Communicating Process Architectures 2006
Peter Welch, Jon Kerridge, and Fred Barnes (Eds.)
IOS Press, 2006

Compositions of Concurrent Processes

Mark BURGIN [a] and Marc L. SMITH [b 1]
[a] *Department of Computer Science, University of California, Los Angeles*
Los Angeles, California 90095, USA
[b] *Department of Computer Science, Vassar College*
Poughkeepsie, New York 12604, USA

Abstract. Using the extended model for view-centric reasoning, EVCR, we focus on the many possibilities for concurrent processes to be composed. EVCR is an extension of VCR, both models of true concurrency; VCR is an extension of CSP, which is based on an interleaved semantics for modeling concurrency. VCR, like CSP, utilizes traces of instantaneous events, though VCR permits recording parallel events to preserve the perception of simultaneity by the observer(s). But observed simultaneity is a contentious issue, especially for events that are supposed to be instantaneous. EVCR addresses this issue in two ways. First, events are no longer instantaneous; they occur for some duration of time. Second, parallel events need not be an all-or-nothing proposition; it is possible for events to partially overlap in time. Thus, EVCR provides a more realistic and appropriate level of abstraction for reasoning about concurrent processes. With EVCR, we begin to move from observation to the specification of concurrency, and the compositions of concurrent processes. As one example of specification, we introduce a description of I/O-PAR composition that leads to simplified reasoning about composite I/O-PAR processes.

Keywords. event, trace, composition, process, true concurrency, I/O-PAR.

Introduction

The motivation for view-centric reasoning (VCR) [13] stemmed from a desire to preserve more information about the history of a computation than is preserved in a traditional CSP trace. With the CSP model of concurrency, Hoare introduced the powerful notion that one could reason about a computation by reasoning about its trace of observable events [4]. CSP is an interleaved model of concurrency, which means while some events may appear to occur simultaneously during a computation, an observer records such occurrences in an arbitrary, interleaved fashion. Such behavior by a computation's observer results in a decrease of entropy (i.e., imposing an order of events that did not actually exist during the computation) within the CSP trace. VCR begins to address this issue with entropy-preserving metaphors and abstractions, such as lazy observation; multiple, possibly imperfect observers; and parallel event traces.

In particular, VCR traces were defined over multisets of observable events, rather than atomic events. The laziness stems from sparing the observer the stressful decision of what order to record simultaneous events. Furthermore, VCR distinguishes two types of trace: a computation's history, and its corresponding views. Since it is possible for a distributed computation to be observed by more than one observer, consequences of relativity theory provide for the possibility of different views of the same computation. Thus, the parallel

[1] Corresponding Author: mlsmith@cs.vassar.edu

event traces of VCR extend CSP in two important ways. First, VCR provides the entropy-preserving abstraction of parallel events, and second, it provides a model for associating a computation's history with its multiple, possibly imperfect, but realistic views.

VCR's contribution to the theory of CSP is not as general as it could be; a model of true concurrency should support abstractions for events whose occurrence partially overlap in time. Thus, the possibility of observed event simultaneity is not merely a Boolean proposition, but rather a continuum: events A and B may overlap entirely, partially, or not at all. Extended VCR (EVCR), which is introduced by the authors in [3], is the next step in the evolution of CSP from a model that reduces concurrency to an interleaving of sequential events, to a model of true concurrency that provides abstractions that represent degrees of interleaving. EVCR provides abstractions that reflect the possibilities of behavior that arise when composing today's loosely-coupled, distributed systems, and should therefore prove beneficial for modeling and reasoning about such computation.

In this paper, we extend the initial exposition of EVCR in [3], paying special attention to compositions of concurrent processes. Compositions of systems and processes become an important tool for the development of modern hardware and software systems with complex sets of requirements. Unfortunately, as Neumann [11] writes, there is a huge gap between theory and common practice: system compositions at present are typically ad hoc, based on the intersection of potentially incompatible component properties, and dependent on untrustworthy components that were not designed for interoperability – often resulting in unexpected results and risks.

Results of this paper give additional evidence to this statement. The analysis of real systems and processes made possible to find new types even for such a well-known operation as sequential composition. This was possible because the authors considered not only data-flow relations between systems and processes used in the conventional sequential composition, but also temporal and causal relations.

In the first section, we develop a rigorous setting for studying and reasoning about concurrent processes. We give strict definitions of different kinds of events and processes. In the second section, we introduce and study basic temporal relations that exist between events in concurrent processes. The topic of the third section is composition of concurrent processes. Section 4 presents a case study of I/O-PAR composition, and the benefits of EVCR abstractions for reasoning about the specification and properties of composite I/O-PAR processes.

1. Systems, Processes and Events

The basic concept of a theory of concurrent processes is event. We consider two types of events: abstract or process events and embodied or system events. Usually, a system belongs to some environment, while a process interacts (communicates) with other processes, which form the environment of the first process. Taking into account this natural structure, we come to three categories of events.

Definition 1. An *event in a system R* (*of a process P*) is a change of [the state, or phase] of the system *R* (in the process *P*).

For instance, an event in such system as a finite automaton or finite state machine is a state transition of this automaton or machine. Sending a message is an event in a network. Printing a symbol or a text, producing a symbol on the screen of the display, and calculating 2 + 2 are events in a computer. The latter event can be also an event in a calculator.

It is natural to consider events as multisets (cf., for example, [13]). Thus, a process can contain several copies of the same event. At the same time, it is possible that two or more processes contain different copies of an event in some cases, while they have a common copy (or several common copies) of the same event. For instance, taking such an event as sending a message, we see that different copies of this event belong, as a rule, to many different processes. At the same, each copy of such an event as information (data) exchange between two processes necessarily belongs to two processes.

Definition 2. An *event by a system R* (*a process P*) is a change in the environment of the system *R* (the process *P*) caused by *R*.

For instance, when a system *R* sends information to another system, it is an event by this system *R*.

Definition 3. An *event for a system R* (*a process P*) is a change in relations between the system *R* (the process *P*) and its environment.

For instance, when a system *R* was connected to another system and then this connection is broken, it is an event for this system *R*.

Definition 4. When the change is detectable, i.e., there are means to discern this change, the corresponding event is called *observable*.

Not all events are observable if there are, for example, indistinguishable states in a system or continuous changes are observed in a system where it is impossible to discern the next state of the system. The question is why do we need unobservable events? The reason for this is that in some cases, a researcher can build a better model of a process taking not only observable events. For instance, in physics, the majority of physical quantities take value in the set of all real numbers. Any interval of real numbers contains uncountable quantity of numbers. However, it is known that measurement can give only a finite number of values. Thus, the majority of values of continuous physical quantities, such as speed, acceleration, temperature, etc., are unobservable but models that use real numbers are more efficient than finite models.

Definition 5. A *process* is a system of related events.

This is a very general definition. For example, one event may be considered as a process. Several events in some system also may be considered as a process. However, it is reasonable to have, at first, such a general definition and then to separate different kinds of processes relevant to concrete situations. It is also necessary to remark that this definition of a process depends on our comprehension and/or description.

There are two types of processes: *abstract processes*, which consist of abstract events, and *embodied processes*, consist of events in some system. There are also two types of systems: structural or semiotic, such as a finite automaton or Turing machine, and physical, such as computers, servers, routers, the World Wide Web. As a result, embodied processes are subdivided into two classes: *structurally* and *physically embodied*.

Definition 6. A process *P* is called *finite* if it consists of a finite number of events. Otherwise, *P* is called *infinite*.

For a long time, computer scientists studied and accepted only finite processes. The finiteness condition was even included in definitions of the main mathematical models of

algorithms, such as Turing machines. However, infinite processes become more and more important in theoretical studies and practical applications (cf., for example [2, 12, 14]).

Definition 7. Any subset of a process P in which all relations between events are induced by relations in P is called a *subprocess* of P.

Proposition 1. Any subprocess of a subprocess of P is a subprocess of P.

2. Temporal Relations between Events

There are different relations between events in a process, as well as between events in a system. The pivotal is the ordering relation that defines time in the process or is defined by time in the system to which these events belong.

Analyzing temporal relations in real computational and other processes, we can distinguish several types of event pairs:

- *A sequential pair of events* consists of two events where one ends before the next starts.
- *A simultaneous pair of events* consists of two events where both events start and end simultaneously.
- *A coexisting pair of events* consists of two events where one starts before the next ends.
- *A separable pair of events* consists of two events that are not coexisting.
- *A separated pair of events* consists of two events for which coexistence is excluded.
- An event *r is included in* an event *q* if the event *q* starts before or when the event *r* starts and the event *q* ends after or when the event *r* ends.

Types of events are formally determined by the corresponding binary relations on sets of events:

- The *coexistence relation* CE_R. This relation allows several interpretations, reflecting different modalities. It can formalize situations in which events must go so that one starts before another is finished (prescriptive coexistence). It can also formalize situations in which one event really starts before another is finished (actual coexistence). It can as well formalize situations in which one event may start before another is finished (possible coexistence).
- The *separability relation* SP_R. This relation shows when events are (or may be) not coexisting.
- The *ordering relation* OD_R. This relation allows several interpretations. It can formalize the order in which events must happen (prescriptive order), e.g., at first, we need to compute the value of a function, and only then to print the result. It can also formalize the order in which events really happened (actual order). It can as well formalize the order in which events may happen (possible order).
- The *simultaneity relation* ST_R. This relation allows several interpretations. It can formalize situations in which events must go simultaneously (prescriptive coexistence). It can also formalize situations in which events really go simultaneously (actual coexistence). It can as well formalize situations in which events may go simultaneously (possible coexistence).
- The *inclusion relation* IC_R. This relation allows several interpretations. It can formalize situations in which events are constrained so that the included event ends

before or when the other is finished and starts after or when the other starts (*prescriptive* inclusiveness). It can also formalize situations in which the included event really ends before or when the other is finished and really starts after or when the other starts (*actual* inclusiveness). It can as well formalize situations in which it is possible that the included event ends before or when the other is finished and starts after or when the other starts (*possible* inclusiveness).

EVCR actually models arbitrary processes. Some illustrations of the above relations may be found in music for the piano, where the musical notes are events. A piece of music is composed from individual notes, each produced by a corresponding key on the piano, according to a musical prescription, represented in the form of sheet music. The sheet music prescribes when to play separated notes in succession (e.g., a run), and when to play notes in different possible coexisting fashions. Chords, arpeggios, melodies, harmonies, and syncopated rhythms are but a few examples of what can be prescribed by different combinations of the above relations. Upon performing such a piece of music, members of the audience subsequently make a determination of how well the actual composition of musical notes match what they believe was prescribed. In some cases, the result is a round of applause.

Returning our attention to grid automata, natural axioms on the above relations are derived from an analysis of real computations and other processes:

Axiom (CP 1). CE_R is a complement of SP_R .

An informal meaning of this is given by the following statement:

Corollary 1. Any two events are either coexisting or separable.

Axiom (CP 2). OD_R is a subset of SP_R .

An informal meaning of this is given by the following statement:

Corollary 2. Any two ordered events are separated.

Axiom (CP 3). ST_R and IC_R are subsets of CE_R .

An informal meaning of this is given by the following statement:

Corollary 3. Any two simultaneous events (of which one is included into another) are coexisting.

Definition 8. A binary relation Q is called a *tolerance* on a set X or of a set X if it is reflexive, i.e. xQx for all x from X, and symmetric, i.e., xQy implies yQx for all x, y from X.

Axiom (CP 4). CE_R is a tolerance.

An informal meaning of this is given by the following statement:

Corollary 4. Any event is coexisting with itself.

Axiom (CP 5). ST_R is an equivalence.

This implies the following property:

Corollary 5. Any event is simultaneous with itself.

Definition 9. A binary relation Q is called a *preorder* (or *quasiorder*) on a set X if it is reflexive, i.e. xQx for all x from X, and transitive, i.e., xQy and yQz imply xQz for all x, y, z from X. A preorder that also is antisymmetric is a *partial order*.

Axiom (CP 6). OD_R is a partial quasiorder.

Axiom (CP 7). SP_R is reflexive and symmetric.

Axiom (CP 8). IC_R is a partial order.

An informal meaning of this is given by the following statement:

Corollary 6. Any event is included into itself.

These relations define connections between events:

Definition 10. If Q is binary relation, then the transitive closure Q* of Q is the smallest transitive binary relation that contains Q.

As the intersection of transitive binary relations is a transitive relation, the transitive closure of a relation is unique.

Definition 11. Two events a and b are existentially connected if aCE_R^*b.

Informally, the existential connectedness of the events a and b means that there is sequence of events a_0 , a_1 , a_2 , a_3 , … , a_n such that $a_0 = a$, $a_n = b$, and a_{i-1} coexists with a_i for all $i = 1, 2, 3, \dots, n$.

Proposition 2. CE_R^* is an equivalence relation.

Definition 12. Two events a and b are *sequentially connected* if aOD$_R b$.

Sequential connectedness is a base for interleaving as the following result demonstrates:

Theorem 1. Processes P_1 , P_2 , P_3 , … , P_n are (potentially) interleaving if the relation OD_R can be extended to a total order on the set of all events from P_1 , P_2 , P_3 , … , P_n.

Definition 13. Two events a and b are *parallelly connected* if aST$_R b$.

Let us consider some important types of complex events:

Definition 14. A *parallel event* is a group of simultaneous events.

Proposition 3. For any event a from a finite number of finite processes, there is a maximal parallel event that contains a.

Definition 15. A *complete parallel event* is a group of all events that are simultaneous to one event from this group.

Proposition 4. A complete parallel event does not depend on the choice of the event to which all other events from this group are simultaneous.

Proposition 5. A complete parallel event is a maximal parallel event.

Proposition 6. If $ST_R = CE_R$, then OD_R induces an order relation on the set of all complete parallel events and it is possible to make this order linear.

Definition 16. A *coexisting event* is a group of events in which each pair of events is coexisting.

Proposition 7. Any parallel event is a coexisting event.

However, in general, not every coexisting event is a parallel event. These concepts coincide if and only if $ST_R = CE_R$.

Definition 17. A *complete coexisting event* is a maximal coexisting event.

As the simultaneity relation ST_R is a subset of the coexistence relation CE_R, we have the following result:

Proposition 8. A complete coexisting event includes as subsets all parallel events of its elements.

Proposition 9. Two events a and b are simultaneous if and only if a is included in b and b is included in a, or formally, $b ST_R a \Leftrightarrow a IC_R b \;\&\; a IC_R b$.

Corollary 7. $ST_R = IC_R \cap IC_R^{-1}$.

It is possible to extend all introduced relations from binary to n-ary relations on sets of events for any $n > 1$. To do this, we assume that each event a takes some interval of time $t(a)$. Note that such an interval $t(a)$ can have a zero length, i.e., to be a point of time. Let us denote by $bt(a)$ time of the beginning of the event a and by $et(a)$ time of the end of the event a. Then:

- The *ordering relation* OD_R^n : for events $a_1 , a_2 , \dots , a_n$, $OD_R^n(a_1 , a_2 , \dots , a_n)$ means that $et(a_i) \le bt(a_{i+1})$ for all $i = 1, 2, \dots, n - 1$.
- The *simultaneity relation* ST_R^n : for events $a_1 , a_2 , \dots , a_n$, $ST_R^n(a_1 , a_2 , \dots , a_n)$ means that $et(a_i) = et(a_j)$ and $bt(a_i) = bt(a_j)$ for all $i , j = 1, 2, \dots, n$.
- The *inclusion relation* IC_R^n : for events $a_1 , a_2 , \dots , a_n$, $IC_R^n(a_1 , a_2 , \dots , a_n)$ means that $et(a_i) \le et(a_{i+1})$ and $bt(a_i) \ge bt(a_{i+1})$ for all $i = 1, 2, \dots, n - 1$.
- The *coexistence relation* CE_R^n: for events $a_1 , a_2 , \dots , a_n$, $CE_R^n(a_1 , a_2 , \dots , a_n)$ means that $\cap_{i=1}^{n} t(a_i)$ is not void.
- The *separability relation* SP_R^n . This relation shows when any pair of events $a_1 , a_2 , \dots , a_n$ is (or may be) not coexisting.

For some of these relations, there is a natural correspondence with their binary analogues:

Proposition 10. $OD_R^2 = OD_R$, $ST_R^2 = ST_R$, $IC_R^2 = IC_R$, $CE_R^2 = CE_R$, and $SP_R^2 = SP_R$.

Proposition 11. $OD_R^n(a_1, a_2, \dots, a_n)$ is true if and only if $a_i OD_R a_{i+1}$ is true for all $i = 1, 2, \dots, n-1$.

Proposition 12. $IC_R^n(a_1, a_2, \dots, a_n)$ is true if and only if $a_i IC_R a_{i+1}$ is true for all $i = 1, 2, \dots, n-1$.

Proposition 13. $ST_R^n(a_1, a_2, \dots, a_n)$ is true if and only if $a_i ST_R a_j$ are true for all $i, j = 1, 2, \dots, n$.

Similar property for the coexistence relation CE_R^n is true only in a linear time. Namely, we have the following result:

Proposition 14. $CE_R^n(a_1, a_2, \dots, a_n)$ is true if and only if $a_i CE_R a_j$ are true for all i , j = 1, 2, …, n.

For non-linear time, this is not always correct as the following example demonstrates:

Example 1. Let us consider cyclic time [1] that goes from 1 to 12, while after 12 goes 1. Taking events a, b, and c for which $t(a) = [1, 7]$, $t(b) = [5, 11]$, and $t(c) = [10, 2]$, we have that aCE_Rb , cCE_Rb and aCE_Rc are true, while $CE_R^3(a, b, c)$ is not true.

3. Compositions of Processes

Definition 18. *Composition* is an operation that combines two or more processes into one process. If **c** is a composition operator and P_i , $i \in I$, are processes, then the result $\mathbf{c}(P_i , i \in I)$ is a process – also called a composition of the processes P_i .

For instance, the result of composition c of processes P and Q is denoted by $c(P, Q)$.

Remark 1. Very often composition of systems induces composition of processes in these systems.

Definition 19. Composition **c** of processes P and Q is called *faithful* if both of them become, i.e., are isomorphic to, subprocesses of the result $\mathbf{c}(P, Q)$ of this composition.

Definition 20. Composition **c** of processes P and Q is called *exact* if both of them become, i.e., are isomorphic to, disjoint subprocesses of the result $\mathbf{c}(P, Q)$ of this composition.

Example 2. Let us consider the sequential composition W of three processes A, B, and C where A is the input process of data 7 and 11, B is the process of calculating/computing 7 + 11, and C is the process of printing the result of this calculation.

If we denote by P the sequential composition of processes A and B, and by Q the sequential composition of processes B and C, then W is a faithful but not exact composition of processes P and Q.

Definitions show that any exact composition is faithful. Proposition 1 implies the following result:

Proposition 15. If **c** is a faithful (exact) composition of processes P and Q and **d** is a faithful (exact) composition of processes $\mathbf{c}(P, Q)$ and R, then $\mathbf{d}(\mathbf{c}(P, Q), R)$ is a faithful (exact) composition of processes P, Q and R.

Definition 21. Composition c of processes P and Q is called *strict* if any event from $\mathbf{c}(P, Q)$ belongs either to P or Q, or to both of them.

Proposition 16. If **c** is a strict composition of processes P and Q and **d** is a strict composition of processes $\mathbf{c}(P, Q)$ and R, then $\mathbf{d}(\mathbf{c}(P, Q), R)$ is a strict composition of processes P, Q and R.

The most popular composition is the sequential composition of systems, functions, algorithms, and processes. The first formalized form of the sequential composition was elaborated for functions in mathematics. A similar construction for processes results in the following concept:

Definition 22. The *sequential composition* $P\mathrm{s}Q$ of processes P and Q combines them so that the process $P\mathrm{s}Q$ contains all events from P and Q and relations between them, all output data of the first process are taken as input data of the second one, and any events a from P and b from Q, we have $a\mathrm{OD}_{\mathrm{R}}b$, i.e., all events from P precede all events from Q.

The sequential composition is often used in computer science to prove different properties of automata, algorithms, and formal languages. For instance, the sequential composition of finite automata is used in [5] to prove that any regular language is the language accepted by some finite automaton or to show that the class of all regular languages is closed with respect to concatenation. In [2], sequential compositions are used to prove that inductive Turing machines can compute (generate) and decide the whole arithmetical hierarchy.

However, for computational and other processes, it is possible to build more kinds of sequential composition:

Definition 23. The *weak sequential composition* $P\mathbf{ws}Q$ of processes P and Q combines them so that the process $P\mathbf{ws}Q$ contains all events and relations between them from P and Q and for any events a from P and b from Q, we have $a\mathrm{OD}_{\mathrm{R}}b$, i.e., all events from P precede all events from Q.

Example 3. If we compute 7 + 11 by a process P and after compute 8·12 by a process Q using the same computer, then the whole process W will be the weak sequential composition of processes P and Q.

Remark 2. When a process P produces no output data, then for any process Q, the weak sequential composition of processes P and Q coincides with the sequential composition of processes P and Q.

Remark 3. When a process P produces no output data, it does not mean that the process P, gives no result. The result of the processes P can be change in some system, information transition or reception, etc.

Proposition 17. Any (weak) sequential composition is exact and strict.

Definition 24. The *causal sequential composition* $P\mathrm{s}Q$ of processes P and Q combines them so that the process $P\mathrm{s}Q$ contains all events from P and Q and relations between them, any events a from P and b from Q, we have $a\mathrm{OD}_{\mathrm{R}}b$, and the first process transmits information to the second one.

It is natural to make a distinction between data and control information (instructions, commands, etc.) This separates the causal sequential composition into three subclasses: data-driven, control-incited and combined sequential composition. *Data-driven composition* is given in Definition 18.

Definition 25. The *control-incited sequential composition* $P\mathrm{s}Q$ of processes P and Q combines them so that the process $P\mathrm{s}Q$ contains all events from P and Q and relations between them, any events a from P and b from Q, we have $a\mathrm{OD}_{\mathrm{R}}b$, and the first process sends some control information (e.g., an instruction or rule) that is used in the second process.

It is possible that the second process uses both data and control information from the first process. In this case, we have the *combined sequential composition*. The sequential composition is related to sequential processes:

Definition 26. A process P is called *sequential* if for any two events a and b from P, we have $b\mathrm{OD}_{\mathrm{R}}a$ or $a\mathrm{OD}_{\mathrm{R}}b$ – i.e. one event from any two precedes the other one.

Lemma 1. Any finite sequential process has the first and last events.

Proposition 18. Any subprocess of a sequential process is a sequential process.

Definition 27. Two events a and b are *directly sequentially connected* if $a\mathrm{OD}_{\mathrm{R}}b$ and there is no event c such that relations $a\mathrm{OD}_{\mathrm{R}}c$ and $c\mathrm{OD}_{\mathrm{R}}b$ are true.

Proposition 19. Any sequential process is a sequence of its directly sequentially connected events.

Definition 28. A subprocess Q of a sequential process P is called *complete* if for any two events a and b from Q, relations $a\mathrm{OD}_{\mathrm{R}}c$ or $c\mathrm{OD}_{\mathrm{R}}b$ imply that c also belongs to Q.

Proposition 20. Any sequential process is the sequential composition of, at least, two of its complete subprocesses.

Theorem 2. The weak sequential composition of two sequential processes is a sequential process.

As any sequential composition of processes is also the weak sequential composition, we have the following result:

Corollary 8. The sequential composition of two sequential processes is a sequential process.

This gives us an algebraic structure on the set of all sequential processes.

Corollary 9. The set of all sequential processes (in some system) is a semigroup with respect to weak sequential composition.

Remark 4. The above result if valid for any composition operator (not all of which are associative).

Definition 29. The *parallel composition* $P\mathbf{par}Q$ of processes P and Q combines them so that (1) the process $P\mathbf{par}Q$ contains all events from P and Q and relations between them and (2) any event a from P is coexisting to some event b from Q (i.e. we have $a\mathrm{CE}_\mathrm{R}b$).

Proposition 21. Any parallel composition is exact.

Proposition 22. If R is a subprocess of P, then $R\mathbf{par}Q$ is a subprocess of $P\mathbf{par}Q$ (for any process Q).

Remark 5. However, if R is a subprocess of Q, then $P\mathbf{par}R$ is not necessarily a subprocess of $P\mathbf{par}Q$ (for any process P). This shows that parallel composition of processes is not symmetric.

Definition 30. The composition $P\mathbf{par}Q$ of processes P and Q is *strictly parallel* if (1) the process $P\mathbf{par}Q$ contains all events from P and Q and relations between them, (2) any event b from Q is coexisting to some event a from P (i.e. we have $b\mathrm{CE}_\mathrm{R}a$), and (3) any event a from P is coexisting to some event b from Q (i.e. we have $a\mathrm{CE}_\mathrm{R}b$).

Lemma 2. Any strictly parallel composition is a parallel composition.

This means that the strictly parallel composition inherits many properties of the parallel composition:

Proposition 23. Any strictly parallel composition **par** of processes is associative: i.e. for any processes P, R and Q, we have $(P\mathbf{par}Q)\mathbf{par}R = P\mathbf{par}(Q\mathbf{par}R)$.

Remark 6. It is possible to show that sequential and parallel compositions of processes are induced by temporal sequential and parallel compositions of systems. Temporal sequential composition allows one to build sequential composition of system with itself.

Parallel composition of processes corresponds to the logical operation "and". It is also possible to build a composition of processes that corresponds to the logical operation "or." It is called selective composition.

Let $\mathbf{P}(x)$ be a predicate (function that takes value 0 and 1):

Definition 31. The *selective composition* $\mathbf{P}(x)(P \mid Q)$ of processes P and Q is equal to the process P when $\mathbf{P}(x) = 1$ and equal to the process Q when $\mathbf{P}(x) = 0$.

Events can be *elementary* and *complex*. Complex events consist of other events. Thus, it is possible to consider a process as complex event. Moreover, any set of events can formally be considered as one complex event when we ignore relations between the initial events.

Proposition 24. If $a\mathrm{IC}_\mathrm{R}b$ and $a\mathrm{IC}_\mathrm{R}c$, then $b\mathrm{CE}_\mathrm{R}c$.

Corollary 10. If a is an elementary event, $a\mathrm{CE}_\mathrm{R}b$ and $a\mathrm{CE}_\mathrm{R}c$, then $b\mathrm{CE}_\mathrm{R}c$.

Proposition 7 implies the following result:

Corollary 11. Coexisting elementary events are simultaneous.

Proposition 25. An event simultaneous with an elementary (for a set E) event is itself elementary (for E).

Definition 32. A faithful composition **c** of processes P and Q is *quasi-parallel* if the processes P and Q considered as complex events are parallel events in **c**(P, Q).

Remark 7. It is not always that a quasi-parallel composition of two processes is parallel and vice versa. If a composition **c** is quasi-parallel, it means that finite processes P and Q start and end at the same time. It gives us the following result.

Proposition 26. If a composition **c** is quasi-parallel, then the first events from P and Q are simultaneous and the last events from P and Q are simultaneous.

There are many other kinds of process composition, one of which is considered in the next Section.

All introduced constructions of EVCR allow one to build VCR as submodel of EVCR. In that submodel, only the simultaneity relation ST_R is considered because all events are instantaneous.

4. Composition of I/O-PAR Processes

Why devote so much time to exploring so many different types of composition? One reason is that thinking deeply about composition in general may lead to new ways of reasoning about known properties of specific types of composition. For example, consider the composition of I/O-PAR processes, studied by Welch, Martin, and others [8, 9, 15, 16]. Welch proved that a network of I/O-PAR processes is deadlock-free, and described composite-I/O-PAR processes, that could in turn be used in further compositions. (The I/O-SEQ design pattern for processes is also presented, but not focused on for the purposes of this discussion.) The proofs of deadlock freedom and that I/O-PAR is closed under composition are not simple, and reasoning about the closure of I/O-PAR under composition from a process's traces is challenging. We illustrate this point, and then present a definition of composition that, in conjunction with EVCR, permits identifying a composite-I/O-PAR process from its trace.

An I/O-PAR process is one that behaves deterministically and cyclically, by first engaging in all the events of its alphabet, then repeating this behavior. For example, consider the I/O-PAR process P, where:

$$P = (a \rightarrow SKIP \;|||\; b \rightarrow SKIP) \; ; \; P$$

Process P's behavior, represented by its set of all possible traces, *traces(P)*, can be succinctly described by the following VCR-style, parallel event trace:

$$tr_P = < \{a, b\}, \{a, b\}, \{a, b\}, \ldots >$$

The benefits of the VCR-style representation of P's behavior are evident. In particular, the set of views of this trace all adhere to the definition of an I/O-PAR process. Now, suppose we wished to reason about I/O-PAR process Q, whose specification and respective parallel event trace is as follows:

$$Q = (b \rightarrow SKIP \;|||\; c \rightarrow SKIP) \; ; \; Q$$

$$tr_Q = < \{b, c\}, \{b, c\}, \{b, c\}, \ldots >$$

Now we have two I/O-PAR processes. It is natural to wish to compose *P* and *Q* in such a way that the composition of *P* and *Q* is also I/O-PAR (technically, composite-I/O-PAR). Notice, though, that *P* and *Q* both engage in event *b*, which implies that *b* is a synchronizing event when *P* and *Q* are composed in parallel. In particular, we'd like to be able to compose *P* and *Q* in such a way that the *traces*(*P* || *Q*) could be described by *tr1*, expressed as a VCR-style parallel event trace:

tr1 = < {*a*, *b*, *c*}, {*a*, *b*, *c*}, {*a*, *b*, *c*}, … >

Such a trace clearly reflects an I/O-PAR process, and traces like *tr1* are possible; but the problem is that under existing parallel composition in CSP, other traces are also possible. It is these other possible traces that make the composition of processes *P* and *Q* difficult to recognize as being I/O-PAR. The existence of these traces also provides some intuition into the difficulty in proving I/O-PAR processes are closed under composition.

Suppose process *P* completes one iteration of its events, then recurses before process *Q* can engage in all of its events. At this point, process *Q* is still waiting to engage in its first event *c*, even though process *P* has engaged in its second event *a*. However, process *P* can't get any further ahead of *Q*, because it's waiting for the synchronizing event *b*, which won't occur until process *Q* first engages in its first event *c*, then recurses. Now suppose *Q* "catches up" to *P*, by engaging in *c* and recursing. At this point, *Q* may engage in either event of its alphabet, since process P is already waiting to engage in the synchronizing event *b*. Suppose processes P and Q engage in synchronizing event b, then *c*. At this point, both processes have completed two cycles of engaging in all the events in their respective alphabets, which results in the following CSP-style trace:

tr2 = < *a*, *b*, *a*, *c*, *b*, *c*, ... >

Trace *tr2* isn't readily recognizable as I/O-PAR, but Martin proved that such a trace of a composite-I/O-PAR process is permissible, and still qualifies as I/O-PAR. However, it is easy to imagine that a few more processes and larger alphabets would make such compositions difficult to recognize as being I/O-PAR. Even the parallel event traces of VCR do not provide much help in this regard:

tr3 = < {*a*, *b*}, {*a*, *c*}, {*b*, *c*}, … >

Trace *tr3* no longer emulates the I/O-PAR design pattern behavior as shown in *tr1*. In particular, it's possible to observe two *a*'s by the time the first *c* is observed. Strictly speaking, for this process to be I/O-PAR, events *a*, *b*, and *c* must each occur once before any of *a*, *b*, or *c* occurs for a second time. However, Martin proved that a limited amount of "slush" is possible, without losing the benefits of deadlock-freedom. In this sense, the process that is capable of producing trace *tr2* is I/O-PAR "enough."

An interesting question is what type of composition could preserve the I/O-PAR property as evident by such a composite-I/O-PAR process's traces? We describe informally how to compose such a process, and in so doing, introduce a new type of composition. The goal of such a composition is to result in a process whose behaviour matches process *R0*:

R0 = (*a* ➔ *SKIP* ||| *b* ➔ *SKIP* ||| *c* ➔ *SKIP*) ; *R0*

Process *R0* does not reflect the composition of processes *P* and *Q*, however. But consider process *R1*, which is a little closer to the desired composition:

R1 = (*a* ➔ *SKIP* ||| *b* ➔ *SKIP*) |{*b*}| (*b* ➔ *SKIP* ||| *c* ➔ *SKIP*) ; *R1*

Process R1 is a little closer in spirit to achieving our goal, since its definition reflects the composition of the definitions of *P* and *Q* (without the recursive invocations of *P* and *Q*). But if we are to define a type of composition for processes, we must be able to treat the processes as black boxes, or at least not resort to changing the definitions (bodies) of the processes being composed. Consider the naïve composition:

$R2 = P \ |\{b\}|\ Q$

This is the standard CSP parallel alphabetized composition, and this is the composition that results in traces like *tr1* and *tr2*. Processes *R0* and *R1* are refinements of each other, because they both produce the same set of traces, *tr1*. Furthermore, processes *R0* and *R1* both represent a refinement of *R2*, since *R2* is capable of producing all the traces of *R0* and *R1* , as well as other possible traces (the traces with some limited "slush").

How then can we compose processes *P* and *Q* so that only traces like *tr1* are possible? If we could solve this problem, we will have defined a type of composition that preserves the I/O-PAR property.

We begin by renaming processes *P* and *Q* to *P'* and *Q'*, respectively. That is, *P'* and *Q'*, are bound to the original definitions of *P* and *Q*. In particular, the recursions still refer to the original process names, *P* and *Q*, within the bodies of *P'* and *Q'*. Our goal is to remove the "slush," by synchronizing the recursive calls of the I/O-PAR processes being composed. So far, we have:

$P' = (a \rightarrow SKIP \ ||| \ b \rightarrow SKIP) ; P$

$Q' = (b \rightarrow SKIP \ ||| \ c \rightarrow SKIP) ; Q$

Notice that *P* and *Q* are no longer recursive calls, since we renamed these processes. We therefore need to provide new definitions for *P* and *Q*. The new definitions for *P* and *Q* are trivial and recursively symmetrical:

$P = x \rightarrow P'$ (where the synchronizing event *x* is not in the alphabets of *P* or *Q*)

$Q = x \rightarrow Q'$

In this way, processes *P'* and *Q'* must synchronize on *x* before recursing. Now let us define the composite-I/O-PAR process *R3* to be the alphabetized parallel composition of *P'* and *Q'*:

$R3 = (P' \ |\{b, x\}|\ Q') \setminus \{x\}$

which specifies the parallel composition of processes *P'* and *Q'*, where events *b* and *x* are synchronizing events. Furthermore, we are hiding event *x* from appearing in the *traces(R)*. The resulting *traces(R)*, expressed in the VCR style, now look like *tr1*, which is recognizably I/O-PAR. It is easy to see that this composition nests, i.e., it extends to three or more processes.

We have exploited renaming to achieve our goal, though it is a renaming of processes, rather than events, which is somewhat unusual in CSP. Still, this approach is not entirely unrelated to the approach used to create multiple instances of a process, each with its own unique instance of the events it engages in. The renaming of processes models what is possible on a real computer, e.g., renaming the games "rogue" or "hack" to "vi" on a Unix system (to appear to one's boss, or advisor, that you are hard at work!); or even renaming a compiled C program from *a.out* to a more meaningful executable name. The renaming of

the original processes being composed allowed us to introduce new definitions for the original process names, and thus "insert" a synchronizing event into the recursion of the processes being composed, without modifying the contents of the black boxes.

We haven't named this type of composition yet, but we know enough about it to describe its properties. Since processes P and P', and Q and Q', are mutually recursive and synchronizing, this type of composition is a *synchronized, mutually recursive renaming composition*. But it seems more appropriate to name this type of composition, I/O-PAR composition.

A more thorough and formal treatment of I/O-PAR composition remains to be done, including composition of three or more processes, and a formal proof of refinement, showing that process *R3* refines *R0* (and vice-versa). In addition, we did not address I/O-SEQ, another important design pattern used in addition to I/O-PAR to ensure deadlock freedom.

5. Conclusions and Future Work

We continued the development of extended view-centric reasoning (EVCR) presented in [3] with compositions of concurrent processes. The new definitions of events, and the many ways events may coincide with other events, borrow abstractions from Grid Automata [2], and form the basis for EVCR as a model that provides abstractions needed for reasoning about properties of modern computational systems. In particular, EVCR provides for the possibility of events A and B to overlap in varying degrees, instead of the all-or-nothing simultaneity of VCR's (and CSP's) instantaneous events. This added dimension of event duration makes EVCR a more natural model for reasoning about Grid Automata and concurrent processes, which must take time and location into account.

We constructed compositions of processes on a system level and studied several kinds of compositions. Doing this, we have only laid the foundation for a model that holds much promise. To support this claim, we considered a case study of I/O-PAR composition, and demonstrated that EVCR provides the basis for a much simpler proof of the compositionality of I/O-PAR processes. A formal statement and proof remain to be shown. The next step in this direction will be to continue to build a comprehensive model of process composition.

Another prospective direction for the further development of EVCR is the automation of commonsense reasoning, a long goal of the field of artificial intelligence. EVCR provides a flexible base for enhancing the event calculus, which is used as an effective technique for commonsense reasoning [6, 10].

Introduced constructions, defined concepts, and obtained results are steps toward the development of a fundamental, adequate, and efficient theory of compositions of systems and processes. Such a theory has to be developed on three levels. The first, organizational or algorithmic level would deal with organization of connections and interactions between composed systems and processes. The second, algebraic level would expand over algebras of compositions. There are two types of such algebras: algebras of systems (processes), in which compositions play the role of operations, and algebras of compositions themselves, in which compositions play the role of elements. These two levels are necessary for the development of reliable, safe, and efficient hardware and software systems from well-designed components. The third, logical level of the theory of compositions of systems and processes would provide means for reasoning about composed systems and processes. This is necessary for assurance and verification of compounded systems.

Acknowledgements

The authors are very grateful to the anonymous CPA referees for their diligent and constructive feedback on our original submission and all the subsequent revisions. Their work is greatly valued.

References

[1] M. Burgin, Elements of the System Theory of Time. LANL, Preprint in Physics 0207055, 2002, 21 p. (electronic edition: **http://arXiv.org**).
[2] M. Burgin, Super-recursive Algorithms. Springer, New York, 2005.
[3] M. Burgin and M.L. Smith, From Sequential Processes to Grid Computation. In Proceedings of the 2006 International Conference on Foundations of Computer Science (FCS'06), Edited by Hamid R. Arabnia and Rose Joshua, CSREA Press, Las Vegas, Nevada, USA, 26-29 June 2006.
[4] C.A.R. Hoare, Communicating Sequential Processes. Prentice Hall International Series in Computer Science. UK: Prentice-Hall International, UK, Ltd., 1985.
[5] J.E. Hopcroft, R. Motwani, and J.D. Ullman, Introduction to Automata Theory, Languages, and Computation. Addison Wesley, Boston/San Francisco/New York, 2001.
[6] R. Kowalski, and M. J. Sergot, (1986) A logic-based calculus of events. New Generation Computing, v. 4, pp. 67-95.
[7] W. Li, S.Ma, Y. Sui, and K. Xu, (2001) A Logical Framework for Convergent Infinite Computations. Preprint cs.LO/0105020 (electronic edition: http://arXiv.org).
[8] J.M.R. Martin, I. East, and S, Jassim. (1994) Design Rules for Deadlock Freedom. Transputer Communications, 3(2):121–133, John Wiley and Sons
[9] J.M.R. Martin, and P.H. Welch, (1996) A Design Strategy for Deadlock-Free Concurrent Systems. Transputer Communications, 3(4):215–232, John Wiley and Sons
[10] E.T. Mueller, Commonsense reasoning. San Francisco, Morgan Kaufmann, 2006.
[11] P.G. Neumann, (2006) Risks relating to System Compositions. Communications of the ACM, v. 49, No. 1, p. 128.
[12] M.O. Rabin, (1969) Decidability of Second-order Theories and Automata on Infinite Trees. Transactions of the AMS, v. 141, pp. 1-35.
[13] M.L. Smith, View-Centric Reasoning about Parallel and Distributed Computation. PhD thesis, University of Central Florida, Orlando, FL 32816-2362, December 2000.
[14] M.Y. Vardi and P. Wolper, (1994) Reasoning about Infinite Computations. Information and Computation, v. 115, No.1, pp. 1—37.
[15] P.H. Welch, (1989) Emulating Digital Logic using Transputer Networks (Very High Parallelism = Simplicity = Performance). International Journal of Parallel Computing, 9, North-Holland.
[16] P.H. Welch, G.R.R. Justo, and C.J. Willcock, Higher-Level Paradigms for Deadlock-Free High-Performance Systems. In R. Grebe, J. Hektor, S.C. Hilton, M.R. Jane, and P.H.Welch, editors, Transputer Applications and Systems '93, Proceedings of the 1993 World Transputer Congress, volume 2, pages 981–1004, Aachen, Germany, September 1993. IOS Press, Netherlands.