CDMTCS
Research
Report
Series

Solving SAT with Bilateral Computing

Joshua J. Arulanandham
Cristian S. Calude
Michael J. Dinneen
Department of Computer Science
University of Auckland
Auckland, New Zealand

CDMTCS-199
December 2002

Centre for Discrete Mathematics and
Theoretical Computer Science

Solving SAT with Bilateral Computing

Joshua J. Arulanandham, Cristian S. Calude, Michael J. Dinneen
Department of Computer Science
The University of Auckland
Auckland, New Zealand
hi_josh@hotmail.com, {cristian,mjd}@cs.auckland.ac.nz

Abstract

We solve a simple instance of the SAT problem using a natural physical computing system based on fluid mechanics. The natural system functions in a way that avoids the combinatorial explosion which generally arises from the exponential number of assignments to be examined. The solution may be viewed as part of a more general type of natural computation called *Bilateral Computing*. The paper will also describe this new computing paradigm and will compare it with Reversible Computing. Our approach is informal: the emphasis is on motivation, ideas and implementation rather than a formal description.

1 Introduction

The *Satisfiability* problem for Boolean formulas, commonly known as the *SAT problem*, is one of the classic "hard problems" in computer science. Due to its NP–completeness, any other hard problem (in NP) can be reduced in polynomial time to SAT. Naturally, an efficient solution for SAT will translate into an efficient solution for every NP problem.

The SAT problem is defined as follows: Given a Boolean formula with n variables, determine whether it can be satisfied (the value of the formula made *true*) by a truth assignment, i.e., by (at least) one possible assignment of the logical values *true* or *false* to each variable. For instance, with the plus symbol denoting a 'logical or' and juxtaposition (implied multiplication) denoting a 'logical and', the formula $(a + c + \bar{e})(\bar{b})(b + c + d + e)(\bar{d} + e)$ is satisfiable, with the assignment: $a = \mathit{false}$, $b = \mathit{false}$, $c = \mathit{true}$, $d = \mathit{false}$ and $e = \mathit{false}$. On the other hand, the formula $(a)(\bar{a} + b)(\bar{a} + \bar{c})(\bar{b} + c)$ is not satisfiable, as there is no possible assignment to a, b and c that can make the formula *true*. (See [5] for a detailed coverage of the topic.)

What makes such an "easy question" hard to answer is that often one has to (exhaustively) test all possible 2^n truth assignments to the n variables in a formula. Naturally, the exponential number of the possible assignments leads to a combinatorial explosion as n gets bigger, and hence the hardness.

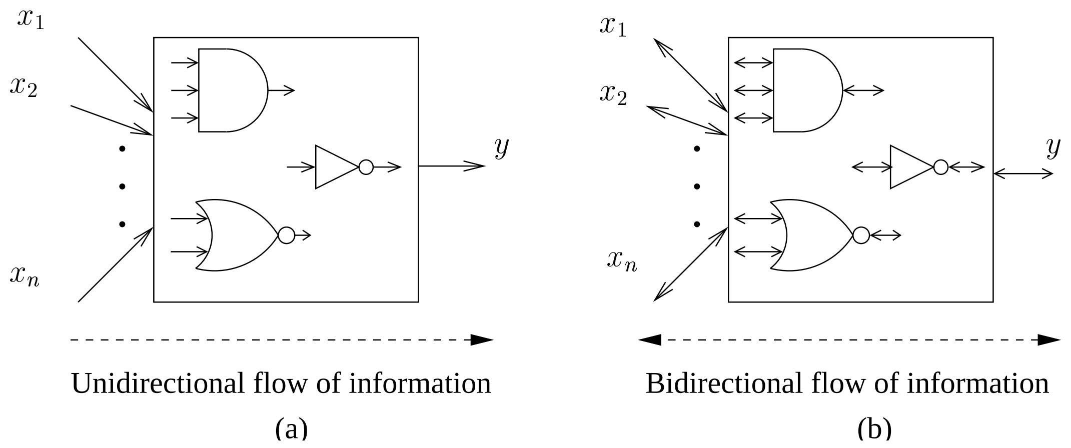
x_1
x_2
x_n
y
Unidirectional flow of information
(a)
x_1
x_2
x_n
y
Bidirectional flow of information
(b)

Figure 1: Unilateral and bilateral logic circuits.

We suggest a solution for SAT that uses a natural physical computing system based on fluid mechanics. The natural system called a "Bilateral Computing System"[1] that we use to solve SAT avoids the combinatorial explosion which generally arises from the exponential number of assignments to be examined. In this paper we start with an informal presentation of the notion of Bilateral Computing in Section 2, discuss the design of "bilateral logic gates" in Section 3 and then proceed to the complete design of the (bilateral) natural computing system for solving SAT in Section 4. We analyze the time complexity in Section 5, give a short description of Bilateral Computing in Section 6, compare Reversible Computing with Bilateral Computing in Section 7 and summarize our results in Section 8.

2 Bilateral Computing

We will describe in detail the notion of a *Bilateral Computing System* (BCS) later in Section 6. Simply put, a BCS is a computing system/circuit that allows a bidirectional flow of information between its input and output "points" or "pins"[2]. Figure 1 illustrates this idea in the context of solving the SAT problem.

Figure 1(a) is the normal unilateral type of a Boolean circuit that can generate an output based on the inputs $x_1, \ldots, x_n$. Figure 1(b) is a hypothetical, bilateral Boolean circuit that can process information in both directions, i.e., from the input to the output and conversely, from the output to the input. For instance, given an 1–signal (*true* input) through its output pin y, the circuit would automatically assume a set of values for the input pins $x_1, \ldots, x_n$ that are consistent with the output $y = 1$. In other words, the input pins would be "forced" to assume one of those possible combinations of 0's and 1's (if any), that could generate the output $y = 1$. The kind of action that the circuit would

[1]See [9, 8, 6, 4, 1, 7] for other variants of Natural Computing.

[2]We use the word "bilateral" as in electrical circuit theory: circuit elements (e.g. resistors) that can conduct current equally well in either direction are bilateral — as opposed to those (like diodes, which are unilateral) that cannot.

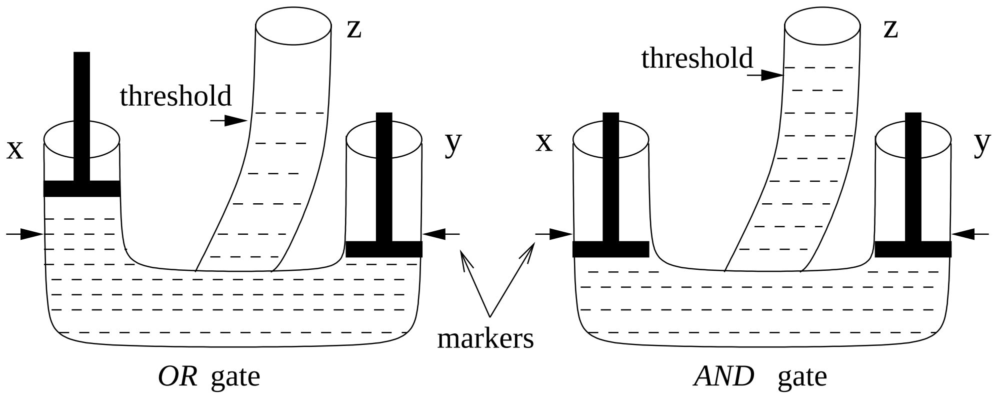

Figure 2: Fluid mechanical logic gates.

take, when there is no such possible input assignment to the inputs $x_1, \ldots, x_n$ (i.e., the formula is unsatisfiable) depends on how exactly a BCS is physically realized.

In what follows we will discuss the design of logic gates that behave as BCSs, by virtue of simple natural laws.

3 Bilateral AND, OR Gates

In this section we describe a design of bilateral logic gates using fluid mechanical systems. Figure 2 shows fluid mechanical designs for the 2–input *AND* and *OR* gates. Each gate consists of three limbs joined together and a working fluid that can move freely within. Pistons are fitted to the two limbs (x and y) that represent the inputs of the gate; they can be pushed in, at will, say by giving a gentle pressure on them or by adding weights.

A "push" on a piston represents an 1–input (logical true); a "no-push" (piston in raised position) is a 0–input (logical false). (See Figure 2; note that there are "markers" on the limbs; a "push" input should move the piston past the "marker" to signal an 1–input.) The output is the fluid level in the third limb (z): a raised fluid level above certain threshold represents 1 and a lower level corresponds to 0. (There is a "marker" in the limb that denotes the threshold level.) In the *OR* gate, the threshold (for output fluid level) is adjusted in such a way that, a "push" on just one of the two inputs would cause the level go past the threshold. In the *AND* gate, it is just the opposite: the threshold is set higher, so as to require both pistons to be pushed simultaneously. Note that each "push" is a predefined, fixed measure and cannot exceed a certain limit.

Why do we call these gates bilateral? The working fluid can be moved *forwards* and *backwards*, and the input pistons can be moved up and down not just by direct manual handling, but also indirectly by raising the fluid level in the output limb. Imagine sucking the fluid level in the output limb up; this would indirectly affect the position of the input pistons, as well. For instance, raising the fluid level in the output limb is the equivalent of pushing (both) the input pistons down[3]. Thus, certain (allowable) input settings of the

[3]Both input pistons are subjected to the same atmospheric pressure.

gate can be indirectly obtained by just maneuvering the output, in the reverse direction and hence, the gates are bilateral.

(a)

x	y	$z = x + y$
0	0	0
0	1	1
1	0	1
1	1	1

(b)

z	x	y
0	0	0
1	1	1

Table 1: Truth table for an individual bilateral *OR* gate.

(a)

x	y	$z = xy$
0	0	0
0	1	0
1	0	0
1	1	1

(b)

z	x	y
0	0	0
1	1	1

Table 2: Truth table for an individual bilateral *AND* gate.

Tables 1(a) and 1(b), respectively, show Truth tables for the operation of the *OR* gate when it is operated *independently* (and not coupled with other bilateral gates), both in the normal, forward direction (input $\rightarrow$ output) and in the reverse direction (output $\rightarrow$ input). Tables 2(a) and 2(b) represent truth tables for the operation of the *AND* gate, both in forward and reverse directions. Note that these truth tables, which show one fixed rule by which the output is mapped to an input combination, do not apply when the gates work as part of a coupled system of bilateral gates. In that case, during the backward operation, there is no unique, fixed rule by which the output is mapped to an input combination. For instance, the bilateral *OR* gate could, by definition, map an 1–output to either one of (1, 1) or (1, 0) or (0, 1) input combinations depending on the constraints by the other gates of the realized BCS.

The design of a *NOT* gate is quite straightforward and will be discussed in the next section.

4 Solving an Instance of SAT

Given the simple Boolean formula $(\bar{a} + b)(a + b)$ whose truth table is given in Table 3, we assemble fluid mechanical gates together to form a BCS, as illustrated in Figure 3. The *OR* gates (gates labeled 1 and 2 in Figure 3) realize $(\bar{a} + b)$ and $(a + b)$ respectively and the *AND* gate (gate 3) combines their outputs.

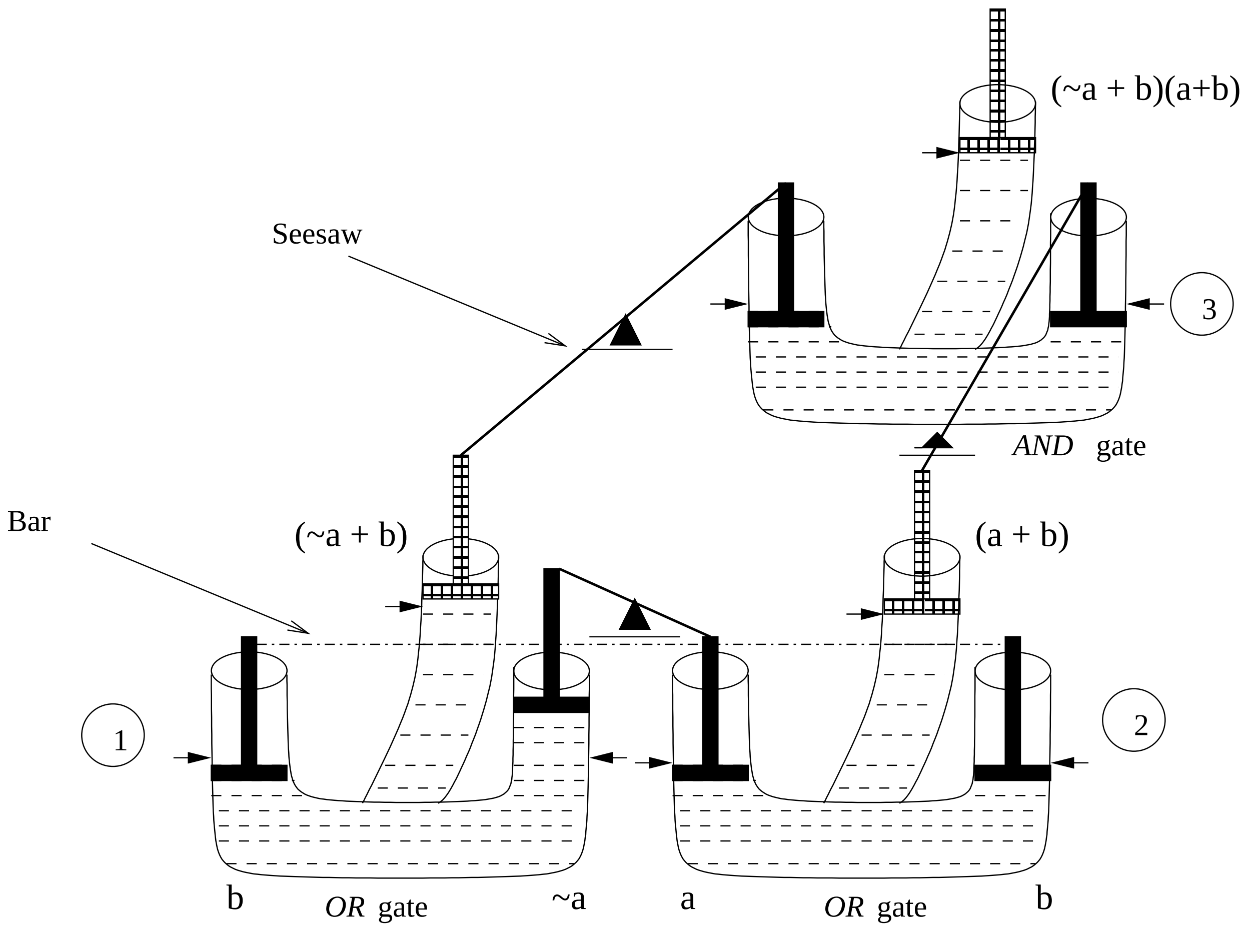

Figure 3: An instance of SAT.

a	b	$(\bar{a}+b)(a+b)$
0	0	0
1	0	0
0	1	1
1	1	1

Table 3: Truth table for $(\bar{a}+b)(a+b)$.

Pistons on the input/output sides of each gate are called input pistons and output pistons. There is a seesaw–like structure[4] connecting the input pistons of gates 1 and 2, which take the inputs $\bar{a}$ and a; it ensures that the input values (push and no-push) assigned to the pistons are complements of each other. Also, observe the bar–like structure connecting the other two pistons representing the two inputs labeled b; it ensures that the values assigned to the pistons are the same. Both these structures ensure consistency in the assignment of values to input variables. Note the long seesaw structures connecting the outputs of the *OR* gates (gates 1 and 2 in Figure 3) with input pistons of the *AND* gate (gate 3); they are the mechanisms that convert the outputs of gates 1 and 2 (indicated by fluid levels) to equivalent push/no-push inputs, which are subsequently applied to the *AND* gate. For instance, suppose, the output of gate 1 is 1 — indicated by a raised fluid level in the output limb (and thus, a raised output piston), the mechanism will, in turn, produce a push input (1–input) to the *AND* gate.

We briefly explain the overall behavior of the setup. It is easy to see that the input pistons connected to the *OR* gates (gates 1 and 2) can be set (say, manually) in one of the two satisfiable input configurations (recall Table 3 for its satisfiable assignments); and the output of the *AND* gate will be 1 in such cases, which is indicated by the fluid level raising above a threshold.

Now, consider the reverse case: suppose, the output of the *AND* gate (gate 3 of Figure 3) is assigned 1, i.e. by forcibly raising the fluid level up, say, by means of suction. This, as a result, would exert a force on the other pistons (through the working fluid), and would set them all "in motion" and would eventually place the pistons (automatically) in one of the (two) satisfiable configurations. (One of the two satisfiable piston configurations is shown in Figure 3; if the seesaw structure connecting gates 1 and 2 is reversed, i.e. a up and $\bar{a}$ down, we get the other possible configuration.[5]) We could thus make the information flow in the reverse direction.

The above described effect will evolve in a sequence, each event triggering the next:

(a) Fluid–level in output limb of gate 3 — forcibly raised.
(b) Input pistons of gate 3 — pushed down.
(c) Output pistons of gates 1 and 2 — raised.

[4]This is an implementation of the bilateral *NOT* gate.

[5]Initially, there might be a state of "oscillation", when pistons move up and down, trying to settle down in one of the (two) possible configurations; but, eventually they will reach a *state of equilibrium* which cannot be one of the non–satisfying configurations, due to the physical arrangement. However, the device might arrive at a different satisfiable configuration each time we run it, due to slight changes in the initial condition.

(d) Input pistons of gates 1 and 2 — some pushed down, some raised (depending on the constraints).

Now, what if the formula chosen is *not* satisfiable? What if there is no allowable input piston configuration leading to an 1–output? In that case, trying to raise the fluid level in the output limb would push the system to an abnormal state, say, a noticeable rise in pressure. This should be detected and suitably dealt with, say, by releasing a safety valve or a cork (which would signal the unsatisfiability of the formula).

To make the device workable, the following assumptions are adopted:

1. *The seesaw structure is perfect.* It *always* keeps the pistons on either side in complementary states — one up and one down and never allows them to assume the same position.
2. *The push inputs given to the gates undergo "squashing".* The push inputs to the *OR* and *AND* gates cannot exceed a certain predefined limit and will be duly "squashed" if they exceed it.
3. *The piston is moved slowly.* Only then, the process can be treated as *adiabatic*, when no heat is gained or lost. (This will simplify the time complexity analysis.)

As the working fluid can be moved forwards and backwards, the system acts as a BCS. When the fluid level of the output limb (of gate 3) is raised forcibly by suction or other means, this will exert a force on the other pistons (through the working fluid), would set them in motion and would eventually place the pistons (automatically) in one of the (two) satisfiable configurations — the two different possible positions of the seesaw structure connecting gates 1 and 2. Note that the suction force (applied at the output limb) will naturally try to push down as many (input) pistons in every gate, as is allowed by the built–in constraints in the hardware representing the formula.

5 Time Complexity

Given a CNF formula we can (uniformly) realize a general SAT device by having a tree of $k-1$ binary *OR* gates for each clause of k literals. and then 'logically and' together m clauses with another tree of $m-1$ binary *AND* gates. We can also consider a general SAT device in which the binary gates (of Figure 3) are replaced with m–ary gates[6]. Thus, the clauses of a CNF formula can be represented using *OR* gates whose outputs are combined by a single *AND* gate. Also, corresponding to each variable in the formula, say a, we assume that there is a generic seesaw structure that connects all a's onto one side and all $\bar{a}$'s on the opposite side.

In terms of a physical realization, the response time t (after suction is applied to the output gate) is the sum of two components: t_1, the time taken by the sound wave (generated by suction) to reach the farthest piston[7] and t_2, the time taken by the pistons

[6]An m–ary *OR* gate can be constructed by adjusting the threshold (for output fluid level) in such a way that, a "push" on just one of the m inputs would cause the level go past the threshold. In the *AND* gate, the threshold is set much higher, so as to require all m pistons to be pushed simultaneously.

[7]The device is inherently parallel, and the sound waves move toward all pistons simultaneously.

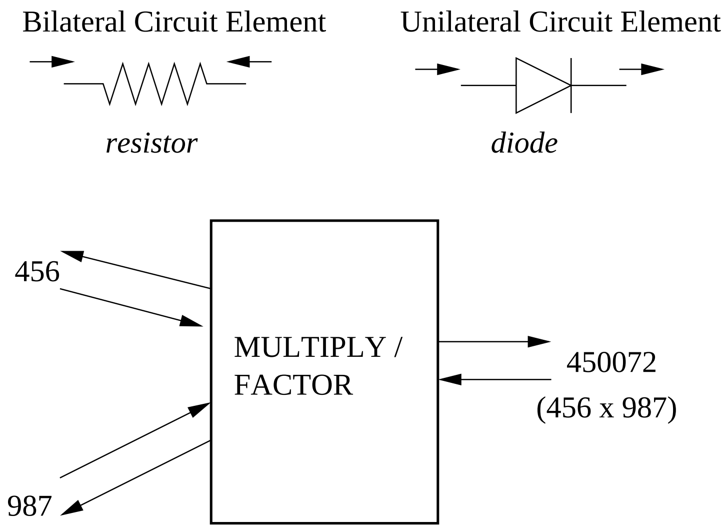

Figure 4: Bilateral computing system.

to react, due to friction. The time t_1 depends on λ, the velocity of sound in the fluid medium and the geometric dimensions of the apparatus. Thus, $t_1 = \lambda \times d$, where d is the maximum distance the wave has to travel — from the point where suction is applied to the farthest piston in the apparatus. Therefore, $t = \lambda \times d + t_2$, where λ and t_2 are constants and d grows linearly with the number of gates, the physical size of the individual gates being uniform (for binary case) or unbounded (for m–ary case).

6 Bilateral Computing: Further Remarks

In general, computing mechanisms, both conceptual and physical, are built upon rudimentary operations like *addition* and *multiplication*. These mechanisms can quickly combine data (using such operations) in a bottom–up fashion, rather than the other way round (say, spontaneously decide "102,390908 = ? × ?"). In a sense, such mechanisms suffer from a fundamental asymmetry in the way they compute and process information. For instance, while we are able to design circuits that can *multiply* quickly, we have limited success computing the other way round, i.e. *factoring*. The same is true with computing Boolean functions; we have fast digital circuits that combine digital data using *AND*, *OR* operations and yet, no fast circuits that determine, "What values should the various inputs assume, in order to make the output 1?". The *same* conventional hardware circuit that is used to *multiply*, cannot be used to *factor*, say, by making it work backwards (see Figure 4). But, it may be possible to design circuits with such an extraordinary facility. In fact, the fluid mechanical system that we discussed exhibits such a backward operation facility.

Bilateral Computing is a paradigm that seeks computational structures, both at conceptual and hardware levels, that would allow us to compute in both directions, using the *same* underlying mechanism and possibly, with the same computational effort. Such a mechanism may — in principle — *factor*, with the same ease as it would *multiply* — by simply working backwards[8]. A BCS can be seen as a black–box that can implement, with the same underlying computational structure, a bidirectional input–output mapping; it can facilitate a bidirectional flow of information between its input and output points. A BCS has no strict distinction between inputs and outputs, but, simply guarantees, for instance, that the inequality $A = B \times C$ is true. A (hypothetical) BCS realizing $A = B \times C$ would act as a multiplier if B and C are *instantiated* [9], and would output A; alternatively, it would behave like a factoring device if A is *instantiated* instead, and would output one of (possibly many) suitable values for B and C that would make the equality true[10]. There is no rigid setup, for instance, like B and C as the designated inputs and A, as the designated output.

We can mathematically characterize a BCS as follows. Consider a surjective (not necessarily bijective) function f: $X \rightarrow Y$ and the set of functions $G = \{g : Y \rightarrow X \mid f(g(y)) = y, \forall y \in Y\}$. A computing system is said to be *bilateral* if it can implement f as well as a $g \in G$, using the same inherent "mechanism" or "structure". Note that a BCS chooses one of the possible functions g "on the fly", i.e. (only) when it is being run; the particular choice of the function is enforced by the *constraints* we set-up while running the system. This choice of the function is not known when the system behavior is being *defined* — prior to the actual *operation* of the system; we can only associate a set of possible functions that achieve the reverse mapping, while defining its behavior.

For instance, consider the bilateral *OR* gate discussed in Section 3, $X = \{(0,0),(0,1),(1,0),(1,1)\}$ and $Y = \{0,1\}$. The function f represents the normal forward operation of the gate: $f(0,0) = 0$, $f(0,1) = 1$, $f(1,0) = 1$ and $f(1,1) = 1$. The reverse operation of the gate, operated independently (see Table 1(b)) is the function g: $Y \rightarrow X$ defined by $g(0) = (0,0)$, $g(1) = (1,1)$. Note that $f(g(0)) = 0$ and $f(g(1)) = 1$ and hence, g is a *partial inverse* of f. Similarly, the bilateral *AND* gate can be described by another pair of functions f and g. The BCS displayed in Figure 3 chooses one such g while in operation.

7 Reversible Computing vs. Bilateral Computing

The goal of Reversible Computing (also known as Conservative Logic; see [2]) is to conserve energy by making a computing device *reversible* in principle, i.e. one that can be made to run backwards in time. The motivation is not to *reverse* the computational process, but rather to *conserve* (energy). In Fredkin's words [3], "*. . . the (reversible) computer could, in principle, run backwards from the end of the computation back to the beginning as a form of proof that it is properly designed. There is, however, no necessity to run the whole computation backwards in order to have the system completely dissipationless.*" Thus the aim is to conserve energy — to avoid heat dissipation — a physical gain which will matter when the hardware gets shrunk to atomic levels.

On the other hand, the very motivation of Bilateral Computing is to literally operate

[8]Note that this is different from "going backwards in time".
[9]To "instantiate" is to *assign* values.
[10]The specific choice made amongst many possible values depends on the physical realization chosen.

the computing device both onwards and backwards: to compute a reverse output–input mapping, besides the normal input–output mapping. Reversible computing can be regarded as a special case of bilateral computing: in cases where a computable inverse could be defined, a reversible computer can be used in the same spirit as a BCS. Also, in most cases, the function to be computed is not bijective and making it bijective (and thereby, reversible) comes with the price of increasing the number of variables (which affects the space complexity).

8 Conclusions

We have solved a simple instance of the SAT problem using a natural physical computing system based on fluid mechanics. The described system does not result in the combinatorial explosion which generally arises from the exponential number of assignments to be examined. One of the reasons for the efficiency of the system is the following: Consider the behavior of the individual components (gates) of the system, when they operate as part of the whole system. During the backward operation, their behavior may be non–deterministic: there may not exist a unique, fixed rule by which the output is mapped on to an input. For instance, the bilateral *OR* gate could map an 1–output to either one of (1, 1) or (1, 0) or (0, 1) input combinations *depending on the constraints set-up during runtime*. In contrast to deterministic devices (as reversible logic gates) whose behavior is given by a unique, fixed *prior to operation* rule, the BCS picks, by virtue of natural laws, a "correct inverse" (which makes the system satisfiable, if possible) on the spur of the moment, while in operation.

9 Acknowledgement

We gratefully acknowledge Professor Boris Pavlov's interest in our work; he critically read initial drafts of the paper and made important suggestions, especially regarding the time complexity.

References

[1] C. S. Calude, G. Păun, Monica Tătărâm. A glimpse into natural computing, *J. Multi–Valued Logic* 7 (2001), 1–28.

[2] E. Fredkin, T. Toffoli. Conservative logic, *Int'l J. Theoret. Phys.* 21 (1982), 219–253.

[3] E. Fredkin. On the soul, Draft, 1982.

[4] A. J. Jeyasooriyan, R. Soodamani. Natural algorithms: A new paradigm for algorithm development, *Proc. 2nd International Conference on Information, Communications and Signal Processing*, CD-ROM, ICICS '99, 1999.

[5] J. L. Balcázar, J. Díaz, J. Gabarró. *Structural Complexity I*, Springer–Verlag, Berlin, 1988.

[6] G. Rozenberg. The natural computing column, *Bulletin of EATCS* 66 (1998), 99.

[7] G. Rozenberg. The nature of computation and the computation in nature, *International Colloquium on Graph Transformation and DNA Computing*, Technical University of Berlin, 14 February 2002.

[8] H. T. Siegelmann, S. Fishman. Analog computation with dynamical systems, *Physica D* 120 (1998), 214–235.

[9] T. Toffoli. What are nature's "natural" ways of computing?, *Workshop On Physics and Computation*, PHYSCOMP '92, IEEE Comp. Soc. Press (1993), 5–9.