4

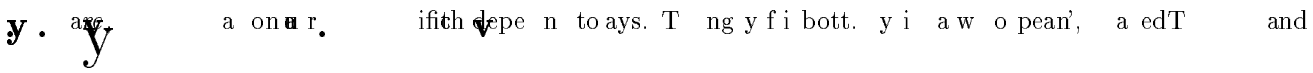
Simplifying Square Roots of Square Roots by Denesting

David J. Jeffrey
The University of Western Ontario

Albert D. Rich
Soft Warehouse, Inc.

[Landau92b], reasons are given for preferring the left-hand side of (4.2). However, we assume that in general users would want denesting to be discovered[4].

Another reason for denesting is *reliable simplification*. For both people and computers, there is a danger that the result of a mathematical simplification will depend upon the order in which rules are applied. For example, the simplification of $\sqrt{(1-\sqrt{2})^2}$ can proceed two ways. The first way is called 'top-down' by those who think of the expression as a tree, and 'outside to middle' by those who look at the printed form. Following this procedure, one applies first the rule $\sqrt{x^2} = |x|$ for any real x, and obtains $\sqrt{(1-\sqrt{2})^2} = \sqrt{2} - 1$. The other way is 'bottom-up' or



but in fact the quadratic can be factored as $(x + 1 - \sqrt{5})(x + 5 + \sqrt{5})$. Therefore the

4.5.1 Square root of a three-term sum

Equation (4.5) can be generalized to the pattern

$$\sqrt{a^2 \pm 2ab + b^2} = |a \pm b| \ , \tag{4.8}$$

where either a or b is a surd other than a square root (i.e., so that the corresponding squared term is still a surd)

We call this the square-root-nesting equation. S

is how the method fails. After all, the majority of nested surds do not denest, and so any implementation must know when to give up. Answering this question turns out to be as difficult as finding the successful part of the algorithm[6]. It might seem frustrating having to spend a lot of time deciding when the system will not succeed, but this decidedly less glamorous activity is essential to the smooth operation of a CAS. Consider, therefore, an example slightly altered from (4.1): $\sqrt{4+2\sqrt{2}}$. Substituting $X = 4$ and $Y = 2\sqrt{2}$ into (4.12) gives

$$\sqrt{4+2\sqrt{2}} = \sqrt{2+\sqrt{\ }}$$

have so far explored. Therefore, adding new cases must be done explicitly, and the code has little hope of working for examples slightly different from those it was designed for. For example, if this code were implemented and a user challenged the system with an example the developer had not considered, like

$$\sqrt{5+2\sqrt{6}+5\sqrt{7}+2\sqrt[4]{700}+2\sqrt[4]{1575}} = \sqrt{2}+\sqrt{3}+\sqrt[4]{175}\ ,$$

then there is no hope that the system could surprise the programmer by rising to the occasion, because the square root contains 5 terms, and that case is not treated[7]. As users report problems that the system 'cannot do', the developer is faced with constantly revisiting the code. This will always happen to some extent, of course, but it is particularly inevitable with this style of programming.

The above code also contains coding that repeats itself. This suggests that a more flexible approach will be a recursive one, meaning one in which the routine will be structured to call itself. The difficulty with a recursive approach is that we must be very careful to have a way of stopping it[8]. We want to abandon the process whenever it looks as though we are no longer making progress. The easiest way to do this is to have a numerical measure of the degree of nesting of a radical, and stop the recursion whenever this measure increases.

4.7 A measure of the degree of nesting of a radical

We now describe the function that is used to control the recursive denesting of radicals. One measure of the nesting of a radical is given in [Landau92b]; the one given here is similar in spirit. Suppose we have a radical x. We wish to associate with it an integer $\mathcal{N}(x)$ that is its nesting level. Our rules are:

4.7.1 If x is a number

A number here means an integer, but more generally it includes rational numbers also. (More abstractly, a member of the base number field.) For a number x, set $\mathcal{N}(x) = 1$. Some measures of nesting start counting at 0, but the starting point is arbitrary. Any undefined symbols are also given $\mathcal{N} = 1$.

4.7.2 If x is an nth root of something

If x has the form $\sqrt[n]{y}$, with $n > 1$, then we assign $\mathcal{N}(\sqrt[n]{y}) = 1 + \mathcal{N}(y)$. This is the fundamental feature that we wish to capture with our measure, so if we think of the radical being built up from other radicals, then every time we take another root we increase the measure. Thus $\mathcal{N}(\sqrt{2}) = 1 + \mathcal{N}(2) = 2$. Notice that the size of n is not used. Therefore the simplification $\sqrt{4} = 2$ is visible to this measure ($\mathcal{N}$ decreases from 2 to 1), but the simplification $\sqrt[4]{4} = \sqrt{2}$ is not, because only the strength of the root is

[7] As a case in point, we were agreeably surprised ourselves when Derive succeeded on this one.

[8] A system experiencing uncontrolled recursion is said to suffer from *recussion* — a special form of concussion. Its name reminds us that the developer often starts cussin' all over again.

reduced. This is acceptable, since nesting is what we are trying to measure, but other applications might require a measure in which n is included somehow.

4.7.3 If x is a product

From the point of view of nesting, $\sqrt{2}\sqrt{3}$ is no more complicated than $\sqrt{6}$. More generally, given radicals $\sqrt[m]{x}$ and $\sqrt[n]{y}$, there are integers a, p such that $\sqrt[m]{x}\,\sqrt[n]{y} = \sqrt[p]{}$

SIMPLIFYING SQUARE R TS F SQUARE R TS BY

4.9 Testing

All CAS developers have test suites that they run their systems on. These suites must contain problems for which the system is expected to obtain the correct simplification, and problems for which the system should correctly find no simplification. Derive has one such suite specifically for denesting problems. It contains all the examples given in this chapter and many others. We challenge readers to use their dexterity and sinisterity to invent some interesting examples to add to our suite. As a starting point, a specific case for which we have not given an example is a square root in which the ordering of the terms is important to the denesting.

An ideal test suite will have at least one example to activate each path through the algorithms of the system. As this simple denesting code illustrates, there are always many places where quantities are tested and execution paths switched. Compiling a thorough test suite even for this small part of a complete system is lengthy and difficult, so it is no wonder that over many years of use, users find examples that activate previously unexercised paths in the code. The test suites of all the CAS contain many examples contributed, often inadvertently, by (we hope temporarily) disgruntled users.

From *Computer Algebra Systems: A Practical Guide*, M.J. Wester, Ed., Wiley 1999.

References

[Chrystal64] G. Chrystal (1964) *Algebra, volumes I and II.* Chelsea Publishing Company, New York, 7th edition.

[Dickson26] L. E. Dickson (1926) *Algebraic theories.* Dover, New York.

[Hall88] H. S. Hall and S. R. Knight (1888) *Higher Algebra.* MacMillan, London, 2nd edition.

[Landau92a] Susan Landau (1992) A Note on Zippel Denesting. *Journal of Symbolic Computation* **13**: 41–45.

[Landau92b] Susan Landau (1992) Simplification of Nested Radicals. *SIAM Journal on Computing* **21**: 85–110.

[Zippel85] R. Zippel (1985) Simplification of Expressions involving Radicals. *Journal of Symbolic Computation* **1**: 189–210.

From *Computer Algebra Systems: A Practical Guide*, M.J. Wester, Ed., Wiley 1999.