Dead-Path-Elimination in BPEL4WS

Franck van Breugel and Mariya Koshkina

Technical Report CS-2005-04

April 2005

Department of Computer Science and Engineering

4700 Keele Street, Toronto, Ontario M3J 1P3, Canada

Dead-Path-Elimination in BPEL4WS*

Franck van Breugel
York University, 4700 Keele Street, Toronto, M3J 1P3, Canada
franck@cs.yorku.ca
Mariya Koshkina
IBM Toronto Lab, 8200 Warden Avenue, Markham, L6G 1C7, Canada
mkoshkin@ca.ibm.com

April 2005

Abstract

Dead-path-elimination (DPE) is a key ingredient of the business process execution language for web services (BPEL4WS). In this paper, we introduce a small language called the BPE-calculus which contains those constructs of BPEL4WS that are most relevant to DPE. We present three models for the BPE-calculus: one without DPE, one with DPE, and one with our proposed modification of DPE. We formulate a condition and show that it is sufficient and necessary for (modified) DPE to be free of (unintended) side effects. More precisely, we prove the following two properties. First of all, if the condition is satisfied, then the behaviour of a BPE-process is the same in the model without DPE and the model with (modified) DPE. Secondly, if the condition is not satisfied, then we can construct a BPE-process that behaves differently in the models. As a consequence, if the condition is satisfied, then DPE becomes an optimisation. In that case, programmers can ignore DPE and, hence, programming in BPEL4WS becomes simpler.

1 Introduction

Recently, BEA, IBM, Microsoft, SAP and Siebel Systems introduced the business process execution language for web services (BPEL4WS) [ACD+03]. BPEL4WS has been designed to compose web services. For an introduction to web services, we refer the reader to, for example, [Que03]. Web service composition is discussed in, for example, [KS03]. BPEL4WS has been submitted to OASIS for standardisation and is expected to become the basis of a de facto standard for web service composition.

BPEL4WS is XML based, that is, its syntax is defined in terms of an XML grammar. For example, the BPEL4WS snippet

```
<invoke partner="producer"
  operation="sell"
  inputVariable="offer"
  outputVariable="confirmation">
</invoke>
```

invokes the web service operation `sell` of the `producer`.

In BPEL4WS, the basic activities include assignments, invoking web service operations, receiving requests, and replying to requests. These basic activities are combined into structured activities using ordinary sequential control flow constructs like sequencing, switch constructs, and while loops.

*This research is supported by IBM Canada Ltd. and the Natural Sciences and Engineering Research Council of Canada.

1.1 Concurrency in BPEL4WS

Concurrency is provided by the flow construct. For example, in

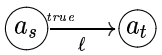
```
<flow>
  buy
  sell
</flow>
```

the activities `buy` and `sell`, whose behaviour has been left unspecified to simplify the example, are concurrent. The pick construct allows for selective communication. Consider, for example,

```
<pick>
  <onMessage partner="consumer">
    sell
  </onMessage>
  <onMessage partner="producer">
    buy
  </onMessage>
</pick>
```

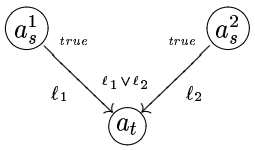
On the one hand, if a message from `consumer` is received then the activity `sell` is executed. In that case, the `buy` activity will not be performed. On the other hand, the receipt of a message from `producer` triggers the execution of the `buy` activity and discards the `sell` activity. In the case that both messages are received almost simultaneously, the choice of activity to be executed depends on the implementation of BPEL4WS. This pick construct is similar to the summation construct found in calculi like, for example, CCS [Mil89].

Synchronisation between concurrent activities is provided by means of links. Each link has a source activity and a target activity. Furthermore, a transition condition is associated with each link. The latter is a Boolean expression that is evaluated when the source activity terminates. Its value is associated with the link. As long as the transition condition of a link has not been evaluated, the value of the link is undefined. In this paper, we will use, for example



to depict that link ℓ has source a_s and target a_t and transition condition *true*. The source and target of a link cannot change at run time and, hence, links cannot be used to express mobility in the way names can in the π-calculus [Mil99].

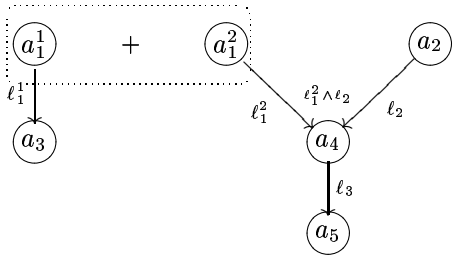
Each activity has a join condition. This condition consists of incoming links of the activity combined by Boolean operators. Only when all the values of its incoming links are defined and its join condition evaluates to true can an activity start. As a consequence, if its join condition evaluates to false then the activity never starts. We will use, for example,



to depict that the join condition of activity a_t is $\ell_1 \vee \ell_2$. In the above example, activity a_t can only start after activities a_s^1 and a_s^2 have finished. From now on, we will use the convention that the default transition condition (*true*) and the default join condition (disjunction of the incoming links) are not depicted.

1.2 Dead-Path-Elimination

Let us consider the following activities and links.



In the above picture, we use + to depict the pick construct. Note that the choice between the activities a_1^1 and a_1^2 determines which of the activities a_3, a_4 and a_5 are performed. For example, if a_1^1 is chosen then a_3 is executed. In that case neither a_4 nor a_5 can ever occur. As a consequence, the activities a_4 and a_5 could be "garbage collected." Here, we use the term "garbage collection" to denote the process of automatically reclaiming activities (rather than memory). The activities can be "garbage collected" as follows.

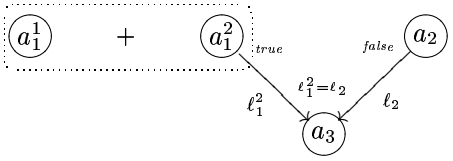
- If a pick construct is executed, then we also assign false to all the outgoing links of those branches of the pick construct that are not chosen.
- If the join condition of an activity evaluates to false, then the activity is "garbage collected" after assigning false to its outgoing links.

The first rule is also applicable to the switch construct. Since the pick and switch constructs are very similar, we only consider the pick construct in this paper. This "garbage collection" scheme is named dead-path-elimination (DPE) [LA94, ACD+03]. DPE not only "garbage collects" activities but also simplifies termination detection of structured activities [Rol03].

Let us briefly return to the above example. Assume that activity a_1^1 is chosen. Then, as a result of DPE, the value of the link ℓ_1^2 becomes false. When activity a_2 terminates, the link ℓ_2 gets the default value true. At this point, the join condition of activity a_4 can be evaluated. Since its value is false, by DPE, false is assigned to link ℓ_3 and activity a_4 is "garbage collected." Subsequently, again exploiting DPE, activity a_5 can be "garbage collected" as well.

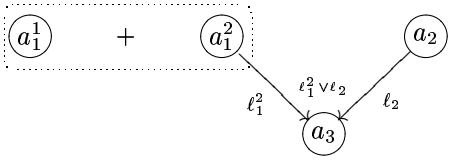
1.3 The Problem

Now let us consider another example.



At first sight, one may be tempted to conclude that activity a_3 will never be executed. Without DPE this is indeed the case. However, DPE may trigger the execution of activity a_3 as follows. Assume that activity a_1^1 is chosen. By DPE, the value of the link ℓ_1^2 becomes false. Since the value of the link ℓ_2 becomes false as well, the join condition $\ell_1^2 = \ell_2$ evaluates to true. Hence, activity a_3 can be performed. The above can be paraphrased as DPE may have side effects. We believe that side effects as in the above example may be introduced unintentionally.

Next, we consider the following example.



On the one hand, if we do not have DPE and the activity a_1^1 is chosen, then the link ℓ_1^2 will be undefined forever. Since join conditions in BPEL4WS are only evaluated once all links that are part of the join condition are defined, the join condition $\ell_1^2 \vee \ell_2$ will never be evaluated in this case and, hence, the activity a_3 will never be executed. On the other hand, if the activity a_1^1 is chosen, then DPE may trigger the execution of activity a_3 as follows. When the activity a_1^1 is chosen, the value of the link ℓ_1^2 becomes false by DPE. After activity a_2 has been performed, the value of the link ℓ_2 is set to true. At this point, the join condition can be evaluated. Since its value is true, activity a_3 is executed in this case. This is another example of a process that behaves differently with and without DPE.

1.4 The Analysis

The side effect in the first example of the previous section is caused by negative occurrences of links in the join condition. The join condition $\ell_1^2 = \ell_2$ is equivalent to the Boolean expression $(\ell_1^2 \wedge \ell_2) \vee (\neg\ell_1^2 \wedge \neg\ell_2)$. The links ℓ_1^2 and ℓ_2 both have a negative occurrence in the disjunctive normal form of the join condition. By disallowing negative occurrences of links in join conditions, side effects like the one in the first example can be eliminated.

In the second example of the previous section, the side effect is caused by the fact that all links need to be defined before a join condition can be evaluated. By evaluating the join condition as soon as its truth value is known, this type of side effect can be removed as well. In that case, $\bot \vee \mathit{true}$ is *true* (where $\bot$ denotes undefined). This type of evaluation is supported in, for example, the web service flow language (WSFL) [Ley01].

1.5 Our Solutions

As we have already mentioned above, by disallowing negative occurrences of links in join conditions and by evaluating the join condition as soon as its truth value is known, DPE becomes free of side effects.

However, there is an alternative remedy. Note that the value of a link is either undefined or it has the value true or false. We cannot distinguish between

- a link whose value is false because its transition condition evaluated to false, and
- a link whose value is false due to DPE.

However, if we can distinguish between these two cases, then side effects like the one in the first example of Section 1.3 do not occur. This can be accomplished by modifying DPE in the following way. Links can either be undefined, have the value true or false, or be defined but have a value different from true and false. The latter value, denoted $\top$, is used to denote that the link's value has been set by DPE. Hence, our proposed modification of DPE amounts to the following ingredients:

- If a pick construct is executed, then we also assign $\top$ to all the outgoing links of those branches of the pick construct that are not chosen.
- If the join condition of an activity evaluates to false, then the activity is "garbage collected" after assigning $\top$ to its outgoing links.

In the first example of Section 1.3, modified DPE sets the value of link ℓ_1^2 to $\top$ rather than false. As a consequence, the join condition $\ell_1^2 = \ell_2$ evaluates to false rather than true and, hence, activity a_3 is not executed. Therefore, this example gives rise to the same behaviour without DPE and with modified DPE.

Let us consider modified DPE for the second example of Section 1.3. If the activity a_1^1 is chosen, then modified DPE may also trigger the execution of activity a_3. When the activity a_1^1 is chosen, the value of the link ℓ_1^2 becomes $\top$ this time. After activity a_2 has been performed, the value of the link ℓ_2 is set to true. At this point, the join condition can be evaluated since both links are defined. Since $\top \vee \mathit{true}$ is *true*, activity a_3 is also executed in this case. Hence, this example behaves differently without DPE and with modified DPE. However, if a join condition is evaluated as soon as its value is known, then modified DPE is free of side effects. In this case, negative occurrences of links in join conditions are allowed.

1.6 The Technical Details

The above examples show that (modified) DPE can have side effects. The proposal of the modification of DPE and the formulation of a condition and the proofs that this condition is sufficient and necessary for (modified) DPE to be free of side effects are the main contributions of this paper. The condition is named SEF for side effect free. The condition is formulated in terms of the evaluation function of join conditions. This function determines the value of a join condition, given the values of the links. The evaluation function of BPEL4WS does not satisfy the condition. Below, we describe how our main results are proved. Furthermore, we mention the other contributions of this paper.

Rather than studying BPEL4WS as a whole, we focus on a small language that includes all the concepts we introduced above. Furthermore, we leave unspecified some syntactic categories, like for example the basic activities. That is, we assume a set of basic activities but we do not specify their syntax. The language so obtained we call the BPE-calculus, as it is similar in flavour to calculi like, for example, CCS [Mil89]. In the BPE-calculus, we abstract from many details to obtain a small language to study the essence of DPE. For a calculus that captures more of BPEL4WS, we refer the reader to [Kos03]. A calculus with compensation handlers, an ingredient of BPEL4WS that is not considered in [Kos03], has been studied in [BLZ03, CGV+02]. Although we only consider an abstraction in the form of the BPE-calculus, we are confident that our results scale up to extended calculi for BPEL4WS and even BPEL4WS as a whole.

In order to prove our main result, we introduce three models for the BPE-calculus. The predominant approach to model calculi like our BPE-calculus is to capture the behaviour of BPE-processes in terms of a labelled transition system, as advocated in, for example, [Plo04]. In this paper, we will follow this approach as well. Alternative approaches to model business processes, namely by means of Petri nets and distributed abstract state machines, are discussed in the concluding section of this paper. The first labelled transition system models the BPE-calculus in the absence of DPE. The second one captures the BPE-calculus with DPE. The third and final system models the BPE-calculus with our proposed modification of DPE.

To show that the condition SEF is sufficient and necessary for (the modification of) DPE to be free of side effects, we prove the following two properties. First of all, if the condition SEF is satisfied, then the behaviour of a BPE-process is the same in the model without DPE and the model with (modified) DPE. Secondly, if the condition SEF is not satisfied, then we can construct a BPE-process that behaves differently in the models. In the next two paragraphs, we discuss how we prove these properties.

To capture those states of a labelled transition system—in our case, these are the BPE-processes— that behave the same, it is common practice to introduce a behavioural equivalence on the states of the system. Many different behavioural equivalences have been introduced in the literature. To prove the first result, we exploit weak bisimilarity which is among the strongest of the weak behavioural equivalences. Recall that a behavioural equivalence is called weak if it abstracts from internal actions. We show that if the condition SEF is satisfied, then a BPE-process in the labelled transition system without DPE is weak bisimilar to the BPE-process in the system with (modified) DPE. We actually prove the following even stronger result. If the condition holds, then a BPE-process in the labelled transition system without DPE is expanded by the BPE-process in the system with (modified) DPE. Expansion, which we denote by $\precsim$, is a behavioural preorder that was first introduced in [AH92]. Since expansion is stronger than weak bisimilarity, the latter result implies the former. To prove the latter result, we exploit the powerful proof technique of expansion up to $\precsim$ which was introduced in [SM92].

In the proof of the second result, we use weak trace equivalence which is one of the weakest behavioural equivalences. We show that if the condition SEF is not satisfied then we can construct a BPE-process such that this process in the labelled transition system without DPE is not trace equivalent to the same process in the system with (modified) DPE.

1.7 Overview

The rest of this paper is organised as follows. In Section 2, we introduce our BPE-calculus. In Section 3 and 4, we present labelled transition systems that model the BPE-calculus without and with DPE. The condition is introduced in Section 5. In Section 6 and 7, we show that SEF is a sufficient and necessary condition. In Section 8, we model our proposed modification of DPE by means of a labelled transition system, adapt the condition to the modified setting, and prove that this adaptation of the condition is also

a sufficient and necessary condition. In Section 9, we focus on the evaluation of join conditions. The final section concludes and discusses some related work.

2 The BPE-calculus

The BPE-calculus concentrates on those constructs of BPEL4WS that are key to DPE. It also abstracts from many details of BPEL4WS that are not relevant for our study of DPE.

Before defining BPE-processes, we first fix

- a set $\mathbb{A}$ of basic activities,
- an infinite set $\mathbb{L}$ of links and
- a set $\mathbb{C}$ of join conditions.

Definition 1 *The set $\mathbb{P}$ of processes is defined by*

$$\begin{array}{lcl} P & ::= & 0 \mid a.P \mid \ell \uparrow b.P \mid c \to P \mid Q + Q \mid P \parallel P \\ Q & ::= & a.P \mid Q + Q \end{array}$$

where $a \in \mathbb{A}$, $\ell \in \mathbb{L}$, $b \in \{true, false\}$ and $c \in \mathbb{C}$.

Most operators of the BPE-calculus are very similar to the operators of calculi like, for example, CCS [Mil89]. Let us focus on the new ingredients of our calculus. In the process $\ell \uparrow b.P$, the Boolean value b is the value of the transition condition associated to the link ℓ. In the process $c \to P$, c is the join condition of process P.

For example, the BPEL4WS snippet described by the last picture of the introductory section is captured by the BPE-process

$$(a_1^1.0 + a_1^2.\ell_1^2 \uparrow true.0) \parallel a_2.\ell_2 \uparrow true.0 \parallel (\ell_1^2 \vee \ell_2) \to a_3.0$$

In BPEL4WS, each link should have a unique source. We capture this restriction by means of the following very simple type system. Only if a process satisfies this restriction can it be typed. Its type will be the set of its outgoing links.

Definition 2 *The relation $\Uparrow \subseteq \mathbb{P} \times 2^{\mathbb{L}}$ is defined by*

(NIL) $0 \Uparrow \emptyset$

(PREF) $$\frac{P \Uparrow L}{a.P \Uparrow L}$$

(OUT) $$\frac{P \Uparrow L \qquad \ell \notin L}{\ell \uparrow b.P \Uparrow L \cup \{\ell\}}$$

(JOIN) $$\frac{P \Uparrow L}{c \to P \Uparrow L}$$

(PICK) $$\frac{P_1 \Uparrow L_1 \qquad P_2 \Uparrow L_2 \qquad L_1 \cap L_2 = \emptyset}{P_1 + P_2 \Uparrow L_1 \cup L_2}$$

(FLOW) $$\frac{P_1 \Uparrow L_1 \qquad P_2 \Uparrow L_2 \qquad L_1 \cap L_2 = \emptyset}{P_1 \parallel P_2 \Uparrow L_1 \cup L_2}$$

Not every process can be typed. For example, the process $\ell \uparrow true.0 \parallel \ell \uparrow true.0$ cannot be typed. However, if a process is well-typed then its type is unique. That is, if $P \Uparrow L_1$ and $P \Uparrow L_2$ then $L_1 = L_2$. Furthermore, each type is finite. That is, if $P \Uparrow L$ then L is a finite set of links.

3 A Model without Dead-Path-Elimination

Since we want to compare the BPE-calculus without DPE and the BPE-calculus with (modified) DPE, we introduce three models: one without DPE, one with DPE, and one with our proposed modification of DPE. All three models are defined in terms of a labelled transition system. Below, we present the labelled transition system without DPE. The labelled transition system with (modified) DPE are given in Section 4 and 8.

In the labelled transition system, we need to keep track of the values of the links. The value of a link is either true, false, or undefined. The latter we denote by $\bot$.

Definition 3 *The set Λ of links statuses is defined by*

$$\Lambda = \mathbb{L} \to \{\mathit{true}, \mathit{false}, \bot\}.$$

The initial links status $\lambda_\bot$ assigns $\bot$ to each link. The links status $\lambda\{L \mapsto b\}$ is defined by

$$\lambda\{L \mapsto b\}(\ell) = \begin{cases} b & \text{if } \ell \in L \\ \lambda(\ell) & \text{otherwise} \end{cases}$$

Instead of $\lambda\{\{\ell\} \mapsto b\}$ we often just write $\lambda\{\ell \mapsto b\}$. To model the evaluation of join conditions, we fix an evaluation function $\mathcal{C} : \mathbb{C} \to \Lambda \to \{\mathit{true}, \mathit{false}, \bot\}$. We assume that this evaluation function $\mathcal{C}$ satisfies

$$\text{if } \mathcal{C}(c)(\lambda\{\ell \mapsto \bot\}) \neq \bot \text{ then } \mathcal{C}(c)(\lambda) = \mathcal{C}(c)(\lambda\{\ell \mapsto \bot\}) \tag{1}$$

for all $c \in \mathbb{C}$, $\lambda \in \Lambda$ and $\ell \in \mathbb{L}$. This is a very natural assumption (compare with, for example, [Win93, Lemma 9.3], where the evaluation of expressions is shown to be continuous).

Next, we define the labelled transition system without DPE. The states of the system are pairs, each consisting of a process P and a links status λ. To distinguish the process P with links status λ in the labelled transition system without DPE and from the process P with links status λ in the system with (modified) DPE, we denote the former by (P, λ) and the latter by (P, λ) (and (P, λ)).

There are two types of labels. A transition is either labelled by a basic activity or by ι. The former captures the execution of a basic activity whereas the latter models an internal action. Whether these internal actions modelled by ι are observable or not has no impact on our results. We use α to denote the labels of the system. That is, α is either a basic activity or it is ι.

The transitions of the system are defined in the following definition.

Definition 4 *The labelled transition relation $\to \subseteq \mathbb{P} \times \Lambda \times (\mathbb{A} \cup \{\iota\}) \times \mathbb{P} \times \Lambda$ is defined by*

(PREF) $(a.P, \lambda) \xrightarrow{a} (P, \lambda)$

(OUT) $(\ell \uparrow b.P, \lambda) \xrightarrow{\iota} (P, \lambda\{\ell \mapsto b\})$

(JOIN$_t$) $$\frac{\mathcal{C}(c)(\lambda) = \mathit{true}}{(c \to P, \lambda) \xrightarrow{\iota} (P, \lambda)}$$

(PICK$_\ell$) $$\frac{(P_1, \lambda) \xrightarrow{\alpha} (P_1', \lambda')}{(P_1 + P_2, \lambda) \xrightarrow{\alpha} (P_1', \lambda')}$$

(PICK$_r$) $$\frac{(P_2, \lambda) \xrightarrow{\alpha} (P_2', \lambda')}{(P_1 + P_2, \lambda) \xrightarrow{\alpha} (P_2', \lambda')}$$

(FLOW$_\ell$) $$\frac{(P_1, \lambda) \xrightarrow{\alpha} (P_1', \lambda')}{(P_1 \parallel P_2, \lambda) \xrightarrow{\alpha} (P_1' \parallel P_2, \lambda')}$$

(FLOW$_r$) $$\frac{(P_2, \lambda) \xrightarrow{\alpha} (P_2', \lambda')}{(P_1 \parallel P_2, \lambda) \xrightarrow{\alpha} (P_1 \parallel P_2', \lambda')}$$

The rules (PREF), (PICK_ℓ), (PICK_r), (FLOW_ℓ) and (FLOW_r) are very similar to the rules that are used to model prefixing, summation and composition in CCS. The rule (OUT) captures that process $\ell \uparrow b.P$ sets the value of link ℓ to Boolean value b. Process $c \to P$ evaluates the join condition c, given the values of the links represented by the links status λ. If this evaluation results in true, then the process $c \to P$ can make a transition.

Next, we show that if a process is well-typed, then all processes reachable from that process by means of a transition are well-typed as well. This property is known as subject reduction.

Proposition 5 *If* $(P, \lambda) \xrightarrow{\alpha} (P', \lambda')$ *and* $P \Uparrow L$ *then* $P' \Uparrow L'$ *for some* $L' \subseteq L$.

Proof We prove this proposition by transition induction.

- Consider the transition $(a.P, \lambda) \xrightarrow{a} (P, \lambda)$. Assume that $a.P \Uparrow L$. Then $P \Uparrow L$.
- Consider the transition $(\ell \uparrow b.P, \lambda) \xrightarrow{\iota} (P, \lambda\{\ell \mapsto b\})$. Assume that $\ell \uparrow b.P \Uparrow L$. Then $P \Uparrow L \setminus \{\ell\}$.
- Consider the transition $(c \to P, \lambda) \xrightarrow{\iota} (P, \lambda)$. Assume that $c \to P \Uparrow L$. Then $P \Uparrow L$.
- Consider the proof

  $$\frac{(P_1, \lambda) \xrightarrow{\alpha} (P_1', \lambda')}{(P_1 + P_2, \lambda) \xrightarrow{\alpha} (P_1', \lambda')}$$

  Furthermore, assume the proof

  $$\frac{P_1 \Uparrow L_1 \qquad P_2 \Uparrow L_2 \qquad L_1 \cap L_2 = \emptyset}{P_1 + P_2 \Uparrow L_1 \cup L_2}$$

  By induction, $P_1' \Uparrow L_1'$ for some $L_1' \subseteq L_1$. Obviously, $L_1' \subseteq L_1 \cup L_2$.
- Consider the proof

  $$\frac{(P_1, \lambda) \xrightarrow{\alpha} (P_1', \lambda')}{(P_1 \parallel P_2, \lambda) \xrightarrow{\alpha} (P_1' \parallel P_2, \lambda')}$$

  Furthermore, assume the proof

  $$\frac{P_1 \Uparrow L_1 \qquad P_2 \Uparrow L_2 \qquad L_1 \cap L_2 = \emptyset}{P_1 \parallel P_2 \Uparrow L_1 \cup L_2}$$

  By induction, $P_1' \Uparrow L_1'$ for some $L_1' \subseteq L_1$. Hence, $L_1' \cup L_2 \subseteq L_1 \cup L_2$. Furthermore, $L_1' \cap L_2 = \emptyset$. Therefore, $P_1' \parallel P_2 \Uparrow L_1' \cup L_2$. □

If process P makes a transition to process P' and the processes have types L and L', respectively, then only the values of links in the set $L \setminus L'$ may have changed as the result of the transition, that is, the links status restricted to $(\mathbb{L} \setminus L) \cup L'$ does not change.

Proposition 6 *If* $(P, \lambda) \xrightarrow{\alpha} (P', \lambda')$ *and* $P \Uparrow L$ *and* $P' \Uparrow L'$ *then* $\lambda \upharpoonright (\mathbb{L} \setminus L) \cup L' = \lambda' \upharpoonright (\mathbb{L} \setminus L) \cup L'$.

Proof We prove this proposition by transition induction.

- Consider the transition $(a.P, \lambda) \xrightarrow{a} (P, \lambda)$. Obviously, the proposition is vacuously true in this case.
- Consider the transition $(\ell \uparrow b.P, \lambda) \xrightarrow{\iota} (P, \lambda\{\ell \mapsto b\})$. Assume that $\ell \uparrow b.P \Uparrow L$. Then $P \Uparrow L \setminus \{\ell\}$ and $\ell \in L$. Note that $(\mathbb{L} \setminus L) \cup (L \setminus \{\ell\}) = \mathbb{L} \setminus \{\ell\}$. Obviously, $\lambda \upharpoonright \mathbb{L} \setminus \{\ell\} = \lambda\{\ell \mapsto b\} \upharpoonright \mathbb{L} \setminus \{\ell\}$.
- Consider the transition $(c \to P, \lambda) \xrightarrow{\iota} (P, \lambda)$. Obviously, the proposition is vacuously true in this case.
- Consider the proof

  $$\frac{(P_1, \lambda) \xrightarrow{\alpha} (P_1', \lambda')}{(P_1 + P_2, \lambda) \xrightarrow{\alpha} (P_1', \lambda')}$$

Furthermore, assume the proof

$$\frac{P_1 \Uparrow L_1 \qquad P_2 \Uparrow L_2 \qquad L_1 \cap L_2 = \emptyset}{P_1 + P_2 \Uparrow L_1 \cup L_2}$$

Assume $P_1' \Uparrow L_1'$. By induction, $\lambda \upharpoonright (\mathbb{L} \setminus L_1) \cup L_1' = \lambda' \upharpoonright (\mathbb{L} \setminus L_1) \cup L_1'$. Hence, $\lambda \upharpoonright (\mathbb{L} \setminus (L_1 \cup L_2)) \cup L_1' = \lambda' \upharpoonright (\mathbb{L} \setminus (L_1 \cup L_2)) \cup L_1'$.

- Consider the proof

$$\frac{(P_1, \lambda) \xrightarrow{\alpha} (P_1', \lambda')}{(P_1 \parallel P_2, \lambda) \xrightarrow{\alpha} (P_1' \parallel P_2, \lambda')}$$

Furthermore, assume the proof

$$\frac{P_1 \Uparrow L_1 \qquad P_2 \Uparrow L_2 \qquad L_1 \cap L_2 = \emptyset}{P_1 \parallel P_2 \Uparrow L_1 \cup L_2}$$

Assume $P_1' \Uparrow L_1'$. Then $P_1' \parallel P_2 \Uparrow L_1' \cup L_2$. By induction, $\lambda \upharpoonright (\mathbb{L} \setminus L_1) \cup L_1' = \lambda' \upharpoonright (\mathbb{L} \setminus L_1) \cup L_1'$. Hence, $\lambda \upharpoonright (\mathbb{L} \setminus (L_1 \cup L_2)) \cup (L_1' \cup L_2) = \lambda' \upharpoonright (\mathbb{L} \setminus (L_1 \cup L_2)) \cup (L_1' \cup L_2)$. □

We restrict our attention to (P, λ)'s with $\lambda(\ell) \neq \perp$ for finitely many $\ell \in \mathbb{L}$, $P \Uparrow L$ and $\lambda(\ell) = \perp$ for all $\ell \in L$. We will call them valid.

Proposition 7 *If $(P, \lambda) \xrightarrow{\alpha} (P', \lambda')$ and (P, λ) is valid then (P', λ') is valid.*

Proof Assume $P \Uparrow L$. Hence, by Proposition 5, $P' \Uparrow L'$ for some $L' \subseteq L$. Let $\ell \in L'$. Then

$$\begin{aligned} \lambda'(\ell) &= \lambda(\ell) \quad [\lambda \upharpoonright (\mathbb{L} \setminus L) \cup L' = \lambda' \upharpoonright (\mathbb{L} \setminus L) \cup L' \text{ according to Proposition 6}] \\ &= \perp \quad [(P, \lambda) \text{ is valid}] \end{aligned}$$

Since $\lambda(\ell) \neq \perp$ for finitely many $\ell \in \mathbb{L}$ and $\lambda \upharpoonright (\mathbb{L} \setminus L) \cup L' = \lambda' \upharpoonright (\mathbb{L} \setminus L) \cup L'$ according to Proposition 6 and the set $L \setminus L'$ is finite (since the sets L and L' are finite), we can conclude that $\lambda'(\ell) \neq \perp$ for finitely many $\ell \in \mathbb{L}$. □

Since initially all links are undefined, we can conclude from the above proposition that at any point only finitely many links are defined, that is, their value is different from $\perp$.

The behavioural equivalence strong bisimilarity, which we denote by $\sim$, is a congruence with respect to the flow construct.

Proposition 8 *If $(P_1, \lambda) \sim (P_1', \lambda')$ then $(P_1 \parallel P_2, \lambda) \sim (P_1' \parallel P_2, \lambda')$.*

Proof The proof of this proposition is a straightforward modification of the proof that bisimilarity is a congruence with respect to composition in CCS as shown in, for example, [Mil89, Proposition 4.10]. One shows that

$$\{ ((P_1 \parallel P_2, \lambda), (P_1' \parallel P_2, \lambda')) \mid (P_1, \lambda) \sim (P_1', \lambda') \}$$

is a bisimulation. □

4 A Model with Dead-Path-Elimination

In the previous section, we modelled the BPE-calculus without DPE. Next, we present a labelled transition system for the BPE-calculus with DPE.

The states of the labelled transition system are of the form (P, λ), where P is a process and λ is a links status. Besides basic activities and ι, a transition can also be labelled by τ. The τ-transitions correspond to the "garbage collection" caused by DPE. These transitions are therefore not observable. The transitions are given in the following definition.

Definition 9 *The labelled transition relation* $\rightarrow \subseteq \mathbb{P} \times \Lambda \times (\mathbb{A} \cup \{\iota, \tau\}) \times \mathbb{P} \times \Lambda$ *is defined by* (PREF), (OUT), (JOIN_t), (FLOW_ℓ), (FLOW_r) *and*

$$(\text{JOIN}_f)\ \frac{\mathcal{C}(c)(\lambda) = \mathit{false} \qquad P \Uparrow L}{(c \rightarrow P, \lambda) \xrightarrow{\tau} (0, \lambda\{L \mapsto \mathit{false}\})}$$

$$(\text{PICK}_\ell^\delta)\ \frac{(P_1, \lambda) \xrightarrow{\alpha} (P_1', \lambda') \qquad P_2 \Uparrow L_2}{(P_1 + P_2, \lambda) \xrightarrow{\alpha} (P_1', \lambda'\{L_2 \mapsto \mathit{false}\})}$$

$$(\text{PICK}_r^\delta)\ \frac{(P_2, \lambda) \xrightarrow{\alpha} (P_2', \lambda') \qquad P_1 \Uparrow L_1}{(P_1 + P_2, \lambda) \xrightarrow{\alpha} (P_2', \lambda'\{L_1 \mapsto \mathit{false}\})}$$

$$(\text{FLOW}_\ell^\tau)\ \frac{(P_1, \lambda) \xrightarrow{\tau} (P_1', \lambda')}{(P_1 \parallel P_2, \lambda) \xrightarrow{\tau} (P_1' \parallel P_2, \lambda')}$$

$$(\text{FLOW}_r^\tau)\ \frac{(P_2, \lambda) \xrightarrow{\tau} (P_2', \lambda')}{(P_1 \parallel P_2, \lambda) \xrightarrow{\tau} (P_1 \parallel P_2', \lambda')}$$

Note that the rules (PICK_ℓ) and (PICK_r) of Definition 4 are replaced with the rules (PICK_ℓ^δ) and (PICK_r^δ). They capture the first ingredient of DPE: if a pick construct is executed, then we assign false to all the outgoing links of the branch of the pick construct that is not chosen. Recall the second ingredient of DPE: if the join condition of a process evaluates to false, then the process is "garbage collected" after assigning false to all its outgoing links. This is captured by the rule (JOIN_f). If (a part of) process P_1 can be "garbage collected", then (that part of) process P_1 can also be "garbage collected" in the process $P_1 \parallel P_2$. This is captured by the rules (FLOW_ℓ^τ) and (FLOW_r^τ).

The properties formulated in Proposition 5, 6 and 7 also hold for the above introduced labelled transition system.

Proposition 10 *If* $(P, \lambda) \xrightarrow{\alpha} (P', \lambda')$ *and* $P \Uparrow L$ *then* $P' \Uparrow L'$ *for some* $L' \subseteq L$.

Proof We prove this proposition by transition induction. Most cases are the same as in the proof of Proposition 5.

- Consider the transition $(c \rightarrow P, \lambda) \xrightarrow{\tau} (0, \lambda\{L \mapsto \mathit{false}\})$. Since $0 \Uparrow \emptyset$, this case vacuously holds.
- Consider the proof

  $$\frac{(P_1, \lambda) \xrightarrow{\tau} (P_1', \lambda')}{(P_1 + P_2, \lambda) \xrightarrow{\tau} (P_1' + P_2, \lambda')}$$

  Assume that

  $$\frac{P_1 \Uparrow L_1 \qquad P_2 \Uparrow L_2 \qquad L_1 \cap L_2 = \emptyset}{P_1 + P_2 \Uparrow L_1 \cup L_2}$$

  By induction, $P_1' \Uparrow L_1'$ for some $L_1' \subseteq L_1$. Hence, $L_1' \cup L_2 \subseteq L_1 \cup L_2$. Furthermore, $L_1' \cap L_2 = \emptyset$. Therefore, $P_1' + P_2 \Uparrow L_1' \cup L_2$. □

Proposition 11 *If* $(P, \lambda) \xrightarrow{\alpha} (P', \lambda')$ *and* $P \Uparrow L$ *and* $P' \Uparrow L'$ *then* $\lambda \restriction (\mathbb{L} \setminus L) \cup L' = \lambda' \restriction (\mathbb{L} \setminus L) \cup L'$.

Proof We prove this proposition by transition induction. Most cases are the same as in the proof of Proposition 6.

- Consider the transition $(c \rightarrow P, \lambda) \xrightarrow{\tau} (0, \lambda\{L \mapsto \mathit{false}\})$ where $P \Uparrow L$. Since $0 \Uparrow \emptyset$, it suffices to show that $\lambda \restriction (\mathbb{L} \setminus L) = \lambda\{L \mapsto \mathit{false}\} \restriction (\mathbb{L} \setminus L)$ which is vacuously true.

- Consider the proof

$$\frac{(P_1, \lambda) \xrightarrow{\tau} (P_1', \lambda')}{(P_1 + P_2, \lambda) \xrightarrow{\tau} (P_1' + P_2, \lambda')}$$

Assume that

$$\frac{P_1 \Uparrow L_1 \qquad P_2 \Uparrow L_2 \qquad L_1 \cap L_2 = \emptyset}{P_1 + P_2 \Uparrow L_1 \cup L_2}$$

Assume that $P_1' \Uparrow L_1'$. Then $P_1' + P_2 \Uparrow L_1' \cup L_2$. By induction, $\lambda \upharpoonright (\mathbb{L} \setminus L_1) \cup L_1' = \lambda' \upharpoonright (\mathbb{L} \setminus L_1) \cup L_1'$. Hence, $\lambda \upharpoonright (\mathbb{L} \setminus (L_1 \cup L_2)) \cup (L_1' \cup L_2) = \lambda' \upharpoonright (\mathbb{L} \setminus (L_1 \cup L_2)) \cup (L_1' \cup L_2)$. □

Proposition 12 *If* $(P, \lambda) \xrightarrow{\alpha} (P', \lambda')$ *and* (P, λ) *is valid then* (P', λ') *is valid.*

Proof Similar to the proof of Proposition 7. □

5 The Side Effect Free Condition

Next, we present the condition which exactly captures when DPE is free of side effects. In the following two sections, we show that this condition, named SEF for side effect free, is sufficient and necessary for DPE to be free of side effects. In Section 8, we will discuss how SEF can be adapted for modified DPE.

To formulate the condition SEF, we introduce the following relation on the values of links and extend it to a relation on links statuses.

Definition 13 *The relation* $\sqsubseteq \subseteq \{true, false, \bot\}^2$ *is the smallest reflexive relation containing* $\bot \sqsubseteq false$*. The relation* $\sqsubseteq \subseteq \Lambda \times \Lambda$ *is defined by*

$\lambda_1 \sqsubseteq \lambda_2$ *if* $\lambda_1(\ell) \sqsubseteq \lambda_2(\ell)$ *for all* $\ell \in \mathbb{L}$.

This relation roughly captures the following. If a link is undefined in the model without DPE then it either is undefined or has value *false* in the one with DPE, and if a link has value *true* or *false* in the model without DPE then it also has this value in the other model.

Now we are ready to formulate the condition SEF on $\mathcal{C}$: for all λ_1, $\lambda_2 \in \Lambda$ and $c \in \mathbb{C}$,

$$\text{if } \mathcal{C}(c)(\lambda_2) = true \text{ and } \lambda_1 \sqsubseteq \lambda_2 \text{ then } \mathcal{C}(c)(\lambda_1) = true. \tag{2}$$

The above condition roughly requires that a join condition evaluates to true in the model without DPE if it evaluates to true in the model with DPE (the converse always holds given that the evaluation function $\mathcal{C}$ satisfies (1)).

6 SEF is Sufficient

Below, we show that if the evaluation function $\mathcal{C}$ satisfies the SEF condition (2) and condition (1), then DPE is free of side effects. More precisely, we prove that if both conditions hold, then DPE has no impact on the behaviour of a process. That is, we show that $(P, \lambda_\bot)$ and $(P, \lambda_\bot)$ are behaviourally equivalent. The stronger the behavioural equivalence, the stronger the result. We prove it for weak bisimilarity, which is one of the strongest behavioural equivalences that abstracts from τ-transitions. In fact, we prove an even stronger result. We show that $(P, \lambda_\bot)$ expands $(P, \lambda_\bot)$. Expansion, introduced in [AH92] and denoted by $\precsim$, is a behavioural preorder derived from weak bisimilarity by, essentially, comparing the number of τ-transitions performed by the processes. It enjoys the powerful proof technique of expansion up to $\precsim$ which was introduced in [SM92].

For the rest of this section, we assume that (1) and (2) both hold. For the definition of expansion, we refer the reader to, for example, [SM92, Definition 3.1]. Below, we do not rely on the definition of expansion. We only make use of a few simple properties of expansion. We also do not present the definition of expansion up to $\precsim$. It can be found in [SM92, Definition 3.4]. In Theorem 18, we prove that

$$\mathcal{E} = \{ ((P, \lambda_1), (P, \lambda_2)) \mid \lambda_1 \sqsubseteq \lambda_2 \}$$

is an expansion up to $\precsim$. According to the definition of expansion up to $\precsim$, it suffices to show that

1. if $(P, \lambda_1) \xrightarrow{\alpha} (P', \lambda_1')$ and $\lambda_1 \sqsubseteq \lambda_2$ then $(P, \lambda_2) \xrightarrow{\alpha} (P', \lambda_2')$ for some λ_2' such that $(P', \lambda_1') \sim \mathcal{E} \precsim (P', \lambda_2')$;
2. if $(P, \lambda_2) \xrightarrow{\alpha} (P', \lambda_2')$ and $\lambda_1 \sqsubseteq \lambda_2$ then $(P, \lambda_1) \xrightarrow{\alpha} (P', \lambda_1')$ for some λ_1' such that $(P', \lambda_1') \precsim \mathcal{E} \precsim (P', \lambda_2')$;
3. if $(P, \lambda_2) \xrightarrow{\tau} (P', \lambda_2')$ and $\lambda_1 \sqsubseteq \lambda_2$ then $(P, \lambda_1) \precsim \mathcal{E} \precsim (P', \lambda_2')$.

To prove 1, 2, and 3, we use the following three lemmas.

Lemma 14 *If* $(P, \lambda_1) \xrightarrow{\alpha} (P', \lambda_1')$ *and* $\lambda_1 \sqsubseteq \lambda_2$ *then* $(P, \lambda_2) \xrightarrow{\alpha} (P', \lambda_2')$ *for some* λ_2' *such that* $\lambda_1' \sqsubseteq \lambda_2'$.

Proof We prove this lemma by transition induction.

- Consider the transition $(a.P, \lambda_1) \xrightarrow{a} (P, \lambda_1)$. Then $(a.P, \lambda_2) \xrightarrow{a} (P, \lambda_2)$.
- Consider the transition $(\ell \uparrow b.P, \lambda_1) \xrightarrow{\iota} (P, \lambda_1\{\ell \mapsto b\})$. Then $(\ell \uparrow b.P, \lambda_2) \xrightarrow{\iota} (P, \lambda_2\{\ell \mapsto b\})$. Since $\lambda_1 \sqsubseteq \lambda_2$, we also have that $\lambda_1\{\ell \mapsto b\} \sqsubseteq \lambda_2\{\ell \mapsto b\}$.
- Consider the transition $(c \to P, \lambda_1) \xrightarrow{\iota} (P, \lambda_1)$. Since $\mathcal{C}(c)(\lambda_1) = \mathit{true}$ and $\lambda_1 \sqsubseteq \lambda_2$, we can conclude from (1) that $\mathcal{C}(c)(\lambda_2) = \mathit{true}$ and, hence, $(c \to P, \lambda_2) \xrightarrow{\iota} (P, \lambda_2)$.
- Consider the proof

  $$\frac{(P_1, \lambda_1) \xrightarrow{\alpha} (P_1', \lambda_1')}{(P_1 + P_2, \lambda_1) \xrightarrow{\alpha} (P_1', \lambda_1')}$$

  Assume that

  $$\frac{P_1 \Uparrow L_1 \qquad P_2 \Uparrow L_2 \qquad L_1 \cap L_2 = \emptyset}{P_1 + P_2 \Uparrow L_1 \cup L_2}$$

  By induction, $(P_1, \lambda_2) \xrightarrow{\alpha} (P_1', \lambda_2')$ for some λ_2' such that $\lambda_1' \sqsubseteq \lambda_2'$. Then $(P_1 + P_2, \lambda_2) \xrightarrow{\alpha} (P_1', \lambda_2'\{L_2 \mapsto \mathit{false}\})$. Since $(P_1 + P_2, \lambda_1)$ is valid and $P_1 + P_2 \Uparrow L_1 \cup L_2$, we have that $\lambda_1(\ell) = \bot$ for all $\ell \in L_2$. Assume $P_1' \Uparrow L_1'$. According to Proposition 11, $\lambda_1 \restriction (\mathbb{L} \setminus L_1) \cup L_1' = \lambda_1' \restriction (\mathbb{L} \setminus L_1) \cup L_1'$. Since $L_1 \cap L_2 = \emptyset$, we can conclude that $\lambda_1'(\ell) = \bot$ for all $\ell \in L_2$. Therefore, $\lambda_1' \sqsubseteq \lambda_2'\{L_2 \mapsto \mathit{false}\}$.
- Consider the proof

  $$\frac{(P_1, \lambda_1) \xrightarrow{\alpha} (P_1', \lambda_1')}{(P_1 \parallel P_2, \lambda_1) \xrightarrow{\alpha} (P_1' \parallel P_2, \lambda_1')}$$

  By induction, $(P_1, \lambda_2) \xrightarrow{\alpha} (P_1', \lambda_2')$ for some λ_2' such that $\lambda_1' \sqsubseteq \lambda_2'$. Hence, $(P_1 \parallel P_2, \lambda_2) \xrightarrow{\alpha} (P_1' \parallel P_2, \lambda_2')$. □

Lemma 15 *If* $(P, \lambda_2) \xrightarrow{\alpha} (P', \lambda_2')$ *and* $\lambda_1 \sqsubseteq \lambda_2$ *then* $(P, \lambda_1) \xrightarrow{\alpha} (P', \lambda_1')$ *for some* λ_1' *such that* $\lambda_1' \sqsubseteq \lambda_2'$.

Proof We prove this lemma by transition induction. Most cases are the same as in the proof of Lemma 14.

- Consider the transition $(c \to P, \lambda_2) \xrightarrow{\iota} (P, \lambda_2)$. Since $\mathcal{C}(c)(\lambda_2) = \mathit{true}$ and $\lambda_1 \sqsubseteq \lambda_2$, we can conclude from (2) that $\mathcal{C}(c)(\lambda_1) = \mathit{true}$ and, hence, $(c \to P, \lambda_1) \xrightarrow{\iota} (P, \lambda_1)$.
- Consider the proof

  $$\frac{(P_1, \lambda_2) \xrightarrow{\alpha} (P_1', \lambda_2') \qquad P_2 \Uparrow L_2}{(P_1 + P_2, \lambda_2) \xrightarrow{\alpha} (P_1', \lambda_2'\{L_2 \mapsto \mathit{false}\})}$$

  Assume that

  $$\frac{P_1 \Uparrow L_1 \qquad P_2 \Uparrow L_2 \qquad L_1 \cap L_2 = \emptyset}{P_1 + P_2 \Uparrow L_1 \cup L_2}$$

  By induction, $(P_1, \lambda_1) \xrightarrow{\alpha} (P_1', \lambda_1')$ for some λ_1' such that $\lambda_1' \sqsubseteq \lambda_2'$. Hence, $(P_1 + P_2, \lambda_1) \xrightarrow{\alpha} (P_1', \lambda_1')$. Since $(P_1 + P_2, \lambda_1)$ is valid and $P_1 + P_2 \Uparrow L_1 \cup L_2$, we have that $\lambda_1(\ell) = \bot$ for all $\ell \in L_2$. Assume $P_1' \Uparrow L_1'$. According to Proposition 6, $\lambda_1 \restriction (\mathbb{L} \setminus L_1) \cup L_1' = \lambda_1' \restriction (\mathbb{L} \setminus L_1) \cup L_1'$. Since $L_1 \cap L_2 = \emptyset$, we can conclude that $\lambda_1'(\ell) = \bot$ for all $\ell \in L_2$. Therefore, $\lambda_1' \sqsubseteq \lambda_2'\{L_2 \mapsto \mathit{false}\}$. □

Lemma 16 *If* $(P, \lambda_2) \xrightarrow{\tau} (P', \lambda_2')$ *and* $\lambda_1 \sqsubseteq \lambda_2$ *then* $(P, \lambda_1) \sim (P', \lambda_1)$.

Proof We prove this lemma by transition induction.

- Consider the transition $(c \to P, \lambda_2) \xrightarrow{\tau} (0, \lambda_2\{L \mapsto \mathit{false}\})$. Then $\mathcal{C}(c)(\lambda_2) = \mathit{false}$. Since $\lambda_1 \sqsubseteq \lambda_2$ we can conclude from (1) that $\mathcal{C}(c)(\lambda_2) \neq \mathit{true}$. Hence, $(c \to P, \lambda_1) \sim (0, \lambda_1)$.
- Consider the proof

$$\frac{(P_1, \lambda_2) \xrightarrow{\tau} (P_1', \lambda_2')}{(P_1 \parallel P_2, \lambda_2) \xrightarrow{\tau} (P_1' \parallel P_2, \lambda_2')}$$

  By induction, $(P_1, \lambda_1) \sim (P_1', \lambda_1)$. According to Proposition 8, $(P_1 \parallel P_2, \lambda_1) \sim (P_1' \parallel P_2, \lambda_1)$. □

In the proof of 3, we also exploit

Proposition 17 *If* $(P, \lambda) \xrightarrow{\tau} (P', \lambda')$ *then* $\lambda \sqsubseteq \lambda'$.

Proof We prove this proposition by transition induction.

- Consider the transition $(c \to P, \lambda) \xrightarrow{\tau} (0, \lambda\{L \mapsto \mathit{false}\})$ where $P \Uparrow L$. Since $(c \to P, \lambda)$ is valid, $\lambda(\ell) = \bot$ for all $\ell \in L$. Hence, $\lambda \sqsubseteq \lambda\{L \mapsto \mathit{false}\}$.
- Consider the proof

$$\frac{(P_1, \lambda) \xrightarrow{\tau} (P_1', \lambda')}{(P_1 \parallel P_2, \lambda) \xrightarrow{\tau} (P_1' \parallel P_2, \lambda')}$$

  By induction, $\lambda \sqsubseteq \lambda'$. □

Now we are ready to prove

Theorem 18 $\mathcal{E}$ *is an expansion up to* $\precsim$.

Proof Obviously, the identity relation $=$ is a bisimulation and an expansion and, hence, $= \subseteq \sim$ and $= \subseteq \precsim$. Consequently, 1 and 2 follow from Lemma 14 and 15, respectively. Next, we prove 3. As shown in [SM92, Theorem 3.3], $\sim \subseteq \precsim$. Assume that $(P, \lambda_2) \xrightarrow{\tau} (P', \lambda_2')$ and $\lambda_1 \sqsubseteq \lambda_2$. Then,

$$\begin{array}{lll} (P, \lambda_1) & \precsim & (P', \lambda_1) \quad [\sim \subseteq \precsim \text{ and Lemma 16}] \\ & \mathcal{E} & (P', \lambda_2') \quad [\lambda_1 \sqsubseteq \lambda_2 \text{ and } \lambda_2 \sqsubseteq \lambda_2' \text{ by Proposition 17}] \\ & \precsim & (P', \lambda_2') \quad [= \subseteq \precsim] \end{array}$$

which proves 3. □

This brings us to our first main result: $(P, \lambda_\bot)$ and $(P, \lambda_\bot)$ are weak bisimilar.

Corollary 19 $(P, \lambda_\bot) \approx (P, \lambda_\bot)$.

Proof Since $\mathcal{E}$ is an expansion up to $\precsim$, we know that $\mathcal{E} \subseteq \precsim$ according to [SM92, Theorem 3.5]. As shown in [SM92, Theorem 3.3], we have that $\precsim \subset \approx$. Hence, $\mathcal{E} \subset \approx$. Since $(P, \lambda_\bot)\ \mathcal{E}\ (P, \lambda_\bot)$, we can conclude that $(P, \lambda_\bot) \approx (P, \lambda_\bot)$. □

7 SEF is Necessary

Next, we prove that if the evaluation function $\mathcal{C}$ does not satisfy the SEF condition (2), then DPE may have side effects. We show this by constructing a BPE-process P such that $(P, \lambda_\perp)$ and $(P, \lambda_\perp)$ behave differently. In this case, the weaker the behaviour equivalence, the stronger the result. We prove it for weak trace equivalence, which is one of the weakest behavioural equivalences.

Assume that the evaluation function $\mathcal{C}$ does not satisfy the SEF condition (2). That is,

$$\mathcal{C}(c)(\lambda_2) = \mathit{true} \text{ and } \lambda_1 \sqsubseteq \lambda_2 \text{ and } \mathcal{C}(c)(\lambda_1) \neq \mathit{true}. \tag{3}$$

for some $\lambda_1, \lambda_2 \in \Lambda$ and $c \in \mathbb{C}$. Given this assumption, we can prove our second main result.

Theorem 20 *There exists a process P such that $(P, \lambda_\perp)$ and $(P, \lambda_\perp)$ are not weak trace equivalent.*

Proof Assume that $\lambda_1 \sqsubseteq \lambda_2$. Without loss of generality, we can assume that $\lambda_1(\ell_i) \neq \perp$ and $\lambda_2(\ell_i) \neq \perp$ and for all $1 \leq i \leq n$ and $\lambda_1(\ell_{n+i}) = \perp$ and $\lambda_2(\ell_{n+i}) = \mathit{false}$ for all $1 \leq i \leq m$ and $\lambda_1(\ell) = \perp$ and $\lambda_2(\ell) = \perp$ for all other $\ell \in \mathbb{L}$. Now consider the processes

$$\begin{aligned} P &= \ell_1 \uparrow \lambda_1(\ell_1).\ell_2 \uparrow \lambda_1(\ell_2) \ldots \ell_n \uparrow \lambda_1(\ell_n).Q \\ Q &= (a.c \to a.0) + (a.\ell_{n+1} \uparrow \mathit{true}.\ell_{n+2} \uparrow \mathit{true} \ldots \ell_{n+m} \uparrow \mathit{true}.0) \end{aligned}$$

Since (3), we have that

$$(P, \lambda_\perp) \xrightarrow{\iota^n} (Q, \lambda_1) \xrightarrow{a} (c \to a.0, \lambda_1) \not\to$$

and

$$(P, \lambda_\perp) \xrightarrow{\iota^n} (Q, \lambda_1) \xrightarrow{a} (c \to a.0, \lambda_2) \xrightarrow{\iota} (a.0, \lambda_2) \xrightarrow{a} (0, \lambda_2) \not\to$$

Therefore, $(P, \lambda_\perp)$ and $(P, \lambda_\perp)$ are not weak trace equivalent. □

8 A Modification of Dead-Path-Elimination

Below, we model the BPE-calculus with the modification of DPE as proposed in the introduction. Furthermore, we adapt the SEF condition to this setting. This adapted condition is necessary and sufficient for the modification of DPE to be free of side effects.

The key difference between DPE and its proposed modification is the value $\top$. This value is assigned to a link when its source activity is "garbage collected" by modified DPE. It captures that the link has been defined (the opposite of being undefined), but its value is different from *true* and *false*. Since a link can have the value $\top$, we redefine the set of links statuses as follows.

Definition 21 *The set Λ of links statuses is defined by*

$$\Lambda = \mathbb{L} \to \{\mathit{true}, \mathit{false}, \perp, \top\}.$$

As before, we assume that the evaluation function $\mathcal{C} : \mathbb{C} \times \Lambda \to \{\mathit{true}, \mathit{false}, \perp, \top\}$ satisfies (1). Note that the value of a join condition can be $\top$.

This time, we denote a state of the labelled transition system as (P, λ), where P is a BPE-process and λ is an (extended) links status. The labels are the same as before. The transitions are defined in the following definition.

Definition 22 *The labelled transition relation $\to \subseteq \mathbb{P} \times \Lambda \times (\mathbb{A} \cup \{\iota, \tau\}) \times \mathbb{P} \times \Lambda$ is defined by* (PREF), (OUT), (JOIN_t), (FLOW_ℓ), (FLOW_r), (FLOW_ℓ^τ), (FLOW_r^τ), *and*

$$(\text{JOIN}_f)\ \frac{\mathcal{C}(c)(\lambda) = \mathit{false} \qquad P \Uparrow L}{(c \to P, \lambda) \xrightarrow{\tau} (0, \lambda\{L \mapsto \top\})}$$

$$(\text{JOIN}_\top)\ \frac{\mathcal{C}(c)(\lambda) = \top \qquad P \Uparrow L}{(c \to P, \lambda) \xrightarrow{\tau} (0, \lambda\{L \mapsto \top\})}$$

$$(\text{PICK}^\delta_\ell)\ \frac{(P_1, \lambda) \xrightarrow{\alpha} (P_1', \lambda') \qquad P_2 \Uparrow L_2}{(P_1 + P_2, \lambda) \xrightarrow{\alpha} (P_1', \lambda'\{L_2 \mapsto \top\})}$$

$$(\text{PICK}^\delta_r)\ \frac{(P_2, \lambda) \xrightarrow{\alpha} (P_2', \lambda') \qquad P_1 \Uparrow L_1}{(P_1 + P_2, \lambda) \xrightarrow{\alpha} (P_2', \lambda'\{L_1 \mapsto \top\})}$$

Note that the only difference between the rules $(\text{PICK}^\delta_\ell)$, (PICK^δ_r) and (JOIN_f) in the above definition and Definition 9 is the use of $\top$ instead of *false*. The rule $(\text{JOIN}_\top)$ also uses $\top$ rather than *false*. If the value of the join condition is $\top$, the process is "garbage collected." Proposition 5, 6 and 7 also hold for the modified labelled transition system.

To adapt the SEF condition to the current setting, we only have to extend the relation $\sqsubseteq$ so that the new value $\top$ is taken into consideration as well. The value $\top$ takes over the role of *false*.

Definition 23 *The relation* $\sqsubseteq \subseteq \{\mathit{true}, \mathit{false}, \bot, \top\}^2$ *is the smallest reflexive relation containing* $\bot \sqsubseteq \top$.

SEF can be shown to be necessary and sufficient for the modification of DPE to be free of side effects. The proofs of Section 6 and 7 can be adapted in a straightforward manner to the current setting.

9 Evaluation of Join Conditions

Next, we focus on the evaluation of join conditions. First, we define the syntax of join conditions. Second, we describe a number of different evaluation functions of join conditions and check if these satisfy SEF.

Without loss of expressiveness, we can simplify the syntax of the join conditions of BPEL4WS as follows.

Definition 24 *The set* $\mathbb{C}$ *of join conditions is defined by*

$$c ::= \mathit{true} \mid \ell \mid \neg c \mid c \wedge c \mid c \vee c$$

The evaluation function $\mathcal{C}$ is defined by

$$\begin{array}{rcl} \mathcal{C}(\mathit{true})(\lambda) & = & \mathit{true} \\ \mathcal{C}(\ell)(\lambda) & = & \lambda(\ell) \\ \mathcal{C}(\neg c)(\lambda) & = & \neg\mathcal{C}(c)(\lambda) \\ \mathcal{C}(c_1 \wedge c_2)(\lambda) & = & \mathcal{C}(c_1)(\lambda) \wedge \mathcal{C}(c_2)(\lambda) \\ \mathcal{C}(c_1 \vee c_2)(\lambda) & = & \mathcal{C}(c_1)(\lambda) \vee \mathcal{C}(c_2)(\lambda) \end{array}$$

Below, we present different interpretations of the operators $\neg$, $\wedge$ and $\vee$. First, we consider the case where each link can only have three values (*true*, *false* and $\bot$). In this case, we obtain three valued logics. Next, we also consider $\top$ as a possible value for links, giving rise to four valued logics.

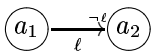
9.1 Three valued logics

In this section, we assume that each link is either *true*, *false* or $\bot$. Thus, the value of a join condition is either *true*, *false* or $\bot$. Obviously, the operator $\neg$ is interpreted as follows.

	true	*false*	$\bot$
$\neg$	*false*	*true*	$\bot$

Now, consider the join condition $\neg\ell$. Assume that the value of the link ℓ is undefined in the model without DPE and that it has the value *false* in the model with DPE, that is, $\lambda_1(\ell) = \bot$ and $\lambda_2(\ell) = \mathit{false}$. Obviously,

$\lambda_1 \sqsubseteq \lambda_2$. In the model without DPE, $\mathcal{C}(\neg\ell)(\lambda_1) = \bot$ but in the model with DPE we have that $\mathcal{C}(\neg\ell)(\lambda_2) = \mathit{true}$. Hence, SEF is not satisfied. The fact that this evaluation function $\mathcal{C}$ that does not satisfy SEF is caused by the presence of negative occurrences of links in the join conditions, as we already discussed in the introduction. However, note that not every negative occurrence is problematic. For example, in



the link ℓ has a negative occurrence in the join condition $\neg\ell$, but DPE does not get triggered and, hence, no side effects arise. We will come back to this in the concluding section of this paper.

For the rest of this section, we restrict ourselves to join conditions without any negative occurrences of links by removing $\neg c$ from Definition 24.

There are two natural ways to interpret the operators $\wedge$ and $\vee$. In the first interpretation, a join condition is only evaluated if all links that are part of the join condition are defined. That is, the operators $\wedge$ and $\vee$ are strict with respect to $\bot$. Hence, the operators $\wedge$ and $\vee$ are interpreted as follows.

$\wedge$	*true*	*false*	$\bot$
true	*true*	*false*	$\bot$
false	*false*	*false*	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$

$\vee$	*true*	*false*	$\bot$
true	*true*	*true*	$\bot$
false	*true*	*false*	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$

We denote the corresponding evaluation function by $\mathcal{C}_1$. This is the way join condition evaluation is defined in BPEL4WS [ACD+03]. One can easily verify that the evaluation function $\mathcal{C}_1$ satisfies (1). Let us consider the join condition $\ell \vee \mathit{true}$. Assume that the link ℓ has value $\bot$ in the model without DPE and value *false* in the model with DPE, that is, $\lambda_1(\ell) = \bot$ and $\lambda_2(\ell) = \mathit{false}$. In the model without DPE, $\mathcal{C}_1(\ell \vee \mathit{true})(\lambda_1) = \bot$ but in the model with DPE we have that $\mathcal{C}_1(\ell \vee \mathit{true})(\lambda_2) = \mathit{true}$. Hence, SEF is not satisfied.

In the second interpretation, a join condition is evaluated as soon as its value is known. This approach is supported in, for example, the web services flow language (WSFL) [Ley01]. In this case, the operators $\wedge$ and $\vee$ are interpreted as follows.

$\wedge$	*true*	*false*	$\bot$
true	*true*	*false*	$\bot$
false	*false*	*false*	*false*
$\bot$	$\bot$	*false*	$\bot$

$\vee$	*true*	*false*	$\bot$
true	*true*	*true*	*true*
false	*true*	*false*	$\bot$
$\bot$	*true*	$\bot$	$\bot$

The induced evaluation function $\mathcal{C}_2$ satisfies (1) and SEF. Hence, if we disallow negative occurrences of links in join conditions and evaluate a join condition as soon as its value is known, then DPE is free of side effects.

9.2 Four valued logics

In the proposed modification of DPE, a link can have four different values: *true*, *false*, $\bot$ and $\top$. Consequently, the value of a join condition can either be *true*, *false*, $\bot$ or $\top$. In this case,the operator $\neg$ is interpreted as follows.

	true	*false*	$\bot$	$\top$
$\neg$	*false*	*true*	$\bot$	$\top$

Next, we discuss how we can extend the two scenarios described above. In the first scenario, we only evaluate a join condition once all its links are defined. That is, the operators are strict with respect to $\bot$. Hence,

$\wedge$	*true*	*false*	$\bot$	$\top$
true	*true*	*false*	$\bot$	
false	*false*	*false*	$\bot$	
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
$\top$			$\bot$	

$\vee$	*true*	*false*	$\bot$	$\top$
true	*true*	*true*	$\bot$	
false	*true*	*false*	$\bot$	
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
$\top$			$\bot$	

Next, we discuss how the above tables can be completed such that (1) and SEF are satisfied. Let us focus on the most interesting entry of the table defining disjunction. Again we consider the join condition $\ell \vee \mathit{true}$.

Assume that the value of the link ℓ is $\top$, that is, the source activity of link ℓ is "garbage collected" by DPE. The most obvious choice seems to define $\mathit{true} \vee \top$ as true. However, in this way we obtain an evaluation function $\mathcal{C}_3$ that does not satisfy SEF. Assume that the link ℓ has value $\bot$ in the model without DPE and that it has the value $\top$ in the model with modified DPE. That is, $\lambda_1(\ell) = \bot$ and $\lambda_2(\ell) = \top$. Obviously, $\lambda_1 \sqsubseteq \lambda_2$. Since $\mathcal{C}_3(\ell \vee \mathit{true})(\lambda_1) = \bot$ and $\mathcal{C}_3(\ell \vee \mathit{true})(\lambda_2) = \mathit{true}$, we can conclude that SEF is not satisfied.

As an alternative choice, we could set the value of the join condition $\ell \vee \mathit{true}$ to $\top$ (and as a consequence the corresponding activity will be "garbage collected") in case the value of the link ℓ is $\top$ (since its source has been "garbage collected"). This choice gives rise to the following table.

$\vee$	*true*	*false*	$\bot$	$\top$
true	*true*	*true*	$\bot$	$\top$
false	*true*	*false*	$\bot$	$\top$
$\bot$	$\bot$	$\bot$	$\bot$	$\bot$
$\top$	$\top$	$\top$	$\bot$	$\top$

The induced evaluation function $\mathcal{C}_4$ satisfies (1) and SEF. To prove that $\mathcal{C}_4$ satisfies SEF, we can show that

if $\mathcal{C}_4(c)(\lambda_2) = \mathit{true}$ and $\lambda_1 \sqsubseteq \lambda_2$ then $\mathcal{C}_4(c)(\lambda_1) = \mathit{true}$ and
if $\mathcal{C}_4(c)(\lambda_2) = \mathit{false}$ and $\lambda_1 \sqsubseteq \lambda_2$ then $\mathcal{C}_4(c)(\lambda_1) = \mathit{false}$

by structural induction on c.

Note that a join condition evaluates to $\top$ if all its incoming links are defined and at least one of them has the value $\top$. In this case, once an activity is "garbage collected", all other activities that can be reached from the activity along links will never be executed and, hence, will eventually be "garbage collected" as well.

In the second scenario, we evaluate a join condition as soon as we know its value. Therefore,

$\wedge$	*true*	*false*	$\bot$	$\top$
true	*true*	*false*	$\bot$	
false	*false*	*false*	*false*	*false*
$\bot$	$\bot$	*false*	$\bot$	
$\top$		*false*		

$\vee$	*true*	*false*	$\bot$	$\top$
true	*true*	*true*	*true*	*true*
false	*true*	*false*	$\bot$	
$\bot$	*true*	$\bot$	$\bot$	
$\top$	*true*			

Again, we want to complete the above tables in such a way that (1) and SEF are satisfied. This can be done as follows.

$\vee$	*true*	*false*	$\bot$	$\top$
true	*true*	*true*	*true*	*true*
false	*true*	*false*	$\bot$	$\top$
$\bot$	*true*	$\bot$	$\bot$	$\bot$
$\top$	*true*	$\top$	$\bot$	$\top$

Also in this case, the resulting evaluation function $\mathcal{C}_5$ satisfies (1) and SEF.

10 Conclusion

Let us first summarise our main contributions. We observed that DPE may give rise to unintended side effects. We presented a calculus, named the BPE-calculus, that contains those constructs of BPEL4WS that are key to DPE. We modelled our BPE-calculus, both in the absence and in the presence of (our proposed modification of) DPE. The model without DPE and the model with modified DPE are new. The other model was already presented in [KB04]. We formulated a condition that exactly captures when (modified) DPE has side effects. We proved that this condition is sufficient and necessary for DPE (and its modification) to be free of side effects. We are confident that these results scale up from the BPE-calculus to BPEL4WS.

The model without DPE is simpler than the model with DPE (compare Definition 4 with Definition 9). However, since DPE may have side effects in BPEL4WS, one cannot rely on the simpler model (the one without DPE). From time to time, these side effects may be introduced unintentionally. Therefore, we believe

that it is important that one is aware of DPE and its possible side effects. In the previous section, we have presented three ways to avoid DPE having side effects. As a consequence, in these cases one can rely on the simpler model without DPE.

There are three ways to avoid DPE having side effects. First of all, we could disallow negative occurrences of links in join conditions and evaluate a join condition once its value is known. Secondly, we could consider our proposed modification of DPE and evaluate a join conditions once its value is known. Thirdly, we could consider our proposed modification of DPE and evaluate a join condition once all its links are defined. In these three cases, DPE does not have side effects. As a consequence, in these cases one can rely on the simpler model without DPE.

10.1 Related Work

In our study, we used labelled transition systems to model the BPE-calculus. Petri nets provide an alternative approach to model business processes. For an overview of modelling business processes by means of Petri nets, we refer the reader to, for example, [AH02a]. We believe that there are at least two major advantages of labelled transition systems over Petri nets to address the question whether DPE has side effects. First of all, in our proofs we fruitfully exploited induction on the structure of BPE-processes and transition induction. Such proof techniques are less applicable to Petri nets. Secondly, as also pointed out in [AH02b], advanced synchronisation patterns like DPE cannot easily be captured by means of Petri nets (although it can be done, as shown in [Mar03]).

A paper that also considers DPE is [CKLW03]. The focus of that paper is exception handling in BPEL4WS. Using a rather informal model, it is shown that DPE can be seen as a particular type of exception handling.

In [FGV04], distributed abstract state machines are used to model BPEL4WS processes. That model abstracts away from the details of join conditions and, hence, does not consider DPE. In [DBLL04], a logical model for abstract BPEL4WS processes is sketched. Not many details are provided. This makes it very difficult to determine whether this model is appropriate for studying DPE.

10.2 Future Work

The different evaluation strategies seem to have an impact on the expressiveness. For example, consider the behaviour of the BPE-process $a_1.\ell_1 \uparrow \mathit{true}.0 \parallel a_2.\ell_2 \uparrow \mathit{true}.0 \parallel (\ell_1 \vee \ell_2) \rightarrow a_3.0$ when a join condition is evaluated as soon as its value is known. We conjecture that this behaviour cannot be expressed by any BPE-process if join conditions are evaluated once all their links are defined. Studying the expressiveness of these two evaluation strategies is one of the topics for further research.

We are interested to see how frequent links are used in BPEL4WS programs and what proportion of those links are used negatively in join conditions. Of the limited number of BPEL4WS programs that we have considered, most contain many links (for example, one BPEL4WS program contained 96 basic activities and 136 links) and only a few negative occurrences of links.

We implemented a tool that provides the programmer with a warning if a negative occurrence of a link may give rise to side effects. The details can be found in [HFBO05]. Although the tool detects all negative occurrences of links that cause side effects, it also finds negative occurrences of links that do not. The tool has to be improved to reduce the number of false positives.

10.3 Acknowledgements

The authors would like to thank Jon Bennett, Bill O'Farrell, Kien Huynh, Stan Li, Marin Litoiu, Dieter Roller and Eric Ruppert for fruitful discussions and the referees for helpful comments.

References

[ACD+03] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business process execution language for

web services, version 1.1. Available at www.ibm.com/developerworks/library/ws-bpel.pdf, May 2003.

[AH92] S. Arun-Kumar and M. Hennessy. An efficient preorder for processes. *Acta Informatica*, 29(8):737–760, December 1992.

[AH02a] W.M.P. van der Aalst and K.M. van Hee. *Workflow Management: Models, Methods, and Systems*. The MIT Press, Cambridge, Massachusetts, 2002.

[AH02b] W.M.P. van der Aalst and A.H.M. ter Hofstede. Workflow patterns: on the expressive power of (Petri-net-based) workflow languages. In K. Jensen, editor, *Proceedings of the Fourth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools*, volume 560 of *DAIMI PB series*, pages 1–20, Aarhus, August 2002. University of Aarhus.

[BLZ03] L. Bocchi, C. Laneve, and G. Zavattaro. A calculus for long running transactions. In E. Najm, U. Nestmann, and P. Stevens, editors, *Proceedings of the 6th IFIP WG 6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems*, volume 2884 of *Lecture Notes in Computer Science*, pages 124–138, Paris, November 2003. Springer-Verlag.

[CGV+02] M. Chessell, C. Griffin, D. Vines, M. Butler, C. Ferreira, and P. Henderson. Extending the concept of transaction compensation. *IBM Systems Journal*, 41(4):743–758, 2002.

[CKLW03] F. Curbera, R. Khalaf, F. Leymann, and S. Weerawarana. Exception handling in the BPEL4WS language. In W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske, editors, *Proceedings of the International Conference on Business Process Management*, volume 2678 of *Lecture Notes in Computer Science*, pages 276–290, Eindhoven, June 2003. Springer-Verlag.

[DBLL04] Z. Duan, A. Bernstein, P. Lewis, and S. Lu. Semantics based verification and synthesis of BPEL4WS abstract processes. In *Proceedings of the IEEE International Conference on Web Services*, pages 734–737, San Diego, June 2004. IEEE.

[FGV04] R. Farahbod, U. Glässer, and M. Vajihollahi. Abstract operational semantics of the business process execution language for web services. Report SFU-CMPT-TR-2004-03, Simon Fraser University, Burnaby, British Columbia, April 2004. Available at fas.sfu.ca/pub/cs/techreports/2004/CMPT2004-03.pdf.

[HFBO05] K. Huynh, J. Fung, F. van Breugel, and B. O'Farrell. Analysis through reflection, walking the EMF model of BPEL4WS. Report CS-2005-05, York University, Toronto, April 2005. Available at www.cs.yorku.ca/techreports/2005.

[KB04] M. Koshkina and F. van Breugel. Modelling and verifying web service orchestration by means of the concurrency workbench. *ACM SIGSOFT Software Engineering Notes*, 29(5), September 2004.

[Kos03] M. Koshkina. Verification of business processes for web services. Master's thesis, York University, Toronto, December 2003. Available at www.cs.yorku.ca/~franck/students.

[KS03] J. Koehler and B. Srivastava. Web service composition: current solutions and open problems. In *Proceedings of the Workshop on Planning for Web Services*, pages 28–35, Trento, June 2003. Available at icaps03.itc.it/satellite_events/workshops/workshops_5.htm.

[LA94] F. Leymann and W. Altenhuber. Managing business processes as an information resource. *IBM Systems Journal*, 33(2):326–348, 1994.

[Ley01] F. Leymann. Web services flow language. Available at www.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, May 2001.

[Mar03] A. Martens. *Verteilte Geschäftsprozesse — Modellierung und Verifikation mit Hilfe von Web Services*. WiKu, Berlin, 2003.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall International, Englewood Cliffs, New Jersey, 1989.

[Mil99] R. Milner. *Communicating and Mobile Systems: the π-Calculus.* Cambridge University Press, Cambridge, 1999.

[Plo04] G.D. Plotkin. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming*, 60/61:17–139, July/December 2004.

[Que03] *Queue*, 1(1), March 2003.

[Rol03] D. Roller. Personal communication. June 2003.

[SM92] D. Sangiorgi and R. Milner. The problem of "weak bisimulation up to". In R. Cleaveland, editor, *Proceedings of the 3rd International Conference on Concurrency Theory*, volume 630 of *Lecture Notes in Computer Science*, pages 32–46, Stony Brook, August 1992. Springer-Verlag.

[Win93] G. Winskel. *The Formal Semantics of Programming Languages.* The MIT Press, Cambridge, Massachusetts, 1993.