Towards a Proof Method for Paradigm

L.P.J. Groenewegen[1], R. Kuiper[2], and E.P. de Vink[2,3,⋆]

[1] Leiden Institute of Advanced Computer Science
Leiden University, The Netherlands
[2] Department of Mathematics and Computer Science
Eindhoven University of Technology, The Netherlands
[3] Centrum Wiskunde & Informatica, Amsterdam, The Netherlands

Abstract. The paper describes two perspectives on a verification approach for Paradigm, a coordination modeling language specifying an architecture in terms of components and their collaborations. One perspective concentrates on a single collaboration: per collaboration, properties can be derived through a small set of proof rules. The other perspective concentrates on dynamic dependencies between collaborations: guided by the architecture and driven by shared components behavioral properties of the complete model can be established. Two Paradigm models, a parallel assignment and a linear pipeline of workers and buffers, illustrate the approach.

1 Introduction

Investigation of all kinds of parallel phenomena, such as communication, interaction, concurrency, collaboration and cooperation, arising inside as well as around ICT, has led to the development of the coordination language Paradigm. Most characteristic for Paradigm are its two notions of *phase* and *trap*: phase for a dynamic temporary constraint on currently possible process behavior; trap, within a phase, for a further dynamic constraint on the behavior for as long as the phase constraint remains imposed.

Phases and traps are the key ingredients to specify the coordination of components. Paradigm is organized such that consistency between group behavior of elaborating components described at their coordination level and separate component behavior decribed at the lower level is guaranteed. Based on phases and traps, mutual behaviour can be understood, specified and organized and their coordination can be designed and analyzed, see e.g. [3].

In the 90's Paradigm has been used for modeling software processes resulting in a combination of object-orientation and Paradigm, see e.g. [11, 7]. Later Paradigm was gradually tuned to model self-adaptation as well, see e.g. [6, 4]. The way we do this is by treating self-adaptation as a special form of normal on-the-fly coordination, be it originally unforeseen, which comes down to just-in-time foreseen coordination.

⋆ Corresponding author, email `evink@win.tue.nl`.

So far, formal analysis of Paradigm models has been done via model checking, in particular using `mCRL2` and `Prism`, see [3, 1, 2, 4]. But we have the strong impression, that it is possible to take advantage of Paradigm's modeling focus on collaboration of components rather more substantially. Here we put first steps into this direction and investigate a verification approach to Paradigm which on the one hand considers 'correctness formulas' capturing the interaction of a small group of components, and on the other hand distills behavioral properties of the total architecture from such formulas, thus exploiting the way the overall system is actually composed.

Compositional proof systems reduce the verification of a system to simpler, independent verification of its constituents. There are many choices of what constitutes a system: parallel components are an obvious one [12], but also concepts from object-orientation like class or subclass qualify [10]. We consider parallelism. As described in [13], a compositional proof system can be devised by including interference information in the specification of the components. The next step is then to split specifications in two parts, pertaining to the environment respectively to the component, conform rely-guarantee approaches as initiated in [9]. For Paradigm, the rules that govern a collaboration join the 'rely' and 'guarantee': a component engaged guarantees to restrict to the behavior that is committed to while relying on other components to behave accordingly.

In this paper we present a formalism to express properties of components and also of subsystems in a compositional manner. Moreover, the approach uses Rely/Guarantee ideas. In Paradigm a component consists of an STD, with its possible collaborations denoted at the level of and in terms of phases and traps: collaboration intentions. A system is constructed out of components and consistency rules. Consistency rules are specified in terms of the collaboration intentions denoting the allowed collaborations. Note, composition in Paradigm is not provided in a fixed manner, by combinators like sequential of parallel composition, but more flexibly: it is determined by the consistency rules. An execution trace then starts in a certain configuration and, by applying a sequence of consistency rules, it leads to another configuration. Our formalism exploits this by expressing properties as triples of a description of starting configurations, a sequence of consistency rules, and a description of end configurations.

We firstly consider properties of one component that may be involved in several collaborations simultaneously. For this case we provide compositional derivation rules to compose properties, enabling to combine properties for separate components as well as for shuffling sequences. Secondly, we extend the description to combinations of subsystems, each consisting of several components. The format to express properties remains essentially the same, but is used in a more general manner: both the configurations and the consistency rules may now pertain to different subsystems. We do not yet have proof rules to formally derive that such a combination of subsystems satisfies a specification, but we do propose a method to argue the satisfaction of a specification of a complete system by the combination of properties of the constituting component groups. The approach can, in the component case, be viewed as being of Rely/Guarantee na-

ture in the sense that the consistency rules both determine what the component will rely on and what it will provide, but also what the environment will rely on and what it will provide. In the case of the subsystems, a Rely/Guarantee relation between components (now from different subsystems) can additionally be expressed in the description of the configurations.

In order to highlight the mainstay underlying our approach, on the one hand a formalism and proof rules to express and derive properties of the collaboration within a subsystem, and on the other hand the combination of such properties to prove global system properties, the paper is structured as follows: Section 2 provides an introduction to Paradigm and discusses a small model for the parallel assignment example. Section 3 introduces triples $\varphi\,[\,\Omega\,]\,\psi$ to express collaboration properties and gives basic proof rules to fuse triples concerning a group of components. A larger Paradigm model, concerning a linear pipeline of workers and buffers, is described in Section 4. Finally, in Section 5, we prove a property for the complete pipeline based on the analysis of smaller collaborations.

Acknowledgment The authors, all present and former colleagues of Frank de Boer at TU/e, VU Amsterdam, Leiden University, and CWI, acknowledge inspiring discussions with Frank on semantics, proof theory, object-orientation, and many other topics in computer science and beyond.

2 A Paradigm primer

The coordination modeling language Paradigm addresses coordination of interacting components. At the component level, detailed state transition diagrams specify the local, independent behavior of each individual component. At the coordination level, global roles specify the potential activity of a single component within a collaboration, while consistency rules describe the synchronization of the actual interaction of a group of collaborating components as a whole. In more detail, component interaction in Paradigm is specified through eight coherent definitions, see also [4]. Interaction is between sequentially defined component behaviour specified through state transition diagrams.

1. An *STD* Z (*state-transition diagram*) is a triple $Z = \langle \mathtt{ST}, \mathtt{AC}, \mathtt{TR} \rangle$ with $\mathtt{ST}$ the set of *states*, $\mathtt{AC}$ the set of *actions* and $\mathtt{TR} \subseteq \mathtt{ST} \times \mathtt{AC} \times \mathtt{ST}$ the set of *transitions* of Z; notation $x \xrightarrow{a} x'$ is used for a transition $(x, a, x') \in \mathtt{TR}$.

To keep track of an STD's actual running, we define

2. A *computation* κ of an STD $Z = \langle \mathtt{ST}, \mathtt{AC}, \mathtt{TR} \rangle$ is a finite or infinite string $\kappa = x_0 \xrightarrow{a_1} x_1 \xrightarrow{a_2} x_2 \xrightarrow{a_3} \ldots$ with $x_{i-1} \xrightarrow{a_i} x_i \in \mathtt{TR}$ for all indices $i \geqslant 1$ from κ. State x_0 is called the *starting state* of computation κ. In case of a finite computation κ, the string ends with a state: $x_0 \xrightarrow{a_1} x_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} x_n$ and such a state x_n is called the *final state* of κ.

Although time will remain implicit, it is assumed any transition is zero-time consuming. In addition we call a state x_i occurring in a computation κ the

current state at the moment transition $x_{i-1} \xrightarrow{a_i} x_i$ occurs as well as at all later moments until (but not including) the next transition $x_i \xrightarrow{a_{i+1}} x_{i+1}$ occurs.

Built on the notion of STDs, Paradigm has four key notions: (i) *phase*, (ii) *trap*, (iii) *role*, and (iv) *consistency rule*. See items 3, 4, 6, and 7 below.

During collaboration a component gets influenced through a temporary constraint imposed on it from "elsewhere": a *phase*, which restricts a component to a sub-STD of the ongoing STD for a while. Similarly, during collaboration an ongoing STD contributes information towards the same "elsewhere" about progress within the phase: a *trap*, being a non-empty subset of the states of a phase which, as a subset, cannot be left as long as the phase remains imposed. As will get clear below, the "elsewhere" is the protocol regulating phase transfers on the basis of traps entered. Formally we have

3. A *phase* S of an STD $Z = \langle \mathtt{ST}, \mathtt{AC}, \mathtt{TR} \rangle$ is an STD $S = \langle \mathtt{st}, \mathtt{ac}, \mathtt{tr} \rangle$ such that $\mathtt{st} \subseteq \mathtt{ST}$, $\mathtt{ac} \subseteq \mathtt{AC}$ and $\mathtt{tr} \subseteq \{\, (x, a, x') \in \mathtt{TR} \mid x, x' \in \mathtt{st},\ a \in \mathtt{ac} \,\}$.
4. A *trap* t of a phase $S = \langle \mathtt{st}, \mathtt{ac}, \mathtt{tr} \rangle$ of STD Z is a non-empty set of states $t \subseteq \mathtt{st}$ such that $x \in t$ and $x \xrightarrow{a} x' \in \mathtt{tr}$ imply $x' \in t$. If $t = \mathtt{st}$, the trap is called *trivial*. A trap t of phase S of STD Z *connects* phase S to a phase $S' = \langle \mathtt{st}', \mathtt{ac}', \mathtt{tr}' \rangle$ of Z if $t \subseteq \mathtt{st}'$. Such trap-based connectivity between two phases of Z is called a *phase transfer* and is denoted as $S \xrightarrow{t} S'$.

Thus a trap (of a phase) entered can be seen as a further constraint, where the STD commits to stay within that phase, allowing 'synchronization' at the coordination level to occur safely. Here 'committing' actually results from the behaviour-restricting effect of that phase: within the context of the phase, entering this trap of it means that a certain amount of progress has been established within the phase, a kind of final stage within the phase. Quite specifically, as long as the phase continues to be imposed, that progress cannot be undone since within the phase it is a trap of, the trap cannot be left.

Figure 1 visualizes three STDs, to be discussed in more detail later. Please note, starting and final states are indicated in UML 2.0 style: parts (a,b) have their starting states at the top and they have their final states at the bottom; part (c) has "$x = 0$" as its starting state and doesn't have a final state indicated.

Figure 2 parts (a,c,e) visualize three sets (so-called partitions) of phases and traps, one such set per STD from Figure 1. One should observe, each phase from part (a) is a (carefully) shrunken fragment from STD P_1; per phase, one trap is indicated by a red rectangle (a red polygon in general); part (c) has phase PhX_0 containing many disjoint traps; it also has phase PhX_1 with one trap ξ_1 containing all states, so trap ξ_1 is trivial. In this figure every smallest possible trap, containing one state, corresponds to maximal progress; in general, a trivial trap corresponds to no particular progress yet, but within concrete phase PhX_1 this doesn't mean anything special, as there cannot be any progress at all. Contrarily (and this is common for most Paradigm models), Figure 4 to be discussed in Section 4 presents examples of nontrivial traps which are relatively large and expressing progress in between "no progress yet" and "maximal progress": traps *doneT*, *doneG*, *notFull*, *nonEmpty*, *readyC* and *readyP*. It is the intuitive idea of

measuring "sufficient progress" within a phase through keeping track of entering a particular trap, that paves the way to connecting traps, to phase transfers and from there to role dynamics as global STDs.

If a trap of a phase is moreover connecting to another phase, meaning that all states of the trap are states of the other phase too (but not necessarily a trap of it), the other phase is a possible candidate for being imposed next. This applies only after sufficient progress within the previous phase has occurred, i.e. after that connecting trap has indeed been entered. The trap being connecting, such a phase transfer from the old phase to the new phase will be sufficiently 'smooth': The restrictions of the old phase are being withdrawn, the restrictions of the new phase are being imposed, without an immediate need to change the current state. Thus, when changing the old phase to the new, the current state doesn't have to change before a transition (in accordance with the new phase imposed) can occur: Whatever that current state may be, it must as yet belong to the particular connecting trap used for the phase transfer. So from then on, steps are being taken according to the new phase imposed. In particular this means that any two subsequent phases of the same STD, must have a non-empty set of states in common, because a connecting trap is non-empty by definition.

A *role*, yet another STD, specifies the dynamics of phases and of connecting traps which can be used for a phase transfer. Thus, a role STD of an ordinary STD has phases of the ordinary STD as states and has connecting traps between these phases as transition labels. A *partition* is the set of phases and their traps underlying a role.

5. A *partition* $\pi = \{ (S_i, T_i) \mid i \in I \}$ of an STD $Z = \langle \mathtt{ST}, \mathtt{AC}, \mathtt{TR} \rangle$, I a non-empty index set, is a set of pairs (S_i, T_i) consisting of a phase S_i of Z and of a set T_i of traps of S_i.
6. A *role* $Z(\pi)$ at the level of a partition $\pi = \{ (S_i, T_i) \mid i \in I \}$ of an STD $Z = \langle \mathtt{ST}, \mathtt{AC}, \mathtt{TR} \rangle$ is an STD $Z(\pi) = \langle \widehat{\mathtt{ST}}, \widehat{\mathtt{AC}}, \widehat{\mathtt{TR}} \rangle$ with $\widehat{\mathtt{ST}} \subseteq \{ S_i \mid i \in I \}$, $\widehat{\mathtt{AC}} \subseteq \bigcup_{i \in I} T_i$ and $\widehat{\mathtt{TR}} \subseteq \{ S_i \xrightarrow{t} S_j \mid i, j \in I, t \in \widehat{\mathtt{AC}} \}$ a set of phase transfers. Z is called the *detailed* STD underlying *global* STD $Z(\pi)$, being role $Z(\pi)$.

Loosely speaking, coordination of component activity takes place at the level of the global or role STDs, while component computation takes place in the component's detailed STD itself. Such coordination exerts a combined and varying constraining effect on ongoing component computations, but always on the basis of relevant progress information provided by these components. A local computation step is allowed only if permitted by all current phases imposed; a 'coordination' step is enabled only if all collaborators have entered the relevant trap. By requiring traps to be connecting between phases, Paradigm syntactically guarantees vertical dynamic consistency between an ongoing STD and any of its likewise ongoing roles, see [3].

With respect to the three roles in parts (b,d,f) of Figure 2, it is relevant to note that traps labeling a phase transfer are indeed connecting "from the previous phase to the next phase", i.e. from the old constraint imposed to the new constraint imposed via the connecting trap, thus guaranteeing smoothness of phase transfer in case of specific progress within the old phase.

Finally, a *consistency rule* synchronizes role steps from different roles in the collaboration. Such a rule can be seen as a *protocol step*, a coordination step belonging to the collaboration. All such protocol steps together then constitute the full protocol for the collaboration, and the dynamics of such a protocol consist of its subsequent protocol steps taken, i.e. of consistency rules being applied. More precisely: a group of consistency rules constitutes a *protocol* if all roles mentioned in the consistency rules from that group are not mentioned in consistency rules not from that group. In this way we can structurally and behaviorally separate collaborations from each other, as each collaboration has its own protocol, i.e. refers to a specific set of roles and of consistency rules.

7. A *consistency rule* ϱ for an ensemble of roles $Z_1(\pi_1), \ldots, Z_k(\pi_k)$ is a synchronization of one or more phase transfers from roles in the ensemble. A consistency rule ϱ is denoted as a nonempty '$\|$'-separated list of phase transfers taken from different roles from the ensemble.
8. A Paradigm *model* is an ensemble of STDs, roles thereof and consistency rules for these. A *protocol* P of a Paradigm model M is a subset of the set R of consistency rules belonging to M such that for any role $Z_i(\pi_i)$ occurring in a rule $\varrho \in P$ role $Z_i(\pi_i)$ does not occur in whatever consistency rule $\varrho' \in (R \backslash P)$. Any consistency rule ϱ belonging to a protocol P is called a *protocol step* of P.

Within the scope of this paper, we omit other types of consistency rules than described above. In [3] we distinghuish between consistency rules modelling orchestration and choreography.

Somewhat below Figure 2, two groups of consistency rules are given: $CR_1^i(\mathrm{j})$, $CR_2^i(\mathrm{j,k})$, $CR_3^i(\mathrm{k})$ for $i = 1$ and $CR_1^i(\mathrm{j})$, $CR_2^i(\mathrm{j,k})$, $CR_3^i(\mathrm{k})$ for $i = 2$, actually together constituting one protocol as they all have role $X(FS)$ in common. Even without going into the specific details of phases and traps mentioned, one can understand such rules as follows; e.g. take rule $CR_1^1(j)$ for a fixed j

$$P_1(Asg)\colon Ph_0^1 \xrightarrow{\theta_0} Ph_1^1(j) \parallel X(FS)\colon PhX_0 \xrightarrow{\xi_0(j)} PhX_1$$

After sufficient progress both within (1) phase Ph_0^1 of role Asg of STD P_1, which means trap θ_0 has been entered, and within (2) phase PhX_0 of role FS of STD X, which means trap $\xi_0(j)$ has been entered, the two phase transfers

$$Ph_0^1 \xrightarrow{\theta_0} Ph_1^1(j) \quad \text{and} \quad PhX_0 \xrightarrow{\xi_0(j)} PhX_1$$

occur simultaneously. This immediately leads to new behavioral freedom for the two detailed STDs mentioned: As no other constraining effects from other roles can be taken into account (there are none), P_1 now is to proceed in accordance with phase $Ph_1^1(j)$, and X now is to proceed in accordance with phase PhX_1.

Next, we illustrate Paradigm for a relatively small and simple problem situation: the parallel program $(x := x+1) \parallel (x := x+2)$, consisting of two parallel assignments concerning a shared variable x. For the presentation below we distinguish

sequential components P_1 and P_2, where $P_1 \triangleq x := x + 1$ and $P_2 \triangleq x := x+2$. In view of modeling program $P_1 \parallel P_2$ as a Paradigm model we describe P_1 and P_2 by a separate STD. Figure 1ab visualizes these.

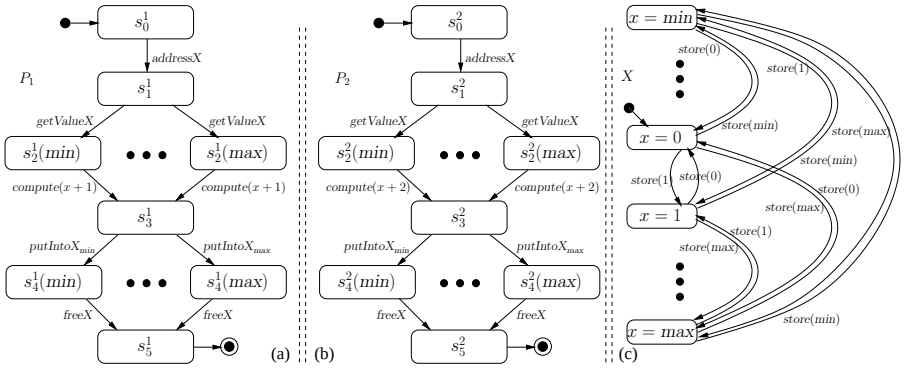

Fig. 1. STDs of (a) P_1, (b) P_2 and (c) X.

Because Paradigm does not address shared data and their usage in its models, we cannot so easily incorporate variable x, being shared by P_1 and P_2, into a Paradigm model. But in this case we must, as using variable x by one sequential program, P_1 say, might influence the usage of variable x by the other. In view thereof we model variable x, and its particular usage by P_1 and P_2, by means of a separate STD, referred to as X in Figure 1c.

As one can see from Figure 1ab, for one assignment both P_i execute a sequence of five steps: (1) addressing STD X by asking for its current value (fetch request), (2) getting X's current value through fetching (any value between $min, \ldots, max$ is possible), (3) computing a certain expression depending on that value (for simplicity we assume a single-valued outcome, without errors occurring), (4) informing STD X about the computation result as the new value to be stored (store request), (5) releasing STD X.

Contrarily, STD X lacks such step sequencing, as it is a complete graph: each state of it is reachable from any other state in one step. A state of X reflects the current value of variable x modeled by X. In each of its states $x = p$ where $p = min, \ldots, max$ it can either answer a fetch request or a store request. In case of a fetch it just stays in state $x = p$. In case of a store it goes directly to another state $x = r$ where $r = min, \ldots, max$ by taking the transition from $x = p$ to $x = r$ labelled by action $store(r)$, or it just stays in state $x = p$ in case r equals p. As specified, X starts from having value 0. At its detailed level X executes no steps at all (6) whenever it is to send its current value: instead, fetching and sending a value is done by a role step (one out of many from PhX_0 to PhX_1 in Figure 2f). In addition, X executes one step (7) whenever it receives a value to be stored: storing that value by overwriting the previous value through changing its current state – but only if needed.

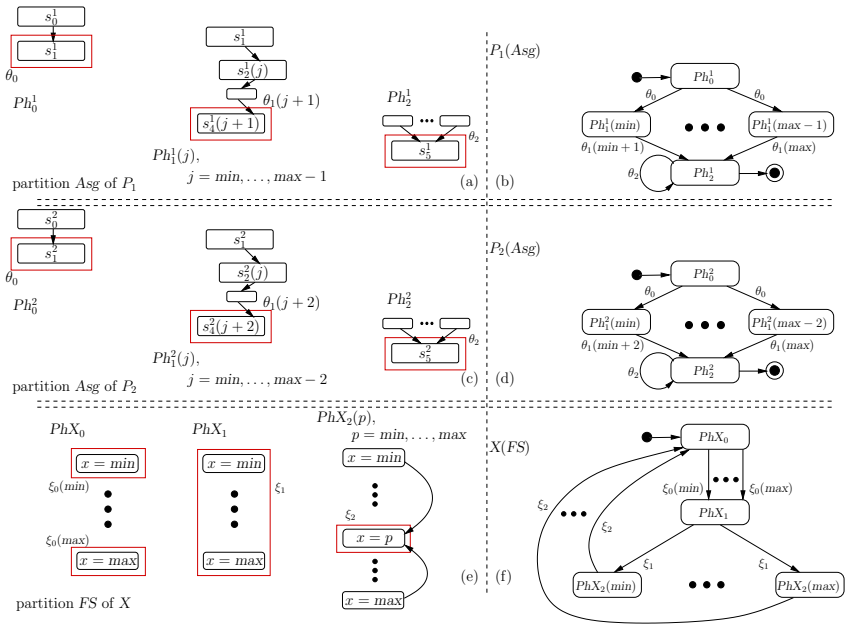

Fig. 2. *Asg* partition and role of P_1 (a,b) and P_2 (c,d), *FS* partition and role of X (e,f).

Given the result of computing x+1 by P_1 in state s_3^1 (right after its third step labelled *compute*(x+1)), P_1 should select action *putInto*X_{j+1} (whose index j+1 corresponds to the value computed) dependent on which of its second steps *getValue*X it has taken. This in turn fully depends on the current state of X at the moment step *getValue*X has been selected. (For P_2 similar observations can be made.) As can be seen below, the latter dependence is assured by consistency rule $CR_1^1(j)$, whereas the former dependence follows from phase $Ph_1^1(j)$ and its trap $\theta_1(j+1)$ as defined in Figure 2a.

The coordination we want to model, i.e. the mutual interaction between the three STDs, assumes an atomic fetch-store cycle for variable x. Thus we model the details concerning both fetching and storing via one role of each STD of P_1 and P_2 for both requests as well as via one role of STD X for handling such requests, always in pairs. Roles are called $P_i(Asg)$, for $i = 1, 2$, and $X(FS)$, respectively, and visualized in the right part of Figure 2. The partitions of the phases and traps comprising the roles *Asg* and *FS* are displayed on the left.

For P_i, index j of phase $Ph_1^i(j)$ is meant to reflect the state of X at the time of fetching. By our assumption errors do not occur, phase $Ph_1^i(max)$ of P_1 and the two phases $Ph_1^i(max{-}1), Ph_1^i(max)$ of P_2 are not present. During any phase $Ph_1^i(j)$ the actual computation is being carried out. Within such a phase $Ph_1^i(j)$ of P_i the value resulting from computing $j + i$, for $i = 1, 2$, is indicated via the parameter of trap $\theta_1(j{+}i)$, which will be entered certainly.

The consistency rules based on these roles, are as follows:

$$P_i(Asg)\colon Ph_0^i \xrightarrow{\theta_0} Ph_1^i(j) \;\|\; X(FS)\colon PhX_0 \xrightarrow{\xi_0(j)} PhX_1 \qquad (CR_1^i(\mathrm{j}))$$

$$P_i(Asg)\colon Ph_1^i(j) \xrightarrow{\theta_1(k)} Ph_2^i \;\|\; X(FS)\colon PhX_1 \xrightarrow{\xi_1} PhX_2(k) \qquad (CR_2^i(\mathrm{j,k}))$$

$$P_i(Asg)\colon Ph_2^i \xrightarrow{\theta_2} Ph_2^i \;\|\; X(FS)\colon PhX_2(k) \xrightarrow{\xi_2} PhX_0 \qquad (CR_3^i(\mathrm{k}))$$

where rule $CR_1^i(j)$ is for fetching value j, rule $CR_2^i(j,k)$ for initiating storing value k and rule $CR_3^i(k)$ for finishing storing value k. This completes the specification of the Paradigm model of the parallel assignment example. We next turn to its behavior.

Via consistency rule $CR_1^i(\mathrm{j})$, imposing phase $Ph_1^i(j)$ on P_i (and thereby later entering trap $\theta_1(j{+}i)$, as we have seen), is coupled to the old value of x, thus starting a new role cycle of $X(FS)$. Via rule $CR_2^i(\mathrm{j,k})$, this is coupled to the new value of x. Rule $CR_3^i(\mathrm{k})$ then couples termination of P_i to X's finishing its full cycle. Only after having finished such a cycle, X can get involved in a new cycle, which assures the atomicity of the fetch-store cycle. Hence a full fetch-store cycle of X at the level of its FS role is

$$PhX_0 \xrightarrow{\xi_0(j)} PhX_1 \xrightarrow{\xi_1} PhX_2(k) \xrightarrow{\xi_2} PhX_0$$

Examining the consistency rules we see that in this particular example they will be applied in a sequential order. Starting from a configuration with process X in phase PhX_0 only the rules CR_1^i applies, for $i = 1, 2$. After execution of rule CR_1^i, based on the phases, rule CR_2^i is the only one that can be applied next. Only after application of rule CR_2^i, rule CR_3^i becomes enabled.

In the above, we assumed appropriate settings of the rule parameters j and k. If we consider them more closely, we see that if $CR_2^i(j', k')$ is executed after $CR_1^i(j)$, we must have $j' = j$. For $CR_1^i(j)$ leaves process P_i in phase $Ph_1^i(j)$, while $CR_2^i(j', k')$ assumes P_i to be in phase $Ph_1^i(j')$. Application of rule $CR_2^i(j', k')$ requires $\theta_1(k')$ to be a trap of phase $Ph_1^i(j')$. From Figure 2 we see that $Ph_1^i(j')$ only has one trap, viz. trap $\theta_i(j' + i)$, for $i = 1, 2$. Thus, we must have $k' = j' + i$. Finally, similar as for parameters j and j' in $CR_1^i(j)$ and $CR_2^i(j', k')$, we have that application of rule $CR_3^i(k'')$ following an application of rule $CR_2^i(j', k')$ is only possible when $k'' = k'$: regarding the process X, the target phase $Ph_2^i(k')$ of $CR_2^i(j', k')$ must be the same as the source phase $Ph_2^i(k'')$ of $CR_3^i(k'')$.

Taking the range of the parameters into account, we conclude that in the respective collaborations, i.e. between P_1 and X, and between P_2 and X, the only complete sequences of rule actions are of the form $CR_1^1(j)CR_2^1(j, j{+}1)CR_3^1(j{+}1)$, for $min \leqslant j \leqslant max{-}1$, and $CR_1^2(k)CR_2^2(k, k{+}2)CR_3^2(k{+}2)$, for $min \leqslant k \leqslant max{-}2$. For brevity we will use $CR_*^1(j)$ and $CR_*^2(k)$ for these sequences below.

So far we have relied on ad hoc reasoning in the context of the semantics of the Paradigm model for the two parallel assignments. In the next section we describe how sequences of rule actions are verified more systematically.

3 Proving collaboration properties

Let us refer by $\mathcal{PA}$ to the Paradigm model of the parallel assignments. We will explain the operational semantics of Paradigm for the model $\mathcal{PA}$. As usual, the operational semantics is based on configurations and transitions between them [5, 8]. Configurations for $\mathcal{PA}$ are of the form $\langle s, Ph; s', Ph'; x{=}h, PhX\rangle$ were s, s' and $x{=}h$ are states of Ph, Ph' and PhX, which are in turn phases of P_1, P_2 and X, respectively. A transition for $\mathcal{PA}$ can be

(i) a *local transition* based on a transition of one of the STDs, or
(i) a *global transfer* based on one of the consistency rules.

A local transition only changes state for one of the components, i.e. P_1, P_2 or X. Reflecting the constraints put by the phases, a transition is allowed if present in all current phase(s) of the component. For example, because $s_0^1 \to s_1^1$ for action *addressX* is in Ph_0^1 we have

$$\langle s_0^1, Ph_0^1; s_0^2, Ph_0^2; x{=}0, PhX_0\rangle \longrightarrow \langle s_1^1, Ph_0^1; s_0^2, Ph_0^2; x{=}0, PhX_0\rangle$$

but, since $s_1^1 \to s_2^1(0)$ is not in Ph_0^1,

$$\langle s_1^1, Ph_0^1; s_0^2, Ph_0^2; x{=}0, PhX_0\rangle \not\longrightarrow \langle s_2^1(0), Ph_0^1; s_0^2, Ph_0^2; x{=}0, PhX_0\rangle$$

Note, for a local transition, the state may change but not the phases.

A global transfer effects the phases of one or more components of a configuration. The transfer is determined by the selected consistency rule. The consistency rule can be applied if the components involved are in the phase as required by the rule and have moreover reached the specified traps within the phases. If multiple consistency rules are enabled, one is selected non-deterministically. It is a design obligation to see to it that the collaboration and coordination proceeds as desired, in particular that deadlock is avoided. As an example of a global transfer, the transfer

$$\langle s_1^1, Ph_0^1; s_5^2, Ph_2^2; x{=}2, PhX_0\rangle \xrightarrow{CR_1^1(2)} \langle s_1^1, Ph_1^1(2); s_5^2, Ph_2^2; x{=}2, PhX_0\rangle$$

is driven by consistency rule $CR_1^1(2)$, which requires P_1 to be in phase Ph_0^1 and in trap $\theta_0 = \{s_1^1\}$, and requires X to be in phase PhX_0 in trap $\xi_0(2) = \{x{=}2\}$. The rule $CR_1^1(2)$ labels the transfer and is in such a situation referred to as a rule action. For the Paradigm model $\mathcal{PA}$, the sets of tags or rule actions in $\{\, CR_1^1(j), CR_2^1(j,k), CR_3^1(k) \mid min \leqslant j \leqslant max{-}1 \,\}$ and $\{\, CR_1^2(j), CR_2^2(j,k), CR_3^2(k) \mid min \leqslant j \leqslant max{-}2 \,\}$ are called the collaboration alphabets Σ_1 and Σ_2 of $\mathcal{PA}$, respectively. We put $\Sigma = \Sigma_1 \cup \Sigma_2$.

Let $\mathcal{L}$ be a logic in which one can express properties of configurations. In particular, $\mathcal{L}$ includes characteristic formulas for sets of configurations. For example, in the context of the previous section, the formula $\varphi_{\langle x=0,\, PhX_0\rangle}$ is satisfied precisely by all configurations $\langle s, Ph; s', Ph'; x{=}0, PhX_0\rangle$ with s a state of Ph, Ph a phase of P_1, s' a state of Ph', Ph' a phase of P_2. We write $\gamma \models \varphi$, for a configuration γ and a formula φ, if γ satisfies φ. Thus $\langle s_0^1, Ph_0^1; s_0^2, Ph_0^2; x{=}0, PhX_0\rangle \models \varphi_{\langle x=0,\, PhX_0\rangle}$. For clarity we often write $\langle x{=}0, PhX_0\rangle$ in place of $\varphi_{\langle x=0,\, PhX_0\rangle}$.

Definition 1. *A triple* $\varphi\,[\,\Omega\,]\,\psi$ *with* $\varphi, \psi \in \mathcal{L}$, $\Omega \subseteq \Sigma^*$ *is valid, if for every configuration* γ *of a Paradigm model such that* $\gamma \models \varphi$, *and every computation* $\gamma \xrightarrow{w}{}^* \gamma'$ *such that* $w \in \Omega$ *it holds that* $\gamma' \models \psi$.

The set Ω is a set of sequences of rule actions, reflecting the behaviour at the coordination level. We provide non-trivial examples of valid triples below. However, in general, we have

- $\varphi\,[\,\Omega\,]\,\mathbf{true}$ always holds
- $\varphi\,[\,\Omega\,]\,\mathbf{false}$ holds if for no configuration γ such that $\gamma \models \varphi$ there is a computation $\gamma \xrightarrow{w}{}^* \gamma'$ with $w \in \Omega$

Typically, we obtain a triple of the form $\langle s, Ph_1, \ldots, Ph_n\rangle\,[\,\sigma\,]\,\psi$ for a component having n roles and a sequence σ from one or two collaboration alphabets, by inspection of the changes of the component when executing the transfers based on the rules mentioned in σ, possibly interspersed with local transitions. Recall, a local transition between two configurations does not carry a label, while a global transfer between to configurations is labelled by a rule action.

By application of the compositional rules discussed next, we obtain more complicated triples.

conjunctive rule If $\varphi_1\,[\,\Omega\,]\,\psi_1$ and $\varphi_2\,[\,\Omega\,]\,\psi_2$, then also $(\varphi_1 \wedge \varphi_2)\,[\,\Omega\,]\,(\psi_1 \wedge \psi_2)$.

The soundness of the conjunctive rule is direct from the definition. If $\gamma \models \varphi_1 \wedge \varphi_2$ and $\gamma \xrightarrow{w}{}^* \gamma'$ for a string of rule actions $w \in \Omega$, then $\gamma' \models \psi_1$ because $\gamma \models \varphi_1$, and $\gamma' \models \psi_2$ because $\gamma \models \varphi_2$. The rule is typically used to combine triples for separate components. We shall see below that the triple $\langle s_0^1, Ph_0^1\rangle\,[\,CR_1^1(0)CR_2^1(0,1)CR_3^1(1)\,]\,\langle s_5^1, Ph_2^1\rangle$ for process P_1, and the triple $\langle x{=}0, PhX_0\rangle\,[\,CR_1^1(0)CR_2^1(0,1)CR_3^1(1)\,]\,\langle x{=}1, PhX_0\rangle$ for process X can be combined into the triple

$$\langle s_0^1, Ph_0^1; x{=}0, PhX_0\rangle\,[\,CR_1^1(0)CR_2^1(0,1)CR_3^1(1)\,]\,\langle s_5^1, Ph_2^1; x{=}1, PhX_0\rangle$$

for the two processes together.

sequential rule Let σ, ϱ be two arbitrary rule sequences and φ and ψ two formulas. If $\varphi\,[\,\sigma\,]\,\chi$ and $\chi\,[\,\varrho\,]\,\psi$ hold for some formula χ then $\varphi\,[\,\sigma\varrho\,]\,\psi$ also holds.

Regarding the soundness, if $\gamma \models \varphi$ and $\gamma \xrightarrow{\sigma\varrho}{}^* \gamma''$, we can split the computation into $\gamma \xrightarrow{\sigma}{}^* \gamma' \xrightarrow{\varrho}{}^* \gamma''$. From $\varphi\,[\,\sigma\,]\,\chi$ we obtain $\gamma' \models \chi$, and from $\chi\,[\,\varrho\,]\,\psi$ we obtain $\gamma'' \models \psi$. The reverse of the rule only applies if we are able to characterize in the logic the set of configurations that can be reached from any γ such that $\gamma \models \varphi$ by a computation labeled σ. This can then serve as the intermediate formula χ. In the elaborated example below we will have that the triple

$$\langle x{=}0, PhX_0\rangle\,[\,CR_1^1(0)CR_2^1(0,1)CR_3^1(1)\,]\,\langle x{=}1, PhX_0\rangle$$

and the triple

$$\langle x{=}1, PhX_0\rangle\,[\,CR_1^2(1)CR_2^2(1,3)CR_3^2(3)\,]\,\langle x{=}3, PhX_0\rangle$$

yield the triple

$$\langle x{=}0, PhX_0\rangle\,[\,CR_1^1(0)CR_2^1(0,1)CR_3^1(1)CR_1^2(1)CR_2^2(1,3)CR_3^2(3)\,]\,\langle x{=}3, PhX_0\rangle$$

by the sequential rule.

interleaving rule There exist formulas ψ_w for all $w \in \Omega$ such that $\varphi\,[\,w\,]\ \psi_w$ iff $\varphi\,[\,\Omega\,]\,\psi$ holds for $\psi = \bigvee\{\ \psi_w \mid w \in \Omega\ \}$.

The rule is referred to as the interleaving rule, since when applied the set of sequences of rule actions Ω is typically a set of the form $\sigma_1 \| \cdots \| \sigma_n$, a shuffle of sequences $\sigma_1, \ldots, \sigma_n$ stemming from different collaborations. The implication from left to right is direct from the definition. If $\gamma \models \varphi$ and $\gamma \xrightarrow{w}{}^{*} \gamma'$ for $w \in \Omega$ then $\gamma' \models \psi_w$, and, by the form of ψ, $\gamma' \models \psi$. For the implication from right to left, we can take $\psi_w = \psi$ for all w. In the example below we will encounter an enumeration of all the interleavings of two sequences σ and ϱ in a situation where we know that strict interleavings of σ and ϱ are not possible, thus $\psi_w = \mathbf{false}$, in that case. For the combinations $\sigma\varrho$ and $\varrho\sigma$ we will have $\psi_{\sigma\varrho} = \psi_{\varrho\sigma} = \psi$ in the example. Since **false** vanishes in the disjunction we obtain from the interleaving rule a triple for ψ.

As a further illustration of the use of the above proof rules we consider the example of the two parallel assignments of the previous section. We claim that the Paradigm model $\mathcal{PA}$ satisfies

$$\langle s_0^1, Ph_0^1; s_0^2, Ph_0^2; x{=}0, PhX_0\rangle\,[\,CR_*^1(j) \| CR_*^2(k)\,]\,\langle s_5^1, Ph_2^1; s_5^2, Ph_2^2; x{=}3, PhX_0\rangle$$

for suitable parameters j and k and with $CR_*^1(j) = CR_1^1(j)CR_2^1(j,j{+}1)CR_3^1(j{+}1)$ and $CR_*^2(k) = CR_1^2(k)CR_2^2(k,k{+}2)CR_3^2(k{+}2)$. Thus, any computation started in configuration $\langle s_0^1, Ph_0^1; s_0^2, Ph_0^2; x{=}0, PhX_0\rangle$ that executes the rules $CR_1^1(j)$, $CR_2^1(j,j{+}1)$, and $CR_3^1(j{+}1)$ consecutively in the collaboration of P_1 and X, and the rules $CR_1^2(k)$, $CR_2^2(k,k{+}2)$, and $CR_3^2(k{+}2)$, in that order, in the collaboration of P_2 and X, for suitable values of j and k, end in configuration $\langle s_5^1, Ph_2^1; s_5^2, Ph_2^2; x{=}3, PhX_0\rangle$.

Computations starting from configuration $\langle s_0^1, Ph_0^1; s_0^2, Ph_0^2; x{=}0, PhX_0\rangle$ do not exhibit arbitrary interleavings of sequences $CR_*^1(j)$ and $CR_*^2(k)$. Once CR_1^i has been executed by P_i, the other process P_{3-i} remains in its phase Ph_0^{3-i} and has to wait till X has returned in phase PhX_0 before any of the rules it is involved in can be executed. Therefore, there are no computations showing a sequence $w \in CR_*^1(j) \,\|\, CR_*^2(k)$ of a shape different from $CR_*^1(j)CR_*^2(k)$ or $CR_*^2(k)CR_*^1(j)$. Thus, it holds that

$$\langle s_0^1, Ph_0^1; s_0^2, Ph_0^1; x{=}0, PhX_0\rangle\,[\,w\,]\,\mathbf{false}$$

for $w \neq CR_*^1(j)CR_*^2(k), CR_*^2(k)CR_*^1(j)$.

Next, we determine the possible values for j and k. In configurations where $x{=}h$ and $j{\neq}h$ we have that X isn't in trap $\xi_0(j)$ and cannot reach it either. Also, no local transition is possible from state $x{=}h$ in phase PhX_0. Thus, it holds that $\langle x{=}h, PhX_0\rangle[\, CR_1^i(j)\,]$ **false**. However, if $j{=}h$, relying on cooperation from process P_i such that the sequence $CR_*^i(h)$ is executed, we obtain that x has been increased by i and has obtained value $h+i$. Therefore, on the one hand,

$$\langle x{=}h, PhX_0\rangle[\, CR_*^i(j)\,]\ \textbf{false} \tag{1}$$

for $j{\neq}h$, while on the other hand

$$\langle x{=}h, PhX_0\rangle[\, CR_*^i(h)\,]\langle x{=}h{+}i, PhX_0\rangle \tag{2}$$

From triple (1) we obtain $\langle x{=}0, PhX_0\rangle\,[\, CR_*^1(j)CR_*^2(k)\,]\,\textbf{false}$ for $j \neq 0$. Since by triple (2) $\langle x{=}0, PhX_0\rangle\,[\, CR_*^1(0)\,]\,\langle x{=}1, PhX_0\rangle$, and $\langle x{=}1, PhX_0\rangle\,[\, CR_*^2(k)\,]\,\textbf{false}$ by (1) for $k \neq 1$, we get $\langle x{=}0, PhX_0\rangle\,[\, CR_*^1(0)CR_*^2(k)\,]\,\textbf{false}$, for $k \neq 1$, by the sequential rule. This combines into $\langle x{=}0, PhX_0\rangle\,[\, CR_*^1(j)CR_*^2(k)\,]\,\textbf{false}$ for $j \neq 0$ or $k \neq 1$. Using $\langle s_0^1, Ph_0^1; s_0^2, Ph_0^2\rangle\,[\, CR_*^1(j)CR_*^2(k)\,]\,\textbf{true}$ and the conjunctive rule, we arrive at

$$\langle s_0^1, Ph_0^1; s_0^2, Ph_0^2; x{=}0, PhX_0\rangle\,[\, CR_*^1(j)CR_*^2(k)\,]\,\textbf{false} \tag{3}$$

for $j \neq 0$ or $k \neq 1$. Similarly, interchanging $CR_*^1(j)$ and $CR_*^2(k)$, we have

$$\langle s_0^1, Ph_0^1; s_0^2, Ph_0^2; x{=}0, PhX_0\rangle\,[\, CR_*^2(k)CR_*^1(j)\,]\,\textbf{false} \tag{4}$$

for $k \neq 0$ or $j \neq 2$.

We next aim to derive a triple for the sequence $CR_*^1(0)CR_*^2(1)$. Analysis of the behavior of P_1 and X with respect to the rules in $CR_*^1(0)$ yields, respectively, the triples $\langle s_0^1, Ph_0^1\rangle\,[\, CR_*^1(0)\,]\,\langle s_5^1, Ph_2^1\rangle$ and $\langle x{=}0, PhX_0\rangle\,[\, CR_*^1(0)\,]\,\langle x{=}1, PhX_0\rangle$. Thus, $\langle s_0^1, Ph_0^1; x{=}0, PhX_0\rangle\,[\, CR_*^1(0)\,]\,\langle s_5^1, Ph_2^1; x{=}1, PhX_0\rangle$ by the conjunctive rule. Since P_2 is not involved in any of the rules of $CR_*^1(0)$, it holds that $\langle s_0^2, Ph_0^1\rangle\,[\, CR_*^1(0)\,]\,\langle s_0^2, Ph_0^2\rangle$, which can be used with the above for the conjunctive rule to deduce

$$\langle s_0^1, Ph_0^1; s_0^2, Ph_0^1; x{=}0, PhX_0\rangle[\, CR_*^1(0)\,]\langle s_5^1, Ph_2^1; s_0^2, Ph_0^1; x{=}1, PhX_0\rangle \tag{5}$$

Likewise we can derive

$$\langle s_5^1, Ph_2^1; s_0^2, Ph_0^1; x{=}1, PhX_0\rangle[\, CR_*^2(1)\,]\langle s_5^1, Ph_2^1; s_5^2, Ph_2^1; x{=}3, PhX_0\rangle \tag{6}$$

Therefore, by the sequential rule, we obtain from (5) and (6) that

$$\langle s_0^1, Ph_0^1; s_0^2, Ph_0^1; x{=}0, PhX_0\rangle[\, CR_*^1(0)CR_*^2(1)\,]\langle s_5^1, Ph_2^1; s_5^2, Ph_2^1; x{=}3, PhX_0\rangle \tag{7}$$

By symmetry, we conclude the formula

$$\langle s_0^1, Ph_0^1; s_0^2, Ph_0^1; x{=}0, PhX_0\rangle[\, CR_*^2(0)CR_*^1(2)\,]\langle s_5^1, Ph_2^1; s_5^2, Ph_0^1; x{=}3, PhX_0\rangle \tag{8}$$

to hold as well, settling the case for the sequence $CR^2_*(0)CR^1_*(2)$.

Finally, we combine the cases gathered so far using the interleaving rule. Put

$$\begin{array}{ll} \psi_w = \mathbf{false} & \text{for } w \neq CR^1_*(0)CR^2_*(1), CR^2_*(0)CR^1_*(2) \\ \psi_w = \langle s^1_5, Ph^1_2; s^2_5, Ph^1_0; x{=}3, PhX_0 \rangle & \text{for } w = CR^1_*(0)CR^2_*(1), CR^2_*(0)CR^1_*(2) \end{array}$$

Then we have from (3), (4), (5), and (8) that

$$\langle s^1_0, Ph^1_0; s^2_0, Ph^1_0; x{=}0, PhX_0 \rangle \, [\, w \,] \, \psi_w$$

for all $w \in CR^1_*(j) \parallel CR^2_*(k)$ and suitable parameters j and k. Hence, by the interleaving rule

$$\langle s^1_0, Ph^1_0; s^2_0, Ph^1_0; x{=}0, PhX_0 \rangle \, [\, CR^1_*(j) \parallel CR^2_*(k) \,] \, \langle s^1_5, Ph^1_2; s^2_5, Ph^1_0; x{=}3, PhX_0 \rangle \quad (9)$$

since $\bigvee \{\, \psi_w \mid w \in CR^1_*(j) \parallel CR^2_*(k) \,\} = \langle s^1_5, Ph^1_2; s^2_5, Ph^1_0; x{=}3, PhX_0 \rangle$. The triple of Equation (9) captures the property that we aimed to prove.

4 A Paradigm model of a pipeline

As a larger Paradigm model we consider a linear pipeline with three workers Wrk_1, Wrk_2, Wrk_3, two buffers Buf_1, Buf_2, an input Buf_0, and an output Buf_3. They cooperate as follows, Wrk_i consuming one item from Buf_{i-1} (from input, if $i = 1$), working on it and producing it towards Buf_i (to output, if $i = 3$). All buffers have a fixed capacity, n say. When empty, no item can be taken from a buffer; when full, no item can be put into it. Contrarily, input and output are never restrictive. See Figures 3 and 6(a,c) for STDs of workers and buffers.

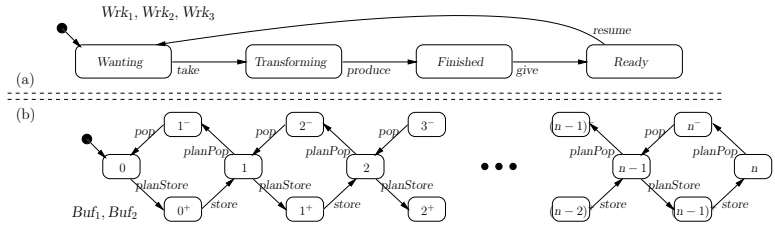

Fig. 3. (a) Three STDs Wrk_1, Wrk_2, Wrk_3, (b) two STDs Buf_1, Buf_2.

A worker visits four states sequentially, starting from state *Wanting*. A buffer precedes an actual *store* action or *pop* action by a preparatory step, *planStore* and *planPop*, respectively. Initially the buffers are empty.

The *Cons* and *Prod* partitions of the workers and the *Src* and *Snk* partitions of the buffers are drawn in Figure 4. Roles at the level of theses partitions are given in Figure 5; parts (a,b) group the roles from the protocol given by rules

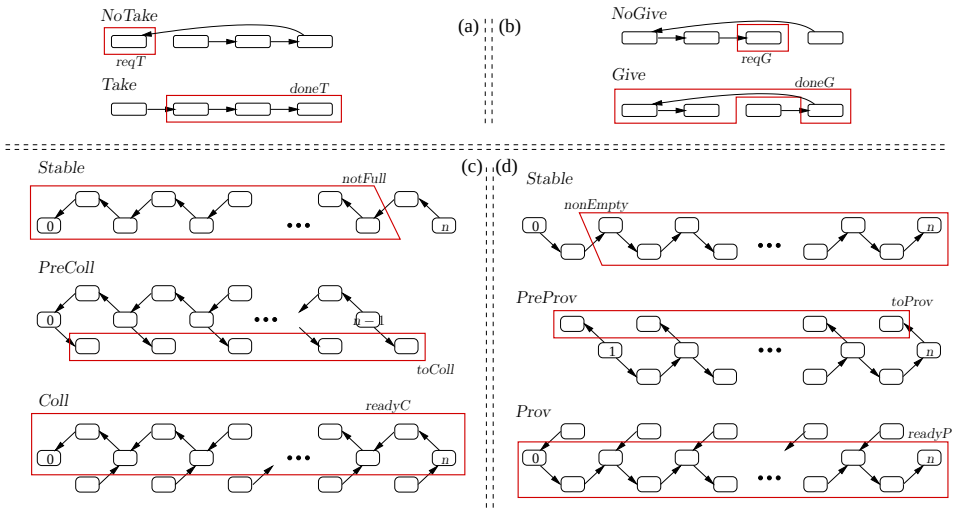

Fig. 4. Partitions (a) *Prod* (b) *Cons* for workers, (c) *Snk* and (d) *Src* for buffers.

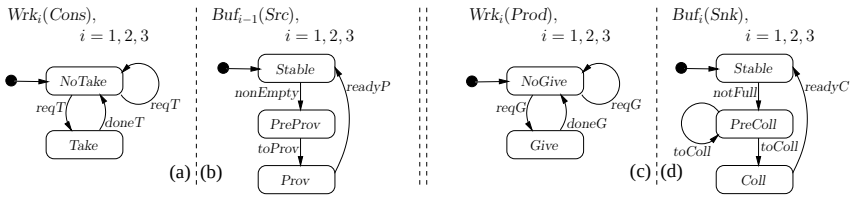

Fig. 5. Roles (a) *Cons*, (c) *Prod* for workers, (b) *Src* and (d) *Snk* for buffers.

CR_1^i–CR_4^i, fixed i, below; similarly, parts (c,d) group the roles from the protocol given by rules PR_1^i–PR_4^i, fixed i.

In its *Cons* role worker Wrk_i alternates between phase *Take* and phase *NoTake*. In phase *Take* the local action *take* is possible but no *resume* action. In phase *NoTake* this is the other way around. Reaching of the trap *reqT* triggers the collaboration with the providing buffer Buf_{i-1}, the interaction of $Wrk_i(Cons)$ and $Buf_{i-1}(Src)$ as captured by rules CR_1^i to CR_4^i. Clearly, Buf_{i-1} can only provide an item if non-empty, explaining the trap *nonEmpty* of phase *Stable*. Phase *PreProv* characterizes that the buffer is able to provide the item, but is actually doing so in phase *Prov*. Once trap *readyP* is reached in phase *Prov* the buffer can return –in its *Cons* role– to the stable situation.

A buffer also has a *Snk* partition and role, like a worker also has a *Prod* partition and role. These partitions and roles are very similar to the *Src* and *Cons* ones, although we treat the collaboration of $Wrk_i(Prod)$ and $Buf_i(Sink)$ slightly differently, see rules PR_1^i to PR_4^i below.

Two additional components Buf_0 and Buf_3 represent input and output located at beginning and end of the pipeline. Both have no counting behaviour

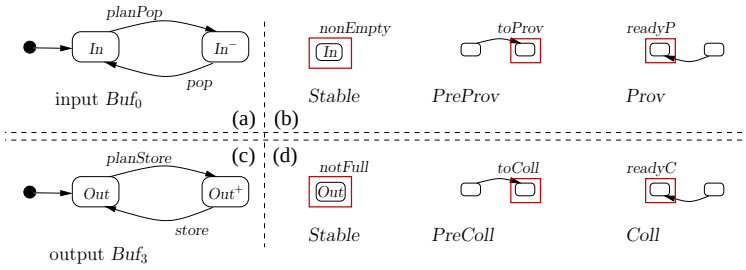

Fig. 6. STDs for (a) Buf_0 and (c) Buf_3, partitions for roles $Buf_0(Src)$ and $Buf_3(Snk)$.

specified. Their respective roles, $Buf_0(Src)$ and $Buf_3(Snk)$, are the same as the earlier *Src* and *Snk* roles of Buf_1 and Buf_2, see Figure 6 for the STDs' partitions.

We have two sets of consistency rules, representing two protocols for each $i = 1, 2, 3$ separately, together constituting six protocols. Consistency rules CR_1^i–CR_4^i specify the collaboration of Wrk_i and Buf_{i-1}, for $i = 1, 2, 3$, in their roles *Cons* and *Src*.

$$Wrk_i(Cons)\colon NoTake \xrightarrow{reqT} NoTake \parallel Buf_{i-1}(Src)\colon Stable \xrightarrow{nonEmp} PreProv \quad (CR_1^i)$$

$$Buf_{i-1}(Src)\colon PreProv \xrightarrow{toProv} Prov \quad (CR_2^i)$$

$$Wrk_i(Cons)\colon NoTake \xrightarrow{reqT} Take \parallel Buf_{i-1}(Src)\colon Prov \xrightarrow{readyP} Stable \quad (CR_3^i)$$

$$Wrk_i(Cons)\colon Take \xrightarrow{doneT} NoTake \quad (CR_4^i)$$

When the worker, in need of an item, is in state *Wanting* that coincides with trap *reqT* of phase *NoTake*, and the providing buffer is stable and non-empty, i.e. is in the corresponding phase and trap, the buffer proceeds the provisioning (CR_1^i). Once this has been accomplished, via phase *PreProv* and subsequently via phase *Prov* (CR_2^i), then by reaching trap *readyP* of the buffer's phase *Prov*, the worker is allowed to take it, while the buffer returns to its stable phase (CR_3^i). Once the worker has actually taken the item, being in trap *doneT* of phase *Take*, it returns to phase *NoTake* again (CR_4^i).

Consistency rules PR_1^i–PR_4^i specify the collaboration of worker Wrk_i and Buf_i, for $i = 1, 2, 3$, in their roles *Prod* and *Snk*.

$$Wrk_i(Prod)\colon NoGive \xrightarrow{reqG} NoGive \parallel Buf_i(Snk)\colon Stable \xrightarrow{notFull} PreColl \quad (PR_1^i)$$

$$Wrk_i(Prod)\colon NoGive \xrightarrow{reqG} Give \parallel Buf_i(Snk)\colon PreColl \xrightarrow{toColl} PreColl \quad (PR_2^i)$$

$$Wrk_i(Prod)\colon Give \xrightarrow{doneG} NoGive \parallel Buf_i(Snk)\colon PreColl \xrightarrow{toColl} Coll \quad (PR_3^i)$$

$$Buf_i(Snk)\colon Coll \xrightarrow{readyColl} Stable \quad (PR_4^i)$$

The worker can dispense an item when in phase *NoGive* trap *reqG* has been reached, while the collecting buffer or output is stable, but not full, i.e. in phase

Stable and trap *notFull* (PR_1^i). The worker proceeds to delivering the item by moving to phase *Give*, while the buffer remains stand-by in phase *PreColl* (PR_2^i). The worker signals that the item has been dispensed off, via trap *doneG*, for the buffer to collect it in phase *Coll* (PR_3^i). Once the item is collected by the buffer, witnessed by trap *readyC*, the buffer returns to its *Stable* phase (PR_4^i).

5 Proving model properties

In this section we show how triples for specific collaborations can be glued together to verify a property for the complete system. Initially we focus on small subsystems, and we combine the local properties obtained to yield a property of the system as a whole. In this paper, we will consider a data flow property of the pipeline example.

For the Paradigm model of the pipeline, we shall verify that the number of *store* actions by the output Buf_3 doesn't exceed the number of *pop* actions by the input Buf_0. More precisely, we will show

$$\#pop_0 - b_1 - b_2 - 4 \leqslant \#store_3 \leqslant \#pop_0 - b_1 - b_2 \tag{10}$$

where the *pop*s and *store*s are executed by Buf_0 and Buf_3, respectively, and b_1, b_2 are the number of items in Buf_1 and Buf_2 at the moment of inspection. For simplicity, we assume the buffers Buf_1 and Buf_2 to be of capacity 3 rather than of arbitrary capacity n. Verification of this property by means of model checking, e.g. with the `mCRL2` toolset, may be a chancy undertaking. However, application of the collaboration-driven approach as sketched in the previous section shows to be quite feasible.

The general idea is to first consider the situation for a worker, examining its simultaneous interaction with a providing buffer and a collecting buffer, to conclude a causality among the two rule sets involved. Next, we consider the situation for a buffer, establishing a relationship among the rule sets of the collaboration in which it has a role. Since *store* and *pop* actions correspond to specific phases, and phases correspond to specific rules, we are able to relate *pop*s of the input all the way through the pipeline to *store*s of the output.

To be able to use patterns in observed behavior of consistency rules to prove a property in terms of local actions, we need to couple the actions to the rules: A *pop* of Buf_0 is only possible in its phase *Prov*. However, in this phase the action can only be executed once. Therefore, every *pop* implies that consistency rule CR_1^0 has again taken place. Similarly, action *store* by Buf_3 is only allowed in phase *Coll*. Only when trap *readyC* of this phase has been reached we are sure that the action has indeed been executed. Such is confirmed by consistency rule PR_4^3. So, for the analysis of the property we need to relate occurrences of CR_1^0 to those of PR_4^3.

When looking closer at the collaborations of Buf_{i-1}, Wrk_i, and Buf_i, for $i = 1, 2, 3$, these are governed by the groups of rules $CR_1^i, \ldots, CR_4^i$ vs. $PR_1^i, \ldots, PR_4^i$. Inspection of the operational semantics reveals that one can find characteristic

formula φ_X and φ_Y, representing finite sets of configurations, such that

$$\varphi_X \,[\,\{CR_1^i CR_2^i CR_3^i\} \cup (CR_1^i CR_2^i CR_3^i \parallel PR_4^i)\,]\,\varphi_Y \quad \text{and}$$
$$\varphi_Y \,[\,\{PR_1^i PR_2^i PR_3^i\} \cup (PR_1^i PR_2^i PR_3^i \parallel CR_4^i)\,]\,\varphi_X \qquad (11)$$

It follows that for the subsystem of Buf_{i-1}, Wrk_i, and Buf_i, for given index i, application of rules $CR_1^i CR_2^i CR_3^i$ *precedes* application of rules $PR_1^i PR_2^i PR_3^i$. Reversely, all but the first occurrence of the subsequence $CR_1^i CR_2^i CR_3^i$, and possibly the last, are enclosed by two subsequences $PR_1^i PR_2^i PR_3^i$. Moreover, considering the phases we readily see that the two rule sets are applied cyclically. Rules CR_1^i to CR_4^i can only be applied in that order. The same holds for PR_1^i to PR_4^i.

Next we shift to subsystems involving Wrk_i, Buf_i, and Wrk_{i+1}, for $i = 1, 2$. Because of the assumed buffer capacity of 3, we distinguish characteristic formula ψ_j^i, $j = 0, 1, 2, 3$, for which it holds that

$$\psi_{j-1}^i \,[\, CR_2^{i+1} \parallel (\varepsilon + PR_4^i)\,]\,\psi_j^i \quad \text{and}$$
$$\psi_j^i \,[\, PR_2^i PR_3^i \parallel (\varepsilon + CR_3^{i+1} + CR_3^{i+1} CR_4^{i+1})\,]\,\psi_{j-1}^i \qquad (12)$$

for $i = 1, 2$, $j = 1, 2, 3$ and were ε denotes the empty sequence of rule actions. For configurations satisfying formula ψ_j^i we have that Buf_i holds j elements. Thus, the buffer moves up from $j{-}1$ to j elements when rule CR_2^i is executed. Reversely, the buffer moves down from j to $j{-}1$ elements when rules PR_2^i and PR_3^i are executed.

Now let $\gamma_0 \xrightarrow{w}{}^* \gamma'$ be any computation of the full pipeline system running from its start configuration γ_0, where all buffers are in state *In*, 0, or *Out* and in stable phases for each partition, and where all workers are in state *Wanting* and in phases *NoTake* and *NoGive*. Suppose the number of *pop*s performed by Buf_0, denoted by $\#pop_0$, equals p. Then, as argued above, we have $\#CR_2^0 = p$, with $\#CR_2^0$ denoting the number of occurrences of rule action CR_2^0 in w. Since CR_2^0 is part of a subsequence $CR_1^0 CR_2^0 CR_3^0$, and the subsequences $CR_1^0 CR_2^0 CR_3^0$ and $PR_1^1 PR_2^1 PR_3^1$ of w alternate, as concluded from property (11), it follows that $p{-}1 \leqslant \#PR_2^1 \leqslant p$. Because of the relationship of PR_2^1, CR_2^2 and the content b_1 of Buf_1 of property (12), we obtain $p{-}b_1{-}1 \leqslant \#CR_2^2 \leqslant p{-}b_1$. Continuing along the pipeline we similarly obtain $p{-}b_1{-}2 \leqslant \#PR_2^2 \leqslant p{-}b_1$, and $p{-}b_1{-}b_2{-}3 \leqslant \#PR_2^3 \leqslant p{-}b_1{-}b_2$. Therefore, $p{-}b_1{-}b_2{-}4 \leqslant \#PR_4^3 \leqslant p{-}b_1{-}b_2$. Thus, for $\#store_3$, the number of *store*s performed by Buf_3, we have $p{-}4{-}b_1{-}b_2 \leqslant \#store_3 \leqslant p{-}b_1{-}b_2$. Substituting $\#pop_0$ for p we obtain Equation (10), as was to be shown.

Although only a sketch of a proof for the pipeline property has been given, the presentation aims to stress the component-driven nature of the approach guided by the form of the underlying architecture of the system. The split required by Paradigm, distinguishing local behaviour vs. global interaction, is rather useful for the modularity. Vertical reasoning focuses on the consequences of local transitions for consistency rules to become enabled, at least partially, when a transition makes the component to enter a trap of a phase, as well as on the restriction of the locus of control by phases and traps, being in a phase or trap implies

being in a specific subset of states. Horizontal reasoning concerns the order in which consistency rules can be applied, thus it considers the synchronicity among phases and traps of multiple collaborating components. Clearly, more examples need to be examined and more case studies need to be performed to underpin the proposed method of (i) deriving triples for interacting component groups, and (ii) combining theses triples to verify the complete architecture. However, we argued that by reasoning about specific collaborations and by combining the emerging patterns we are able to deal with properties of larger systems modeled than a straightforward model checking approach of a Paradigm model, that is deemed to face state space explosion, would allow. The separation of global collaboration and local computation underlying Paradigm is expected to be helpful in pursuing this approach.

References

1. S. Andova, L.P.J. Groenewegen, J. Stafleu, and E.P. de Vink. Formalizing adaptation on-the-fly. In G. Salaün and M. Sirjani, editors, *Proc. FOCLASA 2009*, pages 23–44. ENTCS 255, 2009.
2. S. Andova, L.P.J. Groenewegen, and E.P. de Vink. Towards dynamic adaptation of probabilistic systems. In T. Margaria and B. Steffen, editors, *Proc. ISOLA 2010*, pages 143–159. LNCS 6414, 2010.
3. S. Andova, L.P.J. Groenewegen, and E.P. de Vink. Dynamic consistency in process algebra: From Paradigm to ACP. *Science of Computer Programming*, 76(8):711–735, 2011.
4. S. Andova, L.P.J. Groenewegen, and E.P. de Vink. Dynamic adaptation with distributed control in Paradigm. *Science of Computer Programming*, 94:333–361, 2014.
5. J.W. de Bakker and E.P. de Vink. *Control Flow Semantics*. Foundations of Computing Series. The MIT Press, 1996.
6. B.H.C. Cheng, R. de Lemos, H. Giese, P. Inverardi, and J. Magee, editors. *Software Engineering for Self-Adaptive Systems*. LNCS 5525, 2009.
7. G. Engels and L. Groenewegen. Specifications of coordinated behaviour by SOCCA. In B. Warboys, editor, *Proc. EWSPT*, pages 128–151. LNCS 772, 1994.
8. L. Groenewegen and E. de Vink. Operational semantics for coordination in Paradigm. In F. Arbab and C. Talcott, editors, *Proc. Coordination 2002*, pages 191–206. LNCS 2315, 2002.
9. C. B. Jones. *Development Methods for Computer Programs including a Notion of Interference*. PhD thesis, Oxford University, 1981.
10. C. Pierik and F.S. de Boer. A proof outline logic for object-oriented programming. *Theoretical Computer Science*, 343:413–442, 2005.
11. W. Reimer, W. Schäfer, and T. Schmal. Towards a dedicated object-oriented software process modeling language. In J. Bosch and S. Mitchell, editors, *Proc. ECOOP'97*, pages 299–302. LNCS 1357, 1998.
12. W.-P. de Roever, F. de Boer, U. Hannemann, J. Hooman, Y. Lakhnech, M. Poel, and J. Zwiers. *Concurrency Verification: Introduction to Compositional and Noncompositional Methods*. Cambridge University Press, 2001.
13. Q. Xu, W.P. de Roever, and J. He. The rely-guarantee method for verifying shared variable concurrent programs. *Formal Aspects of Computing*, 9(2):149–174, 1997.