Does Dead-Path-Elimination have Side Effects?

Franck van Breugel and Mariya Koshkina

Technical Report CS-2003-04

April 2003

Department of Computer Science

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

Does Dead-Path-Elimination have Side Effects?*

Franck van Breugel and Mariya Koshkina

York University, Department of Computer Science
4700 Keele Street, Toronto, Canada M3J 1P3

franck@cs.yorku.ca and mkosh@cs.yorku.ca

April 4, 2003

Abstract

Dead-path-elimination (DPE) can be viewed as a garbage collection strategy used in 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 two models for the BPE-calculus: one without DPE and one with DPE. We formulate a condition, named SEF, and show that it is sufficient and necessary for DPE being free of 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 both models. Secondly, if the condition SEF is not satisfied, then we can construct a BPE-process that behaves differently in the two models.

Introduction

Recently, BEA, IBM and Microsoft introduced the business process execution language for web services (BPEL4WS). This language has been designed to specify interactions between various web services. For an introduction to web services, we refer the reader to, for example, [BGE+02, Que03]. The initial public draft release of the BPEL4WS specification can be found in [CGK+02]. 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"
  inputContainer="offer" outputContainer="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.

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

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

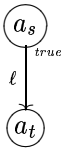
the activities `buy` and `sell`, whose behaviour has been left unspecified to simplify the example, are concurrent. The pick construct allows for selective communication. Consider, for example,

*This research is supported by IBM.

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

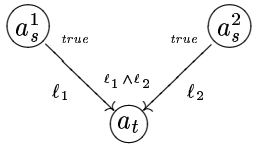
On the one hand, if a message from `consumer` is received then the activity `sell` is executed. In that case, the `buy` activity will not be performed. On the other hand, the receipt of a message from `producer` triggers the execution of the `buy` activity and discards the `sell` activity. In the case 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 the π-calculus [Mil99] and is also reminiscent to the `choose` construct of Concurrent ML [Rep99].

Synchronization 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 to 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



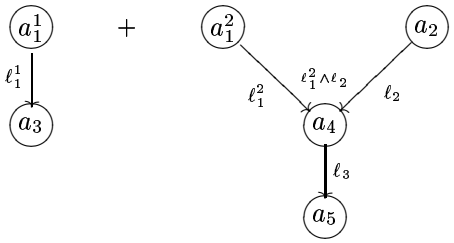
to depict that link ℓ has source a_s and target a_t and transition condition *true*.

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, an activity can 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 $\ell_1 \wedge \ell_2$. In the above example, activity a_t can only start after activities a_s^1 and a_s^2 have finished.

Let us consider the following activities and links.



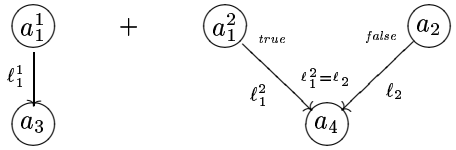
In the above picture, we use + to depict the pick construct. Note that the choice between the activities a_1^1 and a_1^2 determines which of the activities a_3, a_4 and a_5 are performed. For example, if a_1^1 is chosen then a_3 is executed. In that case neither a_4 nor a_5 can ever occur. As a consequence, the activities a_4 and a_5 could be garbage collected. This can be achieved 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.

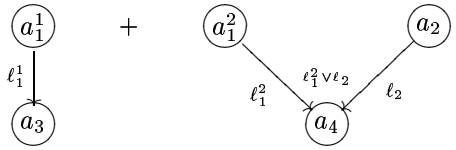
This garbage collection scheme is named dead-path-elimination (DPE) in [CGK+02]. Let us briefly return to the above example. Assume that activity a_1^1 is chosen. Then, as a result of DPE, the value of the link ℓ_1^2 becomes false. When activity a_2 terminates, the link ℓ_2 gets the default value true. 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.

Now let us consider another example.



At first sight, one may be tempted to conclude that activity a_4 will never be executed. Without DPE this is indeed the case. However, DPE may trigger the execution of activity a_4 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 is false as well, the join condition $\ell_1^2 = \ell_2$ evaluates to true. Hence, activity a_4 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 accidentally.

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_4 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_4 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_4 is executed in this case. This is another example of a process that behaves differently with and without DPE.

The above examples show that DPE can have side effects. The formulation of a condition and the proofs that this condition is sufficient and necessary for DPE to be free of side effects are the main results of this paper. The condition is named SEF for side effect free. SEF 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. As we will see, this evaluation function of BPEL4WS does not satisfy SEF. In the concluding section of this paper, we will discuss how BPEL4WS may be changed so that DPE becomes free of side effects. In particular, we will outline how this can be accomplished by disallowing negations in join conditions and by making the evaluation of the remaining join conditions lazy. 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 sublanguage 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 so obtained language we call the BPE-calculus, as it is similar in flavour to calculi like, for example, CCS [Mil89] and the π-calculus [Mil99]. In the BPE-calculus, we abstract from many details that are irrelevant for our study. This will simplify matters considerably and will make our objectives feasible (it is very difficult, if not almost impossible, to carry out such a study for BPEL4WS as a whole). For example, the above picture corresponds to the (considerably simplified) BPEL4WS process

```
<flow>
  <links>
    <link name="l11"/>
    <link name="l12"/>
    <link name="l2"/>
  </links>
  <pick>
    <onMessage partner="client" operation="a11" container="request">
      <empty>
        <source linkName="l11" transitionCondition="true"/>
      </empty>
    </onMessage>
    <onMessage partner="client" operation="a12" container="request">
      <empty>
        <source linkName="l12" transitionCondition="true"/>
      </empty>
    </onMessage>
  </pick>
  <invoke partner="provider" operation="a2" inputContainer="query" outputContainer="result">
    <source linkName="l2" transitionCondition="true"/>
  </invoke>
  <invoke partner="provider" operation="a3" inputContainer="request" outputContainer="reply"
    joinCondition="bpws:getLinkStatus('l11')">
      <target linkName="l11"/>
  </invoke>
  <invoke partner="provider" operation="a4" inputContainer="request" outputContainer="reply"
    joinCondition="bpws:getLinkStatus('l11') or bpws:getLinkStatus('l2')">
      <target linkName="l12"/>
      <target linkName="l2"/>
  </invoke>
<flow>
```

In our BPE-calculus, it is expressed as

$$(a_1^1.\ell_1^1 \uparrow true.0 + a_1^2.\ell_1^2 \uparrow true.0) \parallel a_2.\ell_2 \uparrow true.0 \parallel (\ell_1^1 \rightarrow a_3.0) \parallel (\ell_1^2 \vee \ell_2 \rightarrow a_4.0)$$

In order to prove our main result, we introduce two 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, [AFV01]. In this paper, we will follow this approach as well. An alternative approach to model business processes, namely by means of Petri nets [AH02a], is 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.

To show that SEF is a sufficient and necessary condition for DPE being 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 both models. Secondly, if the condition SEF is not satisfied, then we can construct a BPE-process that behaves differently in the two models. In the next two paragraphs, we discuss how we prove these properties.

To capture that states of a labelled transition system—in our case, these are the BPE-processes—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. For an overview, we refer the reader to, for example, [Gla90, Gla93]. To prove the first result, we exploit weak bisimilarity which is one of the strongest weak[1] behavioural equivalences. We show that if the SEF condition is satisfied, then a BPE-process in the labelled transition system without DPE is weak bisimilar to the BPE-process in the system with DPE. We actually prove the following even stronger result. If SEF holds, then a BPE-process in the labelled transition system without DPE is expanded by the BPE-process in the system with DPE. Expansion, which we denote by $\precsim$, is a behavioural preorder that was first introduced in [AH92]. Since

[1] A behavioural equivalence that abstracts from internal actions is called weak.

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 SEF condition 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 DPE.

The rest of this paper is organized as follows. In Section 1, we introduce our BPE-calculus. In Section 2 and 3, we present labelled transition systems that model the BPE-calculus without and with DPE. The condition SEF is introduced in Section 4. In Section 5 and 6, we show that SEF is a sufficient and necessary condition. The final section concludes and discusses some related work.

1 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

$$P ::= 0 \mid a.P \mid \ell \uparrow b.P \mid c \to P \mid P + P \mid P \parallel P$$

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. 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.

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, it can 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.

2 A Model without Dead-Path-Elimination

Since we want to compare the BPE-calculus without DPE and the BPE-calculus with DPE, we introduce two models: one without DPE and one with DPE. Both models are defined in terms of a labelled transition system. Below, we present the labelled transition system without DPE. The labelled transition system with DPE is given in the next section.

In both labelled transition systems, 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 $\bot$. Initially, all links are undefined. As we will see, at any point only finitely many links are defined, that is, their value is different from $\bot$.

DEFINITION 3 The set Λ of *links status* is defined by

$$\Lambda = \{\, \lambda \in \mathbb{L} \to \{\mathit{true}, \mathit{false}, \bot\} \mid \lambda(\ell) \neq \bot \text{ for finitely many } \ell \in \mathbb{L} \,\}.$$

⌟

The initial link status $\lambda_\bot$ assigns $\bot$ to each link. 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

$$\text{if } \mathcal{C}(c)(\lambda[{}^{\bot}\!/_{\ell}]) \neq \bot \text{ then } \mathcal{C}(c)(\lambda) = \mathcal{C}(c)(\lambda[{}^{\bot}\!/_{\ell}]) \tag{1}$$

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 link status λ. To distinguish the process P with link status λ in the labelled transition system without DPE and from the process P with link status λ in the system with DPE, we denote the former by $\langle P, \lambda \rangle$ and the latter by $\langle P, \lambda \rangle$.

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

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) $\langle a.P, \lambda \rangle \xrightarrow{a} \langle P, \lambda \rangle$

(OUT) $\langle \ell \uparrow b.P, \lambda \rangle \xrightarrow{\iota} \langle P, \lambda[{}^{b}\!/_{\ell}] \rangle$

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

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

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

⌟

The rules (PREF), (PICK$_\ell$), (PICK$_r$), (FLOW$_\ell$) 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 link 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* $\langle P, \lambda \rangle \xrightarrow{\alpha} \langle P', \lambda' \rangle$ *and* $P \Uparrow L$ *then* $P' \Uparrow L'$ *for some* $L' \subseteq L$.

PROOF We prove this proposition by transition induction.

- Consider the transition $\langle a.P, \lambda\rangle \xrightarrow{a} \langle P, \lambda\rangle$. Assume that $a.P \Uparrow L$. Then $P \Uparrow L$.
- Consider the transition $\langle \ell \uparrow b.P, \lambda\rangle \xrightarrow{\iota} \langle P, \lambda[^b/_\ell]\rangle$. Assume that $\ell \uparrow b.P \Uparrow L$. Then $P \Uparrow L \setminus \{\ell\}$.
- Consider the transition $\langle c \to P, \lambda\rangle \xrightarrow{\iota} \langle P, \lambda\rangle$. Assume that $c \to P \Uparrow L$. Then $P \Uparrow L$.
- Consider the proof

$$\frac{\langle P_1, \lambda\rangle \xrightarrow{\alpha} \langle P_1', \lambda'\rangle}{\langle P_1 + P_2, \lambda\rangle \xrightarrow{\alpha} \langle P_1', \lambda'\rangle}$$

  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

$$\frac{\langle P_1, \lambda\rangle \xrightarrow{\alpha} \langle P_1', \lambda'\rangle}{\langle P_1 \parallel P_2, \lambda\rangle \xrightarrow{\alpha} \langle P_1' \parallel P_2, \lambda'\rangle}$$

  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.

PROPOSITION 6 *If $\langle P, \lambda\rangle \xrightarrow{\alpha} \langle P', \lambda'\rangle$ 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.

- Consider the transition $\langle a.P, \lambda\rangle \xrightarrow{a} \langle P, \lambda\rangle$. Obviously, the proposition is vacuously true in this case.
- Consider the transition $\langle \ell \uparrow b.P, \lambda\rangle \xrightarrow{\iota} \langle P, \lambda[^b/_\ell]\rangle$. 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, $\lambda \restriction \mathbb{L} \setminus \{\ell\} = \lambda[^b/_\ell] \restriction \mathbb{L} \setminus \{\ell\}$.
- Consider the transition $\langle c \to P, \lambda\rangle \xrightarrow{\iota} \langle P, \lambda\rangle$. Obviously, the proposition is vacuously true in this case.
- Consider the proof

$$\frac{\langle P_1, \lambda\rangle \xrightarrow{\alpha} \langle P_1', \lambda'\rangle}{\langle P_1 + P_2, \lambda\rangle \xrightarrow{\alpha} \langle P_1', \lambda'\rangle}$$

  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, $\lambda \restriction (\mathbb{L} \setminus L_1) \cup L_1' = \lambda' \restriction (\mathbb{L} \setminus L_1) \cup L_1'$. Hence, $\lambda \restriction (\mathbb{L} \setminus (L_1 \cup L_2)) \cup L_1' = \lambda' \restriction (\mathbb{L} \setminus (L_1 \cup L_2)) \cup L_1'$.

- Consider the proof

$$\frac{\langle P_1, \lambda\rangle \xrightarrow{\alpha} \langle P_1', \lambda'\rangle}{\langle P_1 \parallel P_2, \lambda\rangle \xrightarrow{\alpha} \langle P_1' \parallel P_2, \lambda'\rangle}$$

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 $\langle P, \lambda\rangle$'s with $P \Uparrow L$ and $\lambda(\ell) = \perp$ for all $\ell \in L$. We will call them valid.

PROPOSITION 7 *If $\langle P, \lambda\rangle \xrightarrow{\alpha} \langle P', \lambda'\rangle$ and $\langle P, \lambda\rangle$ is valid then $\langle P', \lambda'\rangle$ is valid.*

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

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

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 $\ell \in \mathbb{L}$. □

We conclude this section by showing that the behavioural equivalence strong bisimilarity, which we denote by $\sim$, is a congruence with respect to the pick and flow constructs.

PROPOSITION 8 *If $\langle P_1, \lambda\rangle \sim \langle P_1', \lambda'\rangle$ then*

1. *$\langle P_1 + P_2, \lambda\rangle \sim \langle P_1' + P_2, \lambda'\rangle$ and*
2. *$\langle P_1 \parallel P_2, \lambda\rangle \sim \langle P_1' \parallel P_2, \lambda'\rangle$.*

PROOF The proof of this proposition is a straightforward modification of the proof that bisimilarity is a congruence with respect to summation and composition in CCS as shown in, for example, [Mil89, Proposition 4.10]. For example, to prove 2., one shows that $\{\, (\langle P_1 \parallel P_2, \lambda\rangle, \langle P_1' \parallel P_2, \lambda'\rangle) \mid \langle P_1, \lambda\rangle \sim \langle P_1', \lambda'\rangle \,\}$ is a bisimulation. □

3 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 $\langle P, \lambda\rangle$, where P is a process and λ is a link 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

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

$$(\text{JOIN}_f)\ \frac{\mathcal{C}(c)(\lambda) = \mathit{false} \qquad P \Uparrow L}{\langle c \rightarrow P, \lambda\rangle \xrightarrow{\tau} \langle 0, \lambda[{}^{\mathit{false}}/_{L}]\rangle}$$

$$(\text{PICK}_\ell^\delta)\ \frac{\langle P_1, \lambda\rangle \xrightarrow{\alpha} \langle P_1', \lambda'\rangle \qquad P_2 \Uparrow L_2}{\langle P_1 + P_2, \lambda\rangle \xrightarrow{\alpha} \langle P_1', \lambda'[{}^{\mathit{false}}/_{L_2}]\rangle} \qquad (\text{PICK}_r^\delta)\ \frac{\langle P_2, \lambda\rangle \xrightarrow{\alpha} \langle P_2', \lambda'\rangle \qquad P_1 \Uparrow L_1}{\langle P_1 + P_2, \lambda\rangle \xrightarrow{\alpha} \langle P_2', \lambda'[{}^{\mathit{false}}/_{L_1}]\rangle}$$

$$(\textsc{pick}^{\tau}_{\ell})\ \frac{\langle P_1, \lambda\rangle \xrightarrow{\tau} \langle P_1', \lambda'\rangle}{\langle P_1 + P_2, \lambda\rangle \xrightarrow{\tau} \langle P_1' + P_2, \lambda'\rangle} \qquad (\textsc{pick}^{\tau}_{r})\ \frac{\langle P_2, \lambda\rangle \xrightarrow{\tau} \langle P_2', \lambda'\rangle}{\langle P_1 + P_2, \lambda\rangle \xrightarrow{\tau} \langle P_1 + P_2', \lambda'\rangle}$$

$$(\textsc{flow}^{\tau}_{\ell})\ \frac{\langle P_1, \lambda\rangle \xrightarrow{\tau} \langle P_1', \lambda'\rangle}{\langle P_1 \parallel P_2, \lambda\rangle \xrightarrow{\tau} \langle P_1' \parallel P_2, \lambda'\rangle} \qquad (\textsc{flow}^{\tau}_{r})\ \frac{\langle P_2, \lambda\rangle \xrightarrow{\tau} \langle P_2', \lambda'\rangle}{\langle P_1 \parallel P_2, \lambda\rangle \xrightarrow{\tau} \langle P_1 \parallel P_2', \lambda'\rangle}$$

⌟

Note that the rule ($\textsc{pick}_{\ell}$) and ($\textsc{pick}_{r}$) of Definition 4 are replaced with the rules ($\textsc{pick}^{\delta}_{\ell}$) and ($\textsc{pick}^{\delta}_{r}$). 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 ($\textsc{join}_{f}$). 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 processes $P_1 + P_2$ and $P_1 \parallel P_2$. This is captured by the rules ($\textsc{pick}^{\tau}_{\ell}$), ($\textsc{pick}^{\tau}_{r}$), ($\textsc{flow}^{\tau}_{\ell}$) and ($\textsc{flow}^{\tau}_{r}$). Garbage collection should not resolve nondeterminism. For example, we want $\langle \mathit{false} \to 0 + a.0, \lambda\rangle \xrightarrow{\tau} \langle 0 + a.0, \lambda\rangle$ but definately not $\langle \mathit{false} \to 0 + a.0, \lambda\rangle \xrightarrow{\tau} \langle 0, \lambda\rangle$. This is modelled by means of the rules ($\textsc{pick}^{\tau}_{\ell}$) and ($\textsc{pick}^{\tau}_{r}$). These rules are very similar to the rules for CSP's external choice [Plo82].

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

PROPOSITION 10 *If* $\langle P, \lambda\rangle \xrightarrow{\alpha} \langle P', \lambda'\rangle$ *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 $\langle c \to P, \lambda\rangle \xrightarrow{\tau} \langle 0, \lambda[^{\mathit{false}}/_{L}]\rangle$. Since $0 \Uparrow \emptyset$, this case vacuously holds.
- Consider the proof

  $$\frac{\langle P_1, \lambda\rangle \xrightarrow{\tau} \langle P_1', \lambda'\rangle}{\langle P_1 + P_2, \lambda\rangle \xrightarrow{\tau} \langle P_1' + P_2, \lambda'\rangle}$$

  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* $\langle P, \lambda\rangle \xrightarrow{\alpha} \langle P', \lambda'\rangle$ *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. Most cases are the same as in the proof of Proposition 6.

- Consider the transition $\langle c \to P, \lambda\rangle \xrightarrow{\tau} \langle 0, \lambda[^{\mathit{false}}/_{L}]\rangle$ where $P \Uparrow L$. Since $0 \Uparrow \emptyset$, it suffices to show that $\lambda \upharpoonright (\mathbb{L} \setminus L) = \lambda[^{\mathit{false}}/_{L}] \upharpoonright (\mathbb{L} \setminus L)$ which is vacuously true.

- Consider the proof

$$\frac{\langle P_1, \lambda\rangle \xrightarrow{\tau} \langle P_1', \lambda'\rangle}{\langle P_1 + P_2, \lambda\rangle \xrightarrow{\tau} \langle P_1' + P_2, \lambda'\rangle}$$

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 $\langle P, \lambda\rangle \xrightarrow{\alpha} \langle P', \lambda'\rangle$ and $\langle P, \lambda\rangle$ is valid then $\langle P', \lambda'\rangle$ is valid.*

PROOF Similar to the proof of Proposition 7. □

4 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 being free of side effects.

To formulate the condition SEF, we introduce the following preorder on the values of links and extend it to a preorder on link status.

DEFINITION 13 The relation $\sqsubseteq \subseteq \{true, false, \bot\} \times \{true, false, \bot\}$ is defined by

$$\begin{array}{rcl} \bot & \sqsubseteq & false \\ true & \sqsubseteq & true \\ false & \sqsubseteq & false \\ \bot & \sqsubseteq & \bot \end{array}$$

The relation $\sqsubseteq \subseteq \Lambda \times \Lambda$ is defined by

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

⌟

Now we are ready to formulate the condition SEF:

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

In the concluding section of this paper, we discuss this condition in the context of BPEL4WS.

5 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 $\langle P, \lambda_\bot\rangle$ and $\langle P, \lambda_\bot\rangle$ 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 $\langle P, \lambda_\bot\rangle$ expands $\langle P, \lambda_\bot\rangle$. 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 on 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

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

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

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

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

LEMMA 14 *If $\langle P, \lambda_1 \rangle \xrightarrow{\alpha} \langle P', \lambda_1' \rangle$ and $\lambda_1 \sqsubseteq \lambda_2$ then $\langle P, \lambda_2 \rangle \xrightarrow{\alpha} \langle P', \lambda_2' \rangle$ for some λ_2' such that $\lambda_1' \sqsubseteq \lambda_2'$.*

PROOF We prove this lemma by transition induction.

- Consider the transition $\langle a.P, \lambda_1 \rangle \xrightarrow{a} \langle P, \lambda_1 \rangle$. Then $\langle a.P, \lambda_2 \rangle \xrightarrow{a} \langle P, \lambda_2 \rangle$.
- Consider the transition $\langle \ell \uparrow b.P, \lambda_1 \rangle \xrightarrow{\iota} \langle P, \lambda_1[b/\ell] \rangle$. Then $\langle \ell \uparrow b.P, \lambda_2 \rangle \xrightarrow{\iota} \langle P, \lambda_2[b/\ell] \rangle$. Since $\lambda_1 \sqsubseteq \lambda_2$, we also have that $\lambda_1[b/\ell] \sqsubseteq \lambda_2[b/\ell]$.
- Consider the transition $\langle c \to P, \lambda_1 \rangle \xrightarrow{\iota} \langle P, \lambda_1 \rangle$. 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, $\langle c \to P, \lambda_2 \rangle \xrightarrow{\iota} \langle P, \lambda_2 \rangle$.
- Consider the proof

  $$\frac{\langle P_1, \lambda_1 \rangle \xrightarrow{\alpha} \langle P_1', \lambda_1' \rangle}{\langle P_1 + P_2, \lambda_1 \rangle \xrightarrow{\alpha} \langle P_1', \lambda_1' \rangle}$$

  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, $\langle P_1, \lambda_2 \rangle \xrightarrow{\alpha} \langle P_1', \lambda_2' \rangle$ for some λ_2' such that $\lambda_1' \sqsubseteq \lambda_2'$. Then $\langle P_1 + P_2, \lambda_2 \rangle \xrightarrow{\alpha} \langle P_1', \lambda_2'[\mathit{false}/L_2] \rangle$. Since $\langle P_1 + P_2, \lambda_1 \rangle$ 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, $\lambda_1' \sqsubseteq \lambda_2'[\mathit{false}/L_2]$.
- Consider the proof

  $$\frac{\langle P_1, \lambda_1 \rangle \xrightarrow{\alpha} \langle P_1', \lambda_1' \rangle}{\langle P_1 \parallel P_2, \lambda_1 \rangle \xrightarrow{\alpha} \langle P_1' \parallel P_2, \lambda_1' \rangle}$$

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

□

LEMMA 15 *If $\langle P, \lambda_2 \rangle \xrightarrow{\alpha} \langle P', \lambda_2' \rangle$ and $\lambda_1 \sqsubseteq \lambda_2$ then $\langle P, \lambda_1 \rangle \xrightarrow{\alpha} \langle P', \lambda_1' \rangle$ 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 $\langle c \to P, \lambda_2 \rangle \xrightarrow{\iota} \langle P, \lambda_2 \rangle$. 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, $\langle c \to P, \lambda_1 \rangle \xrightarrow{\iota} \langle P, \lambda_1 \rangle$.

- Consider the proof

$$\frac{\langle P_1, \lambda_2\rangle \xrightarrow{\alpha} \langle P_1', \lambda_2'\rangle \qquad P_2 \Uparrow L_2}{\langle P_1 + P_2, \lambda_2\rangle \xrightarrow{\alpha} \langle P_1', \lambda_2'[^{false}/_{L_2}]\rangle}$$

  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, $\langle P_1, \lambda_1\rangle \xrightarrow{\alpha} \langle P_1', \lambda_1'\rangle$ for some λ_1' such that $\lambda_1' \sqsubseteq \lambda_2'$. Hence, $\langle P_1 + P_2, \lambda_1\rangle \xrightarrow{\alpha} \langle P_1', \lambda_1'\rangle$. Since $\langle P_1 + P_2, \lambda_1\rangle$ 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, $\lambda_1' \sqsubseteq \lambda_2'[^{false}/_{L_2}]$.

□

LEMMA 16 *If* $\langle P, \lambda_2\rangle \xrightarrow{\tau} \langle P', \lambda_2'\rangle$ *and* $\lambda_1 \sqsubseteq \lambda_2$ *then* $\langle P, \lambda_1\rangle \sim \langle P', \lambda_1\rangle$.

PROOF We prove this lemma by transition induction.

- Consider the transition $\langle c \to P, \lambda_2\rangle \xrightarrow{\tau} \langle 0, \lambda_2[^{false}/_{L}]\rangle$. 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, $\langle c \to P, \lambda_1\rangle \sim \langle 0, \lambda_1\rangle$.
- Consider the proof

$$\frac{\langle P_1, \lambda_2\rangle \xrightarrow{\tau} \langle P_1', \lambda_2'\rangle}{\langle P_1 + P_2, \lambda_2\rangle \xrightarrow{\tau} \langle P_1' + P_2, \lambda_2'\rangle}$$

  By induction, $\langle P_1, \lambda_1\rangle \sim \langle P_1', \lambda_1\rangle$. According to Proposition 8, $\langle P_1 + P_2, \lambda_1\rangle \sim \langle P_1' + P_2, \lambda_1\rangle$.
- Consider the proof

$$\frac{\langle P_1, \lambda_2\rangle \xrightarrow{\tau} \langle P_1', \lambda_2'\rangle}{\langle P_1 \parallel P_2, \lambda_2\rangle \xrightarrow{\tau} \langle P_1' \parallel P_2, \lambda_2'\rangle}$$

  By induction, $\langle P_1, \lambda_1\rangle \sim \langle P_1', \lambda_1\rangle$. According to Proposition 8, $\langle P_1 \parallel P_2, \lambda_1\rangle \sim \langle P_1' \parallel P_2, \lambda_1\rangle$.

□

In the proof of 3, we also exploit

PROPOSITION 17 *If* $\langle P, \lambda\rangle \xrightarrow{\tau} \langle P', \lambda'\rangle$ *then* $\lambda \sqsubseteq \lambda'$.

PROOF We prove this proposition by transition induction.

- Consider the transition $\langle c \to P, \lambda\rangle \xrightarrow{\tau} \langle 0, \lambda[^{false}/_{L}]\rangle$ where $P \Uparrow L$. Since $\langle c \to P, \lambda\rangle$ is valid, $\lambda(\ell) = \bot$ for all $\ell \in L$. Hence, $\lambda \sqsubseteq \lambda[^{false}/_{L}]$.
- Consider the proof

$$\frac{\langle P_1, \lambda\rangle \xrightarrow{\tau} \langle P_1', \lambda'\rangle}{\langle P_1 + P_2, \lambda\rangle \xrightarrow{\tau} \langle P_1' + P_2, \lambda'\rangle}$$

  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 $\langle P, \lambda_2 \rangle \xrightarrow{\tau} \langle P', \lambda_2' \rangle$ and $\lambda_1 \sqsubseteq \lambda_2$. Then,

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

which proves 3. □

This brings us to our first main result: $\langle P, \lambda_\perp \rangle$ and $\langle P, \lambda_\perp \rangle$ are weak bisimilar.

COROLLARY 19 $\langle P, \lambda_\perp \rangle \approx \langle P, \lambda_\perp \rangle$.

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 $\langle P, \lambda_\perp \rangle \; \mathcal{E} \; \langle P, \lambda_\perp \rangle$, we can conclude that $\langle P, \lambda_\perp \rangle \approx \langle P, \lambda_\perp \rangle$. □

6 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 $\langle P, \lambda_\perp \rangle$ and $\langle P, \lambda_\perp \rangle$ 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 $\langle P, \lambda_\perp \rangle$ and $\langle P, \lambda_\perp \rangle$ are not weak trace equivalent.*

PROOF Assume that $\lambda_1 \sqsubseteq \lambda_2$. Without loss of generality, we can assume that

$$\lambda_1(\ell_i) \neq \perp \text{ and } \lambda_2(\ell_i) \neq \perp \text{ and } \lambda_1(\ell_i) = \lambda_2(\ell_i)$$

for all $1 \leq i \leq n$ and

$$\lambda_1(\ell_{n+i}) = \perp \text{ and } \lambda_2(\ell_{n+i}) = \mathit{false}$$

for all $1 \leq i \leq m$ and

$$\lambda_1(\ell) = \perp \text{ and } \lambda_2(\ell) = \perp$$

for all other $\ell \in \mathbb{L}$. Now consider the processes

$$\begin{array}{lcl} P & = & \ell_1 \uparrow \lambda_1(\ell_1).\ell_2 \uparrow \lambda_1(\ell_2) \ldots \ell_n \uparrow \lambda_1(\ell_n).Q \\ Q & = & (a_1.c \to a_2.0) + (\ell_{n+1} \uparrow \mathit{true}.\ell_{n+2} \uparrow \mathit{true} \ldots \ell_{n+m} \uparrow \mathit{true}.0) \end{array}$$

Since (3), we have that

$$\langle P, \lambda_\perp \rangle \xrightarrow{\iota^n} \langle Q, \lambda_1 \rangle \xrightarrow{a_1} \langle c \to a_2.0, \lambda_1 \rangle \not\to$$

and

$$\langle P, \lambda_\perp \rangle \xrightarrow{\iota^n} \langle Q, \lambda_1 \rangle \xrightarrow{a_1} \langle c \to a_2.0, \lambda_2 \rangle \xrightarrow{\iota} \langle a_2.0, \lambda_2 \rangle \xrightarrow{a_2} \langle 0, \lambda_2 \rangle \not\to$$

Therefore, $\langle P, \lambda_\perp \rangle$ and $\langle P, \lambda_\perp \rangle$ are not weak trace equivalent. □

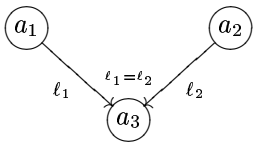
Conclusion

Let us first summarize our contributions. We introduced a new 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 DPE. We formulated a condition that exactly captures when DPE has side effects. We proved that this condition is sufficient and necessary for DPE to be free of side effects.

Next, let us consider the implications of our main results for BPEL4WS. According to [CGK+02], the join conditions of BPEL4WS consist of links combined by means of the Boolean operators of XPath [W3C99]. Roughly, this set of join conditions is defined by

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

Consider the join condition $\ell_1^2 = \ell_2$ that we already saw in the introductory section of the paper. Assume that $\lambda_1(\ell_1^2) = \perp$, $\lambda_1(\ell_2) = \mathit{false}$, $\lambda_2(\ell_1^2) = \mathit{false}$ and $\lambda_2(\ell_2) = \mathit{false}$. Obviously, $\lambda_1 \sqsubseteq \lambda_2$. Furthermore, $\mathcal{C}(\ell_1^2 = \ell_2)(\lambda_1) \neq \mathit{true}$ and $\mathcal{C}(\ell_1^2 = \ell_2)(\lambda_2) = \mathit{true}$. Hence, SEF is not satisfied. The fact that these join conditions give rise to an evaluation function $\mathcal{C}$ that does not satisfy SEF is caused by the presence of negative occurrences of links in the join conditions. For example, 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 this join condition. These negative occurrences can be eliminated from join conditions by removing $c = c$ and $c \neq c$ from the above definition of the set of join conditions. However, note that not every negative occurrence is problematic. For example, in



both the links ℓ_1 and ℓ_2 have a negative occurrence in the join condition $\ell_1 = \ell_2$, but DPE has no side effects.

In BPEL4WS, a join condition is only evaluated if all links that are part of the join condition are defined. For example, if $\lambda_1(\ell_1^2) = \perp$ and $\lambda_1(\ell_2) = \mathit{true}$ then $\mathcal{C}(\ell_1^2 \vee \ell_2) = \perp$, and if $\lambda_1(\ell_1^2) = \mathit{false}$ and $\lambda_1(\ell_2) = \mathit{true}$ then $\mathcal{C}(\ell_1^2 \vee \ell_2) = \mathit{true}$. Clearly, $\lambda_1 \sqsubseteq \lambda_2$ and, hence, SEF is not satisfied. This kind of violation of SEF can be addressed by using lazy evaluation of join conditions. In that case, if $\lambda_1(\ell_1^2) = \perp$ and $\lambda_1(\ell_2) = \mathit{true}$ then $\mathcal{C}(\ell_1^2 \vee \ell_2) = \mathit{true}$. This approach is supported in, for example, the web services flow language (WSFL) [Ley01].

In conclusion, DPE may have side effects in BPEL4WS. From time to time, these side effects may be introduced accidentally. Therefore, we believe that it is important that one is aware of the fact that DPE may have side effects.

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, 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 synchronization patterns like DPE cannot easily be captured by means of Petri nets.

Acknowledgements

The authors would like to thank Bill O'Farrell, Marin Litoiu and Jon Bennett for many fruitful discussions about BPEL4WS in general and DPE in particular.

References

[AFV01] L. Aceto, W.J. Fokkink, and C. Verhoef. Structural operational semantics. In J.A. Bergstra, A. Ponse, and S.A. Smolka, editors, *Handbook of Process Algebra*, pages 197–292. Elsevier, 2001.

[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, 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.

[BGE+02] T. Barclay, J. Gray, S. Ekblad, E. Strand, and J. Richter. TerraService.NET: An introduction to web services. Technical Report MS-TR-2002-53, Microsoft Research, Redmond, WA, June 2002.

[CGK+02] F. Curbera, Y. Goland, J. Klein, F. Leymann, D. Roller, S. Thatte, and S. Weerawarana. Business process execution language for web services, version 1.0. Available at `http://www.ibm.com/developerworks/library/ws-bpel/`, July 2002.

[Gla90] R.J. van Glabbeek. The linear time-branching time spectrum. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings of the 1st International Conference on Concurrency Theory*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297, Amsterdam, August 1990. Springer-Verlag.

[Gla93] R.J. van Glabbeek. The linear time-branching time spectrum II. In E. Best, editor, *Proceedings of the 4th International Conference on Concurrency Theory*, volume 715 of *Lecture Notes in Computer Science*, pages 66–81, Hildesheim, August 1993. Springer-Verlag.

[Ley01] F. Leymann. Web services flow language. Available at `http://www-3.ibm.com/software/solutions/webservices/pdf/WSFL.pdf`, May 2001.

[Mil89] R. Milner. *Communication and Concurrency.* Prentice Hall International, 1989.

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

[Plo82] G.D. Plotkin. An operational semantics for CSP. In D. Bjorner, editor, *Proceedings of IFIP Working Conference on Formal Description of Programming Concepts - II*, pages 199–223, Garmisch-Partenkirchen, June 1982. North-Holland.

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

[Rep99] J.H. Reppy. *Concurrent Programming in ML.* Cambridge University Press, 1999.

[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.

[W3C99] W3C. XML path language, version 1.0. Available at `http://www.w3.org/TR/xpath/`, November 1999.

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