Chapter 3

Series – Parallel Digraphs

Introduction

In this chapter we examine series-parallel digraphs which are a common type of graph. They have a significant use in several applications that make them interesting to examine. Using sp graphs we can successfully visualize flow diagrams, dependency charts, and PERT networks. In the current document we give some preliminary information about series–parallel digraph, how it can be constructed and finally we present two algorithms one for drawing sp digraphs and another for labelling the decomposition tree's nodes.

A sp-graph is first of all a planar graph. As we can conclude of its name, it is a graph with serial and parallel components. In the real world a suitable representation is that of digital design circuits where logical AND gates imply series composition and OR gates parallel.

3.1 Definitions

Accepting that the simple – base case of a sp graph is that of Figure 3.1.a we can recursively define our term:

If $G_1, \ldots\ldots, G_n$ are sp digraphs so are the graphs obtained by the following operations:

1. The series composition of digraphs $G_1, \ldots\ldots, G_n$ with sources $s_1, \ldots\ldots, s_n$ and sinks $t_1, \ldots\ldots, t_n$ by identifying the sink t_i with the source s_{i+1} where $1 = i < k$ (Figure 3.1.d).
2. The parallel composition of digraphs $G_1, \ldots\ldots, G_n$ with sources $s_1, \ldots\ldots, s_n$ and sinks $t_1, \ldots\ldots, t_n$ by identifying $s_1, \ldots\ldots, s_n$ into a single vertex s and identifying $t_1, \ldots\ldots, t_n$ into a single vertex t (Figure 3.1.e).

The above definition can be understood in detail by studying Figure 3.1. In this figure we construct the parallel composition of the base case by joining the sources at the bottom and sinks at the top (see Figure 3.2.b). Similarly we construct the series composition by joining the source with the sink of the two base case graphs. In a same way we can compose more complex graphs like Figure 3.1.d and Figure 3.1.e cases.

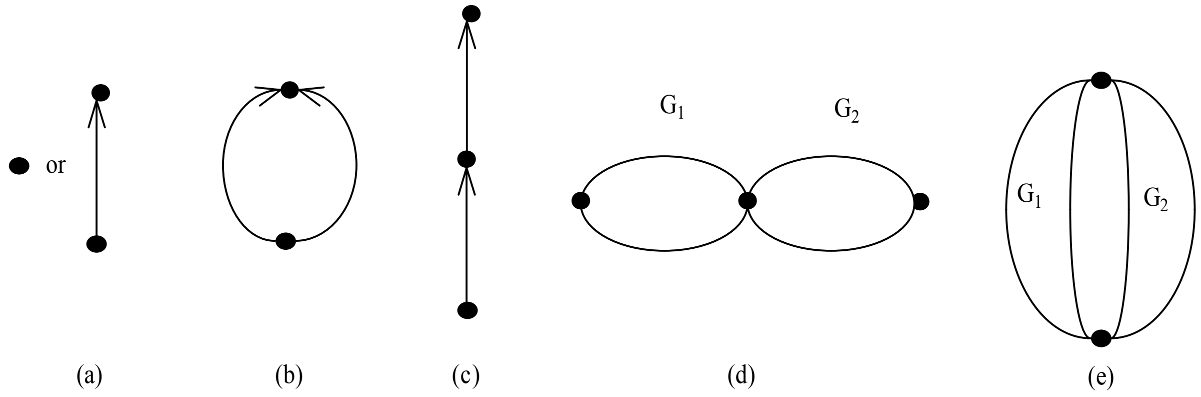

Figure 3.1: Recursive construction of sp graphs: (a) simple-base case, (b) parallel composition of "(a)", (c) series composition of "(a)", (d) general case series composition, (e) general case parallel composition

The composition operation is a divide and conquer technique. Our problem requires more complexity at the divide part. This is better explained below.

Here we adopt the upward drawing without multiple edges. So the source appears at the bottom and the sink at the top of the drawing as we can see at Figure 3.1. Only the graph in Figure 3.1.b doesn't satisfy this condition and we won't use it again. The sink and source of a series-parallel graph are also called poles.

3.2 Decomposition of a Series – Parallel Digraphs

We are now called to analyse the inverse operation of decomposing a series-parallel digraph. To do this we use a decomposition tree also called parse tree. Suppose we have a series – parallel graph *G* and its decomposition tree *T*. We assume that graphs are simple: with no self loops and multiple edges. In this tree there are exist three types of nodes: *s*-nodes to depict series composition, *p*-nodes for parallel composition and *q*-nodes for terminal base cases (see Figure 3.4). Its leaves are always *q*-nodes while the internal nodes are either *p*-nodes or *s*-nodes. An example of a graph's decomposition is given in Figure 3.2.

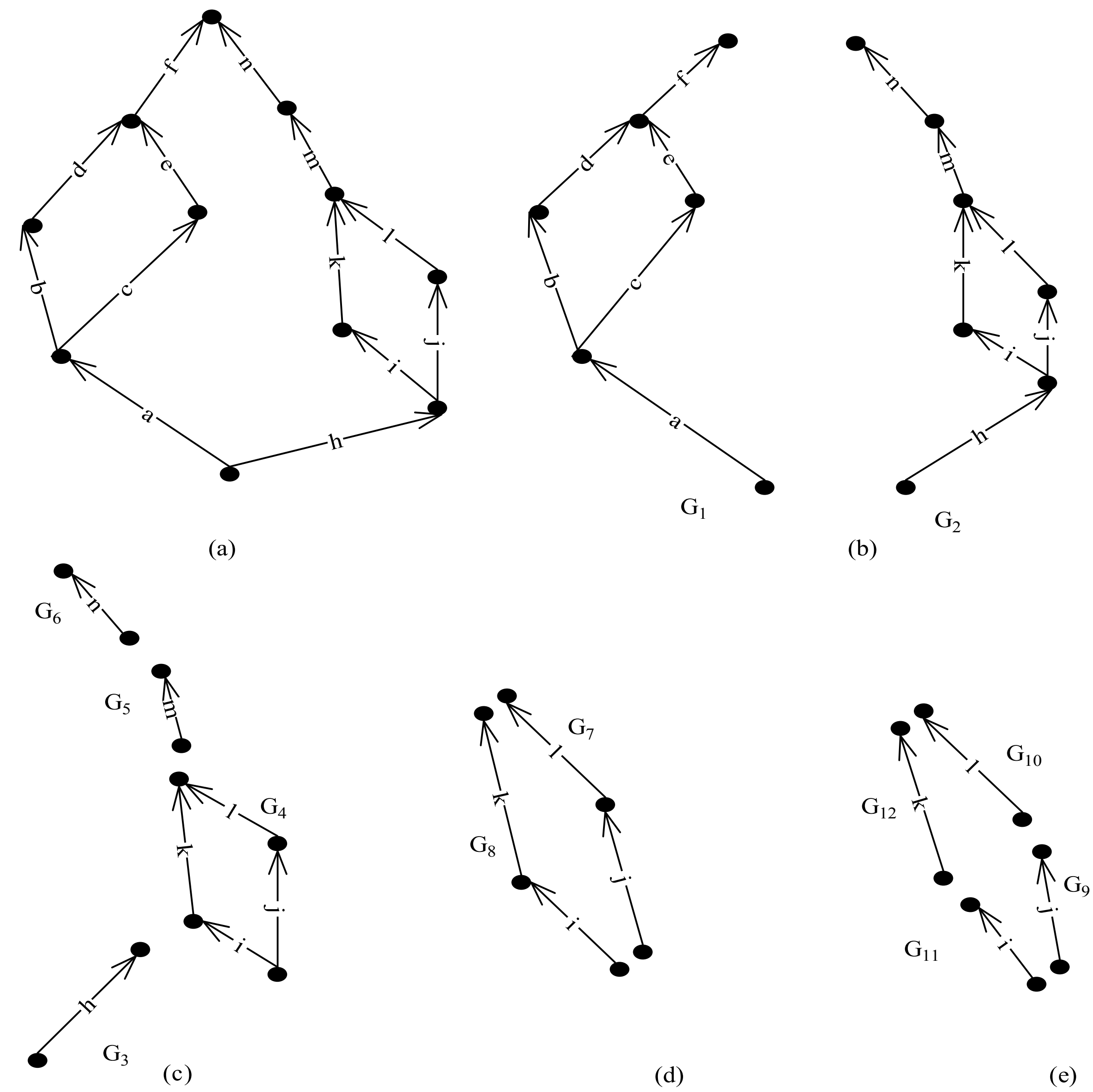

Figure 3.2: Decomposition of a sp graph

In the previous figure we see a sp digraph and its decomposition. We don't provide the whole decomposition, only a representative part of it, the right graph G_2. Starting from the graph's source we separate it into two components according to its composition type. In the current example we break it in the pieces seen in Figure 3.2.b as it has been generated from a parallel composition. The new right component (G_2) derives from a series composition. The components that have not resulted in the base case sp graph are further decomposed in the following figures (Figure 3.2.d, Figure 3.2.e). The decomposition that follows uses also decomposition sub-trees for every component-graph. The operation is terminated when all the sp graphs produced by decomposition, are basic-simple (see Figure 3.1.a). The complexity required for the final result is O(n).

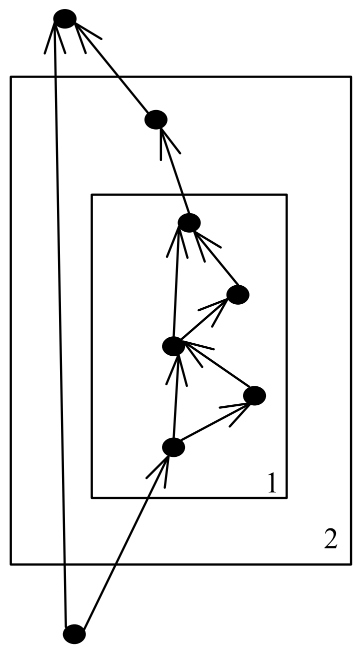

Figure 3.3: Closed and Open components

After constructing the decomposition tree there are two types of components which are defined as follow. If *C* is a maximal path of nodes of *T* of the same type, and let $\mu_1, \ldots\ldots, \mu_k$ are the children of the nodes of *C* that are not on *C*, from left to right. A *closed component* of *G* is either *G* or the composition of the series – parallel digraphs associated with a subsequence $\mu_i,\ldots\ldots,\mu_j$, where $1< i =j < k$ and *C* consists of *s*-nodes. An *open component* of *G* is either *G* or the composition of the series-parallel digraphs associated with a subsequence $\mu_i,\ldots\ldots,\mu_j$, minus its poles where $1=i =j = k$. A component is either an open or a closed component.

In the previous figure we can see an open and a closed component. Rectangle 1 describes a closed component that consists of a series composition of two sp digraphs. On the other hand rectangle 2 describes an open component consisting of a series of nodes except its poles.

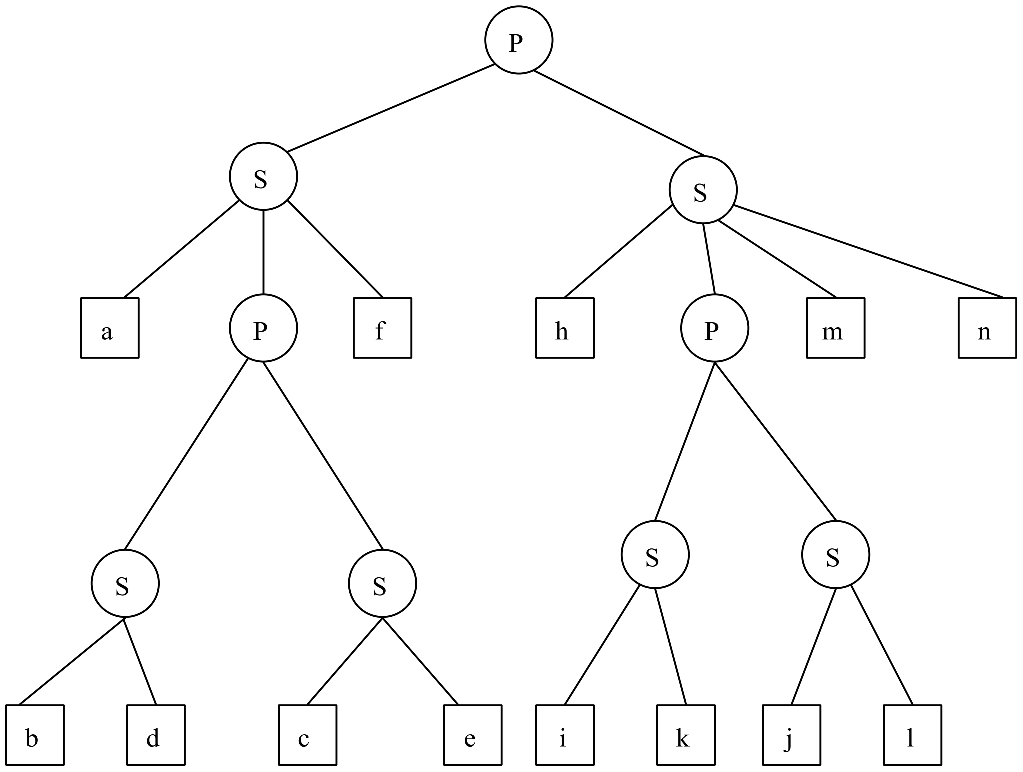

Figure 3.4: The decomposition tree of the sp graph displayed in Figure 3.2.a

To construct the decomposition tree we start from graph's source. By examining the composition that has used this node we conclude to root's type. This means that we think backwards at the composition stage where we had conjunction of two nodes in one (series) or creation of a new node and joining of two nodes in the one new (parallel). So depending on the origin of the node we give him its type. In the figure's example the source of the graph comes from a parallel composition by joining the two sources of sp graphs into a new one. Therefore the root of our tree is a *p*-node. As for its children, they are sub trees that are examined in the same way. So we conclude that the right sub-tree represents a series composition of the graph. Following the same algorithm we analyse the graph as far as we find *q*-nodes, always according to the definition of the sp graph.

3.3 An algorithm for Drawing Series – Parallel Digraphs

The previous operations of decomposition in sp graphs provide us with the decomposition tree *T* which has all the necessary information of sub sp graphs and the compositions between them. This fact gives us the ability to draw the graph from scratch having as sole input its decomposition tree *T*. In practice, it happens some series-parallel graphs to require exponential area for their straight – line upward drawing. In this case there is a need to change its embedding. Having the components of the whole graph we can choose another embedding that serves our purposes.

We will use the algorithm ? -SP Draw to provide us with a drawing of $O(n^2)$ area. To do this we consider the decomposition's tree components. The algorithm uses as basic characteristic a right-angled isosceles triangle (Figure 3.5), inside which there is the drawing of our series – parallel graph. So every series – parallel graph can be illustrated inside this triangle.

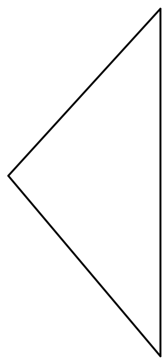

Figure 3.5: a right-angled isosceles triangle

This operation is better understood by starting from the simplest case and continuing using a recursive approach. The way the graphs are put into the bounds of triangle is shown in Figure 3.6.

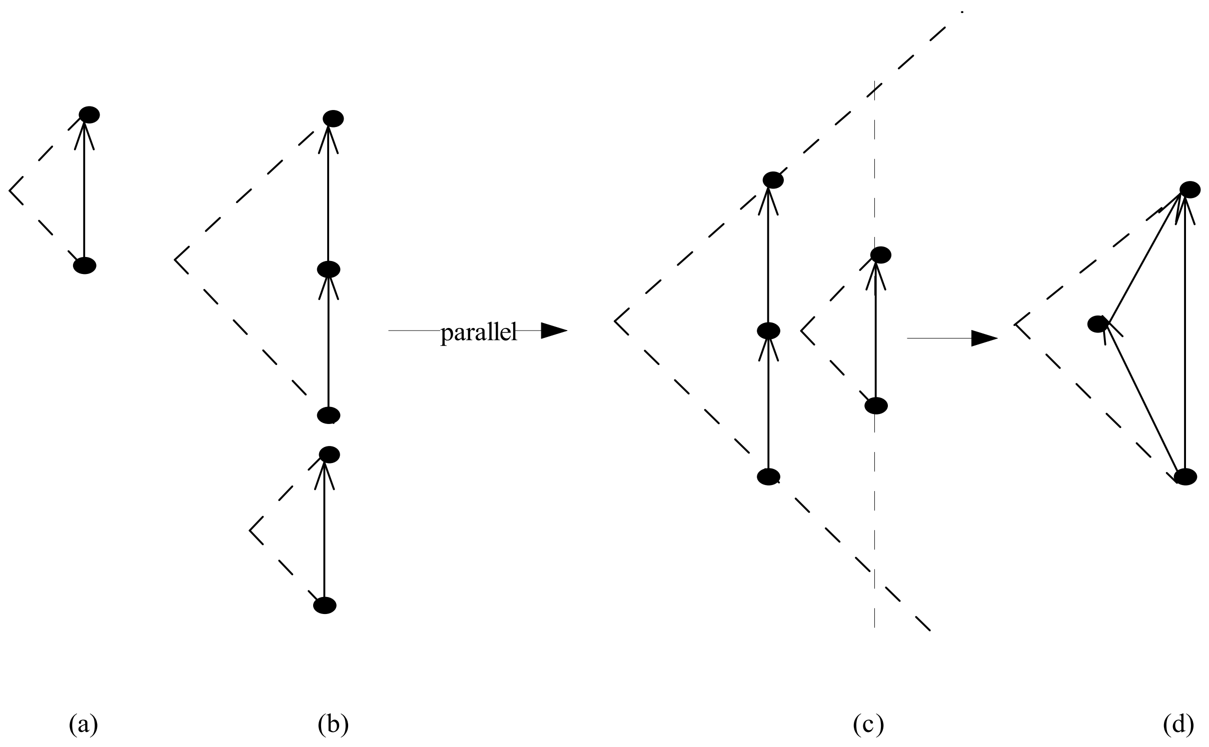

Figure 3.6: applications of algorithm ? -SP Draw

We place the transitive edge along the hypotenuse as we want a right pulsed embedding. While drawing a simple-base sp digraph we just put the edge onto the hypotenuse with the source at the bottom and the sink at the top. In order to have a parallel composition of two graphs (Figure 3.6.b) we have the placement of Figure 3.6.c. Note that the transitive edge is always on the right. The result appears at Figure 3.6.d. A more general case of parallel composition is illustrated in Figure 3.7.

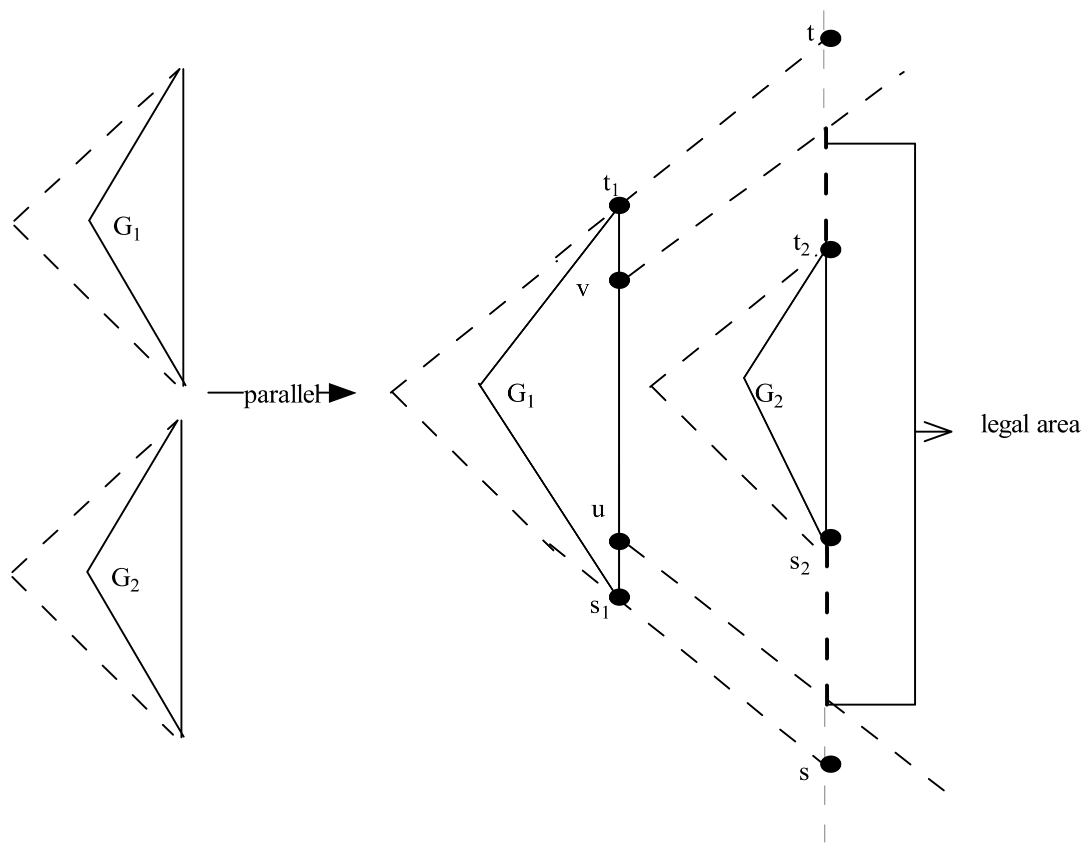

Figure 3.7: A more general parallel composition of sp digraphs

In the general case of the parallel composition the result is also in a special triangle. As it happens in every parallel composition the *s* (source node) of this triangle is the joint of s_1 and s_2 and *t* (sink) is the joint of t_1 and t_2. An additional constraint that we should notice, is to place the G_2 somewhere in the legal area so that we won't have any cross edges.

In series composition the two triangles are put next to each other as in Figure 3.8. They share a node which is the source of the top graph and the sink of the bottom. The result of this composition is a new triangle with sink the sink of the top and source the source of the bottom. To complete the rest of the shape we extend the two sides of the triangles.

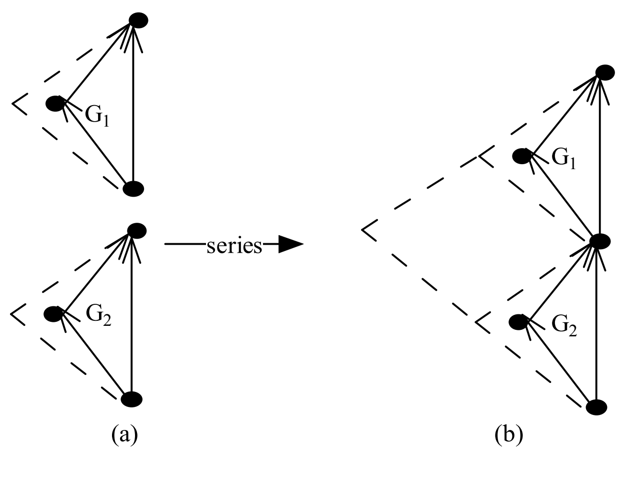

Figure 3.8: Series composition

All the operations that we have mentioned are applied in the below algorithm that gives us the drawing of any sp graph in a right-angled isosceles triangle.

? – SP Draw

Input: a series – parallel digraph G
Output: a strictly upward planar straight – line grid triangle G of G

1. Compute a decomposition tree T of G.

2. Modify the embedding of G into a right-pushed embedding and perform the corresponding modifications on T.

3. If G consists of a single edge (base case sp graph), it is drawn as a vertical segment of length 2, with bounding triangle having width 1 (see Figure 3.6.a)

4. If G is the series composition of G_1 and G_2, the two drawings G_1 and G_2 of G_1 and G_2 are first recursively produced (*divide*). Then (*conquer*), G is drawn by translating G_2 so that the sink of G_1 is identified with the source of G_2 (see Figure 3.8). The bounding triangle $?(G)$ is obtained by extending the bottom side of $?(G_1)$ and the top site of $?(G_2)$.

5. If G is the parallel composition of G_1 and G_2. The two drawings G_1 and G_2 of G_1 and G_2 are first recursively produced (*divide*). Then (*conquer*), we consider the rightmost edges (s_1, u) and (v, t_1) incident on the source and sink of G_1, respectively (see Figure 3.7). The G_2 drawing should be put between the line of the legal area, the line through u that is parallel to the bottom side of the bounding triangle G_1, the line through v that is parallel to the top side of the triangle of G_1 and the hypotenuse of G_1. Then, we identify the sources and sinks of G_1 and G_2 by moving them to the intersections s and t of the base of $?(G_2)$ with the lines extending the top and bottom sides of $?(G_1)$, respectively.

In the third step the length of the hypotenuse is 2 so that the area of the triangle will be 1 ((hypotenuse * height) / 2).

After we have computed the decomposition tree of the sp graph G, which is algorithm's input we work on the root of T. If this node is a q-node (G consists of a single edge) the algorithm returns. In any other case there is a recursive call of the algorithm with argument first the right child and then the left; this is an arbitrary order. Practically we apply the algorithm on the two or more sub-trees of the root. Following the same procedure for each sub-tree we draw first simple sp graphs and then more complex. The final output when every procedure returns is the required G drawing.

For better understanding an example is provided next.

Suppose that we have as input the digraph of Figure 3.2.a. The execution of the algorithm until a depth will be as follows:

1. First we construct the decomposition tree shown in Figure 3.4
2. At the current example our graph has no transitive edge so this step is never executed.
3. It's not this case so we pass at step 5
   Recursive call of the algorithm for graph G_2 and G_1
   1. decomposition tree of G_2
   2. ….
   3. not this case
   4. it is a series call algorithm for G_3, G_4, G_5, G_6

```
      1.  decomposition tree of G_3
      2.  ….
      3.  draw the triangle, return (Figure 3.9.a)
   1.  decomposition tree of G_4
   2.  …..
   3.  not this case
   5.  it is a parallel composition call algorithm for G_7, G_8.
            1. decomposition tree of G_7
            2. …..
            3. not this case
            4. it is a series call algorithm for G_9, G_10
                1. decomposition tree of G_9
                2. …
                3. draw the triangle, return (Figure 3.9.b)

                1.  decomposition tree of G_10
                2.  ….
                3.  draw the triangle, return (Figure 3.9.c)

            conquer step → series draw of triangle (Figure 3.9.d)

            1.  decomposition tree of G_8
            2. …..
            3. not this case
            4. it is a series call algorithm for G_11, G_12

                1. decomposition tree of G_11
                2. …
                3. draw the triangle, return (Figure 3.9.e)

                1. decomposition tree of G_12
                2.  ….
                3. draw the triangle, return (Figure 3.9.f)
                        .
                        .
                        .
```

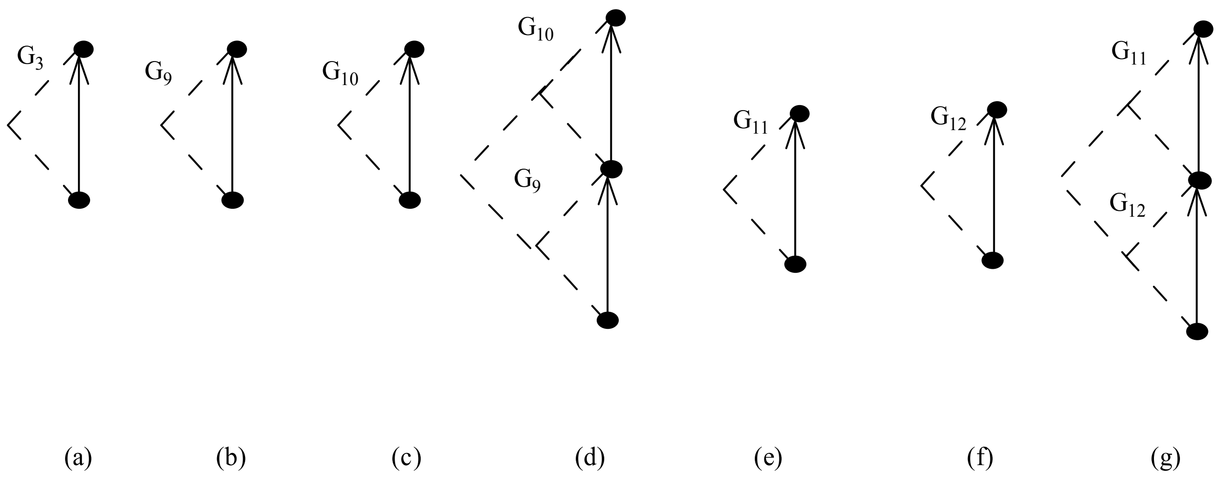

Figure 3.9: Steps of the algorithm

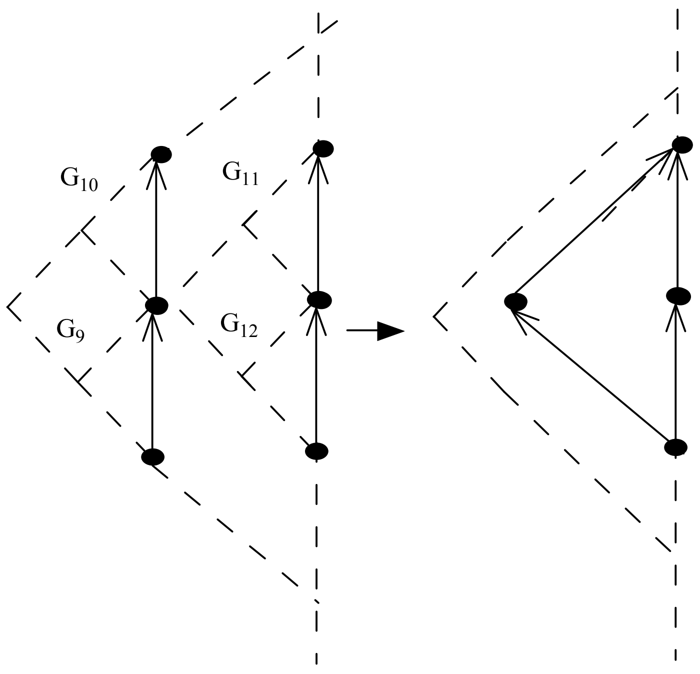

Figure 3.10: Parallel composition of two sp digraphs

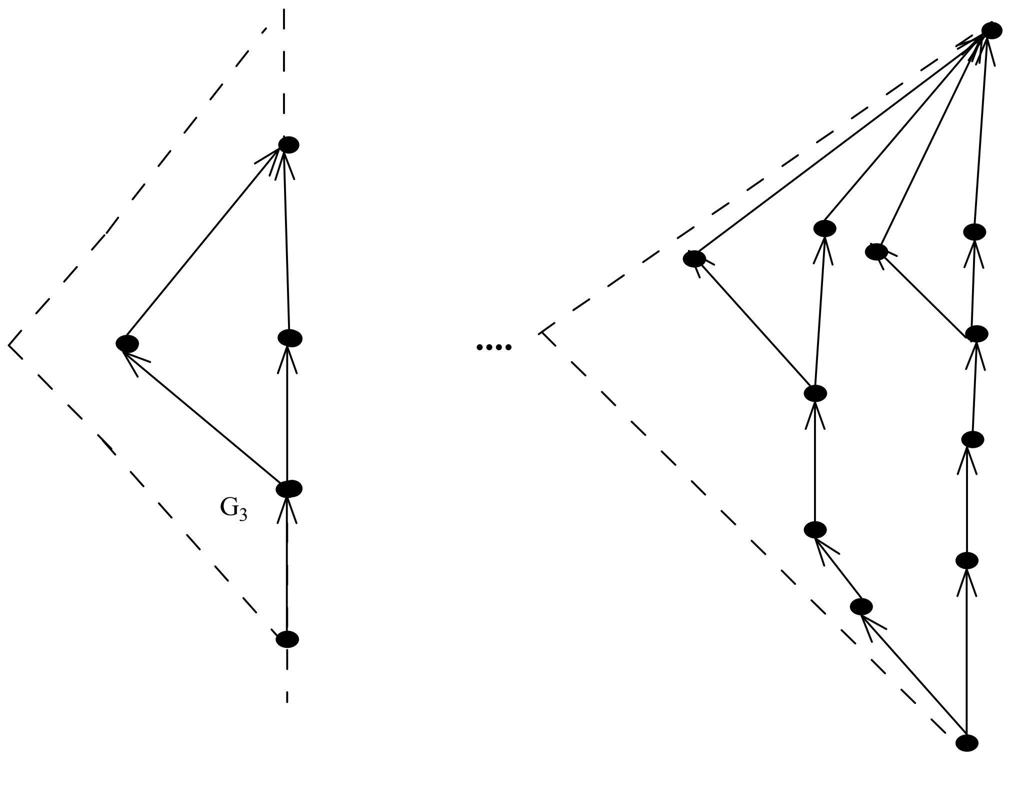

Figure 3.11: Final ? -SP Draw of our graph *G*

The above figures describe the algorithm steps to the final output in Figure 3.11.

The correct operation of the algorithm guarantees the following invariants of every right angled triangle ? (G).

Invariants

a. The graph's drawing is contained into the triangle without filling its critical vertex (right-angled). Moreover the embedding is constructed in a way such that has the transitive edges on the right just like the hypotenuse of the triangle.

b. The source is depicted by the bottom vertex of the triangle whereas the sink by the top.

c. For any adjacent vertex to the source there should be satisfied a condition: the wedge with vertex in this node and rays with slopes –p/2 and –p/4 does not contain any vertex of *G* except *s*. (see Figure 3.12.a)

d. A similar condition should be satisfied for the sink *t* of *G* too. In this case the wedge will have rays with slopes p/2 and p/4. (see Figure 3.12.a)

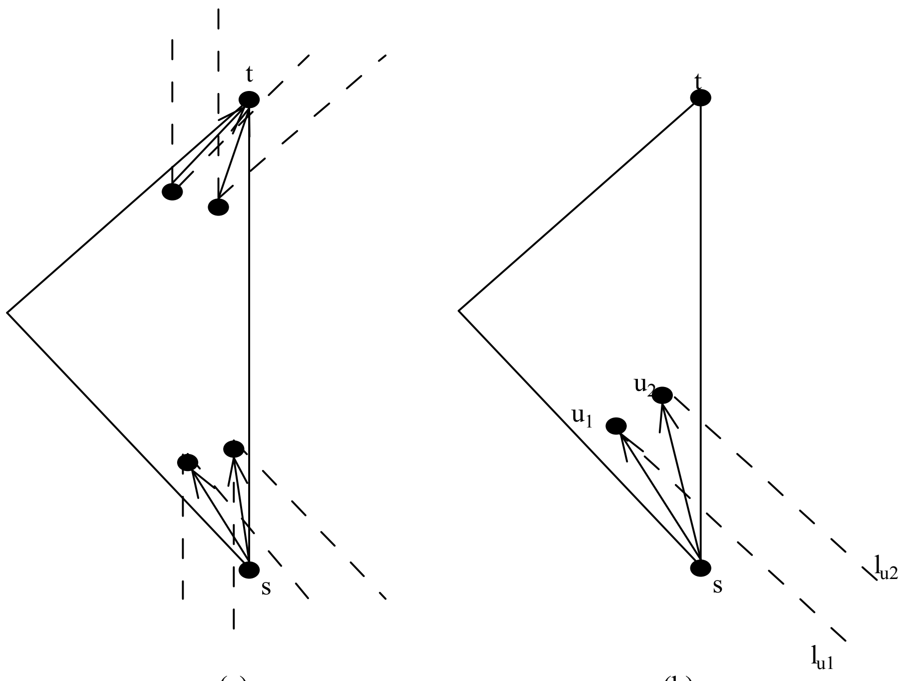

Figure 3.12: Properties of nodes adjacent to poles

Lemma 3.1 *Let u_1 and u_2 be neighbours of the source vertex s, such that edge (s, u_1) is to the left of edge (s, u_2), and let l_{u1} and l_{u2} be the rays of slope –p/4 originating at u_1 and u_2, respectively. If Invariant (c) holds, then l_{u1} is below l_{u2}.*

Proof. Invariant (c) says that the wedge seen at Figure3.12.a contains only vertex *s*. This means that the rays with slopes -p/4 of both u_1 and u_2 are parallel. So one of them will be above the other. The one that will be above is that of the most right vertex(u_2) since otherwise u_2 would be contained into the wedge of u_1 and invariant (c) would hold.

It is interesting to see how the sp graph's invariants are preserved during the compositions. It is obvious why the two first invariants always hold in both kinds of composition.

In series composition invariants (c) and (d) are satisfied since the poles of the composed drawing remain intact. This is because the source of the new drawing is the source of one of the olds and the sink is the sink of the other old. Therefore the properties of the poles are inherited from the components' drawings, since the adjacent vertices in both poles are the same as before.
In parallel composition these invariant are guaranteed by the correct placement of the right graph inside the bounds. This allowable area is better described by Figure 3.7, where u and v are the rightmost vertices which are adjacent to the source and the sink respectively. So invariants (c) and (d) are satisfied.
However this bounded area where we can put the right graph in a parallel composition should be better defined. This purpose serves the ? -SP-Label algorithm. A simple idea is put the G_2 so that the right angled vertex of it to be somewhere across the hypotenuse of G_1.This doesn't produce any problem since that vertex is never occupied by any node. Using this technique we result in a drawing with $O(n^2)$ area, as it derives from the triangle's properties. Considering now the recursive operation of the ? – SP Draw algorithm in a graph *G* we reach the following theorem.

Theorem 3.1 *Let G be a series – parallel digraph with n vertices. Algorithm ?-SP Draw produces a strictly upward planar straight-line grid drawing of G with $O(n^2)$ area such that isomorphic components of G have drawings congruent up to a translation.*

The algorithm that follows computes some parameters of the nodes of a decomposition tree that will allow us to specify the appropriate and exact placement of the components represented by these nodes, during a composition.
For more details see the algorithm that follows.

?-SP-Label

Input: decomposition tree of *T* of a series-parallel digraph *G*
Output: labeling of each sub-trees of *T* with values b, b', and b''

```
1 if the root of T is a Q-node
2 then
3       $b(T) = b''(T) = b'(T) = 2$ (see Figure 3.13.a)
4 else
5       let $T_1$ and $T_2$ be the left and right sub-trees of T, respectively
6       for each i=1, 2 do
7               ?-Sp-Label($T_i$)
8       if the root of T is an S-node (see Figure 3.13.c)
9       then
10              $b(T) = b(T_1)+ b(T_2)$
11              $b'(T) = b'(T_1)$
12              $b''(T) = b''(T_2)$
13      else (the root of T is a P-node)
14              $b(T) = b(T_1)+ b(T_2) + 2D_x$ (see Figure 3.13.b)
15              if $T_2$ is a Q-node (transitive edge)
16              then
17                      $b''(T) = b'(T) = b(T)$
18              else
19                      $b''(T) = b(T_1) + 2D_x - D_y + b''(T_2)$
20                      $b'(T) = b'(T_2) + D_y$
```

The algorithm works recursively starting from the root of the decomposition tree *T* that takes as input. Its role is to label every node of this tree according to its type. Therefore if the node of the root of the tree that examines each time is a q-node it labels this node giving him value "2" in all of it's parameters. These parameters are defined through the Figure 3.13.

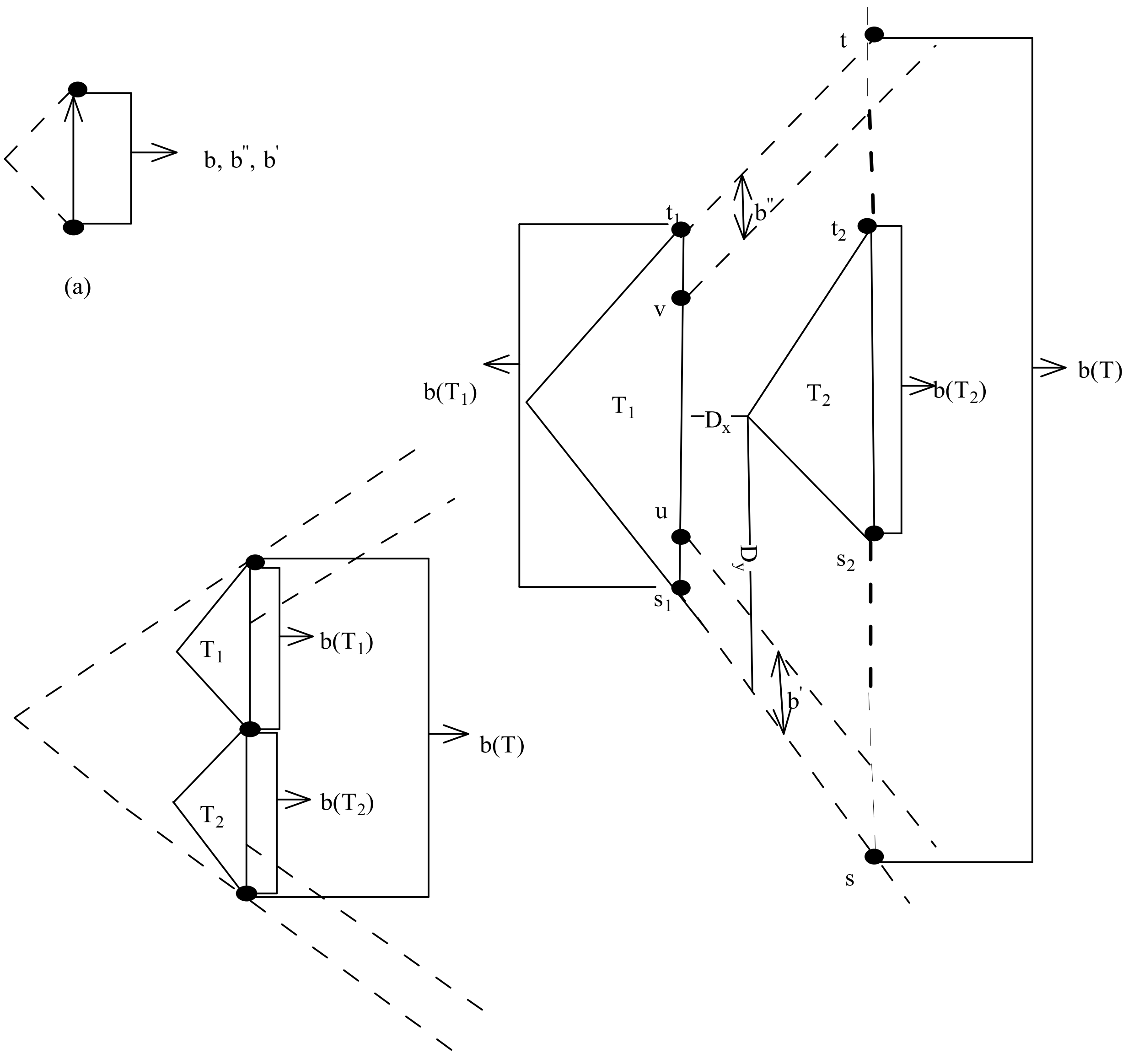

Figure 3.13: (a) Labels in q-nodes, (b) labels in p-nodes and (c) labels in s-nodes

In the other cases of q-nodes or p-nodes we have different values that valuated by the geometry of our drawing. In parallel case (line 13) the equation $b(T) = b(T_1) + b(T_2) + 2D_x$ is produced by the triangle's properties: $b(T)$ = 2*(height of *T*)= 2*(height of T_1 + height of T_2 + D_x). Equations in lines 19, 20 are also proofed by the geometry of our shape.

By examining the way this algorithm works we can conclude the following theorem.

Theorem 3.2 *Algorithm ?-SP Draw can be implemented to run in O(n) time and space on a series – parallel digraph with n vertices.*