RZ 3745 (# 99755) 09/24/09 – Revised Version 07/20/2010
Computer Science 17 pages

Research Report

Simplified Computation and Generalization of the Refined Process Structure Tree

Revised Version, July 2010

Artem Polyvyanyy[1], Jussi Vanhatalo[2] and Hagen Völzer[3]

[1]Hasso Plattner Institute, Prof.-Dr.-Helmert-Str. 2-3, 14482 Potsdam, Germany
Artem.Polyvyanyy@hpi.uni-potsdam.de

[2]UBS AG, Postfach, 8098 Zurich, Switzerland
jussi.vanhatalo@ieee.org

[3]IBM Research – Zurich, Säumerstrasse 4, 8803 Rüschlikon, Switzerland
hvo@zurich.ibm.com

Simplified Computation and Generalization of the Refined Process Structure Tree

Artem Polyvyanyy[1], Jussi Vanhatalo[2], and Hagen Völzer[3]

[1] Hasso Plattner Institute, Prof.-Dr.-Helmert-Str. 2-3, D-14482 Potsdam, Germany
Artem.Polyvyanyy@hpi.uni-potsdam.de
[2] UBS AG, Postfach, CH-8098 Zurich, Switzerland
jussi.vanhatalo@ieee.org
[3] IBM Research – Zurich, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland
hvo@zurich.ibm.com

Abstract. A business process is often modeled using some kind of a directed flow graph, which we call a *workflow graph*. The Refined Process Structure Tree (RPST) [1] is a technique for workflow graph parsing, i.e., for discovering the structure of a workflow graph, which has various applications. In this paper, we provide two improvements to the RPST. First, we propose an alternative way to compute the RPST that is simpler than the one developed originally [1]. In particular, the computation reduces to constructing the *tree of the triconnected components* of a workflow graph in the special case when every node has at most one incoming or at most one outgoing edge. Such graphs occur frequently in applications. Secondly, we extend the applicability of the RPST. Originally, the RPST was applicable only to graphs with a single source and single sink such that the completed version of the graph is biconnected. We lift both restrictions. Therefore, the RPST is then applicable to arbitrary directed graphs such that every node is on a path from some source to some sink. This includes graphs with multiple sources and/or sinks and disconnected graphs.

1 Introduction

Companies widely use business process modeling for documenting their operational procedures. Business analysts develop process models by decomposing business scenarios into business activities and defining their logical and temporal dependencies. The models are then utilized for communicating, analyzing, optimizing, and supporting execution of individual business cases within or across companies. Various modeling notations have been proposed. Many of them, for example the Business Process Modeling Notation (BPMN), Event-driven Process Chains (EPC), and UML activity diagrams, are based on *workflow graphs*, which are directed graphs with nodes representing activities or control decisions, and edges specifying temporal dependencies.

A workflow graph can be parsed into a hierarchy of subgraphs with a single entry and single exit. Such a subgraph is a logically independent subworkflow or subprocess of the business process. The result of the parsing procedure is a *parse tree*, which is the containment hierarchy of the subgraphs. The parse tree has various applications, e.g., translation between process languages [1–3], control-flow and data-flow analysis [4–7], process comparison and merging [8], process abstraction [9], process comprehension [10], model layout [11], and pattern application in process modeling [12].

Vanhatalo, Völzer, and Koehler [1] proposed a workflow graph parsing technique, called the *Refined Process Structure Tree* (RPST), that has a number of desirable properties: The resulting parse tree is unique and *modular*, where *modular* means that a local change in the workflow graph only results in a local change of the parse tree. Furthermore, it is finer grained than any known alternative approach and it can be computed in linear time. The linear time computation is based on the idea by Tarjan and Valdes [13] to compute a parse tree based on the *triconnected components* of a biconnected graph.

In this paper, we improve the RPST in two ways:

- The original RPST algorithm [1] contains, besides the computation of the triconnected components, a post-processing step that is fairly complex. In this paper, we show that the computation can be considerably simplified by introducing a pre-processing step that splits every node of the workflow graph with more than one incoming and more than one outgoing edge into two nodes. We prove that for the resulting graph, the RPST and the triconnected components coincide. Furthermore, we prove that the RPST of the original graph can then be obtained by a simple post-processing step. This new approach reduces the implementation effort considerably, requiring only little more than the computation of the triconnected components, of which an implementation is publicly available [14].
- The original technique [1] is restricted to workflow graphs that have a single source and a single sink such that adding an edge from the sink to the source makes the graph biconnected. This assumption is too restrictive in practice as many business process models have multiple sources and/or sinks, some are not biconnected, and some are not even connected. In this paper, we show how these limitations can be overcome. The resulting technique can be applied to any workflow graph such that each node lies on a path from some source to some sink.

The remainder of the paper is structured as follows: The next section defines the RPST and provides additional preliminary definitions. Sect. 3 proposes the simplified algorithm for computing the RPST, and Sect. 4 then generalizes the algorithm to operate on workflow graphs of arbitrary structure.

2 Preliminaries

This section presents the preliminary notions: the RPST [1] in Sect. 2.1, and the triconnected components of the graph [13, 15] in Sect. 2.2. We refer to the corresponding original articles for additional motivation of the definitions presented in this section.

2.1 The Refined Process Structure Tree

A *multi-graph* $G = (V, E, \ell)$ consists of two disjoint sets V and E of *nodes* and *edges*, respectively, and a mapping ℓ that assigns to each edge either an ordered pair of nodes, in which case G is a *directed multi-graph*, or an unordered pair of nodes, in which case G is an *undirected multi-graph*. A pair of nodes may be connected by more than one edge (hence the name multi-graph). We assume that the mapping ℓ is fixed, so that a subgraph can be identified with a pair (V', E'), where $V' \subseteq V$ and $E' \subseteq E$ such that each edge in E' connects only nodes in V'. Let $F \subseteq E$ be a set of edges, $G_F = (V_F, F)$ is the subgraph *formed by* F if V_F is the smallest set of nodes such that (V_F, F) is a subgraph.

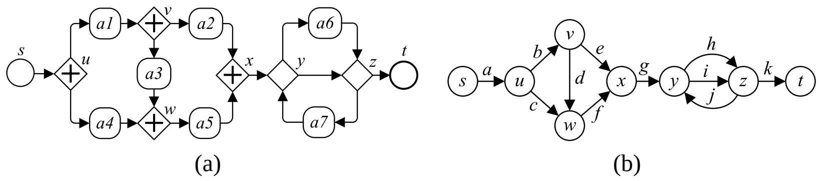


Fig. 1. (a) A workflow graph represented in BPMN, (b) the corresponding TTG (simplified)

A *multi-terminal graph* (*MTG*) is a directed multi-graph G that has at least one source and at least one sink such that each node lies on a path from some source to some sink; G is a *two-terminal graph* (*TTG*) if it has exactly one source and exactly one sink. Fig.1(a) shows a workflow graph in BPMN notation and Fig.1(b) presents the corresponding TTG. Note that the activity nodes ($a1, a2$, etc.) are ignored in the TTG for the sake of simplicity. We assume for simplicity of the presentation that a TTG has at least two nodes and two edges.

Let G be an MTG and $G_F = (V_F, F)$ be a connected subgraph of G that is formed by a set F of edges. A node in V_F is *interior* with respect to G_F if it is connected only to nodes in V_F; otherwise it is a *boundary node* of G_F. A boundary node u of G_F is an *entry* of G_F if no incoming edge of u belongs to F or if all outgoing edges of u belong to F. A boundary node v of G_F is an *exit* of G_F if no outgoing edge of v belongs to F or if all incoming edges of v belong to F. F is a *fragment* of a TTG G if G_F has exactly two boundary nodes, one entry and one exit. The set $\{u, v\}$ containing the entry and the exit node is also called the *entry-exit pair* of the fragment. A fragment is *trivial* if it only contains a single edge. Note that every singleton edge forms a fragment. By definition, the source of a TTG is an entry to every fragment it belongs to and the sink of a TTG is an exit from every fragment it belongs to. Intuitively, control 'enters' the TTG through the source and 'exits' the TTG through the sink. Note also that we represent a fragment as a set of edges rather than as a subgraph.

We say that two fragments F, F' are *nested* if $F \subseteq F'$ or $F' \subseteq F$. They are *disjoint* if $F \cap F' = \emptyset$. If they are neither nested nor disjoint, we say that they *overlap*. A fragment of G is said to be *canonical* (or *objective*) if it does not overlap with any other fragment of G. The *Refined Process Structure Tree (RPST)* of G is the set of all canonical fragments of G. It follows that any two canonical fragments are either nested or disjoint and, hence, they form a hierarchy. This hierarchy can be shown as a tree, where the parent of a canonical fragment F is the smallest canonical fragment that contains F. The root of the tree is the entire graph, the leaves are the trivial fragments.

Fig.2 exemplifies the RPST. Fig.2(a) shows a TTG and its canonical fragments, where every fragment is formed by edges enclosed in or intersecting an area denoted by the dotted border. For example, the canonical fragment $T1$ is formed by edges $\{b, c, d, e, f\}$, has interior nodes $\{v, w\}$ and boundary nodes $\{u, x\}$, with u being an entry and x an exit of the fragment. Fig.2(b) visualizes the RPST as a tree.

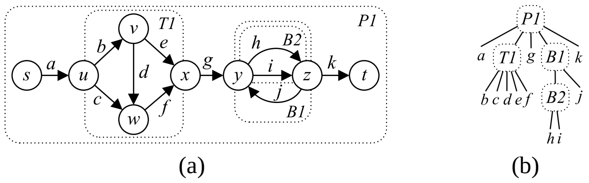


Fig. 2. (a) A TTG and its canonical fragments, (b) the RPST of (a)

2.2 The Triconnected Components

The fragments of a TTG are closely related to its *triconnected components*, which was pointed out by Tarjan and Valdes [13]. This relationship is crucial for the results that are obtained later in this paper. Here, we introduce the triconnected components in detail and we start with some preliminary definitions.

The *completed version* of a TTG G, denoted $C(G)$, is the undirected graph that results from ignoring the direction of all the edges of G and adding an additional edge between the source and the sink. The additional edge is called the *return edge* of $C(G)$. Let G be an undirected multi-graph. G is *connected* if each pair of nodes is connected by a path; G is *biconnected* if G has no self-loops and if for each triple u, v, x of nodes, there is a path from u to v that does not visit x. If a node x witnesses that G is not biconnected, i.e., there exist nodes u, v such that x is on every path between u and v, then x is called a *separation point* of G. G is *triconnected* if for each quadruple u, v, x, y of nodes, there is a path from u to v that visits neither x nor y. A pair $\{x, y\}$ witnessing that G is not triconnected is called a *separation pair* of G, i.e., there exist nodes u, v such that every path from u to v visits either x or y.

The TTG in Fig.1(b) is connected, but not biconnected; the nodes u, x, y, and z are all separation points. Fig.3(a) shows the completed version $C(G)$ of the TTG from Fig.1(b), where r is the return edge. The completed version is biconnected but not triconnected; $\{u, x\}$ and $\{x, z\}$ are two of many separation pairs of $C(G)$.

Fragments are strongly related to triconnectivity and separation pairs. Note that the entry-exit-pair $\{u, x\}$ of fragment $T1$ in Fig.2(a) is also a separation pair of its completed version in Fig.3(a). In fact, each entry-exit pair of a non-trivial fragment of a TTG G is a separation pair of $C(G)$.

An (undirected) graph that is not connected can be uniquely partitioned into *connected components*, i.e., maximal connected subgraphs. A connected graph that is not biconnected can be uniquely decomposed into *biconnected components*, i.e., maximal biconnected subgraphs. The biconnected components can be obtained by splitting the graph into multiple subgraphs at each separation point. Because of the relationship of fragments to triconnectivity, we are interested to decompose a graph into unique triconnected components. That decomposition is explained in the remainder of this section.

Let G be a biconnected multi-graph and u, v be two nodes of G. A *separation class w.r.t. u, v* is a maximal set S of edges such that any two edges in S are connected by a path that visits neither u nor v except as a start or end point. If there is a partition of all edges of G into two sets E_0, E_1 such that both sets contain more than one edge and each separation class w.r.t. u, v is contained in either of these sets, we call $\{u, v\}$ a *split pair*. We can then *split* the graph into two parts w.r.t. the parameters E_0, E_1 and u, v: To this end, we add a fresh edge e between u and v to the graph, which is called a *virtual edge*.

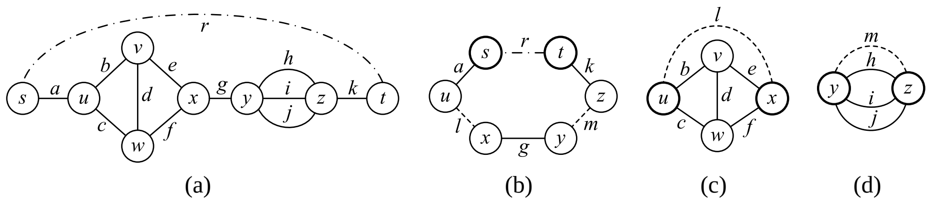


Fig. 3. The completed version of the TTG from Fig.1(b) and its triconnected components: (a) The completed version, (b) a polygon, (c) a rigid component, and (d) a bond

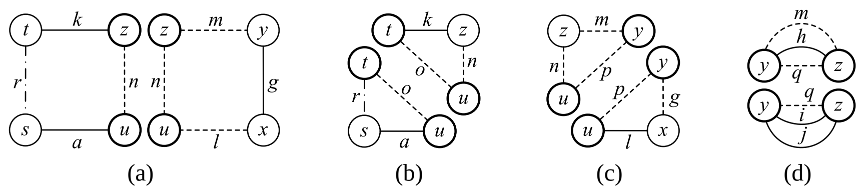


Fig. 4. (a) A split of a hexagon from Fig.3(b), (b)-(c) a split of a tetragon, (d) a split of a bond

The graphs formed by the sets $E_0 \cup \{e\}$ and $E_1 \cup \{e\}$ are the obtained *split graphs* of the performed split operation. A virtual edge is visualized by a dashed line.

For an example of a split operation, consider the hexagon in Fig.3(b). Note that it already contains virtual edges, which are the result of previous splits. The hexagon can be split along the split pair u, z using the sets $E_1 = \{k, r, a\}, E_2 = \{m, g, l\}$. This results in two tetragons, which are shown in Fig.4(a).

It may be possible to split the obtained split graphs further, i.e., into smaller split graphs, possibly w.r.t. a different split pair. A split graph is called a *split component* if it cannot be split further. Special split graphs are *polygons* and *bonds*. A *polygon* is a graph that has $k \geq 3$ nodes and k edges such that all nodes and edges are contained in a cycle, cf., Fig.3(b). A *bond* consists of 2 nodes and $k \geq 2$ edges between them, cf., Fig.3(d). Each split component is either a *triangle*, i.e., a polygon with three nodes, a *triple bond*, i.e., a bond with three edges, or a *simple* triconnected graph, where *simple* means that no pair of nodes is connected by more than one edge [15]. If a split component is the latter, we also call it a *rigid* component. Fig.3(c) shows an example of a rigid component, whereas the split graphs shown in Fig.3(b) and Fig.3(d) are not split components as they can be split further.

The set of split components that can be derived from a biconnected multi-graph is not unique. To see that, we consider polygons and bonds. For instance, a tetragon, cf., Fig.4(a), can be split along a diagonal into two split graphs. Depending on the choice of the diagonal, two different sets of split components are obtained. Fig.4(b) shows one of the two possibilities for splitting the tetragon given on the left in Fig.4(a). Similarly, a bond with more than three edges, cf., Fig.3(d), can be split into two bonds in several ways, depending on the choice of E_1 and E_2. One possibility to split the bond from Fig.3(d) is shown in Fig.4(d). A set of split components for the graph in Fig.3(a) is given by the graphs in Figs.3(c), 4(b), 4(c), and 4(d).

The inverse of a split operation is called a *merge* operation. Two split graphs formed by edges E_0 and E_1, respectively, that share a virtual edge e between a pair u, v of nodes can be merged, which results in the graph formed by the set $(E_0 \cup E_1) \setminus \{e\}$ of edges. If we start with a set of split components of G and then iteratively merge a polygon with a polygon and a bond with a bond until no more such merging is possible, we obtain the unique *triconnected components* of G. Because a merge operation is the inverse of a split operation, we can also obtain the triconnected components by suitable split operations only: Let $\mathcal{C}$ be a *split graph decomposition* of G, i.e., a set of split graphs recursively derived from G. A polygon $P \in \mathcal{C}$ is *maximal* w.r.t. $\mathcal{C}$ if there is no other polygon in $\mathcal{C}$ that shares a virtual edge with P. A bond $B \in \mathcal{C}$ is *maximal* w.r.t. $\mathcal{C}$ if there is no other bond in $\mathcal{C}$ that shares a virtual edge with B. $\mathcal{C}$ is a set of the *triconnected components* of G if each member of $\mathcal{C}$ is either a maximal polygon, a maximal bond, or a rigid split component. The set of the triconnected components of G exists and is unique, cf., [15].

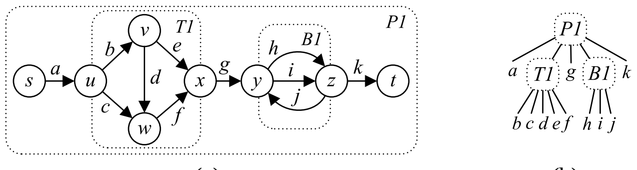


Fig. 5. (a) A TTG and its triconnected component subgraphs, (b) the tree of the triconnected components of (a)

The graphs in Fig.4(c) can be merged along the virtual edge *p*. The obtained tetragon can be merged with the triangles in Fig.4(b) along the virtual edges *n* and *o* to obtain the maximal polygon from Fig.3(b). Figs.3(b), 3(c), and 3(d) show all the triconnected components of the graph from Fig.3(a): Fig.3(d) is a maximal bond, which is obtained by merging the bonds in Fig.4(d), and Fig.3(c) is a rigid component.

Any split graph decomposition can be arranged in a tree: The tree nodes are the split graphs. Two split graphs are connected in a tree if they share a virtual edge. The root of the tree is the split graph that contains the return edge. The *tree of the triconnected components* of G is the tree derived in this way from its triconnected components.

Let C be a triconnected component of graph G. Let F be the set of all edges of G that appear in C or some descendant of C in the tree of the triconnected components. The graph formed by F is called the *triconnected component subgraph* derived from C.

Fig.5 shows the tree of the triconnected components. In Fig.5(a), the triconnected component subgraphs of the workflow graph are visualized; they correspond to the triconnected components from Fig.3. Each triconnected component subgraph is formed by edges enclosed in or intersecting a region with the dotted border, e.g., all the graph edges for $P1$ are derived from the component given in Fig.3(b). Fig.5(b) arranges the triconnected components in a tree. The root of the tree, i.e., node $P1$, corresponds to the triconnected component that contains the return edge r. Note the difference between the tree of the triconnected components in Fig.5 and the RPST in Fig.2

3 Simplified Computation of the Refined Process Structure Tree

In this section, we show how the RPST computation can be simplified compared with the original algorithm. In Sect. 3.1, we discuss the RPST of TTGs in which every node has at most one incoming or at most one outgoing edge. Such TTGs are common in practice. In Sect. 3.2, we address the general case of the RPST computation of any TTG whose completed version is biconnected.

3.1 The RPST of Normalized TTGs

We call a TTG *normalized* if every node has at most one incoming or at most one outgoing edge. In this section, we show that for normalized TTGs, the RPST computation reduces to computing the tree of the triconnected components. In other words, each canonical fragment corresponds to a triconnected component subgraph and each triconnected component subgraph corresponds to a canonical fragment.

Let $C(G)$ be the completed version of a TTG. A pair $\{x,y\}$ of nodes is called a *boundary pair* if there are at least two separation classes w.r.t. $\{x,y\}$. A separation class is *proper* if it does not contain the return edge. The boundary pair $\{u,x\}$ in Fig.3(a) generates two separation classes. The first contains the edges b,c,d,e,f and is therefore proper, whereas the second contains all other edges of the graph and is therefore not proper. Fragments are strongly related to proper separation classes. To describe that relationship, we introduce the notion of a *separation component*.

Definition 1 (Separation component). Let $\{x,y\}$ be a boundary pair of $C(G)$. A *separation component* w.r.t. $\{x,y\}$ is the union of one or more proper separation classes w.r.t. $\{x,y\}$.

The bond from Fig.3(d) without the virtual edge m is a separation component w.r.t. $\{y,z\}$ of the completed version of the TTG from Fig.3(a). It is the union of the three proper separation classes: $\{h\}$, $\{i\}$, and $\{j\}$.

We know that the entry-exit pair $\{x,y\}$ of a fragment is a boundary pair of G and that the fragment is a *separation component* w.r.t. $\{x,y\}$ [1]. Furthermore, it follows from the construction of the triconnected components that each triconnected component subgraph is a separation component. Polyvyanyy et al. [9] observed that every triconnected component subgraph of a normalized TTG is a fragment. For normalized TTGs, we can extend this observation to a full characterization of fragments in terms of separation components.

Lemma 1. *Let F be a set of edges of a normalized TTG. F is a separation component if and only if F is a fragment.*

Proof. For ($\Rightarrow$), let $\{u,v\}$ be the boundary pair of F and let e be an edge in F. As the return edge is not in F, it is in a different separation class w.r.t. $\{u,v\}$ than e. Consider a simple directed path from the source to the sink of the graph that contains e. It follows that the path contains one of the nodes $\{u,v\}$ before e and one after e; otherwise the separation class of e would contain the return edge. Let, without loss of generality, u be the former node and v the latter. It follows that u has an incoming edge outside F and an outgoing edge inside F, and v has an incoming edge inside F and an outgoing edge outside F. Based on the assumption that the TTG is normalized, it is now straightforward to establish that u is an entry and v is an exit of F. Furthermore, there is no other boundary node besides u and v because that would contradict the definition of a separation class. Hence, F is a fragment.

The direction ($\Leftarrow$) is Theorem 2 in [1]. $\square$

It turns out that the set of triconnected component subgraphs of a normalized TTG is exactly the set of all its canonical fragments and, thus, is the RPST of the TTG. Before we prove the statement, we give two auxiliary lemmas which also by themselves deliver interesting insights into separation components of a normalized TTG and their relations.

Lemma 2. *If F is a separation component and F' a triconnected component subgraph, then F and F' do not overlap.*

Proof. If F contains only a single edge or the entire graph, the claim is trivial. Otherwise F can be split off from the main graph into a split graph. We continue the decomposition until we reach a set of split components. Those can be arranged in a tree (of split components) as described above. F corresponds to a subgraph of this tree, i.e., a subtree represents exactly the edges of F. On the other hand, F' also corresponds to a subtree of the tree of split components because the triconnected components are obtained by merging split components, i.e., by collapsing parts of the tree of split components. Since F

and F' both correspond to subtrees of the same tree, they do not overlap. □

It follows from Lemma 2 that triconnected component subgraphs do not overlap. We show now that for a separation component which is strictly contained in a triconnected component subgraph, there always exists another separation component contained in the same triconnected component subgraph that overlaps with it.

Lemma 3. *If F is a separation component that is not a triconnected component subgraph, then there exists a separation component F', such that F and F' overlap.*

Proof. Consider a split graph decomposition that contains F. If F is not a triconnected component subgraph, then F and the parent of F are either bonds w.r.t. the same boundary pair or polygons. In both cases, it is easy to display a bond or polygon, respectively, that overlaps with F. □

We are now ready to prove the main proposition of this section.

Theorem 1. *Let F be a set of edges of a normalized TTG. F is a canonical fragment if and only if F is a triconnected component subgraph.*

Proof.
- ⇒ Let F be a canonical fragment. We want to show that F is a triconnected component subgraph. Because of Lemma 1, F is a separation component. If F is not a triconnected component subgraph, then there exists, because of Lemma 3, a separation component F' that overlaps with F. Because of Lemma 1, F' is a fragment, which contradicts F being canonical.
- ⇐ Let F be a triconnected component subgraph. We want to show that F is a canonical fragment. Because of Lemma 1, F is a fragment. Let F' be any fragment. Because of Lemma 1, F' is a separation component. Because of Lemma 2, F and F' do not overlap. Hence, F is a canonical fragment. □

For normalized TTGs, Theorem 1 implies that the tree of the triconnected components and the RPST coincide, i.e., both deliver the same decomposition on the set of edges of the TTG. Fig.6(a) shows a normalized TTG and its triconnected component subgraphs. The TTG is formed by a subset of edges of the workflow graph from Fig.1(b). The triconnected component subgraphs are also all the canonical fragments of the TTG. Therefore, the RPST of the workflow graph from Fig.6(a), which is given in Fig.6(b), can be computed by constructing the tree of the triconnected components of the workflow graph.

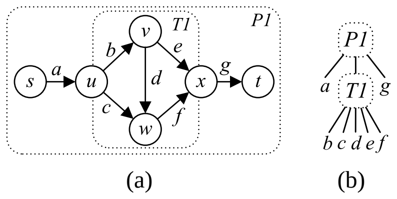


Fig. 6. (a) A TTG and its triconnected component subgraphs, (b) the RPST of (a)

3.2 The RPST of General TTGs

We now show how to compute the RPST of an arbitrary TTG whose completed version is biconnected. To do so, we normalize the TTG by splitting nodes that have more than one incoming and more than one outgoing edge into two nodes. We then compute the RPST of the normalized TTG as in Sect. 3.1. Finally, we project the RPST of the normalized TTG onto the original one and obtain the RPST of the original TTG.

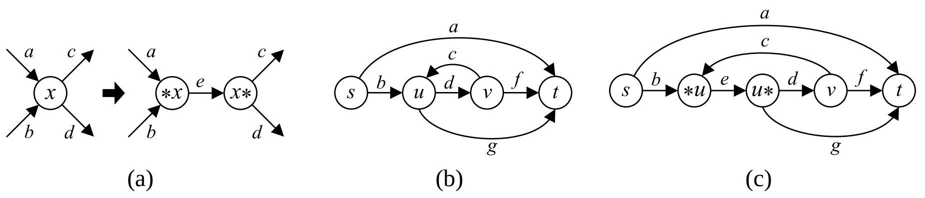


Fig. 7. (a) Node-splitting, (b) a TTG, and (c) the normalized version of (b)

A single node-splitting is sketched in Fig.7(a). For instance, if the splitting is applied to node u of the graph from Fig.7(b), it results in the new graph given in Fig.7(c) with three fresh elements: nodes $*u$ and $u*$, and edge e. This is the only applicable splitting in the example. Hence, the resulting graph is normalized and we call it the *normalized version* of the TTG. The procedure can be formalized as follows.

Definition 2 (Node-splitting). Let $G = (V, E, \ell)$ be a directed multi-graph and $x \in V$ a node of G. A *splitting* of x is *applicable* if x has more than one incoming and more than one outgoing edge. The application results in a graph $G' = (V', E', \ell')$, where $V' = (V \setminus \{x\}) \cup \{*x, x*\}$, $E' = E \cup \{e\}$, where $*x$ and $x*$ are fresh nodes and e is a fresh edge, and ℓ' is such that $\ell'(e) = (*x, x*)$. In addition, $f \in E, \ell(f) = (y, z)$ and $\ell'(f) = (y', z')$ implies that $y' = x*$ if $y = x$, and otherwise $y' = y$; and $z' = *x$ if $z = x$, and otherwise $z' = z$.

Splitting is applicable if and only if the graph is not normalized. It is not difficult to see that the order of different splittings does not influence the final result and, therefore, we indeed get a normal form by applying all applicable splittings in any order.

After normalization, we proceed by computing the tree of the triconnected components of the graph. As we know from Sect. 3.1, the tree coincides with the RPST of the normalized graph. This tree can be projected onto the original graph by deleting all the edges introduced during node-splittings. We will see later that this projection preserves the fragments. However,the deletion of the edges may result in fragments which have a single child fragment. This means that two different fragments of the normalized graph project onto the same fragment of the original graph. We thus clean the tree by deleting redundant occurrences of such fragments. Consequently, the only child fragment of a redundant fragment becomes a child of the parent of the redundant fragment, or the root of the tree if the redundant fragment has no parent. The result is the RPST of the original graph. Alg. 1 details again the sequence of these steps.

Algorithm 1 Simplified computation of the RPST

```
RPST(Directed multi-graph $G = (V, E, \ell)$)
1. $G' = (V', E', \ell')$ is the normalized version of $G$
2. $T'$ is the tree of the triconnected components of $G'$
3. $T$ is $T'$ without trivial fragments in $E' \setminus E$
4. $R$ is $T$ without redundant fragments
5. return $R$ // the RPST of $G$
```

We exemplify Alg. 1 in Fig.8 and Fig.9 by computing the RPST of the TTG from Fig.8(a). Fig.8(a) shows the triconnected component subgraphs $P1$ and $B1$ of the TTG, whereas Fig.8(b) shows the corresponding tree of the triconnected components. The TTG is not normalized: Nodes y and z are incident with multiple incoming and multiple

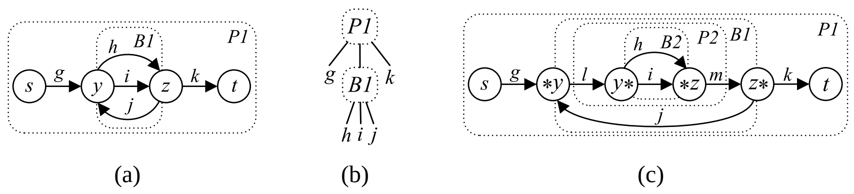


Fig. 8. (a) A TTG and its triconnected component subgraphs, (b) the tree of the triconnected components of (a), and (c) the normalized version of (a) and its triconnected component subgraphs

outgoing edges, and all the triconnected component subgraphs of the TTG are fragments. Fig.8(c) shows the normalized version of the TTG from Fig.8(a); it is obtained by splitting nodes y and z, in any order. The normalization introduces edges l and m to the TTG. The tree of the triconnected components of the normalized version consists of four triconnected components: $P1$, $B1$, $P2$, and $B2$ shown in Fig.8(c). It follows from Lemma 1 that they are all fragments.

Fig.9(a) shows the tree of the triconnected components of the normalized version from Fig.8(c). Because of Theorem 1, the tree is the RPST of the normalized version. In Fig.9(b), one can see the RPST without trivial fragments, which correspond to the edges l and m. Note that $P2$ now specifies the same set of edges of the TTG as $B2$. Therefore, we omit $P2$, which is redundant, to obtain the tree given in Fig.9(c). This tree is the RPST of the original TTG from Fig.8(a). Fig.9(d) visualizes the TTG again together with its canonical fragments. Please note that Alg. 1, in comparison with the triconnected decomposition shown in Fig.8(a) and Fig.8(b), additionally discovered canonical fragment $B2$. $P1$, $B1$, and $B2$ are all the canonical fragments of the TTG.

To show that we indeed obtain the RPST of the original graph, we have to show that (i) each canonical fragment of the normalized version projects onto a canonical fragment of the original graph or onto the empty set, and (ii) for each canonical fragment of the original graph, there is a canonical fragment of the normalized version that is projected onto it. We establish these properties for a single node-splitting step. The claim then follows by induction.

Consider a single node-splitting step transforming a graph G into G', let x be the node that is split into nodes $*x$ and $x*$, and let e be the edge that is added between $*x$ and $x*$. We define the following mappings for the next lemma:

1. A mapping ψ maps a set F of edges of G' to a set $\psi(F)$ of edges of G by $\psi(F) = F \setminus \{e\}$.

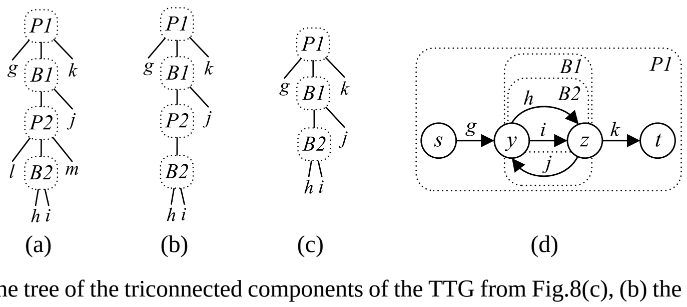


Fig. 9. (a) The tree of the triconnected components of the TTG from Fig.8(c), (b) the tree from (a) without the fresh edges l and m, (c) the RPST of the TTG from Fig.8(a), and (d) the TTG from Fig.8(a) and its canonical fragments

2. A mapping ϕ maps a set of edges H of G to a set $\phi(H)$ of edges of G' by $\phi(H) = H \cup \{e\}$ if H has an incoming edge to x as well as an outgoing edge from x, and otherwise $\phi(H) = H$.

Now, we claim:

Lemma 4. *Let ϕ, ψ and e be defined as above. We have:*

1. *If $F \neq \{e\}$ is a fragment of G', then $\psi(F)$ is a fragment of G.*
2. *If H is a fragment of G, then $\phi(H)$ is a fragment of G'.*
3. *If $F \neq \{e\}$ is a canonical fragment of G', then $\psi(F)$ is a canonical fragment of G.*
4. *If H is a canonical fragment of G, then there exists a canonical fragment F of G' such that $\psi(F) = H$.*

The proof of Lemma 4 is in Appendix A. Lemma 4 and the fact that each step in Alg. 1 can be computed in linear time allow us to conclude:

Theorem 2. *Alg. 1 computes the RPST of a TTG whose completed version is biconnected in linear time.*

4 Generalization of the Refined Process Structure Tree

So far, the RPST decomposition is restricted to TTGs whose completed version is biconnected. In practice this is not sufficient, as a process model may have multiple sources and sinks, cf., Fig.10(b), may be disconnected or may violate biconnectedness assumption. For the latter, consider Fig.10(a). Node u is a separation point of the completed version of the graph as its deletion separates the node labeled with $a1$ from the rest of the graph. Hence, the completed version is not biconnected. Note that process modeling languages such as BPMN and EPC do not impose such structural limitations. In fact, a test of the SAP reference model [16], a collection of industrial process models given as EPCs, showed that more than 80 percent of the models violate one of the restrictions.

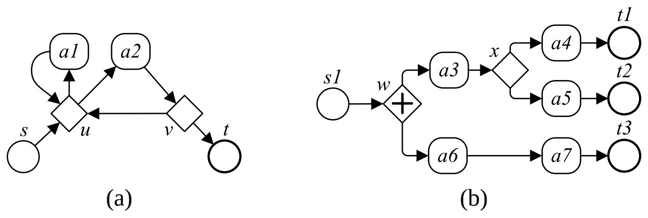


Fig. 10. A workflow graph (a) whose completed version is *not* biconnected, (b) has multiple sinks

In this section, we propose a way to decompose any MTG. The results of this section are also described in detail in a thesis [17]. We start by decomposing arbitrary TTGs.

4.1 The RPST of TTGs

Fig.11(a) shows the TTG that corresponds to the process model in Fig.10(a). As we explained above, its completed version is not biconnected because node u is a separation point. Note that u has multiple incoming as well as multiple outgoing edges. Every separation point has this property:

Lemma 5. *Let G be a TTG. Every separation point of $C(G)$ has more than one incoming and more than one outgoing edge in G.*

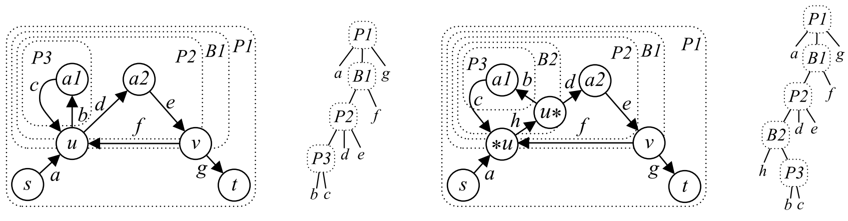


Fig. 11. (a) A TTG whose completed version is *not* biconnected, (b) the RPST of (a), (c) the normalization of (a), and (d) the RPST of (c)

Proof. A source s and a sink t of G are in the same biconnected component of $C(G)$ as they are connected in G and, therefore, biconnected in $C(G)$ after introducing the return edge. Moreover, it is easy to see that $C(G)$ is connected without s or t and, hence, s and t are not separation points of $C(G)$. Let x, without loss of generality, be some separation point of $C(G)$ that results in a set B of biconnected components. Let $b \in B$, without loss of generality, be a biconnected component induced by x that does not contain s and t. Assume y is a node which belongs to b. As every node of G is on a path from s to t, then x is on every path from s to y and from y to t. A path from s to y implies that x has an incoming edge that does not belong to b and an outgoing edge that belongs to b. A path from y to t implies that x has an incoming edge that belongs to b and an outgoing edge that does not belong to b. Hence, the claim holds.

If b consists of a single edge, it is an incoming and an outgoing edge of x. Every path from s to t through x also contains two edges incident with x, an incoming and an outgoing, which do not belong to b. Hence, the claim holds. ☐

It follows that the completed version of the normalization of G is biconnected. Therefore, we can apply Alg. 1 from Sect. 3.2 to decompose an arbitrary TTG. We call the resulting decomposition of G the *RPST* of G. This is a generalization of the previous definition because if $C(G)$ is already biconnected, we get the RPST as defined previously. Note that we obtain the same result by splitting only the separation points of G, computing the RPST of the resulting graph G' (in any way), and then projecting the RPST of G' onto G. As the normalized version and its RPST are unique, it then follows from the construction that the RPST of an arbitrary TTG is unique.

Fig.11 shows the RPST of the example, as well as the way in which it is obtained. Again, the RPST of the original graph is obtained by deleting the edge h, which was generated in the node-splitting, and afterwards removing the redundant fragment $B2$.

Figs. 12(a), 12(b), and 12(c) show more examples of decompositions of TTGs whose completed versions are not biconnected. Every subgraph obtained has either ex-

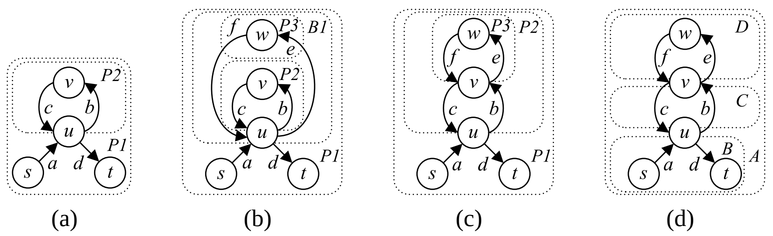


Fig. 12. (a)–(c) The RPST of a TTG, and (d) Valdes's parse tree of the TTG from (c)

actly two boundary nodes, one entry and one exit, or exactly one boundary node, which is *bidirectional*. Let G be a TTG and F be a connected subgraph of G. A boundary node u of F is *bidirectional* if there exist an incoming and an outgoing edge of u inside F, and there exist an incoming and an outgoing edge of u outside F. Note that control flow can both enter and exit F through u.

Valdes [18] has proposed an alternative way to decompose an arbitrary TTG G. He proposed to first compute the *biconnected components* of $C(G)$ and then further decompose each biconnected component into its triconnected components. If we adapt this idea and compute the RPST of each biconnected component of $C(G)$, we obtain a root component that contains all biconnected components as children, which in turn have their RPSTs as subtrees. The result for the graph from Fig.12(c) is shown in Fig.12(d), which is different from the decomposition we propose. Note that the result has a component that has more than two boundary nodes, e.g., B, and another one having two boundary nodes that are both bidirectional, e.g., C. Unlike our decomposition, the decomposition in Fig.12(d) does not reflect the fact that the component containing node w depends on the component that is entered through node u.

4.2 The RPST of MTGs

To decompose an arbitrary MTG, we 'normalize' an MTG into a TTG by constructing a unique source and a unique sink as follows.

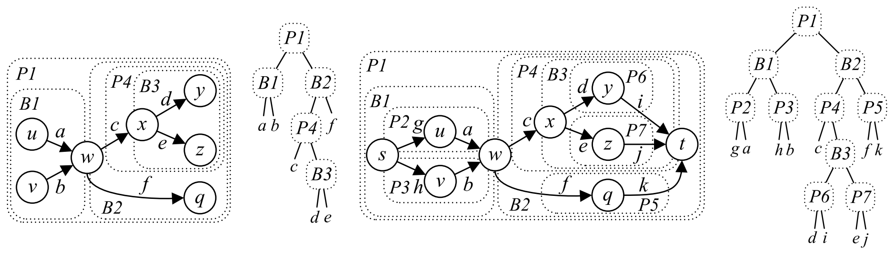
Definition 3. Let G be an MTG. We construct a graph G' from G as follows.
1. If G has more than one source, a new source s is added and for each source node u of G, an edge from s to u is added.
2. If G has more than one sink, a new sink t is added and for each sink node v of G, an edge from v to t is added.

G' is a TTG, which we call the *TTG version* of G. The *normalized version* G^* of G is the normalized version of G'.

By normalizing an MTG, we again obtain a TTG whose completed version is biconnected. The normalized version can be decomposed with the RPST, and the decomposition can be projected onto the original MTG through Alg. 1. The result that is obtained from applying Alg. 1 to the normalized version of an MTG G is called the *RPST* of G. The RPST of an MTG is unique.

Fig.13 shows (a) an MTG G, (b) the RPST of G, (c) the TTG version G' of G, and (d) the RPST of G'. The RPST of G is derived from the RPST of G' with Alg. 1.

Note that for an MTG, the subgraphs formed by the decomposition may have more than two boundary nodes. For example, subgraph $B1$ in Fig.13(a) has two sources u



(a) (b) (c) (d)

Fig. 13. (a) An MTG G, (b) the RPST of G, (c) the TTG version G' of G, and (d) the RPST of G'

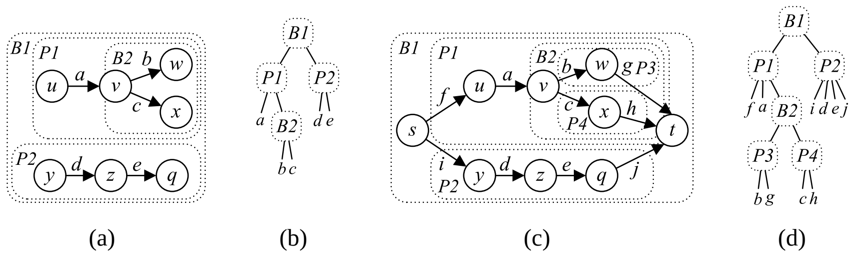


Fig. 14. (a) A disconnected MTG G, (b) the RPST of G, (c) the TTG version G^* of G, and (d) the RPST of G^*

and v as entries, and an exit w. Subgraph $B2$ has an entry w, and three sinks as exits. Subgraph $P1$ two sources as entries, and three sinks as exits.

An RPST-formed subgraph is not necessarily a connected subgraph of an MTG. If an MTG is disconnected, the root fragment of its RPST is a union of the connected components of the MTG. For example, Fig.14 shows an example of (a) a disconnected MTG G, (b) the RPST of G, (c) the TTG (and normalized) version G^* of G, and (d) the RPST of G^*. Note that every connected component of the MTG always becomes a separate component of the RPST decomposition.

Fig.15 shows the RPST-formed fragments of the workflow graphs introduced in Fig.10. We can use these fragments to translate BPMN diagrams into BPEL processes. We have labeled the fragments according to the BPEL blocks they correspond to. For example, *sequence B* in Fig.15(a) is a sequence of a *while* loop and the activity *a2*. These decompositions are not directly obtainable with any prior decomposition technique.

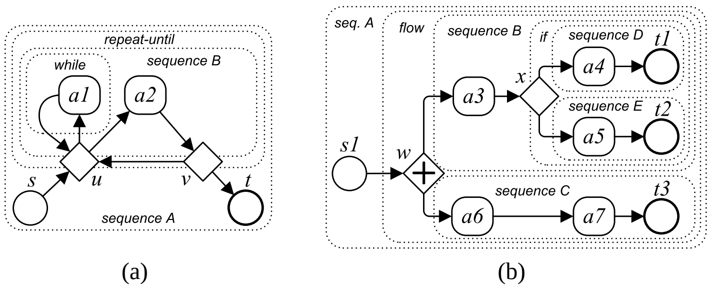


Fig. 15. The RPST-formed fragments of the workflow graphs introduced in Fig.10

5 Conclusion

We simplified the theory for workflow graph parsing into single-entry-single-exit fragments through use of normalized TTGs. This leads to a simplification of the RPST parsing algorithm and its implementation. The implementation effort is essentially reduced to the computation of the triconnected components, of which an implementation is publicly available [14]. In fact, in many applications, nodes have either a single incoming or a single outgoing edge, in which case no pre- and postprocessing steps are required. Together with our previous results [1, 17], we have a parsing technique that produces a unique and modular decomposition in linear time in a simple way. The result has a simple characterization in terms of canonical fragments.

In the second part of the paper, we have shown how the RPST technique gives rise to a decomposition of any workflow graph that may occur in practice. The only remaining assumption is that each node must be on a path from some source to some sink.

We have implemented the simplified RPST computation, as proposed in this paper, and tested its functionally against the implementation of the original RPST technique [1] on the SAP reference model [16], which consists of 604 EPC models. The models were transformed to TTGs that range in size from 2 to 195 edges, with the average of 28.7 edges in one TTG. As it was discovered during evaluation, the models have on average 16.5 non-trivial fragments, ranging from the minimum of 1 fragment to the maximum of 132 fragments in one model.

References

1. Vanhatalo, J., Völzer, H., Koehler, J.: The refined process structure tree. Data & Knowledge Engineering **68**(9) (2009) 793–818
2. García-Bañuelos, L.: Pattern identification and classification in the translation from BPMN to BPEL. In: OTM Conferences (1). Volume 5331 of LNCS. (2008) 436–444
3. Polyvyanyy, A., García-Bañuelos, L., Weske, M.: Unveiling hidden unstructured regions in process models. In: OTM Conferences (1). Volume 5870 of LNCS. (2009) 340–356
4. Fahland, D., Favre, C., Jobstmann, B., Koehler, J., Lohmann, N., Völzer, H., Wolf, K.: Instantaneous soundness checking of industrial business process models. In: BPM. Volume 5701 of LNCS. (2009) 278–293
5. Johnson, R., Pearson, D., Pingali, K.: The program structure tree: Computing control regions in linear time. In: PLDI. (1994) 171–185
6. Johnson, R.: Efficient Program Analysis using Dependence Flow Graphs. PhD thesis, Cornell University, Ithaca, NY, USA (1995)
7. Vanhatalo, J., Völzer, H., Leymann, F.: Faster and more focused control-flow analysis for business process models though SESE decomposition. In: ICSOC 2007. Volume 4749 of LNCS. (2007) 43–55
8. Küster, J., Gerth, C., Förster, A., Engels, G.: Detecting and resolving process model differences in the absence of a change log. In: BPM. Volume 5240 of LNCS. (2008) 244–260
9. Polyvyanyy, A., Smirnov, S., Weske, M.: The triconnected abstraction of process models. In: BPM. Volume 5701 of LNCS. (2009) 229–244
10. Vanhatalo, J., Völzer, H., Leymann, F., Moser, S.: Automatic workflow graph refactoring and completion. In: ICSOC. Volume 5364 of LNCS. (2008) 100–115
11. Battista, G.D., Tamassia, R.: On-line maintenance of triconnected components with SPQR-trees. Algorithmica **15**(4) (1996) 302–318
12. Gschwind, T., Koehler, J., Wong, J.: Applying patterns during business process modeling. In: BPM. Volume 5240 of LNCS. (2008) 4–19
13. Tarjan, R.E., Valdes, J.: Prime subprogram parsing of a program. In: POPL 1980, ACM (1980) 95–105
14. Gutwenger, C., Mutzel, P.: A linear time implementation of SPQR-trees. In: Graph Drawing. Volume 1984 of LNCS. (2000) 77–90
15. Hopcroft, J., Tarjan, R.E.: Dividing a graph into triconnected components. SIAM J. Comput. **2**(3) (1973) 135–158
16. Curran, T., Keller, G., Ladd, A.: SAP R/3 Business Blueprint: Understanding the Business Process Reference Model. Prentice-Hall, Inc., Upper Saddle River, NJ, USA (1997)
17. Vanhatalo, J.: Process structure trees: Decomposing a business process model into a hierarchy of single-entry-single-exit fragments. PhD thesis, University of Stuttgart, Germany (July 2009) Volume 1573, dissertation.de — Verlag im Internet. ISBN: 978-3-86624-473-3.
18. Valdes, J.: Parsing Flowcharts and Series-Parallel Graphs. PhD thesis, Stanford University, CA, USA (1978)

A Appendix — Proofs — NOT TO BE INCLUDED IN THE FINAL PUBLISHED VERSION OF THIS PAPER

Note. The proofs in this appendix are too long to be included in the final published version of this paper, but we will make them available separately, as a technical report. Nevertheless, we provide them for the reviewers of this submission as an appendix.

Consider a single node-splitting step transforming a graph G into G', let x be the node that is split into nodes $*x$ and $x*$, and let e be the edge that is added between $*x$ and $x*$. We define the following mappings for the next lemma:

1. A mapping ψ maps a set F of edges of G' to a set $\psi(F)$ of edges of G by $\psi(F) = F \setminus \{e\}$.
2. A mapping ϕ maps a set of edges H of G to a set $\phi(H)$ of edges of G' by $\phi(H) = H \cup \{e\}$ if H has an incoming edge to x as well as an outgoing edge from x, and otherwise $\phi(H) = H$.

Now, we claim:

Lemma 4. *Let ϕ and ψ be as defined above. We have:*

1. *If $F \neq \{e\}$ is a fragment of G', then $\psi(F)$ is a fragment of G.*
2. *If H is a fragment of G, then $\phi(H)$ is a fragment of G'.*
3. *If $F \neq \{e\}$ is a canonical fragment of G', then $\psi(F)$ is a canonical fragment of G.*
4. *If H is a canonical fragment of G, then there exists a canonical fragment F of G' such that $\psi(F) = H$.*

Proof. We prove each part separately.

1.,2. The proofs of these parts are derived by straightforward applications of the definitions.

3. Suppose $\psi(F)$ are not canonical. Then there exist a fragment H of G that overlaps with $\psi(F)$. We know from part 2 of this lemma that $\phi(H)$ is a fragment, and from $\psi(F) \subseteq F$ and $H \subseteq \phi(H)$ it follows that F and $\phi(H)$ overlap, which contradicts our assumption that F is canonical.
4. Let S_1 and S_2 be two fragments of G such that the exit v of S_1 is the entry of S_2. If $S = S_1 \cup S_2$ is a fragment, we say that S is a *sequence* and S_1 and S_2 are called *segments* of S. We also say that S_1 and S_2 are *in sequence*. It follows from the biconnectedness of $C(G)$ that the entry of S_1 and the exit of S_2 are then distinct. Furthermore, S is a fragment if and only if all nodes that are incident to v belong to S. A sequence S is *maximal* if S is not a segment of another sequence. We know that a sequence is a canonical fragment if and only if it is maximal [1].
   We define the fragment F as follows. If H is a sequence—hence maximal—and x is a boundary node of H such that all outgoing edges or all incoming edges of x are inside H, then we set $F = H \cup \{e\}$. Otherwise, we set $F = \phi(H)$. To show that F is a canonical fragment, we consider both cases separately.
   Let H be a maximal sequence and let x, without loss of generality, be an entry of H such that all outgoing edges of x are inside H. (The exit case is analogous.) We distinguish two cases.

1. An incoming edge of x is in H. Call this edge e_0. Then the sequence H can be divided into two segments S_1, S_2 such that $e_0 \in S_1$. Then $\phi(S_1)$ and $\phi(S_2)$ are both fragments. Moreover they are in sequence, that is, $F = \phi(S_1) \cup \phi(S_2)$ is a sequence. F must be a maximal sequence because otherwise H would not be a maximal sequence. Therefore, F is a canonical fragment.
2. No incoming edge of x is in H. As all outgoing edges of x are in H, H is in sequence with the trivial fragment $\{e\}$ in G'. Because H is a maximal sequence of G, $F = H \cup \{e\}$ is a maximal sequence of G'.

Now we consider the 'otherwise' case, i.e., $F = \phi(H)$. We know from part 2 of this lemma that F is a fragment. Suppose that F were not canonical. Then, there is some fragment F' of G' such that F and F' overlap. Therefore, none of three sets $F \setminus F'$, $F \cap F'$ and $F' \setminus F$ are empty. Lets call an edge f *original* if $f \neq e$. If all three sets $F \setminus F'$, $F \cap F'$ and $F' \setminus F$ contain an original edge, then $H = \psi(F)$ and $\psi(F')$ also overlap, which contradicts H being canonical. Therefore, we have to prove that none of the three sets $F \setminus F'$, $F \cap F'$ and $F' \setminus F$ equals $\{e\}$. To derive a contradiction we suppose that this is the case. It follows immediately that x must then be a boundary node of H. We assume without loss of generality that x is an entry of H.

We know from previous results [1], that there are only two ways in which two fragments F, F' can overlap:

1. F and F' are two non-maximal sequences that share a common segment.
2. F and F' are separation components w.r.t. the same boundary pair $\{u, v\}$ that share a common separation class w.r.t. $\{u, v\}$. (F and F' are then special *bond fragments* in the terminology of [1].)

We consider these two cases now separately.

2. Consider the case $F \cap F' = \{e\}$. The boundary pair of $\{e\}$ is $\{*x, x*\}$, which is therefore also the common boundary pair of F and F'. It follows that x is a separation point of $C(G)$, contradicting our assumption that $C(G)$ is biconnected. The other two cases use exactly the same argument.
1. Let F and F' be two non-maximal sequences that share a common segment.
   (a) If the shared segment is $\{e\}$, then $e \in F$ and because of the definition of ϕ, H contains an outgoing edge from x. Because $\{e\}$ and $F' \setminus \{e\}$ are two fragments in sequence, all the incident edges to $x*$ are in F', which contradicts that H contains an outgoing edge from x.
   (b) Let $F \setminus F' = \{e\}$, then $e \in F$ and because of the definition of ϕ, H contains an incoming edge to x. That edge must be inside $F' \cap F$, the overlapping segment of the two sequences. It follows that $*x$ is a boundary node of this segment in G'. As $x*$ is a boundary of that segment, this contradicts the assumption that $C(G)$ is biconnected.
   (c) Let $F' \setminus F = \{e\}$. As F does not contain e, we have $F = H$. As $F = H$ is a sequence by assumption, some outgoing edge of x is outside F (otherwise we would not be in the top-level 'otherwise' case). Call this edge e_0. By assumption $x*$ is an interior node of F'. Then e_0 must be in $F' \setminus F$, which contradicts the initial assumption of this subcase.