

Available online at www.sciencedirect.com**ScienceDirect**

Journal of Discrete Algorithms 4 (2006) 649–675

**JOURNAL OF
DISCRETE
ALGORITHMS**www.elsevier.com/locate/jda

Average case analysis of DJ graphs

Johann Blieberger*Department of Computer-Aided Automation (183/1), Technical University of Vienna, Treitlstr. 1,
A-1040 Vienna, Austria*

Available online 16 September 2005

Abstract

Sreedhar et al. [V.C. Sreedhar, G.R. Gao, Y.-F. Lee, A new framework for elimination-based data flow analysis using DJ graphs, *ACM Trans. Program. Lang. Syst.* 20 (2) (1998) 388–435; V.C. Sreedhar, Efficient program analysis using DJ graphs, PhD thesis, School of Computer Science, McGill University, Montréal, Québec, Canada, 1995] have presented an elimination-based algorithm to solve data flow problems. A thorough analysis of the algorithm shows that the worst-case performance is at least quadratic in the number of nodes of the underlying graph. In contrast, Sreedhar reports a linear time behavior based on some practical applications.

In this paper we prove that for goto-free programs, the average case behavior is indeed linear. As a byproduct our result also applies to the average size of the so-called dominance frontier.

A thorough average case analysis based on a graph grammar is performed by studying properties of the *j*-edges in DJ graphs. It appears that this is the first time that a graph grammar is used in order to analyze an algorithm. The average linear time of the algorithm is obtained by classic techniques in the analysis of algorithms and data structures such as singularity analysis of generating functions and transfer lemmas.

© 2005 Elsevier B.V. All rights reserved.

Keywords: Average case analysis; Data flow analysis; Dominator tree; DJ graphs; Dominance frontier

E-mail address: blieb@auto.tuwien.ac.at (J. Blieberger).

URL: <http://www.auto.tuwien.ac.at/~blieb>.

1570-8667/\$ – see front matter © 2005 Elsevier B.V. All rights reserved.

doi:10.1016/j.jda.2005.07.002

1. Introduction

Program analysis is a process of estimating properties of programs at each program point. The information provided by program analysis is useful in compiler optimization, code generation, program verification, testing and debugging, and parallelization. In general, program analysis can be divided into: *control flow analysis* and *data flow analysis*. Both methods are usually performed on a graph representation of a program called the Control Flow Graph (CFG). The nodes in a CFG represent basic blocks or statements, while edges of the graph represent flow of control from one basic block to another.

As an example Fig. 2 shows the CFG of the program fragment given in Fig. 1. Node 3 is the if-statement. The edge to node 4 is the then-branch and is followed only if $c1$ is *true*. The edge $3 \rightarrow 2$ has assigned condition $\neg c1 \wedge \neg c3$ and the edge $3 \rightarrow 6$ has assigned $\neg c1 \wedge c3$. In a similar way edges $5 \rightarrow 4$, $5 \rightarrow 2$, and $5 \rightarrow 6$ have assigned conditions $c2$, $c2 \wedge \neg c3$, and $c2 \wedge c3$, respectively. Edge $1 \rightarrow 6$ is only present to facilitate algorithms performed on the CFG and has assigned *false*. All the other edges have assigned *true*.

In literature, control flow problems are typically solved using concepts from graph theory, whereas data flow problems are typically solved using concepts from set theory (or more precisely, lattice theory).

begin	-- Node 1
repeat	-- Node 2
if c1 then	-- Node 3
repeat	-- Node 4
...	-- Node 4
until c2	-- Node 5
endif	
until c3	-- Node 5
end	-- Node 6

Fig. 1. Example: Program source code.

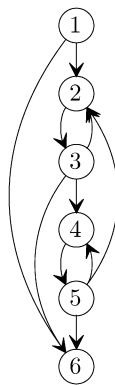


Fig. 2. Example: Control flow graph.

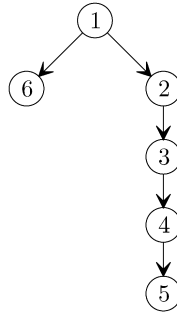


Fig. 3. Example: Dominator tree.

$$\begin{aligned}
 X_1 &= \perp \\
 X_2 &= f(X_1, X_3, X_5) \\
 X_3 &= f(X_2) \\
 X_4 &= f(X_3, X_5) \\
 X_5 &= f(X_4) \\
 X_6 &= f(X_1, X_3, X_5)
 \end{aligned}$$

Fig. 4. Example: Data flow equations.

An example of a control flow analysis is computing the *dominance relation*. Given a CFG, a node u is said to dominate another node v if all paths from the start node to node v always pass through node u , e.g., in Fig. 2 node 2 dominates nodes 3, 4, and 5, but not 1 and 6. The dominance relation can be visualized by a dominator tree [3]. The dominator tree of our example is shown in Fig. 3.

An example of a data flow analysis is the *reaching definitions problem*. The reaching definitions problem is to determine which definitions in a program reach a given point. (A definition of variable a occurs when a is assigned a value.)

A data flow problem can be represented within a framework, called the *data flow framework*. Within this framework we represent data flow information as elements of a lattice, and the effect of a node (a statement or a basic block) as a data flow function. The input–output effect of a node can be represented as a data flow equation, and so we can set up a *system of data flow equations*, one equation per node, whose consistent solution gives the desired estimate of the program property. Fig. 4 shows data flow equations of our example. Symbol \perp denotes a function which does not depend on X_1, \dots, X_6 .

The methods for solving the system of equations can broadly be classified into *iteration methods* and *elimination methods*. Iteration methods are easy to implement but cannot handle all data flow problems. A prominent example which cannot be solved by iteration methods is determining the *worst-case execution time* (WCET) of a program [5]. Elimination methods are derived from Gaussian elimination method for solving simultaneous equations [14]. In general, elimination methods are more complex to implement than iteration methods [17].

Sreedhar et al. [18,19] have presented an efficient and easy to implement elimination-based algorithm to solve data flow problems. The algorithm starts with a general (reducible) directed CFG G . The union of G and the dominator tree of G is called a *DJ graph*. The data flow problem is solved by redirecting and removing edges in the DJ graph until the remaining graph is the dominator tree of G . Because the dominator tree is part of the DJ graph, each node can be assigned a certain level (equal to its distance from the root).

Edges being part of the dominator tree and not being part of G are called *d-edges*. Edges being part of the dominator tree and of G are called *dj-edges*. The remaining edges are *j-edges*.¹

Three different operations are performed in a bottom-up fashion on the graph by the algorithm: *Eager1*, *Eager2a*, and *Eager2b*.

Sreedhar et al. [18,19] give a thorough worst-case performance analysis of the algorithm showing that the number of Eager (*Eager1* + *Eager2a* + *Eager2b*) operations is at most $O(e \cdot n)$ where n denotes the number of nodes and e denotes the number of edges in G . A more detailed description and analysis of Sreedhar's algorithm and how the DJ graph can be used to solve the underlying system of equations, can be found in Section 2.

In contrast, Sreedhar reports a linear, i.e., $O(e)$, time behavior based on some practical application programs.

In this paper we *prove* that for goto-free programs, the average case behavior is indeed linear. By “goto-free” programs we mean programs written in programming languages without a goto statement like *Modula-2* [22] and *Java* [1] or programs not using goto statements or statements with similar effects (cf. [9]). Some programming languages allow to exit loop statements not only at the beginning (while-loops) and at the end (repeat-loops) of loop-statements, but also at certain points within the loop body. Exit-statements are a form of “tamed” goto-statements, which while retaining structured programs, give more freedom to the programmer and often result in more readable and understandable program code. Our analysis covers such *exit-statements*, too. As a byproduct our result also applies to the average size of the so-called *dominance frontier* [7]. The *dominance frontier* $DF(u)$ of a CFG node u is defined as the set of all CFG nodes v such that u dominates a predecessor of v but does not strictly dominate v .²

Cytron et al. [7] have proved that if a program contains only straight-line code, while loops, and if statements, the number of Eager operations is linear. Their result can easily be reproduced by our approach (cf. end of Section 4).

In Section 2 we present a graph grammar which derives DJ graphs for goto-free programs. The number of these DJ graphs (or actually the probability of DJ graphs) with n nodes is determined in Section 3. Finally, the number of *Eager2b* operations is studied in Section 4.

¹ Sreedhar et al. [18,19] only introduced d- and j-edges; we have defined dj-edges in order to facilitate the description of our graph grammar in Section 2.

² Node x strictly dominates y if x dominates y but $x \neq y$.

2. A graph grammar for DJ graphs

In this section we define a graph grammar³ [16] for deriving DJ graphs. A tabular form of the graph grammar can be found in [Appendix A](#). The context-free grammar consists of the non-terminal nodes S , T_0 , and T , of one terminal node (\bigcirc), and of one terminal edge (denoted as usual by a directed arrow). Edges are labeled by d , j , and dj (cf. [18]). In detail, d - and dj -edges form the dominator tree of the control flow graph; j -edges are part of the control flow graph, but not contained in the dominator tree; d -edges are not part of the control flow graph, they are usually generated when the dominator tree is constructed. Our graph grammar builds the dominator tree concurrently to the control flow graph. This can also be used to build the dominator tree in linear time while parsing the program source which otherwise requires sophisticated algorithms (cf. [2,11,15]).

In the second column of [Table A.1](#) we give a textual (BNF-like) description of the production. The third column is the right-hand side (rhs) of the production; the left-hand side (lhs) is simply S for the first two productions, T_0 or T for the third production, and T for the rest. From the non-terminal T_0 in the second column only straight-line code can be derived, i.e., it is mapped to a \bigcirc -node in the third column.

Since the lhs of the productions consists only of one non-terminal, we use a special notation for the embedding relation:

- (1) If terminals and/or non-terminals of the rhs are labeled by a \bullet , this means that all edges originating from the lhs non-terminal are adjoined to all \bullet -nodes on the rhs,⁴ if the production is applied.
- (2) All edges pointing to the lhs non-terminal are adjoined to the uppermost node (root node) of the rhs, which is a \bigcirc -node in all cases.

Note that the two cases do not exclude each other, i.e., there exist root nodes which are labeled by a \bullet .

The number of productions is approximately four times as big as for standard (non-graph) grammars. There are several reasons for this:

- To avoid chains of single-entry/single-exit \bigcirc -nodes, we had to duplicate and slightly modify certain productions for loop-statements by pre-pending a \bigcirc -node to the root node.
- Most productions have two forms. One for the “normal” case, where other statements “follow”, and one for the “pathological” case, where no other statements “follow”. For example, the last production used within the then-branch of an if-statement (before the else-branch starts) is such a “pathological” case.

Finally the fourth column contains the counting expressions for the productions, which are set up in [Section 4](#).

³ To the author’s knowledge this is the first time that a graph grammar is used for analyzing an algorithm.

⁴ If several \bullet -nodes exist on the rhs, edges are duplicated accordingly.

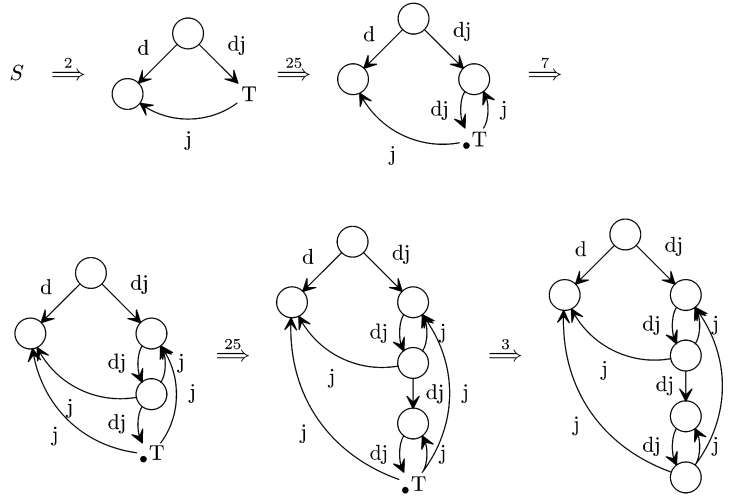


Fig. 5. Example: Derivation of DJ graph.

The graph grammar models straight-line code, if-statements, and all possible loop structures (general, while, and repeat). Each of these loops may be terminated prematurely by an exit-statement.

It should be possible to model multiple exits per loop without violating the analysis based on [Theorem 3](#) and still have a linear average number of Eager2b operations. For space considerations, however, we do not model multiple exit-statements in this paper.

As an example, consider the program fragment shown in [Fig. 1](#). It corresponds to the graph derivation given in [Fig. 5](#), where the number of the production is written above the \Rightarrow .

To illustrate one derivation step consider the application of production 7: we have to replace the dotted non-terminal T with the graph sentential number 7 given in [Table A.1](#). Both the \bigcirc -node and the non-terminal T in this sentential are dotted. This means that all edges originating in the non-terminal T before the replacement have to be duplicated and redirected such that they originate at the dotted nodes after the replacement. In our example these are two j -edges, one pointing to the node above and one to the node at the left above.

Theorem 1. *The graph grammar given in [Table A.1](#) correctly generates DJ graphs for goto-free programs.*

Proof. If in [Table A.1](#) we consider d - and dj -edges only (thereby ignoring all j -edges), only trees can be derived. It is easy to see that by adding j -edges in the way it is done in [Table A.1](#), the above mentioned tree is the dominator tree of the resulting graph. Hence, the graph grammar of [Table A.1](#) generates DJ graphs. \square

Sreedhar's algorithm—as exemplified in [Fig. 6](#)—performs three different operations in a bottom-up fashion on the graph:

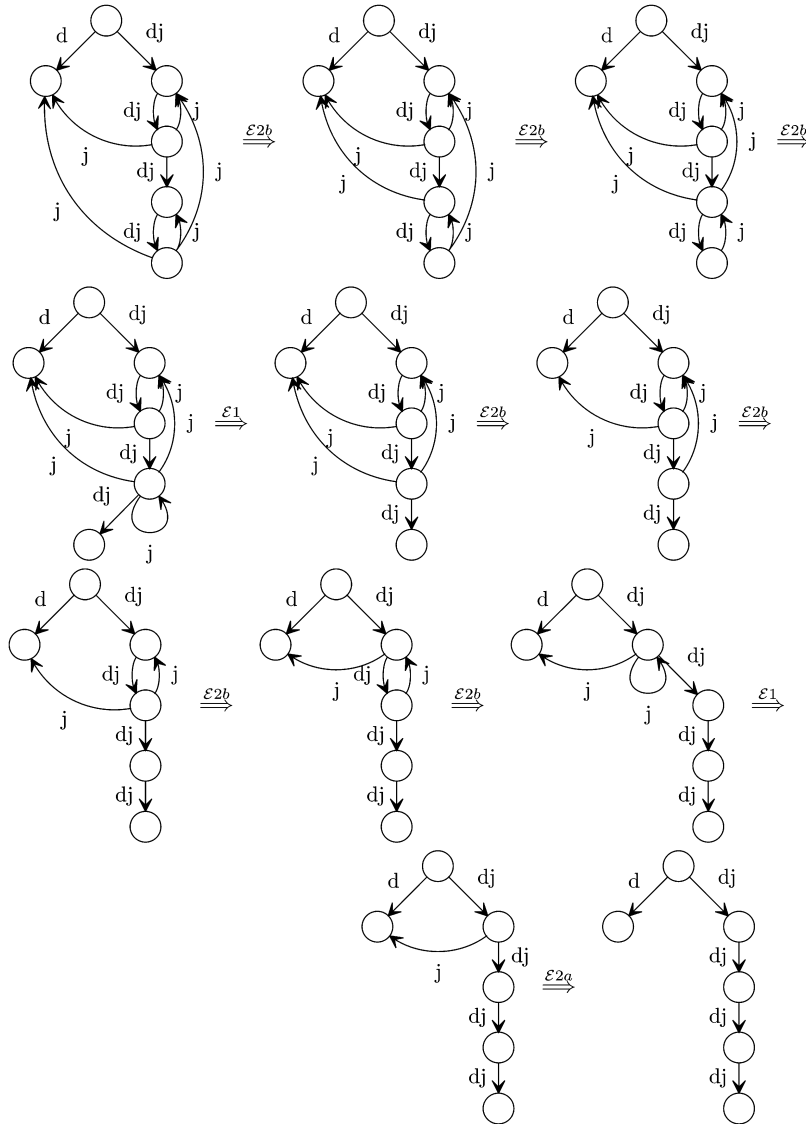


Fig. 6. Sreedhar's algorithm performed on example graph.

Eager1: Removes self-loops.

Eager2a: Removes a j -edge if both its source and target ($\text{source} \neq \text{target}$) have the same level.⁵

⁵ The level of a target of a j -edge is smaller or equal to the level of the source of the j -edge [18].

Eager2b: If the source and the target of a j -edge have different levels, the source of the j -edge is moved up one level to the root along a d - or dj -edge (the level of the source node is now one less than before).

If it happens as a result of this operation that two j -edges coincide, i.e., have the same source and target node, then one of them is removed.

These operations are repeated until no j -edges exist in the remaining graph, which is the dominator tree of the original CFG.

Note that in our example the DJ graph resulting from the five derivation steps has 6 nodes and Sreedhar's algorithm (as shown in Fig. 6) performs 2 Eager1, 1 Eager2a, and 7 Eager2b operations on this graph. Note also that the resulting DJ graph is the union of the CFG (cf. Fig. 2) and the dominator tree (cf. Fig. 3).

In analyzing Sreedhar's algorithm it is clear that the number of Eager1 and Eager2a operations is bounded above by e , the number of edges of the original CFG, because each edge can be deleted at most once. Therefore the interesting quantity is the number of Eager2b operations. In the worst case the number of Eager2b operations performed on one edge is bounded by n , the number of nodes in the original CFG. Thus the overall number of Eager2b operations is $O(n \cdot e)$.

The rest of this paper is devoted to the proof that for goto-free programs the average number of Eager2b operations is linear in e .

We conclude this section by noting that the Eager operations determine a sequence of operations to solve the underlying system of simultaneous data flow equations:

- Whenever a j -edge $u \rightarrow v$ is treated by an Eager2a or Eager2b operation, equation X_u has to be substituted into equation X_v .
- Whenever an Eager1 operation is performed on an edge $r \rightarrow r$, the recursive equation $X_r = f(\dots, X_r, \dots)$ has to be solved such that the rhs of the solution $X_r = f'(\dots)$ does not depend on X_r . This is called a *loop breaking operation* [14] and is denoted by \emptyset in this paper.

Our example produces the following sequence of operations: $5 \rightarrow 6$, $5 \rightarrow 2$, $5 \rightarrow 4$, $4\emptyset$, $4 \rightarrow 6$, $4 \rightarrow 2$, $3 \rightarrow 6$, $3 \rightarrow 2$, $2\emptyset$, and $2 \rightarrow 6$.

As a final step equations have to be substituted along the edges of the remaining dominator tree to obtain the solution of the data flow equations, i.e., in our example $1 \rightarrow 6$, $1 \rightarrow 2$, $2 \rightarrow 3$, $3 \rightarrow 4$, $4 \rightarrow 5$.

3. Enumerating DJ graphs

In this section and in Section 4 we use a method originally due to Darboux [8], the singularity analysis of generating functions, to determine the asymptotic behavior of coefficients of generating functions. The method is based on the following theorem:

Theorem 2 (Darboux). Suppose $A(z) = \sum_{n \geq 0} a_n z^n$ is analytic near 0 and has only algebraic singularities α_k on its circle of convergence $|z| = r$, i.e., in a neighborhood of α_k ,

then we have

$$A(z) \sim \left(1 - \frac{z}{\alpha_k}\right)^{-\omega_k} g_k(z),$$

where $\omega_k \neq 0, -1, -2, \dots$ and $g_k(z)$ denotes a non-zero analytic function near α_k . Let $\omega = \max_k \Re(\omega_k)$ denote the maximum of the real part of ω_k and by α_j , ω_j , and g_j denote the values of α , ω , and g such that $\omega_j = \omega$. Then we have

$$a_n = \sum_j \frac{g_j(\alpha_j)}{\Gamma(\omega_j)} n^{\omega_j-1} \alpha_j^{-n} + o(n^\omega r^{-n}).$$

Another well-known tool which allows to determine the asymptotic behavior of coefficients of generating functions is concerned with functional equations [4,6,13].

Suppose that a generating function $w = y(z) = \sum_1^\infty y_n z^n$ with non-negative coefficients is a formal solution of the functional relation $w = F(z, w)$, where $F(z, w)$ is an analytic function of z and w in some neighborhood of the origin. We let \mathcal{D} denote the interior of the set of points (z, w) such that the series defining $F(z, w)$ converges absolutely. We shall denote by \mathcal{S} the set of all points (ρ, τ) with positive coordinates such that

$$\begin{aligned} (\rho, \tau) &\in \mathcal{D}, \\ \tau &= F(\rho, \tau), \\ 1 &= F_w(\rho, \tau). \end{aligned}$$

We cite a theorem from [13].

Theorem 3 (Meir and Moon). *With the definitions from above, suppose that all the coefficients of $F(z, w)$ are non-negative and that (ρ, τ) is in \mathcal{S} . Then*

$$r = \rho \quad \text{and} \quad y(r) = \tau,$$

where r denotes the radius of convergence of $y(z)$.

Moreover, $z = \rho$ is the only singularity of $y(z)$ in the disk $|z| \leq \rho$.

We use the graph grammar given in Table A.1 to set up a probability generating function (PGF) for DJ graphs. In particular, this means that if we expand the PGF into its Taylor series $S(z) = \sum_{n \geq 0} s_n z^n$, then $s_n = [z^n]S(z)$ denotes the probability that a DJ graph consisting of n \bigcirc -nodes is derived by the graph grammar of Table A.1.

In order to set up $S(z)$ we assign a probability to each production of the graph grammar. In particular we assign $0 \leq p_i \leq 1$ to the production numbered i and assume that $p_1 + p_2 = 1$ and $\sum_{i=3}^{31} p_i = 1$. In addition, we assume that $p_1 > 0$, $p_2 > 0$, $0 < p_3 < 1$, and there is at least one $4 \leq i \leq 31$ such that $p_i > 0$.

Furthermore, we define the following predicate

$$\text{Branch} := \sum_{i \in B} p_i > 0, \tag{1}$$

where $B = \{4, 5, 6, 8, 9, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 28, 29, 30, 31\}$. The branch predicate is true if there appear two non-terminals T on the right-hand side of at least one of the productions. Depending on the value of the branch predicate we obtain different results both for the enumeration of DJ graphs and for the average number of Eager2b operations.

In addition, we assume according to [13] that

$$[z^i]T(z) \cdot [z^j]T(z) > 0 \quad \text{for some } j > i \geq 1 \text{ with } \gcd(i, j) = 1. \quad (2)$$

This is no real restriction, because if Eq. (2) is not fulfilled, there are several $\gcd(i, j) \neq 1$ singularities on the radius of convergence of $T(z)$ and our results can be transferred to this case easily.

3.1. The branch predicate is true

We obtain in a straightforward manner (cf. [21])

$$\begin{aligned} S(z) &= p_1 z^2 + p_2 z^2 T(z), \\ T(z) &= p_3 z + p_4 z T^3(z) + p_5 z T^2(z) + \cdots + p_{30} z^2 T^2(z) + p_{31} z T^2(z), \end{aligned} \quad (3)$$

i.e., each \bigcirc -node is replaced by z (if two \bigcirc -nodes appear in one production, they are replaced by z^2) and each appearance of a non-terminal is replaced by its corresponding PGF.

The asymptotic behavior of s_n for $(n \rightarrow \infty)$ can be derived with help of Theorem 3. For notational convenience, we rewrite Eq. (3)

$$\begin{aligned} S(z) &= p_1 z^2 + p_2 z^2 T(z), \\ T(z) &= z\varphi_1(T(z)) + z^2\varphi_2(T(z)), \end{aligned} \quad (4)$$

where $\varphi_i(t) = \sum_j q_{i,j} t^j$ and $q_{i,j} = \sum [z^i][T(z)^j]\mathcal{R}(z)$ and $\mathcal{R}(z)$ denotes the right-hand side of $T(z)$ in Eq. (3).

In addition, we define $F(z, w) = z\varphi_1(w) + z^2\varphi_2(w)$. In order to derive the radius of convergence of $T(z)$, which plays a central role in the asymptotic behavior of s_n , we apply Theorem 3. We obtain

$$w = z\varphi_1(w) + z^2\varphi_2(w), \quad (5)$$

$$1 = z\varphi'_1(w) + z^2\varphi'_2(w). \quad (6)$$

From Eq. (6) we find that

$$z = \frac{1 - z^2\varphi'_2(w)}{\varphi'_1(w)}. \quad (7)$$

Inserting this into Eq. (5) we obtain

$$w = \frac{\varphi_1(w)}{\varphi'_1(w)} + z^2 \left(\varphi_2(w) - \frac{\varphi_1(w)}{\varphi'_1(w)} \varphi'_2(w) \right).$$

From this we find

$$z^2 = \frac{w\varphi'_1(w) - \varphi_1(w)}{\varphi'_1(w)\varphi_2(w) - \varphi_1(w)\varphi'_2(w)}, \quad (8)$$

which when inserted into Eq. (7) produces

$$z = \frac{\varphi_2(w) - w\varphi'_2(w)}{\varphi'_1(w)\varphi_2(w) - \varphi_1(w)\varphi'_2(w)}.$$

Inserting this into Eq. (8) and finally calling the solution $(z, w) = (\rho, \tau)$ we see that τ can be found from the equation

$$(\varphi_2(\tau) - \tau\varphi'_2(\tau))^2 = (\tau\varphi'_1(\tau) - \varphi_1(\tau)) \cdot (\varphi'_1(\tau)\varphi_2(\tau) - \varphi_1(\tau)\varphi'_2(\tau)) \quad (9)$$

and that

$$\rho = \frac{\varphi_2(\tau) - \tau\varphi'_2(\tau)}{\varphi'_1(\tau)\varphi_2(\tau) - \varphi_1(\tau)\varphi'_2(\tau)}. \quad (10)$$

With the definitions of ρ and τ above and Theorem 3 we see that $T(z)$ has an algebraic singularity at $z = \rho$ and we can use Theorem 2 to prove the following lemma:

Lemma 1. *The PGF $T(z)$ given in Eq. (4) fulfills*

$$T(z) = \tau - \sqrt{\frac{2\rho F_z(\rho, \tau)}{F_{ww}(\rho, \tau)}}(1 - z/\rho)^{1/2} + O(1 - z/\rho)$$

for $(z \rightarrow \rho)$.

From this it follows that

$$t_n = [z^n]T(z) = \sqrt{\frac{\rho F_z(\rho, \tau)}{2\pi F_{ww}(\rho, \tau)}} \rho^{-n} n^{-3/2} \left(1 + O\left(\frac{1}{n}\right)\right) \quad (n \rightarrow \infty).$$

Remark 1. From Eq. (3) and Theorem 2 it follows that similar results hold for $S(z)$ and s_n . In both cases the right-hand sides of Lemma 1 have to be multiplied with $p_2\rho^2$ and the term $p_1\rho^2$ has to be added to the series expansion of $S(z)$.

3.2. The branch predicate is false

In this case we have

$$\begin{aligned} T(z) &= p_3z + p_7zT(z) + p_{10}z^2T(z) + p_{11}zT(z) \\ &\quad + p_{24}z^2T(z) + p_{25}zT(z) + p_{26}z^2T(z) + p_{27}zT(z). \end{aligned} \quad (11)$$

Letting $q_1 = p_7 + p_{11} + p_{25} + p_{27}$ and $q_2 = p_{10} + p_{24} + p_{26}$, we find

$$T(z) = \frac{p_3z}{1 - q_1z - q_2z^2}. \quad (12)$$

If $q_2 > 0$, let σ_1 and σ_2 denote the (complex) zeros of the denominator of $T(z)$. By expanding the right-hand side of Eq. (12) we can prove the following lemma.

Lemma 2. *The coefficients of $T(z)$ given in Eq. (11) fulfill*

$$t_n = \bar{c}_1 \sigma_1^{-n} + \bar{c}_2 \sigma_2^{-n}$$

for some (complex) constants \bar{c}_1 and \bar{c}_2 .

If $q_2 = 0$, let $\sigma_3 = 1/q_1$ and by expanding we have the following lemma.

Lemma 3. *The coefficients of $T(z)$ given in Eq. (12) fulfill $t_n = \bar{c}_3 \sigma_3^{-n}$ for some constant \bar{c}_3 .*

4. The average number of Eager2b operations

We introduce the PGF $S(x, y, z)$ such that $[x^k z^n]S(x, x, z)$ is the probability that a DJ graph consisting of n \bigcirc -nodes requires exactly k Eager2b operations. The variable y is only used to simplify the setup process of the PGF and is not needed later on.

In fact, we will not determine $S(x, y, z)$ and its properties; instead we will study $\frac{d}{dx}S(x, y, z)$ which will enable us to derive the average number of Eager2b operations \mathcal{E}_n by

$$\mathcal{E}_n = \frac{[z^n] \frac{d}{dx} S(x, x, z)|_{x=1}}{s_n},$$

where the average is taken over all DJ graphs with n nodes (cf. e.g. [10]). We will derive asymptotic estimates for the numerator only.

In the following, we assume that the branch predicate (1) is true until Section 4.5.

We start by introducing $T(x, y, z)$, the PGF for the T -productions (number 3 to 31 of Table A.1). As already mentioned, z counts the number of \bigcirc -nodes and x counts the number of Eager2b operations. Variable y counts the nesting level of j-edges.

In order to illustrate the term *nesting level* fix a node u in an arbitrary DJ graph. If there exists a node v dominated by u which is a source of a j-edge pointing to a node w such that the level (in terms of the dominator tree) of w is higher than or equal to that of u , then node u is in the first nesting level. If there are k such j-edges, node u is in nesting level k . We call all such j-edges *critical* in respect to u .

The importance of the nesting level stems from the fact that if u is a non-terminal in a DJ graph sentential which is replaced during a derivation step, then all nodes dominated by u “gain height” (in terms of the dominator tree). Thus the Eager2b operations for all the critical j-edges are increased by an amount to be considered below. Anyway, it is very important to know the number of critical j-edges which is equal to the nesting level to keep track of the correct number of Eager2b operations.

In order to set up a functional equation for $T(x, y, z)$ based on the graph grammar given in Table A.1, we have to obey the following rules.

- Normally all \bullet -T-nodes are replaced with $T(x, y, z)$. These nodes are presumptive sources for j-edges and the nesting level of j-edges, i.e., the variable y , is not changed.

- The \bullet -T-nodes in productions 16, 17, 20, 21, 24, 25, 26, 27, 30, and 31 are different in that they are inside a loop local to the production. Thus, the nesting level of j-edges is increased by one. This is reflected by replacing the variable y in $T(x, y, z)$ with xy .
- All the other T-nodes are independent from existing j-edges. Thus, they are replaced with $T(x, 1, z)$, $T(x, x, z)$, or $T(x, x^2, z)$ depending on how many j-edges have their source in the T-node.

For example, consider production 19: the uppermost T-node is not a source of a j-edge; thus, it is replaced with $T(x, 1, z)$. The left one of the remaining T-nodes is replaced with $T(x, y, z)$ and the right one with $T(x, x, z)$ because one j-edge has its source there.

In addition, we have to consider the number of \bigcirc -nodes in the productions, which result in z^r if there are r \bigcirc -nodes.

Finally, we have to take into account how much the number of Eager2b operations is increased by the production. Locally this is just the difference s of the levels between the source and the target of the j-edges and is mapped to x^s . Globally we also have to consider that the current production is nested inside a j-edge structure. The nesting level is counted in variable y . If there is only one \bullet -node in a certain production, let h denote the height of the \bullet -node, i.e., the distance to the root in the production. Then we have to add y^h to our counting expression. In general, if there is more than one \bullet -node in production p , we define a tree H_p in the following way:

- H_p is a subgraph of production p .
- The root of H_p is the root of production p .
- The leaves of H_p are the \bullet -nodes in p .
- Internal nodes and edges of H_p are found along the shortest paths of d- or dj-edges in p from the root to the leaves of H_p .

Denoting the number of edges in H_p by h , we have to add y^h to our counting expression.

A proof of the correctness of this construction is straightforward. Note that this construction simplifies to the case with only one \bullet -node discussed above.

Note also that our construction correctly takes care of the fact that sometimes one of two coinciding j-edges is removed. For example, consider production 7: all j-edges that get duplicated when this production is applied, are removed when their sources reach the root node of production 7 (cf. also Fig. 6).

Returning to production 19 we get a factor z since there is one \bigcirc -node, a factor x^2 because there is one local j-edge spanning two levels, and a factor y^2 as the height of the \bullet -T-node equals 2.

An approach similar to our nesting level being counted by y has been followed by Knuth [12, 2.3.4.5] to determine the average path length of binary trees. He finds a generating function for the number of binary trees with n nodes and internal path length p

$$B(w, z) = \sum_{n, p \geq 0} b_{n, p} w^n z^p = 1 + zB(w, wz)^2,$$

where z is used to accumulate the internal path length by appropriate powers of w .

Applying the above rules to all productions we get the fourth column in Table A.1. Hence we obtain the following functional equation

$$\begin{aligned}
 T(x, y, z) = & p_3z \\
 & + p_4yzT(x, x, z)^2T(x, y, z) \\
 & + p_5y^2zT(x, y, z)^2 \\
 & + p_6yzT(x, x, z)T(x, y, z) \\
 & + p_7yzT(x, y, z) \\
 & + p_8xy^2z^2T(x, x, z)T(x, y, z) \\
 & + p_9xyzT(x, x, z)T(x, y, z) \\
 & + p_{10}xyz^2T(x, x, z) \\
 & + p_{11}xzT(x, x, z) \\
 & + p_{12}xy^2z^2T(x, x^2, z)T(x, y, z) \\
 & + p_{13}xyzT(x, x^2, z)T(x, y, z) \\
 & + p_{14}x^2y^2z^2T(x, 1, z)T(x, x^2, z)T(x, y, z) \\
 & + p_{15}x^2yzT(x, 1, z)T(x, x^2, z)T(x, y, z) \\
 & + p_{16}x^2y^3z^2T(x, x^2, z)T(x, xy, z) \\
 & + p_{17}x^2yzT(x, x^2, z)T(x, xy, z) \\
 & + p_{18}x^2y^3z^2T(x, 1, z)T(x, x, z)T(x, y, z) \\
 & + p_{19}x^2y^2zT(x, 1, z)T(x, x, z)T(x, y, z) \\
 & + p_{20}x^2y^2z^2T(x, x, z)T(x, xy, z) \\
 & + p_{21}x^2yzT(x, x, z)T(x, xy, z) \\
 & + p_{22}xy^3z^2T(x, x, z)T(x, y, z) \\
 & + p_{23}xy^2zT(x, x, z)T(x, y, z) \\
 & + p_{24}xy^2z^2T(x, xy, z) \\
 & + p_{25}xyzT(x, xy, z) \\
 & + p_{26}xy^2z^2T(x, xy, z) \\
 & + p_{27}xyzT(x, xy, z) \\
 & + p_{28}x^2y^3z^2T(x, 1, z)T(x, x, z)T(x, y, z) \\
 & + p_{29}x^2y^2zT(x, 1, z)T(x, x, z)T(x, y, z) \\
 & + p_{30}x^2y^3z^2T^2(x, xy, z) \\
 & + p_{31}x^2y^2zT^2(x, xy, z).
 \end{aligned} \tag{13}$$

In addition, we clearly have

$$S(x, x, z) = p_1z^2 + p_2z^2T(x, x, z).$$

Thus,

$$\mathcal{E}_n = \frac{[z^n] \frac{d}{dx} S(x, x, z)|_{x=1}}{s_n} = p_2 \frac{[z^n] z^2 (\frac{d}{dy} T(x, y, z) + \frac{d}{dx} T(x, y, z))|_{x=1, y=1}}{s_n}$$

and we have to deal with derivatives of $T(x, y, z)$.

Returning to our example (Fig. 5) we derive (ignoring the probabilities p_i)

$$\begin{aligned} S(x, x, z) &\xrightarrow{2} z^2 T(x, x, z) \xrightarrow{25} x^2 z^3 T(x, x^2, z) \\ &\xrightarrow{7} x^4 z^4 T(x, x^2, z) \xrightarrow{25} x^7 z^5 T(x, x^3, z) \xrightarrow{3} x^7 z^6 \end{aligned}$$

which correctly finds that the number of nodes is 6 (the exponent of z) and the number of Eager2b operations is 7 (the exponent of x). By denoting $T_y(z) = \frac{d}{dy} T(x, y, z)|_{x=1, y=1}$ and $T(z) = T(x, y, z)|_{x=1, y=1}$, which is equivalent to $T(z)$ in Section 3, we find by differentiating Eq. (13) w.r.t. y and letting $x = 1$ and $y = 1$

$$\begin{aligned} T_y(z) = & p_{4z} T(z)^3 + p_{4z} T(z)^2 T_y(z) \\ & + 2p_{5z} T(z)^2 + 2p_{5z} T(z) T_y(z) \\ & + p_{6z} T(z)^2 + p_{6z} T(z) T_y(z) \\ & + p_{7z} T(z) + p_{7z} T_y(z) \\ & + 2p_{8z^2} T(z)^2 + p_{8z^2} T(z) T_y(z) \\ & + p_{9z} T(z)^2 + p_{9z} T(z) T_y(z) \\ & + p_{10z^2} T(z) \\ & + 2p_{12z^2} T(z)^2 + p_{12z^2} T(z) T_y(z) \\ & + p_{13z} T(z)^2 + p_{13z} T(z) T_y(z) \\ & + 2p_{14z^2} T(z)^3 + p_{14z^2} T(z)^2 T_y(z) \\ & + p_{15z} T(z)^3 + p_{15z} T(z)^2 T_y(z) \\ & + 3p_{16z^2} T(z)^3 + p_{16z^2} T(z) T_y(z) \\ & + p_{17z} T(z)^2 + p_{17z} T(z) T_y(z) \\ & + 3p_{18z^2} T(z)^3 + p_{18z^2} T(z)^2 T_y(z) \\ & + 2p_{19z} T(z)^3 + p_{19z} T(z)^2 T_y(z) \\ & + 2p_{20z^2} T(z)^2 + p_{20z^2} T(z) T_y(z) \\ & + p_{21z} T(z)^2 + p_{21z} T(z) T_y(z) \\ & + 3p_{22z^2} T(z)^2 + p_{22z^2} T(z) T_y(z) \\ & + 2p_{23z} T(z)^2 + p_{23z} T(z) T_y(z) \\ & + 2p_{24z^2} T(z) + p_{24z^2} T_y(z) \\ & + p_{25z} T(z) + p_{25z} T_y(z) \\ & + 2p_{26z^2} T(z) + p_{26z^2} T_y(z) \end{aligned}$$

$$\begin{aligned}
& + p_{27}zT(z) + p_{27}zT_y(z) \\
& + 3p_{28}z^2T^3(z) + p_{28}z^2T^2(z)T_y(z) \\
& + 2p_{29}zT^3(z) + p_{29}z^2T^2(z)T_y(z) \\
& + 3p_{30}z^2T^2(z) + 2p_{30}z^2T(z)T_y(z) \\
& + 2p_{31}zT^2(z) + 2p_{31}zT(z)T_y(z).
\end{aligned} \tag{14}$$

We obtain an explicit expression $T_y(z) = \frac{N_y(z)}{D_y(z)}$ where

$$\begin{aligned}
N_y(z) = & p_4zT(z)^3 + 2p_5zT(z)^2 + p_6zT(z)^2 \\
& + p_7zT(z) + p_8z^22T(z)^2 + p_9zT(z)^2 \\
& + p_{10}z^2T(z) + 2p_{12}z^2T(z)^2 + p_{13}zT(z)^2 \\
& + 2p_{14}z^2T(z)^3 + p_{15}zT(z)^3 + 3p_{16}z^2T(z)^2 \\
& + p_{17}zT(z)^2 + 3p_{18}z^2T(z)^3 + 2p_{19}zT(z)^3 \\
& + 2p_{20}z^2T(z)^2 + p_{21}zT(z)^2 + 3p_{22}z^2T(z)^2 \\
& + 2p_{23}zT(z)^2 + 2p_{24}z^2T(z) + p_{25}zT(z) \\
& + 2p_{26}z^2T(z) + p_{27}zT(z) + 3p_{28}z^2T^3(z) \\
& + 2p_{29}zT^3(z) + 3p_{30}z^2T^2(z) + 2p_{31}zT^2(z)
\end{aligned} \tag{15}$$

and

$$\begin{aligned}
D_y(z) = & 1 - p_4zT(z)^2 - 2p_5zT(z) - p_6zT(z) \\
& - p_7z - p_8z^2T(z) - p_9zT(z) - p_{12}z^2T(z) \\
& - p_{13}zT(z) - p_{14}z^2T(z)^2 - p_{15}zT(z)^2 \\
& - p_{16}z^2T(z) - p_{17}zT(z) - p_{18}z^2T(z)^2 \\
& - p_{19}zT(z)^2 - p_{20}z^2T(z) - p_{21}zT(z) \\
& - p_{22}z^2T(z) - p_{23}zT(z) - p_{24}z^2 - p_{25}z \\
& - p_{26}z^2 - p_{27}z - p_{28}z^2T^2(z) \\
& - p_{29}zT^2(z) - 2p_{30}z^2T(z) - 2p_{31}zT(z).
\end{aligned} \tag{16}$$

By denoting $T_x(z) = \frac{d}{dx}T(x, y, z)|_{x=1, y=1}$ and $T(z) = T(x, y, z)|_{x=1, y=1}$ we find by differentiating Eq. (13) w.r.t. x and letting $x = 1$ and $y = 1$

$$\begin{aligned}
T_x(z) = & p_4zT(z)^2T_x(z) + 2p_4zT(z)^2(T_x(z) + T_y(z)) \\
& + 2p_5zT(z)T_x(z) \\
& + p_6zT(z)T_x(z) + p_6zT(z)(T_x(z) + T_y(z)) \\
& + p_7zT_x(z) \\
& + p_8z^2T(z)^2 + p_8z^2T(z)T_x(z) + p_8z^2T(z)(T_x(z) + T_y(z)) \\
& + p_9zT(z)^2 + p_9zT(z)T_x(z) + p_9zT(z)(T_x(z) + T_y(z))
\end{aligned}$$

$$\begin{aligned}
& + p_{10}z^2T(z) + p_{10}z^2(T_x(z) + T_y(z)) \\
& + p_{11}zT(z) + p_{11}z(T_x(z) + T_y(z)) \\
& + p_{12}z^2T(z)^2 + p_{12}z^2T(z)T_x(z) + p_{12}z^2T(z)(T_x(z) + 2T_y(z)) \\
& + p_{13}zT(z)^2 + p_{13}zT(z)T_x(z) + p_{13}zT(z)(T_x(z) + 2T_y(z)) \\
& + 2p_{14}z^2T(z)^3 + 2p_{14}z^2T(z)^2T_x(z) + p_{14}z^2T(z)^2(T_x(z) + 2T_y(z)) \\
& + 2p_{15}zT(z)^3 + 2p_{15}zT(z)^2T_x(z) + p_{15}zT(z)^2(T_x(z) + 2T_y(z)) \\
& + 2p_{16}z^2T(z)^2 + p_{16}z^2T(z)(T_x(z) + T_y(z)) \\
& + p_{16}z^2T(z)(T_x(z) + 2T_y(z)) \\
& + 2p_{17}zT(z)^2 + p_{17}zT(z)(T_x(z) + T_y(z)) + p_{17}zT(z)(T_x(z) + 2T_y(z)) \\
& + p_{18}z^22T(z)^3 + p_{18}z^22T(z)^2T_x(z) + p_{18}z^2T(z)^2(T_x(z) + T_y(z)) \\
& + p_{19}z2T(z)^3 + p_{19}z2T(z)^2T_x(z) + p_{19}zT(z)^2(T_x(z) + T_y(z)) \\
& + p_{20}z^22T(z)^2 + 2p_{20}z^2T(z)(T_x(z) + T_y(z)) \\
& + p_{21}z2T(z)^2 + 2p_{21}zT(z)(T_x(z) + T_y(z)) \\
& + p_{22}z^2T(z)^2 + p_{22}z^2T(z)T_x(z) + p_{22}z^2T(z)(T_x(z) + T_y(z)) \\
& + p_{23}zT(z)^2 + p_{23}zT(z)T_x(z) + p_{23}zT(z)(T_x(z) + T_y(z)) \\
& + p_{24}z^2T(z) + p_{24}z^2(T_x(z) + T_y(z)) \\
& + p_{25}zT(z) + p_{25}z(T_x(z) + T_y(z)) \\
& + p_{26}z^2T(z) + p_{26}z^2(T_x(z) + T_y(z)) \\
& + p_{27}z^2T(z) + p_{27}z(T_x(z) + T_y(z)) \\
& + 2p_{28}z^2T^3(z) + 2p_{28}z^2T^2(z)T_x(z) + p_{28}z^2T^2(z)(T_x(z) + T_y(z)) \\
& + 2p_{29}zT^3(z) + 2p_{29}zT^2(z)T_x(z) + p_{29}zT^2(z)(T_x(z) + T_y(z)) \\
& + 2p_{30}z^2T^2(z) + 2p_{30}z^2T(z)(T_x(z) + T_y(z)) \\
& + 2p_{31}zT^2(z) + 2p_{31}zT(z)(T_x(z) + T_y(z)).
\end{aligned}$$

We obtain an explicit expression $T_x(z) = \frac{N_x(z)}{D_x(z)}$ where

$$\begin{aligned}
N_x(z) = & 2p_{4z}T(z)^2T_y(z) \\
& + p_{6z}T(z)T_y(z) \\
& + p_{8z}^2T(z)^2 + p_{8z}^2T(z)T_y(z) \\
& + p_{9z}T(z)^2 + p_{9z}T(z)T_y(z) \\
& + p_{10z}^2T(z) + p_{10z}^2T_y(z) \\
& + p_{11z}T(z) + p_{11z}T_y(z) \\
& + p_{12z}^2T(z)^2 + p_{12z}^22T(z)T_y(z) \\
& + p_{13z}T(z)^2 + p_{13z}2T(z)T_y(z)
\end{aligned}$$

$$\begin{aligned}
& + 2p_{14}z^2T(z)^3 + 2p_{14}z^2T(z)^2T_y(z) \\
& + 2p_{15}zT(z)^3 + 2p_{15}zT(z)^2T_y(z) \\
& + 2p_{16}z^2T(z)^2 + 3p_{16}z^2T(z)T_y(z) \\
& + 2p_{17}zT(z)^2 + 3p_{17}zT(z)T_y(z) \\
& + 2p_{18}z^2T(z)^3 + p_{18}z^2T(z)^2T_y(z) \\
& + 2p_{19}zT(z)^3 + p_{19}zT(z)^2T_y(z) \\
& + 2p_{20}z^2T(z)^2 + 2p_{20}z^2T(z)T_y(z) \\
& + 2p_{21}zT(z)^2 + 2p_{21}zT(z)T_y(z) \\
& + p_{22}z^2T(z)^2 + p_{22}z^2T(z)T_y(z) \\
& + p_{23}zT(z)^2 + p_{23}zT(z)T_y(z) \\
& + p_{24}z^2T(z) + p_{24}z^2T_y(z) \\
& + p_{25}zT(z) + p_{25}zT_y(z) \\
& + p_{26}z^2T(z) + p_{26}z^2T_y(z) \\
& + p_{27}zT(z) + p_{27}zT_y(z) \\
& + 2p_{28}z^2T^3(z) + p_{28}z^2T^2(z)T_y(z) \\
& + 2p_{29}zT^3(z) + p_{29}zT^2(z)T_y(z) \\
& + 2p_{30}z^2T^2(z) + 2p_{30}z^2T(z)T_y(z) \\
& + 2p_{31}zT^2(z) + 2p_{31}zT(z)T_y(z)
\end{aligned} \tag{17}$$

and

$$\begin{aligned}
D_x(z) = & 1 - 3p_{4z}T(z)^2 - 2p_5zT(z) \\
& - 2p_6zT(z) - p_7z - 2p_8z^2T(z) \\
& - 2p_9zT(z) - p_{10}z^2 - p_{11}z - 2p_{12}z^2T(z) \\
& - 2p_{13}zT(z) - 3p_{14}z^2T(z)^2 - 3p_{15}zT(z)^2 \\
& - 2p_{16}z^2T(z) - 2p_{17}zT(z) - 3p_{18}z^2T(z)^2 \\
& - 3p_{19}zT(z)^2 - 2p_{20}z^2T(z) - 2p_{21}zT(z) \\
& - 2p_{22}z^2T(z) - 2p_{23}zT(z) - p_{24}z^2 - p_{25}z \\
& - p_{26}z^2 - p_{27}z - 3p_{28}z^2T^2(z) \\
& - 3p_{29}zT^2(z) - 2p_{30}z^2T(z) - 2p_{31}zT(z).
\end{aligned} \tag{18}$$

Note that $D_x(z)$ is equivalent to $1 - F_w(z, w)$ of Section 3 which means that for $(z \rightarrow \rho)$ the denominator of $T_x(z)$ approaches zero. Thus, $T_x(z)$ has a pole at $z = \rho$.

The asymptotic behavior of \mathcal{E}_n is determined by the singularities of $D_y(z)$ and $D_x(z)$. We have to discriminate two cases: case (1) is when $D_y(z)$ and $D_x(z)$ have different singularities and case (2) is when the singularities coincide. Hence, we define

$$\Delta(z) = D_y(z) - D_x(z)$$

$$\begin{aligned}
&= z(2p_4T(z)^2 + p_6T(z) + p_8zT(z) + p_9T(z) + p_{10}z + p_{11} + p_{12}zT(z) \\
&\quad + p_{13}T(z) + 2p_{14}zT(z)^2 + 2p_{15}T(z)^2 + p_{16}zT(z) + p_{17}T(z) \\
&\quad + 2p_{18}zT(z)^2 + 2p_{19}T(z)^2 + p_{20}zT(z) + p_{21}T(z) \\
&\quad + p_{22}zT(z) + p_{23}T(z) + 2p_{28}z^2T^2(z) + 2p_{29}zT^2(z)). \tag{19}
\end{aligned}$$

4.1. The case $\Delta(z) \neq 0$

It is easy to see that the Taylor series expansion of $\psi(z) = 1/D_y(z)$ at $z = 0$ has only non-negative coefficients. Thus, the smallest singularity of $\psi(z)$ is located on the real axis.

On the other hand, $D_x(z)$ has its smallest zero at $z = \rho$ which follows from [13]. Now for real argument $D_x(z)$ is monotonically decreasing with $D_x(0) = 1$ and $D_x(\rho) = 0$.

In addition, $\Delta(z)$ is monotonically increasing for real argument with $\Delta(0) = 0$ and $\Delta(\rho) > 0$. Hence, $D_y(z) = D_x(z) + \Delta(z) > 0$ for $0 \leq z \leq \rho$ which implies that the singularity of $\psi(z) > \rho$.

Letting $T(z) = \tau - c_1u^{1/2} + O(u)$ (see Lemma 1), where $u = 1 - z/\rho$, we obtain for $(z \rightarrow \rho)$

$$D_y(z) = c_2 + O(u)$$

for some constant c_2 , and

$$S_x(z) = \frac{d}{dx} S(x, x, z) \Big|_{x=1} = c_3u^{-1/2} + c_4 + O(u^{1/2})$$

for some constants c_3 and c_4 .

Applying Theorem 2, it follows that for some constant c_5 and $(n \rightarrow \infty)$

$$[z^n]S_x(z) = c_5n^{-1/2}\rho^{-n} \left(1 + O\left(\frac{1}{n}\right)\right).$$

Hence, dividing by s_n , we obtain

$$\mathcal{E}_n = c_6 \cdot n + O(1)$$

for some constant c_6 and $(n \rightarrow \infty)$.

4.2. The case $\Delta(z) \equiv 0$

In this case $p_4 = p_6 = p_8 = p_9 = p_{10} = p_{11} = p_{12} = p_{13} = p_{14} = p_{15} = p_{16} = p_{17} = p_{18} = p_{19} = p_{20} = p_{21} = p_{22} = p_{23} = p_{28} = p_{29} = 0$.

We further discriminate the cases where $p_5 = 0$ and where $p_5 \neq 0$.

4.3. The case $\Delta(z) \equiv 0$ and $p_5 \neq 0$

In this case $D_x(z) \equiv D_y(z)$ which implies that $\frac{d}{dx} S(x, x, z)|_{x=1}$ has a double pole at $z = \rho$. Hence, we get for $(z \rightarrow \rho)$

$$S_x(z) = \frac{d}{dx} S(x, x, z) \Big|_{x=1} = c_7u^{-1} + O(u^{-1/2})$$

for some constant c_7 .

This implies that for some constant c_8 and for $(n \rightarrow \infty)$

$$[z^n]S_x(z) = c_8 \rho^{-n} \left(1 + O\left(\frac{1}{n}\right) \right).$$

Dividing by s_n we get for some constant c_9 and for $(n \rightarrow \infty)$

$$\mathcal{E}_n = c_9 \cdot n^{3/2} + O(n^{1/2}).$$

4.4. The case $\Delta(z) \equiv 0$ and $p_5 = 0$

This case contradicts the branch predicate and for this reason is treated in Section 4.5.

4.5. The branch predicate is false

In this case we obtain

$$D_y(z) = 1 - p_7z - p_{24}z^2 - p_{25}z - p_{26}z^2 - p_{27}z$$

and

$$D_x(z) = 1 - q_1z - q_2z^2,$$

where q_1 and q_2 are defined in Section 3.2.

If $\Delta(z) = D_y(z) - D_x(z) \not\equiv 0$, it is easy to see from Eq. (12) that near its singularities $T(z)$ behaves like $D_x(z)^{-1}$. Furthermore, we obtain that $T_y(z)$ behaves like $D_x(z)^{-1}$ and that both $T_x(z)$ and $S_x(z)$ behave like $D_x(z)^{-2}$.

Hence we have

$$\mathcal{E}_n = c_{10} \cdot n + O(1) \quad (n \rightarrow \infty)$$

for some constant c_{10} .

In case of $\Delta(z) \equiv 0$, i.e., $p_{10} = p_{11} = 0$, we find that $T_y(z)$ behaves like $D_x(z)^{-2}$. In addition, $T_x(z)$ and $S_x(z)$ behave like $D_x(z)^{-3}$.

Thus we have

$$\mathcal{E}_n = c_{11} \cdot n^2 + O(n) \quad (n \rightarrow \infty)$$

for some constant c_{11} .

Summing up, we have proved the following theorem.

Theorem 4. *Let G be a DJ graph with n nodes which can be derived by the graph grammar depicted in Table A.1. Then the average number of Eager2b operations performed by Sreedhar's algorithm \mathcal{E}_n can be determined as follows.*

If the branch predicate (1) is true, then

- (a) *if further $p_4 = p_6 = p_8 = p_9 = p_{10} = p_{11} = p_{12} = p_{13} = p_{14} = p_{15} = p_{16} = p_{17} = p_{18} = p_{19} = p_{20} = p_{21} = p_{22} = p_{23} = 0$,*

$$\mathcal{E}_n = c_9 \cdot n^{3/2} + O(n^{1/2}),$$

(b) *otherwise*

$$\mathcal{E}_n = c_6 \cdot n + O(1).$$

If the branch predicate (1) is false and

(c) if further $p_{10} = p_{11} = 0$, we obtain

$$\mathcal{E}_n = c_{11} \cdot n^2 + O(n),$$

(d) *otherwise*

$$\mathcal{E}_n = c_{10} \cdot n + O(1).$$

Taking a closer look at [Theorem 4](#) we find that programming languages of case (c) consist of repeat-until-loops only. Programming languages of case (a) support only restricted forms of if-statements and repeat-until-loops. Except for such very uncommon languages, the following corollary holds.

Corollary 1. *The average number of Eager2b operations performed by Sreedhar’s algorithm for goto-free programs is linear in the size of the program.*

Remark 2. Note that a program P , consisting only of straight-line code and repeat-until-loops written in a certain programming language L that also supports other language features like if-statements and while-loops, implies quadratic running time of Sreedhar’s algorithm for this specific program P .

Since, however, the probabilities for if-statements and while-loops are non-zero, according to [Theorem 4](#) the average case performance of Sreedhar’s algorithm for programs written in L is linear.

It is shown in [18] that the number of Eager2b operations corresponds to the size of the dominance frontier [7] (which is needed for SSA⁶ analysis, a method common in compiler construction).

Thus, we have also proved the following corollary as a byproduct.

Corollary 2. *Under the same assumptions as in [Corollary 1](#), the average size of the dominance frontier is linear in the program size.*

In the following we prove a theorem originally due to [7].

Theorem 5 (Cytron et al.). *For programs comprised of straight-line code, if-then-else and while-do constructs, the dominance frontier of any CFG node contains at most two nodes.*

⁶ Static Single Assignment.

Proof. We prove this theorem by restricting our graph grammar to the productions 1 to 11. Note that in this case no $T(x, xy, z)$ term appears. Thus, we obtain the following functional equation for $T(x, x, z)$.

$$\begin{aligned} T(x, x, z) = & p_3z + p_4xzT(x, x, z)^3 + p_5x^2zT(x, x, z)^2 \\ & + p_6xzT(x, x, z)^2 + p_7xzT(x, x, z) + p_8x^3z^2T(x, x, z)^2 \\ & + p_9x^2zT(x, x, z)^2 + p_{10}x^2z^2T(x, x, z) + p_{11}xzT(x, x, z). \end{aligned} \quad (20)$$

Note that $[x^k][z^n]S(x, x, z) = [x^k][z^n](p_1z^2 + p_2z^2T(x, x, z))$ denotes the exact number of DJGs with n nodes and k Eager2b operations. From Eq. (20) the theorem is obvious. \square

5. Conclusion

First we would like to note that the probability distribution employed in this paper is very general in nature. Although the p_i assigned to the productions are considered independent from each other and from the number of nodes in the graphs, they can be used to model any particular set of programs for which then the average case timing behavior can be predicted quite accurately since the constants of Theorem 4 can be determined exactly if the p_i for each production are known.

In this paper we have proved that for goto-free programs written in usual programming languages, i.e., languages that provide at least for straight-line code, if-statements, repeat-until-loops, and while-loops (or semantically equivalent features such as general loops with exit-statements), the average number of Eager2b operations performed by Sreedhar's algorithm is linear in the size of the input program. Our approach employs a graph grammar to specify goto-free programs and well-known methods for the analysis of algorithms and data-structures to determine the average number of Eager2b operations.

As a consequence it follows that for goto-free programs the average size of the dominance frontier [7] is linear in the size of the underlying program.

An earlier result of Cytron et al. [7] can be reproduced by our approach.

In terms of solving a system of simultaneous equations our result can be interpreted in the following way: if a system of simultaneous equations can be described in terms of a graph which can be derived by our graph grammar, then, on the average, a solution of the system of equations can be obtained in linear time. This also holds for suitable sparse systems of linear equations [20].

It is easy to add productions to our graph grammar that model some kind of multiple entry loops. Table A.2 shows one of the productions required to model such loops. With help of this extension our result also applies to some irreducible graphs [3] for which the average number of Eager2b operations is still linear in the size of the program.

On the other hand, it is still an open question whether the average number of Eager2b operations is linear for all reducible flow graphs [3] under some practically useful probability distribution.

Acknowledgements


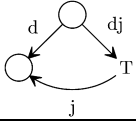

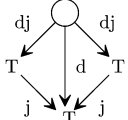
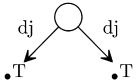
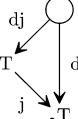

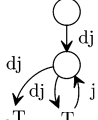

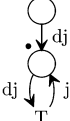
The author would like to thank the anonymous referees for their helpful comments.

Appendix A. The graph grammar in tabular form

This appendix contains the graph grammar underlying this paper.

Table A.1

A graph grammar for DJ graphs

No.	Productions Language	Graph	Count. expr.
1	$S \rightarrow \text{begin end}$		$p_1 z^2$
2	$S \rightarrow \text{begin } T \text{ end}$		$p_2 z^2 T(x, x, z)$
3	$T \rightarrow s, T_0 \rightarrow s$		$p_3 z$
4	$T \rightarrow \text{if } c \text{ then } T \text{ else } T \text{ endif } T$		$p_4 y z T(x, x, z)^2$ $T(x, y, z)$
5	$T \rightarrow \text{if } c \text{ then } T \text{ else } T \text{ endif}$		$p_5 y^2 z T(x, y, z)^2$
6	$T \rightarrow \text{if } c \text{ then } T \text{ endif } T$		$p_6 y z T(x, x, z)$ $T(x, y, z)$
7	$T \rightarrow \text{if } c \text{ then } T \text{ endif}$		$p_7 y z T(x, y, z)$
8	$T \rightarrow T_0 \text{ while } c \text{ loop } T \text{ endloop } T$		$p_8 x y^2 z^2$ $T(x, x, z) T(x, y, z)$
9	$T \rightarrow \text{while } c \text{ loop } T \text{ endloop } T$		$p_9 x y z$ $T(x, x, z) T(x, y, z)$
10	$T \rightarrow T_0 \text{ while } c \text{ loop } T \text{ endloop}$		$p_{10} x y z^2 T(x, x, z)$

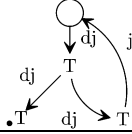
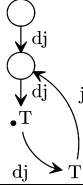
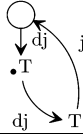
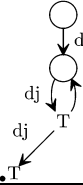
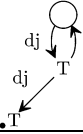
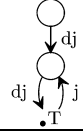

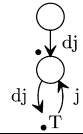

(continued on next page)

Table A.1 (Continued from Table A.1)

No.	Productions Language	Graph	Count. expr.
11	$T \rightarrow \text{while } c \text{ loop } T \text{ endloop}$		$p_{11}xzT(x, x, z)$
12	$T \rightarrow T_0 \text{ while } c \text{ loop}$ $T \text{ exit when } c \text{ endloop } T$		$p_{12}xy^2z^2$ $T(x, x^2, z)T(x, y, z)$
13	$T \rightarrow \text{while } c \text{ loop}$ $T \text{ exit when } c \text{ endloop } T$		$p_{13}xyz$ $T(x, x^2, z)T(x, y, z)$
14	$T \rightarrow T_0 \text{ while } c \text{ loop}$ $T \text{ exit when } c \text{ } T \text{ endloop } T$		$p_{14}x^2y^2z^2T(x, 1, z)$ $T(x, x^2, z)T(x, y, z)$
15	$T \rightarrow \text{while } c \text{ loop}$ $T \text{ exit when } c \text{ } T \text{ endloop } T$		$p_{15}x^2yzT(x, 1, z)$ $T(x, x^2, z)T(x, y, z)$
16	$T \rightarrow T_0 \text{ while } c \text{ loop}$ $T \text{ exit when } c \text{ } T \text{ endloop}$		$p_{16}x^2y^3z^2$ $T(x, x^2, z)T(x, xy, z)$
17	$T \rightarrow \text{while } c \text{ loop}$ $T \text{ exit when } c \text{ } T \text{ endloop}$		$p_{17}x^2yz$ $T(x, x^2, z)T(x, xy, z)$
18	$T \rightarrow T_0 \text{ loop } T \text{ exit when } c$ $T \text{ endloop } T$		$p_{18}x^2y^3z^2T(x, 1, z)$ $T(x, x, z)T(x, y, z)$

(continued on next page)

Table A.1 (Continued from Table A.1)

No.	Productions Language	Graph	Count. expr.
19	$T \rightarrow \text{loop } T \text{ exit when } c$ $T \text{ endloop } T$		$p_{19}x^2y^2zT(x, 1, z)$ $T(x, x, z)T(x, y, z)$
20	$T \rightarrow T_0 \text{ loop } T \text{ exit when } c$ $T \text{ endloop}$		$p_{20}x^2y^2z^2$ $T(x, x, z)T(x, xy, z)$
21	$T \rightarrow \text{loop } T \text{ exit when } c$ $T \text{ endloop}$		$p_{21}x^2yz$ $T(x, x, z)T(x, xy, z)$
22	$T \rightarrow T_0 \text{ repeat } T \text{ until } c$		$p_{22}xy^3z^2$ $T(x, x, z)T(x, y, z)$
23	$T \rightarrow \text{repeat } T \text{ until } c$		$p_{23}xy^2z$ $T(x, x, z)T(x, y, z)$
24	$T \rightarrow T_0 \text{ repeat } T \text{ until } c$		$p_{24}xy^2z^2T(x, xy, z)$
25	$T \rightarrow \text{repeat } T \text{ until } c$		$p_{25}xyzT(x, xy, z)$
26	$T \rightarrow T_0 \text{ while } c \text{ loop}$ $T \text{ exit when } c \text{ endloop}$		$p_{26}xy^2z^2T(x, xy, z)$
27	$T \rightarrow \text{while } c \text{ loop}$ $T \text{ exit when } c \text{ endloop}$		$p_{27}xyzT(x, xy, z)$

(continued on next page)

Table A.1 (Continued from Table A.1)

No.	Productions Language	Graph	Count. expr.
28	$T \rightarrow T_0$ repeat T exit when c T until c T		$p_{28}x^2y^3z^2T(x, 1, z)$ $T(x, x, z)T(x, y, z)$
29	$T \rightarrow$ repeat T exit when c T until c T		$p_{29}x^2y^2zT(x, 1, z)$ $T(x, x, z)T(x, y, z)$
30	$T \rightarrow T_0$ repeat T exit when c T until c		$p_{30}x^2y^3z^2T^2(x, xy, z)$
31	$T \rightarrow$ repeat T exit when c T until c		$p_{31}x^2y^2zT^2(x, xy, z)$

Table A.2

Extension of graph grammar: Multiple entry loop

No.	Productions Description	Graph
32	Multiple entry loop	

References

- [1] K. Arnold, J. Gosling, D. Holmes, *The Java Programming Language*, third ed., Addison-Wesley, Reading, MA, 2000.
- [2] S. Alstrup, D. Harel, P.W. Lauridsen, M. Thorup, Dominators in linear time, *SIAM J. Comput.* 28 (6) (1999) 2117–2132.
- [3] A.V. Aho, R. Seti, J.D. Ullman, *Compilers: Principles, Techniques, and Tools*, Addison-Wesley, Reading, MA, 1986.
- [4] E.A. Bender, Asymptotic methods in enumeration, *SIAM Rev.* 16 (1974) 485–515.
- [5] J. Blieberger, Data-flow frameworks for worst-case execution time analysis, *Real-Time Syst.* 22 (3) (2002) 183–227.
- [6] E.R. Canfield, Remarks on an asymptotic method in combinatorics, *J. Combin. Theory Ser. A* 37 (1984) 348–352.
- [7] R. Cytron, J. Ferrante, B.K. Rosen, M.N. Wegman, F.K. Zadeck, Efficiently computing static single assignment form and the control dependence graph, *ACM Trans. Program. Lang. Syst.* 13 (4) (1991) 451–490.
- [8] G. Darboux, Mémoire sur l’approximation des fonctions de très-grands nombres, et sur une classe étendue de développements en série, *J. Math. Pures Appl.* 4 (3) (1878) 5–56.
- [9] J. Engblom, A. Ermedahl, Modeling complex flows for worst-case execution time analysis, in: *Proc. 21st IEEE Real-Time Systems Symposium (RTSS)*, Orlando, FL, December 2000.
- [10] R.L. Graham, D.E. Knuth, O. Patashnik, *Concrete Mathematics*, Addison-Wesley, Reading, MA, 1989.
- [11] L. Georgiadis, R.E. Tarjan, Finding dominators revisited, in: *Proc. SODA’04*, New Orleans, LA, SIAM, 2004, pp. 869–878.
- [12] D.E. Knuth, *Fundamental Algorithms, The Art of Computer Programming*, vol. 1, third ed., Addison-Wesley, Reading, MA, 1977.
- [13] A. Meir, J.W. Moon, On an asymptotic method in enumeration, *J. Combin. Theory Ser. A* 51 (1989) 77–89.
- [14] M.C. Paull, *Algorithm Design—a Recursion Transformation Framework*, Wiley Interscience, New York, 1988.
- [15] G. Ramalingam, On loops, dominators, and dominance frontiers, *ACM Trans. Program. Lang. Syst.* 24 (5) (2002) 455–490.
- [16] G. Rozenberg, *Handbook of Graph Grammars and Computing by Graph Transformation: vol. I. Foundations*, World Scientific Publishing, Singapore, 1997.
- [17] B.G. Ryder, M.C. Paull, Elimination algorithms for data flow analysis, *ACM Comput. Surv.* 18 (3) (1986) 277–316.
- [18] V.C. Sreedhar, G.R. Gao, Y.-F. Lee, A new framework for elimination-based data flow analysis using DJ graphs, *ACM Trans. Program. Lang. Syst.* 20 (2) (1998) 388–435.
- [19] V.C. Sreedhar, *Efficient program analysis using DJ graphs*, PhD thesis, School of Computer Science, McGill University, Montréal, Québec, Canada, 1995.
- [20] R.E. Tarjan, Graph theory and Gaussian elimination, in: J.R. Bunch, D.J. Rose (Eds.), *Sparse Matrix Computations*, Academic Press, New York, 1976, pp. 3–22.
- [21] J.S. Vitter, P. Flajolet, Average-case analysis of algorithms and data structures, in: J. van Leeuwen (Ed.), *Handbook of Theoretical Computer Science*, vol. A: Algorithms and Complexity, North-Holland, Amsterdam, 1990, pp. 431–524.
- [22] N. Wirth, *Programming in Modula-2*, second ed., Springer-Verlag, Heidelberg, 1983.