on the resulting multiset.

The abstract syntax of Gamma programs can be specified as follows. We use r, R and P to range over the syntactic categories of multiset rewrite-rules, simple programs, and programs respectively.

$$\begin{array}{lcll} r & ::= & \overline{x} \rightarrow m \Leftarrow b & \\ R & ::= & r & \mid \quad R + R \\ P & ::= & R & \mid \quad P \circ P \end{array}$$

A configuration of a program P and a multiset M is written $\langle P, M\rangle$. To define the operational semantics of Gamma we use a labelled multi-step transition relation between configurations. A transition is written as $\langle P, M\rangle \stackrel{\sigma}{\rightsquigarrow} \langle P', M'\rangle$ where the label σ stands for the multiset substitution that transforms M into M'. A terminal configuration is written $\langle P, M\rangle\surd$.

The semantics of Gamma is collected in Figure 1. The multi-step transition relation is defined in terms of a single-step transition relation, that we distinguish by the subscript "1", i.e. $\langle P, M\rangle \stackrel{\sigma}{\longrightarrow}_1 \langle P, M'\rangle$. The various notations that we use in defining the semantics

are best explained by considering the semantic rule for $r = \overline{x} \rightarrow m \Leftarrow b$:

$$\text{if } \overline{v} \subseteq M : b[\overline{x} := \overline{v}] \text{ then } \langle r, M\rangle \xrightarrow{\sigma}_1 \langle r, M[\sigma]\rangle \text{ where } \sigma = m[\overline{x} := \overline{v}]/\overline{v}$$

We write $b[\overline{x} := \overline{v}]$ to denote the boolean expression that results from replacing each free occurrence of x_i by v_i. By $M[\sigma]$ we denote the multiset that results from applying the substitution σ to M. More formally, let $M' = m[\overline{x} := \overline{v}]$, then $M[M'/\overline{v}] = (M \ominus \overline{v}) \oplus M'$, where $\oplus$ and $\ominus$ denote multiset addition and subtraction respectively. Note that for ease of notation we confuse the sequence $\overline{v}$ with the multiset consisting of the same elements as $\overline{v}$.

When multiple transitions transform disjoint parts of the multiset, then these transitions do not interfere with each other, hence they can also happen in parallel. This observation directly leads to the multi-step transition semantics of Gamma as defined in Figure 1. Formally the notion of non-interference can be defined in terms of the labels of the constituent transitions as follows. (In [7] a more liberal definition is given that reflects the possibility of concurrent reading of data.)

Definition 2.1 *Given a multiset M and two multiset substitutions $\sigma_1 = M_1/N_1$ and $\sigma_2 = M_2/N_2$, we say that σ_1 and σ_2 are* independent *in M if $N_1 \oplus N_2 \subseteq M$. We write $M \models \sigma_1 \bowtie \sigma_2$ if σ_1 and σ_2 are* independent *in M.*

The label assigned to a multi-step transition is a combination of the labels of the constituent transitions.

Definition 2.2 *Given two multiset substitutions $\sigma_1 = M_1/N_1$ and $\sigma_2 = M_2/N_2$, the* composition *of σ_1 and σ_2 is defined as $\sigma_1 \oplus \sigma_2 = (M_1 \oplus M_2)/(N_1 \oplus N_2)$.*

The semantics of Gamma as presented here differs from the one in [14]. The latter uses a single step transition relation – suggesting an interleaved semantics. As mentioned before, our coordination language restricts the otherwise nondeterministic behaviour of Gamma programs, hence it cannot introduce new behaviour. Consequently, the semantics we choose for programs, limits the behaviours we can obtain using a coordination language. Because we want to distinguish between parallel and sequential execution at the coordination level, we need this distinction to be present in the semantics of Gamma.

$$(C0)\quad \langle \overline{x} \to m \Leftarrow b, M\rangle \xrightarrow{\sigma}_1 \langle \overline{x} \to m \Leftarrow b, M[\sigma]\rangle \quad \textit{if } \overline{v} \subseteq M \ \wedge\ b[\overline{x} := \overline{v}] \quad \textit{where } \sigma = m[\overline{x} := \overline{v}]/\overline{v}$$

$$(C1)\quad \langle \overline{x} \to m \Leftarrow b, M\rangle\surd \quad \textit{if } \neg(\exists \overline{v} \subseteq M \ :\ b[\overline{x} := \overline{v}])$$

$$(C2)\quad \frac{\langle R_1, M\rangle \xrightarrow{\sigma}_1 \langle R_1, M'\rangle}{\begin{array}{c}\langle R_1 + R_2, M\rangle \xrightarrow{\sigma}_1 \langle R_1 + R_2, M'\rangle \\ \langle R_2 + R_1, M\rangle \xrightarrow{\sigma}_1 \langle R_2 + R_1, M'\rangle\end{array}}$$

$$(C3)\quad \frac{\langle R, M\rangle \xrightarrow{\sigma}_1 \langle R, M'\rangle}{\langle R, M\rangle \overset{\sigma}{\rightsquigarrow} \langle R, M'\rangle}$$

$$(C4)\quad \frac{\begin{array}{c}\langle R, M\rangle \overset{\sigma_1}{\rightsquigarrow} \langle R, M_1\rangle \\ \langle R, M\rangle \overset{\sigma_2}{\rightsquigarrow} \langle R, M_2\rangle\end{array}}{\langle R, M\rangle \overset{\sigma_1 \oplus \sigma_2}{\rightsquigarrow} \langle R, M[\sigma_1 \oplus \sigma_2]\rangle} \quad \textit{if } M \models \sigma_1 \bowtie \sigma_2$$

$$(C5)\quad \frac{\begin{array}{c}\langle R_1, M\rangle\surd \\ \langle R_2, M\rangle\surd\end{array}}{\langle R_1 + R_2, M\rangle\surd}$$

$$(C6)\quad \frac{\begin{array}{c}\langle P_1, M\rangle\surd \\ \langle P_2, M\rangle\surd\end{array}}{\langle P_1 \circ P_2, M\rangle\surd}$$

$$(C7)\quad \frac{\begin{array}{c}\langle P_1, M\rangle\surd \\ \langle P_2, M\rangle \overset{\sigma}{\rightsquigarrow} \langle P_2', M'\rangle\end{array}}{\langle P_2 \circ P_1, M\rangle \overset{\sigma}{\rightsquigarrow} \langle P_2', M'\rangle}$$

$$(C8)\quad \frac{\langle P_1, M\rangle \overset{\sigma}{\rightsquigarrow} \langle P_1', M'\rangle}{\langle P_2 \circ P_1, M\rangle \overset{\sigma}{\rightsquigarrow} \langle P_2 \circ P_1', M'\rangle}$$

Figure 1: Structured Operational Semantics of Gamma

3 Coordination of Gamma Programs

Gamma is an expressive and powerful programming model that allows the basic computations of a program to be expressed in a concise way and with a minimum of control. This enables the programmer to defer efficiency related decisions until a second stage in the design process. In support of this second activity we next introduce a coordination language that exploits the highly nondeterministic behaviour of Gamma to impose additional control with the objective to improve efficiency.

We refer to the programs that are written in the coordination language as *schedules* to emphasize the fact that they are not really programs but rather execution plans or harnesses for an existing program. A schedule is an expression representing an imperative statement over the rules from a Gamma program. The simplest schedule for a program P (next to skip which denotes the empty schedule) has the form $r \rightarrow s[t]$, where r is a rule from P and s and t denote arbitrary schedules. This schedule is executed by first attempting to execute the rule r, if this succeeds, then execution continues with the schedule s. If execution of r fails, then execution continues with t. As a notational convention, we write $r \rightarrow s[\mathsf{skip}]$ as $r \rightarrow s$ and $r \rightarrow \mathsf{skip}$ as r.

The coordination language provides a number of basic combinators that can be used to build more complex schedules. The complete set of combinators that is included in the kernel language is defined by the following abstract syntax for schedules. We use $\mathbb{S}$ to denote the set of schedule expressions, ranged over by s, t, u. The set $\mathcal{S}$ denotes the set of schedule identifiers, ranged over by S, T. A schedule without free schedule variables is called a *ground schedule*. The set of ground schedules is denoted $\mathbb{S}_{ground}$. The substitution of schedule(s) $\overline{t}$ for variables $\overline{X}$ in a schedule s is written $s\{\overline{t}/\overline{X}\}$. A sequence of values is denoted by $\overline{v}$. Variables that range over these values are denoted by $\overline{x}, \overline{y}$. This type of variables are also called *control variables*.

$$s \quad ::= \quad \mathsf{skip} \;\mid\; r \rightarrow s[s] \;\mid\; s; s \;\mid\; s \,\|\, s \;\mid\; c \triangleright s[s] \;\mid\; !s \;\mid\; S(\overline{v})$$

Schedules can be composed sequentially, using the combinator ";" and be composed in parallel using " $\|$ ". The execution of a parallel composition $s \,\|\, t$ proceeds by a step performed by either s or t, or by a parallel step in which both s and t participate. For notational convenience, we write s^k, for $k \geq 0$, to denote k copies of schedule s composed

in parallel. Furthermore, we use $\Pi_{i=1}^{n} s_i$ to denote $s_1 \parallel s_2 \parallel \ldots \parallel s_n$.

Execution of a Gamma program is such that the number of rules that may be executed varies dynamically with the number of available elements in the multiset. In order to describe this dynamic behaviour using schedules, the replication operator "!" is included. The schedule $!s$ denotes an arbitrary number of copies of s executing in parallel.

The occurrence of a schedule identifier $S(\overline{v})$ is accompanied by a corresponding schedule definition of the form $S(\overline{x}) \mathrel{\widehat{=}} s$. The free variables in s are taken from the sequence $\overline{x}$. Schedule definitions are included for structuring purposes, as well as a means to express recursive schedules. The use of recursion is typically accompanied by the use of a conditional schedule $c \triangleright s[t]$. Here, c represents a boolean expression; if c evaluates to *true*, then schedule s is executed, otherwise execution continues with t. Analogously to the rule-conditional, $c \triangleright s[\mathsf{skip}]$ is written as $c \triangleright s$.

Nondeterminism in Gamma arises at two levels:

1. at the selection of a rewrite-rule,
2. in selecting elements from the multiset.

The coordination language as introduced so far is only capable of resolving the first type of nondeterminism. The second type is resolved by *strengthening* the condition of a rewrite-rule. Consider a rule $r = \overline{x} \rightarrow m \Leftarrow b$. Rather than scheduling r directly, we can schedule a rule $r' = x \rightarrow m \Leftarrow b'$, such that $b' \Rightarrow b$. Since b' is a strengthening of b, the rule r' exhibits restricted behaviour compared to r.

To illustrate, we return to the example sorting program consisting of the rule *swap*. A schedule that, for instance, exchanges neighbouring values only, will make use of a rule *swap'* which is obtained from the original rule by strengthening condition $i < j$ to $i = j - 1$ to get

$$swap' \mathrel{\widehat{=}} (x, i), (y, j) \rightarrow (y, i), (x, j) \Leftarrow x > y \ \wedge \ i = j - 1$$

To facilitate this process we shall adopt the notational convention that rule definitions are parameterized in the variables that are used to select elements from the multiset. For sorting this means that we define the rule

$$swap(i,j) \mathrel{\widehat{=}} (x,i),(y,j) \rightarrow (y,i),(x,j) \Leftarrow x > y \ \wedge i < j$$

A schedule that coordinates the sorting program such that it behaves like insertion sort, for instance, can now be specified as $InsertionSort(1)$ where

$$\begin{array}{lcl} InsertionSort(i) & \mathrel{\widehat{=}} & (i \leq n) \triangleright (Insert(i); InsertionSort(i+1)) \\ Insert(i) & \mathrel{\widehat{=}} & (i > 0) \triangleright (swap(i-1,i) \rightarrow Insert(i-1)) \end{array}$$

Here n denotes the length of the sequence. A well known parallel sorting algorithm (see e.g. [18]) is Odd-Even Transposition Sort. Coordination of the sorting program into a corresponding behaviour can be specified as $OddEvenSort(n)$ where

$$\begin{array}{lcl} OddEvenSort(m) & \mathrel{\widehat{=}} & (m \geq 0) \triangleright (Odd \ ; \ Even \ ; \ OddEvenSort(m-2) \\ Odd & \mathrel{\widehat{=}} & \Pi_{i=0}^{n \,\mathtt{div}\, 2-1} swap(2i+1, 2i+2) \\ Even & \mathrel{\widehat{=}} & \Pi_{i=0}^{n+1 \,\mathtt{div}\, 2-1} swap(2i, 2i+1) \end{array}$$

The operational semantics of the coordination language is defined in Figure 2 as a labelled multi-step transition relation between configurations which uses the following structural congruences

---

$$\begin{array}{llcll} (E1) & \mathsf{skip}; s & \equiv & s & \\ (E2) & s_1; (s_2; s_3) & \equiv & (s_1; s_2); s_3 & \\ (E3) & \mathsf{skip} \parallel s & \equiv & s & \\ (E4) & s_1 \parallel (s_2 \parallel s_3) & \equiv & (s_1 \parallel s_2) \parallel s_3 & \\ (E5) & s_1 \parallel s_2 & \equiv & s_2 \parallel s_1 & \\ (E6) & true \triangleright s[t] & \equiv & s & \\ (E7) & false \triangleright s[t] & \equiv & t & \\ (E8) & !\mathsf{skip} & \equiv & \mathsf{skip} & \\ (E9) & S(\overline{v}) & \equiv & \mathsf{skip} & \text{if } S(\overline{x}) \mathrel{\widehat{=}} s \text{ and } s[\overline{x} := \overline{v}] \equiv \mathsf{skip} \end{array}$$

A configuration consists of a schedule-multiset pair $\langle s, M \rangle$. The label λ of a transition is either a multiset substitution or the special symbol ε which denotes an internal transition that does not affect the multiset. Note that the semantics of schedules is defined in terms of the single-step semantics of Gamma from Figure 1.

$$(N0)\qquad \frac{\langle r, M\rangle\surd}{\langle r \to s[t], M\rangle \xrightarrow{\varepsilon} \langle t, M\rangle}$$

$$(N1)\qquad \frac{\langle r, M\rangle \xrightarrow{\sigma}_1 \langle r, M'\rangle}{\langle r \to s[t], M\rangle \xrightarrow{\sigma} \langle s, M'\rangle}$$

$$(N2)\qquad \frac{\langle s_1, M\rangle \xrightarrow{\lambda} \langle s_1', M'\rangle}{\langle s_1 \parallel s_2, M\rangle \xrightarrow{\lambda} \langle s_1' \parallel s_2, M'\rangle}$$

$$(N3)\qquad \frac{\langle s_1, M\rangle \xrightarrow{\lambda} \langle s_1', M'\rangle \quad \langle s_2, M\rangle \xrightarrow{\varepsilon} \langle s_2', M\rangle}{\langle s_1 \parallel s_2, M\rangle \xrightarrow{\lambda} \langle s_1' \parallel s_2', M'\rangle}$$

$$(N4)\qquad \frac{\langle s_1, M\rangle \xrightarrow{\sigma_1} \langle s_1', M_1\rangle \quad \langle s_2, M\rangle \xrightarrow{\sigma_2} \langle s_2', M_2\rangle}{\langle s_1 \parallel s_2, M\rangle \xrightarrow{\sigma_1 \oplus \sigma_2} \langle s_1' \parallel s_2', M[\sigma_1 \oplus \sigma_2]\rangle} \quad \text{if } M \models \sigma_1 \bowtie \sigma_2$$

$$(N5)\qquad \frac{\langle s_1, M\rangle \xrightarrow{\lambda} \langle s_1', M'\rangle}{\langle s_1; s_2, M\rangle \xrightarrow{\lambda} \langle s_1'; s_2, M'\rangle}$$

$$(N6)\qquad \frac{\langle s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle}{\langle !s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle}$$

$$(N7)\qquad \frac{\langle s \parallel !s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle}{\langle !s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle}$$

$$(N8)\qquad \frac{\langle s[\overline{x} := \overline{v}], M\rangle \xrightarrow{\lambda} \langle s', M'\rangle}{\langle S(\overline{v}), M\rangle \xrightarrow{\lambda} \langle s', M'\rangle} \quad \text{where } S(\overline{x}) \mathrel{\widehat{=}} s$$

$$(N9)\qquad \frac{s \equiv t \quad \langle s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle \quad s' \equiv t'}{\langle t, M\rangle \xrightarrow{\lambda} \langle t', M'\rangle}$$

Figure 2: Structured Operational Semantics of Schedules

4 Refinement of Schedules

Our aim is to devise a design method for (parallel) programs where computation is clearly separated from coordination. An essential aspect of such a design method is the ability to reason about coordination. In our framework coordination is achieved by resolving nondeterministic choices in Gamma. As an instrument to eliminate nondeterminism, we propose in this section a notion of refinement for schedules. The problem of finding efficient execution strategies can be broken down in smaller steps by constructing succesive refinements where every subsequent refinement gradually achieves more deterministic control.

In Figure 3 we visualize the intuitive effect of refinement on the possible behaviours of a schedule.

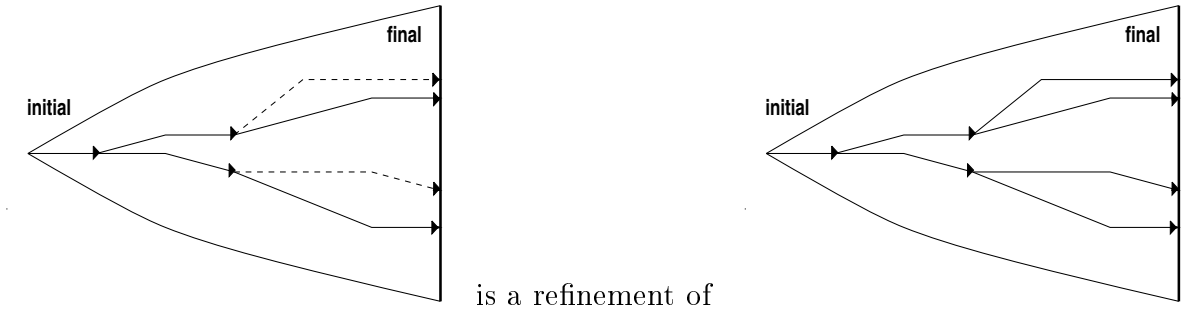

Figure 3: Refinement by Limiting Execution Space

The right hand side of Figure 3 informally depicts the execution space of a schedule. In general, schedules are nondeterministic, hence there are multiple execution paths that a schedule may follow. These execution paths are represented by the branching lines going from the single initial state (on the left) to one of the final states (denoted by the vertical line on the right). All correct schedules must end in one of the terminal states.

The execution space of a refinement of the schedule is depicted on the left hand side of Figure 3. Dotted lines indicate execution paths of the original schedule, that are no longer possible executions of the refining schedule. We will consider a schedule to be a refinement of another schedule if it allows fewer ways to be executed but maintains total correctness. The refinement suggested by Figure 3 can be seen to preserve correctness,

because it retains (at least) one execution path that reaches a final state.

Such a notion of refinement does not automatically yield efficient schedules. It provides a tool that, when used thoughfully, can eliminate the less efficient execution paths, thus retaining the more efficient ones.

Next, we set out to formally define a notion of refinement. We then derive a number of refinement laws that can be used in an algebraic style of reasoning about coordination. Applications of these laws will be illustrated in Section 5.

4.1 Prefix Simulation

Bisimulation is commonly used for models of concurrency where process and state are identified [17][1]. In Gamma and also in its coordination language, communication is implicit through the use of the shared multiset. As a result, the behaviour of (programs and) schedules depends on the state of the multiset. Therefore we use configurations $\langle s, M\rangle$ rather than just schedules in our definition of bisimulation.

Definition 4.1 *A binary relation $\mathcal{R} \subseteq \mathbb{S} \times \mathbb{S}$ is a* bisimulation *if $(s,t) \in \mathcal{R}$ implies, for all λ, for all M,*

$$\begin{array}{ll} 1. & \langle s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle \;\Rightarrow\; \exists t' : \langle t, M\rangle \xrightarrow{\lambda} \langle t', M'\rangle \;\wedge\; (s', t') \in \mathcal{R} \\ 2. & \langle t, M\rangle \xrightarrow{\lambda} \langle t', M'\rangle \;\Rightarrow\; \exists s' : \langle s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle \;\wedge\; (s', t') \in \mathcal{R} \end{array}$$

The obvious, but as it turns out naive, way of obtaining simulation from bisimulation is by breaking the symmetry. For reasons that will be explained shortly, we call this notion *prefix simulation.*

Definition 4.2 *A binary relation $\mathcal{R} \subseteq \mathbb{S} \times \mathbb{S}$ is a* prefix simulation *if $(s,t) \in \mathcal{R}$ implies, for all λ, for all M,*

$$\langle s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle \;\Rightarrow\; \exists t' : \langle t, M\rangle \xrightarrow{\lambda} \langle t', M'\rangle \;\wedge\; (s', t') \in \mathcal{R}$$

Definition 4.3 *Given schedules s and t of program P. Schedule s is a* prefix simulation *of t, written $s \leqslant_p t$, if $(s,t) \in \mathcal{R}$ for some prefix simulation $\mathcal{R}$. This may be equivalently expressed as follows:*

[1] Terminology not explained in this section is from [17]

$$\leqslant_p \ = \bigcup\{\mathcal{R} \mid \mathcal{R} \textit{ is a prefix simulation } \}$$

Following standard techniques it is possible to show that $\leqslant_p$ is the greatest fixed point of the prefix simulation relation.

The definition of prefix simulation says that if s is to be a prefix simulation of t, then for every transition that s makes, t must be able to follow suit and make a transition that performs the same computation. This works as expected for the following example (we abbreviate $r_i \to \mathsf{skip}$ by r_i).

Example 4.1 *Consider the following prefix refinement*

$$r_1; r_2; r_3 \leqslant_p r_1 \,\|\, r_2 \,\|\, r_3$$

If $r_1; r_2; r_3$ executes its first rule r_1 (resulting in $r_2; r_3$) then this can be simulated by $r_1 \,\|\, r_2 \,\|\, r_3$ which leads to a schedule $r_2 \,\|\, r_3$. Next $r_2; r_3$ may proceed by executing r_2 yielding r_3, this can be mimicked by $r_2 \,\|\, r_3$, also ending up as r_3.

We intend to use simulation to repeatedly get successively more refined versions of a schedule. In order to retain correctness, it is necessary that a refined version reaches (at least one of) the same final state(s) as the schedule that it refines. The next example illustrates that this requirement is not guaranteed by the notion of prefix simulation.

Example 4.2

$$r_1 \leqslant_p r_1 \,\|\, r_2 \,\|\, r_3$$

After both sides execute r_1, on the left hand side remains skip*, on the right hand side $r_2 \,\|\, r_3$. Now the left hand side can not make any more transitions, hence the definition of $\leqslant_p$ applies (vacuously).*

From this example we learn that, in general, we have, for any s,

$$\mathsf{skip} \leqslant_p s$$

This would justify the refinement of any schedule s by the empty schedule. (which in general does not compute the same function). Of course this does not satisfy our

intended meaning of refinement.

4.2 Refinement based on Simulation

In the previous section we found out that breaking the symmetry of standard bisimulation [17] does not meet our requirement of preserving total correctness because is allows a refining schedule to terminate prematurely. In this section we set out to remedy this by extending the definition of simulation with the condition that that s can only terminate, if t may terminate. This gives the following definition of simulation.

Definition 4.4 *A binary relation $\mathcal{R}$ on schedules is a* simulation *if $(s,t) \in \mathcal{R}$ implies, for all λ and for all M*

$$\begin{array}{lllll} 1. & \langle s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle & \Rightarrow & \exists t' : \langle t, M\rangle \xrightarrow{\lambda} \langle t', M'\rangle \ \wedge \ (s', t') \in \mathcal{R} \\ 2. & s \equiv \mathsf{skip} & \Rightarrow & t \equiv \mathsf{skip} \end{array}$$

In Figure 4 a Hasse diagram is shown which illustrates the notion of refinement implied by simulation. There is an arc from a node v to a node u if the schedule in u (represented by the possible executions) is a refinement of the schedule in v. A dotted arc from v to u denotes that the schedule in v is only a prefix-refinement of the schedule in u.
We continue, similar to Milner's treatment in [17], by showing some basic properties of simulations.

Lemma 4.1 *Assume that each $\mathcal{R}_i (i = 1, 2, \ldots)$ is a simulation. Then the following relations are all simulations:*

1. $Id_{\mathbb{S}}$
2. $\mathcal{R}_1\mathcal{R}_2$
3. $\bigcup_{i\in I} \mathcal{R}_i$

Proof

1. By reflexivity of $\Rightarrow$,
2. Suppose $(s_1, s_2) \in \mathcal{R}_1\mathcal{R}_2$, then for some t we have $(s_1, t) \in \mathcal{R}_1$ and $(t, s_2) \in \mathcal{R}_2$. Now let $\langle s_1, M\rangle \xrightarrow{\lambda} \langle s_1', M'\rangle$. Because $(s_1, t) \in \mathcal{R}_1$ we have

$$\langle t, M\rangle \xrightarrow{\lambda} \langle t', M'\rangle \text{ and } (s_1', t') \in \mathcal{R}_1$$

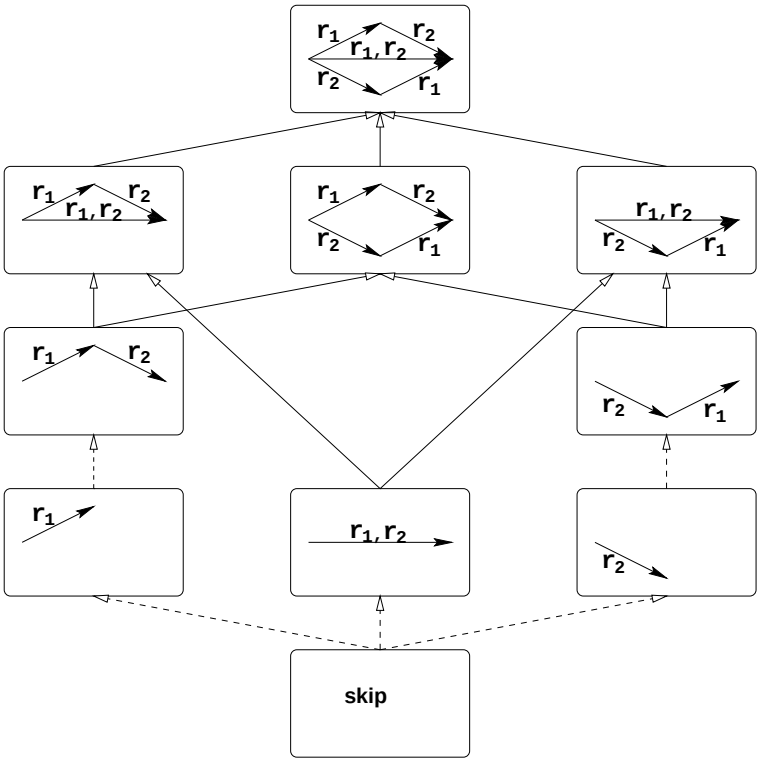

Figure 4: *Hasse diagram of the refinements of* $r_1 \parallel r_2$.

Also because $(t, s_2) \in \mathcal{R}_2$ we have, for some s_2'

$$\langle s_2, M\rangle \xrightarrow{\lambda} \langle s_2', M'\rangle \text{ and } (t', s_2') \in \mathcal{R}_2$$

From $(s_1', t') \in \mathcal{R}_1$ and $(t', s_2') \in \mathcal{R}_2$ follows $(s_1', s_2') \in \mathcal{R}_1\mathcal{R}_2$.

If $s_1 \equiv \mathsf{skip}$ then from $(s_1, t) \in \mathcal{R}_1$ we have $t \equiv \mathsf{skip}$. From $(t, s_2) \in \mathcal{R}_2$ we get $s_2 \equiv \mathsf{skip}$.

3. Let $\mathcal{R} = \bigcup_{i\in I} \mathcal{R}_i$. Suppose $(s_1, s_2) \in \mathcal{R}$, then $(s_1, s_2) \in \mathcal{R}_i$, for some $i \in I$. If $\langle s_1, M\rangle \xrightarrow{\lambda} \langle s_1', M'\rangle$ then, because $\mathcal{R}_i$ is a simulation, we have $\langle s_2, M\rangle \xrightarrow{\lambda} \langle s_2', M'\rangle$ and $(s_1', s_2') \in \mathcal{R}_i$. Because $\mathcal{R}_i \subseteq \mathcal{R}$ also $(s_1', s_2') \in \mathcal{R}$.
   The case $s_1 \equiv \mathsf{skip}$ goes analogously.

□

Definition 4.5 *Let s and t be schedules. Schedule s is a* refinement of t*, written $s \leqslant t$, if $(s,t) \in \mathcal{R}$ for some simulation $\mathcal{R}$. This may be equivalently expressed as follows:*

$$\leqslant \; = \bigcup\{\mathcal{R} \mid \mathcal{R} \text{ is a simulation } \}$$

In Definition 4.6 we use a standard method of generating an equivalence relation from an arbitrary preorder. The equivalence thus obtained is the *kernel* of the preorder induced by $\leqslant$.

Definition 4.6 *We say that schedules s and t are* bisimilar*, denoted $s \cong t$, if $(s \leqslant t) \wedge (t \leqslant s)$. This can be formulated alternatively as $\cong \; = \; \leqslant \cap \leqslant^{-1}$.*

Lemma 4.2

1. *$\leqslant$ is the largest simulation.*
2. *$\leqslant$ is a partial order.*
3. *$\cong$ is an equivalence relation.*

Proof

1. By Lemma 4.1(3) $\leqslant$ is a simulation, and by Definition 4.5 it includes any other such.
2. *Reflexivity:* By Lemma 4.1(1).

   Transitivity: If $s_1 \leqslant s_2$ and $s_2 \leqslant s_3$ then $(s_1, s_2) \in \mathcal{R}_1$ and $(s_2, s_3) \in \mathcal{R}_2$ for some simulations $\mathcal{R}_1$ and $\mathcal{R}_2$. Hence $(s_1, s_3) \in \mathcal{R}_1\mathcal{R}_2$. By 4.1(2) $\mathcal{R}_1\mathcal{R}_2$ is a simulation, by Definition 4.5 $\leqslant$ contains all simulations hence $s_1 \leqslant s_3$.

   Antisymmetry: If $s_1 \leqslant s_2$ and $s_2 \leqslant s_1$ then, by Definition 4.6, $s_1 \cong s_2$.
3. By Lemma 4.1(1 and 2) $\cong$ is reflexive and transitive. Symmetry follows from Definition 4.6.

□

We set out to show that $\leqslant$ is well defined. To this end we use some fixed-point theory (see for example [12]).

Definition 4.7 *Define a function* $\mathbb{F}$ *over binary relations on schedules, i.e.* $\mathbb{F} : \mathbb{S} \times \mathbb{S} \to \mathbb{S} \times \mathbb{S}$*, as follows. If* $\mathcal{R} \in \mathbb{S} \times \mathbb{S}$*, then* $(s,t) \in \mathbb{F}(\mathcal{R})$ *if and only if, for all* λ*, for all* M*,*

1. $\langle s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$ *then* $\langle t, M\rangle \xrightarrow{\lambda} \langle t', M'\rangle \ \wedge \ (s', t') \in \mathcal{R}$
2. $s \equiv \mathsf{skip} \ \Rightarrow \ t \equiv \mathsf{skip}$

Lemma 4.3

1. $\mathbb{F}$ *is monotonic; i.e. if* $\mathcal{R}_1 \subseteq \mathcal{R}_2$ *then* $\mathbb{F}(\mathcal{R}_1) \subseteq \mathbb{F}(\mathcal{R}_2)$
2. $\mathcal{R}$ *is a simulation* if and only if $\mathcal{R} \subseteq \mathbb{F}(\mathcal{R})$

Proof

1. Follows directly from the definition of $\mathbb{F}$.
2. This is a reformulation of the definition of simulation where "implies" is replaced by $\subseteq$.

□

Monotonicity of $\mathbb{F}$ says that it preserves the ordering $\subseteq$ on $\mathbb{P}(\mathbb{S} \times \mathbb{S})$.

We call $\mathcal{R}$ a *fixed-point* of $\mathbb{F}$ if $\mathcal{R} = \mathbb{F}(\mathcal{R})$. Similarly, we say that $\mathcal{R}$ is a *pre-fixed-point* of $\mathbb{F}$ if $\mathcal{R} \subseteq \mathbb{F}(\mathcal{R})$. So simulations are, by Lemma 4.3.2, exactly the pre-fixed-points of $\mathbb{F}$, and we wish to show that $\leqslant$, which is the largest pre-fixed-point, is a fixed-point of $\mathbb{F}$.

Lemma 4.4 $\leqslant$ *is the largest fixed-point of* $\mathbb{F}$*.*

Proof

Because $\leqslant$ is a simulation we have $\leqslant \ \subseteq \mathbb{F}(\leqslant)$.
Monotonicity of $\mathbb{F}$ implies $\mathbb{F}(\leqslant) \subseteq \mathbb{F}(\mathbb{F}(\leqslant))$. But because $\leqslant$ is the largest pre-fixed-point, it includes $\mathbb{F}(\leqslant)$, i.e. $\mathbb{F}(\leqslant) \subseteq \ \leqslant$.

Hence $\mathbb{F}(\leqslant) = \ \leqslant$. Moreover $\leqslant$ must be the largest fixed-point of $\mathbb{F}$ because it is the largest pre-fixed-point.

□

Thus $\leqslant$ is the largest relation that satisfies Definition 4.4. To establish $s \leqslant t$ it suffices to prove that a relation $\mathcal{R}$, such that $(s,t) \in \mathcal{R}$, is a simulation relation, because from

Lemma 4.4 follows, for any such simulation relation $\mathcal{R}$, that $\mathcal{R} \subseteq \leqslant$, hence $s \leqslant t$.

In Definition 4.8 we define a generalization of simulation. In combination with Lemma 4.5 this definition facilitates proving that some relation is a simulation because it allows us to make use of the fact that we have already proven other relations to be refinements.

Definition 4.8 *A binary relation $\mathcal{R} \subseteq \mathbb{S} \times \mathbb{S}$ is a* simulation up to $\leqslant$ *if $(s,t) \in \mathcal{R}$ implies, for all λ, for all M,*

1. $\langle s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle \Rightarrow \exists t' : \langle t, M\rangle \xrightarrow{\lambda} \langle t', M'\rangle \ \wedge \ (s', t') \in \ \leqslant \mathcal{R} \leqslant$
2. $s \equiv \mathsf{skip} \ \Rightarrow \ t \equiv \mathsf{skip}$

From Lemmas 4.5 and 4.6 follows that, in order to show $s \leqslant t$, it suffices to show that s and t are related by some simulation up to $\leqslant$.

Lemma 4.5 *If $\mathcal{R}$ is a simulation up to $\leqslant$, then $\leqslant \mathcal{R} \leqslant$ is a simulation.*

Proof

Let $s \leqslant \mathcal{R} \leqslant t$, hence for some s_1 and t_1, $s \leqslant s_1$, $s_1 \mathcal{R} t_1$ and $t_1 \leqslant t$. Let $\langle s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$. From $s \leqslant s_1$ follows by Lemma 4.4 that $\langle s_1, M\rangle \xrightarrow{\lambda} \langle s_1', M'\rangle$ such that $(s', s_1') \in \ \leqslant$. From $s_1 \mathcal{R} t_1$ follows $\langle t_1, M\rangle \xrightarrow{\lambda} \langle t_1', M'\rangle$ such that $(s_1', t_1') \in \ \leqslant \mathcal{R} \leqslant$. From $t_1 \leqslant t$ follows $\langle t, M\rangle \xrightarrow{\lambda} \langle t', M'\rangle$ such that $(t_1', t') \in \ \leqslant$. From $s' \leqslant s_1'$, $s_1' \leqslant \mathcal{R} \leqslant t_1'$ and $t_1' \leqslant t'$ follows $s' \leqslant \leqslant \mathcal{R} \leqslant \leqslant t'$. By transitivity of $\leqslant$ follows that $\leqslant \leqslant \ \subseteq \ \leqslant$, hence $s' \leqslant \mathcal{R} \leqslant t'$.

The case $s \equiv \mathsf{skip}$ follows directly.

□

Lemma 4.6 *If $\mathcal{R}$ is a simulation up to $\leqslant$, then $\mathcal{R} \subseteq \leqslant$.*

Proof

By Lemma 4.5 $\leqslant \mathcal{R} \leqslant$ is a simulation, hence $\leqslant \mathcal{R} \leqslant \ \subseteq \ \leqslant$. From $Id_{\mathbb{S}} \subseteq \leqslant$ follows $\mathcal{R} \subseteq \leqslant \mathcal{R} \leqslant$, hence $\mathcal{R} \subseteq \leqslant$.

□

We end this section with an example refinement. It illustrates that simulation can be used to verify that one schedule "correctly implements" another.

Example 4.3 *Let r_1 and r_2 be rules, then*

$$(r_1; r_2) \parallel (r_2; r_1) \leqslant !(r_1 \parallel r_2)$$

In order to show this consider the following relation $\mathcal{R}$:

$$\begin{array}{rcll}
\mathcal{R} & = & \{((r_1; r_2) \parallel (r_2; r_1), !(r_1 \parallel r_2))\} & (1) \\
& \cup & \{(r_2 \parallel (r_2; r_1), r_2 \parallel (r_2 \parallel r_1))\} & (2) \\
& \cup & \{((r_1; r_2) \parallel r_1, r_1 \parallel (r_1 \parallel r_2))\} & (3) \\
& \cup & \{(r_1 \parallel r_2, r_1 \parallel r_2)\} & (4) \\
& \cup & \{(r_2; r_1, r_1 \parallel r_2)\} & (5) \\
& \cup & \{(r_1; r_2, r_1 \parallel r_2)\} & (6) \\
& \cup & \{(r_1, r_1)\} & (7) \\
& \cup & \{(r_2, r_2)\} & (8) \\
& \cup & \{(\mathsf{skip}, \mathsf{skip})\} & (9)
\end{array}$$

By considering the possible transitions for each of the elements of $\mathcal{R}$, it follows that $\mathcal{R}$ is a simulation. We depict the (relevant parts of the) transition graphs of these schedules in Figure 5. Note that the numbers used to distinguish subsets of $\mathcal{R}$ correspond to the different states of the computation.

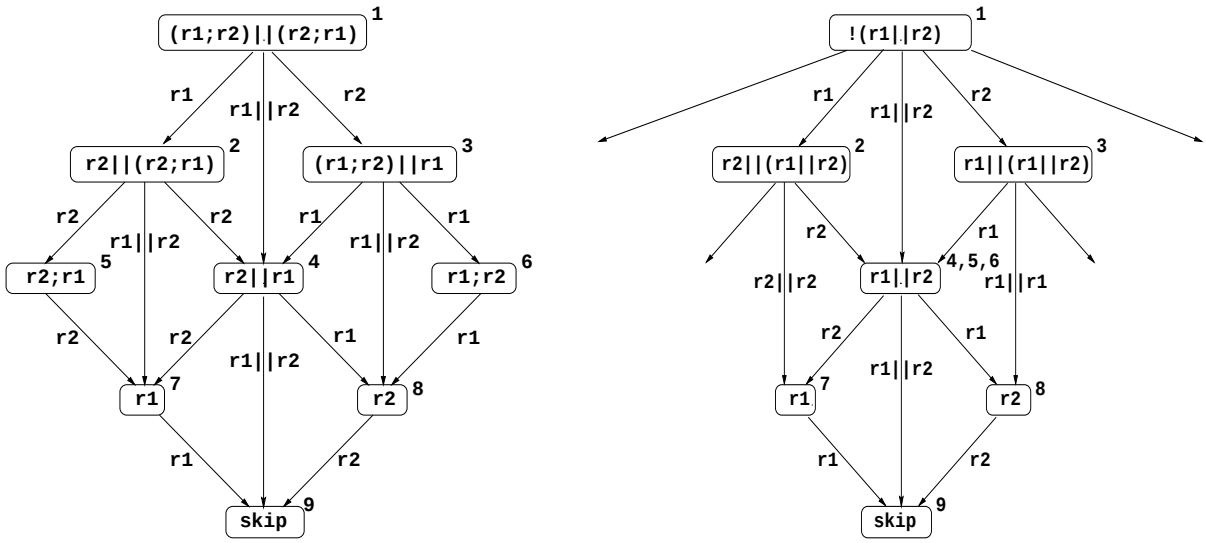

Figure 5: Transition graphs of $(\mathtt{r}_1; \mathtt{r}_2) \parallel (\mathtt{r}_2; \mathtt{r}_1)$ (left) and partially of $!(\mathtt{r}_1 \parallel \mathtt{r}_2)$ (right).

Alternatively, let $\mathcal{R}' = (1) \cup (2) \cup (3) \cup (5) \cup (6)$. *It can be shown that* $\mathcal{R}'$ *is a simulation up–to–* $\leqslant$. *From this example can be seen that considering simulation up–to–* $\leqslant$ *may reduce the complexity of refinemnt proofs.*

4.3 Precongruence Of Refinement

For practical purposes it is desirable that our refinement relation is precongruent, allowing for a modular approach in reasoning about coordination. Precongruence of $\leqslant$ follows from a set of lemma's which state that $\leqslant$ is substitutive under all combinators of the coordination language.

First, we prove Lemma 4.7 which is used in the proofs of the subsequent lemma's. It states that if two terms are considered syntactically equal (structurally congruent), then they are also semantically equal.

Lemma 4.7 *Let* $s, t \in \mathbb{S}$. *If* $s \equiv t$ *then* $s \cong t$.

Proof

- We first prove $s \equiv t \;\Rightarrow\; s \leqslant t$.
  We have to prove the cases "*transition*" and "*termination.*"
  transition

  If $\langle s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$ then, by (N9) and $s \equiv t$ follows $\langle t, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$. By reflexivity of $\leqslant$, we have $s' \leqslant s'$.

  termination

  If $s \equiv \mathsf{skip}$, then by transitivity of $\equiv$ follows $t \equiv \mathsf{skip}$.

- The proof $s \equiv t \;\Rightarrow\; t \leqslant s$ is analogous.

□

Lemma 4.8 *Let* s_1, s_2, t_1 *and* $t_2 \in \mathbb{S}$ *such that* $s_1 \leqslant s_2$ *and* $t_1 \leqslant t_2$, *then* $r \to s_1[t_1] \leqslant r \to s_2[t_2]$.

Proof We have to prove the two cases "*transition*" and "*termination.*"

transition

We consider the possible transitions of the schedule $r \to s_1[t_1]$:

- A transition $\langle r \to s_1[t_1], M\rangle \xrightarrow{\varepsilon} \langle t_1, M\rangle$ can be derived using (N0) if $\langle r, M\rangle\surd$. Then, also by (N0), $\langle r \to s_2[t_2], M\rangle \xrightarrow{\varepsilon} \langle t_2, M\rangle$ and by assumption $t_1 \leqslant t_2$.
- $\langle r \to s_1[t_1], M\rangle \xrightarrow{\sigma} \langle s_1, M'\rangle$ can be derived using (N1) from $\langle r, M\rangle \overset{\sigma}{\rightsquigarrow} \langle r, M'\rangle$. Then, also by (N1), $\langle r \to s_2[t_2], M\rangle \xrightarrow{\sigma} \langle s_2, M'\rangle$ and by assumption $s_1 \leqslant s_2$.

termination

This proof obligation is vacuous.

□

Lemma 4.9 *Let s_1, s_2, t_1 and $t_2 \in \mathbb{S}$ such that $s_1 \leqslant s_2$ and $t_1 \leqslant t_2$, then $s_1; t_1 \leqslant s_2; t_2$.*

Proof We show that the relation $\mathcal{R} = \{(s_1; t_1,\ s_2; t_2) \mid s_1 \leqslant s_2,\ t_1 \leqslant t_2\}$ is a simulation up to $\leqslant$.
transition

Let $(s_1; t_1, s_2; t_2) \in \mathcal{R}$. A transition for $s_1; t_1$ can be derived in two ways:

- By (N5) from $\langle s_1, M\rangle \xrightarrow{\lambda} \langle s_1', M'\rangle$. From $s_1 \leqslant s_2$ follows $\langle s_2, M\rangle \xrightarrow{\lambda} \langle s_2', M'\rangle$ such that $s_1' \leqslant s_2'$. By (N5) we infer $\langle s_2; t_2, M\rangle \xrightarrow{\lambda} \langle s_2'; t_2, M'\rangle$. From $t_1 \leqslant t_2$ and $Id_{\mathbb{S}} \subseteq \leqslant$ now follows that $(s_1'; t_1, s_2'; t_2) \in \ \leqslant \mathcal{R} \leqslant$.
- By (N9) from $s_1 \equiv \mathsf{skip}$ and $\langle t_1, M\rangle \xrightarrow{\lambda} \langle t_1', M'\rangle$. Because $s_1 \leqslant s_2$ and $t_1 \leqslant t_2$ we have $s_2 \equiv \mathsf{skip}$ and $\langle t_2, M\rangle \xrightarrow{\lambda} \langle t_2', M'\rangle$ such that $t_1' \leqslant t_2'$. By (N9) we infer $\langle s_2; t_2, M\rangle \xrightarrow{\lambda} \langle t_2', M'\rangle$. By Lemma 4.7 $t_1' \leqslant \mathsf{skip}; t_1'$ and $\mathsf{skip}; t_2' \leqslant t_2'$. Hence from $(\mathsf{skip}; t_1', \mathsf{skip}; t_2') \in \mathcal{R}$ follows $(t_1', t_2') \in \ \leqslant \mathcal{R} \leqslant$.

termination

$s_1; t_1 \equiv \mathsf{skip}$ only if both $s_1 \equiv \mathsf{skip}$ and $t_1 \equiv \mathsf{skip}$. From $s_1 \leqslant s_2$ and $t_1 \leqslant t_2$ follows that also $s_2 \equiv \mathsf{skip}$ and $t_2 \equiv \mathsf{skip}$, hence $s_2; t_2 \equiv \mathsf{skip}$.

□

Corollary 4.10 *Given s_1, s_2 and $t \in \mathbb{S}$ such that $s_1 \leqslant s_2$, then*

i. $s_1; t \leqslant s_2; t$

ii. $t; s_1 \leqslant t; s_2$

Proof

This follows from Lemma 4.9 and reflexivity of $\leqslant$.

□

Lemma 4.11 *Let s_1, s_2, t_1 and $t_2 \in \mathbb{S}$ such that $s_1 \leqslant s_2$ and $t_1 \leqslant t_2$, then $s_1 \parallel t_1 \leqslant s_2 \parallel t_2$.*

Proof We show that the relation $\mathcal{R} = \{(s_1 \parallel t_1, s_2 \parallel t_2) \mid s_1 \leqslant s_2,\ t_1 \leqslant t_2\}$ is a simulation.

transition
We continue by analyzing the possible ways by which transitions $\langle s_1 \parallel t_1, M\rangle \xrightarrow{\lambda} \langle s_1' \parallel t_1', M'\rangle$ may be derived.

1. By (N2) $\langle s_1, M\rangle \xrightarrow{\lambda} \langle s_1', M'\rangle$ hence $t_1' \equiv t_1$. From $s_1 \leqslant s_2$ follows that $\langle s_2, M\rangle \xrightarrow{\lambda} \langle s_2', M'\rangle$ such that $s_1' \leqslant s_2'$. By (N2) we get $\langle s_2 \parallel t_2, M\rangle \xrightarrow{\lambda} \langle s_2' \parallel t_2, M'\rangle$. From the definition of $\mathcal{R}$ follows that $(s_1' \parallel t_1, s_2' \parallel t_2) \in \mathcal{R}$.

2. By (N2) $\langle t_1, M\rangle \xrightarrow{\lambda} \langle t_1', M'\rangle$ hence $s_1' \equiv s_1$.
   The proof goes analogous to the previous case.

3. By (N3) from $\langle s_1, M\rangle \xrightarrow{\lambda} \langle s_1', M'\rangle$ and $\langle t_1, M\rangle \xrightarrow{\varepsilon} \langle t_1', M\rangle$. From $s_1 \leqslant s_2$ and $t_1 \leqslant t_2$ follows that $\langle s_2, M\rangle \xrightarrow{\lambda} \langle s_2', M'\rangle$ and $\langle t_2, M\rangle \xrightarrow{\varepsilon} \langle t_2', M\rangle$. such that $s_1' \leqslant s_2'$ and $t_1' \leqslant t_2'$. By (N3) we get $\langle s_2 \parallel t_2, M\rangle \xrightarrow{\lambda} \langle s_2' \parallel t_2', M'\rangle$ and $(s_1' \parallel t_1', s_2' \parallel t_2') \in \mathcal{R}$.

4. By (N3) from $\langle s_1, M\rangle \xrightarrow{\lambda} \langle s_1', M'\rangle$ and $\langle t_1, M\rangle \xrightarrow{\varepsilon} \langle t_1', M\rangle$.
   The proof continues analogous to the previous case.

5. By (N4) from $\langle s_1, M\rangle \xrightarrow{\sigma_1} \langle s_1', M_1\rangle$ and $\langle t_1, M\rangle \xrightarrow{\sigma_2} \langle t_1', M_2\rangle$ and $M \models \sigma_1 \bowtie \sigma_2$. From $s_1 \leqslant s_2$ and $t_1 \leqslant t_2$ follows that $\langle s_2, M\rangle \xrightarrow{\sigma_1} \langle s_2', M_1\rangle$ and $\langle t_2, M\rangle \xrightarrow{\sigma_2} \langle t_2', M_2\rangle$

such that $s_1' \leqslant s_2'$ and $t_1' \leqslant t_2'$. Because $M \models \sigma_1 \bowtie \sigma_2$ we conclude by (N4) that $\langle s_2 \parallel t_2, M\rangle \xrightarrow{\sigma} \langle s_1' \parallel t_2', M'\rangle$ and from the definition of $\mathcal{R}$ follows $(s_1' \parallel t_1', s_2' \parallel t_2') \in \mathcal{R}$.

transition
$s_1 \parallel t_1 \equiv \mathsf{skip}$ only if $s_1 \equiv \mathsf{skip}$ and $t_1 \equiv \mathsf{skip}$. From $s_1 \leqslant s_2$ and $t_1 \leqslant t_2$ follows that also $s_2 \equiv \mathsf{skip}$ and $t_2 \equiv \mathsf{skip}$, hence $s_2 \parallel t_2 \equiv \mathsf{skip}$.

□

Lemma 4.12 *Let s_1, s_2, t_1 and $t_2 \in \mathbb{S}$ such that $s_1 \leqslant s_2$ and $t_1 \leqslant t_2$, then $c \triangleright s_1[t_1] \leqslant c \triangleright s_2[t_2]$.*

Proof Consider the cases

- $c = \mathsf{true}$: Then $c \triangleright s_1[t_1] \equiv s_1$ and $c \triangleright s_2[t_2] \equiv s_2$ and by assumption $s_1 \leqslant s_2$.
- $c = \mathsf{false}$: Then $c \triangleright s_1[t_1] \equiv t_1$ and $c \triangleright s_2[t_2] \equiv t_2$ and by assumption $t_1 \leqslant t_2$.

□

Lemma 4.13 *Let s_1 and $s_2 \in \mathbb{S}$ such that $s_1 \leqslant s_2$, then $!s_1 \leqslant !s_2$.*

Proof
Let $\mathcal{R} = \{(t_1 \parallel !s_1, t_2 \parallel !s_2) \mid t_1 \leqslant t_2,\ s_1 \leqslant s_2\} \cup \leqslant$. We show that $\mathcal{R}$ is a simulation by induction on the depth of the inference. We will use the following property of $\mathcal{R}$

$$\text{If } (s_1, s_2) \in \mathcal{R} \text{ and } t_1 \leqslant t_2, \text{ then } (t_1 \parallel s_1, t_2 \parallel s_2) \in \mathcal{R} \quad (*)$$

transition

We consider the possible transitions for $(t_1 \parallel !s_1, t_2 \parallel !s_2)$.
A transition can be derived in the following ways:

1. By (N2) from $\langle t_1, M\rangle \xrightarrow{\lambda} \langle t_1', M'\rangle$. From $t_1 \leqslant t_2$ follows $\langle t_2, M\rangle \xrightarrow{\lambda} \langle t_2', M'\rangle$ such that $t_1' \leqslant t_2'$. By (N2) we infer $\langle t_2 \parallel !s_2, M\rangle \xrightarrow{\lambda} \langle t_2' \parallel !s_2, M'\rangle$. Clearly $(t_1' \parallel !s_1, t_2' \parallel !s_2) \in \mathcal{R}$.

2. By (N2) from $\langle !s_1, M\rangle \xrightarrow{\lambda} \langle s_1', M'\rangle$. This transition can be derived in the following ways.

– By (N6) from $\langle s_1, M\rangle \xrightarrow{\lambda} \langle s_1', M'\rangle$. From $s_1 \leqslant s_2$ follows $\langle s_2, M\rangle \xrightarrow{\lambda} \langle s_2', M'\rangle$ such that $s_1' \leqslant s_2'$. Then by (N6) $\langle !s_2, M\rangle \xrightarrow{\lambda} \langle s_2', M'\rangle$, and by (N2) $\langle t_2 \parallel !s_2, M\rangle \xrightarrow{\lambda} \langle t_2 \parallel s_2', M'\rangle$. From Lemma 4.11 follows that $t_1 \parallel s_1' \leqslant t_2 \parallel s_2'$, hence $(t_1 \parallel s_1', t_2 \parallel s_2') \in \mathcal{R}$.

– By (N7) from $\langle s_1 \parallel !s_1, M\rangle \xrightarrow{\lambda} \langle s_1', M'\rangle$. By the induction hypothesis we get $\langle s_2 \parallel !s_2, M\rangle \xrightarrow{\lambda} \langle s_2', M'\rangle$ such that $(s_1', s_2') \in \mathcal{R}$. By (N7) we infer $\langle !s_2, M\rangle \xrightarrow{\lambda} \langle s_2', M'\rangle$. From (N2) we get $\langle t_2 \parallel !s_2, M\rangle \xrightarrow{\lambda} \langle t_2 \parallel s_2', M'\rangle$. From $(s_1', s_2') \in \mathcal{R}$ and by (*) follows $(t_1 \parallel s_1', t_2 \parallel s_2') \in \mathcal{R}$.

3. By (N3) from $\langle t_1, M\rangle \xrightarrow{\lambda} \langle t_1', M'\rangle$ and $\langle !s_1, M\rangle \xrightarrow{\varepsilon} \langle s_1', M\rangle$.
   The proof of is a routine combination of cases 1. and 2.
4. By (N3) from $\langle t_1, M\rangle \xrightarrow{\varepsilon} \langle t_1', M\rangle$ and $\langle !s_1, M\rangle \xrightarrow{\lambda} \langle s_1', M'\rangle$.
   The remainder of the proof is analogous to case 3.
5. By (N4) from $\langle t_1, M\rangle \xrightarrow{\sigma_1} \langle t_1', M_1\rangle$ and $\langle !s_1, M\rangle \xrightarrow{\sigma_2} \langle s', M_2\rangle$ where $M \models \sigma_1 \bowtie \sigma_2$.
   The proof is analogous to case 3.

termination
$t_1 \parallel !s_1 \equiv \mathsf{skip}$ only if $t_1 \equiv \mathsf{skip}$ and $s_1 \equiv \mathsf{skip}$. From $t_1 \leqslant t_2$ and $s_1 \leqslant s_2$ follows $t_2 \equiv \mathsf{skip}$ and $s_2 \equiv \mathsf{skip}$, hence $t_2 \parallel !s_2 \equiv \mathsf{skip}$. □

Simulation is defined in terms of the possible transitions of schedules. Because the behaviour of schedules that contain schedule-variables is unknown, simulation as we have seen so far deals only with ground schedules. We would also like to manipulate schedule expressions containing variables. Therefore, we extend the definition of $\leqslant$ to cover schedule expressions as follows.

Definition 4.9 *Let s_1 and $s_2 \in \mathbb{S}$ contain control variables $\overline{x}$ at most, and schedule variables $\overline{X}$ at most. Then $s_1 \leqslant s_2$ if, for all values $\overline{v}$ and ground schedules $\overline{t} \in \mathbb{S}_{ground}$, $s_1[\overline{x} := \overline{v}]\{\overline{t}/\overline{X}\} \leqslant s_2[\overline{x} := \overline{v}]\{\overline{t}/\overline{X}\}$.*

We proceed by showing that recursive definitions preserve equivalence.

Lemma 4.14 *Let $S \in \mathcal{S}$ be a schedule variable, and $s \in \mathbb{S}_{ground}$ be a ground schedule (i.e. without free schedule variables) such that $S(\overline{x}) \mathrel{\hat{=}} s$, then $S(\overline{x}) \cong s$.*

Proof

It is straightforward to prove, using (N8), that $S(\overline{x})$ and s have the same derivatives for any $\overline{v}$. Hence, for any $\overline{v}$, both $s[\overline{x} := \overline{v}] \leqslant S(\overline{v})$ and $S(\overline{v}) \leqslant s[\overline{x} := \overline{v}]$.

□

Lemma 4.15 is concerned with the schedule-variables that appear in a schedule. The control variables play no role of importance and have been left out to increase readability.

Lemma 4.15 *Let s_1 and $s_2 \in \mathbb{S}$ contain at most schedule variable X. Let S_1 and $S_2 \in \mathcal{S}$ be defined by $S_1 \mathrel{\widehat{=}} s_1\{S_1/X\}$, and $S_2 \mathrel{\widehat{=}} s_2\{S_2/X\}$. If $s_1 \leqslant s_2$, then $S_1 \leqslant S_2$.*

Proof We show that

$$\mathcal{R} = \{(t\{S_1/X\}, t\{S_2/X\}) \mid t \text{ contains at most the variable } X\}$$

is a simulation up–to– $\leqslant$. By induction on the structure of t for termination and on the depth of the inference of $\langle t\{S_1/X\}, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$ for any transition.

termination

We must show that $t\{S_1/X\} \equiv \mathsf{skip} \;\Rightarrow t\{S_2/X\} \equiv \mathsf{skip}$. We consider the possible cases for t:

- $t \equiv \mathsf{skip}$:
  Then $t\{S_1/X\} \equiv \mathsf{skip} \equiv t\{S_2/X\}$.
- $t \equiv X$:
  Then $t\{S_1/X\} = S_1$ and $t\{S_2/X\} = S_2$. Clearly S_1 does not syntactically equal skip, hence this case holds vacuously.
- $t \equiv r \to t_1[t_2]$:
  There exists no schedules t_1 and t_2 such that $r \to t_1[t_2] \equiv \mathsf{skip}$, hence this case holds vacuously.
- $t \equiv c \triangleright t_1[t_2]$:
  Then $t\{S_1/X\} = c \triangleright t_1\{S_1/X\}[t_2\{S_1/X\}]$
  and $t\{S_2/X\} = c \triangleright t_1\{S_2/X\}[t_2\{S_2/X\}]$.

– If $c = \mathit{true}$ then $t_1\{S_1/X\} \equiv \mathsf{skip}$.
By induction $t_1\{S_2/X\} \equiv \mathsf{skip}$, hence $t\{S_2/X\} \equiv \mathsf{skip}$.

– If $c = \mathit{false}$ the proof proceeds analogously.

- $t \equiv t_1 \,\|\, t_2$:
Then $t_1\{S_1/X\} \,\|\, t_2\{S_1/X\} \equiv \mathsf{skip}$ only if, by (E1), $t_1\{S_1/X\} \equiv \mathsf{skip}$ and $t_2\{S_1/X\} \equiv \mathsf{skip}$. By induction $t_1\{S_2/X\} \equiv \mathsf{skip}$ and $t_2\{S_2/X\} \equiv \mathsf{skip}$. By (E1) we conclude $t_1\{S_2/X\} \,\|\, t_2\{S_2/X\} \equiv \mathsf{skip}$.

- $t \equiv t_1; t_2$:
Analogous to the previous case.

- $t \equiv !t'$:
Then $!t'\{S_1/X\} \equiv \mathsf{skip}$ only if, by (E8), $t'\{S_1/X\} \equiv \mathsf{skip}$.
By induction $t'\{S_2/X\} \equiv \mathsf{skip}$. By (E8) we conclude $!t\{S_2/X\} \equiv \mathsf{skip}$.

- $t \equiv T$, where $T \mathrel{\widehat{=}} t'$ and t' is a schedule without variables:
Then $t\{S_1/X\} = t\{S_2/X\} = t'$. Clearly $t\{S_1/X\} \equiv \mathsf{skip} \;\Leftrightarrow t\{S_2/X\} \equiv \mathsf{skip}$.

transition
We consider the possible transitions for $\langle t, M\rangle \xrightarrow{\lambda} \langle t', M'\rangle$ where t is one of the following:

- $t \equiv X$:
Then $t\{S_1/X\} \equiv S_1$, hence the transition we consider is $\langle S_1, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$. This must be inferred by (N8), using $S_1 \mathrel{\widehat{=}} s_1\{S_1/X\}$, from $\langle s_1\{S_1/X\}, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$. This transition is derived by a shorter inference, hence by the induction hypothesis $\langle s_1\{S_2/X\}, M\rangle \xrightarrow{\lambda} \langle s'', M'\rangle$ with $(s', s'') \in \leqslant\mathcal{R}\leqslant$. From $s_1 \leqslant s_2$ follows $\langle s_2\{S_2/X\}, M\rangle \xrightarrow{\lambda} \langle s''', M'\rangle$ with $(s'', s''') \in \leqslant$. Because $S_2 \mathrel{\widehat{=}} s_2\{S_2/X\}$ and $S_2 \equiv t\{S_2/X\}$ we get by (N8) $\langle t\{S_2/X\}, M\rangle \xrightarrow{\lambda} \langle s''', M'\rangle$ with $(s', s''') \in \leqslant\mathcal{R}\leqslant$ as required.

- $t \equiv r \to t_1[t_2]$:
Then $t\{S_1/X\} \equiv r \to t_1\{S_1/X\}[t_2\{S_1/X\}]$. Here t_1 and t_2 contain at most the variable X. The transitions we have to consider are

– If $\langle r \to t_1\{S_1/X\}[t_2\{S_1/X\}], M\rangle \xrightarrow{\varepsilon} \langle t_2\{S_1/X\}, M\rangle$, then by (N0) also $\langle r \to t_1\{S_2/X\}[t_2\{S_2/X\}], M\rangle \xrightarrow{\varepsilon} \langle t_2\{S_2/X\}, M\rangle$.
By definition of $\mathcal{R}$: $(t_2\{S_1/X\}, t_2\{S_2/X\}) \in \leqslant\mathcal{R}\leqslant$.

– $\langle r \to t_1\{S_1/X\}[t_2\{S_1/X\}], M\rangle \xrightarrow{\sigma} \langle t_1\{S_1/X\}, M'\rangle$. The proof is analogous to the previous case.

- $t \equiv c \triangleright t_1[t_2]$:
  Then $t\{S_1/X\} = c \triangleright t_1\{S_1/X\}[t_2\{S_1/X\}]$ and $t\{S_2/X\} = c \triangleright t_1\{S_2/X\}[t_2\{S_2/X\}]$.
  The following reasoning holds for $i = 1$ if $c = \mathit{true}$ and $i = 2$ if $c = \mathit{false}$.
  A transition can be derived by (N9) from $c \triangleright t_1[t_2] \equiv t_i$ and
  $\langle t_i\{S_1/X\}, M\rangle \xrightarrow{\lambda} \langle t_i', M'\rangle$. This transition is derived by a shorter inference, hence by induction we get $\langle t_i\{S_2/X\}, M\rangle \xrightarrow{\lambda} \langle t_i'', M'\rangle$ such that $(t_i', t_i'') \in \leqslant\mathcal{R}\leqslant$.
  By (N9) we derive $\langle c \triangleright t_1[t_2]\{S_2/X\}, M\rangle \xrightarrow{\lambda} \langle t_i'', M'\rangle$.

- $t \equiv t_1; t_2$:
  Then $t\{S_1/X\} = t_1\{S_1/X\}; t_2\{S_1/X\}$.
  There are two possibilities for deriving a transition:

  – By (N5) from $\langle t_1\{S_1/X\}, M\rangle \xrightarrow{\lambda} \langle t_1', M'\rangle$, hence $t' \equiv t_1'; t_2\{S_1/X\}$.
    This is derived by a shorter inference, so by induction $\langle t_1\{S_2/X\}, M\rangle \xrightarrow{\lambda} \langle t_1'', M'\rangle$ such that $(t_1', t_1'') \in \leqslant\mathcal{R}\leqslant$.
    Then by (N5) $\langle t_1\{S_2/X\}; t_2\{S_2/X\}, M\rangle \xrightarrow{\lambda} \langle t_1''; t_2\{S_2/X\}, M'\rangle$.
    From $(t_1', t_1'') \in \leqslant\mathcal{R}\leqslant$ follows that there are g and g' such that $t_1' \leqslant g$, $(g, g') \in \mathcal{R}$ and $g' \leqslant t_1''$. Then by monotonicity of ; with respect to $\leqslant$ follows that $t_1'; t_2\{S_2/X\} \leqslant g; t_2\{S_2/X\}$ and $g'; t_2\{S_2/X\} \leqslant t_1''; t_2\{S_2/X\}$.
    Because t_2 contains at most variable X, we get by definition of $\mathcal{R}$ that $(g; t_2\{S_2/X\}, g'; t_2\{S_2/X\}) \in \mathcal{R}$, hence $(t_1'; t_2\{S_2/X\}, t_1''; t_2\{S_2/X\}) \in \leqslant\mathcal{R}\leqslant$ as required.

  – By (N9) from $t_1\{S_1/X\} \equiv \mathsf{skip}$ and $\langle t_2\{S_1/X\}, M\rangle \xrightarrow{\lambda} \langle t_2', M'\rangle$, hence $t' \equiv t_2'$.
    By the *termination*-part of this proof we know that $t_1\{S_2/X\} \equiv \mathsf{skip}$.
    This transition is derived by a shorter inference, so by induction $\langle t_2\{S_2/X\}, M\rangle \xrightarrow{\lambda} \langle t_2'', M'\rangle$ such that $(t_2', t_2'') \in \leqslant\mathcal{R}\leqslant$.
    By (N9) we infer $\langle t_1\{S_2/X\}; t_2\{S_2/X\}, M\rangle \xrightarrow{\lambda} \langle t_2'', M'\rangle$.

- $t \equiv t_1 \parallel t_2$:
  Then $t\{S_1/X\} = t_1\{S_1/X\} \parallel t_2\{S_1/X\}$ and a transition can be derived by

  – (N2) from $\langle t_1\{S_1/X\}, M\rangle \xrightarrow{\lambda} \langle t_1', M'\rangle$.
    By induction $\langle t_1\{S_2/X\}, M\rangle \xrightarrow{\lambda} \langle t_1'', M'\rangle$ such that $(t_1', t_1'') \in \leqslant\mathcal{R}\leqslant$.
    By (N2) $\langle t_1\{S_2/X\} \parallel t_2\{S_2/X\}, M\rangle \xrightarrow{\lambda} \langle t_1'' \parallel t_2\{S_2/X\}, M'\rangle$.

From $(t_1', t_1'') \in \leqslant\mathcal{R}\leqslant$ follows that there are g and g' that contain at most variable X such that $t_1' \leqslant g$, $(g, g') \in \mathcal{R}$ and $g' \leqslant t_1''$.
By monotonicity of $\parallel$ with respect to $\leqslant$ follows that
$t_1' \parallel t_2\{S_2/X\} \leqslant g \parallel t_2\{S_2/X\})$ and $g' \parallel t_2\{S_2/X\} \leqslant t_1'' \parallel t_2\{S_2/X\}$.
Because t_2 contains at most variable X, we get by definition of $\mathcal{R}$ that $(g \parallel t_2\{S_2/X\}, g' \parallel t_2\{S_2/X\}) \in \mathcal{R}$, hence $(t_1' \parallel t_2\{S_2/X\}, t_1'' \parallel t_2\{S_2/X\}) \in \leqslant\mathcal{R}\leqslant$ as required.

- (N2) from $\langle t_2\{S_1/X\}, M\rangle \xrightarrow{\lambda} \langle t_2', M'\rangle$.
  The proof is analogous to the previous case.
- (N3) from $\langle t_1\{S_1/X\}, M\rangle \xrightarrow{\lambda} \langle t_1', M'\rangle$ and $\langle t_2\{S_1/X\}, M\rangle \xrightarrow{\varepsilon} \langle t_2', M\rangle$.
  By induction $\langle t_1\{S_2/X\}, M\rangle \xrightarrow{\lambda} \langle t_1'', M'\rangle$ and $\langle t_2\{S_2/X\}, M\rangle \xrightarrow{\varepsilon} \langle t_2'', M\rangle$ such that $(t_1', t_1'') \in \leqslant\mathcal{R}\leqslant$ and $(t_2', t_2'') \in \leqslant\mathcal{R}\leqslant$.
  Hence there are g, g', h, h' that contain at most variable X such that $t_1' \leqslant g$, $(g, g') \in \mathcal{R}$, $g' \leqslant t_1''$ and $t_2' \leqslant h$, $(h, h') \in \mathcal{R}$ $h' \leqslant t_2''$. Then by monotonicity of $\parallel$ w.r.t. $\leqslant$ we have $t_1' \parallel t_2' \leqslant g \parallel h$ and $g' \parallel h' \leqslant t_1'' \parallel t_2''$. By definition of $\mathcal{R}$ we have $(g \parallel h, g' \parallel h') \in \mathcal{R}$, hence $(t_1' \parallel t_2', t_1'' \parallel t_2'') \in \leqslant\mathcal{R}\leqslant$.
- (N3) from $\langle t_1\{S_1/X\}, M\rangle \xrightarrow{\varepsilon} \langle t_1', M\rangle$ and $\langle t_2\{S_1/X\}, M\rangle \xrightarrow{\lambda} \langle t_2', M'\rangle$.
  The proof is analogous to the previous case.
- (N4) from $\langle t_1\{S_1/X\}, M\rangle \xrightarrow{\sigma_1} \langle t_1', M_1\rangle$ and $\langle t_2\{S_1/X\}, M\rangle \xrightarrow{\sigma_2} \langle t_2', M_2\rangle$ where $M \models \sigma_1 \bowtie \sigma_2$. The proof is analogous to the previous case.

- $t \equiv !t'$:
  Then $t\{S_1/X\} = !t'\{S_1/X\}$. A transition can be derived in the following ways:
  - By (N6) from $\langle t'\{S_1/X\}, M\rangle \xrightarrow{\lambda} \langle t'', M'\rangle$.
    The term t' contains at most the variable X, and the transition is derived by a shorter inference hence the induction hypothesis gives $\langle t'\{S_2/X\}, M\rangle \xrightarrow{\lambda} \langle t''', M'\rangle$ such that $(t'', t''') \in \leqslant\mathcal{R}\leqslant$. By (N6) $\langle !t'\{S_2/X\}, M\rangle \xrightarrow{\lambda} \langle t''', M'\rangle$.
  - By (N7) from $\langle t'\{S_1/X\} \parallel !t'\{S_1/X\}, M\rangle \xrightarrow{\lambda} \langle t'', M'\rangle$.
    Because $t = !t'$ contains at most variable X, so does t', hence also $t' \parallel !t'$. The transition is derived by a shorter inference, hence the induction hypothesis gives $\langle t'\{S_2/X\} \parallel !t'\{S_2/X\}, M\rangle \xrightarrow{\lambda} \langle t''', M'\rangle$ such that
    $(t'', t''') \in \leqslant\mathcal{R}\leqslant$. By (N7) $\langle !t'\{S_2/X\}, M\rangle \xrightarrow{\lambda} \langle t''', M'\rangle$.

- $t \equiv T$, where $T \mathrel{\widehat{=}} t'$ and t' is a schedule without variables:
  Then $t\{S_1/X\} \equiv t\{S_2/X\} \equiv t'$. If $\langle t\{S_1/X\}, M\rangle \xrightarrow{\lambda} \langle t'', M'\rangle$ then $\langle t\{S_2/X\}, M\rangle \xrightarrow{\lambda} \langle t'', M'\rangle$. From $(t'', t'') \equiv (t''\{S_1/X\}, t''\{S_2/X\})$ and reflexivity of $\leqslant$ follows $(t'', t'') \in \leqslant \mathcal{R} \leqslant$.

□

Theorem 4.16 $\leqslant$ *is a precongruence on* $\mathbb{S}$ *(the terms of the schedule language).*

Proof From Lemmas 4.8, 4.9, 4.11, 4.12, 4.13 and 4.15. □

It follows that $\cong$ is a congruence on schedules.

Corollary 4.17 $\cong$ *is a congruence relation on schedules.*

Proof Let $u[x]$ be a schedule that has x as a subterm. Let s and t be schedules such that $s \cong t$, we have to prove that $u[s] \cong u[t]$. By Definition 4.6 $s \cong t$ if and only if $s \leqslant t$ and $t \leqslant s$. By Theorem 4.16 follows $u[s] \leqslant u[t]$ and $u[t] \leqslant u[s]$. By Definition 4.6 $u[s] \cong u[t]$. □

4.4 Refinement Laws

In the previous section we showed that simulation is a precongruence. In this section we present a number of the basic refinement laws. These laws give additional insight into the algebraic properties of refinement. Furthermore, the laws give rise to an algebraic style of reasoning about schedules. Precongruence of $\leqslant$ guarantees the modular applicability of the laws.

We continue by presenting the laws grouped per operator.

4.4.1 Laws for Rule Conditional Composition

The first law can be used to move a single rule-conditional out of a parallel composition such that it is scheduled for execution first. The second law is a special case of the first. The fact that ";" enforces a more determined ordering on the execution of schedules,

has as a consequence that the law for ";" is a congruence, while the case for " $\|$ " is a refinement.

Lemma 4.18

$$\begin{array}{lrcl} 1. & r \to (s_1 \,\|\, t)[s_2 \,\|\, t] & \leqslant & (r \to s_1[s_2]) \,\|\, t \\ 2. & r \to (s_1; t)[s_2; t] & \cong & (r \to s_1[s_2]); t \end{array}$$

Proof We consider case 1. Case 2 then follows by Lemma 4.22.
transition
There are two possible transitions:

- If $\langle r \to (s_1 \,\|\, t)[s_2 \,\|\, t], M\rangle \xrightarrow{\sigma} \langle s_1 \,\|\, t, M'\rangle$ then,
  by (N1), $\langle (r \to s_1[s_2]) \,\|\, t, M\rangle \xrightarrow{\sigma} \langle s_1 \,\|\, t, M'\rangle$. By reflexivity, $s_1 \,\|\, t \leqslant s_1 \,\|\, t$.
- If $\langle r \to (s_1 \,\|\, t)[s_2 \,\|\, t], M\rangle \xrightarrow{\varepsilon} \langle s_2 \,\|\, t, M\rangle$ then,
  by (N0), $\langle (r \to s_1[s_2]) \,\|\, t, M\rangle \xrightarrow{\varepsilon} \langle s_2 \,\|\, t, M\rangle$. By reflexivity, $s_2 \,\|\, t \leqslant s_2 \,\|\, t$.

termination
There are no s_1, s_2, t_1 and t_2 such that $r \to (s_1 \,\|\, t)[s_2 \,\|\, t] \equiv \mathsf{skip}$, hence this case hold vacuously. □

4.4.2 Laws for Sequential Composition

The laws from Lemma 4.19 show that ";" is a monoid with unit skip.

Lemma 4.19

$$\begin{array}{lrcl} 1. & \mathsf{skip}; s & \cong & s \\ 2. & s; \mathsf{skip} & \cong & s \\ 3. & s_1; (s_2; s_3) & \cong & (s_1; s_2); s_3 \end{array}$$

Proof
Cases 1. and 3. follow from the structural congruence and Lemma 4.7. We consider the second case $s; \mathsf{skip} \cong s$. We have to prove $s; \mathsf{skip} \leqslant s$ and $s \leqslant s; \mathsf{skip}$. We give the details for the former; the proof of the latter goes analogously.

> Let $\mathcal{R} = \{(s; \mathsf{skip}, s) \mid s \in \mathbb{S}\}$. We show that $\mathcal{R}$ is a simulation.
> *transition*

The only way to derive a transition for $s;\mathsf{skip}$ is by (N5) from $\langle s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$. By definition of $\mathcal{R}$: $(s';\mathsf{skip}, s') \in \mathcal{R}$. The case $s \equiv \mathsf{skip}$ is covered by *termination*.

termination

$s;\mathsf{skip} \equiv \mathsf{skip}$ only if $s \equiv \mathsf{skip}$.

□

4.4.3 Laws for Parallel Composition

The laws for parallel composition follow from structural congruence and Lemma 4.7. They show that "$\|$" is a commutative monoid with unit skip.

Lemma 4.20

1.	$\mathsf{skip} \,\|\, s$	$\cong$	s
2.	$s_1 \,\|\, s_2$	$\cong$	$s_2 \,\|\, s_1$
3.	$s_1 \,\|\, (s_2 \,\|\, s_3)$	$\cong$	$(s_1 \,\|\, s_2) \,\|\, s_3$

Proof By structural congruence and Lemma 4.7. □

4.4.4 Laws of Distributivity of Parallel and Sequential Composition

The next Lemma yields a number of useful laws for the distribution of sequential and parallel composition.

Lemma 4.21 *Let* $s_i \in \mathbb{S}$ *for* $1 \leq i \leq 4$, *then* $(s_1 \,\|\, s_3);(s_2 \,\|\, s_4) \leqslant (s_1; s_2) \,\|\, (s_3; s_4)$

Proof
Let $\mathcal{R} = \{((s_1 \,\|\, s_3);(s_2 \,\|\, s_4), (s_1; s_2) \,\|\, (s_3; s_4)) \mid s_1, s_2, s_3, s_4 \in \mathbb{S}\} \cup Id_{\mathbb{S}}$. We show that $\mathcal{R}$ is a simulation. For any pair $(s, s) \in Id_{\mathbb{S}}$ the conditions for *transition* and *termination* hold by reflexivity of refinement (Lemma 4.1(1)). We consider the interesting case.

transition
We consider the possible transitions.

- By rule (N5) the first transition is derived from $\langle s_1 \parallel s_3, M\rangle \xrightarrow{\lambda} \langle s_1' \parallel s_3', M'\rangle$. This is in turn derived in one of the following ways.

  1. By (N2) from $\langle s_1, M\rangle \xrightarrow{\lambda} \langle s_1', M'\rangle$ hence $s_3' \equiv s_3$.
     By (N5) we get $\langle s_1; s_2, M\rangle \xrightarrow{\lambda} \langle s_1'; s_2, M'\rangle$.
     By (N2) we infer $\langle (s_1; s_2) \parallel (s_3; s_4), M\rangle \xrightarrow{\lambda} \langle (s_1'; s_2) \parallel (s_3; s_4), M'\rangle$.
     And $((s_1' \parallel s_3); (s_2 \parallel s_4), (s_1'; s_2) \parallel (s_3; s_4)) \in \mathcal{R}$.
  2. By (N2) from $\langle s_3, M\rangle \xrightarrow{\lambda} \langle s_3', M'\rangle$ hence $s_1' \equiv s_1$.
     The proof is analogous to the previous case.
  3. By (N3) from $\langle s_1, M\rangle \xrightarrow{\lambda} \langle s_1', M'\rangle$ and $\langle s_3, M\rangle \xrightarrow{\varepsilon} \langle s_3', M\rangle$. By (N5) we get for the former $\langle s_1; s_2, M\rangle \xrightarrow{\lambda} \langle s_1'; s_2, M'\rangle$, and for the latter $\langle s_3; s_4, M\rangle \xrightarrow{\varepsilon} \langle s_3'; s_4, M\rangle$.
     Then by (N3) we obtain $\langle (s_1; s_2) \parallel (s_3; s_4), M\rangle \xrightarrow{\lambda} \langle (s_1'; s_2) \parallel (s_3'; s_4), M'\rangle$. And $((s_1' \parallel s_3'); (s_2 \parallel s_4), (s_1'; s_2) \parallel (s_3'; s_4)) \in \mathcal{R}$.
  4. By (N3) from $\langle s_1, M\rangle \xrightarrow{\varepsilon} \langle s_1', M\rangle$ and $\langle s_3, M\rangle \xrightarrow{\lambda} \langle s_3', M'\rangle$.
     The proof is similar to the previous case.
  5. By (N4) from $\langle s_1, M\rangle \xrightarrow{\sigma_1} \langle s_1', M_1\rangle$ and $\langle s_3, M\rangle \xrightarrow{\sigma_2} \langle s_3', M_2\rangle$ such that $M \models \sigma_1 \bowtie \sigma_2$. The proof is analogous to the previous case where use of (N3) should be replaced by use of (N4).

- By (N9) from $(s_1 \parallel s_3) \equiv \mathsf{skip}$ (hence $s_1 \equiv \mathsf{skip}$ and $s_3 \equiv \mathsf{skip}$), and $\langle s_2 \parallel s_4, M\rangle \xrightarrow{\lambda} \langle (s'), M'\rangle$. From $(\mathsf{skip}; s_2) \parallel (\mathsf{skip}; s_4) \equiv s_2 \parallel s_4$ we get, also by (N9), that $\langle (\mathsf{skip}; s_2) \parallel (\mathsf{skip}; s_4), M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$.
  Clearly $(s', s') \in Id_{\mathbb{S}} \subseteq \mathcal{R}$.

termination

$(s_1 \parallel s_3); (s_2 \parallel s_4) \equiv \mathsf{skip}$ only if $s_i \equiv \mathsf{skip}$ for $1 \leq i \leq 4$.
Then also $(s_1; s_2) \parallel (s_3; s_4) \equiv \mathsf{skip}$.

□

The refinement of Lemma 4.21 is represented graphically by Figure 6 where arrows denote the execution in time of a schedule.

Figure 6: refinements of the Lemma 4.21

The figure on the right hand side corresponds to the schedule $(s_1; s_2) \parallel (s_3; s_4)$. There are two "threads" $s_1; s_2$ and $s_3; s_4$ that can proceed independently of each other. For example, the thread $s_1; s_2$ may terminate while the other thread is still executing s_3. The figure on the left hand side represents the schedule $(s_1 \parallel s_3); (s_2 \parallel s_4)$. In this schedule, the semi-colon forces the two threads to synchronize after termination of s_1 and s_3; i.e. before starting execution of either s_2 or s_4.

Corollary 4.22 *Let s_1, s_2 and $s_3 \in \mathbb{S}$ be schedules, then*

$$\begin{array}{lrcl} 1. & s_1; s_2 & \leqslant & s_1 \parallel s_2 \\ 2. & s_1; (s_2 \parallel s_3) & \leqslant & (s_1; s_2) \parallel s_3 \\ 3. & (s_1 \parallel s_3); s_2 & \leqslant & (s_1; s_2) \parallel s_3 \end{array}$$

Proof

All three follow from Lemma 4.21 by taking one or two terms equal to skip and then using Lemma 4.19.

□

4.4.5 Laws for Conditional Composition

The laws for $c \triangleright s[t]$ follow by propositional calculus and structural congruence.

Lemma 4.23

$$\begin{array}{lrcl} 1. & \mathit{false} \triangleright s[t] & \cong & t \\ 2. & \mathit{true} \triangleright s[t] & \cong & s \\ 3. & c \triangleright \mathsf{skip} & \cong & \mathsf{skip} \\ 4. & (c \triangleright s_1)[t_1]; (c \triangleright s_2)[t_2] & \cong & c \triangleright (s_1; s_2)[t_1; t_2] \\ 5. & (c \triangleright s_1)[t_1] \parallel (c \triangleright s_2)[t_2] & \cong & c \triangleright (s_1 \parallel s_2)[t_1 \parallel t_2] \\ 6. & !(c \triangleright s[t]) & \cong & c \triangleright (!s)[!t] \end{array}$$

In "$c \triangleright r \to s[t]$" the conditional is used to test whether or not the rule "r" needs to be executed. A number of refinement laws apply if some relation between conditions c and b (of rule r) hold. The conditional c and the enabling condition b of r are both formed over the variables $\overline{x}$. In the following laws, we use $c \Rightarrow \neg b$ to mean: for all valuations $\overline{v}$, $c[\overline{x} := \overline{v}] \Rightarrow \neg b[\overline{x} := \overline{v}]$. Lemma 4.24 relates conditional c to a enabling condition b. There, *fail* denotes a rewrite rule that never succeeds (can only make failing transitions). We can think of it as being defined as $fail \mathrel{\widehat{=}} \overline{x} \to m \Leftarrow false$. For any rule r holds $fail \lhd r$, hence *fail* is a lowerbound for the set or multiset rewrite rules ordered by the strengthening relation $\lhd$.

Lemma 4.24 $c \triangleright (fail; s_2)[t] \quad \cong \quad c \triangleright (r \to s_1[s_2])[t] \quad$ *if* $c \Rightarrow \neg b$

Proof

1. If $c = false$, then $c \triangleright (fail; s_2)[t] \equiv t.$ and $c \triangleright (r \to s_1[s_2])[t] \equiv t.$
   By reflexivity $t \cong t$.

   If $c = true$, then $c \triangleright (fail; s_2)[t] \equiv fail; s_2$ and $c \triangleright (r \to s_1[s_2])[t] \equiv r \to s_1[s_2]$.
   The only possible transition for $fail; s_2$ is $\langle fail; s_2, M\rangle \xrightarrow{\varepsilon} \langle s_2, M\rangle$.
   Because $c \Rightarrow \neg b$, the schedule $r \to s_1[s_2]$ can only make the transition $\langle r \to s_1[s_2], M\rangle \xrightarrow{\varepsilon} \langle s_2, M\rangle$. By reflexivity $s_2 \cong s_2$.

□

Corollary 4.25 $fail; t \cong fail \to s[t]$

Proof Follows as a special case from Lemma 4.24 by taking $c = true$. □

Execution of *fail* never changes the input-output behaviour of a schedule (or program). Hence it can always be omitted. However, the notion of refinement does not justify the following law which we desire to remove failing rules

$$\mathsf{skip} \not\leqslant fail$$

Here, the schedule on the left hand side has terminated. In order to constitute a refinement (by Definition 4.4 of simulation) the right hand side should also be terminated. However, the schedule on the right hand side makes an ε-transition before terminating. It seems natural to extend the notion of refinement such that it does not distinguish

between differing numbers of *failing* transition in schedules. Research in this direction will be published in future work [8].

4.4.6 Laws for Replication

In subsequent proofs we will use the notational convention that s^k stands for $k \geq 0$ copies of schedule s composed in parallel. Formally: $s^0 \equiv \mathsf{skip}$, $s^{k+1} \equiv s \parallel s^k$.

Lemma 4.26 *Let $s \in \mathbb{S}$, then $s \leqslant\, !s$.*

Proof
transition

From $\langle s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$ we get by (N6) that $\langle !s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$ and $s' \leqslant s'$ as required.

termination

$s \equiv \mathsf{skip}$ implies, by (E8), $!s \equiv \mathsf{skip}$.

□

Lemma 4.27 *Let $s \in \mathbb{S}$, then $s \parallel !s \leqslant\, !s$*

Proof
transition

From $\langle s \parallel !s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$ we get by (N7) that $\langle !s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$ and $s' \leqslant s'$ as required.

termination

$s \parallel !s \equiv \mathsf{skip}$ only if $s \equiv \mathsf{skip}$, then also $!s \equiv \mathsf{skip}$.

□

Corollary 4.28 justifies the intuition that "$!s$" stands for an arbitrary number of copies of "s" composed in parallel.

Corollary 4.28 *For all $s \in \mathbb{S}$, $k \geq 1 : s^k \leqslant\, !s$*

Proof

By induction on k.

$k = 1$:

$s \leqslant !s$ by Lemma 4.26.

$k > 1$:

$$
\begin{array}{ll}
& s^k \\
\cong & \text{definition of } s^k, \text{ Lemma 4.7} \\
& s \parallel s^{k-1} \\
\leqslant & \text{Induction Hypothesis} \\
& s \parallel !s \\
\leqslant & \text{Lemma 4.27} \\
& !s
\end{array}
$$

□

Corollary 4.29 *For all* $s \in \mathbb{S}$, $k \geq 0 : s^k \parallel !s \leqslant !s$

Proof

By induction on k.

$k = 0$:

From $s^0 \equiv \mathsf{skip}$ follows by Lemma 4.20(1 and 2) that $\mathsf{skip} \parallel !s \cong !s$, hence, by Definition 4.6, $\mathsf{skip} \parallel !s \leqslant !s$.

$k > 0$:

$$
\begin{array}{ll}
& s^k \parallel !s \\
\cong & \text{definition of } s^k, \text{ Lemma 4.7} \\
& (s^{k-1} \parallel s) \parallel !s \\
\cong & \text{Lemma 4.20(3)} \\
& s^{k-1} \parallel (s \parallel !s) \\
\leqslant & \text{Lemma 4.27} \\
& s^{k-1} \parallel !s \\
\leqslant & \text{Induction Hypothesis} \\
& !s
\end{array}
$$

□

An important property of replication is its idempotence. First, we prove the following simpler case.

Lemma 4.30 *Let $s \in \mathbb{S}$, then $!s \parallel !s \leqslant !s$*

Proof

Let $\mathcal{R} = \{(t \parallel (!s \parallel !s), t \parallel !s) \mid s, t \in \mathbb{S}\} \cup Id_{\mathbb{S}}$. We prove that $\mathcal{R}$ is a simulation by induction on the depth of the inference. We will use the following property of $\mathcal{R}$

$$\text{let } (s_1, s_2) \in \mathcal{R} \text{ and } t \in \mathbb{S}, \text{ then } (t \parallel s_1, t \parallel s_2) \in \mathcal{R} \qquad (*)$$

Transition and termination follow directly for the case $(s, t) \in Id_{\mathbb{S}}$. We consider the remaining case.

transition

1. From (N2) by $\langle t, M\rangle \xrightarrow{\lambda} \langle t', M'\rangle$. Then by (N2) also $\langle t \parallel !s, M\rangle \xrightarrow{\lambda} \langle t' \parallel !s, M'\rangle$. Clearly $(t' \parallel (!s \parallel !s), t' \parallel !s) \in \mathcal{R}$.

2. From (N2) by $\langle !s \parallel !s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$. This transition can in turn be derived in five ways. Two of these are symmetric, hence we only need to consider three.

   (a) By (N2) from $\langle !s, M\rangle \xrightarrow{\lambda} \langle s'', M'\rangle$. This can be derived in two ways.

      i. By (N6) from $\langle s, M\rangle \xrightarrow{\lambda} \langle s'', M'\rangle$, hence $s' = s'' \parallel !s$. By (N2) we derive the transition $\langle s \parallel !s, M\rangle \xrightarrow{\lambda} \langle s'' \parallel !s, M'\rangle$. By (N7) we infer $\langle !s, M\rangle \xrightarrow{\lambda} \langle s'' \parallel !s, M'\rangle$. Hence by (N2) $\langle t \parallel !s, M\rangle \xrightarrow{\lambda} \langle t \parallel s'' \parallel !s, M'\rangle$. Clearly $(t \parallel s'' \parallel !s, t \parallel s'' \parallel !s) \in Id_{\mathbb{S}} \subseteq \mathcal{R}$.

      ii. By (N7) from $\langle s \parallel !s, M\rangle \xrightarrow{\lambda} \langle s'', M'\rangle$, hence $s' = s'' \parallel !s$. By (N2) we infer $\langle s \parallel !s \parallel !s, M\rangle \xrightarrow{\lambda} \langle s'' \parallel !s, M'\rangle$. The derivation for this transition is shorter than the derivation of the transition we want to prove the proposition for, hence by induction we get $\langle s \parallel !s, M\rangle \xrightarrow{\lambda} \langle s''', M'\rangle$ such that $(s'' \parallel !s, s''') \in \mathcal{R}$. By (N7) also $\langle !s, M\rangle \xrightarrow{\lambda} \langle s''', M'\rangle$. By (N2) $\langle t \parallel !s, M\rangle \xrightarrow{\lambda} \langle t \parallel s''', M'\rangle$. From $(s'' \parallel !s, s''') \in \mathcal{R}$ follows by (*) that $(t \parallel s'' \parallel !s, t \parallel s''') \in \mathcal{R}$.

   (b) By (N3) from $\langle !s, M\rangle \xrightarrow{\lambda} \langle s'', M'\rangle$ and $\langle !s, M\rangle \xrightarrow{\varepsilon} \langle s''', M\rangle$. If we proceed by induction on the depth of the inference of the former, then the proof proceeds analogously to the previous case (where we use (N3) rather than (N2)).

(c) By (N4) from $\langle !s, M\rangle \xrightarrow{\sigma_1} \langle s_1, M_1\rangle$, and $\langle !s, M\rangle \xrightarrow{\sigma_2} \langle s_2, M_2\rangle$ where $M \models \sigma_1 \bowtie \sigma_2$. This case goes analogously to the previous case (where we use (N4) instead of (N3)).

3. By (N3) from $\langle t, M\rangle \xrightarrow{\lambda} \langle t', M'\rangle$ and $\langle !s \parallel !s, M\rangle \xrightarrow{\varepsilon} \langle s', M\rangle$.
   The proof is analogous to the previous case.

4. By (N3) from $\langle t, M\rangle \xrightarrow{\varepsilon} \langle t', M\rangle$ and $\langle !s \parallel !s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$.
   The proof is analogous to the previous case.

5. By (N4) from $\langle t, M\rangle \xrightarrow{\sigma_1} \langle t', M_1\rangle$ and $\langle !s \parallel !s, M\rangle \xrightarrow{\sigma_2} \langle s', M_2\rangle$
   such that $M \models \sigma_1 \bowtie \sigma_2$. The proof is analogous to the previous case.

termination

$t \parallel !s \parallel !s \equiv \mathsf{skip}$ only if $t \equiv \mathsf{skip}$ and $!s \equiv \mathsf{skip}$, hence $t \parallel !s \equiv \mathsf{skip}$.

□

Corollary 4.31 *Let $s \in \mathbb{S}$, then for all $k : k \geq 1 : (!s)^k \leqslant !s$*

Proof
By induction on k.
$k = 1$:

From reflexivity of $\leqslant$.

$k > 1$:

$$
\begin{array}{lll}
 & (!s)^k & \\
\cong & & \text{Definition } t^k\text{, Lemma 4.7} \\
 & !s \parallel (!s)^{k-1} & \\
\leqslant & & \text{Induction Hypothesis} \\
 & !s \parallel !s & \\
\leqslant & & \text{Lemma 4.30} \\
 & !s &
\end{array}
$$

□

Lemma 4.32 *Let $s \in \mathbb{S}$, then $!(!s) \leqslant\ !s$*

Proof
Let $\mathcal{R} = \{(t \parallel !(!s), t \parallel !s) \mid s, t \in \mathbb{S}\} \cup Id_{\mathbb{S}}$. We show that $\mathcal{R}$ is a simulation up–to– $\leqslant$.
We will use the following property of $\mathcal{R}$:

$$\text{If } (s_1, s_2) \in\ \leqslant\mathcal{R}\leqslant\ \text{ and } t \in \mathbb{S}, \text{ then } (t \parallel s_1, t \parallel s_2) \in\ \leqslant\mathcal{R}\leqslant \quad (*)$$

For any pair $(s, s) \in Id_{\mathbb{S}}$ the conditions for *transition* and *termination* hold by reflexivity of refinement (Lemma 4.1(1)). We consider the remaining case.

transition
By induction on the length of the inference of a transition $\langle t \parallel !(!s), M\rangle \xrightarrow{\lambda} \langle t' \parallel s', M'\rangle$.

1. By (N2) from $\langle t, M\rangle \xrightarrow{\lambda} \langle t', M'\rangle$. Then by (N2) $\langle t \parallel !s, M\rangle \xrightarrow{\lambda} \langle t' \parallel !s, M'\rangle$. Clearly $(t' \parallel !(!s), t' \parallel !s) \in\ \leqslant\mathcal{R}\leqslant$.

2. By (N2) from $\langle !(!s), M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$. This transition can be derived in two ways:

   (a) By (N6) from $\langle !s, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$. Then by (N2) $\langle t \parallel !s, M\rangle \xrightarrow{\lambda} \langle t \parallel s', M'\rangle$. And $(t' \parallel s', t' \parallel s') \in Id_{\mathbb{S}} \subseteq\ \leqslant\mathcal{R}\leqslant$.

   (b) By (N7) from $\langle !s \parallel !(!s), M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$. By the induction hypothesis $\langle !s \parallel !s, M\rangle \xrightarrow{\lambda} \langle s'', M'\rangle$ such that $(s', s'') \in\ \leqslant\mathcal{R}\leqslant$. From Lemma 4.30 follows $\langle !s, M\rangle \xrightarrow{\lambda} \langle s''', M'\rangle$ such that $(s'', s''') \in\ \leqslant$. By transitivity of $\leqslant$ follows that $(s', s''') \in\ \leqslant\mathcal{R}\leqslant$. By (N2) follows $\langle t \parallel !s, M\rangle \xrightarrow{\lambda} \langle t \parallel s''', M'\rangle$. From $(s', s''') \in\ \leqslant\mathcal{R}\leqslant$ and (*) follows $(t \parallel s', t \parallel s''') \in\ \leqslant\mathcal{R}\leqslant$.

3. By (N3) from $\langle t, M\rangle \xrightarrow{\lambda} \langle t', M'\rangle$ and $\langle !(!s), M\rangle \xrightarrow{\varepsilon} \langle s', M\rangle$. From case (2) follows for the latter that $\langle !s, M\rangle \xrightarrow{\varepsilon} \langle s'', M\rangle$ such that $(s', s'') \in\ \leqslant\mathcal{R}\leqslant$. From (N3) then follows $\langle t \parallel !s, M\rangle \xrightarrow{\lambda} \langle t' \parallel s'', M'\rangle$ and by (*) we deduce $(t' \parallel s', t' \parallel s'') \in\ \leqslant\mathcal{R}\leqslant$.

4. By (N3) from $\langle t, M\rangle \xrightarrow{\varepsilon} \langle t', M\rangle$ and $\langle !(!s), M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$.
   The proof is analogous to the previous case.

5. By (N4) from $\langle t, M\rangle \xrightarrow{\sigma_1} \langle t', M_1\rangle$ and $\langle !(!s), M\rangle \xrightarrow{\sigma_2} \langle s', M_2\rangle$ where $M \models \sigma_1 \bowtie \sigma_2$.
   The proof is analogous to the previous case.

termination

$t \parallel !(!s) \equiv \mathsf{skip}$ only if $t \equiv \mathsf{skip}$ and $!(!s) \equiv \mathsf{skip}$. From the latter follows by (E8) that $!s \equiv \mathsf{skip}$, hence $t \parallel !s \equiv \mathsf{skip}$.

□

Lemma 4.33 *Let* $s \in \mathbb{S}$, *then* $!(!s) \cong !s$

Proof

$!s \leqslant !(!s)$ follows from Lemma 4.26,
$!(!s) \leqslant !s$ follows from Lemma 4.32.

□

The next lemma proves a refinement concerning distributivity of replication over parallel composition.

Lemma 4.34 *Let* $s_1, s_2 \in \mathbb{S}$, *then* $!(s_1 \parallel s_2) \leqslant (!s_1) \parallel (!s_2)$

Proof
Let $\mathcal{R} = \{(t \parallel !(s_1 \parallel s_2), t \parallel (!s_1) \parallel (!s_2))\} \cup Id_{\mathbb{S}}$. We show that $\mathcal{R}$ is a simulation up–to– $\leqslant$.

We will use that $\mathcal{R}$ satisfies the following property

$$\text{If } (s_1, s_2) \in \ \leqslant \mathcal{R} \leqslant \ \text{ and } t \in \mathbb{S}, \text{ then } (t \parallel s_1, t \parallel s_2) \in \ \leqslant \mathcal{R} \leqslant \quad (*)$$

For any pair $(s, s) \in Id_{\mathbb{S}}$ the conditions for *transition* and *termination* hold by reflexivity of refinement (Lemma 4.1(1)). We consider the remaining case.

transition
By induction on the depth of the inference.

1. By (N2) from $\langle t, M\rangle \xrightarrow{\lambda} \langle t', M'\rangle$. Then by (N2) $\langle t \parallel !s_1 \parallel !s_2, M\rangle \xrightarrow{\lambda} \langle t' \parallel !s_1 \parallel !s_2, M'\rangle$. Clearly $(t' \parallel !(s_1 \parallel s_2), t' \parallel (!s_1) \parallel (!s_2) \in \ \leqslant \mathcal{R} \leqslant$.

2. By (N2) from $\langle !(s_1 \parallel s_2), M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$. This transition can be derived in 2 ways.

   (a) by (N6) from $\langle s_1 \parallel s_2, M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$. Transitions for $s_1 \parallel s_2$ can be derived in five ways. By symmetry of $s_1 \parallel s_2 \equiv s_2 \parallel s_1$ we only have to consider three cases.

i. By (N2) from $\langle s_1, M\rangle \xrightarrow{\lambda} \langle s_1', M'\rangle$ hence $\langle !(s_1 \parallel s_2), M\rangle \xrightarrow{\lambda} \langle s_1' \parallel s_2, M'\rangle$. By (N6) we infer from the former $\langle !s_1, M\rangle \xrightarrow{\lambda} \langle s_1', M'\rangle$. By (N2) we obtain $\langle t \parallel !s_1 \parallel !s_2, M\rangle \xrightarrow{\lambda} \langle t \parallel s_1' \parallel !s_2, M'\rangle$. Because $s_2 \leqslant\!\!\!\!\cdot\; !s_2$ and $(t \parallel s_1' \parallel s_2, t \parallel s_1' \parallel s_2) \in \mathcal{R}$ we have $(t \parallel s_1' \parallel s_2, t \parallel s_1' \parallel !s_2) \in \;\leqslant\!\!\!\!\cdot\, \mathcal{R} \,\leqslant\!\!\!\!\cdot$.

ii. By (N3) from transitions $\langle s_1, M\rangle \xrightarrow{\lambda} \langle s_1', M'\rangle$ and $\langle s_2, M\rangle \xrightarrow{\varepsilon} \langle s_2', M\rangle$, hence $\langle !(s_1 \parallel s_2), M\rangle \xrightarrow{\lambda} \langle s_1' \parallel s_2', M'\rangle$. By (N6) we infer $\langle !s_1, M\rangle \xrightarrow{\lambda} \langle s_1', M'\rangle$ and $\langle !s_2, M\rangle \xrightarrow{\varepsilon} \langle s_2', M\rangle$. By (N3) we get $\langle !s_1 \parallel !s_2, M\rangle \xrightarrow{\lambda} \langle s_1' \parallel s_2', M'\rangle$. By (N2) we obtain $\langle t \parallel !s_1 \parallel !s_2, M\rangle \xrightarrow{\lambda} \langle t \parallel s_1' \parallel s_2', M'\rangle$. Because $Id_{\mathbb{S}} \subseteq \mathcal{R}$ we have $(t \parallel s_1' \parallel s_2', t \parallel s_1' \parallel s_2') \in \;\leqslant\!\!\!\!\cdot\, \mathcal{R} \,\leqslant\!\!\!\!\cdot$.

iii. By (N4) from $\langle s_1, M\rangle \xrightarrow{\sigma_1} \langle s_1', M_1\rangle$ and $\langle s_2, M\rangle \xrightarrow{\sigma_2} \langle s_2', M_2\rangle$ with $M \models \sigma_1 \bowtie \sigma_2$. The proof proceeds analogously to the preceding case.

(b) by (N7) from $\langle (s_1 \parallel s_2) \parallel !(s_1 \parallel s_2), M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$. By the induction hypothesis we get $\langle (s_1 \parallel s_2) \parallel (!s_1) \parallel (!s_2), M\rangle \xrightarrow{\lambda} \langle s'', M'\rangle$ such that $(s', s'') \in \;\leqslant\!\!\!\!\cdot\, \mathcal{R} \,\leqslant\!\!\!\!\cdot$. By Lemma 4.20(3) this can be equivalently written as $\langle (s_1 \parallel !s_1) \parallel (s_2 \parallel !s_2), M\rangle \xrightarrow{\lambda} \langle s'', M'\rangle$. From Lemma 4.27 and Lemma 4.11 follows $s_1 \parallel !s_1 \parallel s_2 \parallel !s_2 \leqslant\!\!\!\!\cdot\; !s_1 \parallel !s_2$, hence $\langle (!s_1) \parallel (!s_2), M\rangle \xrightarrow{\lambda} \langle s''', M'\rangle$ such that $(s'', s''') \in \;\leqslant\!\!\!\!\cdot$. By (N2) we infer $\langle t \parallel (!s_1) \parallel (!s_2), M\rangle \xrightarrow{\lambda} \langle t \parallel s''', M'\rangle$. From $(s', s'') \in \;\leqslant\!\!\!\!\cdot\, \mathcal{R} \,\leqslant\!\!\!\!\cdot$ and $(s'', s''') \in \;\leqslant\!\!\!\!\cdot$ we get by transitivity of $\leqslant\!\!\!\!\cdot$ that $(s', s''') \in \;\leqslant\!\!\!\!\cdot\, \mathcal{R} \,\leqslant\!\!\!\!\cdot$, hence by (*) follows $(t \parallel s', t \parallel s''') \in \;\leqslant\!\!\!\!\cdot\, \mathcal{R} \,\leqslant\!\!\!\!\cdot$.

3. by (N3) from $\langle t, M\rangle \xrightarrow{\lambda} \langle t', M'\rangle$ and $\langle !(s_1 \parallel s_2), M\rangle \xrightarrow{\varepsilon} \langle s', M\rangle$. The proof is a routine combination of the preceding cases.

4. by (N3) from $\langle t, M\rangle \xrightarrow{\varepsilon} \langle t', M\rangle$ and $\langle !(s_1 \parallel s_2), M\rangle \xrightarrow{\lambda} \langle s', M'\rangle$. The proof is a routine combination of the preceding cases.

5. by (N4) from $\langle t, M\rangle \xrightarrow{\sigma_1} \langle t', M_1\rangle$ and $\langle !(s_1 \parallel s_2), M\rangle \xrightarrow{\sigma_1} \langle s', M_2\rangle$ where $M \models \sigma_1 \bowtie \sigma_2$. The proof is analogous to the preceding case.

termination

$t \parallel !(s_1 \parallel s_2) \equiv \mathsf{skip}$ implies $t \equiv \mathsf{skip}$ and $s_1 \equiv \mathsf{skip}$ and $s_2 \equiv \mathsf{skip}$. Then also $t \parallel (!s_1) \parallel (!s_2) \equiv \mathsf{skip}$.

□

We recall the laws for replication presented in this section

$$\begin{array}{lllll} 1. & s & \leqslant & !s \\ 2. & s \parallel !s & \leqslant & !s \\ 3. & !s \parallel !s & \leqslant & !s \\ 4. & !(s_1 \parallel s_2) & \leqslant & !s_1 \parallel !s_2 \\ 5. & !(!s) & \cong & !s \end{array}$$

We end this section by returning to the refinement of Example 4.3 at the end of Section 4.2. There, simulation was used to prove the validity of the refinement. Here we will use the refinement laws. The example shows that the same refinement can be proven much more concisely using equational reasoning.

Example 4.4 Let r_1 and r_2 be rules, then

$$(r_1; r_2) \parallel (r_2; r_1) \leqslant !(r_1 \parallel r_2)$$

In order to show this consider the following proof

Proof

$$\begin{array}{lll} & (r_1; r_2) \parallel (r_2; r_1) & \\ \leqslant & & s_1; s_1 \leqslant s_1 \parallel s_2 \\ & (r_1 \parallel r_2) \parallel (r_2 \parallel r_1) & \\ \cong & & s_1 \parallel s_2 \cong s_2 \parallel s_1 \\ & (r_1 \parallel r_2) \parallel (r_1 \parallel r_2) & \\ \leqslant & & s \leqslant !s \\ & (r_1 \parallel r_2) \parallel !(r_1 \parallel r_2) & \\ \leqslant & & s \parallel !s \leqslant !s \\ & !(r_1 \parallel r_2) & \end{array}$$

□

4.5 Refinement and Determinism

In [14], Hankin et al. define a refinement ordering on programs by considering the input-output relation they induce: a program P_1 is considered a refinement of P_2 if the set of

possible outcomes of P_1 is contained in the set of possible outcomes of P_2. Using this ordering, the intermediate steps taken by a program to achieve its output are not taken into account.

For the refinement of schedules, as studied in this report, we are interested in the *behaviour* that schedules exhibit in order to compute correct output. Consequently, our notion of refinement considers a schedule s_1 to be a correct implementation of another schedule s_2 if the set of behaviours of s_1 are contained in the set of behaviours of s_2.

In this section we prove that refinement of schedules reduces the nondeterminism of the input-output behaviour and hence respects the (relational) ordering on programs (as used in [14]).

Definition 4.10 *We define the divergence predicate $\uparrow$ on configurations:*
$\langle s, M\rangle\uparrow$ if and only if $\langle s, M\rangle = \langle s_0, M_0\rangle$ and for all $i \geq 0$ there exists a λ_i such that $\langle s_i, M_i\rangle \xrightarrow{\lambda_i} \langle s_{i+1}, M_{i+1}\rangle$.

The capability of a configuration is defined as the set of possible multisets it may produce, plus the special symbol $\perp$ if the configuration may never terminate. The capability function thus associates with any configuration, its *input-ouput* behaviour. Using our notation, the definition of the capability function for Gamma programs from [14] becomes:

Definition 4.11 *Let $\mathbb{P}$ be the set of Gamma programs. The capability function for programs $\mathcal{C} : \mathbb{P} \times \mathbb{M} \to \mathcal{P}(\mathbb{M}) \cup \{\perp\}$ is defined as*

$$\begin{aligned}\mathcal{C}(s, M) = \ & \{\perp \mid \langle P, M\rangle\uparrow\} \cup \\ & \{M' \mid \langle P, M\rangle \overset{\overline{\lambda}}{\rightsquigarrow}{}^{*} \langle P', M'\rangle\surd\}\end{aligned}$$

Definition 4.12 *Analogously, we define the capability function for schedules $\mathcal{C} : \mathbb{S} \times \mathbb{M} \to \mathcal{P}(\mathbb{M}) \cup \{\perp\}$ as*

$$\begin{aligned}\mathcal{C}(s, M) = \ & \{\perp \mid \langle s, M\rangle\uparrow\} \cup \\ & \{M' \mid \langle s, M\rangle \xrightarrow{\overline{\lambda}}{}^{*} \langle \mathsf{skip}, M'\rangle\}\end{aligned}$$

In [9] it was shown how to construct, for any Gamma program P, a schedule Γ_P such

that the behaviours of the schedule Γ_P and the program P correspond. This was used as justification for using schedules as a representation of the behaviour of Gamma programs. This behavioural correspondence entails *input-output* equivalence.

Lemma 4.35 *Let P be a Gamma program and let Γ_P be its most general schedule. Then, for all M, $\mathcal{C}(P, M) = \mathcal{C}(\Gamma_P, M)$.*

Proof
$\mathcal{C}(P, M) \subseteq \mathcal{C}(\Gamma_P, M)$:
Let $x \in \mathcal{C}(P, M)$. If $x = M'$ (for some M'), then P terminates, and the result follows from Theorem 1 in (Section 5 of) [7].
If $x = \perp$, then the result can be proven as follows:
By induction on the structure of program P:

- P is simple:
  First prove that $\langle P, M\rangle \overset{\sigma}{\rightsquigarrow} \langle P, M'\rangle \Rightarrow \langle \gamma, M\rangle \overset{\sigma}{\longrightarrow} \langle \gamma', M'\rangle$ where γ is a Γ_P derived schedule. The case $\langle P, M\rangle \overset{\overline{\lambda}}{\rightsquigarrow}{}^* \langle P, M'\rangle \Rightarrow \langle \gamma, M\rangle \overset{\overline{\lambda}}{\longrightarrow}{}^* \langle \gamma', M'\rangle$ then follows straigthforwardly by induction on the length of the sequence of transitions.

- $P = P_1 \circ P_2$:
  Follows staightforwardly using Theorem 1 from [7] and the previous case.

$\mathcal{C}(\Gamma_P, M) \subseteq \mathcal{C}(P, M)$:
Let $x \in \mathcal{C}(\Gamma_P, M)$. If $x = M'$ (for some M'), then Γ_P terminates, and the result follows from Theorem 2 in (Section 5 of) [7].
If $x = \perp$, then the result follows easily using Lemma 3 of (Section 5 in) [7] and induction on the structure of P. □

Refinement of schedules preserves the relational ordering on $\mathcal{P}(\mathbb{M})$:

Theorem 4.36 *Let s and t be two schedules such that $s \leqslant t$, then, for all M, $\mathcal{C}(s, M) \subseteq \mathcal{C}(t, M)$.*

Proof Let $x \in \mathcal{C}(s, M)$, we have to show that $x \in \mathcal{C}(t, M)$. Consider the following cases:

- $x = \perp$:
  Hence $\langle s, M\rangle = \langle s_0, M_0\rangle$ and for all $i \geq 0$ there exists a λ_i such that

$\langle s_i, M_i\rangle \xrightarrow{\lambda_i} \langle s_{i+1}, M_{i+1}\rangle$. By $s \leqslant t$ follows $\langle t, M\rangle = \langle t_0, M_0\rangle$ and for all $i \geq 0$ there exists a λ_i such that $\langle t_i, M_i\rangle \xrightarrow{\lambda_i} \langle t_{i+1}, M_{i+1}\rangle$. Hence $\perp \in \mathcal{C}(t, M)$.

- $x = M'$:
  Hence $\langle s, M\rangle \xrightarrow{\overline{\lambda}}{}^* \langle \mathsf{skip}, M'\rangle$. By $s \leqslant t$ follows $\langle t, M\rangle \xrightarrow{\overline{\lambda}}{}^* \langle \mathsf{skip}, M'\rangle$, hence $M' \in \mathcal{C}(t, M)$.

□

5 Application: Coordinating a Shortest Paths Program

We demonstrate application of the refinement laws by considering an example problem that is commonly known as the single source shortest paths problem. Pursuing separation between computation and coordination we shall first specify the basic computation that is required to solve the problem in Gamma. After that we shall relate several coordination strategies.

The problem description is as follows. Assume we are given a directed graph $G = (V, E)$ with n vertices $v_1, \ldots, v_n$. A function L associates with every edge $(u, v) \in E$ a non-negative length $L(u, v)$. If there is no edge between vertices u and v, then we take $L(u, v) = \infty$. Also $L(u, u) = 0$ for all vertices u. Given a source vertex $s \in V$, the problem is to determine for every vertex $v \in V$, the length of a shortest path from s to v.

A Gamma program for solving this problem is given by the rule:

$$\mathit{find}(u, v) \mathrel{\widehat{=}} (u, x), (v, y) \rightarrow (u, x), (v, x + L(u, v)) \Leftarrow x + L(u, v) < y$$

The multiset consists of pairs (v, x), where v is a vertex number and x is the length of a path from the source s to v. The initial multiset is given by: $M_0 = \{(s, 0)\} \cup \{\ (v, \infty)\ \mid\ v \in V - \{s\}\ \}$.

Though the program performs the required computation, as can be proven formally using techniques from [4], it is hopelessly inefficient because of its unstructured search through the graph. We may coordinate the program's activities into a coherent (more deterministic) searching strategy by conducting a directed search on the graph starting from the source. From a given vertex u the search proceeds by an attempt to construct a shorter path to every adjacent vertex v (in no preferred order). If the attempt succeeds, the search is continued; otherwise the search at v is aborted. A schedule that expresses this strategy is given by $\mathit{Search}(s)$, where

$$\begin{array}{lcl} \mathit{Search}(u) & \widehat{=} & \mathit{Visit}(1,u) \\ \mathit{Visit}(i,u) & \widehat{=} & (i \leq n) \triangleright (\mathit{find}(u,i) \rightarrow \mathit{Search}(i)) \parallel \mathit{Visit}(i+1,u) \end{array}$$

Note that the schedule still exhibits highly nondeterministic behaviour. The paths in the graph are traversed in any order (possibly in parallel). Using the refinement laws, however, we can transform the schedule into more deterministic versions. To illustrate, we shall derive a depth-first, a breadth-first, and a parallel breadth-first ordering from the schedule *Search*.

5.1 Depth-First Search

Using Corollary 4.22.1 we may replace the parallel composition, which is at the basis of schedule *Search*, by a sequential composition. This results in a strategy where the vertices in the graph are visited in a depth-first order.

$$\begin{array}{lcl} \mathit{DepthFirst}(u) & \widehat{=} & \mathit{DFVisit}(1,u) \\ \mathit{DFVisit}(i,u) & \widehat{=} & (i \leq n) \triangleright (\mathit{find}(u,i) \rightarrow \mathit{DepthFirst}(i)); \mathit{DFVisit}(i+1,u) \end{array}$$

The proof that the depth-first ordering is a correct refinement of the parallel strategy, i.e. $\mathit{DepthFirst}(s) \leqslant \mathit{Search}(s)$, follows immediately using Corollary 4.22.1 and Lemma 4.15.

5.2 Breadth-First Schedule

An alternative to Corollary 4.22.1 to introduce sequential behaviour is presented by Lemma 4.18.1. It appears that repetitive application of this law to the parallel composition of schedule *Search* ultimately yields a breadth-first ordering.

As is standard, the schedule for a breadth-first search maintains a sequence of vertices that are yet to be visited. We write $v \cdot \overline{w}$ to denote the sequence $\overline{w}$ with the element v prepended, and $\overline{w} \cdot u$ for u appended to $\overline{w}$. We use $\langle\,\rangle$ to denote the empty sequence.

A breadth-first ordering can now be expressed by the following recursive schedule definitions:

$$
\begin{array}{lcl}
\mathit{BreadthFirst}(\langle\,\rangle) & \widehat{=} & \mathsf{skip} \\
\mathit{BreadthFirst}(u \cdot \overline{w}) & \widehat{=} & \mathit{BFVisit}(1, u, \overline{w})
\end{array}
$$

$$
\begin{array}{lcl}
\mathit{BFVisit}(i, u, \overline{w}) & \widehat{=} & (i \leq n) \;\triangleright\; \mathit{find}(u, i) \rightarrow \mathit{BFVisit}(i+1, u, \overline{w} \cdot i) \\
 & & \qquad\qquad\qquad\qquad\qquad\qquad [\mathit{BFVisit}(i+1, u, \overline{w})] \\
 & & \qquad\qquad [\mathit{BreadthFirst}(\overline{w})]
\end{array}
$$

The proof that breadth-first search is a correct refinement of the parallel strategy, i.e. $\mathit{BreadthFirst}(\langle\, s\,\rangle) \leqslant \mathit{Search}(s)$, is somewhat more involved than the case of a depth-first ordering, therefore we present it here as a more detailed example of application of the refinement laws.

The proof is largely based on an intermediate result that we present first. We use $\overline{x} \# \overline{y}$ to denote the set of interleavings of sequences $\overline{x}$ and $\overline{y}$. More formally:

Definition 5.1 *Let $\overline{x}$ and $\overline{y}$ be two finite sequences, the set of their interleavings, denoted $\overline{x} \# \overline{y}$, is defined by*

$$
\begin{array}{lcll}
1. & \langle\,\rangle \# \overline{x} & = & \{\overline{x}\} \\
2. & \overline{x} \# \langle\,\rangle & = & \{\overline{x}\} \\
3. & x_1 \cdot \overline{x'} \# y_1 \cdot \overline{y'} & = & \{x_1 \cdot \overline{z} \mid \overline{z} \in \overline{x'} \# (y_1 \cdot \overline{y'})\} \cup \{y_1 \cdot \overline{z} \mid \overline{z} \in (x_1 \cdot \overline{x'}) \# \overline{y'}\}
\end{array}
$$

Associativity of concatenation carries over onto interleaving. From the symmetry of interleaving, we deduce that # is a commutative operator; furthermore # has unit $\langle\,\rangle$. Summarized, the interleaving operator "#" has the following properties:

$$
\begin{array}{lll}
(I0) \quad 1. & \overline{x} \# (\overline{y} \# \overline{z}) = (\overline{x} \# \overline{y}) \# \overline{z} & \text{associativity} \\
\quad\;\; 2. & \overline{x} \# \overline{y} = \overline{y} \# \overline{x} & \text{commutativity}
\end{array}
$$

$(I1)$ If $\overline{w} \in \overline{w_1} \# \overline{w_2}$ and $\overline{w} = \langle\,\rangle$, then $\overline{w_1} = \langle\,\rangle$ and $\overline{w_2} = \langle\,\rangle$.

$(I2)$ If $u \cdot \overline{w} \in \overline{w_1}\#\overline{w_2}$, then $\overline{w_1} = u \cdot \overline{w_1'}$ and $\overline{w} \in \overline{w_1'}\#\overline{w_2}$ or $\overline{w_2} = u \cdot \overline{w_2'}$ and $\overline{w} \in \overline{w_1}\#\overline{w_2'}$.

$(I3)$ If $\overline{w} \in \overline{w_1}\#\overline{w_2}$, then $(\overline{w} \cdot i) \in (\overline{w_1} \cdot i)\#\overline{w_2}$.

Lemma 5.1 *Let $\overline{w}, \overline{w_1}$ and $\overline{w_2}$ be sequences of vertices such that $\overline{w} \in \overline{w_1}\#\overline{w_2}$. Let $n \geq 0$, $1 \leq i \leq n$, and $u \in V$, then $BFVisit(i, u, \overline{w}) \leqslant BFVisit(i, u, \overline{w_1}) \parallel BF(\overline{w_2})$*

Proof

The result follows by showing that $\mathcal{R}$, defined as

$$\begin{aligned} \mathcal{R} \;=\; & \{BFVisit(i, u, \overline{w}), BFVisit(i, u, \overline{w_1}) \parallel BreadhtFirst(\overline{w_2}) \mid \\ & i \geq 0,\ 1 \leq u \leq n,\ w \in \overline{w_1}\#\overline{w_2},\ \overline{w_1}, \overline{w_2} \in V^* \ \} \end{aligned}$$

is a simulation.

transition

We consider the possible transitions of $BFVisit(i, u, \overline{w})$. First consider the case $i \leq n$. There are two possible transitions:

1. Assume, $\langle find(u, i) \rightarrow BFVisit(i+1, u, \overline{w} \cdot i)[BFVisit(i+1, u, \overline{w})], M\rangle \xrightarrow{\sigma} \langle BFVisit(i+1, u, \overline{w} \cdot i), M'\rangle$

   Using (N8) we get

   $$\langle BFVisit(i, u, \overline{w}), M\rangle \xrightarrow{\sigma} \langle BFVisit(i+1, u, \overline{w} \cdot i), M'\rangle$$

   Analogously, we get for $BFVisit(i, u, \overline{w_1})$

   $$\langle BFVisit(i, u, \overline{w_1}), M\rangle \xrightarrow{\sigma} \langle BFVisit(i+1, u, \overline{w_1} \cdot i), M'\rangle$$

   hence, by (N2), for $BFVisit(i, u, \overline{w_1} \cdot i) \parallel BreadhtFirst(\overline{w_2})$:

   $$\langle BFVisit(i, u, \overline{w_1}) \parallel BreadhtFirst(\overline{w_2}), M\rangle \xrightarrow{\sigma} \langle BFVisit(i+1, u, \overline{w_1} \cdot i) \parallel BreadhtFirst(\overline{w_2}), M'\rangle$$

   From $\overline{w} \in \overline{w_1}\#\overline{w_2}$ follows by $(I3)$ that $\overline{w} \cdot i \in (\overline{w_1} \cdot i)\#\overline{w_2}$, hence $(BFVisit(i+1, u, \overline{w} \cdot i), BFVisit(i+1, u, \overline{w_1} \cdot i) \parallel BreadthFirst(\overline{w_2})) \in \mathcal{R}$.

2. The proof for $\langle \mathit{find}(u,i) \to BFVisit(i+1,u,\overline{w}\cdot i)[BFVisit(i+1,u,\overline{w})], M\rangle \xrightarrow{\varepsilon} \langle BFVisit(i+1,u,\overline{w}), M'\rangle$ is analogous to the previous case.

Next, we consider the case $i > n$ and $\overline{w} = u'\cdot\overline{w'}$. Then $BFVisit(i,u,\overline{w}) \cong BreadthFirst(\overline{w})$ and $BreadthFirst(\overline{w}) \cong BFVisit(1,u',\overline{w'})$. From $u'\cdot\overline{w'} \in \overline{w_1}\#\overline{w_2}$ follows by (I2) that $\overline{w_1} = u'\cdot\overline{w_1'}$ or $\overline{w_2} = u'\cdot\overline{w_2'}$.

- $\overline{w_1} = u'\cdot\overline{w_1'}$: Then

$$\begin{array}{ll} & BFVisit(i,u,\overline{w_1}) \parallel BreadthFirst(\overline{w_2}) \\ \simeq & \\ & BreadthFirst(\overline{w_1}) \parallel BreadthFirst(\overline{w_2}) \\ \simeq & \\ & BFVisit(1,u',\overline{w_1'}) \parallel BreadthFirst(\overline{w_2}) \end{array}$$

  The previous case ($i \le n$), then gives
  $(BFVisit(1,u',\overline{w'}), BFVisit(1,u',\overline{w_1'}) \parallel BreadthFirst(\overline{w_2})) \in \mathcal{R}$.

- $\overline{w_2} = u'\cdot\overline{w_2'}$: Then

$$\begin{array}{ll} & BFVisit(i,u,\overline{w_1}) \parallel BreadthFirst(\overline{w_2}) \\ \simeq & \\ & BreadthFirst(\overline{w_1}) \parallel BreadthFirst(\overline{w_2}) \\ \simeq & \\ & BreadthFirst(\overline{w_1}) \parallel BFVisit(1,u_1',\overline{w_2'}) \end{array}$$

  The remainder of the proof is analogous to the previous case.

termination
$BFVisit(i,u,\overline{w}) \equiv \mathsf{skip}$ only if $i > n$ and $\overline{w} = \langle\,\rangle$. Then, from the definition of *BreadthFirst* follows: $BreadthFirst(\overline{w}) \cong \mathsf{skip}$. From $(I1)$ follows $\overline{w_1} = \langle\,\rangle$ and $\overline{w_2} = \langle\,\rangle$, hence by definition of *BreadthFirst* follows $BreadthFirst(\overline{w_1}) \parallel BreadthFirst(\overline{w_2}) \equiv \mathsf{skip}$.
□

Using Lemma 5.1 we reason as follows:

$$
\begin{array}{ll}
 & BFVisit(i, u, \overline{w}) \\
\cong & \\
 & (i \leq n) \ \triangleright \quad find(u, i) \rightarrow \quad BFVisit(i+1, u, \overline{w} \cdot i) \\
 & \qquad\qquad\qquad\qquad\quad [BFVisit(i+1, u, \overline{w})] \\
 & \qquad [BreadthFirst(\overline{w})] \\
\leqslant & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad Lemma\ 5.1 \\
 & (i \leq n) \ \triangleright \quad find(u, i) \rightarrow \quad BFVisit(i+1, u, \overline{w}) \parallel BreadthFirst(\langle\, i \,\rangle) \\
 & \qquad\qquad\qquad\qquad\quad [BFVisit(i+1, u, \overline{w})] \\
 & \qquad [BreadthFirst(\overline{w})] \\
\leqslant & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad Lemmas\ 4.20, \text{ and } 4.18.1 \\
 & (i \leq n) \ \triangleright \quad (find(u, i) \rightarrow \quad BreadthFirst(\langle\, i \,\rangle)) \parallel BFVisit(i+1, u, \overline{w}) \\
 & \qquad [BreadthFirst(\overline{w})]
\end{array}
$$

Finally, we prove the specific case $BreadthFirst(\langle\, s \,\rangle) \leqslant Search(s)$ as follows

$$
\begin{array}{ll}
 & BFVisit(i, u, \langle\, \rangle) \\
\cong & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{def. } BFVisit \\
 & (i \leq n) \ \triangleright \quad (find(s, i) \rightarrow \quad BreadthFirst(\langle\, i \,\rangle)) \parallel BFVisit(i+1, s, \langle\, \rangle) \\
 & \qquad [BreadthFirst(\langle\, \rangle)] \\
\cong & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \text{def. } BreadthFirst,\ Lemma\ 4.20 \\
 & (i \leq n) \ \triangleright \quad (find(s, i) \rightarrow \quad BreadthFirst(\langle\, i \,\rangle)) \parallel BFVisit(i+1, s, \langle\, \rangle)
\end{array}
$$

The latter schedule term can be seen to equal $Visit(1, s)$, by substituting $Search(i)$ for $BreadthFirst(\langle\, i \,\rangle)$ and $Visit(i, u)$ for $BFVisit(i, u, \langle\, \rangle)$. Hence the refinement $BreadthFirst(\langle\, s \,\rangle) \leqslant Search(s)$ follows from the schedule definitions of *BreadthFirst* and *Search* and Lemma 4.15.

5.3 Parallel Breadth-first Search

We conclude the examples with a parallel version of breadth-first search that recursively divides the searching process into two if the amount of work exceeds some predefined threshold $k \geq 1$. A schedule that conducts this kind of search is a variation of the previous schedule and is defined as follows, where the schedule *ParBFVisit* is a renamed version of *BFVisit* from Section 5.2

$$
\begin{array}{lcl}
ParBF(\langle\,\rangle) & \widehat{=} & \mathsf{skip} \\
ParBF(\langle\, v_1, \ldots, v_m \,\rangle) & \widehat{=} & \\
\qquad (m > k) \quad \triangleright & & ParBF(\langle\, v_1, \ldots, v_{m \,\mathtt{div}\, 2} \,\rangle) \parallel ParBF(\langle\, v_{m \,\mathtt{div}\, 2+1}, \ldots, v_m \,\rangle) \\
& & [ParBFVisit(1, v_1, \langle\, v_2, \ldots, v_m \,\rangle)]
\end{array}
$$

$$
\begin{array}{lcl}
ParBFVisit(i, u, \overline{w}) & \widehat{=} & (i \leq n) \;\triangleright\; find(u, i) \rightarrow \quad ParBFVisit(i+1, u, \overline{w} \cdot i) \\
& & \qquad\qquad\qquad\qquad\qquad\quad [ParBFVisit(i+1, u, \overline{w})] \\
& & \qquad\qquad\qquad [ParBF(\overline{w})]
\end{array}
$$

Lemma 5.2 $ParBF(\langle\, s \,\rangle) \leqslant Search(s)$

Proof
Let $\mathcal{R} = \{(\Pi_{j=1}^{m} ParBFVisit(i_j, u_j, \overline{w_j}), \Pi_{j=1}^{m} (\, Visit(i_j, u_j) \parallel \Pi_{k=1}^{|w_j|} Search(w_{j_k}))) \mid m \geq 0\}$.
The result follows by showing that $\mathcal{R}$ is a simulation.
This is a routine proof by induction on m. □

As an alternative to the proof of $BreadthFirst(\langle\, s \,\rangle) \leqslant Search(\langle\, s \,\rangle)$ of Section 5.2, we could use transitivity of $\leqslant$ and combine Lemma 5.2 with a proof of $BreadthFirst(\langle\, s \,\rangle) \leqslant ParBF(\langle\, s \,\rangle)$.

Lemma 5.3 $BreadthFirst(\langle\, s \,\rangle) \leqslant ParBF(\langle\, s \,\rangle)$

Proof
Let $\mathcal{R} = \{(BFVisit(i, u, \overline{w}), ParBFVisit(i, u,) \parallel \Pi_{j=1}^{q} ParBF(\overline{w_j})) \mid i \geq 1, 1 \leq u \leq n,$
$\overline{w} \in \#_{j=1}^{q} w_j\}$. It is straightforward to show that $\mathcal{R}$ is a simulation. □

We now arrive at the refinement ordering

$$BreadthFirst(\langle\, s \,\rangle) \leqslant ParBF(\langle\, s \,\rangle) \leqslant Search(s)$$

6 Related Work

The computational models of action systems [1] and UNITY [6] resemble that of Gamma in that a program, consisting of a set of actions, operates upon a shared storage by nondeterministically selecting actions for execution. This nondeterminism in the selection

of actions makes them in principle suitable for coordination by schedules.

The method for refinement of action systems [2] is based on the *weakest-precondition* calculus and proceeds by transforming an initial sequential program into a parallel one. Refinement of UNITY programs [16] also uses *wp*-based reasoning, but more prominently employs temporal logic. An initial (non-executable) specification is successively refined into a specification that is suitable for execution on a particular architecture.

In the approach we presented here, a Gamma program constitutes an executable specification of the input-output behaviour with high potential for parallelism. The freedom this offers for operational behaviour is subsequently handled by schedules. Hence, in contrast with action systems, we start with parallel behaviour and *decrease* the parallelism.

Furthermore, both action systems and UNITY change their initial program or specification to incorporate more operational detail. Using our approach, operational behaviour is specified using a coordination language, thereby achieving an explicit separation between computation and coordination.

7 Conclusion

Our aim is to develop a design method for (parallel) programs where computation is separated from coordination. The choice for Gamma as a language to specify the basic computations of a program is motivated by its highly nondeterministic and inherently parallel execution model. This enables the programmer to first concentrate on the computational aspects of a given problem. Efficiency issues are addressed in a second phase of the design process, where one or more optimized versions of the program are created using a coordination language. The coordination component is specified separately, leaving the computational part of the program unaffected.

Using such an approach, it is important that we are able to reason about coordination. In this paper we proposed a compositional notion of refinement that can be used to prove that one coordination strategy is a (totally) correct implementation of another. Our refinement relation is an adaptation of strong bisimulation to the shared state model. It

induces a number of basic laws that, as we have illustrated, can be applied in an algebraic way of reasoning about coordination. The present notion of refinement is able to resolve only one type of nondeterminism in Gamma: the selection of rules. Research currently proceeds on a larger refinement relation that also supports the increase of determinacy in the selection of elements from the multiset.

Acknowledgments

I thank Edwin de Jong for many discussions and detailed comments on earlier versions of this report.

References

[1] R.J.R. Back. *Correctness Preserving Program Refinements: Proof Theory and Applications*, volume Mathematical Centre Tracts 131. Mathematical Center, Amsterdam, 1980.

[2] R.J.R. Back. Refinement calculus, part ii: Parallel and reactive programs. In *LNCS 430: Stepwise Refinement of Distributed Systems '89*, pages 67–93. Springer-Verlag, 1989.

[3] J.-P. Banâtre, A. Coutant, and D. Le Métayer. A parallel machine for multiset transformation and its programming style. *Future Generation Computer Systems*, 4:133–144, 1988.

[4] J.-P. Banâtre and D. Le Métayer. The GAMMA model and its discipline of programming. *Science of Computer Programming*, 15:55–77, November 1990.

[5] J.-P. Banâtre and D. Le Métayer. Programming by multiset transformation. *Communications of the ACM*, 36(1):98–111, January 1993.

[6] K.M. Chandy and J. Misra. *Parallel Program Design: A Foundation*. Addison-Wesley, 1988.

[7] M. Chaudron. Schedules for multiset transformer programs. Technical Report 94-36, Rijksuniversiteit Leiden, Departement of Mathematics and Computing Science, P.O. Box 9512, 2300 RA Leiden, The Netherlands, December 1994.

[8] M. Chaudron and E. De Jong. Notions of refinement for a coordination language for gamma. In *Proceedings of the Theory and Formal Methods Workshop 1996.* (to be published).

[9] M. Chaudron and E. De Jong. Schedules for multiset transformer programs. In *Coordination Programming: Mechanisms, Models and Semantics.* Imperial College Press, 1996.

[10] M. Chaudron and E. De Jong. Towards a compositional method for coordinating gamma programs. In *LNCS 1061, Proceedings Coordination '96*, pages 107–123. Springer Verlag, 1996.

[11] P. Ciancarini, R. Gorrieri, and G. Zavattaro. An alternative semantics for the calculus of GAMMA programs. In *Coordination Programming: Mechanisms, Models and Semantics.* Imperial College Press, 1996. (to appear).

[12] B. A. Davey and H. A. Priesty. *Introduction to Lattices and Order.* Cambridge University Press, 1990.

[13] D. Gelernter and N. Carriero. Coordination languages and their significance. *Communications of the ACM*, 35(2):97–107, February 1992.

[14] C. Hankin, D. Le Métayer, and D. Sands. A calculus of GAMMA programs. In *5^{th} International Workshop on Languages and Compilers for Parallel Computing, LNCS 757*, pages 342–355. Springer-Verlag, 1992.

[15] C. Hankin, D. Le Métayer, and D. Sands. A parallel programming style and its algebra of programs. In *PARLE '93, LNCS 694*, pages 367–378. Springer-Verlag, 1993.

[16] Singh A. K. Program refinement in fair transition systems. *Acta Informatica*, 30:503–535, 1993.

[17] R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.

[18] M. J. Quinn. *Parallel Computing: Theory and Practice (2^{nd} ed).* McGraw-Hill, 1994.

[19] D. Sands. A compositional semantics of combining forms for gamma programs. In *LNCS 735, Formal Methods in Programming and Their Applications*, pages 43–56. Springer-Verlag, 1993.

[20] D. Sands. Laws of synchronised termination. In *Proceedings of the First Imperial College Departement of Computing Workshop on Theory and Formal Methods*, pages 276–288. Springer-Verlag, 1993.