A Fixpoint Theory for Non-monotonic Parallelism

Yifeng Chen

Department of Mathematics and Computer Science,
University of Leicester,
University Road, Leicester LE1 7RH, UK

Abstract

This paper studies parallel recursion. The trace specification language used in this paper incorporates sequentiality, nondeterminism, reactiveness (including infinite traces), three forms of parallelism (including conjunctive, fair-interleaving and synchronous parallelism) and general recursion. In order to use Tarski's theorem to determine the fixpoints of recursions, we need to identify a well-founded partial order. Several orders are considered, including a new order called the *lexical order*, which tends to simulate the execution of a recursion in a similar manner as the Egli-Milner order. A theorem of this paper shows that no appropriate order exists for the language. Tarski's theorem alone is not enough to determine the fixpoints of parallel recursions. Instead of using Tarski's theorem directly, we reason about the fixpoints of terminating and nonterminating behaviours separately. Such reasoning is supported by the laws of a new composition called *partition*. We propose a fixpoint technique called the *partitioned fixpoint*, which is the least fixpoint of the nonterminating behaviours after the terminating behaviours reach their greatest fixpoint. The surprising result is that although a recursion may not be lexical-order monotonic, it must have the partitioned fixpoint, which is equal to the least lexical-order fixpoint. Since the partitioned fixpoint is well defined in any complete lattice, the results are applicable to various semantic models. Existing fixpoint techniques simply become special cases of the partitioned fixpoint. For example, an Egli-Milner-monotonic recursion has its least Egli-Milner fixpoint, which can be shown to be the same as the partitioned fixpoint. The new technique is more general than the least Egli-Milner fixpoint in that the partitioned fixpoint can be determined even when a recursion is not Egli-Milner monotonic. Examples of non-monotonic recursions are studied. Their partitioned fixpoints are shown to be consistent with our intuition.

Key words: Recursion, Fixpoint, Reactive programs, Parallelism, Semantics
PACS:

Email address: y.chen@mcs.le.ac.uk (Yifeng Chen).

1 Introduction

Recursion is notoriously tricky to model in denotational semantics. A general recursion is normally written as an equation:

$$X \;=\; f(X) \tag{1}$$

in which X is called the *recursive argument*, and $f(X)$ called the *recursion*. For example $X \;=\; (x := x+1 \,⨾\, X)$ defines a recursion that increases variable x infinitely many times sequentially. If nondeterminism is allowed, equation (1) does not guarantee a unique fixpoint. Among all fixpoints, we must determine the fixpoint that is consistent with our understanding and at the same time convenient to our semantic studies. In this paper, the fixpoint of a recursion $f(X)$ is denoted by $\phi X \cdot f(X)$ or ϕf for short. Loops are special recursions:

$$\texttt{do}\ \ b \rightarrow P\ \ \texttt{od} \quad \widehat{=} \quad \phi X \cdot (\texttt{if}\ \ b\ \ \texttt{then}\ \ (P \,⨾\, X)\ \ \texttt{else}\ \ \mathrm{I\!I})$$

where $\mathrm{I\!I}$ (skip, no operation) is the unit of sequential composition. For example, the simplest recursion $\phi X \cdot X$ corresponds to the empty loop ($\texttt{do true} \rightarrow \mathrm{I\!I}\ \texttt{od}$) that never terminates.

There are three basic styles [30] of fixpoint semantics based on Hoare, Smyth and Plotkin powerdomain constructions respectively. The Hoare powerdomain associated with the *Hoare order* corresponds to a simple semantic style called *partial correctness* in which two programs with the same terminating behaviours are not distinguishable. The Smyth powerdomain associated with the *Smyth order* corresponds to a semantic style called *total correctness* in which possible termination and necessary termination are not distinguishable: a program that may not terminate from any initial state is as bad as a program that never terminates from any state. The Plotkin powerdomain associated with the *Egli-Milner order* corresponds to the most concrete semantic style. We call it *factual correctness* in this paper. Factual correctness reveals 'what actually happens' and distinguishes possible termination from necessary termination. The three semantic styles reflect observation at different abstraction levels. Which one to choose depends on the nature of the programming language and the purpose of semantic modelling.

Dijkstra's original Guarded-Command Language (GCL for short [11]) allows only finite nondeterminism. This restriction reflects computability, but it also limits the use of unboundedly nondeterministic specifications for program refinement [1]. Dijkstra dropped the restriction in his later work [13]. Recursions with unbounded nondeterminism are monotonic but may not be continuous. Tarski's fixpoint theorem [33] provides a standard technique to determine the least fixpoint of a recursion that is monotonic with regard to a well-founded partial order (see Section 2). All recursions must be monotonic with regard to the order, and their least-fixpoint semantics must be consistent with our intuition.

Another restriction of GCL is that a recursion must be a guarded loop. Guardedness simplifies semantics by requiring the recursive argument to appear only in the second argument of any sequential composition. A guarded recursion is hence monotonic with regard to many partial orders. A semantic model without general recursion cannot incorporate procedure calls. Dijkstra studied general recursion in his later paper [12] based on the refinement order. Nelson [27], instead, used the Egli-Milner order, with regard to which unguarded sequential recursions are also monotonic.

GCL uses a healthiness condition to exclude 'miracles'. A miracle is a non-executable specification that allows no behaviour from some initial states. Miracles are found useful for specification purposes including detection of pre-compilation errors and type conflicts [35]. Complete theories of program development have been developed based on semantics with miracles [1]. Our study on global synchrony [8] used miracles for the compositional reasoning about safety and liveness properties. The inclusion of miracles is also essential to the integrity of a semantic space. A semantic space containing miracles is normally a complete lattice (under the refinement order). Complete lattices are simple and rich, and enjoy better properties than domains in general [7,22]. It is hence not surprising that most modern semantic models allow miracles (e.g. [15,22,27]). For example, Nelson dropped the restriction in his generalisation of Dijkstra's calculus [27].

GCL does not allow reactiveness. Reactive processes [2,5,8,10,21,22,29,31,32] are very different from sequential programs. A semantic model allowing infinite reactive behaviours (e.g. [8,10,31,32]) is much trickier than a model without them (e.g. [2,21,22]). If we intend to reason about safety and liveness properties, infinite reactive behaviours are inevitable. The combination of unbounded nondeterminism and infinite behaviours leads to different kinds of nontermination, which must have different semantic denotations. Thus factual correctness, associated with the Egli-Milner order [26,27], becomes more appropriate than the other two styles of semantics and will be the focus of this paper.

Another challenge is involved with loops whose bodies are skip (i.e. the unit of sequential composition) or any other command that does not generate intermediate states. If we intend to unify reactiveness and sequentiality, skip-like state transitions are inevitable. This problem is approached in different ways. For example, ACP [2] does not allow skip but uses a silent event to represent a similar but different concept. Timed CSP [10] allows zero-time transitions but does not guarantee a valid semantics for every recursion. We believe that it is essential to define valid semantics for recursion in general, as was done in domain theory.

An infinite loop whose body takes some minimum time must take infinite time [10], but the infinite loop of a zero-time transition (e.g. skip) is less obvious: should it be zero, nondeterministically arbitrary or infinite? 'Zero time' (known as *Zeno effect* or perhaps better termed as *infinitesimal time*) is needed in the context where the actual amount of time taken by a computation is irrelevant to us as far as it terminates (*cf.* [18]). We argue that the sequential composition of finitely many zero-time transitions should take zero time, while the sequential composition of infinitely many such transitions takes infinite time. This can be intuitively illustrated by the discontinuous cumulation of zero time:

$$0 + 0 + \cdots + 0 = 0$$

$$0 + 0 + \cdots + 0 + \cdots = \infty\,.$$

Tarski's fixpoint theorem [19,33] provides a standard technique to determine the least fixpoint of a monotonic function over a complete lattice (or a well-founded partial order if the function is known to have some fixpoint). If a language's all compositions are monotonic with regard to the Egli-Milner order, Tarski's theorem guarantees that every recursion in the language has its least fixpoint with regard to the order. Most program compositions of sequential programming are monotonic with regard to the order, but the composition of fair choice is not [14].

The final challenge comes from parallelism. Even the simplest forms of parallelism complicate semantic studies tremendously. For example, conjunction as a parallel composition (e.g. in Logs [8]) is not monotonic with regard to the Egli-Milner order. Thus the techniques developed in [15,27] are not applicable in a language with conjunctive parallel composition. There are various other forms of parallelism such as fair-interleaving parallelism [5,21,29] and synchronous parallelism [8,16,34]. None of them is monotonic with regard to the Egli-Milner order. Park [29] also observed that the fair-interleaving merge relation is neither the least fixpoint nor the greatest fixpoint but the combination of the two.

To make use of Tarski's theorem, we need to devise an appropriate partial order. In the past, numerous variants of Hoare, Symth and Egli-Milner orders [3,15] have been proposed. All of them work well in some circumstances but none of them is universally applicable. In fact Nystrom [28] showed that there is no fully abstract (Tarski's) fixpoint semantics for nondeterministic infinite behaviours. Although Nystrom's language, simply generated from a typical context-free syntax, is not a regular programming language, his concept of full abstraction requires distinction between possible nontermination and necessary nontermination and is closely related to factual correctness. In

Section 5.4, we will prove that there is no order (based on factual correctness) such that all recursions (allowing conjunctive parallelism) are monotonic with regard to the order. Thus Tarski's theorem alone is not directly applicable. Non-monotonicity naturally arise from various forms of parallelism in real applications. The monotonicity requirement of Tarski's fixpoint theorem is too restrictive in many circumstances. Confronting non-monotonicity is inevitable if we want to model parallel recursion in general.

Various non-monotonic fixpoint techniques have been developed recently for different theoretical and practical purposes. Instead of assuming monotonicity, these techniques rely on other alternative preconditions. For example, any *contracting* function over a domain equipped with a real measurement has a unique fixpoint according to well-known facts of mathematical analysis [24]. The result, not surprisingly, found applications in computational mathematics. Another fixpoint technique [9,33] assumes a family of monotonic functions. It is then possible to calculate the least fixpoint by randomly choosing a function each step (for example with some fairness or commutativity between functions) to yield an ascending chain that leads to the least fixpoint. Later research [17] assumes a weaker form of monotonicity but requires each function to be *increasing* (i.e. $x \leqslant f(x)$ for any x). Increasingness was also used as a requirement in early fixpoint theories [4] (see [23] for a review of the history). These techniques have been applied to fixpoint calculation in theorem proving and model checking. On the other hand, the fixpoints of some non-monotonic recursions can be determined if we consider terminating and nonterminating behaviours separately: a recursion with the composition of fair choice may not be monotonic with regard to the Egli-Milner order, but it must have a fixpoint, if the recursion is in a normal form [14]. The normal form is specially designed for sequential specifications and hence applicable to only sequential languages. Fixpoints of non-monotonic functions were also discussed in [3]. In order to tackle the non-monotonicity arising from parallel recursions, we will propose a new fixpoint theory that relies on a different precondition.

In this paper we will study a series of languages. We first review relational semantics of sequential language based on factual correctness and explain the intuition behind the Egli-Milner order. We then focus on a language of trace specifications. Trace-based languages have been considered by many people [5,8,29]. Our language consists of five basic compositions: sequential composition, nondeterministic choice, conjunctive parallel composition, fair-interleaving composition and synchronous composition. The language is very similar to Park's language [29], although we study two more forms of parallelism. The techniques will then be applied to a more realistic parallel language that combines pre-post sequential specifications and trace specifications. A similar language *Logic of Global Synchrony* (or LOGS for short [8]) allowing multiple program variables has been successfully applied to specifications of PRAM [16] and BSP [25].

The following contributions are made in this paper:

(1) Tarski's fixpoint theorem is shown not to be directly applicable to our trace language, because the appropriate order based on factual correctness does not exist;
(2) a new fixpoint technique called *partitioned fixpoint* is proposed to determine the fixpoints of the parallel recursions and shown to be the least fixpoint with regard to the lexical order, although the recursions may not be monotonic with regard to the order;
(3) existing major fixpoint theories become special cases of the new technique, which is applicable to a variety of parallelism.

Section 2 reviews Tarski's fixpoint theorem and some well-known results related to the theorem. Section 3 introduces the technique of partitioned fixpoint. Section 4 reviews relational semantics of sequential programming. Section 5 introduces a trace language from which non-monotonicity naturally arises. In Section 6 the technique of the partitioned fixpoint is used to determine the fixpoints of recursions in the trace language and shown to be more general than existing fixpoint techniques. Section 7 applies the technique to a more realistic parallel language that combines sequentiality and reactiveness.

2 Tarski's fixpoint theorem

A standard technique to determine a fixpoint is Tarski's theorem [9,19,33]: a monotonic function f has its least fixpoint in a complete lattice. The least fixpoint is obtained by 'repeatedly' applying the function to the bottom of the lattice:

$$\bot \sqsubseteq f(\bot) \sqsubseteq f^2(\bot) \sqsubseteq \cdots \sqsubseteq f^\omega(\bot) \sqsubseteq f^{\omega+1}(\bot) \sqsubseteq \cdots \sqsubseteq f^\iota(\bot) \sqsubseteq \cdots \tag{2}$$

where ι can be any ordinal and

$$f^\iota(\bot) \;\widehat{=}\; f\left(\bigsqcup\{f^\kappa(\bot) \mid \kappa \text{ is an ordinal}, \kappa \prec \iota\}\right). \tag{3}$$

The above sequence eventually reaches its limit. Since f is monotonic, the fixpoint must be the least fixpoint with regard to the order $\sqsubseteq$ (or 'the $\sqsubseteq$-least fixpoint' for short), which is written $\mu X \cdot f(X)$ (or μf for short) and formally defined by:

$$\mu f \;\widehat{=}\; \bigsqcup\{f^\iota(\bot) \mid \iota \text{ is an ordinal}\}. \tag{4}$$

Note that $\{f^\iota(\bot) \mid \iota \text{ is an ordinal}\}$ is a *subset* of the complete lattice in which each $f^\iota(\bot)$ is an element (by transfinite induction on the definition (3)).

Tarski's theorem reveals that the monotonicity is a sufficient condition for the existence of the least fixpoint.

In a complete lattice, a monotonic function also has its $\sqsubseteq$-greatest fixpoint $\nu X \cdot f(X)$ (or νf for short), which equals the $\sqsupseteq$-least fixpoint. We may consider a fixpoint starting from an arbitrary element A, if $A \sqsubseteq f(A)$. A monotonic function f has its least fixpoint $\mu_A f$ in the sub complete lattice whose top and bottom are $\top$ and A respectively. The case of $A \sqsupseteq f(A)$ can be treated similarly.

The requirement of complete lattice can be relaxed to a *well-founded partial order*. An order is well founded, if any non-empty subset of the order has a greatest lower bound (or *glb* for short). A monotonic function always has its least fixpoint in a well-founded order as long as the function has *some* fixpoint.

Proposition 1 *Let f be a $\sqsubseteq$-monotonic function in a well-founded partial order $(\mathcal{X}, \sqsubseteq)$. If there exists $L \in \mathcal{X}$ such that $f(L) = L$, then f has its $\sqsubseteq$-least fixpoint.*

Proof. For any $L \in \mathcal{X}$ such that $f(L) = L$, we reason by transfinite induction on ordinals:

(1) $\bot = \bigsqcap \mathcal{X} \sqsubseteq L$;

(2) for any ordinal ι, if L is an upper bound of $\mathcal{Y} \mathrel{\hat{=}} \{f^\kappa(\bot) \mid \kappa \prec \iota\}$, then $\bigsqcup \mathcal{Y}$ exists and equals the glb of all upper bounds of $\mathcal{Y}$, and $\bigsqcup \mathcal{Y} \sqsubseteq L$; thus $f^\iota(\bot) = f(\bigsqcup \mathcal{Y}) \sqsubseteq f(L) = L$.

That means, for any ordinal ι, we have $f^\iota(\bot) \sqsubseteq L$. Let $\mathcal{Z}$ be the *set* $\{f^\iota(\bot) \mid \iota \text{ is an ordinal}\}$. Then L must be an upper bound of $\mathcal{Z}$; therefore the lub $\bigsqcup \mathcal{Z}$ exists and equals the glb of all upper bounds of $\mathcal{Z}$, and $\bigsqcup \mathcal{Z} \sqsubseteq L$. Since L is arbitrarily chosen, $\bigsqcup \mathcal{Z}$ must be the $\sqsubseteq$-least fixpoint. $\square$

The following proposition shows that, to determine the least fixpoint of a function with regard to some partial order, it is sufficient to determine the function's least fixpoint with regard to a partial order that is either finer or coarser.

Proposition 2 *Let $(\mathcal{X}, \sqsubseteq_1)$ and $(\mathcal{X}, \sqsubseteq_2)$ be two partial orders of a set $\mathcal{X}$, and f a function on $\mathcal{X}$. Let L_1 and L_2 be the least fixpoints of f with regard to the orders $\sqsubseteq_1$ and $\sqsubseteq_2$ respectively. If $\sqsubseteq_1$ is coarser than $\sqsubseteq_2$ (i.e. $\sqsubseteq_1 \subseteq \sqsubseteq_2$), then $L_1 = L_2$.*

Proof. L_1 is the $\sqsubseteq_1$-least fixpoint. Thus, $L_1 \sqsubseteq_1 L_2$. If $\sqsubseteq_1 \subseteq \sqsubseteq_2$, then we have $L_1 \sqsubseteq_2 L_2$. Since L_2 is the $\sqsubseteq_2$-least fixpoint, we also have $L_2 \sqsubseteq_2 L_1$ and thus $L_1 = L_2$. $\square$

3 Fixpoints of non-monotonic functions

In this section we will introduce a more general fixpoint technique called *partitioned fixpoint*. We will discuss in the context of a general complete lattice $\mathcal{C}$ so that the results can be translated to other styles of semantics easily. Let $\top$ be the top, $\bot$ the bottom, $\sqcap$ the glb, $\sqcup$ the lub, and $\sqsubseteq$ the order.

3.1 Partitions

A partition $(P\ {}_A|_B\ Q)$ is a general composition where $P,\ Q \in \mathcal{C}$ are called the *left-hand* and *right-hand* parts respectively, and $A, B \in \mathcal{C}$, called *partitioning elements*, are complements of each other. Note that we do not require other elements to have complements.

Definition 1 $\quad P\ {}_A|_B\ Q \ \ \widehat{=}\ \ (P \sqcup A) \sqcap (Q \sqcup B)$
where $A \sqcap B = \bot$ *and* $A \sqcup B = \top$.

If each of A and B is the unique complement of the other, we may write one of them as A and the other as $\sim\! A$ (i.e. the complement of A).

Convention 1

$$P\ {}_A|\ Q \ \ \widehat{=}\ \ P\ {}_A|_{\sim A}\ Q$$
$$P\ |_A\ Q \ \ \widehat{=}\ \ P\ {}_{\sim A}|_A\ Q$$

The two conventions are dual of each other. In the rest of this paper, we only use the latter. Two more conventions of $\sqsubseteq$-monotonic functions are used.

Convention 2

$$(P\ |_A\ \cdot\ ) \ \ \widehat{=}\ \ \lambda X \cdot (P\ |_A\ X\ )$$
$$(\ \cdot\ |_A\ Q\ ) \ \ \widehat{=}\ \ \lambda X \cdot (X\ |_A\ Q\ )$$

Partitions satisfy some readily-proved laws. We list only those to be used in the rest of this paper.

Law 1

(1) $P\ |_A\ P \ = \ P$ (2) $P\ |_A\ Q \ = \ Q\ |_{\sim A}\ P$

(3) $P\ |_\bot\ Q \ = \ Q$ (4) $P\ |_\top\ Q \ = \ P$

(5) $(P\ |_A\ R)\ |_A\ Q \ = \ P\ |_A\ Q$ (6) $P\ |_A\ (R\ |_A\ Q\ ) \ = \ P\ |_A\ Q$

(7) $P\ |_A\ \top \ = \ P \sqcup \sim\! A$ (8) $\top\ |_A\ P \ = \ P \sqcup A$

(9) $P\ |_A\ \bot \ = \ P \sqcap A$ (10) $\bot\ |_A\ P \ = \ P \sqcap \sim\! A$

(11) $(P_1\ |_A\ Q_1\ ) \sqcap (P_2\ |_A\ Q_2\ ) \ = \ (P_1 \sqcap P_2)\ |_A\ (Q_1 \sqcap Q_2)$

(12) $(P_1\ |_A\ Q_1\ ) \sqcup (P_2\ |_A\ Q_2\ ) \ = \ (P_1 \sqcup P_2)\ |_A\ (Q_1 \sqcup Q_2)$

Partitions are (partially) ordered by the *lexical order* in which the left-hand parts are given priority.

Definition 2 $\quad P \sqsubseteq_{\lambda(A)} Q \quad$ *iff either*

(1) $P \,|_A\, \top \;\sqsupset\; Q \,|_A\, \top$ *or*
(2) $P \,|_A\, \top \;=\; Q \,|_A\, \top$ *and* $\top \,|_A\, P \;\sqsubseteq\; \top \,|_A\, Q$ *.*

The bottom of this order is A, the *right-most* element, while the top is $\sim\! A$, the *left-most* element. Any $\sqsubseteq_{\lambda(A)}$-monotonic function has its $\sqsubseteq_{\lambda(A)}$-least fixpoint starting from A.

3.2 Partitioned fixpoint

The *partitioned fixpoint* of a function is the $\sqsubseteq$-least fixpoint of the right-hand part after the left-hand part reaches its $\sqsubseteq$-greatest fixpoint.

Definition 3 (Partitioned fixpoint) $\quad \psi_A f \;\;\widehat{=}\;\; \mu X \cdot f(\nu f \,|_A\, X)\,.$

Some previous fixpoints become special cases of the partitioned fixpoint. For example, the $\sqsubseteq$-least fixpoint becomes a special case when $A = \bot$:

$$\psi_\bot f \;=\; \mu X \cdot f(\nu f \,|_\bot\, X) \;=\; \mu X \cdot f(X) \;=\; \mu f\,. \tag{5}$$

When $A = \top$:

$$\psi_\top f \;=\; \mu X \cdot f(\nu f \,|_\top\, X) \;=\; \mu X \cdot f(\nu f) \;=\; f(\nu f) \;=\; \nu f\,. \tag{6}$$

And when $A \sqsubseteq f(A)$ we have:

Theorem 3 *If the function $f : \mathcal{C} \to \mathcal{C}$ is $\sqsubseteq$-monotonic and $A \sqsubseteq f(A)$, then $\psi_A f = \mu_A f$.*

Proof. Firstly, we notice that $A \sqsubseteq \mu_A f \sqsubseteq \nu f$ due to the monotonicity of f and the lub operator on chains. Thus:

$\psi_A f$
$=$ definition
$\mu X \cdot f(\nu f \,|_A\, X)$
$=$ proved fact $\nu f = (\nu f \sqcup A)$ and Law 1(8)
$\mu X \cdot f((\top \,|_A\, \nu f) \,|_A\, X)$
$=$ mid-part elimination Law 1(5)
$\mu X \cdot f(\top \,|_A\, X)$
$=$ Law 1(8)
$\mu X \cdot f(X \sqcup A)$

$=$ $\quad A \sqsubseteq f(A)$ and monotonicity of f

$\bigsqcup \{f^{\iota}(A) \mid \iota \text{ is an ordinal}\}$

$=$ $\quad$ definition

$\mu_A f$.

$\square$

However the partitioned fixpoint $\psi_A f$ is more general and is, surprisingly, well-defined in some cases that A and $f(A)$ are not $\sqsubseteq$-comparable or f is not even $\sqsubseteq_{\lambda(A)}$-monotonic. The following theorem generalises Tarski's fixpoint theorem to non-$\sqsubseteq_{\lambda(A)}$-monotonic functions and will be the key to our modelling of parallel recursion.

Theorem 4 (Partitioned fixpoint) *Let f be a $\sqsubseteq$-monotonic function. Let $\mathcal{C}$ be a complete lattice in which A has its unique complement. If distributiivty $f(P) \mid_A \top = f(P \mid_A \top) \mid_A \top$ holds for any $P \in \mathcal{C}$, then $\psi_A f$ is the $\sqsubseteq_{\lambda(A)}$-least fixpoint of the function f.*

Proof. We first prove that $\psi_A f \mid_A \top$ equals $\nu f \mid_A \top$:

$\psi_A f \mid_A \top$

$=$ $\quad \psi_A f$ is the $\sqsubseteq$-least fixpoint of $f \circ (\nu f \mid_A \cdot)$

$f(\nu f \mid_A \psi_A f) \mid_A \top$

$=$ $\quad$ distributivity of $(\cdot \mid_A \top)$

$f((\nu f \mid_A \psi_A f) \mid_A \top) \mid_A \top$

$=$ $\quad$ mid-part elimination Law 1(5)

$f(\nu f \mid_A \top) \mid_A \top$

$=$ $\quad$ distributivity of $(\cdot \mid_A \top)$

$f(\nu f) \mid_A \top$

$=$ $\quad \nu f$ is the $\sqsubseteq$-greatest fixpoint of f

$\nu f \mid_A \top$.

Thus $\psi_A f$ is a fixpoint of f:

$\psi_A f$

$=$ $\quad \psi_A f$ is the $\sqsubseteq$-least fixpoint of $f \circ (\nu f \mid_A \cdot)$

$f(\nu f \mid_A \psi_A f)$

$=$ $\quad$ inverse of mid-part elimination Law 1(5)

$f((\nu f \mid_A \top) \mid_A \psi_A f)$

$=$ $\quad$ proved fact

$f((\psi_A f \mid_A \top) \mid_A \psi_A f)$

$=$ $\quad$ mid-part elimination Law 1(1) and Law 1(5)

$f(\psi_A f)$.

We now show that $\psi_A f$ is the $\sqsubseteq_{\lambda(A)}$-least fixpoint. Let $L \in \mathcal{C}$ be a fixpoint of f i.e. $L = f(L)$. νf is the $\sqsubseteq$-greatest fixpoint and hence $\nu f \sqsupseteq L$. Thus $(\nu f \mid_A \top) \sqsupseteq (L \mid_A \top)$ due to the monotonicity of $(\cdot \mid_A \top)$.

(1) If $\psi_A f \mid_A \top \sqsupset L \mid_A \top$, then we immediately have $\psi_A f \sqsubseteq_{\lambda(A)} L$.
(2) If $\psi_A f \mid_A \top = L \mid_A \top$, then we have $L \mid_A \top = \nu f \mid_A \top$ according the proved fact and

$$
\begin{aligned}
& L \\
= \quad & \{ L \text{ is a fixpoint of } f \} \\
& f(L) \\
= \quad & \{ \text{Law 1(1) and Law 1(5)} \} \\
& f((L \mid_A \top) \mid_A L) \\
= \quad & \{ \text{proved fact} \} \\
& f((\nu f \mid_A \top) \mid_A L) \\
= \quad & \{ \text{Law 1(5)} \} \\
& f(\nu f \mid_A L)
\end{aligned}
$$

Thus L is a fixpoint of $f \circ (\nu f \mid_A \cdot)$ and hence $\psi_A f \sqsubseteq L$ and $\top \mid_A \psi_A f \sqsubseteq \top \mid_A L$. Therefore, $\psi_A f \sqsubseteq_{\lambda(A)} L$.

Since L is arbitrarily chosen, $\psi_A f$ must be the $\sqsubseteq_{\lambda(A)}$-least fixpoint. $\square$

4 Semantics of sequential programming

In this section, we focus on relational semantics of sequential programming. Readers familiar with predicate-transformer semantics [11] or axiomatic semantics [20] should be able to translate accordingly.

4.1 Determinism

Early models treat a deterministic sequential program as a partial function on states. Let S be a non-empty set of states. A program is represented as a partial function $f : S_\uparrow \to S_\uparrow$ where $S_\uparrow$ is a flat domain containing elements of S and the 'undefinedness' symbol $\uparrow$ for nontermination. The order of the domain is defined by $x \preccurlyeq y \ \widehat{=}\ (x = y) \vee (x = \uparrow)$ (see Figure 1).

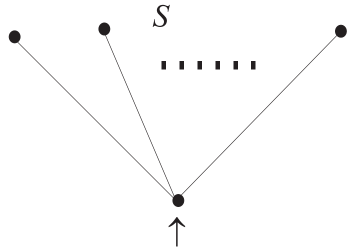

Fig. 1. The flat domain $(S_\uparrow, \preccurlyeq)$

A program that loops forever from some initial state is represented as a function mapping the initial state to $\uparrow$. A valid partial function f must be *strict*, i.e. $f(\uparrow) = \uparrow$. The sequential composition of two functions f and g is simply their functional composition: $f \fatsemi g \mathrel{\widehat{=}} g \circ f$. For example, the function $f(x) = x$ (for any $x \in S_\uparrow$) denotes the command skip (i.e. the unit of sequential composition).

4.2 Nondeterminism

A sequential specification allowing nondeterminism is a binary relation between the initial and final states (or equivalently, a function mapping each initial state to a set of final states). We also use the symbol $\uparrow$ to denote nontermination. A pair $(x, y) \in S_\uparrow \times S_\uparrow$ of states is called a *terminating behaviour* if $y \neq \uparrow$; otherwise, it is called a *nonterminating behaviour*. A sequential specification is a set of behaviours.

There are three basic styles of sequential semantics. Consider the following three programs [30] where the state space S is the set of natural numbers, x is the unique variable, $x := 1$ is an assignment statement, $\cup$ is nondeterministic choice and $\phi X \cdot X$ is the empty loop:

(1) $x := 1$
(2) $x := 1 \ \cup\ \phi X \cdot X$
(3) $\phi X \cdot X$.

The first program always terminates, the second one sometimes terminates sometimes does not, and the third one never terminates. Partial correctness concerns only the terminating behaviours of specifications and thus does not distinguish (1) and (2); total correctness treats all nontermination the same and thus does not distinguish possible nontermination (2) and necessary nontermination (3); factual correctness reveals 'what actually happens' and distinguishes all three programs.

Partial, total and factual-correctness semantics correspond to the Hoare, Smyth and Egli-Milner preorders (of subsets of $S_\uparrow$) respectively. In this paper, we focus on the Egli-Milner order: for any subsets $X, Y \subseteq S_\uparrow$,

$$X \sqsubseteq_{\mathtt{EM}} Y \ \mathrel{\widehat{=}}\ (\forall x \in X\ \exists y \in Y \cdot x \preccurlyeq y) \wedge (\forall y \in Y\ \exists x \in X \cdot x \preccurlyeq y)\ .$$

Because of the bipartite structure of $S_\uparrow$, the Egli-Milner order $\sqsubseteq_{\mathtt{EM}}$ can be reduced to the following well-founded partial order:

$$X \sqsubseteq_\varepsilon Y \ \mathrel{\widehat{=}}\ (\uparrow \notin X \wedge X = Y) \vee (\uparrow \in X \wedge X \backslash \{\uparrow\} \subseteq Y \backslash \{\uparrow\})\ . \quad (7)$$

Note that the original Egli-Milner order $\sqsubseteq_{\mathtt{EM}}$ and the order $\sqsubseteq_{\varepsilon}$ agree on only *non-empty* subsets of $S_{\uparrow}$. For a counterexample, we have $\{\uparrow\} \sqsubseteq_{\varepsilon} \emptyset$ but not $\{\uparrow\} \sqsubseteq_{\mathtt{EM}} \emptyset$.

A miracle is an infeasible specification with empty range of final states from some initial state. For example, the extreme miracle produces the empty set $\emptyset$ of final states from every initial state of S. Since we are committed to support program refinement, miracles must be allowed. The order $\sqsubseteq_{\varepsilon}$ is more convenient for a semantics including miracles.

The use of the Egli-Milner order can be justified by the fact that it simulates the execution of a loop (see Figure 2). For example, the following program may terminate or may never terminate [30]:

$$x, y := 0, 0 \,\mathbin{;}$$
$$\mathtt{do}\ \ y = 0 \ \rightarrow\ (x := x + 1 \ \cup\ y := 1)\ \ \mathtt{od}\,.$$

The calculation of its least Egli-Milner fixpoint starts from pure nontermination $\{\uparrow\}$. As the approximation process proceeds, the nonterminating behaviour step-by-step turns into terminating and/or nonterminating behaviours (see Figure 2).

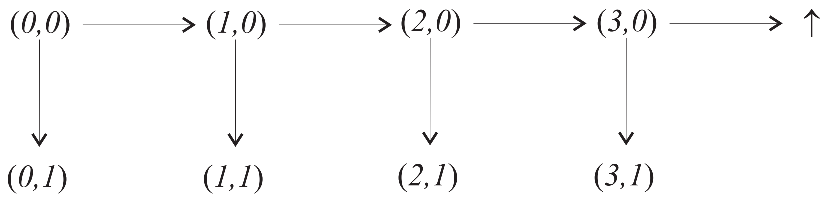

Fig. 2. A snapshot of loop approximation where each (x, y) is a state

Note that Figure 2 is just an intuitive illustration. A general recursion may have a much more complicated structure.

4.3 Sequential semantics based on factual correctness

Table 1 lists the basic sequential specifications based on factual correctness. A specification Q is considered 'better' or 'more refined' than another specification P, if Q is more deterministic. Such *refinement order* is simply modelled as relational containment: $P \supseteq Q$ under which the space of specifications (including miracles) forms a complete lattice whose top and bottom are $\top$ and $\bot$ respectively. Program development is a series of transformations starting from an abstract specification with potentially unbounded nondeterminism and ending with a deterministic and executable program [1]. The extreme specifications $\rhd$ and $\lhd$ satisfy $\rhd \cap \lhd = \top$ and $\rhd \cup \lhd = \bot$. Sequential composition is standard relational composition. The behaviour $(\uparrow, \uparrow)$ belongs to every specification so that nontermination of the first argument of a sequential composition leads to nontermination of the whole composition. The nondeterministic assignment $x :\in E$ changes the state by choosing a state from the set $E(x)$ where E is an expression. Both sequential composition and nondeterministic choice are monotonic with regard to the refinement order $\supseteq$. Each recursion $f(X)$ in the command ϕf is (finitely) constructed from the commands of this table.

Table 1
Sequential specifications of factual correctness

$\top$	$\widehat{=}\ \{(\uparrow,\uparrow)\}$	magic (singleton behaviour of $(\uparrow,\uparrow)$)
$\bot$	$\widehat{=}\ S_\uparrow \times S_\uparrow$	chaos (all behaviours)
$\rhd$	$\widehat{=}\ (S \times S) \cup \top$	termination (all terminating behaviours)
$\lhd$	$\widehat{=}\ S_\uparrow \times \{\uparrow\}$	nontermination (all nonterminating behaviours)
$x :\in E$	$\widehat{=}\ \{(x, y) \mid x \in S,\ y \in E(x)\} \cup \top$	nondeterministic assignment
$P \,;\, Q$	$\widehat{=}\ \{(x, y) \mid \exists z \cdot x\ P\ z \wedge z\ Q\ y\}$	sequential composition
$P \cup Q$		nondeterministic choice (set union)
ϕf		recursion

To determine the fixpoint of a recursion ϕf, we need to lift the Egli-Milner order (7) to the level of specifications. Two specifications $P, Q \subseteq S_\uparrow \times S_\uparrow$ are ordered by the Egli-Milner order $P \sqsubseteq_\varepsilon Q$ if P has no less nonterminating behaviours than Q, Q has no less terminating behaviours than P, and any additional terminating behaviour of $Q \setminus P$ must have a corresponding nonterminating behaviour in P from the same initial state:

$$\begin{aligned} P \sqsubseteq_\varepsilon Q \ \widehat{=}\ \ & (P \cap \rhd) \ \subseteq\ (Q \cap \rhd) \\ & \wedge\ (P \cap \lhd) \ \supseteq\ (Q \cap \lhd) \\ & \wedge\ ((Q \setminus P) \cap \rhd) \,;\, \lhd \ \subseteq\ (P \cap \lhd)\ . \end{aligned} \tag{8}$$

Two specifications are ordered by (8) if and only if, from any initial state, the sets of final states of the two specifications are ordered by (7). The correspondence between the orders (7) and (8) can be established by routine manipulation of the definitions. An order similar to (8) first appeared in [15].

The space of specifications is a well-founded partial order under the Egli-Milner order (8) with regard to which both sequential composition and nondeterministic choice are monotonic. Since any sequential recursion is $\supseteq$-monotonic and at least has the $\supseteq$-least fixpoint, it must have the least $\sqsubseteq_\varepsilon$-fixpoint according to Proposition 1. For example, the least $\sqsubseteq_\varepsilon$-fixpoint of the empty loop

$\phi X \cdot X$ is nontermination $\lessdot$, the bottom of the order $\sqsubseteq_\varepsilon$.

Program refinement and fixpoint calculation are separate issues. They may rely on different partial orders. A semantics based on the Egli-Milner order can also support program refinement.

Other useful commands can be derived from the basic ones (see Table 2).

Table 2
Derived commands

$x := e$	$\widehat{=}$	$x :\in \{e\}$	deterministic assignment
II	$\widehat{=}$	$x := x$	skip, no operation
$(b)_\top$	$\widehat{=}$	$x :\in \{x \mid b(x)\}$	if b then skip else magic
$P \lhd b \rhd Q$	$\widehat{=}$	$((b)_\top \mathbin{;} P) \cup ((\neg b)_\top \mathbin{;} Q)$	if b then P else Q
$\texttt{do}\ \ b \to P\ \ \texttt{od}$	$\widehat{=}$	$\phi X \cdot ((P \mathbin{;} X) \lhd b \rhd \mathrm{II})$	do loop

5 A parallel specification language of reactive processes

In this section we introduce a parallel specification language for reactive processes. A reactive process can be represented as a trace of observable intermediate states [31]. Since we intend to reason about safety and liveness properties, ω-infinite traces are allowed.

5.1 Notations for traces

Let S be a non-empty set of states. Table 3 lists our notations for traces. The first nine notations are standard. The bipartite partial order $\preceq$ will later be used to define the Egli-Milner order at the specification level. Trace interleaving $s \interleave t$ is the set of all fair interleavings of the two traces s and t. All elements of s and t must appear in every fair-interleaving of them. For example, the fair interleaving of the traces $\langle 0, 0, \cdots \rangle$ and $\langle 1, 1, \cdots \rangle$ is the set of all traces containing ω-infinitely-many 0s and 1s. Interestingly, for any ω-infinite trace s, the trace $\langle 0, 1, 2, \cdots \rangle$ is not a member of the set $\langle 0, 1, 2, \cdots \rangle \interleave s$. The synchronous merge $s \bowtie_+ t$ combines the two traces s and t by adding the corresponding elements with the given operation $+ : S \times S \to S$. The synchronous merge is defined recursively:

$$\begin{aligned} \langle\,\rangle \bowtie_+ s &= s \\ s \bowtie_+ \langle\,\rangle &= s \\ \langle x\rangle^\frown s \ \bowtie_+ \ \langle y\rangle^\frown t &= \langle(x+y)\rangle^\frown(s \bowtie_+ t)\,. \end{aligned}$$

For example, let S be $\{\mathtt{true},\mathtt{false}\}$ and $+$ be the logical disjunction $\vee$. Then $\langle\mathtt{false},\mathtt{true}\rangle \bowtie_+ \langle\mathtt{true}\rangle = \langle\mathtt{true},\mathtt{true}\rangle$.

Table 3
Trace notations

$\langle\,\rangle$	empty trace
$\lvert s\rvert$	length of trace (e.g. $\lvert\langle\,\rangle\rvert = 0$ and $\lvert\langle x, x, \cdots\rangle\rvert = \omega$)
s_k	(k+1)-th element of s ($0 \leqslant k < \lvert s\rvert$)
S^*	set of all finite traces
S^ω	set of all ω-infinite traces
$S^\dagger$	set of all traces ($S^* \cup S^\omega$)
$s^\frown t$	trace concatenation ($s^\frown t = s$ if s is ω-infinite)
$s \leqslant t$	prefix partial order (s is a prefix of t)
$t - s$	trace difference ($s^\frown(t-s) = t$ if $s \leqslant t$)
$s \preceq t$	bipartite partial order $(s = t) \vee (s \leqslant t \wedge t \in S^\omega)$
$s \mathbin{\vert\vert\vert} t$	set of all fair interleavings
$s \bowtie_+ t$	synchronous merge

5.2 Trace specifications

A trace specification is a set of behaviours, each of which is a potentially infinite trace. For example, the finite trace $\langle 0, 1\rangle$ is a behaviour, while the singleton set $\{\langle 0, 1\rangle\}$ is a specification.

Table 4 lists the commands of trace specifications. Sequential composition is the pointwise concatenation of traces. There are three forms of parallelism. The specifications in a conjunctive parallel composition must agree on all observables. Inconsistent inference becomes magic. For example $\widetilde{1} \cap \rhd = \widetilde{1}$ but $\widetilde{1} \cap \lessdot = \top$. Fair interleaving of two specifications is the arbitrary pointwise fair interleaving of traces. For example the specification $\widetilde{1} \ \mathbin{\vert\vert\vert} \ \lessdot$ is the set of all ω-infinite traces containing at least one 1. Synchronous composition, the pointwise synchronous merge of traces corresponds to an abstract form of *Bulk-Synchronous Parallelism* [34]. Note that, without further notification, we assume the state space S to be the set of natural numbers and the operation

$+$ to be the arithmetic addition. For example, the specification $\widetilde{1} \parallel_+ \lhd$ is the set of all ω-infinite traces whose first element is no less than 1. The language provides us the context in which we can study parallel recursion in general.

Table 4
Trace specifications

II	$\widehat{=}$	$\{\langle\rangle\}$	singleton set of empty trace
$\widetilde{x}$	$\widehat{=}$	$\{\langle x\rangle\}$	singleton set of 1-step trace ($x \in S$)
$\widetilde{x}^\omega$	$\widehat{=}$	$\{\langle x, x, \cdots\rangle\}$	singleton set of ω-infinite trace
$\rhd$	$\widehat{=}$	S^*	termination (all finite traces)
$\lhd$	$\widehat{=}$	S^ω	nontermination (all ω-infinite traces)
$\top$	$\widehat{=}$	$\emptyset$	magic (no trace)
$\bot$	$\widehat{=}$	$S^\dagger$	chaos (all traces)
$P \mathbin{;} Q$	$\widehat{=}$	$\{s ^\frown t \mid s \in P,\, t \in Q\}$	sequential composition
$P \cup Q$			nondeterministic choice (set union)
$P \cap Q$			conjunctive parallelism (set intersection)
$P \ \vert\vert\vert \ Q$	$\widehat{=}$	$\{u \in (s \mathbin{\vert\vert\vert} t) \mid s \in P,\, t \in Q\}$	fair interleaving parallelism
$P \parallel_+ Q$	$\widehat{=}$	$\{s \bowtie_+ t \mid s \in P,\, t \in Q\}$	synchronous parallelism
ϕf			recursion

The space of specifications forms a complete lattice under the refinement order $\supseteq$ with regard to which all compositions are monotonic. The top and bottom of the complete lattice are $\top$ and $\bot$ respectively. The extreme specifications $\rhd$ and $\lhd$ are unique complement of each other: $\rhd \cap \lhd = \top$ and $\rhd \cup \lhd = \bot$.

We also use an important composition called the *partition*. A partition $P \mid Q$ is the combination of P's terminating behaviours and Q's nonterminating behaviours: $P \mid Q \ \widehat{=}\ (P \cap \rhd) \cup (Q \cap \lhd)$. It is a special case of Definition 1 where the partitioning elements are $\rhd$ and $\lhd$ respectively. Partitions are highly useful to our reasoning. For example, $P \mid \top$ and $\top \mid P$ extract the terminating and nonterminating behaviours from P respectively. Partitions satisfy some simple laws in which Laws 2(7) and 2(8) are called *mid-part elimination laws*. These laws may render reasoning simple and elegant.

Law 2 (1) $P \mid \top \ = \ P \cap \rhd$ (2) $\top \mid P \ = \ P \cap \lhd$

(3) $P \mid \bot \ = \ P \cup \lhd$ (4) $\bot \mid P \ = \ P \cup \rhd$

(5) $P \ = \ (P \mid P)$ (6) $P \ = \ (P \mid \top) \mid (\top \mid P)$

(7) $(P \mid R) \mid Q \ = \ P \mid Q$ (8) $P \mid (R \mid Q) \ = \ P \mid Q$

With partitions, we can reason about terminating and nonterminating behaviours separately. Law 3 lists the partitioned representation of each composition. For example, a sequential composition $P \,;\, Q$ terminates if and only if both P and Q terminate. That means the terminating behaviours of the sequential composition are 'pure' and related to only the terminating behaviours of P and Q. On the other hand, $P \,;\, Q$ never terminates if and only if either P or Q never terminates. The nonterminating behaviours of $P \,;\, Q$ are 'mixed' with terminating and nonterminating behaviours of P and Q. However, the nonterminating behaviours will no longer be 'mixed', if the terminating part is constant or reaches a fixpoint. This motivated us to propose the *partitioned fixpoint* in Section 3.

Law 3

$$
\begin{aligned}
(1)\ P \cup Q &= (P|\top \cup Q|\top) \mid (\top|P \cup \top|Q)\\
(2)\ P \cap Q &= (P|\top \cap Q|\top) \mid (\top|P \cap \top|Q)\\
(3)\ P \,;\, Q &= (P|\top \,;\, Q|\top) \mid ((\top|P \,;\, Q) \cup (P \,;\, \top|Q))\\
(4)\ P \parallel\!| Q &= (P|\top \parallel\!| Q|\top) \mid ((\top|P \parallel\!| Q) \cup (P \parallel\!| \top|Q))\\
(5)\ P \parallel_+ Q &= (P|\top \parallel_+ Q|\top) \mid ((\top|P \parallel_+ Q) \cup (P \parallel_+ \top|Q))
\end{aligned}
$$

5.3 The Egli-Milner order of trace specifications

A semantics allowing infinite traces is no doubt much more complicated than traditional sequential semantics. The main reason is that the latter has a unique nonterminating behaviour $\uparrow$, while the former may have uncountably-many different infinite behaviours.

Partial correctness is not suitable for our trace language in that it does not take infinite behaviours into account. For total correctness, the denotation of nontermination is too simplistic: if a specification does not terminate, its behaviours must be chaotic. Although this view reflects a certain level of abstraction, it is not concrete enough for a trace semantics allowing various infinite behaviours. We believe that factual correctness, describing 'what actually happens', is more appropriate.

The challenge is to devise an appropriate well-founded partial order based on factual correctness so that we could use Tarski's theorem. Before doing so, we first restructure the partial order $\preccurlyeq$ by replacing the elements of S with finite traces of S^* (denoted by $\rhd$) and replacing the nonterminating behaviour $\uparrow$ with infinite traces of S^ω (denoted by $\lhd$). This leads to a bipartite partial order between the finite traces and the infinite ones. We then need to determine the order between finite and infinite traces. The Egli-Milner order is meant to simulate the approximation process in which an infinite behaviour turns into only those finite behaviours that are its prefixes. Thus each finite trace dominates only its infinite extensions. The corresponding bipartite partial order $\preceq$ of traces (refer to Table 3) is illustrated in Figure 3.

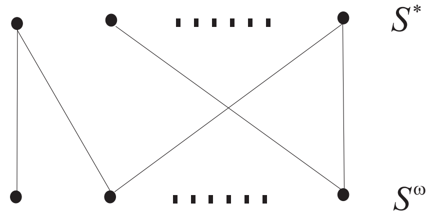

Fig. 3. Bipartite partial order $(S^\dagger, \preceq)$

A variant form of Egli-Milner order can be derived from the bipartite partial order $\preceq$. The new order happens to have the same definition as the order (8), although $\rhd$ and $\lhd$ and $⨟$ have different semantics now. The variant Egli-Milner order between any two specifications $P, Q \subseteq S^\dagger$ can be re-defined using partitions:

$$\begin{aligned} P \sqsubseteq_\varepsilon Q \;\; \widehat{=} \;\; & (P|\top \subseteq Q|\top) \wedge (\top|P \supseteq \top|Q) \\ & \wedge ((Q \setminus P)|\top ⨟ \lhd \subseteq \top|P) \end{aligned} \tag{9}$$

where Q contains no less finite traces than P, P contains no less infinite traces than Q, and all infinite extensions of any additional finite trace in $Q \setminus P$ must be also in P. The first two conjuncts reflect the approximation process in which infinite traces gradually turn into finite ones (*cf.* Figure 2). The last conjunct is more subtle. It reflects the requirement that an infinite trace turns into only its finite prefixes. This also guarantees the monotonicity of sequential composition. We are committed to support various forms of parallelism, but only sequential composition and nondeterministic choice are monotonic with regard to the order. All three parallel compositions are not monotonic with regard to the order. Some counterexamples have been found (see Table 5).

Table 5
Nonmonotonicty of the Egli-Milner order

	$X_1 \sqsubseteq_\varepsilon X_2$	$f(X) =$	$f(X_1) \not\sqsubseteq_\varepsilon f(X_2)$
$\cap$	$\lhd \sqsubseteq_\varepsilon \widetilde{1}$	$X \cap (\widetilde{1} \cup \widetilde{0}^\omega)$	$\widetilde{0}^\omega \not\sqsubseteq_\varepsilon \widetilde{1}$
$\interleave$	$\lhd \sqsubseteq_\varepsilon \mathrm{II}$	$X \interleave \{\langle 0,1,2,\cdots\rangle\}$	$\lhd \interleave \{\langle 0,1,2,\cdots\rangle\} \not\sqsubseteq_\varepsilon \{\langle 0,1,2,\cdots\rangle\}$
$\Vert_+$	$\lhd \sqsubseteq_\varepsilon \mathrm{II}$	$X \Vert_+ \widetilde{1}$	$\lhd \Vert_+ \widetilde{1} \not\sqsubseteq_\varepsilon \widetilde{1}$ *

* We assume S to be the real interval $(0, \infty)$ and $+$ to be the addition operation in this example.

For comparison, we also study an order called the *pairwise order*, which is the conjunction of the first two conjuncts of the variant Egli-Milner order (9):

$$P \sqsubseteq_\pi Q \ \ \widehat{=}\ \ (P|\top\ \subseteq\ Q|\top) \wedge (\top|P\ \supseteq\ \top|Q)\,.$$

The pairwise order is finer than the variant Egli-Milner order (i.e. $\sqsubseteq_\varepsilon\ \subseteq\ \sqsubseteq_\pi$). The space of specifications forms a complete lattice under the pairwise order. The top and bottom are $\rhd$ and $\lhd$ respectively. Nondeterministic choice and conjunctive parallel composition are monotonic with regard to the pairwise order, but sequential, fair-interleaving and synchronous compositions are not (see Table 6 for counterexamples).

Table 6
Nonmonotonicty of the pairwise order

	$X_1\ \sqsubseteq_\pi\ X_2$	$f(X) =$	$f(X_1)\ \not\sqsubseteq_\pi\ f(X_2)$
$\fatsemi$	$\widetilde{0}^\omega\ \sqsubseteq_\pi\ \widetilde{1}$	$X \fatsemi \widetilde{0}^\omega$	$\widetilde{0}^\omega\ \not\sqsubseteq_\pi\ \widetilde{1} \fatsemi \widetilde{0}^\omega$
$\vert\vert\vert$	$\widetilde{0}^\omega\ \sqsubseteq_\pi\ \widetilde{1}$	$X \vert\vert\vert\, \widetilde{0}^\omega$	$\widetilde{0}^\omega\ \not\sqsubseteq_\pi\ \widetilde{1} \vert\vert\vert\, \widetilde{0}^\omega$
$\Vert_+$	$\widetilde{0}^\omega\ \sqsubseteq_\pi\ \widetilde{1}$	$X \Vert_+ \widetilde{0}^\omega$	$\widetilde{0}^\omega\ \not\sqsubseteq_\pi\ \widetilde{1} \Vert_+ \widetilde{0}^\omega$

We may consider another partial order $\sqsubseteq_\lambda$ called the *lexical order*:

$$P \sqsubseteq_\lambda Q \ \ \widehat{=}\ \ (P|\top \subset Q|\top) \vee (P|\top = Q|\top\ \wedge\ \top|P \supseteq \top|Q)\,.$$

This definition is a special case of Definition 1 when the parameter A equals nontermination $\lhd$. The lexical order is even finer than the pairwise order in that the infinite parts of P and Q do not need to be ordered if their finite parts are strictly ordered. Similar to the Egli-Milner order, the lexical order also simulates the execution of recursions, although it emphasises more on the approximation of the finite behaviours. The space of specifications form a complete lattice under the lexical order. The top and bottom are $\rhd$ and $\lhd$ respectively. The lexical order is the main order that we will investigate in this paper. According to Proposition 2, if we are able to determine the least fixpoint with regard to the lexical order, we will also be able to determine the least fixpoints with regard to the pairwise order and the Egli-Milner order (if such least fixpoints do exist). Unfortunately, none of the compositions is monotonic with regard to the lexical order (see Table 7 for counterexamples). Such non-monotonicity has rendered the lexical order seemingly useless.

5.4 Nonexistence of appropriate order for factual correctness

We have explained why factual correctness is more appropriate to trace semantics than total correctness. We must point out that the variant Egli-Milner

Table 7
Nonmonotonicty of the lexical order

	$X_1 \sqsubseteq_\lambda X_2$	$f(X) =$	$f(X_1) \not\sqsubseteq_\lambda f(X_2)$
$⨾$	$\widetilde{0}^\omega \sqsubseteq_\lambda \widetilde{1}$	$X ⨾ \widetilde{0}^\omega$	$\widetilde{0}^\omega \not\sqsubseteq_\lambda \widetilde{1} ⨾ \widetilde{0}^\omega$
$\cup$	$\widetilde{0}^\omega \sqsubseteq_\lambda \widetilde{1} \cup \widetilde{1}^\omega$	$X \cup \widetilde{1}$	$\widetilde{1} \cup \widetilde{0}^\omega \not\sqsubseteq_\lambda \widetilde{1} \cup \widetilde{1}^\omega$
$\cap$	$\widetilde{0}^\omega \sqsubseteq_\lambda \widetilde{1} \cap \widetilde{1}^\omega$	$X \cap (\widetilde{0}^\omega \cup \widetilde{1}^\omega)$	$\widetilde{0}^\omega \not\sqsubseteq_\lambda \widetilde{1}^\omega$
$\vert\vert\vert$	$\widetilde{0}^\omega \sqsubseteq_\lambda \widetilde{1}$	$X \vert\vert\vert \widetilde{0}^\omega$	$\widetilde{0}^\omega \not\sqsubseteq_\lambda \widetilde{1} \vert\vert\vert \widetilde{0}^\omega$
$\Vert_+$	$\widetilde{0}^\omega \sqsubseteq_\lambda \widetilde{1}$	$X \Vert_+ \widetilde{0}^\omega$	$\widetilde{0}^\omega \not\sqsubseteq_\lambda \widetilde{1} \Vert_+ \widetilde{0}^\omega$

order, the pairwise order and the lexical order are not the only orders that we can consider for factual correctness. A natural question is:

Does there exist any other order with which the least factual-correctness fixpoints of all recursions can be determined using Tarski's theorem?

To answer this question, we first identify some necessary conditions that such a factual-correctness order must satisfy.

Any calculation of Tarski's least fixpoint starts from the bottom of a well-founded partial order. Let $\sqsubseteq_{\circledast}$ be the order that we are after. Note that $\sqsubseteq_{\circledast}$ should be a *partial order*, if we want to *uniquely* pinpoint fixpoints using Tarski's theorem. Let $\bot_{\circledast}$ denote the bottom of the order.

A semantics based on factual correctness must distinguish possible nontermination from necessary nontermination. In particular, the empty loop $\phi X \cdot X$ has an equivalent form: $\phi X \cdot (\mathrm{II} ⨾ X)$ where II is the unit of sequential composition. The corresponding function $f(X) = X$ of the empty loop immediately reaches its least fixpoint $\bot_{\circledast}$. The empty loop *never* terminates. According to factual correctness, its semantics must not contain any terminating behaviour; otherwise, for example, if its semantics were chaos $S^\dagger$, we would have an undesirable inequality:

$$(\phi X \cdot X) ⨾ \widetilde{0} \quad \neq \quad (\phi X \cdot X)$$

in which the empty trace $\langle\,\rangle$ is a behaviour of the right-hand side but not a behaviour of the left-hand side. The inequality suggests that the behaviour of a nonterminating process could be altered if it is followed by another process that generates an event 0. Such counterintuitive interpretation is the result of the incorrect semantic assumption on the empty loop. Thus we conclude that $\bot_{\circledast} \subseteq S^\omega$.

On the other hand, the empty loop is an executable program that at least generates some outputs. Thus its semantics must not be empty, i.e. $\emptyset \subset \bot_{\circledast}$.

In summary, the required order $\sqsubseteq_{\circledast}$ must satisfy:

(A) The order $\sqsubseteq_{\circledast}$ is a well-founded partial order of trace specifications;
(B) $\top \subset \bot_{\circledast} \subseteq \lhd$ where $\bot_{\circledast}$ is the least trace specification (i.e. the bottom) with regard to the order $\sqsubseteq_{\circledast}$;
(C) all compositions of our language are $\sqsubseteq_{\circledast}$-monotonic.

None of the five orders that we have considered satisfies all three criteria.

	$\supseteq$	$\subseteq$	$\sqsubseteq_{\varepsilon}$	$\sqsubseteq_{\pi}$	$\sqsubseteq_{\lambda}$
(A)	✓	✓	✓	✓	✓
(B)			✓	✓	✓
(C)	✓	✓			

Indeed the following theorem has ruled out the existence of any such order.

Theorem 5 (Non-monotonicity of parallelism) *No order satisfying* (A), (B) *and* (C) *above exists.*

Proof. Suppose that $\sqsubseteq_{\circledast}$ is an order satisfying (A), (B) and (C). Let P, Q and R be three specifications in our language. We construct two recursions:

$$\begin{aligned} f(X) &\mathrel{\widehat{=}} P \mid ((X \cap Q) \cup ((X|\top \mathbin{;} \lhd) \cap R \cap \bot_{\circledast})) \\ g(X) &\mathrel{\widehat{=}} P \mid ((X \cap R) \cup ((X|\top \mathbin{;} \lhd) \cap Q \cap \bot_{\circledast})) . \end{aligned}$$

Both recursions must be $\sqsubseteq_{\circledast}$-monotonic according to (C). Since $\bot_{\circledast}$ is the bottom of the order $\sqsubseteq_{\circledast}$, we thus have $\bot_{\circledast} \sqsubseteq_{\circledast} \rhd$. This leads to

$$\begin{aligned} P|(Q \cap \bot_{\circledast}) &= f(\bot_{\circledast}) \sqsubseteq_{\circledast} f(\rhd) = P|(R \cap \bot_{\circledast}) \\ P|(R \cap \bot_{\circledast}) &= g(\bot_{\circledast}) \sqsubseteq_{\circledast} g(\rhd) = P|(Q \cap \bot_{\circledast}) . \end{aligned}$$

The order $\sqsubseteq_{\circledast}$ is a *partial* order according to (A). Thus $P|(Q \cap \bot_{\circledast}) = P|(R \cap \bot_{\circledast})$ must hold for arbitrary specifications P, Q and R. Let $P = Q = \top$ and $R = \bot_{\circledast}$. We then have $\top = \bot_{\circledast}$, which contradicts (B). Thus the order that satisfies all three criteria (A), (B) and (C) does not exist. $\square$

Note that, since the counterexample in the proof uses only the conjunctive parallel composition, the result is also valid for the sub-language excluding fair-interleaving and synchronous parallelism.

Since no appropriate order exists, Tarski's fixpoint theorem alone is no longer applicable to our language. This, however, does not exclude the existence of each recursion's least fixpoint with regard to *some* partial order satisfying the three requirements. With additional information about a recursion, we may still be able to determine such a fixpoint. We now take advantage of just that.

6 Applications of partitioned fixpoint

6.1 Partitioned fixpoint in our trace language

Since the space of trace specifications forms a complete lattice under the refinement order, the technique of partitioned fixpoint is now applicable. We list the corresponding notations of complete lattices and those of our trace language in Table 8.

Table 8
Comparison between notations of complete lattices and our trace language

Complete Lattice	Our Language
$\top$	$\top$
$\bot$	$\bot$
$P \sqcap Q$	$P \cup Q$
$P \sqcup Q$	$P \cap Q$
$P \sqsubseteq Q$	$P \supseteq Q$
A	$\lhd$
$\sim\!A$	$\rhd$
$P \mid_A Q$	$P \mid Q$
$P \sqsubseteq_{\lambda(A)} Q$	$P \sqsubseteq_{\lambda} Q$
$\psi_A f$	ϕf

The semantics of the recursion ϕf is defined by a partitioned fixpoint with partitioning element $\lhd$.

Definition 4 $\quad \phi f \;\; \widehat{=} \;\; \psi_{\lhd} f$

In general, if the terminating behaviours of any composition $P * Q$ are 'pure' and depend on only terminating behaviours of P and Q, then the composition satisfies the distributivity condition $(P*Q) \mid \top = (P \mid \top)*(Q \mid \top)$ according to Law 2(7). All compositions of our trace language are 'pure' in this sense according to Law 3. Thus the following proposition on distributivity can be readily proved.

Proposition 6 *Distributivity $f(P) \mid \top = f(P \mid \top) \mid \top$ holds for any specification P and any recursion f in our trace language.*

Example: The calculation of the recursion $f(X) \ \widehat{=} \ ((X \,;\, \widetilde{0}) \cap \widetilde{0}) \cup \mathrm{II}$ starts from $\lhd$ and reaches its fixpoint $\widetilde{0}$ in two steps:

$$f(\lhd) \ = \ \mathrm{II}\,, \quad f(\mathrm{II}) \ = \ \widetilde{0} \cup \mathrm{II} \quad \text{and} \quad f(\widetilde{0} \cup \mathrm{II}) \ = \ \widetilde{0} \cup \mathrm{II}\,.$$

The specification $(\widetilde{0} \cup \mathrm{II})$ is actually the unique fixpoint and therefore the least fixpoint with regard to any partial order. However it cannot be determined using Tarski's theorem based on the variant Egli-Milner order, because $\mathrm{II} \not\sqsubseteq_{\varepsilon} (\widetilde{0} \cup \mathrm{II})$. Fortunately, Theorem 4 is applicable (according to Proposition 6):

ϕf

$=$ definition

$\mu X \cdot f(\nu f \mid X)$

$=$ $\nu f = \widetilde{0} \cup \mathrm{II}$

$\mu X \cdot f((\widetilde{0} \cup \mathrm{II}) \mid X)$

$=$ definition of f

$\mu X \cdot (((\widetilde{0} \cup \mathrm{II}) \mid X \,;\, \widetilde{0}) \cap \widetilde{0}) \cup \mathrm{II}$

$=$ routine simplification

$\mu X \cdot (\widetilde{0} \cup \mathrm{II})$

$=$ fixpoint of constant function

$\widetilde{0} \cup \mathrm{II}$

The result is exactly as expected. □

Similar examples can be constructed for synchronous parallelism and fair-interleaving parallelism [6].

6.2 Previous fixpoints become special cases.

Let $f(X)$ be a recursion in our language. It is hence $\supseteq$-monotonic and distributes $(\cdot \mid \top)$. We now study the relationship between the partitioned fixpoint and previous fixpoints with regard to the five partial orders:

(A) the calculation in (5) guarantees that the $\supseteq$-least fixpoint μf equals the partitioned fixpoint $\psi_{\perp} f$ with partitioning element $\perp$;
(B) similarly, the calculation in (6) guarantees that the $\supseteq$-greatest fixpoint νf equals the partitioned fixpoint $\psi_{\top} f$ with partitioning element $\top$;
(C) Theorem 4 states that the $\sqsubseteq_{\lambda}$-least fixpoint always exists and equals the partitioned fixpoint $\psi_{\lhd} f$ with partitioning element $\lhd$;
(D) since the pairwise order is coarser than the lexical order, i.e. $\sqsubseteq_{\pi} \subseteq \sqsubseteq_{\lambda}$, according to Proposition 2, if there exists the $\sqsubseteq_{\pi}$-least fixpoint then it

must equal the partitioned fixpoint $\psi_{\lhd} f$;

(E) similarly, since $\sqsubseteq_{\varepsilon} \subseteq \sqsubseteq_{\lambda}$, if there exists the $\sqsubseteq_{\varepsilon}$-least fixpoint then it must equal $\psi_{\lhd} f$.

Now all fixpoint techniques based on the five partial orders have been represented as special cases of the partitioned fixpoint.

7 Case study: combining sequential and reactive specifications

Our language of trace specifications in the previous sections is simplistic. Each specification describes only the intermediate states of a reactive process. That means basic imperative commands such as assignment statement, if-then-else conditional and do-loop cannot be incorporated.

In this section, we will discuss a more realistic parallel specification language combining pre-post sequential specification and trace-based reactive specification. A similar language [8] with conjunctive parallelism has been successfully applied to specifications of PRAM and BSP.

Let S be the state space. A *combined specification* is a set of behaviours, each of which is a triple (x, s, y) of the initial state $x \in S$, the final state y and the trace $s \in S^{\dagger}$ of all intermediate states.

Table 9 lists the basic commands of combined specifications (not including parallel compositions).

Table 9
Combined specifications based on factual correctness

$\rhd$	$\widehat{=}$	$\{(x,s,y) \mid x,y \in S,\ s \in S^*\}$	termination
$\lhd$	$\widehat{=}$	$\{(x,s,y) \mid x,y \in S,\ s \in S^\omega\}$	nontermination
$\top$	$\widehat{=}$	$\emptyset$	magic
$\bot$	$\widehat{=}$	$\{(x,s,y) \mid x,y \in S,\ s \in S^\dagger\}$	chaos
`idle`	$\widehat{=}$	$\{(x,\langle x\rangle,x) \mid x \in S\}$	1-step idle with stable state
$x :\in E$	$\widehat{=}$	$\{(x,\langle\rangle,y) \mid x \in S,\ y \in E(x)\}$	nondeterministic assignment
$P \mathbin{;} Q$	$\widehat{=}$	$\{(x, s^\frown t, z) \mid (x,s,y) \in P,\ (y,t,z) \in Q\}$	sequential composition
$P \cup Q$			nondeterminism (set union)
$P \mid Q$	$\widehat{=}$	$(P \cap \rhd) \cup (Q \cap \lhd)$	partition
ϕf	$\widehat{=}$	$\mu X \cdot f(\nu f \mid X)$	recursion (partitioned fixpoint)

Note that, to obtain a more elegant semantics, we can encode the trace s as the difference $tr' - tr$ of two traces $tr, tr' \in S^{\dagger}$ where tr is the prefix of tr'. A specification is then represented as a predicate $P(x, tr, x', tr')$ on four variables [8,22]. The advantage of such encoding is that sequential composition simply becomes relational composition $P \,⨟\, Q \;=\; \exists x_0, tr_0 \cdot P(x, tr, x_0, tr_0) \wedge Q(x_0, tr_0, x', tr')$. In this paper we focus on the presentation with triples, although the results are also applicable to the four-variable predicative presentation or any other subtly different presentation.

More realistic commands including deterministic assignment $x := e$, skip II, conditional magic $(b)_{\top}$, conditional $P \lhd b \rhd Q$ and do-loop $(\texttt{do}\ b \rightarrow P\ \texttt{od})$ can be easily derived from the basic commands (see Table 9). Various forms of parallelism can be defined. We first consider parallel compositions in general.

Definition 5 *A parallel composition $\|$ is regular, if it satisfies the following laws:*

(1) commutativity $P \| Q \;=\; Q \| P$,
(2) disjunctivity $(P \cup Q) \| R \;=\; (P \| R) \cup (Q \| R)$,
(3) weak strictness $\vartriangleleft \| P \;\subseteq\; \vartriangleleft$.

A routinely provable proposition shows that regularity is a sufficient condition for the applicability of the partitioned fixpoint.

Proposition 7 *If all parallel compositions of combined specifications are regular, then any recursion f is $\supseteq$-monotonic and the distributivity law $f(P)|\top = f(P|\top)|\top$ holds for any combined specification P.*

For example, set intersection still represents the conjunctive parallelism and is regular. We may also consider a more realistic form of parallelism that combines conjunctive parallelism for initial and final states and fair-interleaving parallelism for intermediate states.

Definition 6 $P \mathbin{|||_{\cap}} Q \;\;\widehat{=}\;\; \{(x, s, y) \mid (x, u, y) \in P,\ (x, v, y) \in Q,\ s \in u \mathbin{⫴} v\}$

From a common initial state, two specifications in the above composition must agree on the same final state, but their intermediate states are arbitrarily interleaved. The composition is regular; therefore Theorem 4 is applicable to the language of combined specifications including the above parallel composition.

The definitions of the orders $\sqsubseteq_{\varepsilon}$, $\sqsubseteq_{\pi}$ and $\sqsubseteq_{\lambda}$ are still valid for combined specifications, although termination $\vartriangleright$, nontermination $\vartriangleleft$ and sequential composition $⨟$ have different semantics now.

Theorem 8 *The combined parallel composition $|||_{\cap}$ is not monotonic with regard to the orders $\sqsubseteq_{\pi}$, $\sqsubseteq_{\varepsilon}$ or $\sqsubseteq_{\lambda}$.*

The following recursion is a counterexample:

$$f(X) \;\; \widehat{=} \;\; ((X \;\; |||_\cap \; \mathrm{II}) \; ; \; \lessdot) \cup \mathrm{II} \,. \tag{10}$$

Starting from $\lessdot$, recursion $f(X)$ reaches its fixpoint $\lessdot \cup \mathrm{II}$ in two steps. From Table 7, we can see that $\top \mid f(\lessdot) \; = \; \lessdot \;\; |||_\cap \; \mathrm{II}$ and $\top \mid f^2(\lessdot) \; = \; \lessdot$; however, $\lessdot \;\; |||_\cap \; \mathrm{II} \; = \; \{(x, s, x) \mid x \in S, \; s \in S^\omega\}$ leads to inequality $\lessdot \;\; |||_\cap \; \mathrm{II} \; \subset \; \lessdot$ and non-monotonicity: $f(\lessdot) \; \not\sqsubseteq_\pi \; f^2(\lessdot)$, $f(\lessdot) \; \not\sqsubseteq_\varepsilon \; f^2(\lessdot)$ and $f(\lessdot) \; \not\sqsubseteq_\lambda \; f^2(\lessdot)$.

Table 10
Iteration of the recursion (10)

$f^\kappa(\lessdot)$			$f^\kappa(\lessdot) \mid \top$	$\top \mid f^\kappa(\lessdot)$
$\lessdot$	$=$	$\lessdot$	$\top$	$\lessdot$
$f(\lessdot)$	$=$	$(\lessdot \;\; |||_\cap \; \mathrm{II}) \cup \mathrm{II}$	II	$\lessdot \;\; |||_\cap \; \mathrm{II}$
$f^2(\lessdot)$	$=$	$\lessdot \cup \mathrm{II}$	II	$\lessdot$
...		...	...	...
$f^\kappa(\lessdot)$	$=$	$\lessdot \cup \mathrm{II}$	II	$\lessdot$

Fortunately the partitioned fixpoint of the example (10) can be routinely calculated:

ϕf

$=$ definition

$\mu X \cdot f(\nu f \mid X)$

$=$ $\nu f = \lessdot \cup \mathrm{II}$ and laws of partition

$\mu X \cdot f(\mathrm{II} \mid X)$

$=$ definition of f

$\mu X \cdot (((\mathrm{II} \mid X) \;\; |||_\cap \; \mathrm{II}) \; ; \; \lessdot) \cup \mathrm{II}$

$=$ laws of partition

$\mu X \cdot (((\mathrm{II} \cup (\top \mid X)) \;\; |||_\cap \; \mathrm{II}) \; ; \; \lessdot) \cup \mathrm{II}$

$=$ properties of $|||_\cap$

$\mu X \cdot \lessdot \cup ((\top \mid X) \;\; |||_\cap \; \mathrm{II}) \; ; \; \lessdot) \cup \mathrm{II}$

$=$ $(P \; ; \; \lessdot) \; \subseteq \; (\bot \; ; \; \lessdot) \; = \; \lessdot$

$\mu X \cdot \lessdot \; \cup \; \mathrm{II}$

$=$ fixpoint of constant function

$\lessdot \; \cup \; \mathrm{II}$

In the above example, the strongest fixpoint νf happens to be the same as the partitioned fixpoint; however, only by going through the above calculation can we be sure that it is indeed the $\sqsubseteq_\lambda$-least fixpoint.

Numerous other forms of combined parallelism can be defined. For example we may combine conjunctive parallelism of the initial and final states with synchronous parallelism of the intermediate states. All combined parallel compositions are regular but not monotonic with regard to the variant Egli-Milner order.

8 Conclusions

We have studied the modelling of recursion in the style of relational semantics. Most results obtained are also applicable to other formalism such as predicate-transformer semantics [11] and axiomatic semantics [20].

The trace language used in this paper has incorporated sequentiality, nondeterminism, reactiveness (including infinite behaviours), general recursion and three different forms of parallelism. For such a specification language, factual correctness is more appropriate than partial and total correctness. The original Egli-Milner order, constructed from a flat domain, was designed mainly for sequential semantics; the variant Egli-Milner order, constructed from a more general bipartite order, can be used for a semantics allowing infinite behaviours. The pairwise order and the lexical order are finer than the variant Egli-Milner order. All of them can simulate the execution of recursions, and the least fixpoints with regard to the orders are the same if they do exist. Unfortunately, parallel compositions are normally not monotonic with regard to the orders. In fact, a major theorem shows that there exists no well-founded partial order (based on factual correctness) with regard to which all recursions are monotonic. This means that non-monotonicity can naturally arise from parallelism, and Tarski's theorem alone is not enough to tackle non-monotonic parallel recursion.

To determine the fixpoints of potentially non-monotonic recursions, we have proposed a technique called *partitioned fixpoint*. The most surprising property of the partitioned fixpoint is its link with the lexical order. Any recursion of our trace language has its partitioned fixpoint, which is also the least lexical-order fixpoint, although it may not be monotonic with regard to the lexical order. The partitioned fixpoint requires a precondition that terminating behaviours of any composition must not depend on the nonterminating behaviours of its arguments. This requirement is weak. All program constructs that we know (including negation, see [8]) satisfy it. Previous fixpoints including the weakest fixpoint [21,22] of total correctness, the strongest fixpoint [32] of partial correctness and the least Egli-Milner fixpoint [15,27] of factual correctness simply become special cases of the partitioned fixpoint with different partitioning elements. The theorem of partitioned fixpoint can be generalised to allow multiple partitions. In this sense the approach is similar to vector iteration, although we do not require each partition to be monotonic or increasing. Partitioned fixpoint is the combination of the least fixpoint and the greatest fixpoint. In this sense, our approach also coincides with Park's observation on fair interleaving [29]. The partitioned fixpoint of a recursion is also consistent with the fixpoint defined in [14], if the recursion is in a special normal form of sequential language.

In the case study, a language that combines sequential and reactive specifications was introduced. Various combined parallel compositions can be defined. In particular, we have studied the combination of conjunctive parallelism for initial and final states and fair-interleaving parallelism for intermediate states. The technique of partitioned fixpoint is shown to be applicable. The calculated fixpoints are consistent with our intuition.

The semantics of combined specifications can be made more concrete by adding a variable to denote the divergent point. Arbitrary nontermination containing all non-divergent infinite behaviours can then be distinguished from any intermediate failure (e.g. the empty loop): the former never diverges, while the latter diverges after some point. The trace semantics can be further generalised to the semantics based on other temporal domains such as real time and branching time.

Acknowledgements: This paper partly arose from a DPhil thesis. The author is grateful to his supervisor J.W. Sanders for various discussions, comments, suggestions and review of the draft of this paper, and to Roland Backhouse and Steve Dunne for pointing out errors in early versions of this paper. The author also gratefully acknowledge the insightful discussions with Ian Hayes, Jifeng He and Gavin Lowe. The paper of which [6] is a contraction, has benefited from wide-ranging comments from its referees to whom we are grateful.

References

[1] R. J. R. Back and K. Sere. Stepwise refinement of action systems. In *Mathematics of Program Construction*, volume 375 of *LNCS*, pages 115–138. Springer-Verlag, 1989.

[2] J. A. Bergstra and J. W. Klop. Algebra of communicating processes with abstraction. *Theoretical Computer Science*, 37(1):77–121, 1985.

[3] M.M. Bonsangue and J.N. Kok. The weakest precondition calculus: Recursion and duality. *Formal Aspects of Computing*, 6(A):788–800, 1994.

[4] N. Bourbaki. *Elements of Mathematics: Theory of Sets*. Hermann, 1968.

[5] S. Brookes. Full abstraction for a shared-variable parallel language. *Information and Computation*, 127(2):145–163, 1996.

[6] Y. Chen. A fixpoint theorey for non-monotonic parallelism. In *11th Annual Conference of the European Association for Computer Science Logic, CSL'02*, volume 2471 of *LNCS*, pages 120–134. Springer-Verlag, 2002.

[7] Y. Chen. Generic composition. *Formal Aspects of Computing*, 14(2):108–122, 2002.

[8] Y. Chen and J.W. Sanders. Logic of global synchrony. In *12th International Conference on Concurrency Theory*, volume 2154 of *LNCS*, pages 487–501. Springer-Verlag, 2001.

[9] P. Cousot and R. Cousot. Constructive versions of Tarski's fixed point theorems. *Pacific Journal of Mathematics*, 82(1):43–57, 1979.

[10] J. Davies and S. Schneider. A brief history of Timed CSP. *Theoretical Computer Science*, 138(2):243–271, 1995.

[11] E.W. Dijkstra. Guarded commands, nondeterminacy and the formal derivation of programs. *Communications of the ACM*, 18(8):453–457, 1975.

[12] E.W. Dijkstra and C. S. Scholten. Semantics of recursive procedures. EWD 859, 1983.

[13] E.W. Dijkstra and A.J.M. von Gasteren. A simple fixed-point argument without the restriction to continuity. *Acta Informatica*, 23(1):1–7, 1986.

[14] H. Doornbos. A relational model of programs without the restriction to egli-milner constructs. In *PROCOMET'94*, volume A-56 of *IFIP Transactions*, pages 363–382. North-Holland, 1994.

[15] S.E. Dunne. Recasting Hoare and He's relational theory of programs in the context of general correctness. Technical report, School of Computing and Mathematics University of Teesside, 2000.

[16] S. Fortune and J. Wyllie. Parallelism in random access machines. In *10th Annual ACM Symposium on Theory of Computing*, pages 114–118. ACM Press, 1978.

[17] A. Geser and et al. Non-monotone fixpoint iterations to resolve second-order effects. In *6th Intl. Symp. Compiler Construction*, volume 1060 of *LNCS*, pages 106–120, 1996.

[18] E.C.R. Hehner. Specification, programs, and total correctness. *Science of Computer Programming*, 34:191–205, 1999.

[19] P. Hitchcock and D. Park. Induction rules and termination proofs. In *ICALP'72*, pages 225–251, 1972.

[20] C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–583, 1969.

[21] C.A.R. Hoare. *Communicating Sequential Processes.* Prentice Hall, 1985.

[22] C.A.R. Hoare and J. He. *Unifying Theories of Programming.* Prentice Hall, 1998.

[23] J. Lassez, V.L. Nguyen, and L. Sonenberg. Fixed point theorems and semantics: a folk tale. *Information Processing Letters*, 14(3):109–111, 1982.

[24] K. Martin. The measurement process in domain theory. In *ICALP'00*, volume 1853 of *LNCS*, pages 116–126. Springer-Verlag, 2000.

[25] W.F. McColl. Scalability, portability and predictability: The BSP approach to parallel programming. *Future Generation Computer Systems*, 12:265–272, 1996.

[26] C.C. Morgan and A. McIver. Unifying wp and wlp. *Information Processing Letters*, 59:159–163, 1996.

[27] G. Nelson. A generalisation of Dijkstra's calculus. *ACM Transactions on Programming Languages and Systems*, 11(4):517–561, 1989.

[28] S.O. Nystrom. There is no fully abstract fixpoint semantics for non-deterministic languages with infinite computations. *Information Processing Letters*, 60:289–293, 1996.

[29] D.M.R. Park. On the semantics of fair parallelism. In *Abstract Software Specification*, volume 86 of *LNCS*, pages 504–526. Springer-Verlag, 1980.

[30] G.D. Plotkin. Lecture notes on domain theory. The Pisa Notes, 1983.

[31] A. Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13:45–60, 1981.

[32] A.W. Roscoe. *The Theory and Practice of Concurrency.* Prentice Hall, 1998.

[33] A. Tarski. A lattice-theoretical fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.

[34] L.G. Valiant. A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111, 1990.

[35] J. Woodcock and J. Davis. *Using Z: specification, refinement, and proof.* Prentice Hall, 1991.