Source Code's Challenge to U.S. Law

Introducing a Translator from C to English and Back

By Jonathan M. Baccash

Advised by Andrew W. Appel

May 7, 2001

This thesis is my own work in accordance with
University regulations.

Acknowledgements

I would like to thank my adviser, Professor Appel, for all his advice, patience, and reassurances (i.e., “the best is the enemy of the good”) along the way. He was an excellent guide and had a deep understanding and appreciation for the subject matter, and he made the time to help me.

Thanks also to my Dad, who had some excellent suggestions, and helped make English this paper.

Thanks also to John Reppy and Dave MacQueen, who quickly answered my questions concerning the ckit package.

Thanks also to my friends and family, who gave me additional support. Special thanks to Ishan, who gave some good advice in the early stages of writing. And thanks to Laura and Anders for teaching me a few legal principles. And thanks to Emile for critiquing my abstract. But no thanks to Manan, who sent me an email when he had finished writing his thesis in mid-April saying, “I’m going to Florida, how’s the thesis?”

Contents

Source Code's Challenge to U.S. Law

Appendices

Abstract

As technological advancement blurs the distinction between expressive speech and functional devices, Americans may be compelled to relinquish their right to discuss certain computer algorithms. This thesis presents BabelBuster, a program that translates between C and English. In so doing, BabelBuster supports the idea that an injunction against publication of source code is futile without also enjoining the publication of any description of the algorithm in question. The thesis then explores this concept in the context of the pending DeCSS trial, in which the major motion picture companies seek an injunction against publication of the DeCSS source code under the Digital Millennium Copyright Act. This thesis also questions whether courts will give more credence to Americans' First Amendment right to openly discuss computer algorithms or to Congress's Constitutional power to secure copyrights.

Chapter 1

DeCSS and Napsterized Movies

On July 17, 2000, Leon Gold, attorney representing the major motion picture companies, stated in federal court that when they accepted the DVD standard to distribute digital versions of their movies, these companies thought their movies could not be copied and downloaded over the Internet for free, or "Napsterized," like mp3 music files.[1] But these members of the Motion Picture Association of America (MPAA) now realize that if they cannot win their fight to banish the recipe developed by hackers to extract the movies from DVDs, the day DVD movies are Napsterized could be fast approaching.

The copyrighted works on DVDs are protected on two fronts. First, the DVD standard employs an encryption scheme, the Content Scrambling System (CSS), to control access to movies and thwart efforts to make illegal copies of the copyrighted works that are the contents of a DVD. Agreed upon in October 1996, this standard uses a rather weak 40-bit encryption, but one that was the strongest allowed by U.S. government export regulations at the time.[2] Second, the Digital Millennium Copyright Act (DMCA) bolsters protection of DVD movies. The act, which the 105th U.S. Congress signed into law on October 28, 1998, aimed to update copyright law for the digital age in preparation

---

[1] *Universal City Studios, Inc. v. Reimerdes*, 111 F. Supp. 2d 346 (S.D.N.Y.), transcript of testimony, 17 July 2000, at p. 6.

[2] Frank Stevenson, "Cryptanalysis of the Content Scrambling System," <http://www.cs.cmu.edu/~dst/DeCSS/FrankStevenson/analysis.ps>, 13 November 1999, p. 3.

for the ratification of the World Intellectual Property Organization (WIPO) treaties. It was the most comprehensive reform of U.S. copyright law in a generation.[3] The DMCA provisions that strengthened copyright protection systems are found in rule 1201(a):

> (2) No person shall manufacture, import, offer to the public, provide, or otherwise traffic in any technology, product, service, device, component, or part thereof, that--
> (A) is primarily designed or produced for the purpose of circumventing a technological measure that effectively controls access to a work protected under this title [i.e., a copyrighted work];
> (B) has only limited commercially significant purpose or use other than to circumvent a technological measure that effectively controls access to a work protected under this title; or
> (C) is marketed by that person or another acting in concert with that person with that person's knowledge for use in circumventing a technological measure that effectively controls access to a work protected under this title.[4]

These provisions clearly restrict the creation and posting on the Internet of technologies to circumvent CSS, which, according to Judge Lewis Kaplan of the U.S. District Court for the Southern District of New York, "effectively" controls access to copyrighted works.[5]

Although these measures all seem to assure the movie companies a level of comfort in their hope that DVD movies will not be Napsterized, subsequent events have shown that such comfort was perhaps illusory. As can be expected, hackers soon reverse engineered the weak CSS encryption algorithm. In October 1999, 15-year old computer geek Jon Lech Johansen posted his version of DeCSS, which can decrypt a DVD, on the Internet.[6] Then, when the *Hacker Quarterly* website, 2600.com, posted DeCSS on its

---

[3] "Digital Millennium Copyright Act," <http://www.educause.edu/issues/dmca.html>, 20 April 2001.
[4] Digital Millennium Copyright Act (Public Law 105-304) of 28 October 1998, section 1201(a).
[5] *Universal City Studios, Inc. v. Reimerdes*, 111 F. Supp. 2d 294 (S.D.N.Y.), decision, 17 August 2000, at p. 317.
[6] Doug Mellgren, "Speaking in Code: Quiet, Polite Teen at Center of DVD-Copying Firestorm," ABCNews.com <http://abcnews.go.com/>, 26 February 2001, archived at <http://more.abcnews.go.com/sections/scitech/DailyNews/johansen010226.html>.

website, the MPAA filed a lawsuit seeking to enjoin the dissemination of the code and seeking damages from the website. The MPAA claimed that by using DeCSS, users can illegally copy movies, and with increasing network speeds, users who want to share these illegal copies would soon be able to do so.

Although this case (*Universal City Studios, Inc. v. Reimerdes*) is currently pending before the 2nd Circuit Court of Appeals, the proceedings and outcome in the district court may shed light on what is to come. In the district court, Judge Lewis Kaplan held for the plaintiffs by enjoining the publication of DeCSS, finding that it violated the DMCA. Central to the case was whether or not the publication of DeCSS was a right afforded 2600.com by the First Amendment to the Constitution. The First Amendment states:

> Congress shall make no law … abridging the freedom of speech, or of the press; or the right of the people peaceably to assemble, and to petition the Government for a redress of grievances.[7]

Modern Constitutional interpretation extends First Amendment protection to such forms of expression as poetry, dancing, music, and, to some extent, computer source code. However, as Judge Kaplan stated, "Its [DeCSS's] expressive element no more immunizes its functional aspects from regulation than the expressive motives of an assassin immunize the assassin's action."[8] DeCSS's expressive element was protected by the First Amendment, but its function was to circumvent the technological access control of CSS, in violation of the DMCA. And indeed, the DMCA has all the backing of the Constitution as well. The Constitution states:

---

[7] U.S. Constitution, Amendment I.
[8] *Universal City Studios, Inc. v. Reimerdes*, 111 F. Supp. 2d 294 (S.D.N.Y.), decision, 17 August 2000, at p. 304.

> The Congress shall have Power … To promote the Progress of Science and useful Arts, by securing for limited Times to Authors and Inventors the exclusive Right to their respective Writings and Discoveries….[9]

Since DeCSS's function was to circumvent the technological access control in violation of the DMCA, Judge Kaplan ruled in favor of the plaintiffs.

The discussion of First Amendment analysis of computer source code in *Universal v. Reimerdes* raises questions including the extent of computer source code's expressive nature and the limits of free speech. Although the court enjoined the dissemination of the source code version of DeCSS, as requested by plaintiffs, it also indicated that it might be more hesitant to enjoin publication of an English description of the same algorithm. As demonstrated by this excerpt from his final decision, Judge Kaplan did not seem unhappy that this issue had not been presented:

> During the trial, Professor Touretzky of Carnegie Mellon University convincingly demonstrated that computer source and object code convey the same ideas as various other modes of expression, including spoken language descriptions of the algorithm embodied in the code. He drew from this the conclusion that the preliminary injunction irrationally distinguished between the code, which was enjoined, and other modes of expression that convey the same idea, which were not, although of course he had no reason to be aware that the injunction drew that line only because that was the limit of the relief plaintiffs sought. With commendable candor, he readily admitted that the implication of his view that the spoken language and computer code versions were substantially similar was not necessarily that the preliminary injunction was too broad; rather, the logic of his position was that it was either too broad or too narrow. Once again, the question of a substantially broader injunction need not be addressed here, as plaintiffs have not sought broader relief.[10]

Judge Kaplan may not have needed to decide this issue, but a court some day most assuredly will.

---

[9] U.S. Constitution, Article I, Section 8.

Courts distinguish between two types of speech restrictions, content-based and content-neutral. A content-based restriction, the more scrutinized of the two, restricts speech based on the meaning of the messages conveyed. A content-neutral restriction, in contrast, is not motivated by a desire to limit the message. According to Judge Kaplan, a content-neutral restriction will be upheld if it serves a substantial government interest and restricts First Amendment freedoms no more than necessary.[11] Judge Kaplan ruled that the DMCA restrictions are content-neutral and, since they serve the interest of protecting copyrights, the DMCA serves as a valid exercise of Congress's authority.

But opponents of the DMCA claim that it does restrict First Amendment freedoms more than necessary. In their article "Technological Access Control Interferes with Noninfringing Scholarship," Princeton University Computer Science Professors Andrew Appel and Edward Felten point out that technological access controls restrict the use and testing of algorithms to search for a clip of video in a video file or a bar of music in an mp3.[12] According to Appel and Felten, even if such technologies were built into a video-viewing program, without a general-purpose method for examining the video file unencrypted, efforts to develop new and innovative searching techniques would be impossible.

Others also argue that the DMCA unnecessarily restricts fair use. According to Law Professor Julie Cohen,

> Copyright law has long acknowledged that restrictions on reuse of another's copyrighted expression are restrictions on speech. It has also

---

[10] *Universal City Studios, Inc. v. Reimerdes*, 111 F. Supp. 2d 294 (S.D.N.Y.), decision, 17 August 2000, at p. 345.
[11] Ibid., pp. 327-8.
[12] Andrew W. Appel and Edward W. Felten, "Technological Access Control Interferes with Noninfringing Scholarship," Communications of the ACM, 43 (9), September 2000, pp. 21-23, archived at <http://info.acm.org/pubs/articles/journals/cacm/2000-43-9/p21-appel/p21-appel.pdf>.

> acknowledged that some such restrictions frustrate rather than promote creative progress. For these reasons, copyright law forbids authors from controlling the uncopyrightable ideas or functional principles embodied in their work and allows others to make "fair use" even of copyrightable expression for purposes such as criticism, comment, education, and research. The Supreme Court has indicated that these limitations on copyright are required by both the Patent and Copyright Clause and the First Amendment. Trade secret law, meanwhile, does not prohibit the reverse engineering of publicly distributed products to discover embodied secrets, and the Court has said that federal intellectual property law requires this result.
>
> The DMCA and UCITA [Uniform Computer and Information Transactions Act], however, contain no such limitations on the prior restraint of speech. On the contrary, both statutes seem designed for the express purpose of allowing private parties to suppress legitimate public debate about their products.[13]

Thus, although the DMCA itself does not restrict fair use, Congress is in effect giving the movie industries the option to do so.

Furthermore, one could claim that restricting the posting of DeCSS also infringes on the study of cryptography in general. This point is highlighted by Dr. David Touretzky's testimony that (as explained above) in order to effectively stop the decryption of DVDs, it is necessary to completely eradicate all discussion of the algorithm.

In his testimony before the court in *Universal v. Reimerdes*, Touretzky first stated that an order to stop publication of DeCSS is either too broad to fit existing law or too narrow to make sense. He made these claims based on the fact that the DeCSS algorithm as described in computer source code is largely equivalent to the same algorithm as described in various other forms, and it is not possible to discriminate between them;[14] therefore, the injunction, which only banned the source code form, made little sense. He

---

[13] Julie Cohen, "Unfair Use," The New Republic Online <http://www.tnr.com/>, 23 May 2000, archived at <http://www.tnr.com/online/cohen052300.html>.

states in his web page, "If code that can be directly compiled and executed may be suppressed under the DMCA, as Judge Kaplan asserts in his preliminary ruling, but a textual description of the same algorithm may not be suppressed, then where exactly should the line be drawn?"[15]

Touretzky described this idea in more depth in his court testimony. Referring to his "Gallery of CSS Descramblers",[16] which currently has more than 30 examples of the DeCSS algorithm expressed in other forms, he first showed that the screen dump of the C program, which was not enjoined by the court, could be easily converted to the C program. A screen dump is simply a binary image file that shows the contents of the program as displayed on the computer screen by a text editor. Touretzky showed that either a person or a program could trivially convert this program back to the original. Next, he showed that a program in a made-up language was also trivially converted back to the original. In fact, a compiler could be constructed to do this. Touretzky even claimed that a compiler could be constructed to convert the ideas as expressed in any sufficiently expressive language, even English, back into the original C source code. Touretzky's conclusion is that:

> If you're going to be effective about it, if you're really going to stop people from doing it [decrypting DVDs], you've got to stop them from hearing the message that the defendants were trying to convey. You've got to stop them from any description of the algorithm, which could be used to construct one of these devices.[17]

---

[14] *Universal City Studios, Inc. v. Reimerdes*, 111 F.Supp. 2d 346 (S.D.N.Y.), transcript of testimony, 25 July 2000, at p. 1066.
[15] David Touretzky, "Gallery of CSS Descramblers," <http://www.cs.cmu.edu/~dst/DeCSS/Gallery/index.html>, 14 April 2001.
[16] Ibid.
[17] *Universal City Studios, Inc. v. Reimerdes*, 111 F.Supp. 2d 346 (S.D.N.Y.), transcript of testimony, 25 July 2000, at p. 1071.

In other words, to stop decryption of DVDs, we must restrict our freedom to express ourselves, as afforded by modern interpretation of the First Amendment. Such a restriction is a much higher price to pay for copyright enforcement than one might think from a reading of the DMCA; however, it may prove to be a necessary evil.

Chapter 2

Blurring the Lines between Expression and Function

Opponents to the view that source code is speech say that there really is a rational difference between source code and speech in the English language. According to them, an English explanation of an algorithm informs the intellect, while a source code (say, C) description of the algorithm can be utilized on the computer to actually do something. In his attempt to debunk the "myths" of source code non-functionality, Anthony Coppolino spoke on behalf of the government in *Bernstein v. U.S. Department of Justice*:

> The plaintiff's argument is that software is also speech. It is information. He says there is no rational distinction between describing the software in English and describing it in source code, which is computer language. But we think there is a very rational distinction. The English explanation of Snuffle [encryption algorithm] would inform the intellect. The source code can be utilized on the computer to actually encrypt.[18]

Assuming this to be the case, Touretzky's argument that a C description of an algorithm and an English description of an algorithm are largely equivalent would fail.

But opponents of the DMCA, who claim that DeCSS is at least as expressive as an English description of the algorithm, make the point that DeCSS does not actually do anything. It is simply a set of instructions, a "recipe" for performing the algorithm, not a machine. To actually decrypt a DVD, one must carry out the algorithm as described by the program (or have their computer do it for them). The algorithm is quite easily

---

[18] *Bernstein v. U.S. Department of State*, 974 F. Supp. 1288 (N.D. California), transcript of testimony, 20 October 1995, at p. 18.

described in English, although many would claim that the same imperative commands expressed in the source code program are less easily described and understood in an English description.[19]

In this vein, attorney Lee Tien has argued that publishing source code is a speech act because computer scientists and programmers communicate with source code, just as economists and mathematicians communicate with equations;[20] they do so because that is the simplest way to inform other programmers of their discoveries. In the case of source code, the computer scientists are the intellect; the source code *is* used to inform the intellect.

Of Coppolino's "rational" distinctions then, the only one that remains is the fact that a C program can be utilized on a computer to actually encrypt, while an English description cannot. In this section we prove Coppolino's argument false, showing that even an English "program" can be understood by a computer in much the same way as a C program. In so doing, the similarities between source code and English prose are made clearer; courts that are willing to enjoin the publication of source code must be willing to enjoin the publication of the English prose version of that source code.

2.1 c2txt2c

As far as I can tell, c2txt2c was the first attempt to create a program that translates from C to English and back. Its author, Leevi Marttila, was outraged that Judge James Gwin of the Federal District Court of the Northern District of Ohio held that software

[19] Eben Moglen, "Anarchism Triumphant: Free Software and the Death of Copyright," First Monday <www.firstmonday.org>, August 1999, archived at <http://firstmonday.org/issues/issue4_8/moglen/>.
[20] Lee Tien, "Publishing Software as a Speech Act," 15 *Berkeley Technology Law Journal* 629 (Spring 2000), at p. 633.

(namely, blowfish.c, an implementation of the blowfish encryption algorithm) is not protected by the First Amendment because it is a "functional device" like a telephone circuit.[21] Seeking to prove her wrong, he hacked up a few Perl scripts that he called c2txt2c, which he hoped would show the equivalence of C and English.

c2txt2c translates a C program into English text that does not describe how to perform the algorithm. For example, hexadecimal numbers in the C program correspond to poetry in the English version. Thus, hex 243f6a88 might be mapped to:

> ... oops, I don't find sugar coated fruits here. Somebody has stolen them.
> Instead I see some bad poem: "Very humongous red dog fortunately
> dances right of monitor once in hour. "

Although this is not exactly what I would think of as a translation from C to English, the result is more or less English, and it can be machine-translated back to C.

This early attempt at a C to English to C translator met with a fair amount of success; through the lively discussion on a message board following an editorial on the Slashdot website, we can see that many people were enthusiastic about the program:

> > it's not English, it's data that has been encoded into English.
> ….
> A language *is* a form of encoding by nature. The majority of people (last time i checked) can't read minds at will, so societies had to come up with a way to put your thoughts into a form that can be easily transmitted. Language can also help us pore over our own thoughts (thinking out loud).
> - Ex-Cyber
>
> I would say that the output of the C-to-text converter is MORE "purely functional" than the actual C code. This is because the code is basically full of imperatives. The data declarations are declaratives (what a coincidence!), but the meat of the stuff still consists of instructions. Add the fact that it's REALLY HARD to understand exactly what the code is doing after it's been converted – it would be difficult to glean any

[21] Leevi Marttila, "Accurate Language to Inaccurate Language (and back) Translator c2txt2c v0.2.1," <http://personal.sip.fi/~lm/c2txt2c/>, 15 January 2000, citing *Junger v. Daley*, 8 F. Supp. 2d 708 (N.D. Ohio), decision, 2 July 1998, at p. 712.

> computer science ideas from that stuff. So this "English" is basically meaningless to anything but a computer – much like object code.
>
> Maybe if people learned how to interpret the English text really well, they could derive as many ideas from it as from the C code. But it seems like that would be much harder, since the text files are "blown up" so much by the converter.
> - Chris Peikert[22]

Still, others questioned whether c2txt2c's output was even English at all:

> This is quite funny, but I don't think you'd ever be able to convince someone that the output is just plain english.
>
> I could write a program that would convert an arbitrary binary file into some intelligible but meaningless text file, but I don't think I could claim that sensitive information encoded that way is protected by the 1st amendment – it's not English, it's data that has been encoded into English.
>
> Why not try to write a description of the algorithm in English?….
> - Austin Appleby[23]

User "j.e.hahn" replied to such accusations:

> > *While it may use english words, it's not really english unless the sentence structure is at least somewhat passable as english.*
> So then what are the writings of Ginsberg, as well as many other beat poets and others, James Joyce, William Faulkner, and many others who play with words to the point that they are outside the realm of standard sentence structure and grammar? *Ulysses* has sections that are virtually unreadable due to their construction, and Ginsberg & co. played with the "cut-up method". Aren't these still english?
> - j.e.hahn[24]

Some of the ideas that arose from these Slashdot postings were incorporated into later C to English to C translators.

---

[22] "C to English and English to C Translator," from Slashdot message board, <http://slashdot.org/articles/980815/105250.shtml>, 15-20 August 1998.
[23] Ibid.

2.2 c2eng and eng2c

In October 2000, Omri Schwartz released c2eng, which translated from C into an English description of the program, and eng2c, which translated back. The main difference between these and c2txt2c is that the English generated actually describes to a person (or computer) what to do to perform the algorithm, just as the C does. Thus, the C preprocessor statement "#define Pi 3.14159265358979" is translated as "Note: we define Pi to mean '3.14159265358979'."[25] This is exactly what the preprocessor statement tells the C compiler.

Although Schwartz claims that the English output is "grammatically correct", I found that this is not true in some cases. In particular, he ran into difficulty translating highly nested C constructs and in translating preprocessor statements that appeared in the middle of an expression.

For instance, consider the C expression "key[4] = k[4] ^ csstab1[key[4]] ^ key[3]". Here, we might attempt to interpret this in English as "Assign element of array key at index 4 the result of array k's element with index 4 bitwise exclusive-orred with array csstab1's element with index array key's element with index 4 bitwise exclusive-orred with array key's element with index 3." This interpretation has two main problems. First of all, it is very difficult for a person to understand. A person has difficulty understanding long, complex clauses, and when clauses are nested within others, the sentence becomes unwieldy. We want to break the clause up into smaller pieces. Second, the English description is ambiguous. We can interpret the C description (again, in the more understandable C language) as "key[4] = k[4] ^ csstab1[key[4] ^ key[3]]" or

---

[24] Ibid.

"key[4] = k[4 ^ csstab1[key[4] ^ key[3]]]", in addition to the original C. When expressions (clauses) are nested in this manner, ambiguities frequently arise.

Schwartz used two tactics to solve this problem. First, he used parentheses to group clauses, just as expressions are grouped in C. Thus, his program would interpret our example statement as 'Assign to array key's element at address (4) the value "array k's element at address (4) bitwise xorred by array csstab1's element at address (array key's element at address (4)) bitwise xorred by array key's element at address (3)".'[26] This use of parentheses is clearly not standard English. In English, parentheses usually connote that a less important but perhaps interesting piece of information is being conveyed; if we remove the parentheses and all the text they enclose, we end up with grammatically correct English that still describes the proceedings of the program. Furthermore, the translation's use of quotations to connote an expression that produces a value is non-standard. Second, he used phrases like "the 2-level parenthetical expression" to signal that a parenthetical expression has begun. He compares this to a math teacher saying "the quantity x plus y, all over z…."[27] This is reasonable, but again, this does not solve the problem of a person comprehending what is being described. When the parenthetical expressions have nested too deep, we must split up the clause.

The second major problem Schwartz encountered was with preprocessor macros in the middle of an expression. For instance, we could have the following C statement:

```
        x = y + z
#ifdef ADD_W
                + w
#endif
```

[25] Omri Schwartz, "Converting C to English with Perl," The Perl Journal, <www.tpj.com>, Fall 2000, p. 70.
[26] Ibid.
[27] Ibid.

```
                    ;
```

Here, the difficulty is handling the plethora of ways preprocessor macros can be used to modify the meaning of the code. We might wish to translate this code snippet as "set x equal to y plus z (plus w, if ADD_W is defined)". However, Schwartz did not find a satisfactory method to handle all the possible places a preprocessor macro can occur. For example, a preprocessor macro could be used to simply modify which operator is used in a binary operator expression. It might be possible to use duplication to extract the preprocessor declarations outside the expression, as follows:

```
#ifdef ADD_W
        x = y + z + w
#else
        x = y + z;
#endif
```

But Schwartz did not explore such an alternative.

2.3 BabelBuster

I created BabelBuster with the vision that C source code can be machine translated into grammatically correct English that is easy to read, and that this English can be machine translated back into C.

To really translate the C into grammatically correct English, we first solve the problem of compounded nesting of expressions. Suppose we have the expression "1 + 2 * 3". Due to mathematical convention, we first multiply and then add. But the corresponding English, "1 plus 2 times 3", has no such convention. We are left to wonder whether we should add first and then multiply or multiply first and then add. See Figure 1 for a tree diagram of the alternative meanings of this English clause.

The idea in BabelBuster is to replace nested clauses with temporary variables and list the contents of the temporary variables in later clauses. For instance, the above expression becomes “1 plus x, where x is 2 times 3”. Note that in so doing, we have eliminated the ambiguity present in the simpler translation. A more complicated expression, “key[4] = k[4] ^ csstab1[key[4]] ^ key[3]”, becomes “set the element of key at index 4 equal to the bitwise exclusive or of x and y, where x is the element of k at index 4, y is the bitwise exclusive or of z and the element of key at index 3, z is the element of csstab1 at index w, and w is the element of key at index 4”.

Figure 1.

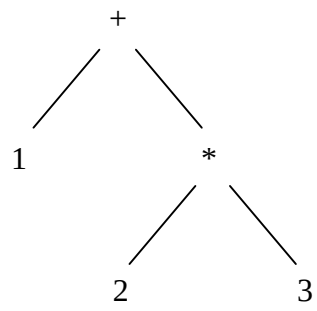


Figure 1a. Tree diagram for 1+2*3.

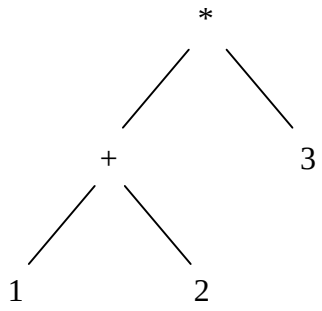


Figure 1b. Tree diagram for (1+2)*3.

Since BabelBuster translates preprocessed C (code in which the preprocessor statements are removed in preparation for compilation), it does not need to address the problem of preprocessor definitions arising in tricky spots. Thus, the two main problems in c2eng are overcome in BabelBuster.

A few other interesting problems arose in translating C to English. For instance, non-functional information such as comments and unary pluses were removed from the program in translation to English. Also, transitions were added to the English description of the algorithm in an attempt to improve readability and avoid the tedious nature of repeated imperative statements. To denote the end of an operation, we say “continuing

on”; in addition, it was found that three other transitions (“then”, “next”, and “after that”) were sufficient. For instance, the following series of statements can be greatly improved: “To perform operation 3, continue to do nothing as long as done is equal to zero. Increment x by 7. Decrement z by 6. Apply x and z to f. Return the result of calling clean_up.” With transitions, it is much better: “To perform operation 3, continue to do nothing as long as done is equal to zero. Continuing on, we next increment x by 7. Then decrement z by 6. Next, apply x and z to f. After that, return the result of calling clean_up.” Here, the importance of transitions is clearly understood.

Putting it all this together, we might translate the following program as:

```
struct A {
     int x;
     unsigned : 5;
     struct B {float f;} t;
} y;

int f(int a, struct A s) {
     if (a > 0)
          while (a++ < y.x) s -= a;
     else {
          y.x = s.x;
          s.x = a;
     }
}
```

Let B be a structure with member f, where f is a floating point number. Let A be a structure with members x, an unnamed 5-bit unsigned integer, and t, where x is an integer and t is a B structure. Let y be an A structure.

Let f be a function returning an integer. It is called with arguments "a" and s, where "a" is an integer and s is an A structure. To perform the function, perform operation 1 provided "a" is greater than 0; otherwise, perform block 2. To perform operation 1, continue to decrement e1 by "a", where e1 is member x of s as long as e1 is less than e2, where e1 is "a" before incrementing "a" by one and e2 is member x of y at the beginning of the iteration. To perform block 2, first set member x of y equal to member x of s. Then set member x of s equal to "a". This ends block 2.

For a technical analysis of BabelBuster, see Appendix A. For more example translations, see Appendix B.

2.4 Discussion

BabelBuster succeeds in correctly translating the entire C language. By "succeeds", I mean that the program compiled from the English will have the same effects as the program compiled from the original C source code. The English is quite readable, and several things could be done to make it even more readable and true to the original C code. From the insights gained in programming a translator from C to English, I conclude that it is certainly possible to machine translate C to English in a very comprehensible manner, and BabelBuster does this in most cases.

Because a machine can translate between C and English, it seems that there is little sense in distinguishing between them in a First Amendment analysis. Source code can be at least as expressive as the English, and the English can be at least as functional as source code. Source code has always been used by computer scientists to express an algorithm's workings. Now English can be used as a functional device. Furthermore, if one is easily converted to the other, nothing is accomplished by banning only one of them. In this sense, a court that bans C code must also be prepared to the English expression of the algorithm, and all other forms of expression whose purpose is to express the idea represented in the C code. Such a ban would test the limits of American speech rights.

Chapter 3

The Limits of American Speech Rights

Americans were fascinated by the expression of individual beliefs even before the signing of the Declaration of Independence. For instance, on December 16, 1773, a band of men disguised as Mohawk Indians boarded three British ships in Boston harbor and dumped their cargo of tea. The Boston Tea Party, as it is referred to now, is still revered by many patriotic Americans today as a political statement, although in fact it seems to push the limits of speech we would be willing to afford our own citizens. By acknowledging such reverence, we gain understanding into the respect possessed by many Americans for courageous (even zealous) expressions of speech.

Indeed, a central notion in creating a Constitution was that the people should have their say; the system of electing leaders depends on people expressing their opinions. Ithiel de Sola Pool writes:

> The cohesion and effective functioning of a democratic society depends upon some sort of public agora in which everyone participates and where all deal with a common agenda of problems, however much they may argue over the solutions.[28]

Clearly, being well informed and expressing one's own opinions is a fundamental element of American society.

Although Americans have guarded their right to speak freely, the freedom of expression is not necessarily the only value at stake in any given First Amendment case.

For example, courts have stopped the publication of texts to preserve human life, especially in the context of war.[29] The American press must be careful to prevent libelous accusations, or risk damage awards. And authors must give credit where credit is due; plagiarism breaks the law, not just school rules.[30]

Two modern cases shed light on just how far the U.S. Supreme Court is willing to go to protect an individual's right to speak freely. First, the case of the Pentagon Papers pitted the free speech claim of *The New York Times* against the government's concern for national security. On June 13, 1971, *The New York Times* printed the first articles and top-secret Defense Department documents that had been leaked to the *Times* in a series that would expose a negative evaluation of the government's handling of the Vietnam War.[31] When Attorney General John Mitchell asked that the Pentagon Papers be returned to the Department of Defense and no further documents be published, the *Times* refused to comply.

A lawsuit was soon filed, in which the government claimed that the papers contained stolen government secrets that, if published, would put many American soldiers at risk and embarrass the United States in the eyes of the world.[32] The Court's struggle to decide the case is exemplified in Justice Stewart's hypothetical question during closing arguments:

> Let us assume that when the members of the Court go back and open up this sealed record, we find something there that absolutely convinces us that its disclosure would result in the sentencing to death of a hundred

---

[28] Ithiel de Sola Pool, Technologies Without Boundaries: On Telecommunications in a Global Age, edited by Eli M. Noam, 15; quoted in Lawrence Lessig, Code, 180.
[29] Lawrence Lessig, Code, 169.
[30] David D. Kirkpatrick, "Court Halts Book Based on 'Gone With the Wind,'" The New York Times, 21 April 2001, late ed., p. A1.
[31] Peter Irons and Stephanie Guitton, ed., May it Please the Court, 167.
[32] Lessig, 169.

> young men whose only offense had been that they were nineteen years old and had low draft numbers. What should we do?[33]

Although the arguments for enjoining the papers' publication had substantial merit, the government in this case was seeking prior restraint, which many believe to be more dangerous to a system of free speech than a post-publication lawsuit.[34] To win such a case, the government must show "irreparable harm" would result from not restraining the speech.[35] The Supreme Court held that the government failed to make the necessary showing, and *The New York Times* was therefore allowed to publish without the threat of prior restraint.

The second free speech case with which U.S. courts struggled began when a left-wing magazine, *The Progressive*, planned to publish an article describing the workings of a hydrogen bomb. *The Progressive* first submitted a draft manuscript of the article to the Department of Energy for review, at which point the government brought an injunction to block its publication.[36] According to Howard Morland, the *Progressive* writer whose manuscript was under scrutiny, the government's official reason for preventing its publication was that it would materially shorten the development time of thermonuclear weapons in other nations, and that this posed a significant risk to national security.[37] One might even think that such a claim was in fact an understatement; such information poses a significant risk to all humanity.

But Morland claims that he wrote the article without official access to classified information; anyone could have come to the same conclusions based on publicly

---

[33] Irons and Guitton, 173.
[34] Lessig, 169.
[35] Ibid.
[36] Ibid., 170

available documents and a little knowledge of physical principles.[38] Indeed, after the district judge considered whether or not to allow publication of the manuscript for two and a half months, on September 16, 1979, the *Press-Connection* of Madison, Wisconsin published a letter written by Chuck Hansen, a computer programmer, detailing his own knowledge of the workings of an H-bomb. The cat already having been let out of the bag, the government dropped its case against *The Progressive* the next day.[39]

The cases of the Pentagon Papers and *The Progressive* presented the courts with difficult decisions, in part due to their content-based motivations for prior restraint. As noted earlier, a content-neutral restriction of speech is within Congress's power if it serves a substantial government interest and restricts First Amendment freedoms no more than necessary. But for a content-based restriction to pass muster in the courts, more scrutiny is necessary. The Supreme Court's opinion in *Turner v. FCC*, in which a cable company was required to carry certain stations, illustrates this point:

> The First Amendment, subject only to narrow and well-understood exceptions, does not countenance governmental control over the content of messages expressed by private individuals. Our precedents thus apply the most exacting scrutiny to regulations that suppress, disadvantage, or impose differential burdens upon speech because of its content. Laws that compel speakers to utter or distribute speech bearing a particular message are subject to the same rigorous scrutiny. In contrast, regulations that are unrelated to the content of speech are subject to an intermediate level of scrutiny, because in most cases they pose a less substantial risk of excising certain ideas or viewpoints from the public dialogue.[40]

In each of these cases, the whole point of the suppression of speech was to excise certain ideas from the public dialogue. Although in each case such excision may have had a

---

[37] Howard Morland, "The Holocaust Bomb: A Question of Time," <http://www.fas.org/sgp/eprint/morland.html>, 15 November 1999.
[38] Ibid.
[39] Lessig, 170.

worthy cause, it is clear that the Framers of the Constitution would not have approved of eradicating an idea from public dialogue without the most detailed scrutiny.

Note that in each of these cases, our freedom to express ourselves is given the utmost attention and respect. Yet, in each of these cases other matters are also considered, and interests of humanity are perhaps held even more sacred. In so doing, the Court has recognized the inability of a government to perform its most vital function of protecting citizens' welfare when it allows people to express themselves in any manner they choose.

Applying these principles, one must consider the freedom of expression in *Universal v. Reimerdes* with the utmost respect and ask whether publication in this case threatens the government's ability to effectively carry out other important interests.

---

[40] *Turner Broadcasting System, Inc. v. Federal Communications Commission*, 512 U.S. 622, decision, 27 June 1994, at pp. 641-2.

Chapter 4

Seeking a Conclusion for DeCSS

The cases that challenge the limits of our right to free speech and the blurring of the lines between speech and machine raise issues that must be considered if we are to come to a satisfying conclusion in the case of DeCSS.

First, one must recognize the fact that CSS decrypting software is already abundant on the Internet, and even available in print media.[41] Compare this to *The Progressive*'s case, where the government dropped its case to enjoin the publication of information on how to make an H-bomb when such information was published in another magazine. Applying this logic to DeCSS, it would seem likely that, as Judge Kaplan analogizes, "the barn is unlocked and this horse is out."[42] Yet the DeCSS case differs from that of *The Progressive* in that if the wrong person learns how to make an H-bomb, he or she can cause great destruction; one person is all it takes; with DeCSS, however, an economic benefit for the MPAA could result from any effort to slow or stop dissemination of DeCSS. Mikhail Reider, manager of worldwide Internet anti-piracy operations for the MPAA, testified that in her opinion a permanent injunction against DeCSS "would send a very strong message to both those that would engage in illegal

[41] Omri Schwartz, "Converting C to English with Perl," The Perl Journal, Fall 2000, p. 70.
[42] *Universal City Studios, Inc. v. Reimerdes*, 111 F.Supp. 2d 346 (S.D.N.Y.), transcript of testimony, 20 July 2000, at p. 670.

behavior and to law enforcement."[43] Thus, by barring the publication of source code may not only deter users from trafficking it and using it, but may also encourage better law enforcement.

Still, it is unclear that it is even possible to effectively enforce a source code ban. Today, members of the hacker community can traffic a program by encrypted email or anonymous email. Hackers can also easily convert a program into a multitude of various forms, many of which do not seem to correspond to the original source code (consider c2txt2c). But if government does not at least attempt to enforce the law, it is essentially succumbing to anarchy; through its attempts, it can gain insights into more effective enforcement strategies.

Another issue is that, assuming the DMCA provisions fall under the content-neutral category as Judge Kaplan determined, one must question whether or not there is a better way to protect the movies' copyrights, and whether the DMCA provisions restrict First Amendment rights more than is necessary. To answer this question, it is first necessary to understand just how restrictive the DMCA provisions are. As a demonstration of the DMCA restrictions, consider the case of the group of researchers who recently cracked digital watermarks in the SDMI Challenge that are used to enforce a technological access control. These researchers have made a valuable contribution to the field of steganography, the science of hiding information in plain sight; upon cracking all the watermarks in the Challenge, they concluded on a note of pessimism: "Ultimately, if it is possible for a consumer to hear or see protected content, then it will be technically

[43] Ibid, p. 669.

possible for the consumer to copy that content."[44] However, the researchers are being threatened with legal action if they publish their paper, on DMCA grounds, and as of 27 April, the researchers had still not published the paper.[45] Here, the DMCA provisions are so restrictive to speech that it is unclear whether they deter progress in the arts and sciences more than promote progress as they were designated to do.

In the fast-paced Internet world, it seems reasonable that attempts to identify a less restrictive means to enforce movies' copyrights may someday succeed.

As a first example of such an attempt, one might think that if we simply restrict the keys used in the algorithm, we can protect the copyright and prevent unnecessary banning of the algorithm. But even in this case, there is still the possibility of publishing an algorithm to determine the keys (for, after all, hackers did follow a procedure to determine them). Thus, it would be pointless to enjoin the keys from publication without also banning publication of the algorithm to determine them.

Second, the DMCA could be revised so that a user's right to fair use is better protected, as noted above. Yet, as also noted above, it would be difficult to completely protect fair use without actually allowing users to circumvent the technological access controls that the DMCA fortifies.

Third, Dan Burk criticizes the law's current categorizations of computer source code. He writes that the law associated with copyright and patent is difficult to apply to source code; accordingly, we should invent a new "method to securing for limited Times to Authors and Inventors the exclusive Right" to their algorithms, as afforded by the

---

[44] Scott A. Craver et. al, "Reading between the Lines: Lessons from the SDMI Challenge," <http://cryptome.org/sdmi-attack.htm>, 27 April 2001.
[45] John Markoff, "Record Panel Threatens Researcher with Lawsuit," The New York Times, <www.nytimes.com>, 24 April 2001.

Constitution.[46] With such a legal categorization for source code, movie industries might attempt to secure the exclusive right to CSS and its decrypting algorithm. Still, details of such a categorization have not been invented, and the possibility of doing so is not certain. Also, Burk conceded that it might be too late to apply such a formulization of law.[47]

In fact, since banning the description of an algorithm in any form is necessary to rationalize the DMCA provisions, one might even question whether the provisions are content-neutral at all. This is a difficult and complex issue. The Supreme Court's opinion in *Turner v. FCC* states:

> Deciding whether a particular regulation is content-based or content-neutral is not always a simple task. We have said that the "principal inquiry in determining content-neutrality…is whether the government has adopted a regulation of speech because of [agreement or] disagreement with the message it conveys." The purpose, or justification, of a regulation will often be evident on its face. But while a content-based purpose may be sufficient in certain circumstances to show that a regulation is content-based, it is not necessary to such a showing in all cases.[48]

It appears that under this "principal inquiry", which examines only the intentions of the regulation, the DMCA is a content-neutral restriction. Indeed, this is the line of reasoning Judge Kaplan followed in his opinion in the DeCSS case. However, the Court notes that this is not the only inquiry.

In this regard, the decision in *Turner v. FCC* refers to *Minneapolis Star & Tribune Co. v. Minnesota Comm'r of Revenue*, where the Supreme Court stated that it has long recognized that "even regulations aimed at proper governmental concerns can restrict

---

[46] Dan L. Burk, "Copyrightable Functions and Patentable Speech," Communications of the ACM, 44 (2), February 2001, p. 75, archived at <http://www.acm.org/pubs/articles/journals/cacm/2001-44-2/p69-burk/p69-burk.pdf>.
[47] Ibid.

unduly the exercise of rights protected by the First Amendment."[49] In support for this statement, the Court cites *Schneider v. State*. In that case, the Court stated:

> This court has characterized the freedom of speech and that of the press as fundamental personal rights and liberties. The phrase is not an empty one and was not lightly used. It reflects the belief of the framers of the Constitution that exercise of the rights lies at the foundation of free government by free men. It stresses, as do many opinions of this court, the importance of preventing the restriction of enjoyment of these liberties. In every case, therefore, where legislative abridgment of the rights is asserted, the courts should be astute to examine the effect of the challenged legislation. Mere legislative preferences or beliefs respecting matters of public convenience may well support regulation directed at other personal activities, but be insufficient to justify such as diminishes the exercise of rights so vital to the maintenance of democratic institutions.[50]

Here, the Court makes clear that another essential examination in determining content-neutrality is the effect of the legislation.

And as noted above, the effects of the DMCA are to ban publication of all descriptions of the DeCSS algorithm, and this contrasts with a typical content-neutral restriction. (For instance, one could choose to express his disapproval of a law by shooting his Congressman. This form of expression is suppressed in a content-neutral manner. However, one could still express that disapproval in another manner, such as speaking out in public or holding a rally.) In the DeCSS case, no alternative mode of expression is afforded; although the motivation for the DMCA is content-neutral, its effect is to ban all expression containing certain content. Such a provision, based on the analysis above, is content-based.

---

[48] *Turner Broadcasting System, Inc. v. Federal Communications Commission*, 512 U.S. 622, decision, 27 June 1994, at p. 642.
[49] *Minneapolis Star & Tribune Co. v. Minnesota Comm'r of Revenue*, 460 U.S. 575, decision, 29 March 1983, at p. 592.
[50] *Schneider v. State*, 308 U.S. 147, decision, 22 November 1939, at pp. 150-1.

Note also that in this case, the DMCA allows movie industries to ban algorithms based on content; if the movie maker wishes to ban a particular decryption algorithm, say, Decrypter, it need only encrypt its DVDs so that Decrypter must be used to gain access to the DVD's content. Such a power would be rejected as content-based if Congress assumed it; it seems strange that movie industries are given this privilege and Congress is not.

Another critical argument pushed by the defense in *Universal v. Reimerdes* is the fact that the DeCSS code was created in an effort to provide DVD support in the open source Linux operating system. How, asked the defense, can an open source implementation of the algorithm be provided without posting DeCSS? But since the code in question actually compiles in the Windows closed source environment rather than the Linux open source environment, the court rejected this argument. Again, since the source code is easily converted into a program that runs on a Linux machine, the difference between a Windows source code program and a Linux source code program is trivial. Furthermore, by the argument that closed source (i.e., machine code) is trivially converted to open source (i.e., source code), we see that it makes no difference whether or not the source code is provided; what matters is whether the disseminating party has a license to provide the code.

In the end, the DeCSS case raises a difficult question. If the court grants defendants the free speech protection they request, the court may be giving up on copyright protection entirely. If it upholds the DMCA, it may be "overruling" one of our most fundamental values – free speech. Judge Kaplan may have averted needing to make such a decision, but the judicial system might not be let off so easy some day. When the

issue is finally decided, none can be certain of the outcome. In this age of rapid technological progress, who can say whether the provisions of DMCA too will fall victim to changing times.

Appendix A

Technical Analysis of BabelBuster

A.1 BabelBuster

BabelBuster is an ML program written for SML/NJ that translates preprocessed C source code into English; it can then translate the English back to C. It aims to translate in such a manner that the English produced describes the workings of the program, just as the C code, and that the English is both grammatically correct and readable.

ckit version 1.0 was modified slightly and used to parse C into an abstract syntax tree and pretty print the C corresponding to an abstract syntax tree. It was written by several programmers at Bell Labs in 1997-98.[51] The only modifications to the original libraries were made to fix a bug parsing large integers.

When using ckit, we have the choice of using the abstract syntax tree or the direct parse tree of the C grammar. Each has advantages and disadvantages. The main disadvantage to using the abstract syntax tree over the parse tree was that some information was not present in the abstract syntax tree that was in the parse tree. For instance, declarations that declared more than one variable ("int x, y;") were separated into multiple declarations of one variable each to simplify the abstract syntax tree ("int x; int y;"). But note that the two are functionally equivalent, and the abstract syntax tree is simpler. Other simplifications were also incorporated into the abstract syntax tree: nested structures and unions were moved to the top level; declarations that defined both a

structure and one or more variables were separated into separate structure definition and variable declarations; and anonymous structures were given names. Furthermore, programs were type-checked in translation from parse tree to abstract syntax tree. Because of the simplicity of using the abstract syntax tree, it was chosen over the parse tree.

To parse the English prose, it was necessary to ensure that the printed English was LALR(1). A C declaration like "type_name x;" was translated into English of the form "let x be a type_name". Translating types in an LALR(1) grammar was rather straightforward. For instance, "int" translated to "integer", "int (**)(int)" translated to "a pointer to a pointer to a function with argument an integer and returning an integer", and "int [5]" translated to "an array of length 5 each element of which is an integer". Although very complicated types could become difficult to comprehend in the English prose ("int (*)(int (*)(int))" translated to "a pointer to a function with argument a pointer to a function with argument an integer and returning an integer and returning an integer"), such complicated types occur quite rarely in practice. (Nonetheless, a possible remedy to this problem is given below.)

Structure definitions and union definitions were also translated with ease once they were simplified (as noted above) by ckit when building the abstract syntax tree. The strategy used was to first list the members present in the structure or union, including unnamed bit-fields, and then list the types of each named member. For example, "struct S {float x,y; int : 5; int i;};" is translated as "Let S be a structure with members x, y, an unnamed 5-bit integer, and i, where x is a floating point number, y is a floating point number, and i is an integer."

---

[51] "ckit," <http://cm.bell-labs.com/cm/cs/what/smlnj/doc/ckit/index.html>, 25 April 2001.

Top-level declarations and definitions were grouped together in paragraphs separate from function definitions, and each function definition was described in a separate paragraph. Paragraphs were delineated by two newlines. A function's workings were described after first giving the function's name and types of arguments and return values, as in the following example: "Let Blowfish_Test be a function returning an integer. It is called with argument ctx, where ctx is a pointer to a BLOWFISH_CTX. To perform the function, …." Return types of "void" were translated as "no result".

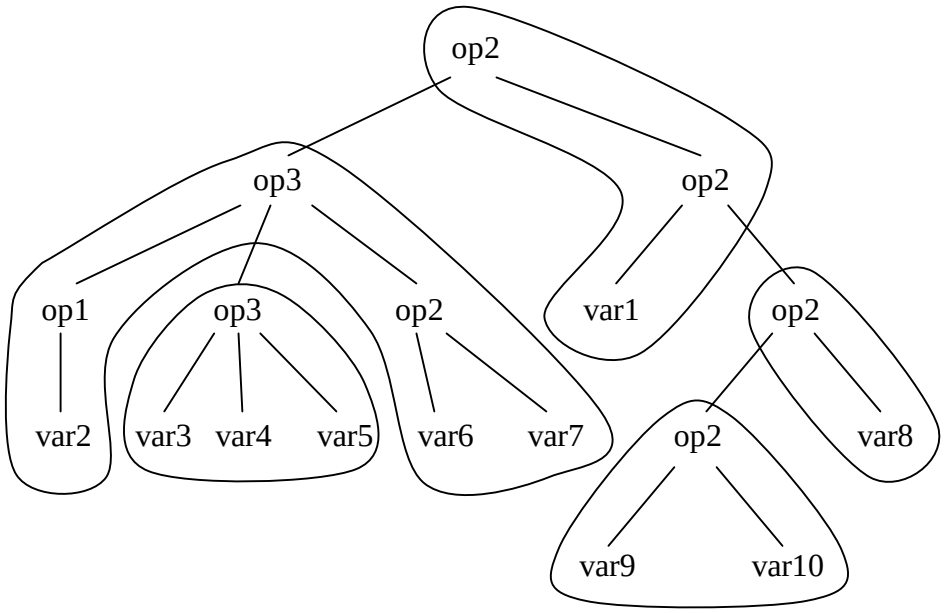
Once a scheme for translating C expressions to English was developed, it was clear the same scheme could be used to translate types, structures, and statements. Each of these C constructs pose difficulty for a translator due to the fact that they can be nested in each other. In a naïve English translation, expressions that are highly nested become ambiguous or difficult to read. For instance, "a[3] + 2 * 3" becomes the impossibly ambiguous "the element of a at index 3 plus 2 times 3" when trying to translate in the most straightforward way. The strategy used for translating C expressions was to break complicated expressions into smaller "tiles" and list the tiles. In our example expression, we might say "x plus y, where x is the element of a at index 3 and y is 2 plus 3."

Tiling schemes have been used for years to translate C expressions to another language that does not allow nested constructs – machine language. Once the tiles are described by tree patterns, the two most common methods for tiling an expression are maximal munch and dynamic programming.[52] BabelBuster uses maximal munch. See Figure A1 for a pictorial representation of the tiling scheme.

[52] Andrew W. Appel, Modern Compiler Implementation in ML, 190-191.

For C expressions that are translated to English, each tile can be composed of an expression with some expressions and temporary variables nested in it. The problem is to determine when to nest and when to create a temporary variable.

Figure A1. Tiling expressions. Each tile represents an English clause.



In the most conservative case, we will never nest an expression. Thus, the C expression "(&x | &y) + 8" becomes "a plus b, where a is the bitwise or of c and d, b is 8, c is the address of e, d is the address of f, e is x, and f is y". While this is certain to be correct, our expression will be more readable if we sometimes decided to nest, as in "a plus 8, where a is the bitwise or of the address of x and the address of y".

Clearly, we must be certain not to nest if it will result in an ambiguity. Also, we must be careful not to nest if it will deter from the readability of the English. BabelBuster was written so that only certain types of expressions can be nested in a given type of expression. Thus, nestability is a predicate on the type of the outer expression and the type of the inner expression.

The names of temporary variables in expressions were e1, e2, and so on. For less complicated expressions, such names might seem rather obtrusive. But for extremely complex expressions, they help the reader of the English prose to find where the temporary variable's contents are described.

To allow flexibility in deciding just what expressions can be nested in what other expressions, the parser rules were chosen to allow the *least* conservative tiling scheme (i.e., "(&x | &y) + 8" becomes "the bitwise or of the address of x and the address of y plus 8"). Note that such a scheme is ambiguous. Thus, it was necessary to be careful to avoid "tickling" the ambiguity by using a conservative enough tiling scheme when producing the English translation. This parsing strategy resulted in a large number of shift-reduce errors, as expected, which somewhat encumbered the debugging process.

Two forms of C initializer expressions were also translated to English – simple and aggregate. Simple expressions were trivially translated. Aggregate expressions (comma-separated initializer expressions enclosed in curly braces) typically correspond to array initializers, but can also be structure initializers. These were both handled in the same way. If the aggregate array was made up of simple initializer expressions, their contents were simply listed. If it was made up of aggregate initializer expressions, it was referred to as "the array with values a0 to aN, where a0 is ….". Thus, the initializer expression "{{{1, 2}, {3, 4}}, {{5, 6}, {7, 8}}}" is translated as "the array with elements a0 to a1, where a0 is the array with elements a0_0 to a0_1, where a0_0 is the array with elements 1 and 2 and a0_1 is the array with elements 3 and 4 and a1 is the array with elements a1_0 to a1_1, where a1_0 is the array with elements 5 and 6 and a1_1 is the array with elements 7 and 8." Note that this is a simplification; to correctly translate

aggregate expressions that correspond to structure initializers, we must first use the type information to determine what type the aggregate initializer is.

Nested statements were handled by the same tiling strategy as nested expressions, except the function to determine nestability takes only the inner statement; thus, a statement can either be nested in all other kinds of statements, or in no other kinds of statements. If a statement is not nestable, we refer to it as an operation (or block) and then explain how to perform the operation in a later sentence.

Transitions were added to the English description of the algorithm in an attempt to improve readability and avoid the tedious nature of repeated imperative statements. To denote the end of an operation, we say "continuing on"; in addition, it was found that three other transitions ("then", "next", and "after that") were sufficient. For instance, the following series of statements can be greatly improved: "To perform operation 3, continue to do nothing as long as done is equal to zero. Increment x by 7. Decrement z by 6. Apply x and z to f. Return the result of calling clean_up." With transitions, it is much better: "To perform operation 3, continue to do nothing as long as done is equal to zero. Continuing on, we next increment x by 7. Then decrement z by 6. Next, apply x and z to f. After that, return the result of calling clean_up." Here, the importance of transitions is clearly understood.

So that the lexer could distinguish between variable (ID) tokens and keyword tokens, variable names that clashed with one of the English keywords were quoted in the English text. To demonstrate this, consider what happens when the C declaration "int one = 1;" is translated to English. If we do not quote the variable name, we get "Let one be an integer whose initial value is 1." This seems quite understandable, but the lexer

recognizes “one” as a keyword. Instead of returning the ID token with value “one” as it should, the lexer returns the ONE token. The parser is thoroughly confused. So if we quote the variable name, as in “Let ‘one’ be an integer whose initial value is 1”, the lexer will recognize that “one” is an ID. Note that quoting objects with unusual names is standard in English text.

Non-functional information such as comments and unary pluses were removed from the program in translation to English.

Finally, lists of variables declared with the same type were described in the same English sentence. For instance, “int x, y, z;” becomes “Let x, y, and z each be an integer.”

One problem that arose in the creation of BabelBuster is that the lexer table becomes extremely large if many large English phrases are lexed at once. To reduce the size of the table, alphanumeric tokens were processed by the lexer, and a hash table was used to determine if they were variable names or special tokens. To further reduce the lexer table size, no tokens with spaces in them (i.e., no English phrases) were lexed, with the exception of a few phrases here and there to simplify the LALR(1) grammar. For instance, the parsing rules for a structure definition are as follows:

```
structdef: structdef' NO MEMBER
         | structdef' MEMBER smnlist COMWHERE smlist
```

Here, the MEMBER token corresponds to the English word “member”, NO corresponds to “no”, and COMWHERE corresponds to “, where”. Additionally, a typical example of a “structdef'” is “let S be a structure with”, a typical “smnlist” might be “x, y, and z”, and a corresponding “smlist” might be “x is an integer, y is a character, and z is a floating-

point number". For these typical examples, the entire structure definition is "let S be a structure with members x, y, and z, where x is an integer, y is a character, and z is a floating-point number".

Suppose we try to split COMWHERE into separate tokens for "," and "where". Then we run into a problem when the parser reaches the "," after "x". Here, it is not possible for an LALR(1) parser to determine if the list has just a single member and the "," is part of COMWHERE or if the "," signals that more members are to follow. Rather than write an extra rule wherever such a situation occurred in the grammar, "," and "where" were combined into a single token.

A.2 Discussion

BabelBuster succeeds in correctly translating the entire C language. By "succeeds", I mean that the program compiled from the English will have the same effects as the program compiled from the original program. (See Appendix B for complete-program examples.) The English is quite readable, but several things could be done to make it even more readable and true to the original C code.

First, types could be treated with the same nestability logic as expressions. Furthermore, since types can be nested within expressions and expressions within types (i.e., "int y[5 + (int)4.5]"), we can treat them both as "clauses." Then we can tile clauses instead of expressions and types separately.

Second, we can experiment with new nestability predicates. For instance, BabelBuster allows us to nest the dereferencing operator within itself. Thus, "****x" becomes "the variable pointed to by the variable pointed to by the variable pointed to by

the variable pointed to by x". While this might be easy enough to understand, it might be possible to nest too deeply. So we might try to stop nesting once we have reached a certain depth.

Next, we can experiment with representing sentences and expressions in multiple forms. For instance, "x+y" is most easily read as "x plus y." But if we have "&x+y", we must write "a plus y, where a is the address of x" because "the address of x plus y" is ambiguous. (It could be interpreted as "&(x+y)" (wrong) or "(&x)+y".) However, we could write "the sum of the address of x and y". This is not ambiguous, and might be better for this particular situation.

We could also experiment with various strategies to lex and parse the English prose. For example, the COMMA token led to problems in parsing because it made it difficult to construct an LALR(1) grammar. But perhaps if we ignore commas (which seem somewhat redundant), it will be easier to create an LALR(1) English grammar. Also, often a trailing "s" at the end of a word was ignored by BabelBuster in lexing. For instance, "argument x" results in the same two tokens as "arguments x". While the trailing "s" was thought to be redundant, perhaps if we do not ignore it, we will be improve our chances of quickly constructing an LALR(1) grammar.

Another interesting problem might be to write a program that determines just what clauses we can nest in other clauses without causing an ambiguity. BabelBuster's nestability logic is hard-coded. But what if we said that a clause could be nested in another if and only if it will not arise in an ambiguity? Such a constraint seems difficult to implement.

Finally, the C produced from translation to English and back could be more true to the original version. For instance, comments could be turned into footnotes, or could be parenthesized in the English, and returned to comments when translating back to C. Structure initializers could be indicated by "the structure with contents …" rather than "the array with values …." Unary plus could be indicated in the English text, and hexadecimal numbers could retain their base (rather than converting to decimal).

A.3 Abridged English Grammar

```
program: ast

ast: declarationList newpar ast
  | functionDef ast
  | (* empty *)

declarationList:   declaration
 | vardecList
 | declarationList DOT declaration
 | declarationList DOT vardecList

vardecList: LET smnlist EACH BE article typ

declaration : tydec
          | tydef
          | vardec

tydef : structdef
     | uniondef
     | tydef'
     | enumdef

enumdef': LET id BE article ENUMERATION WITH

enumdef: enumdef' NO VALUE
   | enumdef' VALUE memintlist

memintlist: memintlist' ", and" memint
         | memint AND memint
         | memint
```

```
memintlist': memint COMMA memint
          | memintlist' COMMA memint

memint: id EQUAL TO INT

tydef': LET id BE ANOTHER NAME FOR article typ

uniondef': LET id BE article UNION WITH

uniondef: uniondef' NO MEMBER
  | uniondef' MEMBER smnlist “, where” smlist

structdef':LET id BE article STRUCTURE WITH

structdef: structdef' NO MEMBER
  | structdef' MEMBER smnlist “, where” smlist
  | structdef' MEMBER smnlist

smlist: smlist' “, and” sm
     | sm AND sm
     | sm

smlist': sm COMMA sm
      | smlist' COMMA sm

sm: id IS article bitopt typ

smnlist: smnlist' “, and” smn
      | smn AND smn
      | smn

smnlist': smn COMMA smn
       | smnlist' COMMA smn

smn: article UNNAMED bitopt typ
  | id

bitopt: (* empty *)
     | NBITS

tydec : ASSUME id IS article STRUCTURE
     | ASSUME id IS article UNION

vardec : LET id BE article typ
      | LET id BE article typ COMMA WHOSE INITIAL VALUE IS initExpr
```

```
initExpr: expr
      | initArrExpr

tagInitExprList: tagInitExpr
           | tagInitExpr AND tagInitExpr
           | tagInitExprList' “, and” tagInitExpr

tagInitExprList': tagInitExpr COMMA tagInitExpr
         | tagInitExprList' COMMA tagInitExpr

tagInitExpr: ELEMID IS initExpr

initArrExpr: THE ARRAY WITH NO ELEMENT
         | THE ARRAY WHOSE ONLY ELEMENT IS initExpr
         | THE ARRAY WITH ELEMENT ELEMID TO ELEMID “, where”
tagInitExprList

typ : VOID
    | ARGUMENT LIST
    | qual typ
    | numeric
    | stClass typ
    | POINTER TO article typ
    | id
    | id STRUCTURE
    | ARRAY OF LENGTH expr EACH ELEMENT OF WHICH IS article typ
    | FUNCTION WITH NO ARGUMENT result
    | FUNCTION WITH ARGUMENT artyplist “, and” result
    | UNKNOWN TYPE

result: WITH NO RESULT
     | RETURNING article typ

artyplist: artyplist' “, and” article typ
        | article typ AND article typ
        | article typ

artyplist': article typ COMMA article typ
         | artyplist' COMMA article typ

numeric : sat frac sign ik

sign: SIGNED
   | UNSIGNED
   | (* blank *)
```

```
ik: CHARACTER
  | INTEGER
  | EXTRALONG INTEGER
  | FLOATPN
  | DOUBLEP FLOATPN

expr: coreExpr

coreExpr: simpleExpr
       | simpleExpr “, where” simpleTaggedExprList

tag: TEMPEXP IS

simpleTaggedExprList: tag simpleExpr
    | tag simpleExpr AND tag simpleExpr
    | simpleTaggedExprList' “, and” tag simpleExpr

simpleTaggedExprList': tag simpleExpr COMMA tag simpleExpr
                     | simpleTaggedExprList' COMMA tag simpleExpr

simpleExprList: simpleExprList' “, and” simpleExpr

simpleExprList': simpleExpr COMMA simpleExpr
              | simpleExprList' COMMA simpleExpr

binop: simpleExpr PLUS simpleExpr
  | simpleExpr MINUS simpleExpr
  | NONZERO IF AND ONLY IF simpleExpr IS GREATER THAN simpleExpr
  | THE BITWISE OR OF simpleExpr AND simpleExpr
  | simpleExpr SHIFTED LEFT simpleExpr BITS
  | simpleExpr AFTER INCREMENTING simpleExpr BY simpleExpr

unop: ZERO IF AND ONLY IF simpleBoolExpr
  | THE ADDITIVE INVERSE OF simpleExpr
  | THE BITWISE COMPLEMENT OF simpleExpr
  | simpleExpr AFTER INCREMENTING simpleExpr BY ONE
  | simpleExpr BEFORE DECREMENTING simpleExpr BY ONE

call: THE RESULT OF CALLING simpleExpr
    | THE RESULT OF PASSING simpleExprList TO simpleExpr

simpleBoolExpr: simpleExpr IS GREATER THAN simpleExpr
   | simpleExpr IS LESS THAN simpleExpr
   | BOTH simpleExpr AND simpleExpr ARE NONZERO
   | EITHER simpleExpr OR simpleExpr IS NONZERO
```

```
  | ZERO EQUALS simpleExpr
  | ZERO DOES NOT EQUAL simpleExpr

simpleExpr: TEMPEXP
        | INT | REAL | string' | id | call
        | simpleExpr IF simpleBoolExpr ANDOTHERWISE simpleExpr
        | THE CONTENTS OF simpleExpr UPON ASSIGNING simpleExpr TO IT
        | THE ELEMENT OF simpleExpr ATINDEX simpleExpr
        | MEMBER id OF simpleExpr
        | MEMBER id OF THE STRUCTURE POINTED TO BY simpleExpr
        | THE VARIABLE POINTED TO BY simpleExpr
        | THE ADDRESS OF simpleExpr
        | binop
        | unop
        | simpleExpr CAST TO article typ
        | ENUMERATION id
        | THE SIZE OF article typ

simpleExprStmt: COMPUTE NOTHING
      | EVALUATE simpleExpr
      | CALL simpleExpr
      | PASS simpleExprList TO simpleExpr
      | SET simpleExpr EQUAL TO simpleExpr
      | INCREMENT simpleExpr BY simpleExpr
      | ASSIGN simpleExpr TIMES simpleExpr TO simpleExpr
      | INCREMENT simpleExpr BY ONE
      | DECREMENT simpleExpr BY ONE

exprStmt: simpleExprStmt “, where” simpleTaggedExprList
      | simpleExprStmt

boolExpr: simpleBoolExpr
      | simpleBoolExpr “, where” simpleTaggedExprList

blockend: FIRST declarationList blockend'
      | FIRST declarationList DOT stmtList blockend'
      | FIRST stmtList blockend'
      | DO NOTHING
blockend': DOT THIS ENDS BLOCK INT

coreStmt: exprStmt (A.Expr exprStmt)
      | blockInS COMMA WHICH IS DEFINED AS FOLLOWS DOT declaration
      | CONTINUE TO coreStmt AS LONG AS boolExpr AT
            THE BEGINNING OF THE ITERATION
      | REPEATEDLY coreStmt startExpr loopExpr
      | LABEL THIS POINT IN THE CODE id DOT coreStmt
```

```
      | LET THIS BE THE START OF CASE INT DOT coreStmt
      | GO TO LABEL id
      | BREAK FROM THE NEAREST ENCLOSING LOOP OR CASEBLOCK
      | CONTINUE EXECUTION AT THE START OF THE LOOP
      | RETURN FROM THE FUNCTION
      | RETURN expr
      | blockInS PROVIDED boolExpr DOT followUpStmt
      | opInS PROVIDED boolExpr DOT followUpStmt
      | coreStmt “, starting” AT CASE expr “, or” THE DEFAULT
            CASE IF NO SUCH CASE EXISTS

startExpr: SEMIC BEFORE STARTING THIS LOOP COMMA exprStmt
       | (*nothing*)

loopExpr: SEMIC UPON COMPLETION OF EACH ITERATION OF THE LOOP
COMMA exprStmt
       | (*nothing*)

blockInS: PERFORM BLOCK INT
opInS: PERFORM OPERATION INT

followUpStmt: TO opInS COMMA coreStmt
         | TO blockInS COMMA blockend
         | TO blockInS COMMA declaration
         | TO blockInS COMMA coreStmt

stmtList: coreStmt
      | stmtList DOT coreStmt

function": LET id BE article FUNCTION result DOT IT IS CALLED WITH
        | LET id BE article stClass FUNCTION result DOT IT IS CALLED WITH

function': function" NO ARGUMENT DOT     (function", nil)
       | function" ARGUMENT smnlist DOT
       | function" ARGUMENT smnlist “, where” smlist DOT

functionDef: function' TO PERFORM THE FUNCTION COMMA DO NOTHING
newpar
        | function' TO PERFORM THE FUNCTION COMMA declarationList newpar
        | function' TO PERFORM THE FUNCTION COMMA declarationList
                    DOT stmtList newpar
        | function' TO PERFORM THE FUNCTION COMMA stmtList newpar

id : ID
  | QUOTEID
```

string': THE STRING QUOTEID

newpar: DOT EOP

Appendix B

Example Translations

C source:

```
extern printf ();

enum S {
  x1, x2
};

main () {

  struct S {
    int abdd;
  } j;
  int i = x1;

  switch(i) {
    case 1:
    case 2: printf ("%d\n",45);
    default: printf ("%d\n",45);;;
  }

  if(i) { printf ("%d\n",i); }

  while(i--) { 2; }

  for(i; i; i++);

  do {3;} while (i);
}
```

English translation:

Let printf be an external function with no arguments returning an integer. Let S be an enumeration with values x1 equal to 0 and x2 equal to 1.

Let main be a function returning an integer. It is called with no arguments. To perform the function, let S be a structure with member abdd, where abdd is an integer. Let j be a S structure. Let i be an integer, whose initial value is enumeration x1. Perform block 1, starting at case i, or the default case if no such case exists. To perform block 1, first let this be the start of case 1. Let this be the start of case 2. Pass the string "%d\n" and 45 to printf. Then let this be the start of the default case. Pass the string "%d\n" and 45 to printf. Next, compute nothing. After that, compute nothing. This ends block 1. Continuing on, we next perform block 2 provided zero does not equal i. To perform block 2, pass the string "%d\n" and i to printf. Continuing on, we next continue to perform block 3 as long as zero does not equal i before decrementing i by one at the beginning of the iteration. To perform block 3, evaluate 2. Continuing on, we next continue to compute nothing as long as zero does not equal i prior to the iteration; before starting this loop, evaluate i; upon completion of each iteration of the loop, increment i by one. Then continue to perform block 4 as long as zero does not equal i at the end of the iteration. To perform block 4, evaluate 3.

C source:

```
struct hi {
  int i;
  struct hi_nest {
    int k, j : 5;
    double : 4;
  }
  p;
} *k;
```

```
int (*f)(int, int*);
int (*g)();

int main() {
  double d, e;
  int i;
  i = f(k->p.j, (int)d + (&(k->p.k)));
}
```

English translation:

Let hi_nest be a structure with members k, j, and an unnamed 4-bit double-precision floating point number, where k is an integer and j is a 5-bit integer. Let hi be a structure with members i and p, where i is an integer and p is a hi_nest structure. Let k be a pointer to a hi structure. Let f be a pointer to a function with arguments an integer and a pointer to an integer, and returning an integer. Let g be a pointer to a function with no arguments returning an integer.

Let main be a function returning an integer. It is called with no arguments. To perform the function, let d and "e" each be a double-precision floating point number. Let i be an integer. Set i equal to the result of passing member j of member p of the structure pointed to by k and e1 to f, where e1 is e2 plus e3, e2 is d cast to an integer, and e3 is the address of member k of member p of the structure pointed to by k.

References

Appel, Andrew W. Modern Compiler Implementation in ML. Cambridge: University Press, 1998.

Appel, Andrew W., and Edward W. Felten. "Technological Access Control Interferes with Noninfringing Scholarship." Communications of the ACM, 43 (9), September 2000. Archived at <http://info.acm.org/pubs/articles/journals/cacm/2000-43-9/p21-appel/p21-appel.pdf>.

Bernstein v. U.S. Department of State, 974 F. Supp. 1288 (N.D. California). Transcript of testimony, 20 October 1995.

Burk, Dan L. "Copyrightable Functions and Patentable Speech." Communications of the ACM. 44 (2), February 2001. Archived at <http://www.acm.org/pubs/articles/journals/cacm/2001-44-2/p69-burk/p69-burk.pdf>.

"C to English and English to C Translator." From Slashdot message board, <http://slashdot.org/articles/980815/105250.shtml>. 15-20 August 1998.

"ckit." <http://cm.bell-labs.com/cm/cs/what/smlnj/doc/ckit/index.html>. 25 April 2001.

Cohen, Julie. "Unfair Use." The New Republic Online <www.tnr.com>, 23 May 2000. Archived at <http://www.tnr.com/online/cohen052300.html>.

Craver, Scott A., Drew Dean, Edward W. Felten, Bede Liu, John R. McGregor, Adam Stubblefield, Ben Swartzlander, Dan S. Wallach, and Min Wu. "Reading between the Lines: Lessons from the SDMI Challenge." <http://cryptome.org/sdmi-attack.htm>. 27 April 2001.

De Sola Pool, Ithiel. Technologies Without Boundaries: On Telecommunications in a Global Age, edited by Eli M. Noam. Cambridge: Harvard University Press, 1990, 15. Quoted in Lawrence Lessig. Code. 180. New York: Basic Books, 1999.

Digital Millennium Copyright Act (Public Law 105-304) of 28 October 1998, section 1201(a).

"Digital Millennium Copyright Act." <http://www.educause.edu/issues/dmca.html>. 20 April 2001.

Irons, Peter, and Stephanie Guitton, ed. May it Please the Court. New York: The New Press, 1993.

Junger v. Daley, 8 F. Supp. 2d 708 (N.D. Ohio). Decision, 2 July 1998.

Kirkpatrick, David D. "Court Halts Book Based on 'Gone With the Wind.'" The New York Times, 21 April 2001, late ed., p. A1.

Lessig, Lawrence. Code. New York: Basic Books, 1999.

Markoff, John. "Record Panel Threatens Researcher with Lawsuit," The New York Times, <www.nytimes.com>. 24 April 2001.

Marttila, Leevi. "Accurate Language to Inaccurate Language (and back) Translator c2txt2c v0.2.1." <http://personal.sip.fi/~lm/c2txt2c/>. 15 January 2000.

Mellgren, Doug. "Speaking in Code: Quiet, Polite Teen at Center of DVD-Copying Firstorm." ABCNews.com <abcnews.go.com>. 26 February 2001. Archived at <http://more.abcnews.go.com/sections/scitech/DailyNews/johansen010226.html>.

Minneapolis Star & Tribune Co. v. Minnesota Comm'r of Revenue, 460 U.S. 575. Decision, 29 March 1983.

Moglen, Eben. "Anarchism Triumphant: Free Software and the Death of Copyright." First Monday <www.firstmonday.org>, August 1999. Archived at <http://firstmonday.org/issues/issue4_8/moglen/>.

Morland, Howard. "The Holocaust Bomb: A Question of Time." <http://www.fas.org/sgp/eprint/morland.html>. 15 November 1999.

Schneider v. State, 308 U.S. 147. Decision, 22 November 1939.

Schwartz, Omri. "Converting C to English with Perl." The Perl Journal <www.tpj.com>, Fall 2000.

Stevenson, Frank. "Cryptanalysis of the Content Scrambling System." <http://www.cs.cmu.edu/~dst/DeCSS/FrankStevenson/analysis.ps>. 13 November 1999.

Tien, Lee. "Publishing Software as a Speech Act." 15 *Berkeley Technology Law Journal* 629 (Spring 2000).

Touretzky, David. "Gallery of CSS Descramblers." <http://www.cs.cmu.edu/~dst/DeCSS/Gallery/index.html>. 14 April 2001.

Turner Broadcasting System, Inc. v. Federal Communications Commission, 512 U.S. 622. Decision, 27 June 1994.

Universal City Studios, Inc. v. Reimerdes, 111 F. Supp. 2d 346 (S.D.N.Y.). Transcript of testimony, 17,20,25 July 2000.

Universal City Studios, Inc. v. Reimerdes, 111 F. Supp. 2d 346 (S.D.N.Y.). Decision, 17 August 2000.