`$Id: $`

Algorithmic self-reflection in the poetry of François le Lionnais, Jodi, Eugen Gomringer, Quirinus Kuhlmann and Graham Harwood.

WHEN WRITING EXECUTES ITSELF

FLORIAN CRAMER

ABSTRACT. Computers and the Internet differ from all other so-called 'new media' in that they are based on formal instruction codes. It thus comes to no surprise that (a) much if not most of the most exciting digital art and poetry of the last few years reflects the aesthetics and poetics of computer codes and (b) in doing so, reveals striking similarities to older concrete poetry and experimental literature. My talk attempts to sketch a history of executable codes in writing from 17th century permutational poetry and 20th century Oulipo/concrete poetry to contemporary, post-Net.art 'codeworks', concluding that the poetics of self-executable writing has shifted significantly since executable code has turned from a hidden and scarce to a visible und abundant cultural resource in the late 1990s.

POETRY AND PROGRAM CODE

London.pl. The first proposal for writing poetry in computer programming languages was published in 1962 in the first manifesto of the Oulipo, the poets and mathematicians founded by Raymond Queneau and François le Lionnais. The manifesto called upon for advancements in "special vocabularies (such as those by ravens, foxes, fish; the Algol language of electronic computers etc." ("des vocabulaires particuliers (corbeaux, renards, marsouins ; langage Algol des ordinateurs électroniques, etc.)". Still, it took another ten years — until 1972 — that the first poem in a computer programming language was written by the mathematician and Oulipo member François le Lionnais:

```
Table
Begin: to make format,
go down to comment
while channel not false
(if not true). End.
```

Date: April 21, 2003.

What reads like a pseudo-haiku which meditates technology instead of nature, or like a piece of concrete poetry in which the title "table" reflects itself, is at the same time a piece of computer sourcecode, written the ALGOL programming language as it had been popular in the 1960s and 1970s. However, the sourcecode doesn't compile and execute. As suggested in the Oulipo manifesto, it rather is a text written in one of many special languages than a text concerned with algorithmics and the borders between language and computation.

Otherwise, the writing would be readable not only in two, but three different ways: First as an English text, second as a sourcecode in a computer programming language, and third as the text/data generated by the sourcecode instructions. In other words: Text becomes executable, and auto-generative by embedding a calculus. (...)

Independently from Oulipo's experiments with ALGOL, programming language poetry became popular in the early 1990s in the community around the Perl scripting language. Coined by Perl creator Larry Wall, Perl poetry took off in the Internet and soon after underwent a systematic taxonomy in Sharon Hopkins' 1991 paper "Camels and Needles: Computer Poetry Meets the Perl Programming Language" `http://www.wall.org/~sharon/plpaper.ps`. Two Perl Poetry contests in 2000 and 2001 helped to keep the genre alive. But to date, most Perl poetry remains disappointing from literary and artistic points of view. An example, taken from the poem "love.pl":

```
our $life = \$love and $togetherness;
   and: foreach my $sweet (@first) {
       little: until ($we . $met) { last 'and' }
   }
   if ($now . $we) {  goto marry; $we . $shall }
   bless our $life, More;
```

In no other digital poetry the relationship of the text and the technology is as intrinsic, to the point, potentially reflexive and virulent as in programming language poetry — given that a program code poem could literally trash a computer (jaromils forkbomb). Yet the sentimentalism and subjective introspection of much if not most Perl poetry passes off as ironic at best and clearly gets tedious over time. So the question is: Is programming language poetry yet another genre of digital art which is conceptually interesting, but suffering from digital kitsch like so much other electronic and so-called multimedia poetry?

Just as the most radical experimental poetry in early modernism was written by visual artists rather than by poets proper, it seems to me as if some of the best digital poetry today is written by net.artists, perhaps because artists have historically been more sophisticated in adapting their work to the specifics of a new material. Which doesn't mean that exciting digital poetry would exist outside the tradition of poetry.

An outstanding example is the newer Perl poetry of Graham Harwood. Harwood is a well known underground and digital artist. In 1988, he co-organized the London Festival of Plagiarism (in which conference participants tENTATIVELY, a cONVENIENCE and Miekal And were involved), in 1989, he began to experiment with artistic modifications of computer games, in 1997, he was the main programm of the experimental web browser end net.art classical I/O/D Webstalker, later he co-founded the Mongrel collective and co-developed the manipulated search engine "Natural Selection". His Perl poem "London.pl" from 2001 is a transcription, or — in programmer jargon – a "port" of William Blake's 1791 poem "London", as indicated by the filename suffix ".pl". Blake's original poem reads:

> London
> I wander thro' each charter'd street,
> Near where the charter'd Thames does flow,
> And mark in every fact I meet
> Marks of weakness, marks of woe.
> In every cry of every Man,
> In every Infant's cry of fear,
> In every voice, in every ban,
> The mind-forg'd manacles I hear.
> How the Chimney-sweeper's cry
> Every black'ning Church appalls;
> And the hapless Soldier's sigh
> Runs in blood down Palace walls.
> But most thro' midnight streets I hear
> How the youthful Harlot's curse
> Blasts the new born Infant's tear,
> And blights with plagues the Marriage hearse.

Harwood's text reads as follows:

```
#  Perl Routines to Manipulate London w.blake@scotoma.org
   #  London.pl, v 0.0.0 1792/02/23 08:16:43
   # UNFINISHED
   #  Copyright (c) 1792-2002 William Blake
   #  Unpublished work reconstituted W.Blake@Scotoma.org.
```

```
#  Permission granted to use and modify and append this library so long
#  copyright above is maintained, modifications are documented, and
#  credit is given for any use of the library.
#
#  Thanks are due to many people for reporting bugs and suggestions

#  For more information, see http://www.scotoma.org
#
#  Grave the sentence deep. My love of London held in torment.
#  Heavy, rains of cruelty, disguised in spectacular investments.
#  Accumulate interest in Jealousy,Terror and Secrecy.
#  The bloated Square mile
#  Gifts this isle.
#
# In this citys dark gates - the tree of knowledge leads to this mansio
# Here the dress code of secrecy cloaks the flesh in fear.
# This is how the proprietary city gets built,
# Hidden in every proprietary street,
# In every proprietary house,
# In every proprietary possession we meet.

# NAME
#    London - Simple Act Redress

# The American War was the last judgment on England.
# Inoculated against the sewer. Albion's Angels
# Rise up on wings of iron & steel, spreadsheet & rule:
# To gift sanitation & sulphurous fire to:
# The wheat of Europe,
# The rice of Asia,
# The potato of America,
# The maize of Africa.
# Massacre-bloated, angels crawl from the corpse of war.
# Five times fatter than when they entered.

# Choking lays the sickening Leveller-republican. Caustic fumes - dusts
# gust from wars - grinding wheels - mills of cruelty - mills of terror
# Every light ray turned to shadow and despair. to rikets - scabies - t
# Until the dark sun never set on the Hanoverian empire.
#
# Rise then the Leveller-republic, rise on wings of knowledge flowing i
# For heaven is more knowledge then one man can muster in a lifetime.
```

```
# For hell is more knowledge then one man can muster in a lifetime.




# SYNOPIS and DESCRIPTION
#  This Library is for redressing the gross loss to Londons Imagination
# beaten enslaved fucked and exploited to death from 1792  to the prese
# We see this loss in every face marked with weakness or marked with wo

use PublicAddressSystem qw(Hampstead Westminster Lambeth Chertsey);
# PublicAddressSystem is an I/O library for the manipulation of the whe
# Vortex4 129db outside warning system.
#
# from Hampstead in the North, to Peckham in the South,
# from Bow in the East to Chertsey in the West.

# Find and calculate the gross lung-capacity of the children screaming
# calculate the air displacement needed to represent the public scream
#  set PublicAddressSystem intance and transmit the output.
# to do this we approximate that there are 7452520 or so faces that liv
# Found near where the charter'd Thames does flow.
#
#  DATATYPES USED:

local @SocialClass = qw(
RentBoy
YoungGirl-Syphalitic-Innoculator
CrackKid WarBeatenKid
ForcedFeatalAbortion
Chimney-Sweeps
UnCategorisedVictim

);


#
#  These are a series of anonymous hashes;
#  At least one is required at compile time:
#

local %DeadChildIndex;
```

```
#  The Data for the DeadChildIndex should be structured as follows:
#
#  %{DeadChildIndex} => {
# IndexValue => {
# Name  =>  " Child name If known else undefined ";
# Age =>  " Must be under 14 or the code will throw an
# exception due to $COMPLICITY";
# Height =>"Height of the child"
# SocialClass =>"RentBoy YoungGirl-Syphalitic-Innoculator
# CrackKid WarBeatenKid ForcedFeatalAbortion
# Chimney-Sweeps UncategorisedVictim "
# }, As many as found
# }
#

# CryOfEveryMan
# First we add the Class attribute to the DeadChild instance under revi
# Next add the VitalLungCapacity of that childs ability to scream

sub CryOfEveryMan {
my $index = shift;
# Because a child may belong to one or more SocialClass
# traverse the list adding the prospects of that SocialClass
foreach my $Class (@SocialClasses){
# Add the contents of this $Class to $DeadChildIndex->{$Index}
# Class attribute
if( $Class eq $DeadChildIndex->{$Index}->{Class}){
$DeadChildIndex->{$Index}->{Class} = %{$Class} ;
}else{
warn "$DeadChildIndex->{$Index}->{Name} is not a member of = $Class\n";
}
}
$DeadChildIndex->{$Index}->{Class} = %{UncategorisedVictim} if ! $DeadC
# The average daily scream output of fear for the period 1792-2002 is 6
my $TotalDaysLived = ($DeadChildIndex->{$Index}->{Class}->{LifeExpectar
# Calculate the gross $Lung Capacity For Screaming for this child
my $LungCapacityForScreaming = &Get_VitalLungCapacity(\%{$DeadChildInde
# asign to $DeadChildIndex->{$Index}->{ScreamInFear}
$DeadChildIndex->{$Index}->{ScreamInFear} = $LungCapacityForScreaming;
}
```

```
#    need a function to play the sound file for
# lenght of time * volume of speaker system * air displacement


# The Get_VitalLungCapacity routine uses the Age and Height entry of th
# to calculate the Lung-Capacity of the dead child. This is then used t
# volume and capacity of screams when terrified.
sub Get_VitalLungCapcity{
my $DeadChild = shift;
my (
$VitalLungCapcity,#  vital lung capacity in liters of air
$Height,#  is height in centimeters
$Age,#  is age in years
);

$Height  = $DeadChild->{Height}  unless ! defined $DeadChild->{Height};
$Age  = $DeadChild->{Age}  unless ! defined $DeadChild->{Age};

if ($Height && $Age){
#
#   lung capacity increases with height, but decreases with age.
#   So a person screams the most when they are as tall as they're going
#   (Probably about 18 or 20 years old.)
# This falls outside of our basic parameter of 0 to 14 years.
# But the calculation is still useful
$VitalLungCapacity = ((0.041 * $Height) - (0.018 * $Age)) - 2.69 ;
return $VitalLungCapacity;
}else{
# we may not know the height, try to guess from SocialClass
if(! $Height){$Height  = Get_HeightFromClass(Height => $DeadChild->{Soc
# we may not know the Age, try to guess from SocialClass
if(! $Age){$Age  = Get_AgeFromClass(Age => $DeadChild->{SocialClass})}
if($Age && $Height){
$VitalLungCapcity = ((0.041 * $Height) - (0.018 * $Age)) - 2.69 ;
return $VitalLungCapacity;
}else{
# Approximate it
# The average 6 year old child is about 120 cm tall. So $Height =130.0
# Put this into our equation and we get that the VitalLungCapacity is a
# The average 14 year old teenager is about 160 cm tall. So Height=160
# This gives us a vital lung capacity of about 3.6 liters.
if($Age){
```

```
$VitalLungCapacity = ((3.6) - (2.1) / 8.0) * $Age;
return $VitalLungCapcity;
}else{
$VitalLungCapacity = ((3.6) - (2.1) / 8.0) * int(rand(14)) ;
return $VitalLungCapcity;
}

}

}


}
```

While the Perl code is syntactically correct and executable in itself, it still does not properly run because it relies upon a non-existing, imaginary software component, namely the module “PublicAddressSystem.pm”. 58 of the 189 lines of “London.pl” contain program code, the rest are comments. To Perl-literate readers, it thus looks as if the code was unfinished and parts were missing. Aside from the comments, the sourcecode contains a definition of what in Perl is called an “anonymous array”, i.e. a variable storing several values at once, called “@SocialClass”, a database (or, in programmer’s lingo: “nested hashtable”) “%DeadChildrenIndex”, and two sub-programs (“subroutines”) “CryOfEveryMan” and “Get_VitalLungCapacity”. Thus, Harwood’s London.pl translates what Blake’s poem “London” describes into a symbolic machinery. It is an interpretation of the older poem in a double sense: as code executed by a programming language “interpreter”, and as a social-political reading of Blake’s poem, focusing the subject onto dead children:

```
#       SYNOPIS and DESCRIPTION
#       This Library is for redressing the gross loss to Londons
#       Imagination of children
#       beaten enslaved fucked and exploited to death from 1792  to the
#       present.
#       We see this loss in every face marked with weakness or marked
#       with woe.
```

By transcribing William Blake’s walk through the city into a program and juxtaposing its walk with the stack-execution of program code, Blake’s observations and the woes he tells turn into a macabre process. Which within the sourcecode is described as follows:

```
#       Find and calculate the gross lung-capacity of the children
#       screaming from 1792 to the present
#       calculate the air displacement needed to represent the public
#       scream
#       set PublicAddressSystem intance and transmit the output.
#       to do this we approximate that there are 7452520 or so faces
#       that live in the charter'd streets of London.
#       Found near where the charter'd Thames does flow.
```

The machine described and set into motion via is imaginary to a great extent, given that the code provides only a fragment of the calculation. But exactly through is fragmentation, it gains its monstrosity: A conflation of writing and machinery using the concept of Perl poetry for something that could be compared to Duchamp's equally imaginary and monstrous bridal machines.

Kuhlmann. But is calculus inscribed into poetry an invention of digital and programming language poetry? If one would write the history of literature more broadly as a history of text processing, encompassing all kinds of writing practices including religious and mystical next to poetical ones, poetic calculus would begin with the numerological and gematrical codes inscribed Babylonian graveyards (as analyzed in Franz Dornseiff's book "Mystik und Magie in der Sprache"), anagrammatical and word permutational language games as they are known from classical rhetoric and last not least the jewish and christian kabbalah.

It was in the 17th century, with the rediscovery of Ramon Lull's Ars Magna, that these rhetorical and mysticist-speculative traditions coincided; the word-permutational Proteus poetry of, among others, Thomas Lansius, Georg Philiip Harsdoerffer and Quirinus Kuhlmann at once reflected rhetoric, kabbalah and linguistic and philosophical theories of its time which conceived of language and knowledge as a combinatory system. The most systematic and extremist application can be found in the writings of the German late 17th century poet Quirinus Kuhlmann (who ended up travelling as a self-acclaimed mystical "monarch" and "prophet" through Europe and was burned in Moscow as a heretic).

Kuhlmann's poem the "41st Kiss of Love" from 1671 on the „change in human matters" expands the form of the one-line word permutational proteus verse to a multi-line word permutational Proteus sonnet.

Auf Nacht / Dunst / Schlacht / Frost / Wind / See / Hitz /
Süd / Ost / West / Nord / Sonn / Feur und *Plagen* /

Folgt Tag / Glantz / Blutt / Schnee / Still / Land / Blitz / Wärmd / Hitz / Lust / Kält / Licht / Brand und *Noth*:
Auf Leid / Pein / Schmach / Angst / Krig / Ach / Kreutz / Streit / Hohn / Schmertz / Qual / Tükk / Schimpf / als *Spott* /
Wil Freud / Zir / Ehr / Trost / Sig / Rath / Nutz / Frid / Lohn / Schertz / Ruh / Glükk / Glimpf / stets *tagen*.
Der Mond / Glunst / Rauch / Gems / Fisch / Gold / Perl / Baum / Flamm / Storch / Frosch / Lamm / Ochs / und *Magen*
Libt Schein / Stroh / Dampf / Berg / Flutt / Glutt / Schaum / Frucht / Asch / Dach / Teich / Feld / Wiß / und *Brod*:
Der Schütz / Mensch / Fleiß / Müh / Kunst / Spil / Schiff / Mund / Printz / Rach / Sorg / Geitz / Treu / und *GOtt* /
Suchts Zil / Schlaff / Preiß / Lob / Gunst / Zank / Port / Kuß / Thron / Mord / Sarg / Geld / Hold / *Danksagen*
Was Gutt / stark / schwer / recht / lang / groß / Weiß / eins / ja / Lufft / Feur / hoch / weit *genennt* /
Pflegt Böß / schwach / leicht / krum / breit / klein / schwarz / drei / Nein / Erd / Flutt / tiff / nah / *zumeiden* /
Auch Mutt / lib / klug / Witz / Geist / Seel / Freund / Lust / Zir / Ruhm / Frid / Schertz / Lob muß *scheiden* /
Wo Furcht / Haß / Trug / Wein / Fleisch / Leib / Feind / Weh / Schmach / Angst / Streit / Schmertz / Hohn *schon rennt*
Alles wechselt ; alles libet ; alles scheint was zu hassen:
Wer nur disem nach wird=denken / muß di Menschen Weißheit fassen.

(Short summary: The poem is an allegorical reflection of permutation. The permuting words — monosyllabic substantives — are an inventory of macrocosm and microcosm, derived partly from Lull's tables, from Salomo's proverbs and even intertextually from other 17th century permutational poems. The poem also has linguistic and hermetic subtexts I won't go into the details of here.)

In an afterword, the poet speculates that there are more permutations of the poems than grain of sands in the world and conceives of a "permutation wheel", a proto-computer hardware which would automatically calculate the permutations of the poem. If this wheel is the hardware of Kuhlmann's poetics, the poem is software, or: sourcecode. Generating excessive output from its compact input, it permutes itself syntactically all the while it tells of permutation semantically. It thus creates a unity of signifiers and signified, and words and things. Kuhlmann is quite aware of this when he conceives

of himself as a Christian kabbalist seeking to restore the language Adam spoke in Paradise and which, according to the Genesis, allowed him to exert demiurgic power upon the creatures by naming them.

Kuhlmann extended his vision into new foundation of knowledge based on combinatorics and algorithmics. Among others, he conceived of a "ars magna librum scribendi", a "great art of writing books" which would mechanically generate all existing and future books of the world. (A topic that reappears as a parody 50 years later in the Lagado chapter of Swift's Gulliver's Travels, and as an ironic philosophical and poetic speculation in Borges' Library of Babel.)

I digressed into these historical examples of algorithmic sourcecode literature to show that calculus in language and executable code in literature is nothing new, and doesn't even require computing hardware or formalized computer programming languages, as long as there is a formal instruction written into the text. (Other examples are Fluxus scores like those of George Brecht and La Monte Young, or the "Frame Tale" in John Barth's novel "Lost in the Funhouse" which only consists of the sentence "ONCE UPON A TIME THERE WAS A STORY THAT BEGAN" written as an endlessly looping sentence onto a Moebius.)

From Babylonian anagrammatics, Renaissance permutational poetry, 20th century experimental writing to today, I however think there a fundamental paradigm shift has occured with net.art and electronic poetry: Since the mid-1990s, algorithmic sourcecode is no longer a synthetical clean-room construct (even in those cases where it is being used to create collage-like poetry), but has become an abundant good subject to collage and pastiche itself. It is factually the advent of the Open Source/Free Software principle in digital art, and what Alan and other people call "Codework" is exactly founded on that. An example:

jodi vs. Eugen Gomringer. The following text was sent by the net.artists jodi to the rhizome mailing list on Oktober 22nd, 2001. One month after 9/11, it could be read as a commentary to world politics and its discussion on net cultural mailing lists:

```
$cd ug/models/soldier3
$origin 0 -6 24
$base base
$skin skin

$frame soldierc
$frame soldierd
```

```
/*

*/

void() army_fire;

void() army_stand1 =[ $soldierc,army_stand2 ]
{ai_stand();};
void() army_stand2 =[ $soldierc,army_stand3 ]
{ai_stand();};
void() army_stand3 =[ $soldierc,army_stand4 ]
{ai_stand();};
void() army_stand4 =[ $soldierc,army_stand5 ]
{ai_stand();};
void() army_stand5 =[ $soldierc,army_stand6 ]
{ai_stand();};
void() army_stand6 =[ $soldierc,army_stand7 ]
{ai_stand();};
void() army_stand7 =[ $soldierc,army_stand8 ]
{ai_stand();};
void() army_stand8 =[ $soldierc,army_stand1 ]
{ai_stand();};

void() army_walk1 =[ $soldierc,army_walk2 ] {
if (random() < 0.2)
sound (self, CHAN_VOICE, "soldier/idle.wav", 1, ATTN_IDLE);
ai_walk(1);};
void() army_walk2 =[ $soldierc,army_walk3 ]
{ai_walk(1);};
void() army_walk3 =[ $soldierc,army_walk4 ]
{ai_walk(1);};
void() army_walk4 =[ $soldierc,army_walk5 ]
{ai_walk(1);};
void() army_walk5 =[ $soldierc,army_walk6 ]
{ai_walk(2);};
void() army_walk6 =[ $soldierc,army_walk7 ]
{ai_walk(3);};
void() army_walk7 =[ $soldierc,army_walk8 ]
{ai_walk(4);};
void() army_walk8 =[ $soldierc,army_walk9 ]
{ai_walk(4);};
```

```
void() army_walk9 =[ $soldierc,army_walk10 ]
{ai_walk(2);};
void() army_walk10 =[ $soldierc,army_walk11 ]
{ai_walk(2);};
void() army_walk11 =[ $soldierc,army_walk12 ]
{ai_walk(2);};
void() army_walk12 =[ $soldierc,army_walk13 ]
{ai_walk(1);};
void() army_walk13 =[ $soldierc,army_walk14 ]
{ai_walk(0);};
void() army_walk14 =[ $soldierc,army_walk15 ]
{ai_walk(1);};
void() army_walk15 =[ $soldierc,army_walk16 ]
{ai_walk(1);};
void() army_walk16 =[ $soldierc,army_walk17 ]
{ai_walk(1);};
void() army_walk17 =[ $soldierc,army_walk18 ]
{ai_walk(3);};
void() army_walk18 =[ $soldierc,army_walk19 ]
{ai_walk(3);};
void() army_walk19 =[ $soldierc,army_walk20 ]
{ai_walk(3);};
void() army_walk20 =[ $soldierc,army_walk21 ]
{ai_walk(3);};
void() army_walk21 =[ $soldierc,army_walk22 ]
{ai_walk(2);};
void() army_walk22 =[ $soldierc,army_walk23 ]
{ai_walk(1);};
void() army_walk23 =[ $soldierc,army_walk24 ]
{ai_walk(1);};
void() army_walk24 =[ $soldierc,army_walk1 ]
{ai_walk(1);};

void() army_run1 =[ $soldierc,army_run2 ] {
if (random() < 0.2)
sound (self, CHAN_VOICE, "soldier/idle.wav", 1, ATTN_IDLE);
ai_run(11);};
void() army_run2 =[ $soldierc,army_run3 ]
{ai_run(15);};
void() army_run3 =[ $soldierc,army_run4 ]
{ai_run(10);};
void() army_run4 =[ $soldierc,army_run5 ]
```

```
{ai_run(10);};
void() army_run5 =[ $soldierc,army_run6 ]
{ai_run(8);};
void() army_run6 =[ $soldierc,army_run7 ]
{ai_run(15);};
void() army_run7 =[ $soldierc,army_run8 ]
{ai_run(10);};
void() army_run8 =[ $soldierc,army_run1 ]
{ai_run(8);};

void() army_atk1 =[ $soldierc,army_atk2 ] {ai_face();};
void() army_atk2 =[ $soldierc,army_atk3 ] {ai_face();};
void() army_atk3 =[ $soldierc,army_atk4 ] {ai_face();};
void() army_atk4 =[ $soldierc,army_atk5 ] {ai_face();};
void() army_atk5 =[ $soldierc,army_atk6 ]
{ai_face();army_fire();

};
void() army_atk6 =[ $soldierc,army_atk7 ] {ai_face();};
void() army_atk7 =[ $soldierc,army_atk8 ]
{ai_face();SUB_CheckRefire (army_atk1);};
void() army_atk8 =[ $soldierc,army_atk9 ] {ai_face();};
void() army_atk9 =[ $soldierc,army_run1 ] {ai_face();};


void() army_pain1 =[ $soldierc,army_pain2 ] {};
void() army_pain2 =[ $soldierc,army_pain3 ] {};
void() army_pain3 =[ $soldierc,army_pain4 ] {};
void() army_pain4 =[ $soldierc,army_pain5 ] {};
void() army_pain5 =[ $soldierc,army_pain6 ] {};
void() army_pain6 =[ $soldierc,army_run1 ]
{ai_pain(1);};

void() army_painb1 =[ $soldierc,army_painb2 ] {};
void() army_painb2 =[ $soldierc,army_painb3 ]
{ai_painforward(13);};
void() army_painb3 =[ $soldierc,army_painb4 ]
{ai_painforward(9);};
void() army_painb4 =[ $soldierc,army_painb5 ] {};
void() army_painb5 =[ $soldierc,army_painb6 ] {};
void() army_painb6 =[ $soldierc,army_painb7 ] {};
void() army_painb7 =[ $soldierc,army_painb8 ] {};
```

```
void() army_painb8 =[ $soldierc,army_painb9 ] {};
void() army_painb9 =[ $soldierc,army_painb10] {};
void() army_painb10=[ $soldierc,army_painb11] {};
void() army_painb11=[ $soldierc,army_painb12] {};
void() army_painb12=[ $soldierc,army_painb13] {ai_pain(2);};
void() army_painb13=[ $soldierc,army_painb14] {};
void() army_painb14=[ $soldierc,army_run1 ] {};

void() army_painc1 =[ $soldierc,army_painc2 ] {};
void() army_painc2 =[ $soldierc,army_painc3 ]
{ai_pain(1);};
void() army_painc3 =[ $soldierc,army_painc4 ] {};
void() army_painc4 =[ $soldierc,army_painc5 ] {};
void() army_painc5 =[ $soldierc,army_painc6 ]
{ai_painforward(1);};
void() army_painc6 =[ $soldierc,army_painc7 ]
{ai_painforward(1);};
void() army_painc7 =[ $soldierc,army_painc8 ] {};
void() army_painc8 =[ $soldierc,army_painc9 ]
{ai_pain(1);};
void() army_painc9 =[ $soldierc,army_painc10]
{ai_painforward(4);};
void() army_painc10=[ $soldierc,army_painc11] {ai_painforward(3);};
void() army_painc11=[ $soldierc,army_painc12] {ai_painforward(6);};
void() army_painc12=[ $soldierc,army_painc13] {ai_painforward(8);};
void() army_painc13=[ $soldierc,army_run1] {};

void(entity attacker, float damage) army_pain =
{
local float r;

if (self.pain_finished > time)
return;

r = random();

if (r < 0.2)
{
self.pain_finished = time + 0.6;
army_pain1 ();
sound (self, CHAN_VOICE, "soldier/pain1.wav", 1, ATTN_NORM);
}
```

```
else if (r < 0.6)
{
self.pain_finished = time + 1.1;
army_painb1 ();
sound (self, CHAN_VOICE, "soldier/pain2.wav", 1, ATTN_NORM);
}
else
{
self.pain_finished = time + 1.1;
army_painc1 ();
sound (self, CHAN_VOICE, "soldier/pain2.wav", 1, ATTN_NORM);
}
};


void() army_fire =
{
local vector dir;
local entity en;

ai_face();

sound (self, CHAN_WEAPON, "soldier/sattck1.wav", 1, ATTN_NORM);

// dodging player
en = self.enemy;

dir = en.origin - en.velocity*0.2;
dir = normalize (dir - self.origin);

FireBullets (4, dir, '0.1 0.1 0');
};



void() army_die1 =[ $soldierd,army_die2 ] {};
void() army_die2 =[ $soldierd,army_die3 ] {};
void() army_die3 =[ $soldierd,army_die4 ]
{self.solid = SOLID_NOT;self.ammo_shells = 5;DropBackpack();};
void() army_die4 =[ $soldierd,army_die5 ] {};
void() army_die5 =[ $soldierd,army_die6 ] {};
void() army_die6 =[ $soldierd,army_die7 ] {};
```

```
void() army_die7 =[ $soldierd,army_die8 ] {};
void() army_die8 =[ $soldierd,army_die9 ] {};
void() army_die9 =[ $soldierd,army_die10 ] {};
void() army_die10 =[ $soldierd,army_die10 ] {};

void() army_cdie1 =[ $soldierd,army_cdie2 ] {};
void() army_cdie2 =[ $soldierd,army_cdie3 ]
{ai_back(5);};
void() army_cdie3 =[ $soldierd,army_cdie4 ]
{self.solid = SOLID_NOT;self.ammo_shells = 5;DropBackpack();ai_back(4);};
void() army_cdie4 =[ $soldierd,army_cdie5 ]
{ai_back(13);};
void() army_cdie5 =[ $soldierd,army_cdie6 ]
{ai_back(3);};
void() army_cdie6 =[ $soldierd,army_cdie7 ]
{ai_back(4);};
void() army_cdie7 =[ $soldierd,army_cdie8 ] {};
void() army_cdie8 =[ $soldierd,army_cdie9 ] {};
void() army_cdie9 =[ $soldierd,army_cdie10 ] {};
void() army_cdie10 =[ $soldierd,army_cdie11 ] {};
void() army_cdie11 =[ $soldierd,army_cdie11 ] {};


void() army_die =
{
// check for gib
if (self.health < -35)
{
sound (self, CHAN_VOICE, "player/udeath.wav", 1, ATTN_NORM);
ThrowHead ("progs/h_guard.mdl", self.health);
ThrowGib ("progs/gib1.mdl", self.health);
ThrowGib ("progs/gib2.mdl", self.health);
ThrowGib ("progs/gib3.mdl", self.health);
return;
}

// regular death
sound (self, CHAN_VOICE, "soldier/death1.wav", 1, ATTN_NORM);
if (random() < 0.5)
army_die1 ();
else
army_cdie1 ();
```

```
};


/*UG monster_army (1 0 0) (-16 -16 -24) (16 16 40) Ambush
*/
void() monster_army =
{
if (deathmatch)
{
remove(self);
return;
}
precache_model ("progs/soldier.mdl");
precache_model ("progs/h_guard.mdl");
precache_model ("progs/gib1.mdl");
precache_model ("progs/gib2.mdl");
precache_model ("progs/gib3.mdl");

precache_sound ("soldier/death1.wav");
precache_sound ("soldier/idle.wav");
precache_sound ("soldier/pain1.wav");
precache_sound ("soldier/pain2.wav");
precache_sound ("soldier/sattck1.wav");
precache_sound ("soldier/sight1.wav");

precache_sound ("player/udeath.wav"); // gib death


self.solid = SOLID_SLIDEBOX;
self.movetype = MOVETYPE_STEP;

setmodel (self, "progs/soldier.mdl");

setsize (self, '-16 -16 -24', '16 16 40');
self.health = 30;

self.th_stand = army_stand1;
self.th_walk = army_walk1;
self.th_run = army_run1;
self.th_missile = army_atk1;
self.th_pain = army_pain;
self.th_die = army_die;
```

```
walkmonster_start ();
};
```

Jodi's text is, as you may agree or not, an impressive piece of concrete poetry. Military order has inscribed the grammatical and typographical structure of the text. So it reads like the coded inventory of a weapon's arsenal, like a secret strategic plan, or like a military code language gone autistic.

Compare this to a classical concrete poetry "constellation" of Eugen Gomringer, one of the "three variations to 'no error in the system"' from 1969 (in my translation of the German original):

no error in the system
no reror in the system
no rreor in the system
no rroer in the system
no rrore in the system
no rror ein the system
no rror ien the system
no rror ine the system
no rror in ethe system
no rror in tehe system
no rror in thee system
no rror in thee system
no rror in the esystem
no rror in the seystem
no rror in the syestem
no rror in the sysetem
no rror in the systeem
no rror in the systeem
no rror in the systeme
eno rror in the system
neo rror in the system
noe rror in the system
no error in the system

Gomringer's poem could as well be called a programmed text as well. The error as embodied by the letter "e" shifts one position to the right in every next line, thus permuting the text, i.e. shuffling the order of its letters according to a precise algorithm. This algorithm could as well be implemented as a computer program, and indeed Gomringer generated some of his poems with computers still in the early 1970s.

Perhaps you agree with me that jodi's text is better than Gomringer's. It is more complex, more clever and more interesting both as a typographic constellation and as a reflection on language and systems. As some of you may have recognized, jodi's text is — just like the Algol and Perl poems of Le Lionnais and Harwood — a software sourcecode, this time written in the C programming language. While the sourcecode of Gomringer's algorithm is implicit and must be reconstructed by those who read the poem output (which makes it, in my opinion, a tedious and redundant read once you have got its idea) Jodi's and Graham Harwood's texts are interesting both as poem output and program sourcecode, as phenotext (i.e. the text generated) and genotext (the score for the generation of text).

The executable machine code generated from jodi's C sourcecode is, as a matter of fact, their software piece "untitled game" which in itself is a modification of the popular ego shooter game "Quake". In other words: Jodi's concrete poetry is an objet trouvé, a ready-made from program code. Since jodi's obscure the origin and the function of this code in their mailing list posting, the code of Quake/untitled game for the first time becomes readable in its aesthetic and political subtexts.

Conclusion. What are the conclusions to be drawn from this?

- computer code can artistic material like any other. It by no means is "machine code" in the sense that it would be code not written by humans and meant only to trigger machine processes.
- art and poetry using computer code as their material address human imagination, not machines even if they pretend to do the opposite. (A pretension which itself is an artistic device.) The sourcecode poems of Quirinus Kuhlmann, Graham Harwood and jodi are outstanding examples for such imaginative and even fantastic code, and Gomringer's poem is so poor in comparison because it is intentionally deprived of imagination. (There is no aesthetics of the machine itself — contrary to the beliefs of techno-determinist media theory —, but a cultural imagination invested into the machines which artists can not only play with, but also participate in.)
- Poetry is at once expansion and restraint of media. Since it exists (i.e. since Archilochos and Sappho), it simultaneously involves sound (through phonetics), visuals (through typography), semantics (through the referential function of language), and — let me add this — calculus (numerical structures and formal instruction coded into language, and be it only in the numerical equivalence of a meter, a rhyme scheme and arrangement of stanzas).

Calculus has always been part of text and poetry, although it tends to be overlooked and is more obscure than phonetics, semantics and even typography. (Jacques Derrida reflected all this already in 1967 in his book “On Grammatology”)

- Plagiarism and intertextuality could be seen as “weak” form of poetic calculus because it means to consider work as sourcecode and transform it. Open source/Free Software is the historical principle of mathematics. Historically, there has never been copyright on mathematical equations and proofs. The reinvigoration of calculus in poetry via the computer could also lead to a reinvigoration of anti-copyright aesthetics. (So it is no coincidence that Graham Harwood organized the Festival of Plagiarism, that jodi’s code is plagiarism in itself and that several people here at this conference have historical ties with plagiarist subcultures, like Miekal And.)
- Computer poetry gives new importance to poetic calculus. A computer is not a medium itself; it is a calculating device capable of triggering analog media. Computer poetry and digital art does not enrich poetry in regards to its media. Visuality for example is more opulent in books, paintings or sculptural objects than on screens, sound poetry is richer when performed by human speakers than by machines as well. What computers instead allow is a more complex coupling of media, which is an effect of mathematical computation/calculus.
- Codeworks show that digital poetry might also be about media-independence, since code can take any media form. This, in fact, is the historical virtue of literature in general which has been proliferated since several thousand years through several old and new media. Sappho’s poetry could be printed as soon as printing was invented, it could be sound-recorded as soon as the grammophone was invented, etc.; so literature in itself is both multimedia *and* media-agnostic.

I think ”new media poetry“ in contrast has the disadvantage of limiting and restraining the multitude of media channels available for literature, because it is tied to one particular output technology, maybe even one particular piece of output software or browser plugin. But apart from that personal sidenote, I think Oulipo provides a good model for digital poetics where it thinks in the terms of “constraints”, not infinite openness, to make poetic imagination possible. Thus, we could talk about “new media poetry” as a formal constraint instead of fuzzy and pointless expansion.