Journal of Universal Computer Science, vol. 10, no. 10 (2004), 1429-1468
submitted: 9/6/04, accepted: 15/10/04, appeared: 28/10/04 © *J.UCS*

Synchronization Can Improve Reactive Systems Control and Modularity

Cristina Cerschi Seceleanu
(Turku Centre for Computer Science and Åbo Akademi, Turku, Finland
ccerschi@abo.fi)

Tiberiu Seceleanu
(University of Turku, Department of Information Technology, Turku, Finland
tiberiu.seceleanu@utu.fi)

Abstract: We concentrate on two major aspects of reactive system design: behavior control and modularity. These are studied from a formal point of view, within the framework of action systems. The traditional interleaving paradigm is completed with a barrier synchronization mechanism. This is achieved by introducing a new parallel composition operator, applicable to both discrete and hybrid models. While offering improvements with respect to control and modularity, the approach uses the correctness preserving mechanisms provided by the underlying reasoning environment.[1]

Key Words: Reactive systems, Action systems, Modular design, Concurrency
Category: F.3.1, F.3.2, D.1.3

1 Introduction

The global behavior of a reactive system results from the cooperation of its components. Hence, designing for reactivity entails dealing with communication, composability, concurrency and preemption. Out of these, concurrency is related to the fundamental aspects of systems with *multiple*, *simultaneously active* computing agents that interact with one another. The complexity of such systems comes as an inherent byproduct, which leads further to problems concerning the correctness of the steps performed in the development flow. On one hand, component-based design is a solution towards partially reducing the task of the designer of complex systems, while on the other hand, employing formal methods in system design tries to solve the aspects related to correctness.

In this study, we tackle problems regarding the design of reactive systems, be they software or hardware-targeted systems. A feasible design methodology requires the designer to compose the system from parallel concurrent components called *modules*. Such modules are modeled here by *action systems*. We approach aspects of concurrency and modular design from the perspective of the system–level integrator that has

[1] A shorter version of this study appeared as "Modular Design of Reactive Systems", in Proceedings of the 28^{th} Annual International Computer Software and Applications Conference (COMPSAC 2004), IEEE Computer Society Press, September 2004, Hong Kong. Pages 265-271.

access to a library of predefined subsystems. His only task is to appropriately connect them in order to obtain the system functionality.

Action systems, introduced by Back and Kurki-Suonio [Back and Suonio 1988], is a state-based formalism, relying on an extended version of Dijkstra's language of *guarded commands* [Dijkstra 1976]. Recently, the formalism has been extended to *continuous action systems* [Back et al. 2001], which provides a unified framework for handling both discrete and continuous behavior. The higher-order logic of the *refinement calculus* [Back and von Wright 1998] is used for reasoning about properties of action systems and proving that the correctness of the specifications of abstract modules are preserved by their implementations.

Our favorite formalism uses a built-in interleaving semantics for handling concurrency. Parallel behavior is modeled by interleaved actions that can be executed in any order. This approach goes together with behavioral nondeterminism, as observations of an interleaved model are sequential, therefore the updates of two systems executing in parallel may not be consistent over a set of executions [Montanari and Rossi 1995]. Hence, though versatile and general, this way of modeling large systems can have a negative impact on the data flow control and the *composability* of the modules that interact concurrently. When plugging modules together, we have to specify additional details about the order in which they exchange information. This requirement may compromise the data abstraction of the interface of a module. We will illustrate these symptoms by examples.

In this paper, we provide a solution to the above mentioned problems, by introducing an additional concurrency mechanism for action systems, namely *barrier synchronization*, as a way to describe controllable behavior. For this purpose, we define a new parallel composition operator. We show that the concepts that we formulate continue to rely on the established mathematical techniques of action systems, while providing the designer with additional means for system development. We also extend the new method to continuous action systems, since concurrency, in all its flavors, plays a crucial role in real-time / hybrid systems design, too. Moreover, for the hybrid case, we propose a new semantics for continuous action systems, which guarantees the absence of *timelocks*, both at the action and system level. Fortunately, our goal is completed by showing that the new virtual execution environment also enhances the capabilities of our framework, for modular design. Components may be picked up from existing libraries and just plugged into the system representation. The traditional techniques of *trace refinement* [Back and von Wright 1994] are used to ensure that the implementation is correct with respect to a specification that faithfully captures the system's global reaction to all sets of inputs.

We believe that the result of our contribution, that is, the positive effect of the proposed synchronized environments on the design modularity, could also be applied to other formal frameworks, to obtain the same benefits.

Related work. The approximation of concurrency by interleaving is used in most process algebras like CSP [Hoare 1984], CCS [Milner 1989], as well as in input-output automata [Lynch and Tuttle 1989] and UNITY [Chandy and Misra 1988]. The nondeterministic behavior induced by the interleaved model requires solutions for controlling the data flow. However, resolving control issues reduces the design independence across the different levels of the design process. Several recent studies have analyzed aspects of control and / or composability within different formal frameworks, all of which deal with a certain interleaved environment.

The motivation behind the product operator of Milner's Synchronous Calculus of Communicating Systems (SCCS) [Milner 1983] is the same as ours, that is, the system response to stimuli is the composition of the individual reactions of the included subsystems. Up to a point, our approach resembles the semantics of this operator. However, in the following sections, the differences become apparent.

The study presented by Treharne and Schneider [Treharne and Schneider 1999] employs CSP processes to control B-machines. Butler [Butler 2000] analyzes the impact of a similar combination on composition related issues. In the latter, synchronization aspects appear when discussing composability of interacting CSP processes only.

Bellegarde et al. analyze modularity, as a first concern [Bellegarde et al. 2002], and define the synchronized parallel composition of event B [Abrial 1996] structures. Synchronization may be specified only for a subset of events. Except for the synchronization idea, this approach has several common characteristics with the one taken by Back and von Wright for action systems, where modularity is analyzed in terms of the usual parallel composition [Back and von Wright 2003].

Parallel composition of hybrid models has also been studied extensively. In the temporal logic of actions (TLA) [Lamport 2002], barrier synchronization is specified as a way of applying *non-interleaving* to system design, since the author considers that interleaving "blurs" the distinction between the components used in design. The composition of timed systems expressed as communicating processes is also analyzed by Bornot and Sifakis, who strive for *maximal progress* [Bornot and Sifakis 1998]. In our case, the latter is ensured by the synchronized semantics that we propose.

In our view on reactive systems design, we subscribe to the idea that, while *internally*, components may exhibit complex, nondeterministic behaviors, for an observer, only the *external* reaction is relevant [Olderog 1989]. Therefore, in the following, we show how we decrease the external nondeterminism by providing a barrier synchronization mechanism for action systems. This will also help us to improve modularity characteristics of the design process, in our framework.

The rest of the paper is organized as follows. In section 2, we briefly give an overview of the action systems formalism. The execution of action systems, the interleaving model for parallel composition and an illustrative example are outlined in section 3. Following this, in section 4, we introduce our synchronized model, by defining the new parallel operator. Refinement notions and their application to both execution models are illustrated in section 5. Further, we extend the model of synchronized parallel action systems to continuous action systems, in section 6. In section 7, we discuss our contribution and show some comparison to other similar approaches. All the proofs of new theorems and lemmas are given in the Appendix.

2 Action Systems

Back and Kurki-Suonio proposed the action systems formalism, as a framework for specifying and refining concurrent programs [Back and Suonio 1988]. An *action system* (AS) is in general a collection of *actions* or guarded commands, which are executed one at a time.

An AS is built according to the following syntax:

$$\mathcal{A}(z : T_z) \triangleq \mathsf{begin\ var}\ x : T_x \ \bullet \ Init\,;\ \mathsf{do}\ A_1 [\!] \ \ldots [\!]\, A_n\ \mathsf{od}\ \ \mathsf{end} \qquad (1)$$

Here, $\mathcal{A}$ contains the declaration of *local* variables x (of type T_x), followed by an *initialization* statement $Init$ and the *actions* $A_1, \ldots, A_n$. Variables z (of type T_z) are *global* to the action system. The initialization statement assigns starting values to the global and local variables. After that, *enabled* actions are repeatedly chosen and executed. In this paper, we consider an action A_i as being of the form $g_i \rightarrow S_i$. Thus, A_i is *enabled* and its *body* S_i is executed, when the *guard* g_i evaluates to true. The chosen actions change the values of the variables in a way that is determined by the action body.

An action system is not usually regarded in isolation, but rather as a part of a more complex structure, the rest of which, that is, the *environment*, communicates with the action system via *shared* (read and written) variables. In the following, we assume and extend the notations defined in [Back and Sere 1994]. The set of state variables accessed by some action A is denoted vA, and is composed of the *read* variable set of action A, denoted rA, and the *write* variable set of action A, denoted wA. We have that $vA = rA \cup wA$. We can also build the same sets at the system level, considering the local / global partition of the variables. Thus, for a given action system $\mathcal{A}$, we have the access set, $v\mathcal{A}$, split into the global read / write variables, denoted by $gr\mathcal{A}/gw\mathcal{A}$ and the local read / write variables, denoted by $lr\mathcal{A}/lw\mathcal{A}$. We say that an action A of $\mathcal{A}$ is *global*, if $gw\mathcal{A} \cap wA \neq \emptyset$ or *local*, if $wA \subseteq lw\mathcal{A}$.

A statement S_i is defined by the following grammar:

$$\begin{array}{rll} S_i ::= & \mathsf{skip} & (stuttering,\ empty\ statement) \\ & \mid x := e & ((multiple)\ assignment) \\ & \mid S_m\,; \ldots ; S_n & (sequential\ composition) \\ & \mid g_m \rightarrow S_m [\!]\ \ldots [\!]\ g_n \rightarrow S_n & (nondeterministic\ choice) \\ & \mid x := x'.Q & (nondeterministic\ assignment), \end{array}$$

where $S_m, \ldots, S_n$ are statements, $g_m, \ldots, g_n$ and Q are predicates (boolean conditions), x a variable or a list of variables, and e an expression or a list of expressions. Actions can be much more general, but this simple syntax suffices for the purpose of this paper. A *loop* can be reduced to iterations [Back and von Wright 1998]. Therefore, the *while* loop is written as while g do S od $=$ do $g \rightarrow S$ od. In this paper, we do not consider nested loops.

Statements in AS are defined by the *weakest precondition semantics*, consistent with Dijkstra's original semantics for the language of guarded commands [Dijkstra 1976]. For statement S and postcondition Q, the formula $\mathrm{wp}(S, Q)$, called the weakest precondition of S with respect to Q, gives the largest set of initial states from which statement S is guaranteed to terminate in a state satisfying Q. Here, we assume that all statements are *conjunctive predicate transformers* (functions from predicates to predicates), that is, $\forall p, q \bullet \mathrm{wp}(S, (p \wedge q)) = \mathrm{wp}(S, p) \wedge \mathrm{wp}(S, q)$. As an example, the wp of the nondeterministic choice of n conjunctive predicate transformers is given as:

$$\mathrm{wp}(g_1 \rightarrow S_1 \,[\!]\, \ldots \,[\!]\, g_n \rightarrow S_n, Q) \;\hat{=}\; g_1 \Rightarrow \mathrm{wp}(S_1, Q) \wedge \ldots \wedge g_n \Rightarrow \mathrm{wp}(S_n, Q)$$

For statement S_i, $\mathrm{wp}(S_i, false)$ represents the set of initial states for which S_i is guaranteed to establish any postcondition (even $false$), that is, behave miraculously. We take the view that a statement is enabled only in those initial states in which it behaves non-miraculously. Therefore, the guard of S_i is defined as $g_i \;\hat{=}\; \neg\mathrm{wp}(S_i, false)$.

When at least one action is enabled in a given AS $\mathcal{A}$, we say that $\mathcal{A}$ is enabled. We obtain information about the enabledness of a system $\mathcal{A}$ given by (1), by evaluating the predicate gg_A, where $gg_A \;\hat{=}\; \bigvee_{k=1}^{n} g_k$.

3 Execution of Action Systems: the Traditional Model

As stated in the original paper of Back and Kurki-Suonio [Back and Suonio 1988], AS are executed in a sequential manner. Parallel executions are modeled by interleaving actions.

Execution of Action Systems. The initialization places the systems in a stable, starting state. From there, any enabled action may be selected for execution. Only one action is chosen, in a (demonically) nondeterministic way. The statements inside the do $-$ od loop of a system $\mathcal{A}$, as illustrated by (1), are iterated as long as $gg_A \equiv true$. Termination is normal if the exit condition ($\neg gg_A$) holds.

Thus, the execution of an AS assumes that there exists a virtual external entity - the **execution controller** (**controller** in short) - which, at any moment, knows what actions are enabled. After initialization, the controller, nondeterministically, selects for execution, any of the enabled actions. After the completion of the action execution, the system moves to a new state. We call this operation an *execution round*. Notice that an execution round is equivalent to the execution of an action. After this, the controller evaluates the new state, observes the enabled actions and starts another execution round.

3.1 Parallel Composition of Action Systems

Let our protagonists be $\mathcal{A}$ and $\mathcal{B}$, two action systems of the form

$$\mathcal{A}(z_A) \;\hat{=}\; \mathsf{begin\ var}\ x_A \;\bullet\; Init_A\,;\ \mathsf{do}\ g_A \rightarrow S_A\ \mathsf{od}\ \ \mathsf{end}$$

$$\mathcal{B}(z_B) \triangleq \mathsf{begin\ var}\ x_B \bullet Init_B\ ;\ \mathsf{do}\ g_B \rightarrow S_B\ \mathsf{od\ end}$$

Then, the parallel composition [Back 1990] of $\mathcal{A}$ and $\mathcal{B}$ is the system $\mathcal{P} = \mathcal{A} \,||\, \mathcal{B}$:

$$\mathcal{P}(z_P) \triangleq \mathsf{begin\ var}\ x_P \bullet Init_A\ ; Init_B\ ;\ \mathsf{do}\ g_A \rightarrow S_A [\!] \ g_B \rightarrow S_B\ \mathsf{od\ end}$$

The composed action system essentially combines the variables, the initialization statements and the actions of the two subsystems. The initialization of the common variables z must be consistent, that is, they are assigned the same initial values by both initialization statements, $Init_A$ and $Init_B$. Some of the previously global variables of $\mathcal{A}$ and $\mathcal{B}$ may become local variables of $\mathcal{P}$. We add these to the reunion of the individual local variables of $\mathcal{A}$ and $\mathcal{B}$, thus obtaining the set of local variables of $\mathcal{P}$, x_P. The global variables z_P are defined as $z_P \triangleq z_A \cup z_B - x_P$.

Given the above formalization, $\mathcal{A} \,||\, \mathcal{B}$ is executed by first initializing the local and global variables, and then interleaving the execution of the enabled actions of $\mathcal{A}$ and $\mathcal{B}$. Termination occurs when both action systems terminate, which means that there is no enabled action, in neither of the systems, that is, $gg_A \vee gg_B \equiv \mathit{false}$. In short, the controller observes the composition as the single system $\mathcal{P}$.

3.2 Example: A Digital Filter

Let us illustrate the interleaved execution model by a simple, yet relevant example. We consider the task of modeling a digital *filter* [Ifeachor and Jervis 1997]. Briefly, a filter is a module that takes as input a sequence of samples, performs certain operations on it and delivers as output a corresponding sequence of samples. The incoming sequence is described as $X[n]$, where X is the input signal and n identifies the sample position; a similar notation applies to the output signal Y, for which we have the samples $Y[n]$. The relation between the input and output is given by $Y[n] = \sum_{k=0}^{N-1} h[k] \times X[n-k]$, where the vector $h[0..N-1]$ contains the filter *coefficients*. Hence, apart from the incoming current sample of X, $N-1$ previous samples are stored in a buffer and can be accessed by the filter. Finally, a filter may have either a software or a hardware implementation.

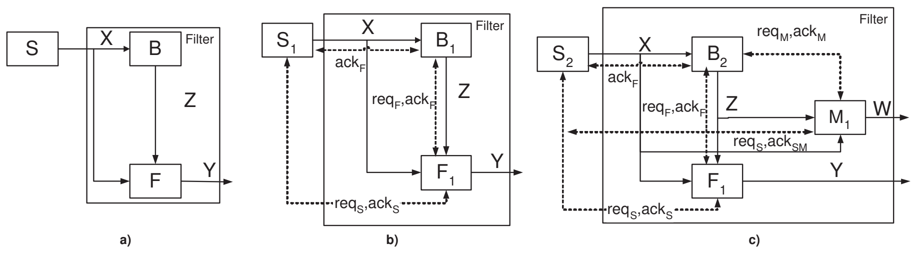

Figure 1: Simple filter representation.

From the above informal description of the filter we can identify two submodules of such a device: the storage First-In-First-Out (FIFO)-like buffer, and the actual implementation of the filter functionality. In the following, we model the signal source by system $\mathcal{S}$, the buffer by system $\mathcal{B}$, whereas system $\mathcal{F}$ models the actual filter. This is illustrated in Figure 1 a). The complete AS description, that is, $\mathcal{P} \triangleq \mathcal{S} \parallel \mathcal{B} \parallel \mathcal{F}$, and the composition elements are given in Figure 2.

$$
\begin{array}{ll}
& \mathcal{S}(X:T) \\
\triangleq & \textsf{begin} \quad \bullet\ X := x_0; \\
& \quad \textsf{do}\ X := X'.(X' \in T)\ \textsf{od} \\
& \textsf{end}
\end{array}
\qquad
\begin{array}{ll}
& \mathcal{B}(X, Z[0..N-2]:T) \\
\triangleq & \textsf{begin} \quad \bullet\ X, Z[0..N-2] := x_0, z_0; \\
& \quad \textsf{do}\ Z[0],..,Z[N-2] := X,..,Z[N-3]\ \textsf{od} \\
& \textsf{end}
\end{array}
$$

$$
\begin{array}{ll}
& \mathcal{F}(X, Z[0..N-2], Y:T) \\
\triangleq & \textsf{begin var}\ h[0..N-1]:T \quad \bullet \\
& \qquad X, Z[0..N-2], h[0..N-1], Y := x_0, z_0, h_0, y_0; \\
& \quad \textsf{do}\ Y := \sum_{k=1}^{N-1} h[k] \times Z[k-1] + h[0] \times X\ \textsf{od} \\
& \textsf{end}
\end{array}
$$

$$
\begin{array}{ll}
& \mathcal{P}(Y:T) \\
\triangleq & \textsf{begin var}\ X, Z[0..N-2], h[0..N-1]:T \ \bullet \\
& \qquad X, Z[0..N-2], h[0..N-1], Y := x_0, z_0, h_0, y_0; \\
& \quad \textsf{do}\ X := X'.(X' \in T) \\
& \qquad [\!]\ Z[0],..,Z[N-2] := X,..,Z[N-3] \\
& \qquad [\!]\ Y := \sum_{k=1}^{N-1} h[k] \times Z[k-1] + h[0] \times X \\
& \quad \textsf{od} \\
& \textsf{end}
\end{array}
$$

Figure 2: Filter model.

Observe first that an interleaved execution of $\mathcal{P}$ would not ensure that every signal emitted by $\mathcal{S}$ is correspondingly received by $\mathcal{B}$ and $\mathcal{F}$: several executions of $\mathcal{S}$ may be selected, before any of $\mathcal{B}$ or $\mathcal{F}$. Moreover, different values can be assigned to Y for the same sequence of samples provided by $\mathcal{S}$, depending on the order of selecting for execution the systems $\mathcal{B}$ and $\mathcal{F}$. Only one of these results is the correct one.

Both problems may be solved by specifying a certain order in which the submodules of $\mathcal{P}$ should be executed. This can be achieved by introducing the communication variables req_S, req_F and ack_S, ack_F, and by devising a communication protocol such that the desired order is enforced. Hence, the systems should know about the status of the partners, as indicated by the elements of the communication channel. The systems may be remodeled as in Figure 3, resulting in the block diagram described by Figure 1 b), where the communication variables are shown as dotted lines. Notice that after emitting one signal sample, the system $\mathcal{S}_1$ is deactivated, after which $\mathcal{F}_1$ performs the filtering, and then informs $\mathcal{B}_1$, by setting $req_F := true$, that it can perform its operation. When this is accomplished, $\mathcal{B}_1$ signals to $\mathcal{F}_1$ ($ack_F := true$). Next, $\mathcal{F}_1$ signals the end of the operation, to $\mathcal{S}_1$ ($ack_S := true$), followed by a couple of rounds for resetting the acknowledge signals ack_F, ack_S. Another sample can now be presented by $\mathcal{S}_1$, and so on.

$$\begin{array}{ll}
 & \mathcal{S}_1(req_S, ack_S, ack_F : Bool\ ;\ X : T) \\
\triangleq & \mathsf{begin} \quad \bullet\ req_S, ack_S, ack_F := false\ ;\ X := x_0; \\
 & \quad \mathsf{do}\ \neg(req_S \vee ack_S \vee ack_F) \rightarrow X := X'.(X' \in T_X)\ ;\ req_S := true \\
 & \quad\quad [\!]\ req_S \wedge ack_S \wedge \neg ack_F \rightarrow req_S := false \\
 & \quad \mathsf{od} \\
 & \quad \mathsf{end}
\end{array}$$

$$\begin{array}{ll}
 & \mathcal{B}_1(req_F, ack_F : Bool\ ;\ X, Z[0..N-2] : T) \\
\triangleq & \mathsf{begin} \quad \bullet\ req_F, ack_F : false\ ;\ X := x_0\ ;\ Z[0..N-2] := z_0; \\
 & \quad \mathsf{do}\ req_F \wedge \neg ack_F \rightarrow Z[0], .., Z[N-2] := X, .., Z[N-3]\ ;\ ack_F := true \\
 & \quad\quad [\!]\ \neg req_F \wedge ack_F \rightarrow ack_F := false \\
 & \quad \mathsf{od} \\
 & \quad \mathsf{end}
\end{array}$$

$$\begin{array}{ll}
 & \mathcal{F}_1(req_S, req_F, ack_S, ack_F : Bool\ ;\ X, Z[0..N-2], Y : T) \\
\triangleq & \mathsf{begin\ var}\ h[0..N-1] : T \quad \bullet\ req_S, req_F, ack_S, ack_F := false; \\
 & \quad\quad X, Z[0..N-2], h[0..N-1], Y := x_0, z_0, h_0, y_0; \\
 & \quad \mathsf{do}\ req_S \wedge \neg(req_F \vee ack_F) \rightarrow Y := \sum_{k=1}^{N-1} h[k] \times Z[k-1] + h[0] \times X\ ;\ req_F := true \\
 & \quad\quad [\!]\ req_F \wedge ack_F \rightarrow req_F := false\ ;\ ack_S := true \\
 & \quad\quad [\!]\ \neg req_S \wedge ack_S \rightarrow ack_S := false \\
 & \quad \mathsf{od} \\
 & \quad \mathsf{end}
\end{array}$$

Figure 3: Communicating models.

Consider further that, in the above example, X is an audio signal and $\mathcal{F}_1$ models a low-pass filter. The output of $\mathcal{F}_1$ would go to the woofer speaker of one's audio system. We would also like to have a high-pass filter, the output of which would go to the corresponding speakers of the same audio system. We want to reuse the previously designed modules and then add one that can detect the high frequencies of the incoming signal. The high-frequency filter is modeled by the new system $\mathcal{M}_1$ - Figure 4.

In order to accommodate the introduction of $\mathcal{M}_1$, the system $\mathcal{B}_1$ has to wait for the two filters to read its data, once a new sample has been issued by $\mathcal{S}_1$. Consequently, we have to change the representation of $\mathcal{B}_1$. The same is required for $\mathcal{S}_1$, since it has to communicate with $\mathcal{M}_1$, too. The new system $\mathcal{B}_2$ is described in Figure 4.

Discussion. The interleaving model of execution brings the benefit of a very simple concept. In order to reduce the implicit nondeterministic behavior of the model, inappropriate in certain situations, as shown in our example, one may introduce control channels, to ensure that the data emitted by one source is not missed by any of the intended targets, or that data is processed in a correct manner.

However, there is another aspect of the problem, not yet solved by the exemplified introduction of the communication channels. An observer of the composed system $\mathcal{P}_2 \triangleq \mathcal{S}_2 \parallel \mathcal{B}_2 \parallel \mathcal{F}_1 \parallel \mathcal{M}_1$ (the listener, in the example) has access to both output

$\mathcal{M}_1(req_S, req_M, ack_{SM}, ack_M : Bool\,;\, X, Z[0..N-2], W : T)$
$\triangleq$ begin var $h[0..N-1] : T \quad \bullet\ req_S, req_M, ack_{SM}, ack_M := false;$
$\quad X, Z[0..N-2], h[0..N-1], W := x_0, z_0, h_0, w_0;$
do $req_S \wedge \neg(req_M \vee ack_M) \rightarrow W := \sum_{k=1}^{N-1} h[k] \times Z[k-1] + h[0] \times X;$
$\quad req_M := true$
$[\!]\ req_M \wedge ack_M \rightarrow req_M := false\,;\, ack_{SM} := true$
$[\!]\ \neg req_S \wedge ack_{SM} \rightarrow ack_{SM} := false$
od
end

$\mathcal{B}_2(req_F, ack_F, req_M, ack_M : Bool\,;\, X, Z[0..N-2] : T)$
$\triangleq$ begin $\bullet\ req_S, ack_S, req_F, ack_F : false\,;\, X := x_0\,;\, Z[0..N-2] := z_0;$
do $req_F \wedge req_M \wedge \neg ack_F \wedge \neg ack_M \rightarrow Z[0], .., Z[N-2] := X, .., Z[N-3];$
$\quad ack_F, ack_M := true$
$[\!]\ \neg req_F \wedge \neg req_M \wedge ack_F \wedge ack_M \rightarrow ack_F, ack_M := false$
od
end

Figure 4: The systems $\mathcal{M}_1$ and $\mathcal{B}_2$.

sequences, $Y(n)$ and $W(n)$ (Figure 1). Depending on the execution order of $\mathcal{F}_1$ and $\mathcal{M}_1$, until the listener observes the new output $(Y(n+1), W(n+1))$, it also observes the intermediate state, either $(Y(n), W(n+1))$ or $(Y(n+1), W(n))$, which is also an incorrect aspect of the design. A solution is provided, again, by introducing new communication channels, between $\mathcal{F}_1$ and $\mathcal{M}_1$, on one side, and the observer, on the other. What happens if multiple, different observers become necessary in the design?

Any extension / reduction of the design elements requires an internal change of the involved subsystems. This clearly destroys any hope for a modular design flow and the reuse of components in future projects. We may assign meanings like "data valid", "operation finished", etc., to the signals of the communication channels, thus the interleaved approaches are suitable for asynchronous designs [Theodoropoulos 2002]. Unfortunately, these signals are global variables of the model. In hardware, generally, this translates into "more wires"; in software, this violates the principle of information hiding [Behrends and Stirewalt 2000]. In the following section, we propose a solution to this kind of design issues.

4 Synchronized Parallel Environments

Synchronized environment. We want to build an environment in which the response of the system is a collection of the individual component reactions to the input stimuli. The solution that we propose requires that the subsystems synchronize when the global variables of the compound system are updated. This is achieved by extending the execution round concept, as described in Section 3, to an *execution cycle*. A cycle is defined by the activities carried out by the system between two global states: it is a sequence of rounds in which each participating AS updates the local variables, as necessary, followed by a last round, in which, simultaneously, *all* the global variables are updated, accordingly. Notice that between rounds, the global state of the system does not change.

From the controller's point of view, we can imagine the following scenario. It selects for execution an enabled action from one component AS. If the action updates global variables, the system is marked as "executed", and no other action can be selected from that system. However, the other participants, or possible external observers do not see the changes yet. Another action is then selected, from an "unexecuted" AS. The process continues until all the components are marked "executed". This also signals the end of a cycle, when all the global variables are updated.

Proper Action Systems. The translation of the above scenario into our framework requires certain characteristics for the AS employed in the design. These requirements are introduced by the following definition.

Definition 1 *Consider the action system $\mathcal{A}$*

$$\mathcal{A}(z : T_z) \overset{\wedge}{=} \mathsf{begin\ \ var}\ \ x : T_x \bullet\ Init\,;\ \mathsf{do}\, g_L \rightarrow L \,[\!]\, g_S \rightarrow S\, \mathsf{od\ \ end} \tag{2}$$

We say that $\mathcal{A}$ is a **proper** *action system if:*

1. *$gw\mathcal{A} \subseteq wS$ – meaning that S is a global action of $\mathcal{A}$. Notice that wS may also contain local variables of $\mathcal{A}$.*
2. *$wL \subseteq lw\mathcal{A}$ – meaning that L is a local action of $\mathcal{A}$.*
3. *$\mathrm{wp}(\ \mathsf{do}\ g_L \rightarrow L\ \mathsf{od}, \neg g_L \wedge g_S) \equiv true$ – meaning that the execution of L, taken separately, terminates, and establishes the precondition for executing S.*

Notice that the specification $\mathcal{A}$, as given by Definition 1, encodes more visibly than (1), the mechanism that triggers the global state changes.

Definition 2 *Let us consider n proper action systems ($k = 1 \ldots n$):*

$$\mathcal{A}_k(z_k) \overset{\wedge}{=} \mathsf{begin\ \ var}\ \ x_k \bullet\ Init_k\ ;\ \mathsf{do}\, g_L^k \rightarrow L_k \,[\!]\, g_S^k \rightarrow S_k\, \mathsf{od\ \ end},$$

for which we also have that $\forall j, k \in [1..n], j \neq k \bullet ((gw\mathcal{A}_j \cap gw\mathcal{A}_k = \emptyset) \wedge (\bigcap_k x_k = \emptyset))$. The **synchronized parallel composition** *of the above systems is a new action system $\mathcal{P} = \mathcal{A}_1 \sharp \ldots \sharp \mathcal{A}_n$, given by:*

$$
\begin{array}{lll}
\mathcal{P}(z) & \overset{\wedge}{=} & \mathsf{begin\ \ var}\ x : T_x, sel[1..n] : \mathsf{Bool}, run : \mathsf{Nat} \bullet Init; \\
 & & \ \mathsf{do} \\
 & & \quad gg_P \rightarrow (run = 0 \wedge \neg sel[1] \rightarrow sel[1] := true\,; run := 1 \\
 & & \qquad [\!]\ \ldots \\
 & & \qquad [\!]\ run = 0 \wedge \neg sel[n] \rightarrow sel[n] := true\,; run := n \\
 & & \qquad [\!]\ (run = 1 \wedge g_L^1 \rightarrow L_1 [\!]\ run = 1 \wedge \neg g_L^1 \wedge g_S^1 \rightarrow wS_1c := wS_1\ ; S_1'\ ; run := 0 \\
 & & \qquad\qquad\quad [\!]\ run = 1 \wedge \neg gg_{A_1} \rightarrow run := 0) \\
 & & \qquad [\!]\ \ldots \\
 & & \qquad [\!]\ (run = n \wedge g_L^n \rightarrow L_n [\!]\ run = n \wedge \neg g_L^n \wedge g_S^n \rightarrow wS_nc := wS_n\ ; S_n'\ ; run := 0 \\
 & & \qquad\qquad\quad [\!]\ run = n \wedge \neg gg_{A_n} \rightarrow run := 0)) \\
 & & \qquad [\!]\ \ sel \wedge run = 0 \rightarrow Update\,; sel := false \\
 & & \ \mathsf{od} \\
 & & \mathsf{end}
\end{array}
$$

The operator '$\sharp$' ('sharp') is called the **synchronized parallel** *operator.*

The set z of global variables of $\mathcal{P}$ is, initially, the union of the global variables sets of each individual system: $z = \bigcup_k z_k$. It may be possible that communication between several submodules of $\mathcal{P}$ (the composing systems $\mathcal{A}_k$) should not be disclosed at the interface of $\mathcal{P}$. Therefore, the variables that model such channels will be **hidden** *within the system $\mathcal{P}$. They will not be mentioned in z.*

Further, the local variables x of the new action system $\mathcal{P}$ are the union of the local variables x_k, to which we add the hidden variables. We also add copies (wS_kc) of the original write variables of each action body S_k. They replace the original variables wS_k, therefore we have $S'_k = S_k[wS_kc/wS_k]$. Finally, the list x is completed by adding the array sel *and the execution indicator,* run. *The predicate gg_P is a short notation for the disjunction of the guards of all the actions in the systems $\mathcal{A}_k$: $gg_P \triangleq \bigvee_1^n gg_{A_k}$, where $gg_{A_k} = g_L^k \vee g_S^k$.*

The Init *statement is the sequential composition of the individual* Init_k *statements to which we add the initialization of variables wS_kc,* sel *and* run*:*

$$Init \triangleq Init_1 \,;..;\, Init_n \,;\, wS_1c,..,wS_nc := wS_1,..,wS_n \,;\, run := 0 \,;\, sel := false$$

The action $Update$ is given by:

$$Update \triangleq Update_1 \,;\ldots;\, Update_n, \text{ where } Update_k \triangleq wS_k := wS_kc.$$

The above definition of the '$\sharp$' operator says that, whenever there is a change in the input, such a composition of action systems reacts based on the state of all its components, and the result is composed of the individual reaction of each of the subsystems. The system composition reacts only if at least one component is enabled ($\exists k \in [1..n] \bullet gg_{A_k} \equiv true$). Moving certain variables to the local level, within the system $\mathcal{P}$, is motivated by the containment of local communication. The variable *run* identifies the system that is selected for execution. The variable *sel* stores the information on the executing or already executed systems. Whenever all of the elements of the array sel become $true$ ($sel = sel[1] \wedge .. \wedge sel[n]$) and $run = 0$, we have reached the end of an execution cycle. At this moment, the assignment $sel := false$ is understood as a shorthand notation for $sel[1] := false \,;\ldots;\, sel[n] := false$.

The assignment $wS_kc := wS_k$ that precedes the action S'_k, takes into account that the (local) variables of wS_k could have been also updated by L_k. As they may also belong to rS_k, their current values must be taken into consideration. In the case when actions L_k do not modify rS_k, the presence of this assignment is not necessary. The same applies when there are no local actions in a given proper AS.

We may further be able to find useful properties for the system $\mathcal{P}$, expressed by the following theorem (the proof is shown in Appendix A).

Theorem 1 *Assume that the proper action systems $\mathcal{A}_1$ and $\mathcal{A}_2$ are of the form given by (2). Then, the synchronized parallel composition $\mathcal{A}_1 \sharp \mathcal{A}_2$ satisfies the following properties:*

(a) $\mathcal{A}_1 \sharp \mathcal{A}_2$ *is a proper action system* *(properness of $\sharp$)*
(b) $\mathcal{A}_1 \sharp \mathcal{A}_2 = \mathcal{A}_2 \sharp \mathcal{A}_1$ *(commutativity of $\sharp$)*

The Update Action. In a more general view, we would avoid imposing the restriction that $\forall j, k = 1 \ldots n, j \neq k.gw\mathcal{A}_j \cap gw\mathcal{A}_k = \emptyset$. Designs where the system models do not have disjoint sets of global write variables are not necessarily examples of bad designs. A well known example of such situations is the bus-based design of digital systems, where multiple participants in the processing effort share a common resource, the bus lines. Of course, the situation requires a thorough analysis, and there exist multiple solutions that resolve the inevitable conflicts. Therefore, the action $Update$ might not be merely the action that updates the respective variables; instead it can rather be the action that "resolves" such conflicts. Intuitively, this means that the system $\mathcal{P}$ must also allow the designer to separately specify the action $Update$. This is subject to further studies.

4.1 Design Implications

We revisit briefly the example proposed in Section 3.2. Consider that instead of the parallel composition $\mathcal{P} = \mathcal{S} \parallel \mathcal{B} \parallel \mathcal{F}$, we write the description of our system as $\mathcal{P} = \mathcal{S} \sharp \mathcal{B} \sharp \mathcal{F}$. It is easy to check that the components $\mathcal{S}, \mathcal{B}, \mathcal{F}$ are proper AS. Therefore, we do not have to add communication channels to any of the subsystems, which all remain as described in Figure 2. Also, in case of a synchronized environment, the multiplicity of targets stops being an issue for the system $\mathcal{P}$. We can introduce as many $\mathcal{F}$-like systems as required, without modifying $\mathcal{B}$ or $\mathcal{S}$ in order to accommodate the presence of the new modules. Additionally, an external observer will always observe only the state $(Y(n+1), W(n+1))$, regardless of the order in which the systems $\mathcal{F}$ and the corresponding $\mathcal{M}$ ($\mathcal{M}_1$ without the communication variables) are selected for execution.

5 Design Process

Faced with the complexity of modern day devices, the designer of such systems has to start the design process at higher levels of abstraction, which may provide him with a simpler model of the whole system. A correct partitioning and identification of the necessary components is the next step. Crucial to a module-based design context is the possibility to separately analyze and, if necessary, improve the functionality of the subsystems, optimize them for a given technology, or map them to existent library elements. All these actions involve, most usually, certain transformations of the initial representations. One has to certify that the modifications imposed on the modules represent a correct transformation of the initial specification, with respect to behavior. Within the refinement calculus [Back and von Wright 1998], the correctness of such transformations is ensured by action-level and system-level refinement rules. In the following, we firstly introduce the basic notions that help us produce correct-by-construction models, and then we continue with exemplifications of how the mentioned rules apply to system design. We analyze the techniques from both an interleaved, and also a synchronized perspective.

5.1 Refinement of actions and AS

A predicate $I(vA)$ – I in short – is an *invariant* of the action $A \mathrel{\widehat{=}} g \rightarrow S$, if it holds prior to and after the execution of A. We then say that I is *preserved* by A, that is, $g \wedge I \Rightarrow \mathrm{wp}(S, I)$. At the system level, a predicate $I(v\mathcal{A})$ is an *invariant* of the AS $\mathcal{A}$, given by (1), if it is established by $Init$, that is, $true \Rightarrow \mathrm{wp}(Init, I)$, and also if it is preserved by each action A_i.

An action A is *(algorithmically) refined* by the action C, written $A \leq C$, if, whenever A establishes a certain postcondition, so does C [Back 1990]:

$$A \leq C \;\triangleq\; \forall Q \;\bullet \mathrm{wp}(A, Q) \Rightarrow \mathrm{wp}(C, Q)$$

Next, let $R(a, c, z)$ (simply written as R) be a boolean *abstraction* relation, which links the abstract local variables a to the concrete local variables c. Additionally, let $I(c, z)$ be an invariant of the action C. Then, action A is *data refined* by action C using the relation R and the invariant I, that is, $A \leq_{R,I} C$, if

$$\forall Q \;\bullet R \wedge I \wedge \mathrm{wp}(A, Q) \Rightarrow \mathrm{wp}(C, \exists a \;\bullet R \wedge I \wedge Q),$$

where Q is a predicate on the variables a, z, and $(\exists a.R \wedge I \wedge Q)$ is a predicate on a, c, z. If R is the identity relation ($R \;\triangleq\; a = c$), we then write $A \leq_I C$. Similarly, if $I \equiv true$, we write $A \leq_R C$. If both are trivial, we run into the usual algorithmic refinement of actions, $A \leq C$, as defined above.

The semantics of an AS is given in terms of behaviors [Back and von Wright 1994]. A *behavior* of an AS is a sequence of states, $b = \langle (x_0, y_0), (x_1, y_1) \ldots \rangle$, where each state has two components. The first component is the *local state* and the second is the *global state*. Behaviors can be finite or infinite. A finite behavior is called *terminating* if it ends in a proper state, or *aborting* if it ends improperly, indicated by the symbol $\perp$. A *trace* of a behavior is obtained by removing all finite stuttering (no change of the visible states), and the local state component in each state of a given system.

In a general, less formal manner, we say that an action system $\mathcal{C}$ refines $\mathcal{A}$, written as $\mathcal{A} \sqsubseteq \mathcal{C}$, if every trace of $\mathcal{C}$ contains a trace of $\mathcal{A}$. In practice, we use the following lemma to prove *trace refinement* of action systems [Back and Sere 1994-2].

Lemma 1 *Given the action systems*

$$\begin{array}{rl}\mathcal{A}(z_{\mathcal{A}}) \;\triangleq & \mathsf{begin\ var}\ a \;\bullet\; a, z_A := a_0, z_{A0}\,;\ \mathsf{do}\ A\ \mathsf{od} \\ \mathcal{C}(z_{\mathcal{C}}) \;\triangleq & \mathsf{begin\ var}\ c \;\bullet\; c, z_C := c_0, z_{C0}\,;\ \mathsf{do}\ C \,[\!]\, X\ \mathsf{od}\ \ \mathsf{end},\end{array}$$

let $R(a, c, z_A, z_C)$ be an abstraction relation and $I(c, z_C)$ an invariant of the system $\mathcal{C}$. The concrete system $\mathcal{C}$ (trace) refines the abstract system $\mathcal{A}$, denoted $\mathcal{A} \sqsubseteq_{R,I} \mathcal{C}$, if:

1. *Initialization:* $R(a_0, c_0, z_{A0}, z_{C0}) \wedge I(c_0, z_{C0}) \equiv true$
2. *Main action:* $A \leq_{R,I} C$

3. *Auxiliary action:* skip $\leq_{R,I} X$
4. *Continuation condition:* $R \wedge I \wedge gA \Rightarrow gC \vee gX$
5. *Internal convergence:* $R \wedge I \Rightarrow \mathrm{wp}(\textsf{ do } X \textsf{ od }, true)$.

5.2 Refinement Example

Let us see next how the refinement procedure is applied to the design example outlined in section 3.2. Considering a hardware implementation of our example, a direct mapping of the filter functionality on hardware elements (registers, multipliers, adders, etc) is represented in Figure 5 a). However, characteristic to this implementation of system $\mathcal{F}$ is the parallel processing and the large area occupied by the hardware elements. A functionally equivalent implementation (Figure 5 b)) would result from a serial representation of the filtering device, which will require a more reduced silicon area. We transform the original system $\mathcal{F}$ into $\mathcal{F}_S$, with the AS model given in Figure 6. Is this a correct transformation of $\mathcal{F}$? Is the whole system still working according to the functional specification ?

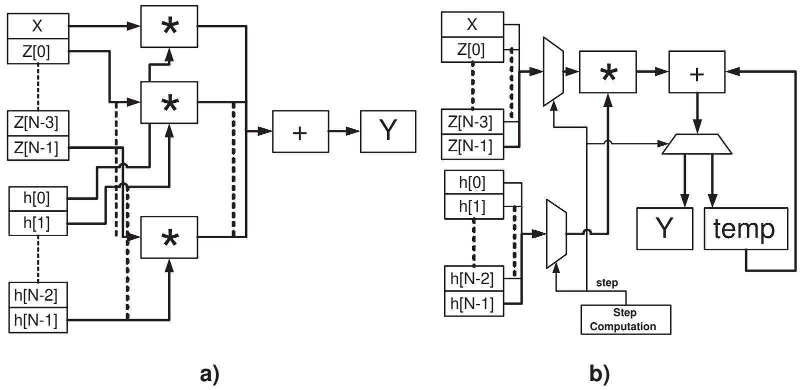

Figure 5: The hardware implementation of the filter $\mathcal{F}_S$.

In *isolation*, one may prove (see Appendix E), using Lemma 1, that the system $\mathcal{F}_S$ is a refinement of $\mathcal{F}$, under the invariant I:

$$\mathcal{F} \sqsubseteq_I \mathcal{F}_S, \;\; I \;\triangleq\; step \in [2..N] \Rightarrow temp = \sum_{k=1}^{step-1} h[k] \times Z[k-1]$$

However, our task is not completed yet. From a system level point of view, we should check that $\mathcal{S} \parallel \mathcal{B} \parallel \mathcal{F} \sqsubseteq_I \mathcal{S} \parallel \mathcal{B} \parallel \mathcal{F}_S$. Unfortunately, as $\mathcal{B}$ does not *respect I*, the refinement is not possible (see [Back and von Wright 2003] for details). This fact

$$
\begin{array}{ll}
 & \mathcal{F}_S(X, Z[0..N-2], Y : T) \\
\triangleq & \textsf{begin var } h[0..N-1], temp : T \,;\, step : 0..N \quad \bullet \\
 & \quad X, Z, h, Y := x_0, z_0, h_0, y_0 \,;\, temp := 0 \,;\, step := 0; \\
 & \quad \textsf{do } step = 0 \rightarrow temp := 0 \,;\, step := step + 1 \\
 & \quad\quad [\!]\ step \in [1..N-1] \rightarrow temp := temp + h[step] \times Z[step-1] \,;\, step := step + 1 \\
 & \quad\quad [\!]\ step = N \rightarrow Y := temp + X \times h[0] \,;\, step := 0 \\
 & \quad \textsf{od} \\
 & \textsf{end}
\end{array}
$$

Figure 6: The new system $\mathcal{F}_S$.

has a simple explanation; since the controller may choose an enabled action from either $\mathcal{B}$ or $\mathcal{F}_S$, let us suppose that it chooses only actions from $\mathcal{F}_S$, until $step = N$, after which it selects $\mathcal{B}$ for execution. Hence, following the update on Z, the invariant I is no longer valid. The solution comes, again, from employing communication channels as described in section 3.2. The invariant I has to be rewritten so as to take into account these channels, and the systems will gain some independence in this respect. Still, the same problems arise when introducing another filtering unit ($\mathcal{M}$), in which case both the system models and the invariant must be reshaped.

5.3 Refinement of Component Systems in Synchronized Environments

Definition 3 *A predicate I is a* **proper invariant** *of a proper system $\mathcal{A}$, if $\forall z \notin wS \cdot g_S \Rightarrow (I[w'S/wS, z] \equiv I[w'S/wS])$, where $w'S$ is the new value of wS, after the execution of S.*

The above definition says that, following the execution of the global action $g_S \rightarrow S$, the computed value of a proper invariant I depends on the variables in wS, only. Next, we give a lemma that can be used to prove trace refinement of proper action systems.

Lemma 2 *Given the proper action systems*

$$
\begin{aligned}
\mathcal{A}(z_{\mathcal{A}}) &\triangleq \textsf{begin var } a \bullet a, z_A := a_0, z_{A0} \,;\ \textsf{do } g_L^A \rightarrow L_A [\!]\ g_S^A \rightarrow S_A \textsf{ od end} \\
\mathcal{C}(z_{\mathcal{C}}) &\triangleq \textsf{begin var } c \bullet c, z_C := c_0, z_{C0} \,;\ \textsf{do } g_L^C \rightarrow L'_A [\!]\ g_S^C \rightarrow S'_A [\!]\ g_X \rightarrow X \textsf{ od end},
\end{aligned}
$$

let $R(a, c, z_A, z_C)$ be an abstraction relation and $I(c, z_C)$ a proper invariant of the system $\mathcal{C}$. The system $\mathcal{A}$ is (trace) refined by the system $\mathcal{C}$, $\mathcal{A} \sqsubseteq_{R,I} \mathcal{C}$, if:

1. *Initialization:* $R(a_0, c_0, z_{A0}, z_{C0}) \wedge I(c_0, z_{C0}) \equiv true$
2. *Main actions:* $(g_L^A \rightarrow L_A \leq_{R,I} g_L^C \rightarrow L'_A) \wedge (g_S^A \rightarrow S_A \leq_{R,I} g_S^C \rightarrow S'_A)$
3. *Auxiliary action:* $\textsf{skip} \leq_{R,I} g_X \rightarrow X$
4. *Continuation condition:* $R \wedge I \wedge (g_L^A \vee g_S^A) \Rightarrow g_L^C \vee g_S^C \vee g_X$

5. *Properness:* $R \wedge I \Rightarrow \mathrm{wp}(\ \mathsf{do}\ g_X \rightarrow X \,[\!]\, g_L^C \rightarrow L'_A\ \mathsf{od}\ , \neg(g_X \vee g_L^C) \wedge g_S^C)$.

The first four requirements of the lemma are adaptations of the original ones (given by Lemma 1), to our case. The fifth, however, strengthens the original request by specifying that not only the auxiliary action $g_X \rightarrow X$, taken in isolation, must terminate, but also that the new group of local actions, $g_X \rightarrow X \,[\!]\, g_L^C \rightarrow L'_A$ must terminate and, moreover, must also establish the necessary conditions for the (possibly) new global action $g_S^C \rightarrow S'_A$ to execute. The proof that the above Lemma establishes the conditions for a trace refinement is given in Appendix B.

5.3.1 Modularity

Along the line established by Lemma 2, we prove the following theorem.

Theorem 2 *Consider the synchronized environment* $\mathcal{P} \triangleq \mathcal{A}_1 \sharp \ldots \sharp \mathcal{A}_n$, *where each of the component systems preserves the proper invariants* $I_1, \ldots, I_n$, *respectively. We then have that* $I \triangleq I_1 \wedge \ldots \wedge I_n \wedge \bigwedge_{k \in [1..n]} (sel[k] \wedge \neg(run = k) \Rightarrow I'_k),\ where\ I'_k \triangleq I_k[wS_kc/wS_k]$, *is a proper invariant of* $\mathcal{P}$.

The theorem states that in a synchronized environment, the global properties of the system are obtained from the individual properties of the components, as $I \Rightarrow I_1 \wedge \ldots \wedge I_n$. The additional terms of I help us make the connection between the copies of the write variables and the respective original variables, at the moment when the action *Update* is executed. The theorem is proved in Appendix C.

Corollary 1 *Consider the proper action systems* $\mathcal{A}_k$ *as in Definition 2, and the abstraction relation* R_j. *Moreover, the system* $\mathcal{A}_j$ *preserves its respective proper invariant* I_j. *Then*

$$\frac{\mathcal{A}_j \sqsubseteq_{R_j, I_j} \mathcal{A}'_j}{\mathcal{A}_1 \sharp \ldots \sharp \mathcal{A}_j \sharp \ldots \sharp \mathcal{A}_n \sqsubseteq_{R_j, I_j} \mathcal{A}_1 \sharp \ldots \sharp \mathcal{A}'_j \sharp \ldots \sharp \mathcal{A}_n}, \forall j \in [1 \ldots n]$$

The statement of the corollary follows from Theorem 2 and Lemma 2 (see Appendix D).

The interpretation of Corollary 1 is that each component of a synchronized parallel composition may be refined in isolation, without knowledge about the invariants of the other components. The system designer may then employ the modules without knowing their respective internal details of functionality. The module designer is responsible with improving the performance of the modules, in total transparency for the integrator designer. This is a consequence of the fact that the systems exchange information at the end of an execution cycle, rather than after each execution round. Observe that, if I_j is a *new* invariant for $\mathcal{A}'_j$, it will just be a new entry in the definition of I, as specified by Theorem 2.

A similar conclusion as ours is reached in [Back and von Wright 2003] for the parallel composition of AS. However, this is achieved while requiring that the invariants of all the subsystems are known, and a noninterference relation between them proves to hold. The corresponding noninterference condition corresponds to our requirement that the invariant I_j is proper. Still, checking the properness of an invariant concerns the respective module designer only, who does not have to obtain information about the other invariants. Therefore, we have increased the independence of the module designer.

5.4 Refinement Example

Considering the analysis presented in section 5.2, if we check $\mathcal{F} \sqsubseteq_I \mathcal{F}_S$ in the context of Lemma 2, meaning that we adopt a synchronized perspective on the subsystem composition, we will immediately obtain that $\mathcal{S} \sharp \mathcal{B} \sharp \mathcal{F} \sqsubseteq_I \mathcal{S} \sharp \mathcal{B} \sharp \mathcal{F}_S$ (notice that $\mathcal{F}_S$ is a proper AS). Besides this, a previous addition of module $\mathcal{M}$ would not change the refinement, and we could have $\mathcal{S} \sharp \mathcal{B} \sharp \mathcal{F} \sharp \mathcal{M} \sqsubseteq_I \mathcal{S} \sharp \mathcal{B} \sharp \mathcal{F}_S \sharp \mathcal{M}$.

6 Continuous Action Systems

A *continuous action system* (in short, CAS) [Back et al. 2001] is of the form

$$\begin{aligned} \mathcal{C}(z : \mathsf{Real}_+ \to T_z) \;\hat{=}\; & \mathsf{begin}\;\; \mathsf{var}\; x : \mathsf{Real}_+ \to T_x \bullet Init; \\ & \mathsf{do}\; g_1 \to S_1 \,[\!]\; \ldots \,[\!]\; g_m \to S_m \;\; \mathsf{od}\;\; \mathsf{end} \end{aligned}$$

Here, Real_+ stands for the non-negative reals, and models the time domain.

The execution of a CAS uses an implicit variable now, which denotes the present time. The actions may refer the value of now, but they can not change it. After the initialization, the system starts evolving, with time (measured by now) moving forward continuously. The execution of a CAS resembles closely that of an ordinary AS, with the difference that, after the changes stipulated by S_i have been done, the system evolves to the next time instance when one of the actions is enabled. We write $x :- e$ rather than $x := e$, to emphasize that only the future behavior of the variables x is changed.

We explain the meaning of $\mathcal{C}$ by translating it into an ordinary (discrete) AS, $\overline{\mathcal{C}}$:

$$\begin{aligned} \overline{\mathcal{C}}(z) \;\hat{=}\; & \mathsf{begin}\;\; \mathsf{var}\; now : \mathsf{Real}_+, x : \mathsf{Real}_+ \to T_x \bullet now := 0 \,;\, Init \,;\, N \,; \\ & \quad \mathsf{do}\; g_1.now \to S_1 \,;\, N \,[\!]\ldots[\!]\; g_m.now \to S_m \,;\, N \;\; \mathsf{od} \\ & \mathsf{end} \end{aligned} \tag{3}$$

$$N \;\hat{=}\; now := \mathsf{next}.gg_C.now,$$

$$\mathsf{next}.gg_C.t \;\hat{=}\; \begin{cases} min\{t' \geq t \mid gg_C.t'\}, & \text{if exists } t' \geq t \text{ such that } gg_C.t', \\ t, & \text{otherwise} \end{cases}$$

In $\overline{\mathcal{C}}$, the variable now is declared, initialized and updated explicitly. It models the starting time and the succeeding moments when some action is enabled. The value of a variable v or of an expression e at a given moment of time t is identified by $v.t$ or $e.t$, respectively. Their values at the current moment are consequently given by $v.now$ and

$e.now$. The value of now is updated by the statement N. The function next gives the minimum moment of time when at least one action is enabled. If, after some point in time, no action is ever enabled, the second branch of the definition will be followed, and now will denote the moment of time when the last discrete action has been executed, the system terminating with the last assigned values for the variables.

Parallel CAS - Traditional Model. The parallel composition of two or more CAS is defined by using the same method as for composing ordinary AS: it combines the variables and the actions of the component systems, executing the enabled actions in an interleaved manner. One needs to combine the component CAS before translating them into discrete action systems, to ensure that the composed system uses a unique now variable.

Remarks. The original semantics of CAS, as given by (3), does not guarantee a *timelock* free model of the hybrid system in question. This claim is supported by the definition of the statement N itself. In order to advance time, the virtual controller calculates the minimum moment of time greater than or equal to the current value of now, when the disjunction of the guards holds. This means that certain actions could be executed more than once at the same moment of time. Therefore, they might never become disabled, with time locked at now. Thus, the system is prevented from evolving, yet it does not terminate. In consequence, we may be faced with the situation of executing some action forever, without advancing time. We call this situation a timelock. As an example, we consider the following action system.

$$
\begin{array}{lll}
\overline{S}(\theta : \mathsf{Real}_+ \to \mathsf{Real}) \triangleq & \mathsf{begin} \ \mathsf{var}\, now : \mathsf{Real}_+ \bullet now := 0\,;\theta : -(\lambda t \cdot 4 * t)\,; N \\
& \quad \mathsf{do}\ \theta.now = 10 \to \\
& \qquad\quad \theta : -(\lambda t \cdot \theta.now - 2 * (t - now))\,; N \\
& \qquad [\!]\ \theta.now = 5 \to \\
& \qquad\quad \theta : -(\lambda t \cdot \theta.now + 3 * (t - now))\,; N \\
& \quad \mathsf{od}\ \ \mathsf{end}
\end{array}
$$

In the model above, the guard of the first action (that decreases the temperature θ) remains true after having executed the corresponding action body. This happens because the next minimum moment of time when an action is enabled is the same as the previous one (at now, $\theta.t = \theta.now = 10$). Therefore, looping in the same state at the same moment of time goes on forever, thus the temperature never gets the chance to increase again, and the AS does not terminate. Simulating such models is not possible. This kind of problem can be avoided by disabling each action after it was executed at moment now. Thus, we add a $state$ variable that stores the current state of the system ($state.now$), and also the different future state. However, this solution works well for systems with a small number of states, but it might be difficult to apply to systems having a large number of states. In the latter case, determining the next state of the system, based on the information given by the values of the guards at time point now, could be a heavy, if not impossible task for the modeler. Therefore, we propose a generic modeling solution that fixes the mentioned problem of the original CAS semantics.

Timed Action Systems as New CAS Semantics. We translate a CAS into a *Timed Action System* (TAS) as follows.

$$
\begin{array}{l}
\overline{\mathcal{C}}(z : Real_+ \rightarrow T_z) \\
\hat{=}\ \mathsf{begin\ var}\ now, now_c : Real_+, x : Real_+ \rightarrow T_x, \\
\qquad\qquad u_1, \ldots, u_m : Real_+ \rightarrow \mathsf{Bool}\ \bullet \\
\qquad now := 0\ ;\ Init\ ;\ u_1, \ldots, u_m : -(\lambda t \cdot false)\ ;\ N\ ;\ now_c := now; \\
\quad \mathsf{do}\ \neg u_1.now \wedge g_1.now \rightarrow u_1 : -(\lambda t \cdot true)\ ;\ S_1\ ;\ N \\
\quad\ \ [\!]\ \ldots \\
\quad\ \ [\!]\ \neg u_m.now \wedge g_m.now \rightarrow u_m : -(\lambda t \cdot true)\ ;\ S_m\ ;\ N \\
\quad\ \ [\!]\ now \neq now_c \rightarrow u_1, \ldots, u_m : -(\lambda t \cdot false)\ ;\ now_c := now \\
\quad \mathsf{od} \\
\mathsf{end}
\end{array} \tag{4}
$$

The variables $u_1, \ldots, u_m$, in (4), force the system to execute each action only once at the same moment of time, by disabling the respective action. In this way, we prevent timelocks at the action level. We also use the copy of the variable now, now_c, to be able to store both the previous (now_c), and the current (now) moments of time when the system has taken a discrete transition. In order to avoid timelocks at the system level, we reset the variables $u_1, \ldots, u_m$ (m - the number of actions), thus enabling a new execution cycle, only if the currently computed value of now is different from the previous one, stored in now_c. In this way, the system continues its execution only if time progresses, otherwise it terminates.

Synchronized Parallel CAS. In this paragraph, we give semantics to the synchronized composition of CAS, which is similar in spirit to the one defined for the discrete case, yet bearing its own particular features.

Firstly, we introduce the *proper continuous action systems* (PCAS):

$$
\begin{array}{l}
\mathcal{A}(z : Real_+ \rightarrow T_z) \\
\hat{=}\ \mathsf{begin\ var}\ x : Real_+ \rightarrow T_x \bullet Init\ ;\ \mathsf{do}\ g_L \rightarrow L [\!]\ g_S \rightarrow S\ \mathsf{od}\ \ \mathsf{end}
\end{array} \tag{5}
$$

Observe that in (5), we have separated the local actions from the global actions, as we have also done for proper AS. Moreover, we impose similar requirements as stated by Definition 1.

We explain the meaning of a PCAS by translating it into the following TAS:

$$
\begin{array}{l}
\overline{\mathcal{A}}(z : Real_+ \rightarrow T_z) \\
\hat{=}\ \mathsf{begin\ var}\ now, now_c : Real_+, x : Real_+ \rightarrow T_x, \\
\qquad u_L, u_S : Real_+ \rightarrow \mathsf{Bool} \bullet \\
\qquad now := 0\ ;\ Init\ ;\ u_L, u_S : -(\lambda t \cdot false)\ ;\ N\ ;\ now_c := now; \\
\quad \mathsf{do}\ \neg u_L.now \wedge g_L.now \rightarrow u_L : -(\lambda t \cdot true)\ ;\ L\ ;\ N \\
\quad\ \ [\!]\ \neg u_S.now \wedge g_S.now \rightarrow u_S : -(\lambda t \cdot true)\ ;\ S\ ;\ N \\
\quad\ \ [\!]\ now \neq now_c \rightarrow u_L, u_S : -(\lambda t \cdot false)\ ;\ now_c := now \\
\quad \mathsf{od} \\
\mathsf{end}
\end{array} \tag{6}
$$

Further, let us consider n PCAS of the form given by (5). Their **synchronized parallel composition** is a new system, $\mathcal{P} = \mathcal{A}_1 \sharp \ \ldots \ \sharp \mathcal{A}_n$. Its semantics is defined in Figure 7. Apart from the specific differences due to using proper CAS rather than proper AS, the synchronized parallel composition of PCAS does not need the test of gg_P (as in the discrete version) upon the entrance of the loop. This is motivated by the fact that $\overline{\mathcal{P}}$ cannot execute skip actions forever.

$$
\begin{array}{ll}
& \overline{\mathcal{P}}(z : \mathsf{Real}_+ \rightarrow T_z) \\
\triangleq & \mathsf{begin}\ \ \mathsf{var}\ x : \mathsf{Real}_+ \rightarrow T_x, sel[1..n], u_L^1, \ldots, u_S^n : \mathsf{Real}_+ \rightarrow \mathsf{Bool}, \\
& run : \mathsf{Real}_+ \rightarrow \mathsf{Nat}, now, now_c : \mathsf{Real}_+ \quad \bullet \\
& now := 0 \,;\, Init \,;\, N \,;\, now_c := now; \\
& \ \mathsf{do} \\
& \qquad\quad run.now = 0 \wedge \neg sel[1].now \rightarrow sel[1] :- (\lambda t \cdot true) \,;\, run :- (\lambda t \cdot 1) \\
& \qquad\ \ [\!] \ \ldots \\
& \qquad\ \ [\!] \ run.now = 0 \wedge \neg sel[n].now \rightarrow sel[n] :- (\lambda t \cdot true) \,;\, run :- (\lambda t \cdot n) \\
& \quad\ [\!] \ (run.now = 1 \wedge \neg u_L^1.now \wedge g_L^1.now \rightarrow L_1 \,;\, u_L^1 :- (\lambda t \cdot true) \,;\, N \\
& \qquad\ \ [\!] \ run.now = 1 \wedge \neg u_S^1.now \wedge \neg g_L^1.now \wedge g_S^1.now \rightarrow wS_1c :- wS_1 ; S_1'; \\
& \qquad\qquad run :- (\lambda t \cdot 0) \,;\, u_S^1 :- (\lambda t \cdot true) \,;\, N \\
& \qquad\ \ [\!] \ run.now = 1 \wedge \neg u_S^1.now \wedge \neg gg_{A_1}.now \rightarrow run :- (\lambda t \cdot 0); \\
& \qquad\qquad u_S^1 :- (\lambda t \cdot true) \,;\, N) \\
& \quad\ [\!] \ \ldots \\
& \quad\ [\!] \ (run.now = n \wedge \neg u_L^n.now \wedge g_L^n.now \rightarrow L_n \,;\, u_L^n :- (\lambda t \cdot true) \,;\, N \\
& \qquad\ \ [\!] \ run.now = n \wedge \neg u_S^n.now \wedge \neg g_L^n.now \wedge g_S^n.now \rightarrow wS_nc :- wS_n ; S_n'; \\
& \qquad\qquad run :- (\lambda t \cdot 0) \,;\, u_S^n :- (\lambda t \cdot true) \,;\, N \\
& \qquad\ \ [\!] \ run.now = n \wedge \neg u_S^n.now \wedge \neg gg_{A_n}.now \rightarrow run :- (\lambda t \cdot 0); \\
& \qquad\qquad u_S^n :- (\lambda t \cdot true) \,;\, N) \\
& \ \ [\!] \ sel.now \wedge run.now = 0 \wedge now_c \neq now \rightarrow Update \,;\, sel :- (\lambda t \cdot false); \\
& \qquad u_L^1, \ldots, u_S^n :- (\lambda t \cdot false) \,;\, now_c := now \\
& \ \ [\!] \ sel.now \wedge run.now = 0 \wedge now_C = now \rightarrow Update \\
& \ \mathsf{od} \\
& \mathsf{end}
\end{array}
$$

$$
\begin{array}{lcl}
Init & \triangleq & Init_1 \,;\, \ldots ;\, Init_n \,;\, wS_1c, \ldots, wS_nc :- wS_1, \ldots, wS_n \,;\, run :- (\lambda t \cdot 0); \\
& & \quad sel, u_L^1, \ldots, u_S^n :- (\lambda t \cdot false) \\
Update & \triangleq & Update_1 \,;\, \ldots ;\, Update_n, \ \text{where } Update_k \ \triangleq \ wS_k :- wS_kc \\
S_k' & \triangleq & S_k[wS_kc/wS_k] \\
gg_{A_k} & \triangleq & g_L^k \vee g_S^k
\end{array}
$$

Figure 7: Synchronized parallel composition of PCAS.

In subsection 6.1, we present an example of a simple linear hybrid system, which is representative for the class of hybrid systems that requires the synchronized environment, in order to model a functionally correct behavior.

6.1 Example - Hybrid system analysis

Let us consider the abstract model of a hybrid control system, which evolves according to the function plotted in Figure 8, above the time axis. The output Y is either increased or decreased, at different speeds (v_1, v_2, v_3), respectively. From an observer point of view, we are interested in the behavior of the system (modeled by CAS $\mathcal{S}_1$) between the output value Y_L, and the output value Y_H. As one may notice in Figure 8, the continuous evolution of the hybrid model has a number of discontinuities. The specification requires an external controller to take certain actions whenever the trajectory change points are encountered. This happens when the output signal, Y, reaches the values Y_L, Y_1 or Y_2. Therefore, the external observer must be able to record, at any moment, the number of times when the output did change its trajectory. For this to happen, a second module ($\mathcal{S}_2$) is brought into the system representation. The CAS $\mathcal{S}_2$ models a counter, which provides the observer with the required information that has to be supplied to the external controller. The counter increments the value of an observable variable ($counter$), each time the condition $Y = Y_L \ \vee Y = Y_1 \vee Y = Y_2$ holds. The intended behavior of the counter is shown in Figure 8, under the time axis.

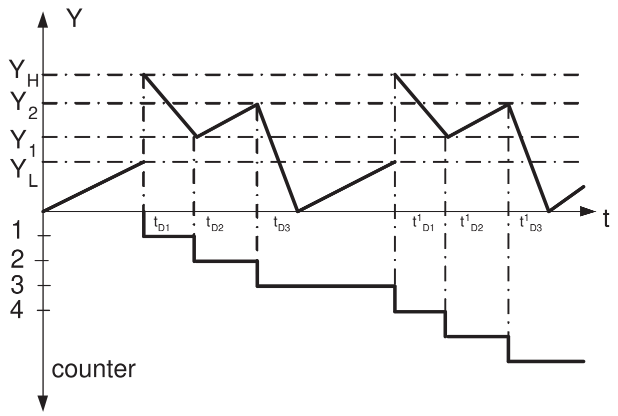

Figure 8: The graph of the timed evolution of variables Y and $counter$.

Even if the system is simple enough to be designed as a monolith, we would rather design it modularly, to create the premises for further extensions, which may require additional modules. The two CAS modules, $\mathcal{S}_1$, $\mathcal{S}_2$, are represented in Figure 9.

Interleaved design approach. Following the interleaved execution model, of the parallel composition $S = \mathcal{S}_1||\mathcal{S}_2$, at some moment in time, when $Y.now = Y_L$, both first actions of $\mathcal{S}_1$ and $\mathcal{S}_2$ are simultaneously enabled. If the controller chooses for execution the respective action of $\mathcal{S}_1$, first, the variable Y will be updated to Y_H, therefore disabling the enabled action of $\mathcal{S}_2$. Thus, the counter will miss to record this trajectory change. Alternatively, in both other situations when such an event happens, that

$$
\begin{array}{ll}
 & \mathcal{S}_1(Y : \mathsf{Real}_+ \to \mathsf{Real}_+) \\
\triangleq & \mathsf{begin} \\
 & \quad Y :- (\lambda t \cdot 0); \\
 & \quad \mathsf{do}\ \ Y.now = 0 \to Y :- (\lambda t \cdot v_1 * (t - now)) \\
 & \quad\quad [\!]\ Y.now = Y_L \to Y :- (\lambda t \cdot Y_H - v_2 * (t - now)) \\
 & \quad\quad [\!]\ Y.now = Y_1 \to Y :- (\lambda t \cdot Y_1 + v_1 * (t - now)) \\
 & \quad\quad [\!]\ Y.now = Y_2 \to Y :- (\lambda t \cdot Y_2 - v_3 * (t - now)) \\
 & \quad \mathsf{od} \\
 & \mathsf{end}
\end{array}
$$

$$
\begin{array}{ll}
 & \mathcal{S}_2(Y : \mathsf{Real}_+ \to \mathsf{Real}_+, counter : \mathsf{Real}_+ \to \mathsf{Nat}) \\
\triangleq & \mathsf{begin} \\
 & \quad counter :- (\lambda t \cdot 0)\,;\, Y :- (\lambda t \cdot 0); \\
 & \quad \mathsf{do}\ \ Y.now = Y_L \vee Y.now = Y_1 \vee Y.now = Y_2 \to \\
 & \quad\quad\quad counter :- (\lambda t \cdot counter.now + 1) \\
 & \quad \mathsf{od} \\
 & \mathsf{end}
\end{array}
$$

Figure 9: The timed action systems $\overline{\mathcal{S}}_1$ and $\overline{\mathcal{S}}_2$.

is, when $Y.now = Y_1$ and $Y.now = Y_2$, the interleaved model allows a correct update of the variable $counter$, since the value $Y.now$ is the same as the one mentioned by the guard of the first action in $\mathcal{S}_2$. Hence, the "old" value of Y is still possible to be observed by $\mathcal{S}_2$, even though the latter has been selected for execution in the second round.

This situation can be solved by adding extra information to the component systems, regarding their communication, or by employing other operators on CAS, which could determine the component systems to react in a way that produces a correct output. However, either of the solutions implies extra coding effort, and moreover, it deteriorates the system modularity.

Synchronized design approach. Let us now compose the CAS $\mathcal{S}_1$ and $\mathcal{S}_2$, by using our newly defined operator, '$\sharp$', given the fact that they are PCAS. As a result, we get the new PCAS $\mathcal{S}_{new} = \mathcal{S}_1 \sharp \mathcal{S}_2$. Then, we translate $\mathcal{S}_{new}$ into $\overline{\mathcal{S}}_{new}$ (Figure 10), by applying the definition given in Figure 7.

If we repeat the previously described scenario, when $Y.now = Y_L$, the semantics of $\overline{\mathcal{S}}_{new}$ lets us preserve the old values of Y, thus enabling the system $\overline{\mathcal{S}}_2$ to correctly update the variable $counter$, even when $\overline{\mathcal{S}}_2$ is selected for execution after $\overline{\mathcal{S}}_1$. Thus, due to the synchronized semantics, we can design the overall system modularly, by simply plugging $\mathcal{S}_1$ and $\mathcal{S}_2$ together, without encoding any kind of communication between these subsystems. Also, in case we need to add similar modules to the composed system, the synchronized composition lets us reuse the already existing components, and at the same time it ensures correct outputs to all inputs.

Finally, since the semantics of PCAS is given in terms of discrete AS (6), the theoretical results of the discrete synchronized proper AS apply to PCAS, too.

$\overline{\mathcal{S}}_{new}(counter : \mathsf{Real}_+ \to \mathsf{Nat})$
$\triangleq$ begin var $Y, Y_c : \mathsf{Real}_+ \to \mathsf{Real}_+, counter_c : \mathsf{Real}_+ \to \mathsf{Nat}, now, now_c : \mathsf{Real}_+,$
$sel[1..n] : \mathsf{Real}_+ \to \mathsf{Bool}, run : \mathsf{Real}_+ \to \mathsf{Nat}, u_1, u_2 : \mathsf{Real}_+ \to \mathsf{Bool}$ •
$counter, counter_c :- \ , run(\lambda t \cdot 0) ; Y, Y_c :- (\lambda t \cdot 0) ; u_1, u_2 :- (\lambda t \cdot false);$
$sel :- (\lambda t \cdot false) ; N ; now_c := now;$
do
$\neg sel[1].now \wedge run.now = 0 \to sel[1] :- (\lambda t \cdot true) ; run :- (\lambda t \cdot 1)$
$[\!] \ \neg sel[2].now \wedge run.now = 0 \to sel[2] :- (\lambda t \cdot true) ; run :- (\lambda t \cdot 2)$
$[\!] \ (run.now = 1 \wedge \neg u_1.now \wedge Y.now = 0 \to Y_c :- (\lambda t \cdot v_1 * (t - now))$
$[\!] \ run.now = 1 \wedge \neg u_1.now \wedge Y.now = Y_L \to Y_c :- (\lambda t \cdot Y_H - v_2 * (t - now))$
$[\!] \ run.now = 1 \wedge \neg u_1.now \wedge Y.now = Y_1 \to Y_c :- (\lambda t \cdot Y_1 + v_1 * (t - now))$
$[\!] \ run.now = 1 \wedge \neg u_1.now \wedge Y.now = Y_2 \to Y_c :- (\lambda t \cdot Y_2 - v_3 * (t - now)));$
$run :- (\lambda t \cdot 0) ; u_1 :- (\lambda t \cdot true) ; N$
$[\!] \ run.now = 1 \wedge \neg u_1.now \wedge \neg(Y.now = 0 \vee Y.now = Y_L \vee Y.now = Y_1 \vee Y.now = Y_2) \to$
$run :- (\lambda t \cdot 0) ; u_1 :- (\lambda t \cdot true) ; N$
$[\!] \ run.now = 2 \wedge \neg u_2.now \wedge (Y.now = Y_L \vee Y.now = Y_1 \vee Y.now = Y_2) \to$
$counter_c :- (\lambda t \cdot counter.now + 1) ; run :- (\lambda t \cdot 0) ; u_2 :- (\lambda t \cdot true) ; N$
$[\!] \ run.now = 2 \wedge \neg u_2.now \wedge \neg(Y.now = Y_L \vee Y.now = Y_1 \vee Y.now = Y_2) \to$
$run :- (\lambda t \cdot 0) ; u_2 :- (\lambda t \cdot true) ; N$
$[\!] \ sel.now \wedge run.now = 0 \wedge now \neq now_c \to Y :- Y_c ; counter :- counter_c;$
$sel :- (\lambda t \cdot false) ; u_1, u_2 :- (\lambda t \cdot false) ; now_c := now$
$[\!] \ sel.now \wedge run.now = 0 \wedge now = now_c \to Y :- Y_c ; counter :- counter_c$
od
end

Figure 10: The system $\overline{\mathcal{S}}_{new}$.

7 Conclusions

This study was motivated by an analysis of control aspects and of modular design techniques, as supported by the current AS formal framework. We exemplified that the interleaved model of concurrency may not suffice for modeling parallel reactive systems. Our solution comes as a synchronization mechanism, implying a new virtual execution model of AS, applicable to both discrete and hybrid designs. We eliminate intermediate results that could affect the global state, as the system gives complete answers to the stimuli provided by the environment.

The product operator of SCCS [Milner 1983] offers a somewhat similar approach to synchronization. However, while we synchronize on the updates of a group of variables, the SCCS approach is based on simultaneous execution of actions, which we only reach in the last execution round of a synchronized composition. Moreover, synchronization *restrictions* must be analyzed for each particular synchronized composition, thus decreasing the possibility of reuse.

Bellegarde et al. introduce a similar idea of synchronized parallel composition for event-B systems [Bellegarde et al. 2002]. In opposition to our model, which increases the *external* determinacy, while preserving the *internal* nondeterminism, the event-B solution preserves also the external nondeterminism. Moreover, a *gluing invariant* is necessary when synchronized modules are refined. This requirement comes from the fact that the synchronization is performed only with regard to selected events, collected in a synchronization specification. Therefore, the supplier of modules should also deliver to the system integrator, besides the modules themselves, the synchronization specification. From this point of view, the approach is similar to the one adopted by Back and von Wright [Back and von Wright 2003], where information about the invariants of all the composing subsystems must be known in order to perform refinements.

In the temporal logic of actions of Lamport [Lamport 2002], synchronization is specified as a way of applying *noninterleaving* to system design. This is reached by employing *joint actions*, a concept nonexistent in our framework. The conclusion, however, supports our point of view: interleaving "blurs" the distinction between the components used in design.

Treharne and Schneider [Treharne and Schneider 1999] employ CSP processes to control B-machines. The basic problems are raised by the interleaved execution semantics of both formalisms. Playing the state-based formalism (B), against the event-based approach (CSP), one may get a controllable environment for modeling certain applications. Our study shows, on the other hand, that it is possible, within the same state-based framework, to obtain the desired controlled behaviors.

An execution mechanism quite close to our synchronized environment is described by the semantics of statecharts, as offered in [Harel and Naamad 1996]. We can identify the execution of local actions that come from a single component, in an execution cycle, as a *compound transition* – CT. There are as many such *nonconflicting* CTs, as subsystems in a synchronized composition. By adding the initial and final state corresponding to a given execution cycle, we obtain a full CT.

In VHDL [Ashenden 2002], the update mechanisms for variables and signals are relatively similar to our solution concerning local and global variables. The difference resides in the fact that already executed processes (assimilated to action systems) may be rescheduled for execution, within the same VHDL execution cycle, due to new values of watched signals, assigned by other processes. The validity of such an approach is supported by the fact that, targeting a hardware implementation, the VHDL designer may assume that eventually, such reaction-triggering events will cease to appear (the combinational logic outputs will eventually settle to some value).

One important remark is that our approach does not necessarily address *synchronous* designs. The existence of a common clock signal is not suggested by any of our constructs. It is true that synchronous designs can be easily obtained from our models. This is furthermore supported by the underlying "synchrony hypothesis", as the time to perform individual actions is assumed to be null. From this perspective, we are close to the synchronous group of languages (Esterel, Lustre, Signal, etc.).

By providing the new virtual execution environment, we have tackled two important problems of system design: behavior control and modularity. The essential result of the study is mentioned by Corollary 1. Based on this, we can say that the system level integrator and the module designers gain an increased independency with respect to each other, during the design process. We believe that our achievement of using maximal synchronization to increase the modular design capabilities of the AS framework is a contribution that could be easily adapted to other similar formal environments.

Acknowledgements

The authors thank Viorel Preoteasa and Victor Bos for their invaluable help in clarifying certain aspects of the topic of the paper.

References

[Abrial 1996] J.-R. Abrial. Extending B without Changing it (for developing distributed systems). In 1^{st} Conference on the B method, pages 169-190, Nantes, France, November 1996.

[Ashenden 2002] P. Ashenden. The Designer's Guide to VHDL - Second Edition. Morgan Kaufmann Publishers, 2002.

[Back 1990] R. J. R. Back. Refinement Calculus, part II: Parallel and reactive programs. J. W. de Bakker, W.-P. de Roever, and G. Rozenberg. Stepwise Refinement of Distributed Systems: Models, Formalisms, Correctness. LNCS vol. 430. Springer-Verlag, pp. 67-93, 1990.

[Back and Suonio 1988] R. J. R. Back, R. Kurki-Suonio. Distributed Cooperation with Action Systems. ACM Transactions on Programming Languages and Systems, Vol. 10, No. 4.1988, pp. 513-554.

[Back et al. 2001] R. J. Back, L. Petre, I. Porres-Paltor. Continuous Action Systems as a Model for Hybrid Systems. Nordic Journal of Computing, vol. 8, pp. 2-21, 2001.

[Back and Sere 1994] R.-J. R. Back and K. Sere. From Action Systems to Modular Systems. Formal Methods Europe'94, Spain, October 1994, Lecture Notes in Computer Science. Springer–Verlag, 1994.

[Back and Sere 1994-2] R.J.R. Back, K. Sere. Action Systems with Synchronous Communication. Programming Concepts, Methods and Calculi. In E.-R. Olderog. IFIP Trans. A-56, pp. 107-126, 1994.

[Back and von Wright 1994] R. J. R. Back, J. von Wright. Trace refinement of action systems. Proceedings of CONCUR-94, Springer–Verlag, 1994.

[Back and von Wright 1998] R. J. R. Back, J. von Wright. Refinement Calculus: A Systematic Introduction. Springer–Verlag, 1998.

[Back and von Wright 2003] R. J. R. Back and J. von Wright. Compositional Action System Refinement. Formal Aspects of Computing (2-3): 103-117 2003.

[Back and Xu 1998] R.J.R. Back and Q. W. Xu. Refinement of Fair Action Systems. Acta Informatica 35, 131-165, 1998.

[Berry 1998] G. Berry. The Foundations of Esterel. Proof, Language and Interaction: Essays in Honour of Robin Milner, G. Plotkin, C. Stirling and M. Tofte, eds., MIT Press, 1998.

[Bellegarde et al. 2002] F.Bellegarde, J.Julliand, O.Kouchnarenko. Synchronized Parallel Composition of Event Systems in B. D. Bert et al.: ZB 2002, Springer-Verlag LNCS 2272, pp. 436-457, 2002.

[Behrends and Stirewalt 2000] R. Behrends and R. Stirewalt. The Universe Model: An Approach for Improving the Modularity and Reliability of Concurrent Programs. Proc. of ACM SIGSOFT Symposium on Foundations of Software Engineering (FSE'2000).

[Bornot and Sifakis 1998] S. Bornot and J. Sifakis. On the composition of hybrid systems. In First International Workshop Hybrid Systems : Computation and Control (HSCC'98), Springer-Verlag, LNCS 1386, pp. 49-63, 1998.

[Butler 1996] M. Butler. Stepwise Refinement of Communicating Systems. Science of Computer Programming 27(2):139-173, 1996.

[Butler 2000] M. Butler. csp2B: A Practical Approach to Combining CSP and B. Formal Aspects of Computing (2000) 12: 182–198.

[Borkowski 2001] J. Borkowski. Interrupt and Cancellation as Synchronization Methods. R. Wyrzykowski et al. (Eds.): PPAM 2001, LNCS 2328, pp. 3–9.

[Cavalcanti and Woodcock 2002] A. Cavalcanti and J. Woodcock. A Predicate Transformer Semantics for a Concurrent Language of Refinement. Communicating Process Architectures - 2002 J.Pascoe, P.Welch, R.Loader and V.Sunderam (Eds.) IOS Press, 2002, 147-165.

[Chandy and Misra 1988] K. M. Chandy and J. Misra. Parallel Program Design. A Foundation. Addison-Wesley, Reading, MA, 1988.

[Charpentier 2002] M. Charpentier. An Approach to Composition Motivated by wp. In The 5^{th} International Conference on Fundamental Approaches to Software Engineering, 2002, LNCS 2306, pages 1-14.

[Dijkstra 1976] E. W. Dijkstra. A Discipline of Programming. Prentice-Hall International, 1976.

[Gupta et al. 2002] V. Gupta, R. Jagadeesan, and V.A. Saraswat. Truly concurrent constraint programming. Theoretical Computer Science 278, 223-255, 2002.

[Harel and Naamad 1996] D. Harel, A. Naamad The STATEMATE Semantics of Statecharts. ACM Trans. on Software Engineering and Methodology, Vol. 5, No. 4, pp. 293-33, 1996.

[Hoare 1984] C. A. R. Hoare. Communicating sequential processes. Prentice–Hall, New-York, 1984.

[Hawblitzel and v. Eicken 2002] C. Hawblitzel, T. von Eicken. Luna: a Flexible Java Protection System. USENIX Association: Proceedings of the 5^{th} Symposium on Operating Systems Design and Implementation, 2002.

[Ifeachor and Jervis 1997] E.C.Ifeachor, B.W.Jervis Digital Signal Processing Practical Approach. Addison Wesley Publishing Company, 1997.

[Lamport 2002] L. Lamport. Specifying Systems - The TLA+ Language and Tools for Hardware and Software Engineers. Addison Wesley Publishing Company, 2002.

[Lynch and Tuttle 1989] N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. CWI Quarterly 2, pp. 219-246, 1989.

[Milner 1983] R. Milner. Calculi for synchrony and asynchrony. Theoretical Computer Science, Volume 25, Issue 3, pp. 267-310, 1983.

[Milner 1989] R. Milner. Communication and Concurrency. Prentice–Hall, London, 1989.

[Morgan 1998] C. Morgan. Programming from Specifications. Prentice-Hall International, 1998.

[Montanari and Rossi 1995] U. Montanari, F. Rossi. Concurrency and Concurrent Constraint Programming. In A. Podelski ed., Constraint Programming: Basics and Trends, Springer-Verlag, LNCS 910, pp. 171-193, 1995.

[Marlow et al 2001] S. Marlow, S. Peyton, J. A. Moran, J. Reppy Asynchronous Exceptions in Haskell. Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation, 2001, pp. 274–285.

[Olderog 1989] E-R. Olderog. Correctness of Concurrent Processes. In G.Goos, J. Hartmanis, Mathematical Foundations of Computer Science 1989.Springer-Verlag, LNCS 379, pp. 107-132.

[Petri 1966] C. A. Petri. Communication with Automata. Tech. Rep. RADC-TR-65-377, Vol. 1, Suppl. 1, Applied Data Research, Princeton, NJ, 1966.

[Plosila 1999] J. Plosila. Self-Timed Circuit Design - The Action Systems Approach. Ph.D. Thesis, University of Turku, Dep. of Applied Physics, Lab. of Electronics and Information Technology, Turku, Finland, 1999.

[Rudys and Wallach 2002] A. Rudys, D. S. Wallach. Termination in Language-Based Systems. ACM Trans. on Information and System Security Vol. 5, Issue 2, 2002, pp. 138–168.

[Seceleanu 2001] T. Seceleanu. Systematic Design of Synchronous Digital Circuits. Ph.D. Thesis, Abo Akademi, Turku, Finland, 2001.

[Sekerinski and Sere 1998] E. Sekerinski, K. Sere. A Theory of Prioritized Composition. The Computer Journal, VOL. 39, No 8, pp. 701–712. The British Computer Society. Oxford University Press.

[Theodoropoulos 2002] G. K. Theodoropoulos. Distributed Simulation of Asynchronous Hardware: The Program Driven Synchronization Protocol. Journal of Parallel and Distributed Computing, Vol. 62, Issue 4, April 2002, Pages 622-655.

[Treharne and Schneider 1999] H.Treharne and S.Schneider. Using a Process Algebra to control B Operations. In K.Araki, A.Galloway, K.Taguchi (Eds.), Proceedings of the 1^{st} International Conference on Integrated Formal Methods, IFM 99. Springer 1999, pp. 437-456.

Appendix

Before presenting the proofs of the statements of the paper, we introduce some results that help us achieve our goals.
• We will make use of Corollary 27 proved in [Back and von Wright 1999]:

Corollary 2 *Assume that G and H are conjunctive predicate transformers and that $g \wedge h \equiv false$. Then*

$$\mathsf{do}\ g \rightarrow G \,[\!]\ h \rightarrow H\ \mathsf{od}\ =\ \mathsf{do}\ h \rightarrow H\ \mathsf{od}\ ;\ \mathsf{do}\ g \rightarrow (G\ ;\ \mathsf{do}\ h \rightarrow H\ \mathsf{od}\ )\ \mathsf{od}$$

In the above, intuitively speaking, the condition $g \wedge h \equiv false$ states that the statements G and H exclude each other, that is, they cannot be enabled simultaneously.
• We will also make use of the Theorem 31 proved in [Back and von Wright 1999]:

Theorem 3 *Assume that G and H are conjunctive predicate transformers. Assume further that $\mathrm{wp}(H, true) \equiv true \wedge g \notin wH$, which means that statement H terminates and preserves g. Then*

$$\mathsf{do}\ g \rightarrow G \,[\!]\ \neg g \wedge h \rightarrow H\ \mathsf{od}\ =\ \mathsf{do}\ g \rightarrow G\ \mathsf{od}\ ;\ \mathsf{do}\ h \rightarrow H\ \mathsf{od} \qquad (7)$$

• We recall some of the weakest precondition rules [Dijkstra 1976], which we apply:

1. wp rule for guarded action: $\mathrm{wp}(g \rightarrow A, Q) \ \hat{=}\ g \Rightarrow \mathrm{wp}(A, Q)$
2. wp rule for choice: $\mathrm{wp}(A_1 \,[\!]\, A_2, Q) \ \hat{=}\ \mathrm{wp}(A_1, Q) \wedge \mathrm{wp}(A_2, Q))$
3. wp rule for assignment statement: $\mathrm{wp}(x, Q) \ \hat{=}\ Q[e/x]$
4. wp rule for sequential composition: $\mathrm{wp}(A_1\ ; A_2, Q) \ \hat{=}\ \mathrm{wp}(A_1, \mathrm{wp}(A_2, Q))$

• We additionally recall the definition of a loop as the least fixed point of the unfolding function [Back and von Wright 1998]:

$$\mathsf{do}\ g \rightarrow S\ \mathsf{od}\ \ \hat{=}\ (\mu X \cdot\ \mathsf{if}\ g\ \mathsf{then}\ S\ ; X\ \mathsf{else\ skip\ fi}) \qquad (8)$$

• We state here another helpful theorem, and three loop transformation rules (g, α predicates), as follows.

Theorem 4 *Assume that G, H and W are conjunctive predicate transformers. Then*

$$(G \,[\!]\, H)\ ; W = (G\ ; W) \,[\!]\, (H\ ; W)$$

• Loop elimination rule [Back and von Wright 1998].

$$\{\neg g\}\ ;\ \mathsf{do}\ g \rightarrow S\ \mathsf{od}\ = \{\neg g\} \qquad (9)$$

• Remove one iteration loop.

$$\{g\}\ ;\ \mathsf{do}\ g \rightarrow S\ ; \{\neg g\}\ \mathsf{od}\ = S\ ; \{\neg g\} \qquad (10)$$

Proof.

$$
\begin{aligned}
& \{g\}\,;\ \mathsf{do}\ g \to S\,;\{\neg g\}\ \mathsf{od} \\
= & \{\text{ definition } (8),\ \text{unfolding}\} \\
& \{g\}\,; \mathsf{if}\ g\ \mathsf{then}\ S\,;\{\neg g\}\,;\ \mathsf{do}\ g \to S\,;\{\neg g\}\ \mathsf{od}\ \mathsf{else}\ \mathsf{skip}\ \mathsf{fi} \\
= & \{\text{logic}\} \\
& S\,;\{\neg g\}\,;\ \mathsf{do}\ g \to S\,;\{\neg g\}\ \mathsf{od} \\
= & \{\text{loop elimination rule } (9)\} \\
& S\,;\{\neg g\}
\end{aligned}
$$

• Propagation of assertion inside loop [Back and von Wright 1998]:

$$\{\alpha\}\,;\ \mathsf{do}\ g \to S\ \mathsf{od}\ = \{\alpha\}\,;\ \mathsf{do}\ g \to \{\alpha\}\,; S\ \mathsf{od} \tag{11}$$

A Proof of Theorem 1

(a) By Definition 2, the synchronized parallel composition of the proper action systems

$$
\begin{aligned}
\mathcal{A}_1(z_1) &\triangleq \mathsf{begin}\ \mathsf{var}\ x_1 \bullet\ Init_1\,;\ \mathsf{do}\ g_L^1 \to L_1 \parallel g_S^1 \to S_1\ \mathsf{od}\ \mathsf{end} \\
\mathcal{A}_2(z_2) &\triangleq \mathsf{begin}\ \mathsf{var}\ x_2 \bullet\ Init_2\,;\ \mathsf{do}\ g_L^2 \to L_2 \parallel g_S^2 \to S_2\ \mathsf{od}\ \mathsf{end}
\end{aligned}
$$

is given by the system

$$
\begin{aligned}
& \mathcal{P}(z) \\
\triangleq\ & \mathsf{begin}\ \mathsf{var}\ x\,; sel[1..2] : \mathsf{Bool}\,; run : \mathsf{Nat} \bullet Init; \\
& \quad \mathsf{do}\ gg_A \to (run = 0 \wedge \neg sel[1] \to sel[1] := true\,; run := 1 \\
& \qquad \parallel run = 0 \wedge \neg sel[2] \to sel[2] := true\,; run := 2 \\
& \qquad \parallel run = 1 \wedge g_L^1 \to L_1 \parallel run = 1 \wedge \neg g_L^1 \wedge g_S^1 \to wS_1c := wS_1\,; S_1'\,; run := 0 \\
& \qquad\quad \parallel run = 1 \wedge \neg gg_{A_1} \to run := 0 \\
& \qquad \parallel run = 2 \wedge g_L^2 \to L_2 \parallel run = 2 \wedge \neg g_L^2 \wedge g_S^2 \to wS_2c := wS_2\,; S_2'\,; run := 0 \\
& \qquad\quad \parallel run = 2 \wedge \neg gg_{A_2} \to run := 0) \\
& \qquad \parallel \quad sel \wedge run = 0 \to Update\,; sel := false \\
& \quad \mathsf{od} \\
& \mathsf{end}
\end{aligned}
$$

We denote the actions of $\mathcal{P}$ as (where $j \in [1..2]$):

$$
\begin{aligned}
Sel &\triangleq run = 0 \to Sel_1 \parallel Sel_2 \\
Sel_1 &\triangleq \neg sel[1] \to sel[1] := true\,; run := 1 \\
Sel_2 &\triangleq \neg sel[2] \to sel[2] := true\,; run := 2 \\
A_j &\triangleq A_j^1 \parallel A_j^2 \parallel A_j^3 \\
A_j^1 &\triangleq run = j \wedge g_L^j \to L_j \\
A_j^2 &\triangleq run = j \wedge \neg g_L^j \wedge g_S^j \to C_j\,; S_j'\,; run := 0 \\
A_j^3 &\triangleq run = j \wedge \neg gg_{A_j} \to run := 0 \\
C_j &\triangleq wS_jc := wS_j \\
U &\triangleq sel \wedge run = 0 \to Update\,; sel := false
\end{aligned}
$$

Notice that the composition $L \triangleq gg_A \rightarrow Sel [\!] A_1 [\!] A_2$ constitutes the local action of $\mathcal{P}$, while the action U is its global action. The first two requirements for showing that $\mathcal{P}$ is a proper AS are immediate. Next, we analyze the third one, that is, we have to show that $wp(\text{ do } L \text{ od }, \neg gL \wedge gU) \equiv true$.

We start by observing that only the situation when $gg_A \equiv true$ is of interest, otherwise the whole system $\mathcal{P}$ is disabled. Therefore, we only have to show that

$$wp(\text{ do } Sel [\!] A_1 [\!] A_2 \text{ od }, \neg gL \wedge gU) \equiv true$$

We proceed as follows:

$$\begin{array}{ll}
& \text{do } Sel [\!] A_1 [\!] A_2 \text{ od} \\
= & \{ \text{ Corollary 2 } \} \\
& \text{do } Sel \text{ od } ; \text{ do } A_1 [\!] A_2 ; \text{ do } Sel \text{ od od} \\
\geq & \{ \text{Init or } A_j^2 [\!] A_j^3 \text{ establish } run = 0, \ \{p\} \leq \text{skip}, \text{ introduce assertions } \} \\
& \{run = 0\} ; \text{ do } run = 0 \rightarrow Sel_1 [\!] Sel_2 ; \{run \neq 0\} \text{ od } ; \\
& \qquad \text{do } A_1 [\!] A_2 ; \text{ do } Sel \text{ od od} \\
= & \{ \text{ rule (10), drop assertion } \} \\
& (Sel_1 [\!] Sel_2) ; \text{ do } A_1 [\!] A_2 ; \text{ do } Sel \text{ od od} \\
= & \{ \text{ Theorem 4 } \} \\
& (Sel_1 ; \text{ do } A_1 [\!] A_2 ; \text{ do } Sel \text{ od od }) \\
& \qquad [\!] (Sel_2 ; \text{ do } A_1 [\!] A_2 ; \text{ do } Sel \text{ od od }) \\
= & \{ \text{ Theorem 4, notation: } Choice_2 \triangleq (Sel_2 ; \text{ do } A_1 [\!] A_2 ; \text{ do } Sel \text{ od od })\} \\
& (Sel_1 ; \text{ do } A_1 ; \text{ do } Sel \text{ od } [\!] A_2 ; \text{ do } Sel \text{ od od }) [\!] Choice_2 \\
= & \{ \text{ Corollary 2 } \} \\
& (Sel_1 ; \text{ do } A_1 ; \text{ do } Sel \text{ od od } ; \\
& \qquad \text{do } A_2 ; \text{ do } Sel \text{ od } ; \text{ do } A_1 ; \text{ do } Sel \text{ od od od }) [\!] Choice_2
\end{array} \tag{13}$$

We continue by focusing on the sequence $Sel_1 ; \text{ do } A_1 ; \text{ do } Sel \text{ od od}$:

$$\begin{array}{ll}
& Sel_1 ; \text{ do } A_1 ; \text{ do } Sel \text{ od od} \\
= & \{ \text{ definition of } A_1 \text{, Theorem 4 } \} \\
& Sel_1 ; \text{ do } A_1^1 ; \text{ do } Sel \text{ od } [\!] A_1^2 ; \text{ do } Sel \text{ od } [\!] A_1^3 ; \text{ do } Sel \text{ od od} \\
= & \{ \text{ Theorem 3 w.r.t. } A_1^1 ; \text{ do } Sel \text{ od } [\!] A_1^2 ; \text{ do } Sel \text{ od and } A_1^3\} \\
& Sel_1 ; \text{ do } A_1^1 ; \text{ do } Sel \text{ od } [\!] A_1^2 ; \text{ do } Sel \text{ od od } ; \text{ do } A_1^3 ; \text{ do } Sel \text{ od od} \\
= & \{ \text{ Theorem 3 w.r.t. } A_1^1 ; \text{ do } Sel \text{ od and } A_1^2 ; \text{ do } Sel \text{ od } \} \\
& Sel_1 ; \text{ do } A_1^1 ; \text{ do } Sel \text{ od od } ; \text{ do } A_1^2 ; \text{ do } Sel \text{ od od } ; \\
& \qquad \text{do } A_1^3 ; \text{ do } Sel \text{ od od}
\end{array} \tag{14}$$

Next, since $gg_A \equiv true$, the last element of the sequence $\textsf{do}\ A_1^3\ ;\ \textsf{do}\ Sel\ \textsf{od}\ \textsf{od}$ can be replaced by skip. We focus on the first two terms of the sequence:

$$
\begin{aligned}
& Sel_1\ ;\ \textsf{do}\ A_1^1\ ;\ \textsf{do}\ Sel\ \textsf{od}\ \textsf{od} \\
=\ & \{\ run \notin wA_1^1\ \} \\
& Sel_1\ ;\ \textsf{do}\ A_1^1\ ; \{run = 1\}\ ;\ \textsf{do}\ Sel\ \textsf{od}\ \textsf{od} \\
=\ & \{\ \text{definition of } Sel, \text{ rule (9), drop assertion}\ \} \\
& Sel_1\ ;\ \textsf{do}\ A_1^1\ \textsf{od}
\end{aligned}
\tag{15}
$$

We already know ($\mathcal{A}_1$ is proper) that $\textsf{do}\ g_L^1 \to L_1\ \textsf{od}$ terminates and establishes $\neg g_L^1 \wedge g_S^1$. Hence, $(run = 1 \to\ \textsf{do}\ g_L^1 \to L_1\ \textsf{od}\ ) =\ \textsf{do}\ A_1^1\ \textsf{od}$ terminates and establishes $run = 1 \wedge \neg g_L^1 \wedge g_S^1$. As $sel[1] \notin wA_1^1$, we actually have that, after executing $\textsf{do}\ A_1^1\ \textsf{od}$, $sel[1] \wedge run = 1 \wedge \neg g_L^1 \wedge g_S^1$ holds.

We continue with the analysis of $\textsf{do}\ A_1^2\ ;\ \textsf{do}\ Sel\ \textsf{od}\ \textsf{od}$. Considering the above, we have

$$
\begin{aligned}
& \{sel[1] \wedge run = 1 \wedge \neg g_L^1 \wedge g_S^1\}\ ;\ \textsf{do}\ A_1^2\ ;\ \textsf{do}\ Sel\ \textsf{od}\ \textsf{od} \\
=\ & \{\text{general rule: } \{p \wedge q\} = \{p\}\ ; \{q\}\} \\
& \{sel[1]\}\ ; \{run = 1 \wedge \neg g_L^1 \wedge g_S^1\}\ ;\ \textsf{do}\ A_1^2\ ;\ \textsf{do}\ Sel\ \textsf{od}\ \textsf{od} \\
=\ & \{\ \text{rule (11), as } sel[1] \notin wA_1^2\ \} \\
& \{sel[1]\}\ ; \{run = 1 \wedge \neg g_L^1 \wedge g_S^1\}\ ;\ \textsf{do}\ A_1^2\ ; \{sel[1]\}\ ;\ \textsf{do}\ Sel\ \textsf{od}\ \textsf{od} \\
=\ & \{\ A_1^2 \text{ establishes } run = 0, \text{ strengthen assertion}\ \} \\
& \{sel[1]\}\ ; \{run = 1 \wedge \neg g_L^1 \wedge g_S^1\}\ ;\ \textsf{do}\ A_1^2\ ; \{sel[1] \wedge run = 0\}\ ;\ \textsf{do}\ Sel\ \textsf{od}\ \textsf{od} \\
=\ & \{\ \text{definition of } Sel, \text{ rewrite using context information } (\{sel[1] \wedge run = 0\})\ \} \\
& \{sel[1]\}\ ; \{run = 1 \wedge \neg g_L^1 \wedge g_S^1\}\ ;\ \textsf{do}\ A_1^2\ ; \{sel[1] \wedge run = 0\}\ ; \\
& \qquad\qquad \textsf{do}\ run = 0 \to Sel_2\ \textsf{od}\ \textsf{od} \\
=\ & \{\ \text{introduce assertion } \{run \neq 0\}\} \\
& \{sel[1]\}\ ; \{run = 1 \wedge \neg g_L^1 \wedge g_S^1\}\ ;\ \textsf{do}\ A_1^2\ ; \\
& \qquad\qquad \{sel[1] \wedge run = 0\}\ ;\ \textsf{do}\ run = 0 \to Sel_2\ ; \{run \neq 0\}\ \textsf{od}\ \textsf{od} \\
=\ & \{\ \text{rule (10), drop assertions}\ \} \\
& \textsf{do}\ A_1^2\ ; Sel_2\ \textsf{od}
\end{aligned}
\tag{16}
$$

The above loop terminates, since both A_1^2 and Sel_2 terminate. Moreover, $wp(\ \textsf{do}\ A_1^2; Sel_2\ \textsf{od}\ , run = 2 \wedge sel[2]) \equiv true$.

Applying a similar reasoning for the action $Choice_2$, we eventually come to the conclusion that:

$$\text{wp}(\ \textsf{do}\ Sel [\!]\ A_1 [\!]\ A_2\ \textsf{od}\ , sel \wedge run = 0) \equiv true,$$

which means that the local action terminates, and it enables the execution of the global action of the system $\mathcal{P}$.

It is easy to check that the other requirements of Definition 1 are also satisfied, thus, $\mathcal{P}$ is a proper AS.
(b) Follows from the commutativity of the choice operator.

B Proof that Lemma 2 is a valid trace refinement lemma for proper AS

Since the first four requirements of Lemma 2 are only adaptations of the original four requirements of Lemmma 1, we concentrate here on showing that the fifth requirement of Lemma 2 implies the corresponding requirement of the original trace refinement lemma:

$$\begin{aligned}&R \wedge I \Rightarrow \mathrm{wp}(\ \mathsf{do}\ g_X \rightarrow X \,\|\, g_L^C \rightarrow L'_A\ \mathsf{od}\ , \neg(g_X \vee g_L^C) \wedge g_S^C)\\ \Rightarrow\ &R \wedge I \Rightarrow \mathrm{wp}(\ \mathsf{do}\ g_X \rightarrow X\ \mathsf{od}\ , true)\end{aligned}$$

We consider the definition of the weakest precondition of a loop, to establish some postcondition Q, as given by Dijkstra [Dijkstra 1976]:

$$\begin{aligned}\mathrm{wp}(\ \mathsf{do}\ g \rightarrow A\ \mathsf{od}\ , Q) = \exists k \geq 0 \bullet H_k,&\\ H_0 = Q \wedge \neg g,&\\ H_k = H_0 \vee wp(g \rightarrow A, H_{k-1})&\end{aligned} \tag{17}$$

In our context, established by Lemma 2, the new local action of the refined system ($\mathcal{C}$) is:

$$L_{new} = g_X \rightarrow X \,\|\, g_L^C \rightarrow L'_A$$

We need to prove that $\mathrm{wp}(\ \mathsf{do}\ L_{new}\ \mathsf{od}\ , \neg(g_X \vee g_L^C) \wedge g_S^C) \Rightarrow \mathrm{wp}(\ \mathsf{do}\ g_X \rightarrow X\ \mathsf{od}\ , true)$ holds.

In order to compute $\mathrm{wp}(\ \mathsf{do}\ L_{new}\ \mathsf{od}\ , \neg(g_X \vee g_L^C) \wedge g_S^C)$, we apply (17):

$$\begin{aligned}H'_0 &= \neg(g_X \vee g_L^C) \wedge g_S^C\\ H'_k &= H'_0 \vee \mathrm{wp}(g_X \rightarrow X \,\|\, g_L^C \rightarrow L'_A, H'_{k-1})\\ &= \{\ \text{wp rule for choice}\ \}\\ &\quad H'_0 \vee (\mathrm{wp}(g_X \rightarrow X, H'_{k-1}) \wedge \mathrm{wp}(g_L^C \rightarrow L'_A, H'_{k-1}))\\ &= \{\ \text{logic}\ \}\\ &\quad (H'_0 \vee \mathrm{wp}(g_X \rightarrow X, H'_{k-1})) \wedge (H'_0 \vee \mathrm{wp}(g_L^C \rightarrow L'_A, H'_{k-1}))\\ &\Rightarrow \{\ \text{logic}\ \}\\ &\quad H'_0 \vee \mathrm{wp}(g_X \rightarrow X, H'_{k-1})\end{aligned} \tag{18}$$

We observe that $H'_k \Rightarrow H'_{k+1}$ (by induction and monotonicity of wp):

$$\begin{aligned}&H'_k\\ =\ &\{\ \text{definition}\ \}\\ &(H'_0 \vee \mathrm{wp}(g_X \rightarrow X, H'_{k-1})) \wedge (H'_0 \vee \mathrm{wp}(g_L^C \rightarrow L'_A, H'_{k-1}))\\ \Rightarrow\ &\{\ \text{assumption}\ H'_{k-1} \Rightarrow H'_k \}\\ &(H'_0 \vee \mathrm{wp}(g_X \rightarrow X, H'_k)) \wedge (H'_0 \vee \mathrm{wp}(g_L^C \rightarrow L'_A, H'_k))\\ =\ &\{\ \text{definition}\ \}\\ &H'_{k+1}\end{aligned} \tag{19}$$

Next, by applying (17), we compute $\mathrm{wp}(\ \mathsf{do}\ g_X \to X\ \mathsf{od}\ , true)$:

$$\begin{aligned} H_0^X &= \neg g_X \\ H_k^X &= H_0^X \vee \mathrm{wp}(g_X \to X, H_{k-1}^X) \\ &= \neg g_X \vee \mathrm{wp}(g_X \to X, H_{k-1}^X) \end{aligned}$$

In a similar manner as above (by induction and monotonicity of wp), we obtain that

$$H_k^X \Rightarrow H_{k+1}^X \tag{20}$$

Observe next that also:

$$\begin{aligned} & H'_0 \Rightarrow H_0^X \\ \Rightarrow & \{ \text{ monotonicity of wp} \} \\ & \mathrm{wp}(g_X \to X, H'_0) \Rightarrow \mathrm{wp}(g_X \to X, H_0^X) \\ \Rightarrow & \{ \text{ monotonicity of wp, induction, logic, (19), (20)} \} \\ & H'_0 \vee \mathrm{wp}(g_X \to X, H'_{k-1}) \Rightarrow H_0^X \vee \mathrm{wp}(g_X \to X, H_{k-1}^X) \end{aligned} \tag{21}$$

Summing up the results of (18) . . . (21) we conclude that

$$\mathrm{wp}(\ \mathsf{do}\ L_{new}\ \mathsf{od}\ , \neg(g_X \vee g_L^C) \wedge g_S^C) \Rightarrow \mathrm{wp}(\ \mathsf{do}\ g_X \to X\ \mathsf{od}\ , true)$$

Thus, considering that the requirements 1 to 5 of the Lemma 2 are satisfied, also the requirements of Lemma 1 are satisfied, hence, $\mathcal{A} \sqsubseteq \mathcal{C}$.

C Proof of Theorem 2

We assume the system $\mathcal{P}$ as being the synchronized composition of two AS:

$$\begin{aligned} & \mathcal{P}(z) \\ \triangleq\ & \mathsf{begin}\ \ \mathsf{var}\ x\ ;\ sel[1..2] : \mathsf{Bool}\ ;\ run : \mathsf{Nat} \bullet Init; \\ & \ \mathsf{do}\ gg_A \to (run = 0 \wedge \neg sel[1] \to sel[1] := true\ ;\ run := 1 \\ & \quad [\!]\ \ run = 0 \wedge \neg sel[2] \to sel[2] := true\ ;\ run := 2 \\ & \quad [\!]\ run = 1 \wedge g_L^1 \to L_1 [\!]\ run = 1 \wedge \neg g_L^1 \wedge g_S^1 \to wS_1c := wS_1\ ;\ S'_1\ ;\ run := 0 \\ & \qquad [\!]\ \ run = 1 \wedge \neg gg_{A_1} \to run := 0 \\ & \quad [\!]\ run = 2 \wedge g_L^2 \to L_2 [\!]\ run = 2 \wedge \neg g_L^2 \wedge g_S^2 \to wS_2c := wS_2\ ;\ S'_2\ ;\ run := 0 \\ & \qquad [\!]\ \ run = 2 \wedge \neg gg_{A_2} \to run := 0) \\ & \quad [\!]\ \ \ sel \wedge run = 0 \to Update\ ;\ sel := false \\ & \ \mathsf{od} \\ & \mathsf{end} \end{aligned}$$

First, we give, without proof, three simple invariants of system $\mathcal{P}$:

$$\begin{aligned} & 22.1 : (\textstyle\bigvee_{j \in [0..2]} (run = j)) \equiv true \\ & 22.2 : \forall k \in \{1, 2\}\ \bullet sel \wedge (run = 0) \Rightarrow g_S^k \\ & 22.3 : \forall k \in \{1, 2\}\ \bullet (run = k) \Rightarrow sel[k] \end{aligned} \tag{22}$$

We state further that

$$I_0^1 \triangleq I_1 \ \wedge \ (sel[1] \wedge \neg(run = 1) \Rightarrow I_1') \tag{23}$$

is an invariant of the system $\mathcal{P}$, where

- I_1 is the proper invariant respected by the system $\mathcal{A}_1$. Therefore, we also have that $g_S^1 \Rightarrow (I_1[w'S_1/wS_1, w] \equiv I_1[w'S_1/wS_1])$.
- $I_1' = I_1[wS_1^c/wS_1]$,

In the following, we show that I_0^1 is an invariant of every action of $\mathcal{P}$.

1. I_0^1 is preserved by action $A_1 \triangleq \neg sel[1] \wedge (run = 0) \rightarrow sel[1] := true ; run := 1$. We have:

$$
\begin{array}{ll}
 & wp(A_1, I_0^1) \\
\equiv & \{\text{definition of } A_1, wp \text{ rules for guarded action, sequential comp., assignment}\} \\
 & sel[1] \vee \neg(run = 0) \vee I_0^1[true/sel[1], 1/run]) \\
\equiv & \{\text{definition of } I_0^1,\ sel, run \text{ do not appear in } I_1, \text{ or in } I_1', \text{ logic}\} \\
 & sel[1] \vee \neg(run = 0) \vee I_1 \\
\Leftarrow & \{\text{definition (23), logic}\} \\
 & I_0^1
\end{array}
$$

2. I_0^1 is preserved by action $A_2 \triangleq \neg sel[2] \wedge (run = 0) \rightarrow sel[2] := true ; run := 2$. We have:

$$
\begin{array}{ll}
 & wp(A_2, I_0^1) \\
\equiv & \{\text{definition of } A_2, wp \text{ rules for guarded action, sequential comp., assignment}\} \\
 & sel[2] \vee \neg(run = 0) \vee I_0^1[true/sel[2], 2/run] \\
\equiv & \{\text{definition of } I_0^1,\ sel, run \text{ do not appear in } I_1, \text{ or in } I_1', \text{ logic}\} \\
 & sel[k] \vee \neg(run = 0) \vee (I_1 \wedge (sel[2] \Rightarrow I_1')) \\
\Leftarrow & \{\text{logic, relation (22.1)}\} \\
 & I_0^1
\end{array}
$$

3. That I_0^1 is preserved by the action $A_3 \triangleq (run = 2) \wedge g_L^2 \rightarrow L_2$, follows from the fact that A_3 does not write any of the variables mentioned by I_0^1, therefore the latter is an invariant, trivially.
4. I_0^1 is preserved by the action $A_4 \triangleq (run = 1) \wedge g_L^1 \rightarrow L_1$ comes from the fact that I is an invariant of $g_L^1 \rightarrow L_1$.
5. I_0^1 is preserved by the action $A_5 \triangleq (run = 1) \wedge \neg g_L^1 \wedge g_S^1 \rightarrow wS_1c := wS_1 ; S_1' ; run := 0,\ where\ S_1' = S_1[wS_1c/wS_1]$. We first have that:

$$
\begin{aligned}
& wp(x := y \; ; S[x/y], Q[x/y]) \\
\equiv \; & \\
& wp(x := y, wp(S[x/y], Q[x/y])) \\
\equiv \; & \\
& wp(x := y, wp(S, Q)[x/y]) \\
\equiv \; & \\
& (wp(S, Q)[x/y])[y/x] \\
\equiv \; & \\
& wp(S, Q)
\end{aligned}
\tag{24}
$$

Next, we get:

$$
\begin{aligned}
& wp(A_5, I_0^1) \\
\equiv \; & \{wp \text{ rules for guarded action, sequential comp., assignment}\} \\
& \neg(run = 1) \vee g_L^1 \vee \neg g_S^1 \vee wp(wS_1c := wS_1 \; ; S_1', I_0^1[0/run]) \\
\equiv \; & \{\text{definition of } I_0^1,\ run \text{ does not appear in } I_1,\ \text{ or in } I_1'\} \\
& \neg(run = 1) \vee g_L^1 \vee \neg g_S^1 \vee wp(wS_1c := wS_1 \; ; S_1', I_1 \wedge (sel[1] \Rightarrow I_1')) \\
\equiv \; & \{wp \text{ rule for conjunctive statements, } wS_1c \text{ does not appear in } I_1\} \\
& \neg(run = 1) \vee g_L^1 \vee \neg g_S^1 \vee (I_1 \wedge wp(wS_1c := wS_1 \; ; S_1', (sel[1] \Rightarrow I_1'))) \\
\equiv \; & \{wp \text{ rule for sequential composition}\} \\
& \neg(run = 1) \vee g_L^1 \vee \neg g_S^1 \vee (I_1 \wedge wp(wS_1c := wS_1, wp(S_1', (sel[1] \Rightarrow I_1')))) \\
\equiv \; & \{\text{definition of } I_1', \text{ relation (24), } sel \text{ does not mention } wS_1 \text{ or } wS_1c\} \\
& \neg(run = 1) \vee g_L^1 \vee \neg g_S^1 \vee (I_1 \wedge wp(S_1, (sel[1] \Rightarrow I_1))) \\
\Leftarrow \; & \{wp(S_1, \neg sel[1] \vee I_1) \Leftarrow wp(S_1, I_1) \vee wp(S_1, \neg sel[1])\} \\
& \neg(run = 1) \vee g_L^1 \vee \neg g_S^1 \vee (I_1 \wedge (wp(S_1, I_1) \vee wp(S_1, \neg sel[1]))) \\
\Leftarrow \; & \{\text{logic}\} \\
& \neg(run = 1) \vee g_L^1 \vee \neg g_S^1 \vee (I_1 \wedge wp(S_1, I_1)) \\
\Leftarrow \; & \{I_1 \text{ is invariant of the original system: } I_1 \Rightarrow wp(S_1, I_1), \text{ logic}\} \\
& I_1 \\
\Leftarrow \; & \{\text{definition of } I_0^1, \text{ logic}\} \\
& I_0^1
\end{aligned}
$$

6. The fact that I_0^1 is preserved by the action $A_6 \triangleq (run = 1) \wedge \neg g_L^1 \wedge \neg g_S^1 \rightarrow run := 0$ follows the lines of the previous proof.

7. I_0^1 is preserved by the action $A_7 \triangleq (run = 2) \wedge \neg g_L^2 \wedge g_S^2 \rightarrow wS_2c := wS_2 \; ; S_2' \; ; run := 0$, where $S_2' = S_2[wS_2c/wS_2]$:

$$
\begin{aligned}
& wp(A_6, I_0^1) \\
\equiv\ & \{wp \text{ rules for guarded action, sequential comp., assignment}\} \\
& \neg(run = 2) \vee g_L^2 \vee \neg g_S^2 \vee wp(wS_2c := wS_2\ ;\ S_2', I_0^1[0/run]) \\
\equiv\ & \{\text{definition of } I_0^1,\ run, wS_2c, wS_2 \text{ do not appear in } I_1, \text{ or in } I_1'\} \\
& \neg(run = 2) \vee g_L^2 \vee \neg g_S^2 \vee (I_1 \wedge (sel[1] \Rightarrow I_1')) \\
\Leftarrow\ & \{\text{logic, relation (22.1)}\} \\
& I_0^1
\end{aligned}
$$

8. The fact that I_0^1 is an invariant of the action $A_8 \triangleq (run = 2) \wedge \neg g_L^2 \wedge \neg g_S^2 \rightarrow run := 0$ has a similar proof to the one for action A_7.

9. Proof of the fact that I_0^1 is preserved by the action $U \triangleq sel \wedge (run = 0) \rightarrow wS_1 := wS_1c\ ; wS_2 := wS_2c\ ; sel := false$:

$$
\begin{aligned}
& wp(U, I_0^1) \\
\equiv\ & \{wp \text{ rules for guarded action, sequential comp., assignment}\} \\
& \neg sel \vee \neg(run = 0) \vee wp(wS_1 := wS_1c\ ; wS_2 := wS_2c, I_0^1[false/sel]) \\
\equiv\ & \{\text{definition of } I_0^1, \text{ succesive application of } wp \text{ rules}\} \\
& \neg sel \vee \neg(run = 0) \vee (I_1[wS_1c, wS_2c/wS_1, wS_2] \wedge \neg(run = 1)) \\
\equiv\ & \{\text{relation (22.1)}\} \\
& \neg sel \vee \neg(run = 0) \vee I_1[wS_1c, wS_2c/wS_1, wS_2]
\end{aligned}
$$

Further, we have to show that $I_0^1 \Rightarrow wp(U, I_0^1)$. We also know that I_1 is proper and that $sel \wedge (run = 0) \Rightarrow g_S^1$ (relation (22.2)).

We denote: $I_1'' = I_1[wS_1c, wS_2c/wS_1, wS_2]$. Then:

$$
\begin{aligned}
& I_0^1 \wedge (g_S^1 \Rightarrow I_1' \equiv I_1'') \wedge (sel \wedge (run = 0) \Rightarrow g_S^1) \\
\Rightarrow\ & \{\text{logic}, \neg sel = \neg sel_1 \vee \neg sel_2\} \\
& \neg sel \vee \neg(run = 0) \vee I_1''
\end{aligned}
$$

The above show that I_0^1 is an invariant of the system $\mathcal{P}$. In a similar manner, we can show that

$$I_0^2 \triangleq I_2 \ \wedge\ (sel[2] \wedge \neg(run = 2) \Rightarrow I_2')$$

is also an invariant of $\mathcal{P}$. Hence, $I \triangleq I_0^1 \wedge I_0^2$ is an invariant of $\mathcal{P}$.

Properness. Notice further that:

$$
\begin{aligned}
& sel \wedge run = 0 \Rightarrow (I_0^1[wS_1c, wS_2c/wS_1, wS_2, false/sel, z] \Rightarrow \\
& \qquad I_0^1[wS_1c, wS_2c/wS_1, wS_2, false/sel]) \\
\equiv\ & \{I_1 \text{ is proper:} \\
& g_S^1 \Rightarrow (I_1[wS_1c/wS_1, wS_2c/wS_2, false/sel, z] \equiv I_1[wS_1c/wS_1]), \\
& \text{relation (22.2), logic}\} \\
& true
\end{aligned}
\tag{25}
$$

Similarly, we obtain that $sel \wedge run = 0 \Rightarrow (I_0^1[wS_1c, wS_2c/wS_1, wS_2, false/sel] \Rightarrow I_0^1[wS_1c, wS_2c/wS_1, wS_2, false/sel, z])$,resulting in the conclusion that I_0^1 is a proper invariant of $\mathcal{P}$.

Repeating the above proof for the other invariant I_0^2 and summing up, we reach the conclusion that $I \triangleq I_0^1 \wedge I_0^2$ is a proper invariant of $\mathcal{P}$.

The results can be generalized to the synchronized composition of $k, k > 2$ proper AS.

D Proof of Corollary 1

Suppose that we have the proper AS $\mathcal{A}_j$, as part of the synchronized composition $\mathcal{P} = \mathcal{A}_1 \sharp..\sharp \mathcal{A}_n$. Additionally, I_j is some invariant respected by $\mathcal{A}_j$, and we also have that $\mathcal{A}_j \sqsubseteq_{R_j,I_j} \mathcal{A}'_j$, following the requirements of Lemma 2. Thus, $\mathcal{A}'_j$ is a proper AS, too. Consequently [Back and von Wright 2003], $I'_j \triangleq R_j \wedge I_j$ is an invariant of $\mathcal{A}'_j$. As R_j does not introduce any new global variables, it is independent of other variables than those of $\mathcal{A}_j, \mathcal{A}'_j$, and using a similar line of proof as in (25), one can prove that I'_j is proper.

We do not insist here on the (trivial, given the above assumptions) task of showing that $\mathcal{P} \sqsubseteq \mathcal{P}'$ ($\mathcal{P}' \triangleq \mathcal{A}_1 \sharp..\sharp \mathcal{A}'_j \sharp..\sharp \mathcal{A}_n$). Relevant is to show that, using the notations of Appendix C, $I' \triangleq I_0^1 \wedge .. \wedge I_0'^j \wedge .. \wedge I_0^n$ is an invariant of $\mathcal{P}'$. This is solved by following the same steps as in Appendix C. Moreover, we have

$$
\begin{aligned}
& I'_j \triangleq R_j \wedge I_j \\
\Rightarrow & \{ \text{ notation: } Q_j = sel[j] \wedge \neg(run = j) \Rightarrow I''_j, \\
& \quad I''_j \triangleq R \wedge I'_j[wS_1^c/wS_1], \\
& \quad \text{definition (23): } I_0'^j \triangleq R_j \wedge I_j \wedge Q_j, \\
& \quad \text{logic}\} \\
& I_0'^j \Rightarrow I_0^j \\
\Rightarrow & \{\text{logic}\} \\
& \quad I_0^1 \wedge .. \wedge I_0'^j \wedge .. \wedge I_0^n \\
& \Rightarrow I_0^1 \wedge .. \wedge I_0^j \wedge .. \wedge I_0^n \\
\Rightarrow & \{\text{notation}\} \\
& \quad I_1 \wedge Q_1 \wedge .. \wedge I'_j \wedge Q_j \wedge .. \wedge I_n \wedge Q_n \\
& \Rightarrow I_1 \wedge Q_1 \wedge .. \wedge I_j \wedge Q_j \wedge .. \wedge I_n \wedge Q_n \\
\Rightarrow & \{\text{logic}\} \\
& \quad I_1 \wedge .. \wedge I'_j \wedge .. \wedge I_n \\
& \Rightarrow I_1 \wedge .. \wedge I_j \wedge .. \wedge I_n
\end{aligned}
$$

The above illustrate the fact that $\mathcal{P} \sqsubseteq_{R_j,I_j} \mathcal{P}'$, and that $\mathcal{P}'$ still preserves the properties of each of the original AS $\mathcal{A}_k,\ k \in [1..n]$, as expressed by the conjunction $I_1 \wedge .. \wedge I_j \wedge .. \wedge I_n$.

E Refinement of *sys* $\mathcal{F}$

We consider the systems:

$$\begin{array}{rl} & \mathcal{F}(X, Z[0..N-2], Y : T) \\ \triangleq & \textsf{begin var } h[0..N-1] : T \quad \bullet\, X, Z[0..N-2], h[0..N-1], Y := x_0, z_0, h_0, y_0; \\ & \quad \textsf{do } Y := h[0] \times X + \sum_{k=1}^{N-1} h[k] \times Z[k-1] \textsf{ od} \\ & \textsf{end} \end{array}$$

and

$$\begin{array}{rl} & \mathcal{F}_S(X, Z[0..N-2], Y : T) \\ \triangleq & \textsf{begin var } h[0..N-1], temp : T\,;\, step : 0..N \quad \bullet \\ & \quad X, Z, h, Y := x_0, z_0, h_0, y_0\,;\, temp := 0\,;\, step := 0; \\ & \quad \textsf{do } step = 0 \to temp := 0\,;\, step := step + 1 \\ & \quad\quad [\!] \; step \in [1..N-1] \to temp := temp + h[step] \times Z[step-1]; \\ & \quad\quad\quad\quad\quad\quad\quad\quad step := step + 1 \\ & \quad\quad [\!] \; step = N \to Y := temp + X \times h[0]\,;\, step := 0 \\ & \quad \textsf{od} \\ & \textsf{end} \end{array}$$

We denote the actions of the above systems as:

$$\begin{aligned} A &\triangleq Y := X \times h[0] + \sum_{k=1}^{N-1} h[step] \times Z[step-1] \\ C &\triangleq C_1\,;\, C_2, \\ C_1 &\triangleq step = N \to Y := temp + X \times h[0], \\ C_2 &\triangleq step := 0 \\ X_1 &\triangleq step = 0 \to temp := 0_T\,;\, step := step + 1, \\ X_2 &\triangleq step \in [1, .., N-1] \to temp := temp + h[step] \times Z[step-1]\,; \\ &\qquad\qquad step := step + 1, \end{aligned}$$

We first show that

$$I \;\triangleq\; step \in [2, .., N] \Rightarrow temp = \sum_{k=1}^{step-1} h[k] \times Z[k-1]$$

is an invariant of $\mathcal{F}_S$. Observe that after the execution of either of the actions X_1 or C, $step \in [2..N] \equiv false$, therefore $I \equiv true$, trivially. We analyze in more detail the situation that concerns action X_2. We have to show that $I \Rightarrow wp(X_2, I)$.

$$\begin{array}{rl} & wp(X_2, I) \\ \equiv & \{wp \text{ rules for guarded action, sequential comp., assignment}\} \end{array}$$

$step \in [1,..,N] \Rightarrow I(step_{new}/step, temp_{new}/temp)$

$\equiv$ {substitutions}

$step \in [1,..,N] \Rightarrow$

$$(step \in [2,..,N] \Rightarrow temp + h[step] \times Z[step-1] = \sum_{k=1}^{step} h[k] \times Z[k-1])$$

$\equiv$ { arithmetic }

$$step \in [1,..,N] \Rightarrow (step \in [2,..,N] \Rightarrow temp = \sum_{k=1}^{step-1} h[k] \times Z[k-1])$$

$\equiv$ { identification }

$step \in [1,..,N] \Rightarrow I$

$\Leftarrow$

I

Even if not necessary in this context, we also show that I is a proper invariant. For this, we only check what happens when the global action (C) becomes activated. The guard of this action is $g = (step = N)$. We have further that $wp(C_1\ ; C_2, I) = wp(C_1, I[0/step]) = g \Rightarrow true$. Hence, trivially, $g \Rightarrow (I[wS'/wS, z] \equiv I[ws'/wS])$. Thus, I is a proper invariant of $\mathcal{F}_S$.

Having decided that I is an invariant of $\mathcal{F}_S$, in order to prove the above specified refinement, we go through the requirements of Lemma 1:

1. Initialization: $I(step = 0, temp = 0_T, h = h_0, Z = z_0) \equiv true$
2. Main action. We have to prove that $A \leq_I C$. For this, we have to show that: $I \wedge wp(A, Q) \Rightarrow wp(C, I \wedge Q), \forall Q$. We identify:

Derivation 1:

$I \wedge wp(A, Q)$

$\equiv$ {wp rule for assignment}

$$(step \in [2,..,N] \Rightarrow temp = \sum_{k=1}^{step-1} h[k] \times Z[k-1])$$

$$\wedge\ Q[X \times h[0] + \sum_{k=1}^{N-1} h[k] \times Z[k-1]/Y]$$

$\equiv$ { notation }

$$(step \in [2,..,N] \Rightarrow temp = \sum_{k=1}^{step-1} h[k] \times Z[k-1]) \wedge\ Q_A$$

$\equiv$ { logic }

$$(step \in [2, .., N] \vee \neg Q_A) \Rightarrow (temp = \sum_{k=1}^{step-1} h[k] \times Z[k-1]) \wedge \ Q_A$$

$$\Rightarrow \{step = N \Rightarrow step \in [2, .., N]\}$$

$$(step = N \vee \neg Q_A) \Rightarrow (temp = \sum_{k=1}^{step-1} h[k] \times Z[k-1]) \wedge \ Q_A$$

and

Derivation 2:

$$wp(C, I \wedge Q)$$

$$\equiv \{wp \text{ rule for sequential composition}\}$$

$$wp(C_1, wp(C_2, I \wedge Q))$$

$$\equiv \{wp \text{ rule for assignment}\}$$

$$wp(C_1, I[0/step] \wedge Q[0/step])$$

$$\equiv \{Q \text{ does not mention } step,\ I[step = 0] \equiv true\}$$

$$wp(C_1, Q)$$

$$\equiv \{wp \text{ rules for guarded action, assignment}\}$$

$$step = N \Rightarrow Q[temp + X \times h[0]/Y]$$

$$\Leftarrow \{\text{logic}\}$$

$$(step = N \Rightarrow Q[temp + X \times h[0]/Y])$$

$$\wedge (step = N \Rightarrow temp = \sum_{k=1}^{N-1} h[k] \times Z[k-1])$$

$$\Leftarrow \{\text{logic}\}$$

$$step = N \Rightarrow Q[\sum_{k=1}^{N-1} h[k] \times Z[k-1] + X \times h[0]/Y]$$

$$\equiv \{ \text{ notation } \}$$

$$step = N \Rightarrow Q_A$$

From the above, we have that:

$$I \wedge wp(A, Q)$$

$$\Rightarrow \{\text{Derivation 1}\}$$

$$(step = N \vee \neg Q_A) \Rightarrow (temp = \sum_{k=1}^{step-1} h[k] \times Z[k-1]) \wedge \ Q_A$$

$$\Rightarrow \{ \text{ logic } \}$$

$$\begin{aligned} & step = N \Rightarrow Q_A \\ \Rightarrow\ & \{\text{Derivation 2}\} \\ & wp(C, I \wedge Q) \end{aligned}$$

3. Auxiliary action. For the auxiliary actions X_1, X_2, we have that $wX_1, wX_2 \in \{step, temp\}$, therefore they behave like skip with respect to the global variables. Hence, skip $\leq_I X_1 \wedge$ skip $\leq_I X_2$.

4. Continuation condition:

$$\begin{aligned} & I \wedge gA \Rightarrow gC \vee gX \\ \equiv\ & \{gX_1 \vee gX_2 \vee gC \equiv true\} \\ & I \wedge gA \Rightarrow true \\ \equiv\ & \\ & true \end{aligned}$$

5. Internal convergence. It is easy to observe that X_1 terminates after one execution as it disables itself, while X_2 disables itself after $N - 1$ executions.

From the above we have that, in *isolation*, the system $\mathcal{F}_S$ is a refinement of $\mathcal{F}$, under the invariant I: $\mathcal{F} \sqsubseteq_I \mathcal{F}_S$.

References

[Back and von Wright 1999] R. J. R. Back, J. von Wright. Reasoning algebraically about loops. Acta Informatica 36, pp. 295-334, Springer–Verlag, 1999.