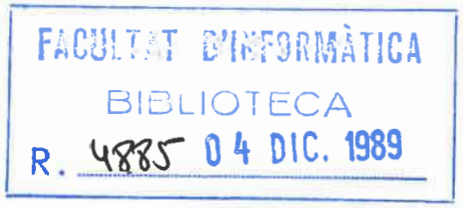
Algebraic simplification in computer algebra: an analysis of bottom-up algorithms

R. Casas
M.I. Fernández Camacho
J.M. Steyaert

Report LSI–88–16



ALGEBRAIC SIMPLIFICATION in COMPUTER ALGEBRA: an ANALYSIS of BOTTOM-UP ALGORITHMS

Abstract: We consider a class of *simplification* algorithms for *algebraic and logical expressions* which are of systematic use in Computer Algebra systems. This class is basically characterized by the fact that algorithms operate in a *bottom-up recursive* way on the expressions, i.e. start from the atomic terms -constants and variables- and perform the simplifications on larger and larger terms until the whole expression is ultimately proceeded; no backtracking or iterated process should intervein in the simplification. We show that under these quite general assumptions, it is possible to analyze precisely, and almost automatically, *the average size* of the resulting expressions -the gain in space- and the *average time complexity* of the process, which happens to be *linear*, whereas the worst-case behaviour is not in gereral.

Resum: Considerem una classe d'algorismes de *simplificació* per a *expressions algebraiques i lògiques* que són d'ús sistemàtic en sistemes d'àlgebra de computació. Aquesta classe està caracteritzada bàsicament pel fet que els seus algorismes operen sobre les expressions d'una manera *recursiva ascendent* (bottom-up), és a dir que parteixen dels termes atòmics -constants i variables- i efectuen simplificacions sobre termes cada cop més grans fins que processen finalment l'expressió completa; en la simplificació no intervenen processos iterats o de backtracking. Posem de manifest que sota aquestes hipòtesis fora generals, és possible analitzar de manera precisa, i quasi automàtica, la *longitud mitjana* de les expressions resultants -el guany en espai- i la *complexitat mitjana en temps* del procés, que resulta ser *lineal*, mentre que el comportament en el cas-pitjor en general no ho és.

ALGEBRAIC SIMPLIFICATION in COMPUTER ALGEBRA: an ANALYSIS of BOTTOM-UP ALGORITHMS [1]

Rafael Casas
Facultat de Informàtica
Universitat Politècnica de Catalunya
Pau Gargallo, 5
08028 Barcelona (SPAIN)

María-Inés Fernández Camacho
Facultad de Matemáticas
Universidad Complutense
28040 Madrid (SPAIN)

Jean-Marc Steyaert
Laboratoire d'Informatique
École Polytechnique
91128 Palaiseau (FRANCE)

Abstract : We consider a class of *simplification* algorithms for *algebraic and logical expressions* which are of systematic use in Computer Algebra systems. This class is basically characterized by the fact that algorithms operate in a *bottom-up recursive* way on the expressions, i.e. start from the atomic terms – constants and variables– and perform the simplifications on larger and larger terms until the whole expression is ultimately proceeded; no backtracking or iterated process should intervein in the simplification. We show that under these quite general assumptions, it is possible to analyze precisely, and almost automatically, *the average size* of the resulting expressions –the gain in space– and the *average time complexity* of the process, which happens to be *linear*, whereas the worst-case behaviour is not in general.

[1] This work has been partially supported by the French and Spanish governments and by the P.R.C. *Mathématiques et Informatique.*

1. Introduction

Simplification is one of the most common and hardest operations that are processed every second in any computer system dedicated to *Symbolic* and *Algebraic* manipulations. In particular, in Computer Algebra general systems –as *Macsyma, Reduce, Maple, Scratchpad, etc.*–, it is crucial to execute these basic operations with the greatest efficiency and to get a result, or better a normal form, as small as possible. Various strategies have been developped to achieve this goal, which always use a recursive definition of simplification, together with heuristics for the strategy and, quite often, hashing for a better efficiency.

In general, no precise estimations of running times nor of space requirements have been given on this type of algorithms. This is mainly due to the fact that these programs are intricate, so that the standard *flow-analysis* "à la Knuth" cannot be easily led. However, it seems to us that tight estimations of various crucial parameters related to the execution of these programs should be of great help in several situations:

- when designing algorithms: a sophisticated algorithm is quite often worse, on the "average", than a simpler one which happens to be very bad with small probability;
- when studying data organization: sparing memory quite often yields running time better efficiency;
- when improving programs: quoting J. Bentley [Be86], *when efficiency is important, the first step is to profile the system to find out where it spends its time*; J. Bentley was intending *profiling by simulation*; we had rather think of *profiling by mathematical analysis*.

In this paper, we propose a method for giving precise estimates of two types of quantities related to simplification algorithms:

- the size of the formula obtained after simplification,
- the running time of the algorithm.

Of course, our method cannot apply to any type of algorithm for at least two reasons: the fact that determining whether a problem is in a given complexity class is, in general, undecidable and the fact that programs can be so intricate that it is not possible to represent their run-time properties in a concise and comprehensive manner that would allow mathematical deductions and developments. Therefore, we have to be more precise about the general characteristics of the class of simplification programs that we are able to handle, which will imply some restrictions on their capabilities. Several attempts have been made in the last decade to provide systematic ways of analyzing the behaviour of programs; B. Wegbreit [We75][We76] and J. W. Ramshaw [Ra79] have proposed complexity assertions systems, analogue to the Floyd-Hoare method in formal semantics, that manipulate formulæ to express inductively probabilistic properties of data at run-time; closer to our approach, W. Burge [Bu75] suggests in his book that some systematic connection could be established between a (Lisp) program and its complexity; P. Flajolet and J.-M. Steyaert [FlSt87][St84] developped this idea for a class of algorithms working on tree structures in a recursive top-down way, without transforming the data. This last method allows a very systematic analysis of algorithms that perform some kind of *generalized syntactical differentiation* on algebraic expressions *without simplification* or, in the setting of formal language theory, of algorithms that behave as finite-state transducers on trees.

It is clear that usual simplification algorithms cannot be designed as top-down algorithms in the former framework, and that, indeed, one has to simplify small sub-expressions then collect them –therefore modifying the data– and recursively apply the simplification rules in a bottom-up traversal of the structure. Some even more involved situations can arise: consider for instance an algorithm that would try to factorize algebraic expressions using, basically, the distributivity law, $(a.b) + (a.c) = a.(b + c)$; then, one can easily see that this process cannot be executed in a strict bottom-up way, but that after a reduction the sub-expressions have to be scanned again, since new possibilities of simplication can have been created by the first reduction. This type of "*doubly recursive*" algorithm is out of the scope of our method: the situation becomes rapidly so complex that we cannot handle it. Nevertheless, we can analyze almost any kind of simplification that, under appropriate conditions on sub-expressions –equality tests in general, transforms this expression into one of its sub-expressions or some fixed expressions –most of the time a constant. To be more precise, we consider simplification rules of one of the following forms:

$$h(e_1, ..., e_n) \to e_j \quad \text{when} \quad e_{i_1} = e_{j_1}, ..., e_{i_k} = e_{j_k} \text{or},$$
$$h'(e_1, ..., e_n) \to c \quad \text{when} \quad e_{i_1} = e_{j_1}, ..., e_{i_k} = e_{j_k},$$

where e_i's are expression variables, $h(...)$ and $h'(...)$ denote generic forms of expressions called *headers*, and c is some constant term.
Many simplification algorithms are based on rewriting rules of this two types, which express, for instance, idempotency or nilpotency of operators, or even more subtil laws, such as $((x + y) - x) = y)$. However, associativity or distributivity, as already stated, are not expressible in their full generality in our system.

The tools for handling this class of programs are those used in [FlSt87], namely *generating functions* and *Complex Analysis*; the method is based on the translation of recursive definitions of combinatorial parameters and computational costs into equations on the power series associated to these quantities; then, we study the analytic properties of the solutions and, via translation lemmas, establish their asymptotic behaviour. The situation is however much more complex: we have to translate the combinatorial properties of the data and of the algorithm into infinite sets of equations over generating functions and complexity descriptors. Then the analytic study of the solutions is again more difficult, since we have to determine inductively the behaviour of the various power series involved in the equations. It turns out that all this can be achieved under very general conditions, and that the conclusions show that in general, the average size of the result is linear in the size of the input, and that the running time is linear on the average.

The paper is divided into three main parts. Section 2 is devoted to a case of simplification of the first type: starting from two constants a and b, we consider a binary operator, say $\wedge$, and given any expression f in the (free) algebra thus generated, we study:

- the average size of the equivalent expression one can obtain when it is known that $\wedge$ is idempotent –i.e. for all expression f, $f \wedge f = f$– and that this simplification rule is systematically applied,
- the average cost of the bottom-up recursive simplification algorithm which naturally achieves this reduction.

On this example, we illustrate in a very systematic way the general method we propose in this paper. We will be as precise and exhaustive as possible, in order to figure out the main points and technicalities.

Section 3 contains a case of simplification of the second type. Again the study is performed using similar methods. The main difference is in the nature of the inductive definitions, and therrefore recursive equations we have to handle.

Finally, Section 4 is devoted to the introduction of commutativity in the simplification system; we show how our method adapts to this new situation. We also suggests other extensions to general systems, our conviction being that in any special case one can estimate precisely average behaviour following the paradigm developped in this paper.

The work reported here was initiated in [CaSt86] and developped in [Fe87] and [Fe88]; some new developments suggested in this paper will receive a full treatment in a forthcoming paper[FeSt88].

2. Idempotent Binary Law

In this section , we consider a prototype of simplification algorithms, which is simple enough to be analyzed in detail and whose behaviour is typical of all more involved situations. On this model we will show the following features:

- the average size of an expression after simplification is linear in the size of the original one,
- the standard deviation of this random variable is proportional to the square root of the size of the

expression,

- the expected running time of the simplification algorithm is proportional to the size n of the input, whereas the worst case running time can be $O(n \log n)$ in some situations.

We first introduce some basic notions and notations that will be used throughout the paper.

Let $\mathcal{B}$ be the family of *pure binary trees*, i.e. binary trees without labels. This family is classically defined by the equation

$$\mathcal{B} = \diamond + \circ(\mathcal{B}, \mathcal{B}), \tag{1}$$

which can be viewed either as an equation in the free algebra where $\diamond$, the *leaf* symbol, is a constant and $\circ$, the *internal node* symbol, is a binary operator which applies to an ordered pair of trees, or as a convenient shortcut for the classical recursive definition of binary trees.

The generating function of the family $\mathcal{B}$ is the power series $B(z) = \sum_{n \geq 0} b_n z^n$, where b_n is the number of binary trees having n internal nodes. Given a tree t, we will denote by $|t|$ its number of internal nodes, referenced as its *size*. Then $B(z)$ can be equivalently defined as $B(z) = \sum_{t \in \mathcal{B}} z^{|t|}$; it is now routine to translate Eq. (1) into the following one for $B(z)$:

$$B(z) = 1 + z.B^2(z), \tag{2}$$

which analytically solves to:

$$B(z) = \frac{1 - \sqrt{1 - 4z}}{2z}, \tag{2'}$$

due to initial conditions.

The Catalan numbers b_n are known to evaluate to

$$b_n = \frac{1}{n+1}\binom{2n}{n} = \frac{1}{\sqrt{\pi}} 4^n n^{-3/2}(1 + O(1/n)), \tag{3}$$

as n tends to infinity.

Let $\mathcal{D}$ be the family of binary trees with two types of leaves a and b, which therefore satisfies the equation

$$\mathcal{D} = a + b + \circ(\mathcal{D}, \mathcal{D}). \tag{4}$$

We consider trees in $\mathcal{D}$ as machine representations of algebraic expressions built from a and b by repeated use of the binary operator $\circ$. We now assume that $\circ$ is moreover idempotent; hence expressions can in general be somehow reduced, and a natural way of computing the minimal equivalent expression, which exists in this case, is shown in Figure 1,written in some pascalish dialect. From the program it should be clear that, for instance, any expression (or tree as we will often say) in $\mathcal{D}$ whose leaves are all labelled with a simplifies to the one-leaf tree a itself, and similarly for b. More generally, any tree which can be obtained by substituting all the leaves of a binary tree (in $\mathcal{B}$) with a unique tree $t \in \mathcal{D}$, simplifies to a single copy of t, hence of $simplify(t)$. It should be mentioned too, that equality test really means structural identity; the way the test is implemented is not crucial: function *equal* has been given here for the sake of completeness.

We now turn to the proof of the three main results mentionned above.

2.1 Average simplification ratio

The goal of this paragraph is to study the average size of a random expression after simplification. As it is commonly done, it is reasonable to assume that all the expressions of a given size, say n the number of internal nodes in the tree, are equally likely. This assumption is not a restriction as to principles and does not affect the general form of the results.

We then perform the simplification algorithm on all the expressions of size n, and consider the average number of internal nodes in the trees thus obtained. In order to derive the asymptotic behaviour of this quantity, as n tends to infinity, we process in two main steps:

```
function simplify(tree:𝒟):𝒟; local tl,tr:𝒟
        if tree.degree = 0 then simplify:=tree
                           else  tl:=simplify(tree.left);
                                 tr:=simplify(tree.right);
                                 if equal(tl,tr) then simplify:=tl
                                                 else  simplify:=o(tl,tr) fi fi.

function equal(tree1,tree2:𝒟):boolean;
        if tree1.info≠tree2.info
                           then equal:=false
                           else  if tree1.degree=0
                                                 then equal:=true
                                                 else  if equal(tree1.left,tree2.left)
                                                       then equal:=equal(tree1.right,tree2.right)
                                                       else  equal:=false fi fi fi.
```

Figure 1: *Programs for equality testing and simplification of expressions with an idempotent law.*
Tree-variables consist in four fields: *degree* (0 for leaves, 2 for internal nodes), *info* (a, b or o) , *left* (for left subtree) and *right* (for right subtree).

- first, we obtain through combinatorial considerations, a set of algebraic equations on the power series associated to families $\mathcal{D}_t$ of trees which simplify to t, for each irreducible t,
- then, we study the analytical properties of these series and obtain a local equivalent of the size series around its dominant singularity; we then conclude using classical transfer lemmas.

Let $D(z) = \sum_{t\in\mathcal{D}} z^{|t|}$ denote the generating function of $\mathcal{D}$. From Eq. (4) we deduce that

$$D(z) = 2 + z\,D^2(z),$$

from which we derive the analytic expression

$$D(z) = \frac{1-\sqrt{1-8z}}{2z}. \tag{5}$$

This expression shows furthermore that $D(z)$ has radius of convergence $1/8$, and an easy combinatorial argument yields a closed formula for the number of trees in $\mathcal{D}$ of size n: $[z^n]D(z) = 2^{n+1}b_n$.

We turn now to the study of the generating function of the size of simplified trees,

$$S(z) = \sum_{t\in\mathcal{D}} |simplify(t)|\, z^{|t|}.$$

This series, at least apparently, does not satisfy any kind of nice functional equation of the above type; however it is possible to define it by an infinite recursive scheme.
Let $\mathcal{I}$ be the set of irreducible elements of $\mathcal{D}$, that is those trees which satisfy $t = simplify(t)$. This set is inductively defined by:

$$\mathcal{I} = a + b + o(\mathcal{I},\mathcal{I}) - \{o(t,t)|t\in\mathcal{I}\}. \tag{6}$$

For each $t\in\mathcal{I}$, let D_t denote the power series of trees which simplify to t:

$$D_t(z) = \sum_{simplify(u)=t} z^{|u|}, \tag{7}$$

so that:

$$S(z) = \sum_{t\in\mathcal{I}} |t|\, D_t(z). \tag{8}$$

Since a tree reduces to a single leaf iff all its leaves are labelled with the same letter, we have as induction basis the equations:

$$D_a(z) = D_b(z) = B(z). \tag{9}$$

Furthermore, for any pair (u, v) of distinct elements of $\mathcal{I}$, the tree $o(u, v)$ belongs to $\mathcal{I}$ and we have:

$$D_{o(u,v)}(z) = z\, D_u(z) D_v(z) + z(D_{o(u,v)}(z))^2. \tag{10}$$

Eq. (10) expresses the fact that whenever a tree simplifies to $o(u, v)$, either its left subtree simplifies to u, its right one to v and $u \neq v$, or both of them simplify to $o(u, v)$ producing $o(o(u, v), o(u, v))$ which in turn simplifies to $o(u, v)$. This inductive definition of series $D_t(z)$, for $t \in \mathcal{I}$, allows us to characterize algebraically the series $S(z)$:

PROPOSITION 1: *The power series $S(z)$ of the sizes of simplified trees is given by*

$$S(z) = \frac{D(z) - 2 - z \sum_{t \in \mathcal{I}} (|t| + 1) D_t^2(z)}{1 - 2z D(z)}.$$

PROOF: Let us first remark that

$$D(z) = \sum_{t \in \mathcal{I}} D_t(z). \tag{11}$$

In order to evaluate $S(z)$, we introduce the series in two variables

$$\Delta(z, y) = \sum_{t \in \mathcal{I}} y^{|t|} D_t(z), \tag{12}$$

so that we classically get an expression equivalent to that of Eq. (8) by differentiating Δ w.r.t. y and setting $y = 1$:

$$S(z) = \frac{\partial}{\partial y} \Delta(z, y)|_{y=1}. \tag{13}$$

Introducing variable y which marks the size of the simplified tree, we obtain from Eq. (10), under the same conditions:

$$y^{|o(u,v)|} D_{o(u,v)}(z) = yz\, y^{|u|} D_u(z)\, y^{|v|} D_v(z) + y^{-|o(u,v)|} z\, (y^{|o(u,v)|} D_{o(u,v)}(z))^2. \tag{14}$$

We now sum all these equations for all $t \in \mathcal{I}$, which gives after some transformations on the summations and reductions using Eq. (7):

$$\Delta(z, y) = 2 + yz\Delta^2(z, y) - z \sum_{t \in \mathcal{I}} (y^{2|t|+1} - y^{|t|}) D_t^2(z). \tag{15}$$

Differentiation w.r.t. y produces a linear equation in $S(z)$ which solves easily giving the expected expression. ∎

In the expression thus obtained for $S(z)$ the denominator is quite classical; it appears in a number of situations where the observed quantity is linear in the input size. Combinatorially, it corresponds to a recursion over the subtrees; analytically, it is easy to derive from Eq. (4') that

$$\frac{1}{1 - 2z\, D(z)} = \frac{D'(z)}{D^2(z)}.$$

The numerator is somewhat harder to analyze. From combinatorial considerations (the size of simplified trees is smaller than the original size), we know that the radius of convergence of $S(z)$ equals that of $D(z)$; the point is to determine the local behaviour of $\delta(z) = \sum_{t \in \mathcal{I}} (|t| + 1) D_t^2(z)$ in a neighbourhood of $z = 1/8$. There is no general analytical result for this kind of summations; almost any situation can be met. However, we have the following important fact:

PROPOSITION 2: *The series $\delta(z) = \sum_{t \in \mathcal{I}} (|t| + 1) D_t^2(z)$ is analytic for $|z| \leq \rho_\delta$, with $\rho_\delta > 1/8$.*

Before proving this proposition, let us make two remarks:

1. It is now easy to get a local equivalent for $S(z)$ around $z = 1/8$, its dominant singularity, and therefore, using the Darboux-Pólya method, to derive an asymptotic expression for the average size of simplified expressions.
2. This phenomenon is going to appear quite generally in all the types of simplification that we consider in this paper; intuitively, it seems to come from a reasonably fair repartition of trees in the simplication classes $\mathcal{D}_t$.

PROOF: The proof of Proposition 2 is based on Weierstrass's convergence criterion together with sharp local estimates of several series.

FACTS:

1. For all $t \in \mathcal{I}$, the series $D_t(z)$ has radius of convergence $1/4$ and $D_t(1/4) = 2$; the proof is easy by structural induction on elements of $\mathcal{I}$:
   - $D_a(z) = D_b(z) = B(z)$ satisfy this property,
   - $D_{\circ(u,v)}(z)$ has an explicit form in terms of $D_u(z)$ and $D_v(z)$,

$$D_{\circ(u,v)}(z) = (1 - \sqrt{1 - 4z^2 D_u D_v})/2z, \tag{16}$$

   from which one checks the inductive step, since all the series $D_t(z)$ have positive coefficients, thus defining increasing functions on the real positive axis.

The series $D_t(z)$ are therefore analytic functions in the disk $|z| < 1/4$.

2. For all $t \in \mathcal{I}$, with $|t| > 1$, we verify by induction on $|t|$ that $D_t(3/16) \leq \gamma^{|t|}$, for $\gamma = .3$ for instance. This step uses Eq. (16) and the standard local development of $(1 - x)^{1/2}$ as x vanishes.

3. We now consider the series $I(z)$ of irreducibles trees. From Eq. (6) which defines the set $\mathcal{I}$, we derive for $I(z)$ the following functional equation:

$$I(z) = 2 + z\, I^2(z) - z\, I(z^2). \tag{17}$$

This type of equation cannot be solved explicitely; however, the Darboux-Pólya method [Pó37] gives precise asymptotics for the coefficients of $I(z)$. Let us first remark that $[z^n]I(z) \leq [z^n]D(z)$, since $\mathcal{I} \subset \mathcal{D}$. Hence the radius of convergence ρ_I of $I(z)$ is greater than or equal to $\rho_D = 1/8$; in fact, the "implicit function" theorem asserts that it is the smallest positive real solution of Eq. (17) which satisfies simultaneously $2z\, I(z) = 1$; it is easy to approximate this value numerically, and to get $\rho_I = .1471742...$
Since $\rho_I^2 < \rho_I$, $I(z^2)$ is analytic for $|z| < \sqrt{\rho_I}$, and thus can be considered as a regular function independent of $I(z)$in a neighbourhood of $z = \rho_I$. As a consequence, we have:

$$[z^n]I(z) = [z^n]\frac{1 - \sqrt{1 - 4z(2 - zI(z^2))}}{2z} = [z^n]\frac{-(1 - \frac{z}{\rho_I})^{1/2}}{2\rho_I}(1 + O(\frac{1}{n})),$$

hence

$$[z^n]I(z) = \frac{1}{4\rho_I\sqrt{\pi}}\rho_I^{-n} n^{-3/2}(1 + O(\frac{1}{n})),$$

the last equation providing from the classical Newton binomial formula and Stirling approximation of factorials, or from the Darboux theorem (cf. [He74], e.g.).

4. Coming back to $\delta(z)$ as defined in Proposition 2 we evaluate the sum for $z = 3/16$; from Fact 2, we deduce that $\delta(3/16) = O(\sum_{t\in\mathcal{I}}(|t|+1)\,\gamma^{2|t|})$, then using Fact 3 and the uniformity of all the preceeding estimations, we get that $\delta(3/16) = O(\sum_{n>0} n^{-1/2}\,(\rho_I\gamma^2)^n$, with $\rho_I\gamma^2 < 1$; hence this numerical sum converges, and $\delta(z)$ having positive coefficients converges absolutely in the disk $|z| < 3/16$. The proof of Proposition 2 is thus completed. It should be noticed that our estimations are very loose, but sufficient for our purpose.■

We are now able to prove the main result of this paragraph, which concerns the asymptotic behaviour of the average simplification ratio, i.e. the average value of the ratio between the sizes of an expression after and before simplification: $\mathbf{s}_n = \dfrac{[z^n]S(z)}{n\,[z^n]D(z)}$.

THEOREM 1A: *The average simplification ratio* s_n *tends to a constant* σ *as* n *tends to infinity, where*

$$\sigma = 1 - \delta(1/8)/16 = .8196...$$

and $\delta(z) = \sum_{t\in\mathcal{I}}(|t|+1)D_t^2(z)$.

PROOF: From Proposition 1 we have

$$S(z) = \frac{D(z) - 2 - z\,\delta(z)}{1 - 2z\,D(z)};$$

since $\delta(z)$ is regular for $|z| < 3/16$, the dominant singularity of $S(z)$ is located at $z = 1/8$, and in this neighbourhood, $S(z)$ is seen to be equivalent to

$$S(z) = \frac{D(1/8) - 2 - \delta(1/8)/8}{\sqrt{1-8z}} + O(1).$$

Hence by the transfert lemmas,

$$[z^n]S(z) = \frac{2 - \delta(1/8)/8}{\sqrt{\pi}}\,8^n\,n^{-1/2}\,(1 + O(1/n)) \quad \text{and} \quad [z^n]D(z) = \frac{2}{\sqrt{\pi}}\,8^n\,n^{-3/2}\,(1 + O(1/n)). \tag{18}$$

The Theorem follows.■

Eq. (15) was the starting point for computing the expected size of trees after simplification, hence the average value of the simplification ratio; one can expect that, in the same manner, we will be able to estimate the variance of the size of simplified expressions. We give the main lines of this , quite often, tedious exercise.

Denoting this variance, as a function of the input size n, by v_n, we recall the wellknown fact that

$$v_n = \frac{[z^n]\left(\frac{\partial^2}{\partial y^2}\,\Delta(z,y)|_{y=1} + \frac{\partial}{\partial y}\,\Delta(z,y)|_{y=1}\right)}{[z^n]D(z)} - \left(\frac{[z^n]S(z)}{[z^n]D(z)}\right)^2. \tag{19}$$

Differentiating twice Eq. (15) w.r.t. y we obtain again a linear equation in $\frac{\partial^2}{\partial y^2}\,\Delta(z,y)$ which, after substituting $y = 1$, solves to

$$\Delta''_y(z,1) = \frac{\partial^2}{\partial y^2}\,\Delta(z,y)|_{y=1} = \frac{4z\,D(z)\,S(z) + 2z\,S^2(z) - \xi(z)}{1 - 2z\,D(z)}, \tag{20}$$

where

$$\xi(z) = 3z\sum_{t\in\mathcal{I}}|t|(|t|+1)\,D_t^2(z). \tag{21}$$

In order to evaluate asymptotics for the Taylor coefficients of $\Delta''_y(z,1)$ we encounter the same problem as for $S(z)$. Function $\xi(z)$ has to be shown to be analytic in a domain larger than the disk of convergence of $\Delta''_y(z,1)$, namely $\{z/|z| < 1/8\}$. It is very similar to $\delta(z)$ and the proof of Proposition 2 clearly applies to $\xi(z)$, the only change appearing in Fact 4, without modifying the conclusion that $\xi(z)$ is analytic for $|z| < \rho_\xi$ for some $\rho_\xi > 1/8$.

Local developments, around $z = 1/8$, of $S(z)$ and $\Delta''_y(z,1)$ are now possible although tedious. In fact we have to expand these functions to the third term since contributions of the first two terms are going to vanish, as is usually the case in this kind of situation. The following expansions have been first done by hand

(cf. [Fe88]), then checked on the *Maple* system. We ultimately obtain:

$$
\begin{aligned}
D(z) &= \frac{1-\sqrt{1-8z}}{2z} \\
&= 4 - 4(1-8z)^{1/2} + 4(1-8z) - 4(1-8z)^{3/2}) + O((1-8z)^2), \\
S(z) &= \frac{D(z) - 2 - z\delta(z)}{1 - 2zD(z)} \\
&= (2 - \frac{1}{8}\delta(\frac{1}{8}))(1-8z)^{-1/2} - 4 + (4 + \frac{1}{8}\delta(\frac{1}{8}) + \frac{1}{64}\delta'(\frac{1}{8})(1-8z)^{1/2} + O(1-8z), \\
\Delta''_y(z,1) &= d_1\,(1-8z)^{-3/2} + d_2\,(1-8z)^{-1/2} + d_3\,(1-8z)^{1/2} + d_4, \\
\text{with} \quad d_1 &= (1 - \delta(1/8)/16)^2, \\
d_2 &= -\frac{1}{256}(1280 - 64\delta(1/8) + 256\xi(1/8) + 32\delta'(1/8) - 2\delta(1/8)\delta'(1/8) + 3\delta(1/8)^2), \\
d_3 &= \frac{1}{256}(-2048 + 64\delta'(1/8) - 32\delta(1/8) - 256\xi(1/8) - 6\delta(1/8)\delta'(1/8) \\
&\qquad + 3\delta(1/8)^2 + \delta'(1/8)^2 - 16\delta''(1/8) + \delta(1/8)\delta''(1/8)), \\
d_4 &= 16 - 64z + O((1-8z)^2).
\end{aligned}
$$

Applying the Darboux theorem to these local expansions we obtain from Eq.(19) the leading term for the variance v_n.

THEOREM 1B: *The variance v_n of the distribution of the sizes of trees after simplification is asymptotically linear in the size of the input tree:*

$$v_n = \overline{v}\,n(1 + O(1/n))$$

with $\overline{v} = 1/16\delta(1/8) - 1/8\delta'(1/8) - 3/256\delta^2(1/8) + 1/128\delta(1/8)\delta'(1/8) - 1/2\xi(1/8)$ *is a constant which numerically evaluates to* 0.2166...

2.2 Average cost of simplification

We now turn to the evaluation of the average cost of simplification. Throughout this section and the following ones, we will consider as running time an ideal measure based essentially on pointer manipulations in a Pascal implementation of the algorithm, or on their equivalent (in terms of *cons*'s, *car*'s and *cdr*'s) in a Lisp implementation; the overhead due to control in loops, branchings or procedure calls will be neglected, the main reason being that although the general flavour of the costs keep identical in real or idealized life, mathematical formulæ have a nicer look in the later than in the former, and are therefore easier to handle and to expose. A full and exact treatment would be possible in every special case, along the same lines.

Quite generally, following the notations and the ideas developped in [FlSt87], we will associate to a program (or a portion of program) $\mathcal{P}$, a series $\tau P(z)$ that relates the cost of execution of $\mathcal{P}$ to the input size; more precisely, denoting the cost of running $\mathcal{P}$ on input e by $\tau\mathcal{P}(e)$ we define this *complexity descriptor* $\tau P(z)$ by

$$\tau P(z) = \sum_e \tau\mathcal{P}(e)\, z^{|e|}. \tag{22}$$

The rules of the *Complexity Calculus* of [FlSt87] still apply in many situations: composition, conditionals, iterations over subtrees... However, a new phenomenon arises now, in connection with bottom-up recursion: the comparison test in the algorithm is performed on the trees obtained *after* simplification of the inputs and not, as previously studied on the inputs themselves; the cost relative to this test is therefore more difficult to evaluate. There are two ways of deriving an expression for $\tau simp(z)$, the complexity descriptor of function *simplify* defined in Fig 1: one comes back to the definition of complexity descriptors and by successive rearrangements yields the expression, while the second one is closer to the decomposition in "reduction" classes already used in the previous paragraph. We will prefer this later more concise treatment.

In the sequel we will make the convention that testing any combination of atomic propertys on roots of trees will be charged one unit; assignments ":=" will be free, as well as branching in conditional statements. Thus, in the analysis of equality test between two trees u and v, the cost $\tau eq(u,v)$ will be the number of nodes visited in one of the two trees.

We know prove:

PROPOSITION 3: *The complexity descriptor $\tau simp(z)$ of the simplification algorithm has the algebraic expression:*

$$\tau simp(z) = \frac{D(z) + M(z)}{1 - 2z\,D(z)},$$

where

$$M(z) = z \sum_{u,v \in \mathcal{I}} \tau eq(u,v)\, D_u(z)\, D_v(z).$$

PROOF: For all $t \in \mathcal{I}$, let us consider the series $\tau simp_t(z)$ of the simplification costs of those trees which reduce to t. We thus have

$$\tau simp_a(z) = D_a(z) + 2z\, D_a(z)\, \tau simp_a(z) + z\, D_a^2(z), \tag{23a}$$

where the first term represents the first test in procedure *simplify*, the second the recursive calls, and the third one the test for equality between the simplified resulting subtrees which has here always unit cost. For any $t = o(u,v) \in \mathcal{I}$, we now have, by the same analysis and use of general complexity rules:

$$\begin{aligned} \tau simp_t(z) = \; & D_t(z) \\ & + z\, \tau simp_u(z)\, D_v(z) + z\, D_u(z)\, \tau simp_v(z) + \tau eq(u,v) z\, D_u(z)\, D_v(z) \\ & + 2z\, \tau simp_t(z)\, D_t(z) + (2|t| + 1) z\, D_t^2(z). \end{aligned} \tag{23b}$$

In this equation the first term is the same as above, the second, third and fourth ones to the case where the left subtree reduces to u whilst the right one reduces to v, and the last two terms to the situation where both subtrees reduce to t itself.

We just have to sum all these equations, rearrange the righthand-side expression, solve the linear equation in $\tau simp(z) = \sum_{t \in \mathcal{I}} \tau simp_t(z)$ to obtain the expression of Proposition 3.∎

In this expression, all the quantities, but $M(z)$, are now wellknown. In fact all we know about $M(z)$ is that its radius of convergence is at least $1/8$, since simplification is polynomial, as will be shown later on. A nice situation would be that this radius be strictly greater than $1/8$, and we could use the scheme of subsection 2.1. Unfortunately this is not the case: let us simply remark that the coefficients of series $M(z)$ are exponent-wise larger than those of series $\sum_{u,v \in \mathcal{I}} D_u(z) D_v(z) = D^2(z)$ and, therefore, that $M(z)$ has radius of convergence $1/8$. Next, and this is not a rigourous statement, we know, by experience, that comparing two random trees has a constant cost; we are therefore inclined to think that $M(z)$ has its dominant singularity in $z = 1/8$, and that it expands locally as a $O(\sqrt{1-8z})$. The next proposition will establish this crucial property more formally; for a fully detailed proof, the reader should refer to [Fe88].

PROPOSITION 4: *The series $M(z)$ is analytic for $|z| \leq 1/8|$, $z \neq 1/8$, and in a neighbourhood of $z = 1/8$ has the asymptotic equivalent:*

$$M(z) = g(z)\sqrt{1-8z} + h(z),$$

where $g(z)$ and $h(z)$ are analytic in a neighbourhood of $z = 1/8$.

The proof of Proposition 4 has to be divided into three parts; first, we give for $M(z)$ an equivalent form, basically by expanding the recursion once, thus getting an equivalent expression more convenient to forthcoming analytic considerations; we then show that series of the type $\sum_{u,v \in \mathcal{I}} D_u^i(z) D_v(z)$ have the property expected for $M(z)$ and we conclude by analytic continuation of $M(z)$ outside its circle of convergence, except in $z = 1/8$.

LEMMA 1: *The series $M(z)$ satisfies the equation*

$$M(z) = \frac{D(z)\left(1 - 2z^2 \sum_{t\in\mathcal{I}} D_t^2(z)\right) + 2z^2(1 - zD(z))\, M_2(z) - 2z^3 M_3(z) + H(z)}{1 - z^2\left(D^2(z) + \sum_{t\in\mathcal{I}} D_t^2(z)\right)},$$

where

$$H(z) = z^3 \Big(\sum_{t\in\mathcal{I}} D_t^2(z)\Big)^2 + z^3 \sum_{t\in\mathcal{I}} (2|t|+1) D_t^4(z) - 2 \quad ,$$

and, for $i = 2, 3$,

$$M_i(z) = \sum_{u,v\in\mathcal{I}} \tau eq(u,v)\, D_u^i(z)\, D_v(z).$$

PROOF: Let us consider the quantity $\tau eq(u,v)$; it is defined recursively by the system:

$$\tau eq(u,v) = \begin{cases} 1, & \text{if } |u| = 0 \text{ or } |v| = 0; \\ 1 + \tau eq(u.left, v.left) + \tau eq(u.right, v.right)\, \chi eq(u.left, v.left), & \text{otherwise.} \end{cases} \quad (24)$$

Translating this definition in terms of power series we obtain for $M(z)$:

$$M(z) = 4z\, B(z)\, D(z) - 4z\, B^2(z) + z^2\,(N_1(z) + N_2(z)),$$

where, with $\mathcal{I}'$ denoting $\mathcal{I}\backslash\{a,b\}$,

$$N_1(z) = z \sum_{u,v\in\mathcal{I}'} \tau eq(u,v)\, D_u^2(z)\, D_v^2(z) + 2z \sum_{u,v\in\mathcal{I}'} \tau eq(u,v)\, D_u^2(z)\, D_{v.left}(z)\, D_{v.right}(z),$$

and

$$N_2(z) = z \sum_{u,v\in\mathcal{I}'} \tau eq(u,v)\, D_{u.left}(z)\, D_{u.right}(z)\, D_{v.left}(z)\, D_{v.right}(z).$$

After some transformations these last two equations ultimately produce:

$$N_1(z) = \sum_{u,v\in\mathcal{I}} \tau eq(u,v) D_u^2(z) D_v(z) - 2B^2(z)\, D(z) - 2B(z) \sum_{t\in\mathcal{I}} D_t^2(z) + 4B^3(z) - z \sum_{u,v\in\mathcal{I}'} \tau eq(u,v) D_u^2(z) D_v(z),$$

and

$$N_2(z) = z\left(D^2(z) - \sum_{t\in\mathcal{I}} D_t^2(z)\right)^2 + D^2(z)\, M(z) - 2z\, D(z) M_2(z) + 4z\, B^2(z) \sum_{t\in\mathcal{I}} D_t^2(z) - 4z\, B^4(z)$$
$$+ M(z) \sum_{t\in\mathcal{I}} D_t^2(z) - 2z\, M_3(z) + z \sum_{u,v\in\mathcal{I}} \tau eq(u,v) D_u^2(z) D_v^2(z).$$

We thus obtain a linear equation in $M(z)$ which gives the expression of Lemma 1. ∎

In this expression for $M(z)$ it is easy to see (and prove) that the denominator does not vanish for $|z| \leq 1/8$, and that, in the numerator, the series $H(z)$ has a radius of convergence strictly greater than $1/8$. Therefore, we just have to know the precise behaviour of quantities $M_2(z)$ and $M_3(z)$ to determine that of $M(z)$. Let us start by a local expansion around $z = 1/8$. We prove the following:

LEMMA 2: *The series $M_2(z)$ and $M_3(z)$ have radius of convergence $1/8$, and around $z = 1/8$ satisfy the local developments:*

$$M_i(z) = g_i(z)\sqrt{1 - 8z} + h_i(z), \quad i = 2, 3,$$

where $g_i(z)$ and $h_i(z)$ are analytic around $z = 1/8$.

PROOF : The value of the radius of convergence is immediate from inequalities

$$D(z)\sum_{t\in\mathcal{I}} D_t^i(z) \le M_i(z) \le D(z)\sum_{t\in\mathcal{I}} (2|t|+1)\, D_t^i(z).$$

Local expansions are more tricky. We first consider, for $u \in \mathcal{I}$ the series $\mu_u(z) = \sum_{v\in\mathcal{I}} \tau eq(u,v)\, D_v(z)$, so that $M_i(z) = \sum_{u\in\mathcal{I}} \mu_u(z) D_u^i(z)$. Obviously, $\mu_a(z) = \mu_b(z) = D(z)$, and for all $t = o(u,v) \in \mathcal{I}$ we have from Eq. (24) the inductive equation

$$\mu_t(z) = D(z) + z\,\mu_u(z)\, D(z) + z\,\mu_v(z)\, D_u(z) + z\,\nu_{u,v}(z), \tag{25}$$

where $\nu_{u,v}(z) = \sum_{t\neq a,b} \tau eq(u, t.left) D_t^2(z) + \sum_{t\neq u} \tau eq(v,t) D_{o(u,t)}^2(z) - \sum_t \tau eq(u,t) D_t^2(z) - \tau eq(u,v) D_u^2(z)$, all summations being taken over $\mathcal{I}$, as usual.
From Eq. (25) we derive by induction:

1- since for all $u, v \in \mathcal{I}$, series $\nu_{u,v}(z)$ are analytic for $|z| \le \rho_\delta$, with $\rho_\delta > 1/8$, series $\mu_t(z)$ are analytic for $|z| \le 1/8, z \neq 1/8$;

2- around $z = 1/8$, series $\mu_t(z)$ admit local developments of the form: $\mu_t(z) = g_t(z)\sqrt{1-8z} + h_t(z)$; furthermore,in a neighbourhood of $z = 1/8$, series $g_t(z)$ and $h_t(z)$ are analytic and their modulii $|g_t(z)|$ and $|h_t(z)|$ are strictly and uniformly bounded above by $\kappa + \lambda|t|$: one can choose for instance $\kappa = 5$ and $\lambda = 4(3\sqrt{2}-2)$; this property derives easily from the fact that $D_t(1/8) \le B(1/8) = 4 - 2\sqrt{2}$.

Using now again Weierstrass convergence criterion and the fact that $I(z)$, the series of irreducible trees, has a radius of convergence $\rho_{\mathcal{I}} > 1/8$, we obtain by summation, that series $g_2(z) = \sum_{t\in\mathcal{I}} g_t(z) D_t^2(z)$ and $h_2(z) = \sum_{t\in\mathcal{I}} h_t(z) D_t^2(z)$ converge absolutely and uniformly in a neighbourhood of $z = 1/8$. So, by Weierstrass double series theorem (see [He 74] e.g.), they are analytic inside this neighbourhood and therefore we obtain the desired decomposition, $M_2(z) = g_2(z)\sqrt{1-8z} + h_2(z)$. The same property holds true for $M_3(z)$. ∎

The last part of the proof of Proposition 4 consists in showing that $M(z)$ has no other singularity on its circle of convergence than $z = 1/8$. It is clearly sufficient to prove this property for $M_2(z)$ and $M_3(z)$; but these series can easily be continued analytically outside their circle of converge by growth considerations analoguous to those presented in the proof of Lemma 2.

This completes the proof of Proposition 4.∎

We can now conclude as to the average running time of the simplification algorithm.

THEOREM 2: The average cost $\overline{\tau simp_n} = \dfrac{[z^n]\tau simp(z)}{[z^n]D(z)}$ of running the simplification algorithm for an idempotent law on binary trees of size n is asymptotically linear as n tends to infinity:

$$\overline{\tau simp_n} = \sigma\, n + O(1),$$

where σ is a real number which depends on the actual implementation.

PROOF : From the considerations above, the series $\tau simp(z)$ has $1/8$ as radius of convergence, a unique dominant singularity at $z = 1/8$ and around this point a local development of the form:

$$\tau simp(z) = \frac{a(z)}{\sqrt{1-8z}} + b(z),$$

where $a(z)$ and $b(z)$ are analytic around $z = 1/8$. We can therefore apply the Darboux theorem and deduce that $[z^n]\tau simp(z) = \dfrac{a(1/8)}{\sqrt{\pi}}\, 8^n\, n^{-1/2}\,(1 + O(1/n))$. Simple algebraic manipulations lead to

$$a(1/8) = 4 + \frac{1}{48-\delta}\left(128 - 8\delta + \frac{\delta^2}{8} + \frac{1}{8}\sum_{t\in\mathcal{I}} (2|t|+1) D_t^4(1/8) - M_2(1/8) - \frac{M_3(1/8)}{4}\right),$$

with $\delta = \sum_{i \in \mathcal{I}} D_i^2 (1/8)$.
From the already known value of $[z^n]D(z)$, we conclude.■

3 Nilpotent binary law

In this section, we study the second type of simplification rules mentionned in the introduction: rules where a whole subexpression is replaced by a constant or a small expression, independent of the original size. This situation is complementary of the previous one and happens to introduce non trivial modifications in the analytic treatment that we have to perform. The general aspect of the results is however quite similar: the average size of the result is in general linear, the variance behaves similarly and the average running time of the program keeps linear. As before, we propose to illustrate the method and principles on an easy, but characteristic, example, inspired by algebraic substraction or division.

Let us now consider again a class $\mathcal{E}$ of binary trees with two types of leaves denoted a and e, defined by equation:

$$\mathcal{E} = a + e + \circ(\mathcal{E}, \mathcal{E}). \tag{26}$$

Class $\mathcal{E}$ is clearly isomorphic to class $\mathcal{D}$, and is introduced mainly for clarity: element e is playing the role of neutral element in our algebra, element a being a generator. The binary operator $\circ$ is now supposed to be nilpotent, so that any expression of the form $\circ(t, t')$ for some $t, t' \in \mathcal{E}$ can be rewritten as e, as soon as t and t' are shown to be equivalent. Obviously, every expression has a normal form which can be computed rather efficently according to a classical recursive scheme. An implementation of this algorithm is given in Figure 2. In order to distinguish it from Section 2 algorithm we call it *reduce* instead of *simplify*.

```
function reduce(tree:E):E; local tl,tr:E
        if tree.degree = 0 then reduce:=tree
                           else  tl:=reduce(tree.left);
                                 tr:=reduce(tree.right);
                                 if equal(tl,tr) then reduce:=e
                                                 else  reduce:=o(tl,tr) fi fi.
```

Figure 2: *Program for reduction of expressions with a nilpotent law.*
Implementation details and the code for function *equal* are given in Figure 1.

This algorithm is obviously rather different from the previous one; as a matter of fact an infinity of trees -the whole family $\mathcal{D}_a$- *simplify* to a, whereas a is just the normal form of itself w.r.t. *reduce*. The family of expressions which *reduce* to e -the neutral element- is rather complex as will be seen later on.

The combinatorial and algebraic parts of our study are performed along the same lines as in Section 2; the analytic part will be less easy. What seems important is that any other rewriting system of this kind is a combination of these two extreme cases, and apparently can be treated along these schemes.

We now turn to the study of the reduction ratio and of the cost of the reduction.

3.1 Average reduction ratio.

In this subsection we show how to estimate the average size of expressions after reduction. We will also derive the asymptotic behaviour of the variance of the distribution. As usual we start from the uniform distribution over expressions of the same size and the generating function $E(z) = \sum_{t\in\mathcal{E}} z^{|t|}$ will be of constant use. Actually we trivially derive from Eq. (26) that

$$E(z) = D(z) = \frac{1-\sqrt{1-8z}}{2z}, \tag{27}$$

and therefore has radius of convergence $1/8$.

The generating function of the size of reduced trees is defined by

$$R(z) = \sum_{t\in\mathcal{E}} |reduce(t)|\, z^{|t|}.$$

In order to get analytic information about its behaviour on its circle of convergence, we introduce the set $\mathcal{J}$ of irreducible trees: $\mathcal{J} = \{t \in \mathcal{E} | t = reduce(t)\}$. This set can be defined inductively by

$$\mathcal{J} = a + e + \circ(\mathcal{J}, \mathcal{J}) - \{\circ(t,t) | t \in \mathcal{J}\},$$

and is therefore isomorphic to set $\mathcal{I}$ of Section 2 although the algorithms are different. Its generating function is therefore the series $\mathcal{I}$ defined and studied previously.

Now, for every $t \in \mathcal{J}$, we consider the set $\mathcal{E}_t$ of expressions whose normal form, under reduction, is t, and the associated power series

$$E_t(z) = \sum_{reduce(u)=t} z^{|u|}. \tag{27}$$

Knowing these series, it will be possible to study the generating function of the sizes of expressions after reduction, namely the power series:

$$R(z) = \sum_{t\in\mathcal{E}} |reduce(t)|\, z^{|t|}, \tag{28a}$$

since we can express it as:

$$R(z) = \sum_{t\in\mathcal{J}} E_t(z). \tag{28b}$$

It is now easy to observe that power series $E_t(z)$ are defined inductively by:

$$\begin{aligned} E_a(z) &= 1, \\ E_e(z) &= 1 + z \sum_{t\in\mathcal{J}} E_t^2(z), \\ E_{\circ(u,v)}(z) &= z\, E_u(z)\, E_v(z), \text{ for all } u \neq v \in \mathcal{J}. \end{aligned} \tag{29}$$

The first and third equations are immediate from the definition of reduction; the second one expresses the fact that whenever the left and right subtrees have the same normal form the expression reduces to the neutral element.

We can now give an algebraic expression for $R(z)$.

PROPOSITION 5: *The power series $R(z)$ of the sizes of reduced expressions is given by*

$$R(z) = \frac{E(z) - 2 - z\sum_{t\in\mathcal{E}}(2|t|+1)E_t^2(z)}{1-2z\,E(z)}.$$

PROOF: The argument is quite similar to that in Proposition 1. Let us consider the power series in two variables

$$H(z,y) = \sum_{t\in\mathcal{J}} y^{|t|} E_t(z), \tag{30}$$

so that

$$R(z) = \frac{\partial}{\partial y} H(z,y)|_{y=1}. \tag{31}$$

Introducing variable y which marks the size of the simplified tree, we obtain from Eq. (29), under the same conditions:

$$y^{|o(u,v)|} E_{o(u,v)}(z) = yz\, y^{|u|} E_u(z)\, y^{|v|} E_v(z).$$

We now sum all these equations for all $t \in \mathcal{J}$, which gives after some transformations :

$$H(z,y) = 2 + yz\, H^2(z,y) - z \sum_{t\in\mathcal{J}} (y^{2|t|+1} - y^{|t|}) E_t^2(z). \tag{32}$$

Differentiation w.r.t. y produces a linear equation in $R(z)$ which solves easily giving the expected expression.■

The expression obtained for $R(z)$ is formally quite similar to that obtained for $S(z)$. Since $1 \leq |reduce(t)| \leq |t|$, for all $t \in \mathcal{E}$, the radius of convergence of $R(z)$ is again 1/8. The only problem is now to obtain information on the analytic behaviour of the sum in the numerator: $\eta(z) = \sum_{t\in\mathcal{J}} (2|t|+1) E_t^2(z)$. In fact, $\eta(z)$ behaves analogously to $\delta(z)$ in Proposition 2, and we prove similarly:

PROPOSITION 6: *The series $\eta(z) = \sum_{t\in\mathcal{E}} (2|t|+1) E_t^2(z)$ is analytic for $|z| \leq \rho_\eta$ with $\rho_\eta > 1/8$.*

PROOF: The main difference with Proposition 2 lies in the general form of the induction. In System (29), the series $E_e(z)$ is now defined implicitely. The main steps are as follows:

1. Let $|t|_e$ denote the number of leaves of t labelled by e.By induction on $t = o(u,v) \in \mathcal{J}$ we have from System (29):

$$E_t(z) = z^{|t|} (E_e(z))^{|t|_e}, \text{ for all } t \in \mathcal{J}. \tag{33}$$

Therefore all the series $E_t(z)$, except $E_a(z) = 1$, have the same radius of convergence as $E_e(z)$, say ρ_e, and $E_e(z)$ satisfies equation:

$$E_e(z) = 1 + z \sum_{t\in\mathcal{J}} z^{2|t|} (E_e(z))^{2|t|_e}. \tag{34}$$

2. Since the generating function of set $\mathcal{J}$ is series $I(z)$ defined by Eq. (17), whose radius of convergence is $\rho_I = .1471742...$ and which satisfies on the real positive axis, for $0 < z < \rho_I$, the inequality:

$$I(z) < \frac{1 - \sqrt{1 - 8z + 8z^2}}{2z},$$

and since, from Eq.(34), $E_e(z)$ satisfies, on the real positif axis, the inequality:

$$E_e(z) \leq 1 + z\, I((zE_e(z))^2),$$

a continuity argument and elementary numerical computations show that $E_e(z)$ can be computed for real positive z as long as $z \leq .1680541$ (at least). Hence, since coefficients of $E_e(z)$ are positive, this series is defined and analytic for $|z| \leq \rho_{E_e} \geq .1680541...$ Hence all the series $E_t(z)$ have this property.

3. Using now Eq. (33), the property above and the asymptotics for coefficients of $I(z)$, we conclude as in the proof of Proposition 2.■

The average value of the size of reduced expressions is now easily obtained by the Pólya-Darboux method from the explicit formula of Proposition 5. Similarly, the variance of the distribution can be computed from Eq. (32), which allows to derive an expression for $\frac{\partial^2}{\partial y^2} H(z,y)|_{y=1}$, as has been done in Section 2.

THEOREM 3: : *The average reduction ratio* $\mathbf{r}_n = \dfrac{[z^n]R(z)}{n[z^n]E(z)}$, *tends to a constant* $\mathbf{r}$ *as* n *tends to infinity, with*

$$\mathbf{r} = 1 - \eta(1/8)/16 = .8162...$$

where $\eta(z) = \sum_{t\in\mathcal{E}}(2|t|+1)E_t^2(z)$.
The variance v_n *of the size of reduced expressions of size* n *is asymptotically linear in* n, $v_n = \overline{v}\,n(1+O(1/n))$, *with*

$$\overline{v} = -3\mathbf{r}^2 + (5 + \frac{1}{64}\eta'(1/8))\,\mathbf{r} - \frac{1}{2}\chi(1/8) - 2 = .2469...$$

where $\chi(z) = z\sum_{t\in\mathcal{J}} 2|t|(2|t|+1)\,E_t^2(z)$.

3.2 Average cost of reduction

The average running time of the *reduce* process can be studied along the same lines as that of simplification. As was already noticed the two algorithms are structurally isomorphic, and in fact we can use the same method to estimate its asymptotic value. We show again that it happens to be linear in the size of the input, whereas the worst case can be $O(n\log n)$. Since most equations and developments are very close to those of Section 2, we present the main points and intermediate results. A full detailed version can be found in [Fe88].

Let $\tau red(z)$ be the complexity descriptor of the reduction algorithm, i.e. the power series $\tau red(z) = \sum_{t\in\mathcal{E}} \tau reduce(t)\,z^{|t|}$. As before, we consider the decomposition of this series according to the computed normal form. For each $t\in\mathcal{J}$, let $\tau red_t(z)$ denote the restriction of $\tau red(z)$ to all expressions which reduce to t. A direct inspection of the program shows that these restricted complexity descriptors can be defined inductively by:

$$\begin{aligned}
&\tau red_a(z) = E_a(z),\\
&\tau red_e(z) = E_e(z) + 2z\sum_{t\in\mathcal{J}} \tau red_t(z)E_t(z) + z\sum_{t\in\mathcal{J}}(2|t|+1)E_t^2(z),\\
&\tau red_t(z) = E_t(z) + z\,\tau red_u(z)E_v(z) + z\,\tau red_v(z)E_u(z),\ \text{for all } t = o(u,v)\in\mathcal{J}.
\end{aligned} \tag{34}$$

Summing all these equations we obtain an explicit formula for the overall complexity descriptor:
PROPOSITION 7: : *The complexity descriptor* $\tau red(z)$ *of the reduction algorithm is given by the expression:*

$$\tau red(z) = \frac{E(z)+N(z)}{1-2z\,E(z)},$$

where $N(z) = z\sum_{u,v\in\mathcal{J}} \tau eq(u,v)E_u(z)E_v(z)$.

Furthermore, from Eq. (24) and the inductive definition of $E_t(z)$ in Eq. (29), we obtain after some transformations a linear equation in $N(z)$ which solves to:

$$N(z) = \frac{E(z) - 2z^3\,(E(z)N_2(z) + N_3(z)) + V(z)}{1 - z^2\,(E^2(z) + \sum_{t\in\mathcal{J}} E_t^2(z)}, \tag{35}$$

where

$$V(z) = z^3\sum_{u,v\in\mathcal{J}} \tau eq(u,v)E_u^2(z)E_v^2(z) + z^3\sum_{t\in\mathcal{J}}(2|u|+1)E_t^4(z) - 2,$$

and, for i=2,3,

$$N_i(z) = \sum_{u,v\in\mathcal{J}} \tau eq(u,v)E_u^i(z)E_v(z).$$

In Eq. (35), the denominator does not vanish for $|z|\le 1/8$; in the numerator, the series $V(z)$ has a radius of convergence strictly greater than 1/8, and series $N_2(z)$ and $N_3(z)$ have radius of convergence 1/8 precisely.

The main step in the development is to determine the precise behaviour of these last two series on their circle of convergence and, in fact, around $z = 1/8$. This behaviour is summarized in the following technical lemma:

LEMMA 3: : *The series $N_2(z)$ and $N_3(z)$ are analytic for $|z| \leq 1/8$, except in $z = 1/8$ where they satisfy local developments of the form*

$$N_i(z) = g_i(z)\sqrt{1-8z} + h_i(z), \ i = 2, 3,$$

where $g_i(z)$ and $h_i(z)$ are analytic in a neighbourhood of $z = 1/8$.

The proof of Lemma 3 is analaguous to that of Lemma 2. It proves convenient to use series $\nu_u(z) = \sum_{v \in \mathcal{J}} \tau eq(u, v) E_v(z)$, for each $u \in \mathcal{J}$, which verify the inductive equations:

$$\nu_a(z) = \nu_e(z) = E(z),$$
$$\nu_t(z) = E(z) + zE(z)\nu_u(z) + zE_u(z)\nu_v(z) + \nu^{\star}_{u,v}(z), \text{ for all } t = o(u, v) \in \mathcal{J},$$

where

$$\nu^{\star}_{u,v}(z) = -z \sum_{t \in \mathcal{J}} \tau eq(u, t) E_t^2(z) - z\, \tau eq(u, v) E_u^2(z).$$

As to the bounds, one can choose $\kappa = \lambda = 6$, for instance. The regularity of the series for $z \neq 1/8$ is also derived from these expressions.

The same property holds true for $M(z)$, and we conclude:

THEOREM 4: *The average cost $\overline{\tau red_n} = \dfrac{[z^n]\tau red(z)}{[z^n]E(z)}$ of running the reduction algorithm for an nilpotent law on binary trees of size n is asymptotically linear as n tends to infinity:*

$$\overline{\tau red_n} = \overline{\rho}\, n + O(1),$$

where $\overline{\rho}$ is a real number which only depends on the actual implementation.

PROOF : From the considerations above, we obtain for $\overline{\rho}$ the value

$$\overline{\rho} = 4 + \frac{1}{7 - E_e(1/8)}\left(8 - \frac{1}{16}(N_2(1/8) + \frac{1}{4}N_3(1/8)) + \frac{\eta_1 + \eta_2}{128}\right),$$

where $\eta_1 = \sum_{t \in \mathcal{J}} (2|t| + 1) E_t^4(1/8)$ and $\eta_2 = \sum_{u,v \in \mathcal{J}} \tau eq(u, v) E_u^2(1/8) E_v^2(1/8)$.

4. Extensions

In this section we first consider the influence of commutativity on the equality test, therefore approaching more concrete situations. We then suggest that more general families of trees can be treated according to our methodology. These extensions are intended to provide material in favour of a general scheme for solving the analysis of bottom-up algorithms. It is our conviction that although no general quantitative theorem

has been proved, one can manage in any particular case to get estimates of the average running time using our paradigm.

4.1 Simplification with commutativity

The commutativity law is not a priori a simplification rule for expressions since, basically, it does not produce strictly shorter expressions; however, algorithms exist that still terminate in any case; furthermore, a normal form can usually still be defined and happens to be smaller, in general, than its non commutative equivalent. For instance $(x \wedge y) \wedge (y \wedge x)$ can be simplified to $x \wedge y$ only in the commutative case for an idempotemt binary law.

We present here the study of this situation which leads to new developments.

As before, we work with the family of binary trees $\mathcal{D}$, assuming that operator is idempotent and commutative. We analyze then the influence of commutativity on the size of expressions after simplification, as well as on the running time of the algorithm. In order to be more precise, we propose in Figure 3 an implementation of the simplification algorithm we study.

```
function simcom(tree:D):D; local tl,tr:D
        if tree.degree = 0
          then simcom:=tree
          else  tl:=simcom(tree.left);
                tr:=simcom(tree.right);
                if eqcoml(tl,tr)
                  then simcom:=tl
                  else  simcom:=o(tl,tr) fi fi.

function eqcom(tree1,tree2:D):boolean;
        if tree1.info≠tree2.info
          then eqcom:=false
          else  if tree1.degree=0
                  then eqcom:=true
                  else  if eqcom(tree1.left,tree2.left)
                          then eqcom:=eqcom(tree1.right,tree2.right)
                          else  if not eqcom(tree1.left,tree2.right)
                                  then eqcom:=false
                                  else eqcom:=eqcom(tree1.right,tree2.left) fi fi fi fi.
```

Figure 3: *Programs for equality testing and simplification of expressions with an idempotent and commutative law .*

Implementation details are given in Figure 1.

The main difference, when commutativity is introduced, is in the definition of reduction classes. Let $\mathcal{I}_c$ denote the set of irreducible trees t defined as $t = simcom(t)$:

$$\mathcal{I}_c = a + b + \{o(u,v) | \, u,v \in \mathcal{I}_c \quad \neg eqcom(u,v)\}. \tag{36a}$$

This definition is however not very convenient for future developments, so that we had rather rewrite it as:

$$\mathcal{I}_c = a + b + o(\mathcal{I}_c, \mathcal{I}_c) - \{o(u,v) | \, u,v \in \mathcal{I}_c \quad eqcom(u,v)\}. \tag{36b}$$

It should be observed that algorithm *simcom* does not produce a unique element for each equivalence class w.r.t. *eqcom*: for instance both $o(a,b)$ and $o(b,a)$ can be produce as output although they are equivalent. For each $t \in \mathcal{I}_c$, we denote by $COM(t)$ the equivalence class of t in $\mathcal{I}_c$.

Keeping in mind the remark above, we can now define recursively the reduction classes $\mathcal{D}_t = \{u \in \mathcal{D} | t = simcom(u)\}$ produced by the algorithm, and express their generating functions $D_t(z)$ by the system:

$$\begin{aligned} D_a(z) &= D_b(z) = B(z), \\ D_{o(u,v)}(z) &= z\, D_u(z) D_v(z) + z\, D_{o(u,v)}(z) \sum_{t \in COM(o(u,v))} D_t(z), \text{ for each } o(u,v) \in \mathcal{I}_c. \end{aligned} \tag{37}$$

Furthermore, we have the two following basic properties of reduction classes and equivalence classes in $\mathcal{I}_c$:

1. for all $u, v \in \mathcal{I}_c$, if $u \in COM(v)$ then $D_u(z) = D_v(z)$,
2. for each $t \in \mathcal{I}_c$, $|COM(t)| = 2^{|t|}$.

These two fact are proved inductively on the structure.
We can now rewrite the System (37) as:

$$\begin{aligned} D_a(z) &= D_b(z) = B(z), \\ D_t(z) &= z\, D_u(z) D_v(z) + 2^{|t|} z\, D_t(z), \text{ for all } t = o(u,v) \in \mathcal{I}_c. \end{aligned} \tag{38}$$

We can now indicate the main steps and intermediate results in the study of the average size of simplified commutative expressions and of the average running time. In the sequel notations are those of Section 2.

1. The generating function $I_c(z)$ of irreducible trees $\mathcal{I}_c$ satisfies:

$$I_c(z) = 2 + z\, I_c^2(z) - z\, I_c(2z^2),$$

and has radius of convergence $\rho_{I_c} > 1/8$.

2. The series $D_t(z)$ are analytic functions in the disk $|z| \leq 4/25$ (at least).

3. For all $t \in \mathcal{I}_c$, $|D_t(z)| < \frac{5}{4}(\frac{2}{9})^{|t|}$ in the disk $|z| \leq 4/25$, and $|D_t(z)| < \frac{5}{4}(\frac{1}{6})^{|t|}$ in the disk $|z| \leq 1/8$.

4. In the study of the cost of commutative simplification, the cost of commutative comparison is defined by:

$$\tau eqcom(u,v) = \begin{cases} 1, & \text{if } |u||v| = 0; \\ 1 + \tau eqcom(u.left, v.left) & \\ +\tau eqcom(u.right, v.right)\, \chi eqcom(u.left, v.left) & \\ +(\tau eqcom(u.left, v.right) & \\ \quad +\tau eqcom(u.right, v.left)\, \chi eqcom(u.left, v.right)) & \\ \quad (1 - \chi eqcom(u.left, v.left)), & \text{otherwise,} \end{cases}$$

where $\chi eqcom(u,v)$ denotes the characteristic function of predicate *eqcom*.
From this inductive definition, we deduce that $\tau eqcom(u,v) \leq (2|u|+1)(2|v|+1)$. Furthermore, commutativity eliminates symmetry: $\tau eqcom(u,v) \neq \tau eqcom(v,u)$; hence we must introduce two power series, for each $u \in \mathcal{I}_c$: $\mu_u(z) = \sum_{v \in \mathcal{I}_c} \tau eqcom(u,v)\, D_v(z)$ and $\nu_u(z) = \sum_{v \in \mathcal{I}_c} \tau eqcom(v,u)\, D_v(z)$.
We then prove that $\mu_u(z) \leq 2^{|u|}(|u|+1)^2$ and $\nu_u(z) \leq 2^{|u|}(|u|+4)$ in the disk $|z| \leq 1/8$.

Apart from these minor technical variations, the algebraic and analytic developments are totally similar to those of Section 2. We therefore state without more details the results of these (sometimes tedious) computations.

THEOREM 5: *The average value of the reduction ratio for expressions of size n under commutative idempotent simplification tends to a constant σ_c, as n tends to infinity; this constant evaluates to:*

$$\sigma_c = 1 - \frac{\xi(1/8)}{16} = .8066...$$

with $\xi(z) = \sum_{t \in \mathcal{I}_c} (|t|+1) 2^{|t|}\, D_t^2(z)$.

The variance v_n of the distribution of the sizes of expressions (of size n) after commutative idempotent simplification verifies asymptotically:

$$v_n = \overline{v}n(1 + O(1/n)),$$

with $\overline{v} = -3\sigma_c^2 + (\frac{1}{64}\xi'(1/8) + 5)\sigma_c - \frac{1}{2}\theta(1/8) - 2 = .2458...$ *where* $\theta(z) = 3z \sum_{t\in\mathcal{I}_c} |t|(|t|+1)2^{|t|}D_t^2(z)$.

THEOREM 6: *The average running time $\overline{\tau simcom_n}$ of the simplification algorithm (for a commutative and idempotent binary law) on expressions of size n is asymptotically linear in n, as n tends to infinity:*

$$\overline{\tau simcom_n} = \overline{\sigma_c}n(1 + O(1/n)),$$

where $\overline{\sigma_c}$ is a constant which depends only on the implementation.

With our usual conventions on the measure, constant $\overline{\sigma_c}$ expresses as:

$$\overline{\sigma_c} = 2 + \frac{1}{32 - 2c_1}(64 - 4c_1 + \frac{1}{16}c_1^2 + \frac{1}{2}c_2 + \frac{1}{4}m_2 - \frac{1}{16}(m_3 + 2n_3)),$$

with

$$\begin{aligned}
c_1 &= \sum\nolimits_{t\in\mathcal{I}_c} 2^{|t|}D_t^2(1/8),\\
c_2 &= \frac{1}{2}/,\sum\nolimits_{u,v\in\mathcal{I}_c \ \&\ eqcom(u,v)} \tau eqcom(u,v)2^{2|u|}D_u^3(1/8)(1 + \frac{1}{4}2^{|u|}D_u(1/8))\\
&\quad + \frac{1}{8}/,\sum\nolimits_{u,v\in\mathcal{I}_c} \tau eqcom(u,v)2^{|u|+|v|}D_u^2(1/8)D_v^2(1/8),\\
m_i &= \sum\nolimits_{u\in\mathcal{I}_c} \mu_u(1/8)2^{(i-1)|u|}D_u^i(1/8), \text{ for } i = 2,3,\\
n_3 &= \sum\nolimits_{u\in\mathcal{I}_c} \nu_u(1/8)2^{2|u|}D_u^3(1/8).
\end{aligned}$$

4.2 Forthcoming extensions

The results presented are just a fraction of what can be analyzed with our method. In fact one can also deal with ternary trees or k-ary trees for any k, or even more generally with any simple family of trees, in the sense of [MeMo78]. It is then possible to take under consideration some very reallistic distributions on the family of expressions. In each particular case the method applies with more or less tricky technicalities. On a practical point of view, some of the mathematical developments could be considered as "useless"; in many situations numerical computations would give excellent approximations of series and of the behaviour of the leading terms in all our developments.

Furthermore we can also analyse more complex rewriting rules, as long as they keep strictly bottom-up. In this class, one could express a one-pass factorisation algorithm inspired by distributivity laws. Part of these investigations is reported in [Fe88]. A more global treatment will appear in [FeSt88].

Bibliography

[Be86] J. Bentley: *Programming Pearls*, Addison-Wesley P.C., Reading (1986).

[Bu75] W. Burge: *Recursive Programming Techniques*, Addison-Wesley P.C., Reading (1975).

[CaSt86] R. Casas & J.-M. Steyaert: "Bottom-Up Recursion in Trees", in Proc. CAAP86, Lect. Notes in Computer Science **214** (1986), 172-182.

[Fe87] M.-I. Fernández-Camacho: "Tamaño medio de árboles simplificados", to appear in Qüestiió.

[Fe88] M.-I. Fernández-Camacho: Análisis medio de algoritmos de reducción sobre árboles", Tesis, Madrid (1988).

[FeSt88] M.-I. Fernández-Camacho & J.-M. Steyaert: Research Report in preparation.

[FlSt87] P. Flajolet & J.-M. Steyaert: "A Complexity Calculus for Recursive Tree Algorithms", Math. Systems Theory, **19** (1987), 301-331.

[MeMo78] A. Meir & J.W. Moon: "On the Altitude of Nodes in Random Trees", Can. J. Math. **XXX** (1978), 997-1015.

[Pó37] G. Pólya: "Kombinatorische Anzahlbestimmungen für Gruppen, Graphen und chemische Verbindungen", Acta Mathematica, **68** (1937), 145-254.

[Ra79] L. Ramshaw: Formalizing the Analysis of Algorithms, Ph.D. thesis, Stanford University, (1979).

[St84] J.-M. Steyaert: Structure et complexité des algorithmes, Thèse, Paris, (1984).

[We75] B. Wegbreit:"Mechanical Program Analysis", Comm. ACM, **18** (1975), 528-532.

[We76] B. Wegbreit: "Verifying Program Performance",J. Assoc. Comput. Mach., **23** (1976), 691-699.