Foreword

In one article a have discussed on how scientific inventions are born. Necessary building blocks are background information and then, blocks just click together and the invention is born. Then some thought is needed to know how to write the article, and then, there is the actual writing. In this book there are more than thirty articles, which are born during my bachelor's and master's studies in Helsinki University of Technology, years 2000-2009. Among these articles, there are some pearls, which had the last impact to me to publish this book. During that time I also completed B.Sc. (Tech.), 2006 and M. Sc. (Tech.), 2009. The theses dealt with information theory. The work continues, I am currently doing Ph. D. studies in University of Eastern Finland on Data Clustering. I hope the reader finds these articles interesting.

Joensuu, Finland 9th March, 2012

Mikko Malinen

Contents

2006

"Proving Simple Algebraic Equations With Prove! 1.0"

"Sudokutehtävän ratkaiseminen tietokoneella" (in Finnish)

"Markovin ketjut ja pokerin todennäköisyydet" (in Finnish)

"On comparison of chess programs"

2005

"Applications of Boolean and First-order Logic"

"Final Temperature of a Heated Object May Be Predicted"

"Cooling Down of Coffee"

2004

"Examples of Using Theorem Prover Otter"

"On Equality Proving of Mathematical Expressions"

2003

"Parametrization of a Bent Rod"

"Circuit Diagrams II"

"Energy of a Lateral Spring"

2002

"Value of a Variable Can Be Read From an Indicator"

"A System for Removing the Room Echo"

"A Search "Algorithm" for Finding the Sum of a Series"

2001

"A Theoretical Sine Wave Oscillator"

"A One-coil Voltage Source And Measurement Of Small Inductances"

"Moore's Law And Chess Programs"

"An Electrostatic Motor"

"A New Compression Method for Analog Signals?"

2000

"Linux c/C++ programming and Latex typesetting"

"Automaatiosovellus Linuxiin"

"Kytkentäkaavioita"

"A chess-playing computer program"

Optimal Transmission Power In Gaussian Channel With Certain Constraints

Mikko Malinen

4th July, 2009

Abstract

This paper shows that optimal transmission power in Gaussian channel is constant when total energy and transmission time are given.

1 Introduction

The capacity of a band-limited Gaussian channel can be written

$$C = W \log_2(1 + \frac{P}{N_0 W}) \tag{1}$$

bits per second, where W is bandwidth, $N_0/2$ is noise spectral density and P is power. In this paper we consider constraints where total energy and transmission time are given and we want to find $P(t)$ as a function of time and such that the amount of transmitted information is maximized.

2 Optimization of transmitted information

We can formulate the problem as a dynamical optimization problem as follows: Optimize

$$\max \int_0^{t_f} W \log_2(1 + \frac{P(t)}{N_0 W}) dt \tag{2}$$

subject to

$$\int_0^{t_f} P(t) dt = \text{constant}. \tag{3}$$

The constraint says that energy is constant. Solving the problem we need derivatives of C with respect to P:

$$\frac{dC}{dP} = W \cdot \frac{1}{1 + \frac{P}{N_0 W}} \cdot \frac{1}{N_0 W} = W \cdot \frac{1}{N_0 W + P} > 0$$

$$\frac{d^2 C}{dP^2} = \frac{-W \cdot 1}{(N_0 W + P)^2} < 0$$

To start with, we take some $P(t)$ which satisfies the constant energy constraint (see Figure 1).

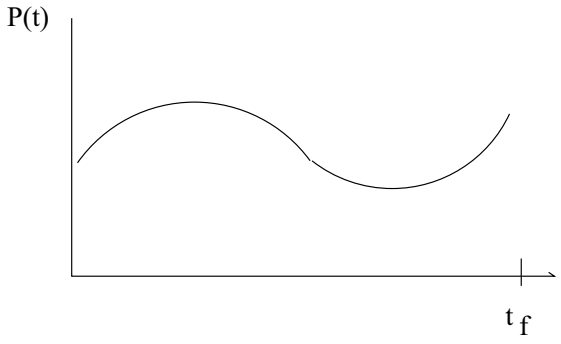


Figure 1. Initial $P(t)$.

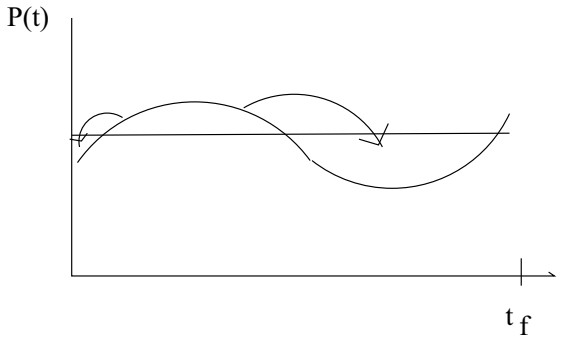


Figure 2. Averaging $P(t)$.

Since C is increasing with respect to P and $\frac{d^2C}{dP^2} < 0$ (truly concave) with respect to P we can move the volume above P_{ave} to volume below P_{ave} and obtain constant P (see Figure 2). We obtain also an increased amount of transferred information. The amount of transmitted information can not be further increased, since there is not anymore volume above P_{ave}. Constant P obtains maximum amount of information transmission in time t when the given energy constraint is satisfied.

3 Conclusion

We optimized the problem (2),(3) and obtained $P(t) =$ constant. This means that given energy and transmission time, the transmitted information is maxi-

mized in Gaussian channel when power P is constant. This may be an useful result to use in situations where we do not have much energy available for transmission but still want to transmit as much information as possible, in given time.

Rank of Variable-Length Binary Strings

Mikko Malinen

5th July, 2009

Abstract

A pseudo code for the rank of variable-length binary strings is presented.

1 Introduction

A rank function takes as an argument an object and returns a positive integer which is unique to different objects. An unrank function takes as an argument a positive integer and returns the corresponding object. These rank and unrank functions are important in combinatorial algorithms where we want for example to traverse through all different objects. In this paper we present a rank function for variable-length binary strings. There are countably infinite number of these strings as there are positive integers. The unrank function for these strings is easy to construct from the basis of the rank function. Rank and unrank functions for some other classes of objects are presented in [1].

2 Rank Function

A C-like pseudo code for rank function is presented here.

```
proc rank(string s[length], length) ≡
  if (length == 0) then return 1; fi
  value = 1; j = 1;
  for i := 1 to length do
    if (s[i − 1] == "0") then value = value ∗ (j:th prime number);
                              a : if (i < length) then
                                                    if s[i] = "0" then
                                                                    i = i + 1;
                                                                    value =
                                                                    value ∗ (j:th prime number);
                                                                    goto a;
                                                    fi
                              fi
    fi
    j = j + 1;
  od
  value = value ∗ (j:th prime number);
  return value;
end
```

3 References

[1] D. L. Kreher and D. R. Stinson, *Combinatorial Algorithms: Generation, Enumeration and Search, CRC Press, 1999*

Electromagnetic Waves Can Be Used As Data Memory

Mikko Malinen

3rd July, 2009

Abstract

This paper shows that data can be stored in electromagnetic waves, especially in radio waves. This extends the memory capacity that we have available in conventional memories, as in electric charge in conventional computers, or in quantum variables in quantum computers.

1 Introduction

In conventional computers, data is stored in a memory where several atoms are allocated for one bit. One memory location has two distinguishable states. In quantum computing, we store the data in quantum variables. The maximum number of perfectly distinguishable states - the number N, is called the *dimension* of the quantum variable. See the list of dimensions N for different quantum variables in table 1.

Variable	N
Photon polarization	2
Photon momentum	∞
Electron spin	2
Electron momentum	∞
Quark spin	2
Quark color	3
Quark momentum	∞

Table 1. Quantum variables.

In this paper we we show that electromagnetic waves, especially radio waves can be used as data memory.

2 Electromagnetic wave memory

Consider a setting in Figure 1.

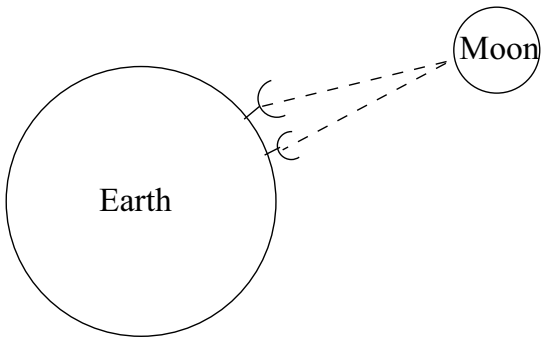


Figure 1. Sending electromagnetic waves from earth to moon.

Earth and moon could be replaced by other objects also. Transmitter sends a set of data to the moon. The moon reflects data back to receiver antenna. The received data is fed back to transmitter and sent again. The capacity of this memory is the same as the amount of data that can be sent in one round-trip-time. $2s = ct$, where s is the distance between the two objects, c is the speed of light and t is the round-trip-time. From this we can calculate the round-trip-time:

$$t = \frac{2s}{c} \tag{1}$$

As we know, the capacity of a band-limited Gaussian channel is

$$C = W \log_2(1 + \frac{P}{N_0 W}) \tag{2}$$

bits per second where W is bandwidth, P is power and $\frac{N_0}{2}$ is noise spectral density. Combining (1) and (2) we get the upper bound for the capacity S of this memory:

$$S < Ct = \frac{2s}{c} \cdot W \log_2(1 + \frac{P}{N_0 W})$$

3 Conclusions

We showed how electromagnetic waves can be used to store data. This is an addition to the conventional way to store data, electric charge and more modern quantum variables. This shows, that there is memory capacity also outside these memories.

On Distances of Distributions

Mikko Malinen

2nd July, 2009

Abstract

This paper presents relative entropy, the most common measure of the "distance" between two probability mass functions. However, relative entropy lacks some properties of a true metric: Relative entropy is not symmetric nor does it satisfy the triangle equality. This paper proposes several alternatives for relative entropy. Some of these are symmetric or satisfy the triangle equality.

1 Introduction

Relative entropy or *Kullback-Leibler distance* is a measure of a "distance" between two probability mass functions p and q. It is defined as

$$D(p||q) = \sum_x p(x) \log_2 \frac{p(x)}{q(x)}.$$

Relative entropy is not a true metric, but it has some properties of a true metric. In particular, it is always non-negative and is zero if and only if $p = q$. However, it is not symmetric nor does it satisfy the triangle equality.

2 Proposals for a measure of the "distance"

In this section, we propose several alternatives for relative entropy.

Proposal 1 $D_a = D(p||q) + D(q||p)$

D_a is symmetric.

Proposal 2 *($\mathcal{L}_1$ norm) p and q two probability mass functions on $\mathcal{H}$.* $D_b = \|p - q\|_1 = \sum_{x \in \mathcal{H}} |p(x) - q(x)|$.

D_b is symmetric. $D \geq 0$, equality when $p = q$. Here we present a lemma [1], which relates relative entropy and D_b:

Lemma 1 $D(p_1 \| p_2) \geq \frac{1}{2 \ln 2} \|p_1 - p_2\|_1^2$.

Proposal 3 $D_c = \frac{\|p-q\|_1}{2} = \sum_{x\in\mathcal{H}} |p(x) - q(x)|/2.$

Properties are same as was written for proposal 2, and in addition $0 \leq D_c \leq 1$, $D_c = 1$ when $\forall x(p(x) \geq 0 \Rightarrow q(x) = 0)$.

Proposal 4 $D_d = \int_{-\infty}^{\infty} |\int_{-\infty}^{x} f(t)dt - \int_{-\infty}^{x} g(t)dt|dx.$

This is for two continuous distributions f and g. D_d is symmetric. $0 \leq D < \infty$, $D = 0$ when $\forall x f(x) = g(x)$.

Proposal 5 $D_e = \int_{-\infty}^{\infty} (\int_{-\infty}^{x} f(t)dt - \int_{-\infty}^{x} g(t)dt)dx.$

D_e is not anymore symmetric, but the triangle equality is satisfied. $-\infty < D_e < \infty$. $D_e = 0$ when mean values of distributions f and g are equal.

3 Conclusion

We proposed several alternatives for relative entropy. These alternatives still lack some properties of a true metric, but some proposals are symmetric or satisfy the triangle equality. This may sometimes be advantageous.

4 References

[1] T. M. Cover and J. A. Thomas, *Elements of Information Theory*, John Wiley & Sons, USA, 1991.

Fountain Codes and Invertible Matrices

Mikko Malinen, *Student Member, IEEE.*

***Abstract*—This paper deals with Fountain codes, and especially with their encoding matrices, which are required here to be invertible. A result is stated that an encoding matrix induces a permutation. Also, a result is that encoding matrices form a group with multiplication operation. An encoding is a transformation, which reduces the entropy of an initially high-entropy input vector. A special encoding matrix, with which the entropy reduction is more effective than with matrices created by the Ideal Soliton distribution is formed. Experimental results with entropy reduction are shown.**

***Index Terms*—Entropy-reducing transformation, Fountain codes, group, Ideal Soliton distribution, permutation.**

I. Introduction

FOUNTAIN codes were first mentioned in [1]. LT-codes [3] are first practical Fountain codes. In Fountain codes we have k input symbols and n output symbols. We call an encoding graph a bipartite graph where on the other side are the input symbols and on the other side the output symbols. There are edges which mark the connections between the input symbols and the output symbols. We call the degree of an output symbol the number of input symbols the output symbol is connected to. The degree distribution $\rho(i)$ tells the probability that an output symbol has i connections. We call an encoding matrix R a $k \times k$ matrix where an element r_{lm} is 1 if the m:th input symbol affects (has a connection) to the l:th output symbol and 0 if it has not. Generally, R may have rank $< k$, but here we restrict the treatment to matrices which have full rank, i.e. all the rows are linearly independent. These matrices are invertible. This way the number of output symbols n equals the number of input symbols k. Fountain codes for which the matrix R is of full rank are always decodable.

The output bits are calculated by

$$y = Rx,$$

where y is output bits in vector form, x is input bits in vector form and the multiplication is done modulo 2 as is the idea in Fountain coding. The decoding is done by

$$x = R^{-1}y.$$

II. Encoding matrix induces a permutation

Our first result (Result 1) is that when there are two different inputs $x^{(1)}$ and $x^{(2)}$, the two outputs $y^{(1)}$ and $y^{(2)}$ are always different. This is due to the fact that when decoding $y^{(i)}$ we end up to an unique $x^{(i)}$.

From the Result 1 follows the next result: By multiplying an input vector x several times by the encoding matrix, we end up back to x at some point. Each multiplication results to different output until we end up to x. Depending on the choice of the initial input we end up to different cycles. Thus, in principle, we could make decoding of an output by multiplying the output, length of the cycle - 1 times. Of course, we have to know the length of the cycle. If we number the different length k vectors by $1, 2, ..., 2^k$ we can say that an encoding matrix R induces an unique permutation. We can use the list presentation of a permutation [2], pp. 52-64, to express the permutation. From the list presentation can be seen that there are $s!$ different possible permutations of s elements. This could be used as the upper bound for the number of different encoding matrices. However, it turns out that $2^k!$ (we have 2^k elements) grows faster than the total number of different 0-1 matrices $2^{(k^2)}$ (in a $k \times k$ matrix there are k^2 elements). Thus this upper bound is practically useless. We may also conclude that invertible 0-1 matrices of size $k \times k$ do not induce all possible permutations of size 2^k.

III. Encoding matrices form a group

One result is that encoding matrices (invertible 0-1 matrices) form a group with modulo 2 multiplication operation. Namely, two such matrices multiplied is also such a matrix. There is a unit element, a unit matrix. Also, the inverse element is the inverse matrix. And the multiplication is associative.

IV. Entropy-reducing transformations

Next, we come to entropy considerations. We state a result, that when all different input vectors are multiplied by an encoding matrix, the entropy remains the same on average. This is due to the fact that if the entropy would change, we could compress the input vector below what the initial entropy indicates. Even if the entropy increased on the average, we could reduce the entropy by using the inverted matrix. However, according to our experimental results, if the entropy is "big", i.e., the number of 0's and 1's are near the same, the entropy decreases on the average when multiplied by an encoding matrix. This we have calculated on a 8 bit long input and the Ideal Soliton degree distribution as described in [3]. As we shall see later, we can construct a special encoding matrix with almost all distribution weigth on degree 2 with which the reduction in entropy is demonstrated also on large input lengths. For the first mentioned 8 bit long input, with 4 zeros and 4 ones we give the probabilities of zero in output in Table I ($\rho(i)$ is the Ideal Soliton degree distribution):
The two last columns multiplied with each other and summed results in 0.47321 probability of zero in output. This is lower than the 0.5 probability at the input. One may think that there is some other degree distribution with even better reduction in entropy or that there exists some degree distribution that gives the best reduction. It can be seen from the Table I that

degree i	$\rho(i)$	Probability of zero in output
1	1/8	0.5
2	1/2	0.42857
3	1/6	0.5
4	1/12	0.52857
5	1/20	0.5
6	1/30	0.42857
7	1/42	0.5
8	1/56	1

TABLE I
IDEAL SOLITON DEGREE PROBABILITIES AND PROBABILITIES OF ZERO IN OUTPUT FOR A 8-BIT LENGTH INPUT WITH EQUAL NUMBER OF ZEROS AND ONES

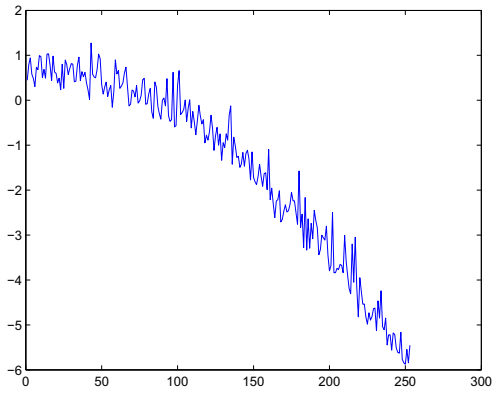


Fig. 1. Saving in bits (vertical direction) when transformation is applied to input of length 30204 with different reduced number of ones in the input (horizontal direction)

degree $i = 2$ gives the lowest probability of zero in the output, 0.42857. We used this degree and formed a special invertible encoding matrix R:

$$r_{lm} = 1, \text{ when } m = l \text{ or } m = l + 1$$

$$r_{lm} = 0, \text{ otherwise.}$$

For example, a 4×4 encoding matrix R would be:

$$R = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

There is a single 1 in the last row at the last column. These matrices are invertible and thus suitable for encoding and decoding. We ran simulations with input length 30204 and 61408. We used different initial proportions of zeros and ones in input. We calculated the effect of the entropy change to the optimal representation of the bits. In Figure 1 we see the saving in bits as the function of uneveness of 0's and 1's at the input. The input vector length is 30204 bits. At the left border there are 0 ones less than zeros in input. Near the right border there are 250 ones less in input than in the left border. With each horizontal value ten simulations were made, with randomly placed ones, and the average was taken. We see that for horizontal values 0..90 the needed number of bits in

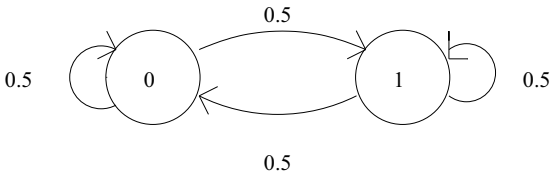


Fig. 2. A stochastic process defined by a Markov chain.

representing the output is lower than the needed number of bits needed in representing the input. The reduction is at most near 1 bit. With larger horizontal values the entropy increases and more bits are needed to represent the output. In principle, we got similar results with input vector length 61408. The space saving is still at most less than one bit on average. The results rise some conjectures which we present here.

Conjecture 1: The space saving using this transformation is on average less than one bit, even when applied to high-entropy input.

Let us consider a stochastic process defined by the Markov chain in Figure 2. The average entropy of a finite length realization of this process can be calculated with the help of a binomial probability density function multiplied by entropy function. It would be tempting to think that the average entropy equals to the entropy calculated from the proportions of 0's and 1's when the number of ones is decreased by the standard deviation of the associated binomial distribution. The standard deviation of a binomial distribution is $\sigma = \sqrt{np(1-p)}$. But, according to our calculations, this is not the case. These entropies differ.

Conjecture 2: The transformation described above can not reduce on average the entropy of a realization of the stochastic process described above below the average entropy of the same process.

V. CONCLUSIONS

Fountain codes and especially their invertible encoding matrices were dealt with. Encoding matrices induces a unique permutation on k bit strings. Encoding matrices with multiplication operation form a group. Encoding matrices sampled from the Ideal Soliton degree distribution reduce entropy at least in a special case. A special transformation matrix model was formed and different input vectors were used in simulations which showed reduction in entropy when the initial entropy was high. Although some entropy reduction was possible, it was on average less than one bit from tens of thousands of bits, so in practice this this method may not be applicable. However, more research is still needed to show if Conjecture 1 holds.

REFERENCES

[1] J. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data", in *Proc. ACM SIGCOMM '98*, Vancouver, BC, Canada, Jan. 1998, pp. 56-67

[2] D. L. Kreher and D. R. Stinson, *Combinatorial Algorithms, Generation, Enumeration and Search*, CRC Press LLC, Boca Raton, FL, 1999

[3] M. Luby, ”LT-codes”, in *Proc. 43rd Annu. IEEE Symp. Foundations of Computer Science (FOCS)*, Vancouver, BC, Canada, Nov. 2002, pp. 271-280

Raptor Codes and Cryptographic Issues

Mikko Malinen, *Student Member, IEEE.*

***Abstract*—In this paper two cryptographic methods are introduced. In the first method the precence of a certain size subgroup of persons can be checked for an action to take place. For this we use fragments of Raptor codes delivered to the group members. In the other method a selection of a subset of objects can be made secret. Also, it can be proven afterwards, what the original selection was.**

***Index Terms*—Presence of subgroup, Raptor codes, Decodability, Private subset, Factoring of numbers, Election predicting, Self-fulfilling prophecies**

I. Checking the presence of a certain size subgroup of persons

CONSIDER, that we have a group of persons. For an action to take place, we require that at least s members of the group are present. The action could be firing a weapon, or we could perform some action for which we need a subset of a certain size of board members of a company, or just some action for which to take place we need to check the presence of a subset of certain size of some interest group.

This checking of presence could be made by Raptor codes (for Raptor codes, see [1]). We could deliver in advance fragments of Raptor code to each member of the group. The fragment size should be such that we obtain the needed key (which is decoded from concatenated fragments) only if at least s group members are present. From the properties of Raptor codes it follows that we can recover the key with any subset of size s of fragments.

In this paragraph we calculate the maximum number of group members whose presence could be checked by this method. Let s be the number of persons whose presece is needed for an action to take place, and let k be the length of the key (number of input symbols). s fragments of Raptor code should be $1.1 \cdot k$ in total length to be decodable with very high probability. As we know from [2], 0.05 overhead can make the code decodable. We require here 0.1 overhead (this is the amount which is a design choice of at least some companies for their Raptor codes [3]). Also, the total length of $s-1$ fragments of Raptor code must be less than k in length for the code to be non-decodable. Let's say that $s-1$ fragments is $0.99 \cdot k$ in length. With one output symbol, at most one input symbol can be recovered. This is why it is not possible to decode whole message with $s-1$ fragments or less. From these we get

$$\lfloor \frac{\text{total length of fragments}}{\text{length difference between } s \text{ and } s\text{-}1 \text{ fragments}} \rfloor$$

$$= \lfloor \frac{1.1k}{1.1k - 0.99k} \rfloor = 10$$

which is the maximum number of group members whose presence could be made required for an action to take place.

It should be noted that the group size is almost not limited from above. The group could be for example the population of a nation. The group members may carry the coded fragments by memory sticks and the key is stored on a computer where the memory sticks will be attached. The group members can be even geographically distributed and for example they attach their memory sticks to their computers which are connected to Internet by a secure connection.

This scheme could be implemented also by passwords. Raptor code method is better when passwords can not be used for some reason, for example if their space requirement is too large due to the big size of the group.

A drawback of this method is that if the key has to be changed, then all the code fragments have to be changed also.

II. Private subset of objects

Let's begin the description of this application with a case. In Finland, for example, in national Lotto the customers guess 7 numbers out of 39. Let's suppose that the Lotto customers want to keep their selection of numbers secret, but send their selection to the Lotto company. This can be done so that to each of 39 Lotto numbers an integer i is assigned, which is the product of two big random prime numbers j and k. Also, an integer l 0..9 is assigned to i, which is formed as follows: If the customer has selected number i, l is the mod 10 sum of the digits of j and k. For example, if j and k are 327...3 and 615...7 then l is 3+2+7+...+3+6+1+5+...+7 (mod 10). In case the Lotto customer has *not* selected number i then l is formed so that we take the l calculated as above and add 1 to it and take mod 10. In this way the customer has i and l associated to each Lotto number 1..39 and he or she sends these to the Lotto company. Because of computational difficulty of factorization of composite numbers formed this way the Lotto company can't know j's and k's and so is not able to check if they match with l to know if the corresponding Lotto number is selected by the customer. The idea of a product of large primes is from the well known RSA [4] public key cryptography algorithm.

The Lotto company may send the file sent by the customer back to the customer decrypted by the company's private key in RSA manner. This "decrypted" file may serve as a receipt from the company.

When the drawing of Lotto numbers have been done,

the winning customers send their j's and k's to Lotto company so that it can check if the customer has won.

Another application very similar with the Lotto example is predicting the election results without telling a priori which candidates he or she predicts to win. The correctness of the prediction can be verified after the election result. This way we can avoid the effect of self-fulfilling prophecies because the prophecies will be kept secret. This method is applicable to almost all situations where a subset of a set has to be chosen and the selection has to be kept secret.

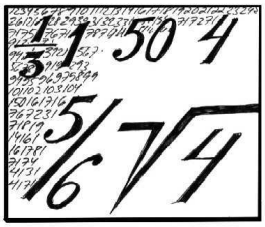
REFERENCES

[1] A. Shokrollahi, "Raptor Codes", *IEEE Transactions on Information Theory*, Vol. 52, No. 6, 2006

[2] D. J. C. MacKay, "Fountain codes", *IEEE Proceedings: Communications*, Vol. 152, No. 6, Dec. 2005, pp. 1062-1068

[3] S. Siikavirta, private conversation

[4] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", *Communications of the ACM*, Vol. 21, Feb. 1978, pp. 120-126



Dynamiikkaa, derivaattoja ja ennustavia kuumemittareita

Mikko Malinen
TkK, opiskelija
Teknillinen korkeakoulu

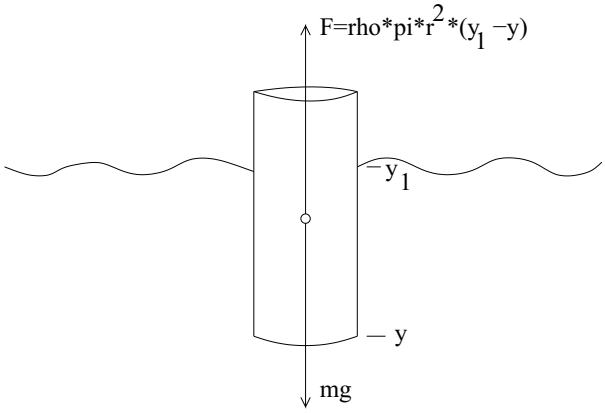
Johdanto

Dynamiikka on mekaniikan osa, joka käsittelee kappaleiden liikkeitä ja niihin liittyviä voimia. Tämän artikkelin ensimmäisessä osassa esitetään mielenkiintoinen dynamiikan sovellus: vedenkorkeuden laskeminen, kun vedenkorkeutta ei suoraan voida mitata, vaan käytetään massaa omaavaa kohoa. Artikkelissa osoitetaan, että vedenkorkeus voidaan laskea, jos tiedetään kohon asema ja kohon aseman toinen derivaatta. Artikkelin toisessa osassa käsitellään lämmön johtumista, aineen lämpenemistä lämmittimen ansiosta. Artikkelissa johdetaan yhtälö aineen loppulämpötilalle. Tämän yhtälön avulla aineen loppulämpötila voidaan laskea, kun tiedetään aineen lämpötila hetkellä t sekä sen lämpötilan ensimmäinen ja toinen derivaatta. Lämmittimen lämpötilaa ei tarvitse tietää. Artikkelin molemmat osat ovat erinomaisia esimerkkejä siitä, miten hyödyllisiä derivaatat ovat.

Vedenkorkeusesimerkki

Tarkastellaan ympyräsylinterin muotoista kohoa vedessä (kuva 1). Haluamme tietää veden korkeuden ja laskea sen kohon aseman avulla. Kohoon vaikuttava kokonaisvoima on kuvan 1 voimien summa:

$$F_1 = \rho \cdot \pi r^2 \cdot (y_1 - y) - mg \qquad (1)$$



Kuva 1. Koho vedessä. Muuttujat kuvassa: ρ veden tiheys, r kohon säde, y_1 veden taso, y kohon asema (pystysuunnassa).

Veden vastusta ei oteta huomioon. Newtonin ensimmäinen laki sanoo, että

$$F_1 = ma. \qquad (2)$$

(1):stä ja (2):sta seuraa

$$ma = \rho \cdot \pi r^2 (y_1 - y) - mg.$$

Koska kiihtyvyys a on toinen aikaderivaatta y:stä, voidaan kirjoittaa

$$my'' - \rho \cdot \pi r^2 (y_1 - y) - mg = 0.$$

Ratkaisemalla tämä y_1:n suhteen saadaan

$$y_1 = \frac{my''}{\rho\pi r^2} + y - \frac{mg}{\rho\pi r^2}.$$

Tämä voidaan kirjoittaa

$$\boxed{y_1 = by'' + y - c}$$

yksinkertaisuuden vuoksi. Tässä b ja c ovat tunnettuja vakioita. Tämä on tuloksemme. Veden taso riippuu kohon aseman toisesta derivaatasta ja itse kohon asemasta.

Kappaleen loppulämpötila voidaan ennustaa

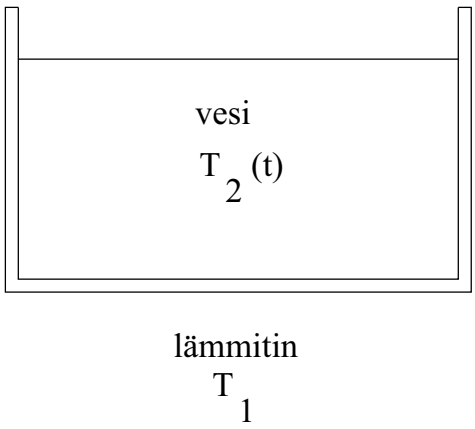
Lämmönvaihtoa kahden kappaleen välillä voidaan mallintaa yhtälöllä

$$\frac{dQ}{dt} = kA\frac{T_H - T_C}{L}$$

(ks. [1]). Tässä Q on kappaleen 2 lämpöenergia, k on lämmönjohtavuus, A on välissä olevan johtavan materiaalin läpileikkauspinta-ala, T_H on kappaleen 1 lämpötila, T_C on kappaleen 2 lämpötila ja L on kappaleiden 1 ja 2 välinen etäisyys. Tämä voidaan kirjoittaa lyhyesti

$$\frac{dQ}{dt} = a(T_1 - T_2(t))$$

missä Q on kappaleen 2 lämpöenergia, a on positiivinen verrannollisuuskerroin ja T_1 ja T_2 ovat kappaleiden 1 ja 2 lämpötilat. Katso esimerkkiasetelma kuvassa 2.



Kuva 2. Esimerkkiasetelma.

Lämpöenergia on verrannollinen lämpötilaan

$$Q - Q_0 = cm(T - T_0)$$

missä Q_0 on juuri sulamislämpötilan yläpuolella olevan kappaleen lämpöenergia, T ja T_0 ovat Celsius-asteikolla ja T:n sallitaan saavan arvoja joissa kappale on nestefaasissa. Esimerkiksi vedellä $0 < T < 100^oC$. T_0 on kappaleen sulamislämpötila. Tästä saadaan

$$T - T_0 = \frac{Q - Q_0}{cm}.$$

Kun muistetaan myös, että

$$\frac{d(Q - Q_0)}{dt} = \frac{dQ}{dt}$$

ja

$$\frac{d(T - T_0)}{dt} = \frac{dT}{dt}$$

voidaan kirjoittaa

$$\begin{aligned}\frac{dT_2(t)}{dt} &= \frac{d(T_2(t) - T_0)}{dt} = \frac{1}{cm} \cdot \frac{d(Q - Q_0)}{dt} \\ &= \frac{1}{cm}\frac{dQ}{dt} = \frac{1}{cm}a(T_1 - T_2(t)) \\ &= b(T_1 - T_2(t)), \quad b > 0, b \neq \infty.\end{aligned}$$

Tämä on differentiaaliyhtälö, jonka ratkaisu on

$$T_2(t) = T_1 + e^{-bt} \cdot C. \tag{3}$$

Derivoimalla tulos saadaan

$$T_2'(t) = -be^{-bt} \cdot C. \tag{4}$$

Derivoimalla uudestaan saadaan

$$T_2''(t) = b^2e^{-bt} \cdot C. \tag{5}$$

(4):sta ja (5):sta saadaan

$$\frac{T_2'(t)}{-b} = \frac{T_2''(t)}{b^2}.$$

Ratkaisemalla b saadaan

$$b = -\frac{T_2''(t)}{T_2'(t)}, \quad T_2''(t) \neq 0, T_2'(t) \neq 0.$$

(4):stä ja sijoittamalla b saadaan

$$C = \frac{T_2'(t)}{\frac{T_2''(t)}{T_2'(t)}e^{\frac{T_2''(t)}{T_2'(t)} \cdot t}} = \frac{(T_2'(t))^2}{T_2''(t)e^{\frac{T_2''(t)}{T_2'(t)} \cdot t}}.$$

Hetkellä $t = 0+$ (juuri 0:n jälkeen) C:stä tulee

$$C_{t=0+} = \frac{(T_2'(0+))^2}{T_2''(0+)}.$$

(3):stä ja sijoittamalla $C_{t=0+}$ saadaan (hetkellä $t = 0+$)

$$\begin{aligned}T_1 &= T_2(0+) - \frac{(T_2'(0+))^2}{T_2''(0+)}, \\ T_2'(0+) &\neq 0, \quad T_2''(0+) \neq 0.\end{aligned} \tag{6}$$

$T_2(t)$:n raja-arvo on (yhtälöstä (3)):

$$\lim_{t\to\infty} T_2(t) = \lim_{t\to\infty} T_1 + e^{-bt} \cdot C = T_1.$$

Tämä raja-arvo on loppulämpötila. Meillä on jo yhtälö (6) T_1:lle. Tämän yhtälön oikealla puolella on vain $T_2(0+)$ ja sen kaksi derivaattaa:

$$T_{2_{final}} = T_2(0+) - \frac{(T_2'(0+))^2}{T_2''(0+)}.$$

Koska mitä tahansa ajanhetkeä voidaan pitää uutena alkuajanhetkenä, voidaan kirjoittaa

$$\boxed{T_{2_{final}} = T_2(t) - \frac{(T_2'(t))^2}{T_2''(t)}, \quad t > 0}$$

joten kappaleen loppulämpötila voidaan ennustaa kun sen lämpötila ja lämpötilan kaksi derivaattaa voidaan mitata hetkellä $t > 0$. Lopuksi esitetään jo otsikossa mainittu sovellus: Menetelmää voitaisiin käyttää elektronisessa kuumemittarissa ennustamaan anturin loppulämpötila eli nopeuttamaan lämpötilan mittaamista.

Johtopäätökset

Tässä artikkelissa esitettiin kaksi menetelmää laskea muuttujan arvo silloin, kun sitä ei voida suoraan mitata. Kumpikin menetelmä on hyödynnettävissä käytännössä ja kummassakin derivaatat ovat merkittävässä osassa. Ne korostavat derivaattojen merkitystä.

Viitteet

[1] Hugh D.Young ja Roger A. Freedman, *University Physics*, 9. laitos, Addison-Wesley, 1996

Keskinäisinformaatiosta

Mikko Malinen

31. heinäkuuta, 2008

1 Johdanto

Keskinäisinformaatio (mutual information) on tärkeitä informaatioteorian käsitteitä. Keskinäisinformaatio $I(X;Y)$ on eräs riippuvuuden mitta kahden satunnaismuuttujan X ja Y välillä. $I(X;Y)$ määritellään

$$I(X;Y) = \sum_{x,y} p(x,y) \log_2 \frac{p(x,y)}{p(x)p(y)}.$$

Keskinäisinformaatiolle on voimassa

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X).$$

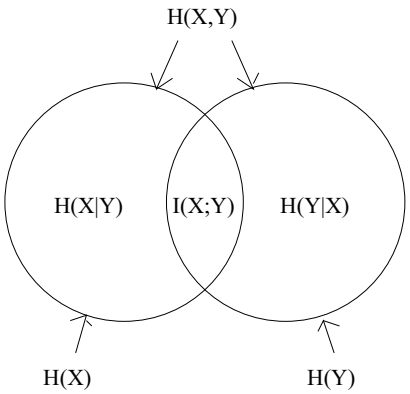
H:lla tarkoitetaan entropiaa, ks. [2][4]. Keskinäisinformaatiolle voidaan kirjoittaa

$$I(X;Y) = D(p(x,y)||p(x)p(y)),$$

missä D on *relatiivinen entropia* eli *Kullback-Leibler divergenssi* [2]. Keskinäisinformaatiolle on myös voimassa

$$I(X;Y) = H(X) + H(Y) - H(X,Y),$$

missä $H(X,Y)$ on yhteisentropia (joint entropy). Kuva 1 esittää entropian ja keskinäisinformaation suhdetta.



Kuva 1. Entropian ja keskinäisinformaation suhde

Keskinäisinformaatio on aina ei-negatiivinen

$$I(X;Y) \geq 0,$$

yhtäsuuruus on voimassa jos ja vain jos X ja Y ovat riippumattomia. Keskinäisinformaation yksikkö on bitti. Jatkuville satunnaismuuttujille yhteistiheysjakaumalla $f(x,y)$ keskinäisinformaatio määritellään

$$I(X;Y) = \int f(x,y) \log_2 \frac{f(x,y)}{f(x)f(y)} dxdy.$$

Seuraavissa kappaleissa esitellään keskinäisinformaation sovelluksia. Tässä kuitenkin ensimmäinen sovellus: Tietoliikennekanavan, jonka sisäänmeno on X ja ulostulo on Y, määrittelemme *kanavakapasiteetin* C seuraavasti:
$C = max_{p(x)} I(X;Y)$. Maksimointi on yli sisäänmenojakauman $p(x)$. Ennen muita sovelluksia muutama sana tämän artikkelin lähteistä. Informaatioteoria sai alkunsa Claude Shannonin artikkelista [4]. Tämän johdannon asiat löytyvät informaatioteorian oppikirjana käytetystä [2]:sta. Hevoskilpailua, osakemarkkinoita ja nopeus-särö teoriaa käsitellään [2]:ssa. Riippumattomien komponenttien analyysistä löytyy tietoa kirjasta [3]. Yksityisyyden vahvistusta on käsitelty artikkelissa [1].

2 Hevoskilpailu

Oletetaan, että m hevosta juoksee kilpailussa. Voittakoon i.s hevonen todennäköisyydellä p_i. Jos i. hevonen voittaa, vedonlyöja voittaa o_i:n suhde yhteen eli yhden euron sijoitus hevoselle i aiheuttaa o_i euron voiton jos hevonen i voittaa ja 0 euron voiton jos hevonen i häviää. Olkoon b_i se osa vedonlyöjän varoista, joka on sijoitettu hevoselle i, missä $b_i \geq 0$ ja $\sum b_i = 1$. Olkoon $S(X) = b(X)o(X)$

kerroin, jolla vedonlyöjän varallisuus nousee jos hevonen X voittaa kilpailun. Hevoskilpailun *tuplausnopeus* on

$$W(\bar{b}, \bar{p}) = E(\log S(X)).$$

Tuplausnopeuden merkitys sselviää seuraavasta lauseesta:

Lause. Olkoon kilpailujen tulokset $X_1, X_2, ..., X_n$ i.i.d. $\sim p(x)$. Tällöin vedonlyöjän varallisuus käyttäen vedonlyöntistrategiaa $\bar{b}$ kasvaa eksponentiaalisesti nopeudella $W(\bar{b}, \bar{p})$ eli $S_n = 2^{nW(\bar{b},\bar{p})}$.

Nyt oletetaan, että vedonlyöjällä on tietoa, joka on relevanttia kilpailun lopputuloksen kannalta. Esimerkiksi, vedonlyöjällä saattaa olla informaatiota hevosten suorituskyvystä aiemmissa kilpailuissa. Mikä on tämän sivuinformaation arvo? Olkoon (X, Y):llä yhteistodennäköisyysmassafunktio $p(x, y)$.

Lause. Tuplausnopeuden kasvu ΔW sivuinformaation Y johdosta hevoskilpailussa X on

$$\Delta W = I(X; Y).$$

3 Osakemarkkina

Osakemarkkinaa esitetään vektorina osakkeita $\bar{X} = (X_1, X_2, ..., X_m), X_i \geq 0, i = 1, 2, ..., m$, missä m on osakkeiden lukumäärä ja X_i esittää päivän lopun hinnan suhdetta päivän alun hintaan. Tyypillisesti X_i on lähellä 1:tä. Esimerkiksi $X_i = 1,03$ tarkoittaa, että i:s osake nousi 3% tuona päivänä. *Salkku* $\bar{b} = (b_1, b_2, ..., b_m), b_i \geq 0, \sum b_i = 1$ on varallisuuden jako eri osakkeille. b_i on se osuus henkilön varallisuudesta, joka on sijoitettu osakkeelle i. Jos henkilö käyttää salkkua $\bar{b}$ ja osakevektori on $\bar{X}$,niin päivän lopun varallisuuden suhde päivän alun varallisuuteen on $S = \bar{b}^t \bar{X}$. Osakemarkkinasalkun *tuplausnopeus* määritellään

$$W(b, F) = \int \log_2 \bar{b}^t \bar{x} dF(\bar{x}) = E(\log \bar{b}^t \bar{X}).$$

Olkoon nyt meillä osakevektori $\bar{X}$, sivuinformaatio Y ja yhteistodennäköisyysmassafunktio $f(\bar{x}, y)$.

Lause. Tuplausnopeuden kasvu ΔW sivuinformaation Y johdosta on rajoitettu seuraavasti:

$$\Delta W \leq I(\bar{X}, Y).$$

Siis tuplausnopeuden kasvu on ylhäältä rajoitettu sivuinformaation Y ja osakemarkkinan $\bar{X}$ keskinäisinformaatiolla.

4 Nopeus-särö teoria (Rate Distortion Theory) ja kvantisointi

Tarkastellaan ongelmaa, miten edustaa yksittäistä näytettä lähteestä. Olkoon satunnaismuuttuja, jota edustetaan X ja olkoon X:n edustaja $\hat{X}(X)$. Jos on annettu R bittiä X:n edustamiseen, niin funktio $\hat{X}$ voi saada 2^R arvoa. Kvantisointiongelma on löytää optimijoukko arvoja $\hat{X}$:lle (rekonstruktiopisteet) ja alueet jotka assosioidaan kullekin arvolle $\hat{X}$. Oletetaan, että meillä on lähde, joka tuotta jonon $X_1, X_2, ..., X_n$ i.i.d. $\sim p(x), x \in \mathcal{H}$. Kooderi kuvailee lähdejonon X^n indeksillä $f_n(X^n) \in \{1, 2, ..., 2^{nR}\}$. Dekooderi esittää X^n:n estimaatilla $\hat{X}^n \in \hat{\mathcal{H}}^n$.

Määritelmä. *Säröfunktio* tai *särömitta* on kuvaus

$$d : \mathcal{H} \times \hat{\mathcal{H}} \rightarrow \mathbf{R}^+$$

joukosta lähdeaakkosto-reproduktioaakkosto pareja joukkoon ei-negatiiviset reaaliluvut. Särö $d(x, \hat{x})$ on mitta sille kustannukselle, että symbolia x edustetaan symbolilla $\hat{x}$.

Määritelmä. Nopeus-särö parin (R, D) sanotaan olevan *saavutettavissa* jos on olemassa jono $(2^{nR}, n)$ nopeus-särö koodeja (f_n, g_n) joille
$\lim_{n\rightarrow\infty} Ed(X^n, g_n(f_n(X^n))) \leq D$.

Määritelmä. Lähteen nopeus-särö alue on saavutettavien nopeus-särö parien (R, D) joukon sulkeuma.

Määritelmä. Nopeus-särö funktio $R(D)$ on nopeuksien R infimum, niin että (R, D) on lähteen nopeus-särö alueessa annetulla säröllä D.

Lause. Nopeus-särö funktio i.i.d. lähteelle X jakaumalla $p(x)$ ja rajoitetulla säröfunktiolla $d(x, \hat{x})$ on

$$R(D) = min_{p(\hat{x}|x):\sum_{(x,\hat{x})} p(x)p(\hat{x}|x)d(x,\hat{x}) \leq D} I(X; \hat{X})$$

missä minimointi on yli kaikkien ehdollisten jakaumien $p(\hat{x}|x)$, joille yhteisjakauma $p(x, \hat{x}) = p(x)p(\hat{x}|x)$ toteuttaa odotetun särörajoitteen.

5 Riippumattomien komponenttien analyysi (Independent Component Analysis, ICA)

Aloitetaan esimerkillä: kolme henkilöä puhuvat samanaikaisesti huoneessa, jossa on kolme mikrofonia. Merkitään mikrofonisignaaleja $x_1(t), x_2(t)$, ja $x_3(t)$:llä. Jokainen näistä on puhesignaalien painotettu summa, merkitään näitä $s_1(t), s_2(t)$ ja $s_3(t)$:llä:

$$x_1(t) = a_{11}s_1(t) + a_{12}s_2(t) + a_{13}s_3(t)$$
$$x_2(t) = a_{21}s_1(t) + a_{22}s_2(t) + a_{23}s_3(t)$$
$$x_3(t) = a_{31}s_1(t) + a_{32}s_2(t) + a_{33}s_3(t)$$

Seuraavaa ongelmaa kutsutaan coctail-kutsu ongelmaksi: estimoi alkuperäiset puhesignaalit $s_i(t)$ *käyttäen vain äänitettyjä signaaleja.* s:lle voidaan kirjoittaa yhtälö $s = Wx$, missä $W = A^{-1}$, ja $x = As$. ICA on lineaarinen muunnos $s = Wx$, niin että komponenttien s_i keskinäisinformaatio minimoituu. ICA:n käytännön laskemisen kannalta merkitystä on sillä, että ICA estimointi minimoimalla keskinäisinformaatio on ekvivalenttia sen kanssa, että maksimoidaan riippumattomien komponenttien estimaattien ei-gaussisuuksien summa (kun estimaateilla on rajoite, että ne eivät ole korreloituneita).

6 Yksityisyyden vahvistus (Privacy Amplification)

Kryptografiassa kommunikointi tapahtuu kommunikoijien Alice ja Bob välillä. Lisäksi oletetaan, että on kolmas toimija, Eve, joka pyrkii saamaan tietoa välitettävistä viesteistä ja/tai vaikuttamaan niihin. Kvanttikryptografiassa, kayttäen Bennett-Brassard -protokollaa, saadaan n bitin mittainen merkkijono W jaettua Alicen ja Bobin kesken. Evellä on kuitenkin tietoa tästä avaimesta. Olkoon $V = e(W)$ mielivaltainen salakuuntelufunktio $e : \{0,1\}^n \to \{0,1\}^t$ jollekin $t < n$, olkoon $s = n - t$ positiivinen turvaparametri ja olkoon $r = n - t - s$. Jos Alice ja Bob valitsevat $K = G(W)$ salaiseksi avaimekseen, missä G on valittu satunnaisesti universaalista luokasta hash funktioita $\{0,1\}^n$:stä $\{0,1\}^r$:ään, niin Even odotettu informaatio salaisesta avaimesta K, kun G ja V on annettu, toteuttaa

$$I(K; GV) \leq 2^{-s} / \ln 2.$$

Huomataan, että Even informaatio kutistuu eksponentiaalisesti kun s kasvaa: jokainen lisäbitti, jonka Alice ja Bob haluavat uhrata pienentää Even informaation ylärajaa lisätekijällä 2.

7 Yhteenveto

Keskinäisinformaatio on satunnaismuuttujien välisen riippuvuuden mitta. Keskinäisinformaatio voidaan laskea, kun yhteisjakauma $p(x, y)$ ja todennäköisyysmassafunktiot $p(x)$ ja $p(y)$ tunnetaan. Monessa sovelluksessa tulee kysymykseen keskinäisinformaation maksimointi tai minimointi.

8 Viitteet

[1] C.H. Bennett, G. Brassard, C. Crépeau ja U.M. Maurer, "Generalized privacy amplification", *IEEE Transactions on Information Theory*, **41**(6), 1915-1923 (Marrask. 1995).

[2] T.M. Cover ja J.A. Thomas, *Elements of Information Theory*, John Wiley & Sons, USA, 1991

[3] A. Hyvärinen, J. Karhunen ja E. Oja, *Independent Component Analysis*, J. Wiley, 2001

[4] C. E. Shannon, "A Mathematical Theory of Communication", *Bell Syst. Tech. J.*, **27**, 379-423, 623-656 (1948).

Information transmission is possible with any amount of energy

Mikko Malinen

22th January, 2008

Abstract

This paper shows that information transmission is possible with any amount of energy, also with very low energy. A basis for this result is the formula for the capacity of a band-limited Gaussian channel with noise.

1 From the capacity equation to the result

The equation

$$C = W \log(1 + \frac{P}{N_0 W})$$

bits per second is one of the most famous formulae of information theory. It gives the capacity of a band-limited Gaussian channel with noise spectral density $N_0/2$ watts/Hz and power P watts.

Now, we could make an assumption that we can achieve half of this capacity in practical communication. This is a realistic amount, and for the sake of generality we mark the proportion of the achievable capacity from the maximum by a, $0 < a \leq 1$.

The equation becomes

$$C_{realistic} = aW \log(1 + \frac{P}{N_0 W}).$$

To transmit certain number of bits, $C_{realistic}$ has to be multiplied by t. Let's multiply the equation from both sides:

$$C_{realistic} \cdot t = aWt \cdot \log(1 + \frac{P}{N_0 W})$$

Now, we want to transfer n bits at whatever capacity and by using whatever amount of time. We keep $C_{realistic} \cdot t$ constant:

$$aWt \cdot \log(1 + \frac{P}{N_0 W}) = \text{constant}$$

We know that $P = \frac{E}{t}$, that is, energy divided by time. We get

$$aWt \cdot \log(1 + \frac{E}{N_0 tW}) = \text{constant}$$

From this equation, we see that when E approaches zero, then t approaches infinity. From the equation and this conclusion we can state the result of this paper:

Transmission of information is possible with any amount of energy, also with very low energy. The time required for transmission grows fast when energy approaches zero.

A Theorem Prover for Mathematical Theorems

Mikko Malinen

17th August, 2007

Abstract

This paper discusses possibilities to create a theorem prover for proving mathematical consequence and presents a theorem prover for mathematical theorems with restricted language.

Keywords: Theorem prover, completeness, Gödel's incompleteness theorem

1 Introduction

In propositional and predicate logic there exist theorem provers. These prove logical consequence given the premises and the goal. In mathematics, we may consider also logical consequence, but instead of having logical formulas as the premises and the goal, we have mathematical theorems. From Gödel's incompleteness theorem it follows that $S \Rightarrow$ C , where S is a mathematical premise set and C is a mathematical consequence, can not be proven true or false in the general case (because $\emptyset \Rightarrow C_2$ or C_2 can not be proven (Gödel). Nevertheless, the language can be restrictd so that $S_3 \Rightarrow C_3$, where S_3 is a premise set formed according to restricted language and C_3 is a consequence formed according to restricted language can be proven true or false. The proof system is so decidable. This paper presents a theorem prover for such a restricted language. The proof system is sound and complete.

2 Accepted syntax in the premises and the goal

In this theorem prover the premises and the goal should follow the following syntax:

$$b_{11}x_1 + b_{12}x_1 + ... + b_{1k}x_1 + b_{21}x_2 + b_{22}x_2 + ... + b_{2l}x_2 + ... + b_{n1}x_n + b_{n2}x_n + ...$$
$$+b_{nm}x_n + d_{11} + d_{12} + ... + d_{1s} = b_{n+1,1}x_1 + b_{n+1,2}x_1 + ... + b_{n+1,k}x_1 + b_{n+2,1}x_2$$
$$+b_{n+2,2}x_2 + ... + b_{n+2,l}x_2 + ... + b_{n+n,1}x_n + b_{n+n,2}x_n + ... + b_{n+n,m}x_n + d_{21}$$
$$+d_{22} + ... + d_{2s} \quad (1)$$

or

$$b_{11}x_1 + b_{12}x_1 + ... + b_{1k}x_1 + b_{21}x_2 + b_{22}x_2 + ... + b_{2l}x_2 + ... + b_{n1}x_n + b_{n2}x_n + ...$$

$$+b_{nm}x_n + d_{11} + d_{12} + ... + d_{1s} \neq b_{n+1,1}x_1 + b_{n+1,2}x_1 + ... + b_{n+1,k}x_1 + b_{n+2,1}x_2$$

$$+b_{n+2,2}x_2 + ... + b_{n+2,l}x_2 + ... + b_{n+n,1}x_n + b_{n+n,2}x_n + ... + b_{n+n,m}x_n + d_{21}$$

$$+d_{22} + ... + d_{2s}, \tag{2}$$

where b_{ij}'s and d_{ij}'s are replaced by numerical constants. Terms in either side of the $=$ or $\neq$ characted may be in any order. (1) and (2) may be written

$$a_1x_1 + a_2x_2 + ... + a_nx_n + d_1 = a_{n+1}x_1 + a_{n+2}x_2 + ... + a_{n+n}x_n + d_2 \tag{3}$$

or

$$a_1x_1 + a_2x_2 + ... + a_nx_n + d_1 \neq a_{n+1}x_1 + a_{n+2}x_2 + ... + a_{n+n}x_n + d_2 \tag{4}$$

where a_i's and d_j's are replaced by numerical constants. If some terms are zero, they may not be written. The normal form of the equation (3) is

$$x_i = \frac{(a_{n+1} - a_1)x_1 + (a_{n+2} - a_2)x_2 + ... + (a_{2n} - a_n)x_n}{a_i - a_{n+i}}$$

$$+\frac{-(a_{n+1} - a_i)x_i + d_2 - d_1}{a_i - a_{n+i}}$$

$$= \frac{a_{n+1} - a_1}{a_i - a_{n+i}}x_1 + \frac{a_{n+2} - a_2}{a_i - a_{n+i}}x_2 + ...$$

$$+\frac{a_{2n} - a_n}{a_i - a_{n+i}}x_n - \frac{a_{n+1} - a_i}{a_i - a_{n+i}}x_i + \frac{d_2 - d_1}{a_i - a_{n+i}}, \tag{5}$$

where coefficients of x's and the last term are calculated to have a numerical value. Note that on the right hand side the terms including x_i's disappear. The terms are arranged in increasing order of indices of x's (except the constant, which is placed last). If some terms are zero, they are left out. But, if the right hand side of (5) from '=' character is zero, then it is written as 0. The normal form with respect to x_i of the disequation (4) is the same as for (3) but having the disequality character $\neq$. The *negation* of (3) is got by changing the $=$ character to $\neq$. The negation of (4) is got by changing the $\neq$ character to $=$.

3 Theorem Prover

The theorem proving algorithm is presented next. In data structures there is a list where equations and disequations are added. When an item is added to list, there are certain things to do, see later. Here is the algorithm.

- For all premises, one at a time:
- for all of the variables involved in the premise:
- add normal form of the premise with respect to variable to list
- for all of the variables involved in the goal:
- add normal form of the goal with respect to variable to list
- if proof not found, then there is no proof

When an item is added to list, do (only one of) the following: 1) If the item is $0 = 0$, do not add it to list, 2) If the item is $0 \neq 0$, there is a proof, 3) Go through the items in the list before the added item and compare it to the added item: a) if there is a contradiction (when a variable equals a value and in the other item there is a disequality character between the variable and that value) there is a proof b) if the two items may be combined, combine them and add to list if not already there. Two items may be combined when the other includes a variable which is the left value of the other.

4 An Example of a Proof

Here is an example proof. Let the premises be

$$2a + 3x + 5 = 0$$

and

$$a = 1.$$

Let the goal be

$$x = -\frac{7}{3}.$$

The proceeding of the algorithm can be seen from the table 1.

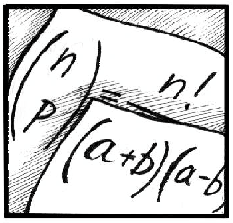
Step (=item number)	Item added to list	By what rule we get this item
(1)	$x = -\frac{2}{3}a - \frac{5}{3}$	First premise
(2)	$a = -\frac{3}{2}x - \frac{5}{2}$	First premise
(3)	$a = 1$	Second premise
(4)	$x = -\frac{7}{3}$	(1),(3) combined
(5)	is already,not added	(2),(3) combined
(6)	$x \neq -\frac{7}{3}$	Goal, negation
(7)	Proof	(4),(6) contradiction

Table 1. Proceeding of the algorithm.

5 Conclusions

This paper presented a theorem prover for mathematical theorems with restricted language. This is only the beginning, the language can be expanded. This leaves room for future work. However, according to Gödel's incompleteness

theorem, all true theorems can never be proven, so the language can never be made unrestricted if we want to preserve the completeness of the proof system.



Klikkimenetelmä kombinatoristen ongelmien ratkaisemisessa

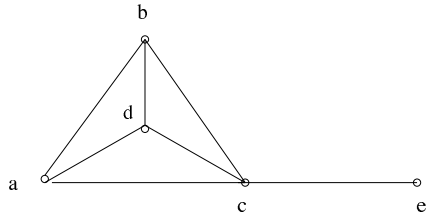
Mikko Malinen
TkK, opiskelija
Teknillinen korkeakoulu

Johdanto

Ratkaistaessa tietyntyyppisiä kombinatorisia yhteensopivuusongelmia tietokoneella voidaan käyttää klikkimenetelmää. Menetelmässä muodostetaan ensin *graafi* ja sitten etsitään siitä tietynkokoinen *klikki*. Tämäntyyppiset ongelmat voitaisiin periaatteessa ratkaista myös järjestelmällisellä eri tapausten läpikäynnillä (brute force) tai peräytyvällä haulla (backtrack search). Ratkaiseminen näillä on usein käytännössä mahdotonta vaadittavan liian suuren laskenta-ajan takia. Menetelmä soveltuu seuraavanlaisten ongelmien ratkaisuun: Olkoon äärellinen määrä alkioita. Jokainen alkio voi saada arvokseen jonkin arvon annetusta äärellisestä arvojoukosta. Jonkin tai joidenkin alkioiden arvo rajoittaa jonkun toisen tai joidenkin toistaen alkioiden mahdollisia arvoja. On löydettävä kaikille alkioille arvot, jotka ovat yhteensopivia kaikkien muiden alkioiden arvojen kanssa. Voidaan olla myös vähemmän kiinnostuneita alkioiden arvoista, mutta halutaan tietää ratkaisujen kokonaismäärä. Edellisen ongelman esimerkkeinä esitetään erään erään yksinkertaisen kombinatorisen ongelman ratkaiseminen, Einsteinin ongelman ratkaiseminen sekä sudokutehtävän ratkaiseminen. Jälkimmäisen ongelman esimerkkinä esitetään virheenkorjaavien koodien määrän laskeminen.

Hieman graafiteoriaa

Graafi on pari $G = (V, E)$, missä V on solmujen joukko ja E on solmuja yhdistävien kaarien joukko (ks. kuva 1). Yksi kaari yhdistää kaksi solmua toisiinsa.



Kuva 1. Eräs graafi. Solmujen joukko V = {a,b,c,d,e}. Solmuja yhdistää joukko kaaria.

Klikki on sellainen V:n osajoukko, jossa jokainen solmu on yhdistetty jokaiseen toiseen solmuun kaarella. Kuvan 3 graafin maksimiklikki on $C_1 = \{a, b, c, d\}$. Myös esim. $C_2 = \{b, c, d\}$ on klikki.

Menetelmä

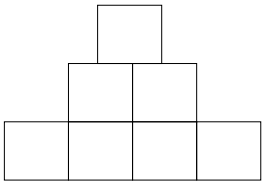
Tarkastellaan graafia, jonka solmuina on ongelman alkiot kaikkine mahdollisisne arvoineen. Jos ongelmas-

sa on n alkiota ja kullakin niistä on k mahdollista arvoa, tulee graafiin $n \cdot k$ solmua. Olkoon graafin kahden solmun välillä kaari jos ja vain jos niiden arvot ovat keskenään yhteensopivat. Ratkaisun löytämiseksi eli alkioiden arvojen määräämiseksi graafista etsitään n:n suuruinen klikki. Mikäli halutaan tietää ratkaisujen lukumäärä, etsitään n:n suuruisten klikkien lukumäärä. Algoritmeja klikin etsimiseksi on esitetty [1]:ssä ja [2]:ssa. Yksi mahdollisuus on käyttää valmista aliohjelmatyökalua, jollainen on esimerkiksi [3].

Esimerkkejä

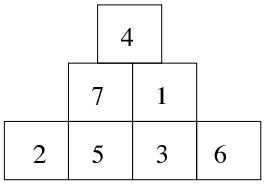
Yksinkertainen tehtävä

Tämä tehtävä on melko helppo ratkaista kynällä ja paperilla, mutta esimerkin yksinkertaisuuden vuoksi esitetään tässä. Tämä on myös esimerkkinä siitä, minkä tyyppisiin tehtäviin menetelmää voidaan käyttää. Tehtävänä on täyttää ruudukkoon numerot 1-7, yksi kutakin, niin, että toisiaan koskettavat ruudut eivät sisällä peräkkäisiä numeroita (ks. kuva 2).



Kuva 2. Täytettävä ruudukko

Graafiin tulee solmuja $n \cdot k = 7 \cdot 7$ kappaletta. Kuhunkin ruutuun assosioidaan 7 solmua, joihin assosioidaan myös numerot 1-7, yksi kuhunkin. Kahden solmun välille asetetaan kaari jos ja vain jos niitä vastaavat ruudut ja numerot ovat yhteensopivia. Esimerkiksi kahden vierekkäisen ruudun numeroiden ollessa peräkkäiset solmujen välillä ei ole kaarta. Ratkaisu löytyy etsimällä seitsemän kokoinen klikki graafista. Ratkaisujen lukumäärä saadaan laskemalla seitsemän kokoisten klikkien lukumäärä graafissa. Eräs ratkaisu on esitetty kuvassa 3.



Kuva 3. Eräs ratkaisu tehtävään

Sudokutehtävän ratkaiseminen klikkimenetelmällä

Latinalaiset neliöt

Astetta n oleva *latinalainen neliö* on $n \times n$ taulukko, joka koostuu n symbolista niin että jokainen symboli esiintyy tasan kerran jokaisella rivillä ja jokaisessa sarakkeessa (ks. kuva 4). Jatkossa oletetaan, että symbolit ovat $1, 2, ..., n$.

1	2	3	4
2	1	4	3
3	4	1	2
4	3	2	1

Kuva 4. Eräs latinalainen neliö

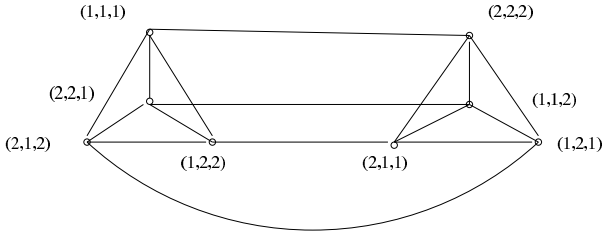
Osittaisessa latinalaisessa neliössä osa alkioista on määrittelemättä. Osittaisen latinalaisen neliön *täydentäminen* on määrittelemättömien alkioiden määrääminen niin, että osittainen neliö täydentyy sallituksi latinalaiseksi neliöksi.

1	2	3	4	5
2	3	.	.	.
3	.	4	.	.
4	.	.	5	.
5	.	.	.	2

Kuva 5. Eräs osittainen latinalainen neliö

Algoritmi

Graafien kielellä osittaisen latinalaisen neliön täydennys voidaan suorittaa seuraavalla tavalla: Tarkastellaan graafin, jonka solmuina on n^3 kpl muotoa (i, j, k) olevaa kolmikkoa; $i, j, k \in \{1, 2, ..., n\}$. Mitkä tahansa kaksi erillistä kolmikkoa on yhdistetty kaarella jos ja vain jos kolmikot ovat yhtäsuuria korkeintaan yhdessä koordinaateista. Kuvassa 6 on arvolla $n = 2$ syntyvä graafi.



Kuva 6. Graafi arvolla $n = 2$

Intuitio on se, että jokainen kolmikko (i, j, k) kertoo, että annamme arvon k alkiolle rivillä i, sarakkeessa j $n \times n$ taulukossa. Huomaa, että n:n asteen latinalaiset neliöt vastaavat n^2 kokoisia graafin klikkejä. Myöskin, osittainen latinalainen neliö voidaan täydentää jos ja vain jos vastaavat kolmikot ilmenevät kokoa n^2 olevassa klikissä.

Algoritmin muokkaaminen sudokun ratkaisemiseksi tapahtuu niin, että sudokun lisärajoitus, luvut 1-9 laatikossa, huomioidaan siten, että kussakin laatikossa alkioiden välillä on kaari jos ja vain jos kyseiset kaksi alkiota eivät ole arvoltaan samat. Ja sudokussahan $n = 9$.

Einsteinin ongelman ratkaiseminen klikkimenetelmällä

Tästä ongelmasta on olemassa erilaisia versioita. Tässä niistä yksi: Pienellä kadulla on viisi eri väristä taloa. Viisi eri kansallisuutta olevaa ihmistä asuu näissä viidessä talossa. Joskaisella asukkaalla on eri ammatti, jokainen pitää eri juomasta ja jokaisella on eri kotieläin. On annettu seuraavat tiedot:

Englantilainen asuu punaisessa talossa.
Espanjalaisella on koira.
Japanilainen on maalari.
Italialainen juo teetä.
Norjalainen asuu vasemmanpuoleisimmassa talossa.
Vihreän talon omistaja juo kahvia.
Vihreä talo on valkoisen talon oikealla puolella.
Kuvanveistäjä kasvattaa etanoita.
Diplomaatti asuu keltaisessa talossa.
Keskimmäisessä talossa joudaan maitoa.
Norjalainen asuu sinisen talon vieressä.
Viulunsoittaja juo hedelmämehua.
Kettu on talossa, joka on lääkärin talon vieressä.
Hevonen on talossa, joka on diplomaatin talon vieressä.

Kysymys kuuluu, kenellä on seepra ja kuka juo vettä? Tämän ongelman ratkaisemiseksi riittää määrittää jokaiselle talolle sen viisi ominaisuutta:

- väri
- omistajan kansallisuus
- omistajan kotieläin
- omistajan ammatti
- omistajan mielijuoma

Taloja on viisi, kullakin talolla on viisi ominaisuutta ja kullakin ominaisuudella on viisi mahdollista arvoa. Muodostetaan graafi, jossa on $5 \cdot 5 \cdot 5 = 125$ solmua. Yhdestä solmusta tiedetään mikä talo on kyseessä, mikä ominaisuus on kyseessä ja mikä on ominaisuuden arvo. Sitten solmujen välille vedetään kaaria niin, että vain yhteensopivien solmujen välille tulee kaari. Ratkaisu saadaan, kun näin muodostuvasta graafista etsitään 25:n kokoinen klikki. Tämä klikki kertoo, mitkä arvot kunkin talon ominaisuudet saavat. Kerrottakoon, että ratkaisu on: Japanilaisella on seepra ja norjalainen juo vettä.

Koodien lukumäärän laskeminen klikkimenetelmällä

Tässä ongelmassa olemme kiinnostuneita ratkaisujen *lukumäärästä*. l:n pituinen binäärinen koodisana on l:n pituinen jono nollia ja ykkösiä. 01001 voisi olla viiden pituinen koodisana. s:n kokoinen koodi on s:n kokoinen joukko koodisanoja. Kahden binäärisen koodisanan välinen Hamming-etäisyys on se bittien lukumäärä, jolla koodisanat eroavat toisistaan. Esimerkiksi koodisanojen 01001 ja 01010 Hamming etäisyys on kaksi, koska ne eroavat toisistaan kahden bitin osalta. Hamming-etäisyydellä on merkitystä virheen havaitsevissa ja -korjaavissa koodeissa. Tässä ongelmassa olemme kiinnostuneita laskemaan l-pituisten, s-kokoisten ja minimi Hamming-etäisyyden h omaavien eri koodien lukumäärän. Graafin solmut muodostetaan kaikista erilaisista l-pituisista bittijonoista. Graafissa on siten 2^{l-1} solmua. Kahden solmun välille muodostetaan kaari, mikäli niiden Hamming-etäisyys on suurempi tai yhtäsuuri kuin h. Koodien lukumäärä saadaan nyt s-kokoisten klikkien lukumääränä.

Johtopäätökset

Tässä artikkelissa esitettiin, kuinka klikkimenetelmää voidaan käyttää tietyntyyppisten kombinatoristen ongelmien ratkaisujen etsimiseen sekä ratkaisujen lukumäärien laskemiseen. Menetelmää voidaan käyttää usein silloinkin, kun kaikkien vaihtoehtojen läpikäynti tai peräytyvä haku ovat laskenta-ajaltaan liian suuria. Menetelmä edellyttää klikinetsintäalgoritmin käyttöä osana menetelmää.

Viitteet

[1] P. Kaski ja P.R.J. Östergård, *Classification Algorithms for Codes and Designs*, Springer, Berlin, 2006

[2] D.L. Kreher ja D.R. Stinson, *Combinatorial algorithms: generation, enumeration and search*, CRC Press, Boca Raton, Florida, 1999

[3] Ohjelmistotyökalu Cliquer,
`http://users.tkk.fi/pat/cliquer.html`

Proving Simple Algebraic Equations With Prove! 1.0

Mikko Malinen

20th November, 2006

1 Leibniz's Dream

Let's begin this article with a text from [1], translated from the Finnish edition: One of the greatest mathematician's, G. W. Leibniz's "brilliant thought": "he would start to find such special alphabet, for which letters would represent concepts instead of phonemes. In the language based on this kind of alphabet it should be possible to determine only with calculations, which theorems written in this language are true and what kind of logical relations hold between them". As many people know, Leibniz never achieved this dream in spite of a lot of work.

2 Other Programs

There exist programs that can prove theorems. Perhaps the most well known is Otter [2]. It's successor is Prover9 [3]. These are first-order logic theorem provers. Searching through *Journal of Automated Reasoning*, 2003-November,2006 gave one program which can do some kind of proving - subproofs of mathematical theorems. It's name is EGPY Theorem Proving Environment [4], and it is intended for students to construct mathematical proofs. The program itself does not make the whole proofs.

3 Prove! 1.0 Program

3.1 Background

In 2003, I programmed a program which proves simple algebraic theorems. The structure of the program was my idea, and the program is quite complicated relative to what it can prove. But the idea can be used in the possible next versions of the program. Prove! 1.0 can prove theorems of the form

$$a_1 + a_2 + ... + a_n = a_{n+1},$$

where a_i:s are constant integers.

3.2 Structure of the program

Prove! 1.0 has implementation of the Proof System Method first explained in [5]. The program has influence from the structure of the Microsoft Windows operating system and it's message queue. In Prove! 1.0 there is also a queue, the expression queue.

When running the program, the user gives two expressions, of the form $a_1+a_2+ ... + a_n$ and a_{n+1}. The first is appended as the first element of the expression queue. The expressions are taken from the queue, one at a time, and rule 1 is applied to the expression. All expressions that we get when applying the rule are appended to the end of the queue. The rule 1 finds all possible additions of subsequent two constants and makes the addition. An expression $1+2+4$ would yield $3+4$ for example. When an expression which is similar to a_{n+1} is got, the program prints the step-by-step proof. If the rule 1 can no more be applied and the proof has not been found, the program prints "Proof not found".

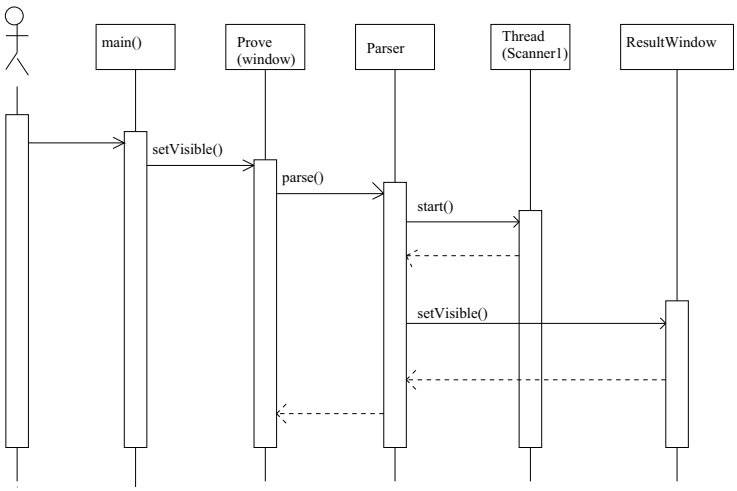


Figure 1. An UML-like interaction diagram for a typical scenario of using the program

4 Completeness and Soundness

By saying that a proof system is *complete* we mean that every true theorem is provable. By saying that a proof system is *sound*, we mean that every provable theorem is true.

Theorem 4.1 *"Prove! 1.0" proof system is complete.*

Proof. In Prove! 1.0 theorems to be proved are of the form $a_1 + a_2 + ... + a_n =$

a_{n+1}, where a_i:s are constant integers. The only rule in the program is equality preserving. By applying this only rule to the left hand side of the equation applying it again to the obtained expression and so on, we at last get a single integer, which is equal to a_{n+1} if the original equation is true. □

Theorem 4.2 *"Prove! 1.0" proof system is sound.*

Proof. The only rule in the program is equality preserving. All expressions added to the expression queue are obtained by the only rule. The right hand side of the equality proved will be found from the expression queue. □

5 Decidability

Theorem 5.1 *Prove 1.0 decides whether* $a_1 + a_2 + ... + a_n = a_{n+1}$ *or not.*

Prrof. The expressions added to the expression queue are equivalent with $a_1 + a_2 + ... + a_n$. At last a single constant is got as an expression and the rule 1 can not be applied any more. Then the proof is found or there is no proof for the theorem. □

6 A Sample Run

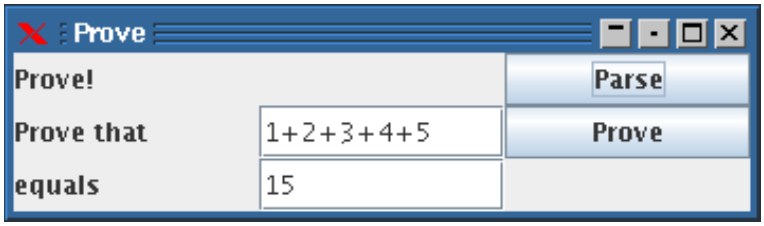


Figure 2. The main window.

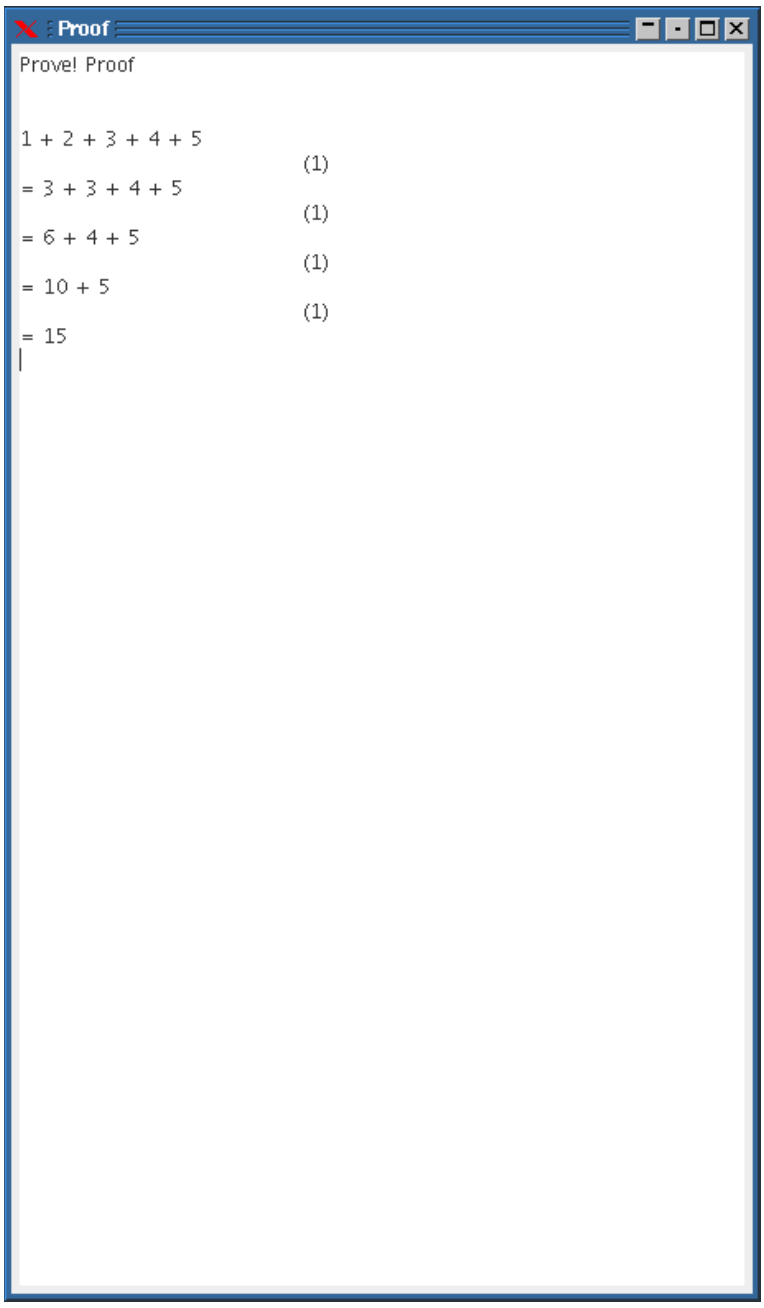


Figure 3. The result window. Text "(1)" means that the statement is obtained by using rule number 1.

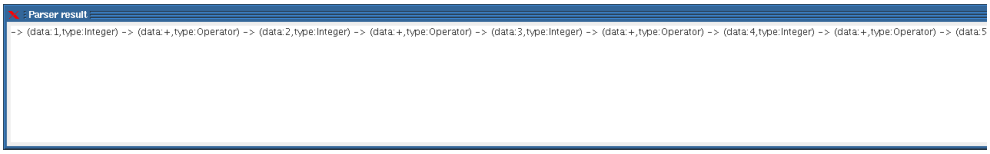


Figure 4. A part of the window that shows the expression parsed.

7 References

[1] Martin Davis, *"Tietokoneen esihistoria Leibnizista Turingiin"*, Art House, Helsinki, 2003. The original book *"The Universal Computer. The Road from Leibniz to Turing."*, W. W. Norton & Company, New York, 2000.

[2] William McCune, Otter 3.0 reference manual and guide, Technical Report ANL-94/6, Argonne National Laboratory, January 1994.

[3] Prover9 page, http://www.cs.unm.edu/∼mccune/prover9/

[4] Richard Sommer and Gregory Nuckols, "A Proof Environment for Teaching Mathematics", *Journal of Automated Reasoning*, **32:** 227-258, 2004

[5] Mikko Malinen, "On Equality Proving of Mathematical Expressions", http://users.tkk.fi/∼mmalinen/pdfdocs/equality.pdf, 2004

Sudokutehtävän ratkaiseminen tietokoneella

Tekn. kand. Mikko Malinen

3. elokuuta 2006

1 Johdanto

Teknillisessä korkeakoulussa järjestettiin graafiteorian kurssi. Kurssin osana ollut ohjelmointiprojekti vaikutti niin mielenkiintoiselta, että tutustuin siihen tarkemmin. Yhtenä tehtävistä oli nimittäin kehittää algoritmi sudokutehtävän ratkaisemiseksi. Suurin osa ohjeista annettiin. Tässä artikkelissa esitetään yksi tapa muodostaa tällainen algoritmi.

2 Latinalaiset neliöt

Astetta n oleva *latinalainen neliö* on $n \times n$ taulukko, joka koostuu n symbolista niin että jokainen symboli esiintyy tasan kerran jokaisella rivillä ja jokaisessa sarakkeessa (ks. kuva 1). Jatkossa oletetaan, että symbolit ovat $1, 2, ..., n$.

1	2	3	4
2	1	4	3
3	4	1	2
4	3	2	1

Kuva 1. Eräs latinalainen neliö

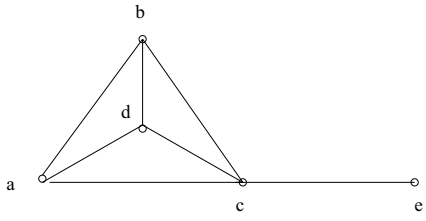
Osittaisessa latinalaisessa neliössä osa alkioista on määrittelemättä. Osittaisen latinalaisen neliön *täydentäminen* on määrittelemättömien alkioiden määrääminen niin, että osittainen neliö täydentyy sallituksi latinalaiseksi neliöksi.

1	2	3	4	5
2	3	.	.	.
3	.	4	.	.
4	.	.	5	.
5	.	.	.	2

Kuva 2. Eräs osittainen latinalainen neliö

3 Hieman graafiteoriaa

Graafi on pari $G = (V, E)$, missä V on solmujen joukko ja E on solmuja yhdistävien kaarien joukko (ks. kuva 3). Yksi kaari yhdistää kaksi solmua toisiinsa.

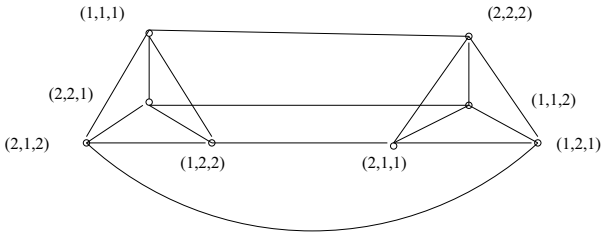


Kuva 3. Eräs graafi. Solmujen joukko V = {a,b,c,d,e}. Solmuja yhdistää joukko kaaria.

Klikki on sellainen V:n osajoukko, jossa jokainen solmu on yhdistetty jokaiseen toiseen solmuun kaarella. Kuvan 3 graafin maksimiklikki on $C_1 = \{a, b, c, d\}$. Myös esim. $C_2 = \{b, c, d\}$ on klikki.

4 Algoritmi

Graafien kielellä osittaisen latinalaisen neliön täydennys voidaan suorittaa seuraavalla tavalla: Tarkastellaan graafin, jonka solmuina on n^3 kpl muotoa (i, j, k) olevaa kolmikkoa; $i, j, k \in \{1, 2, ..., n\}$. Mitkä tahansa kaksi erillistä kolmikkoa on yhdistetty kaarella jos ja vain jos kolmikot ovat yhtäsuuria korkeintaan yhdessä koordinaateista. Kuvassa 4 on arvolla $n = 2$ syntyvä graafi.



Kuva 4. Graafi arvolla $n = 2$

Intuitio on se, että jokainen kolmikko (i, j, k) kertoo, että annamme arvon k alkiolle rivillä i, sarakkeessa j $n \times n$ taulukossa. Huomaa, että n:n asteen latinalaiset neliöt vastaavat n^2 kokoisia graafin klikkejä. Myöskin, osittainen latinalainen neliö voidaan täydentää jos ja vain jos vastaavat kolmikot ilmenevät kokoa n^2 olevassa klikissä.

Algoritmin muokkaaminen sudokun ratkaisemiseksi tapahtuu niin, että sudokun lisärajoitus, luvut 1-9 laatikossa, huomioidaan siten, että kussakin laatikossa alkioiden välillä on kaari jos ja vain jos kyseiset kaksi alkiota eivät ole arvoltaan samat. Ja sudokussahan $n = 9$.

Vaikea osa algoritmista on klikin etsintä graafista. Algoritmeja klikin etsimiseksi on esitetty [1]:ssä ja [2]:ssa. Yksi mahdollisuus on käyttää valmista ohjelmistotyökalua, jollainen on esimerkiksi [3].

5 Pohdintaa

Jopa 1000 euron palkintoiset sudokutehtävät menettävät merkitystään, kun kuka tahansa voi ratkaista tehtävän tietokoneohjelman avulla. Tässä artikkelissa kuvattu algoritmi ratkaissee tavallisen sudokutehtävän, mutta uskoisin, että kaikkien mahdollisten sudokuruudukoiden etsiminen on algoritmille liian työläs tehtävä, sillä niitä on suurin piirtein 6600 triljoonaa. Triljoonassa on 18 nollaa.

6 Viitteet

[1] P. Kaski ja P.R.J. Östergård, *Classification Algorithms for Codes and Designs*, Springer, Berlin, 2006

[2] D.L. Kreher ja D.R. Stinson, *Combinatorial algorithms: generation, enumeration and search*, CRC Press, Boca Raton, Florida, 1999

[3] Ohjelmistotyökalu Cliquer, http://users.tkk.fi/pat/cliquer.html

Markovin ketjut ja pokerin todennäköisyydet

Tekn. kand. Mikko Malinen

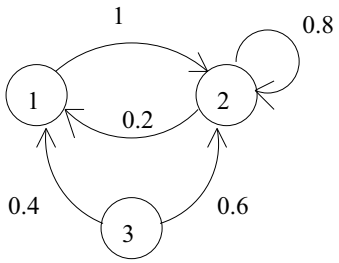
3. elokuuta 2006

1 Johdanto

Markovin ketjuilla on monia sovelluksia. Yksi keksimäni sovellus on pokerikäsien todennäköisyyksien laskenta. Voidaan esimerkiksi haluta tietää, mikä todennäköisyys on saada neljä samanarvoista korttia, kun kahden jaetun kortin jälkeen kädessä on pari. Esitettävällä menetelmällä saadaan tiettyjen käsien todennäköisyys, kun lähtötilanteessa on jaettu 0..5 korttia ja näillä korteilla on jo saatu tietty käsi.

2 Markovin ketjut

Diskreettiaikainen Markovin ketju [1] muodostuu joukosta tiloja sekä eri tilojen välillä vallitsevista siirtymätodennäköisyyksistä. Kustakin tilasta lähtevien siirtymätodennäköisyyksien summa on 1 eli jokaisella askeleella siirrytään johonkin tilaan (ks. kuva 1).



Kuva 1. Eräs Markovin ketju, jonka tilat ovat {1,2,3}.

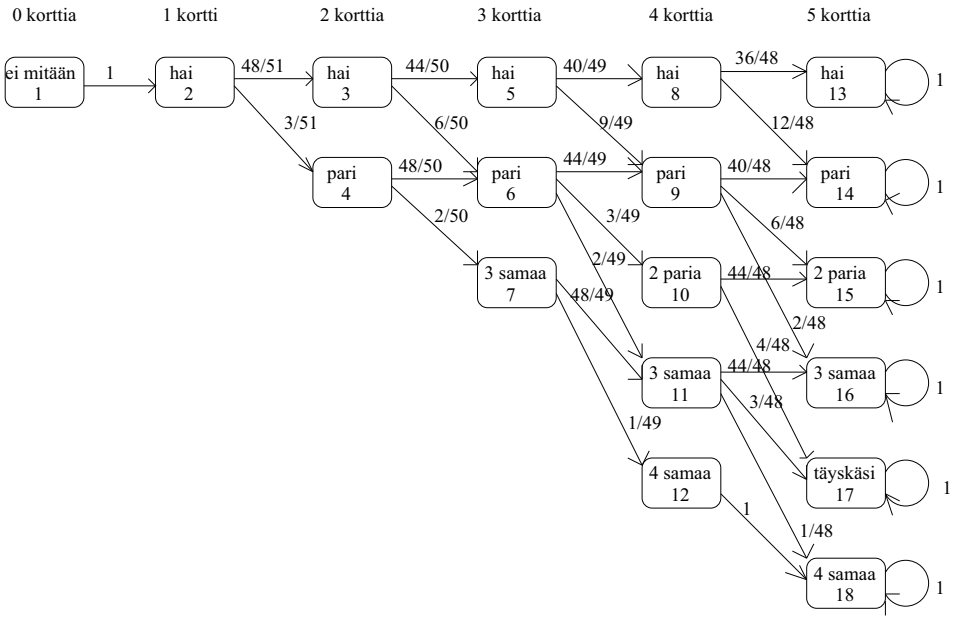
Kuvan 1 ketjun yhden askeleen tilasiirtymämatriisi on

$$P = \begin{pmatrix} 0 & 1 & 0 \\ 0.2 & 0.8 & 0 \\ 0.4 & 0.6 & 0 \end{pmatrix}$$

Rivillä 3 sarakkeessa 2 on luku 0.6, mikä tarkoittaa, että yhdellä askeleella tilasta 3 siirtymätodennäköisyys tilaan 2 on 0.6. Kun P korotetaan n:nteen potenssiin voidaan vastaavasti lukea riviltä 3 sarakkeesta 2 todennäköisyys, että n:n askeleen kuluttua ollaan tilassa 2, kun lähdetään liikkeelle tilasta 3. Matriisin kertolasku, jota potenssiin korottaminen on, on selitetty lähteessä [2]. Vähänkään isompien matriisien potenssiin korotus on kätevää tehdä Matlab- tai Mathematica-ohjelmistolla.

3 Pokerin todennäköisyyksien laskeminen

Muodostetaan Markovin ketju:



Kuva 2. Pokerikädet Markovin ketjuna, tilanumerot merkitty

Siirtymätodennäköisyys esimerkiksi tilasta 2 tilaan 4 on 3/51 ja se saadaan seuraavasti: pakassa on jäljellä 3 sellaista korttia, joilla hai voi korottua pariksi ja pakan korttien yhteismäärä on 51. Huomaa, että tiloista 13..18 on siirtymä itseensä, jotta kyseessä olisi Markovin ketju ja kustakin tilasta siirryttäisiin to-

dennäköisyydellä 1. Tilasiirtymämatriisi on

$$P = \begin{pmatrix}
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & \frac{48}{51} & \frac{3}{51} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{44}{50} & \frac{6}{50} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \frac{48}{50} & \frac{2}{50} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{40}{49} & \frac{9}{49} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{44}{49} & \frac{3}{49} & \frac{2}{49} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{48}{49} & \frac{1}{49} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{36}{48} & \frac{12}{48} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{40}{48} & \frac{6}{48} & \frac{2}{48} & & \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{44}{48} & 0 & \frac{4}{48} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \frac{44}{48} & \frac{3}{48} & \frac{1}{48} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{pmatrix}$$

Kun halutaan laskea eri käsien todennäköisyydet viiden jaetun kortin jälkeen, kun lähdetään tyhjästä kädestä liikkeelle (tila 1), niin lasketaan P^5 (tässä neljän desimaalin tarkkuudella)

$$P^5 = \begin{pmatrix}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.5071 & 0.4226 & 0.0475 & 0.0211 & 0.0014 & 0.0002 \\
\cdots & & & & & & & & & & & & & & & & & \\
\cdots & & & & & & & & & & & & & & & & &
\end{pmatrix}$$

Tässä vain ensimmäinen rivi on kirjoitettuna.

Esimerkiksi kahden parin todennäköisyyden (tila 15) kertoo rivin 1 (tilasta 1 lähdettiin liikkeelle) sarake 15. Jos haluttaisiin tietää esim. kahden parin todennäköisyys neljän jaetun kortin jälkeen, kun kahden jaetun kortin jälkeen kädessä on pari, pitäisi laskea P^2 (toiseen potenssiin siksi, että jaetaan kaksi korttia lisää) ja tutkia mikä luku on matriisin rivillä 4 sarakkeessa 10. P^2:n, P^3:n ja P^4:n laskemisen jätän lukijalle.

4 Viitteet

[1] R. Durrett, *Essentials of Stochastic Processes*, Springer, New York, 1999

[2] E. Kreyszig, *Advanced engineering mathematics*, 8.laitos, Wiley, Singapore, 1999

On comparison of chess programs

Mikko Malinen

27th July, 2006

Abstract

This paper proves that if a chess program A wins a chess program B, this does not prove that A is necessarily a better chess program than B against other chess programs.

1 Introduction

When a new chess program comes available, it is often compared to some other available program. Also, when chess program developers develop a new version of a chess program, the strength of that program is compared to the previous version. This paper proves that it is necessary to compare the new program to many other programs or many older versions to get a reliable picture of the strength of the program.

2 The Proof

Let's consider three chess programs A, B and C. The programs have the same strategy. Following this strategy black always wins white. One such strategy is the following: white starts with the move f2f3. Black follows with e7e6. The following moves are g2g4 d8h4, black wins. When A is white and B is black, B wins. When B is white and C is black, C wins. The winning relation graph is in figure 1. One might consider that the winning relation is transitive and in the next game C wins A.

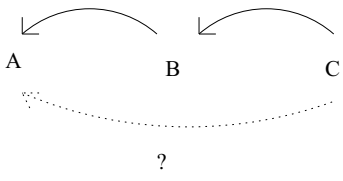


Figure 1. The winning relation after two games

But, when C is white and A is black, A wins. This means that the winning relation is not transitive. So, if B wins A, it does not necessarily mean that B is better than A against other opponents.

3 Conclusions

From this result we may infer that in tournaments it is important to play against every other chess program. It may be, however, that when two programs play, the one that wins is actually a better program. It may be that both programs use minimax-search and the winning program has simply a deeper search tree. However, playing against all other programs gives more reliable results, so that the program that wins the tournament is really the best program. Also, when developing chess programs, it may be important to compare a new version of the program not only to the previous version but the older versions too.

Applications of Boolean and First-order Logic

Mikko Malinen

31st August, 2005

Abstract

This article presents applications of Boolean and first-order logic, i.e. how logic is used in solving certain problems. Some examples list an input file to theorem prover Otter. Prerequisite knowledge of Boolean and first-order logic is assumed, although the languages of Boolean and first-order logic are defined.

1 The Language of Propositional Logic (Boolean Logic)

This section is from [1]:

Propositional formulae (or propositions) are strings of symbols from a countable alphabet defined below, and formed according to certain rules stated in definition (1.2).

Definition 1.1 *(The alphabet for propositional formulae) This alphabet consists of:*
(1) A countable set $\boldsymbol{PS}$ *of proposition symbols:* $P_0, P_1, P_2...$*;*
(2) The logical commectives: $\wedge$ *(and),*$\vee$ *(or),*$\Rightarrow$ *(implication),* $\neg$ *(not) and sometimes* $\Leftrightarrow$ *(equivalence) and the constant* $\perp$ *(false);*
(3) Auxiliary symbols: "(" (left parenthesis), ")" (right parenthesis).

The set *PROP* of propositional formulae (or propositions) is defined as the inductive closure of a certain subset of the alphabet of definition (1.1) under certain operations defined below.

Definition 1.2 *Propositional formulae. The set PROP or propositional formulae (or propositions) is the inductive closure of the set* $\boldsymbol{PS} \cup \{\perp\}$ *under the functions* $C_\neg$,$C_\wedge$,$C_\vee$,$C_\Rightarrow$ *and* $C_\Leftrightarrow$*, defined as follows: For any two strings A,B*

*over the alphabet of definition (*1.1*),*

$$C_{\neg}(A) = \neg A,$$
$$C_{\wedge}(A, B) = (A \wedge B),$$
$$C_{\vee}(A, B) = (A \vee B),$$
$$C_{\Rightarrow}(A, B) = (A \Rightarrow B) \quad \textit{and}$$
$$C_{\Leftrightarrow}(A, B) = (A \Leftrightarrow B).$$

The above definition is the official definition of *PROP* as an inductive closure, but is a bit formal. For that reason, it is often stated less formally as follows: The set *PROP* of propositions is the smallest set of strings over the alphabet of definition (1.1), such that:
(1) Every proposition symbol P_i is in *PROP* and $\bot$ is in *PROP*;
(2) Whenever A is in *PROP*, $\neg A$ is also in *PROP*;
(3) Whenever A,B are in *PROP*,$(A \vee B)$,$(A \wedge B)$,$(A \Rightarrow B)$ and $(A \Leftrightarrow B)$ are also in *PROP*;
(4) A string is in *PROP* only if it is formed by applying the rules (1),(2),(3).

2 First-order Languages

This section is almost entirely from [3]:

We recall what a first-order language is, a notion essentially due to G.Frege. By necessity our treatment is reduced to a list of definitions. A *first-order language* consists of an alphabet and all formulas defined over it. An *alphabet* consists of the following classes of symbols:

- *variables* denoted by *x,y,z,v,u,...,*
- *constants* denoted by *a,b,c,d,...,*
- *function symbols* denoted by *f,g,...,*
- *relation symbols* denoted by *p,q,r,...* or *P,Q,R,...,*
- *propositional constants*, which are **true** and **false**,
- *connectives*, which are $\neg$ (negation), $\vee$ (disjunction), $\wedge$ (conjunction), $\Rightarrow$ (implication) and $\Leftrightarrow$ (equivalence)
- *quantifiers*, which are $\exists$ (there exists) and $\forall$ (for all),
- *parentheses*, which are (and) and the *comma*, that is:,.

Thus the sets of connectives, quantifiers and parentheses is fixed. We assume also that the set of variables is infinite and fixed. Those classes of symbols are called *logical symbols*. The other classes of symbols, that is, constants, relation symbols (or just *relations*) and function symbols (or just *functions*), may vary

and in particular may be empty. They are called *nonlogical symbols.* Each first-order language is thus determined by its nonlogical symbols.
Each function and relation symbol has a fixed *arity*, that is, the number of arguments. We assume that functions have a positive arity - the role of 0-ary functions is played by the constants. In contrast, 0-ary relations are admitted. They are called *propositional symbols*, or simply *propositions.* Note that each alphabet is uniquely determined by its constants, functions and relations.
We now define by induction two classes of strings of symbols over a given alphabet. First we define the class of *terms* as follows:

- a variable is a term
- a constant is a term
- if f is an n-ary function and $t_1, ..., t_n$ are terms then $f(t_1, ..., t_n)$ is a term

Terms are denoted by s, t, u. Finally, we define the class of *formulas* as follows:

- if p is an n-ary relation and $t_1, ..., t_n$ are terms then $p(t_1, ..., t_n)$ is a formula called *atomic formula*, or just an *atom*,
- **true** and **false** are formulas,
- if F and G are formulas then so are $\neg F, (F \wedge G), (F \vee G), (F \Rightarrow G)$ and $(F \Leftrightarrow G)$,
- if F is a formula and x is a variable then $\exists x F$ and $\forall x F$ are formulas.

Sometimes we shall write $(G \Leftarrow F)$ instead of $(F \Rightarrow G)$. Some well known binary functions (like $+$) or relations (like $=$) are usually written in *infix notation* i.e. between the arguments. Atomic formulas are denoted by A, B and formulas in general by F, G. If F is a quantifier-free formula with variables $x_1, ..., x_n$ we write $\exists F$ for $\exists x_1 ... \exists x_n F$ and $\forall F$ for $\forall x_1 ... \forall x_n F$. Formulas of the form $\forall F$ are called *universal formulas.* A term or formula with no variables is called *ground.* Given two strings of symbols e_1 and e_2 from the alphabet, we write $e_1 \equiv e_2$ when e_1 and e_2 are identical. Usually these strins will be terms or formulas.
The definition of formulas is rigorous at the expence of excessive use of parentheses. One way to eliminate most of them is by introducing a *binding order* among the connectives and quantifiers. We thus assume that $\neg, \exists$ and $\forall$ bind stronger than $\vee$ which in turn binds stronger than $\wedge$ which binds stronger than $\Rightarrow$ and $\Leftrightarrow$. Also, we assume that $\vee, \wedge, \Rightarrow$ and $\Leftrightarrow$ *associate to the right* and omit the outer parentheses. Thus, thanks to the binding order, we can rewrite the formula

$$\forall y \forall x((p(x) \wedge \neg r(y)) \Rightarrow (\neg q(x) \vee (A \vee B)))$$

as

$$\forall y \forall x(p(x) \wedge \neg r(y) \Rightarrow \neg q(x) \vee (A \vee B))$$

which, thanks to the convention of the association to the right, further simplifies to

$$\forall y \forall x(p(x) \wedge \neg r(y) \Rightarrow \neg q(x) \vee A \vee B).$$

This completes the definition of a first-order language.

3 About Proof Systems

This section is almost entirely from [1]:

Every logical system consists of a *language* used to write statements called *propositions* or *formulae*. Normally, when one writes a formula, one has some intended *interpretation* of this formula in mind. For example, a formula may assert a true property about the natural numbers, or some property that must be true in a database. This implies that a formula has a well-defined *meaning* or *semantics*. But how do we define this meaning precisely? In logic, we usually define the meaning of a formula as its *truth value*. A formula can be either true (or valid) or false.
Defining rigorously the notion of truth is actually not as obvious as it appears. We shall present a concept of truth due to Tarski. Roughly speaking, a formula is true if it is satisfied in all possible interpretations.
The next question is to investigate whether it is possible to find methods for deciding in a finite number of steps whether a formula is true (or valid). This is a very difficult task. In fact, by a theorem due to Church, there is no such general method for first-order logic.
However, there is another familiar method for testing whether a formula is true: to give a *proof* of this formula.
Of course, to be of any value, a proof system should be *sound*, which means that every provable formula is true.
The branch of logic concerned with the study of proof is known as *proof theory*.
Now, if we have a sound proof system, we know that every provable formula is true. Is the proof system strong enough that it is also possible to prove every true formula (of first-order logic) ?
A major theorem of Gödel shows that there are logical proof systems in which every true formula is provable. This is referred to as the *completeness* of the proof system.
To summarize the situation, if one is interested in algorithmic methods for testing whether of first-order logic is valid, there are two logical results of central importance: one positive (Gödel's completeness theorem), the other one negative (Church's undecidability of validity). Roughly speaking, Gödel's completeness theorem asserts that there are logical calculi in which every true formula is provable, and Church's theorem asserts that there is no decicion procedure (procedure which always terminates) for deciding whether a formula is true (valid). Hence, any algorithmic procedure for testing whether a formula is true (or equivalently, by Gödel's completeness theorem, provable in a complete system) must run forever when given certain non-true formulae as input.
Propositions are much simpler than first-order formulae. Indeed, there are algorithms for deciding truth of propositional formulae.
Different proof systems are Hilbert-style proof system [2][9][10], Gentzen-like sequent calculi [1], Tableau method [10][11] and resolution method [1][2][10][11].

4 Some Graph Theory

Definition 4.1 *A **Hamiltonian graph** is a graph with a spanning cycle, also called a **Hamiltonian cycle**.* [7]

Definition 4.2 *A **drawing** of a graph G is a function f defined on $V(G) \cup E(G)$ that assigns each vertex v a point $f(v)$ in the plane and assigns each edge with endpoints u, v a polygonal $f(u), f(v)$-curve. The images of vertices are distinct. A point in $f(e) \cap f(e')$ that is not a common endpoint is a **crossing**.* [7]

Definition 4.3 *A graph is **planar** if it has a drawing without crossings.* [7]

5 Applications of Boolean Logic

5.1 A Chess Related Problem

Description of the problem:
If a knight is placed at the lower left corner of the chess board, can all chess squares be visited using rules of movement for knight? One variation of this problem is called *Knight's Tour* problem. In knights tour every square may be visited only once. It turns out that there is a solution, i.e. all chess squares can be visited. There exists solutions to even knight's tour.
In this presentation we show how the problem may be solved with theorem prover Otter [6]. In this solution any number of visits to chess squares are allowed.
In the next subsection you may find listing of file knights.in. Here we explain the contents of this file. In section A of the file there is a line "Ra1.". This is a proposition which is interpreted as "Square a1 is reachable". Square a1 is reachable because there we put the knight initially. In section B of the file we put the definitions. In the beginning of section B there is line "Ra1 -> (Rb3 & Rc2).". This is interpreted as "if square a1 is reachable then squares b3 and c2 are reachable. This is where we write the moving possibilities from each square. The other rows are written in the same way. In section C we put the actual query (it's negation). Here we ask if all the squares are reachable.

5.1.1 The file knights.in

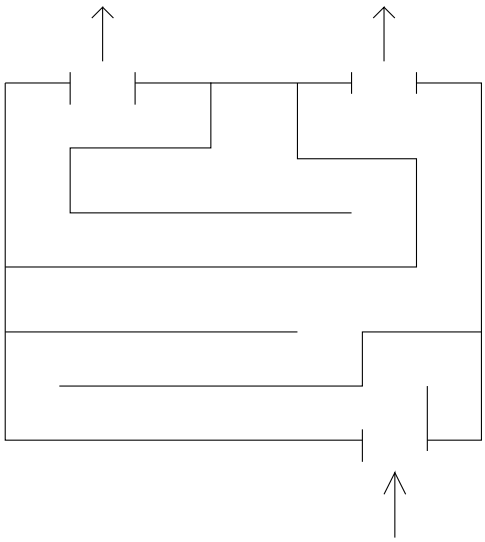
```
set(auto).
formula_list(usable).
% Section A: database
Ra1.

% Section B: definitions
Ra1 -> (Rb3 & Rc2).
Rb1 -> (Ra3 & Rc3 & Rd2).
```

```
Rc1 -> (Ra2 & Rb3 & Rd3 & Re2).
Rd1 -> (Rb2 & Rc3 & Re3 & Rf2).
Re1 -> (Rc2 & Rd3 & Rf3 & Rg2).
Rf1 -> (Rd2 & Re3 & Rg3 & Rh2).
Rg1 -> (Re2 & Rf3 & Rh3).
Rh1 -> (Rf2 & Rg3).

Ra2 -> (Rb4 & Rc3 & Rc1).
Rb2 -> (Ra4 & Rc4 & Rd3 & Rd1).
Rc2 -> (Ra1 & Ra3 & Rb4 & Rd4 & Re3 & Re1).
Rd2 -> (Rb1 & Rb3 & Rc4 & Re4 & Rf3 & Rf1).
Re2 -> (Rc1 & Rc3 & Rd4 & Rf4 & Rg3 & Rg1).
Rf2 -> (Rd1 & Rd3 & Re4 & Rg4 & Rh3 & Rh1).
Rg2 -> (Re1 & Re3 & Rf4 & Rh4).
Rh2 -> (Rf1 & Rf3 & Rg4).

Ra3 -> (Rb5 & Rc4 & Rc2 & Rb1).
Rb3 -> (Ra5 & Rc5 & Rd4 & Rd2 & Rc1 & Ra1).
Rc3 -> (Ra4 & Rb5 & Rd5 & Re4 & Re2 & Rd1 & Rb1 & Ra2).
Rd3 -> (Rb4 & Rc5 & Re5 & Rf4 & Rf2 & Re1 & Rc1 & Rb2).
Re3 -> (Rc4 & Rd5 & Rf5 & Rg4 & Rg2 & Rf1 & Rd1 & Rc2).
Rf3 -> (Rd4 & Re5 & Rg5 & Rh4 & Rh2 & Rg1 & Re1 & Rd2).
Rg3 -> (Re4 & Rf5 & Rh5 & Rh1 & Rf1 & Re2).
Rh3 -> (Rf4 & Rg5 & Rg1 & Rf2).

Ra4 -> (Rb6 & Rc5 & Rc3 & Rb2).
Rb4 -> (Ra6 & Rc6 & Rd5 & Rd3 & Rc2 & Ra2).
Rc4 -> (Ra5 & Rb6 & Rd6 & Re5 & Re3 & Rd2 & Rb2 & Ra3).
Rd4 -> (Rb5 & Rc6 & Re6 & Rf5 & Rf3 & Re2 & Rc2 & Rb3).
Re4 -> (Rc5 & Rd6 & Rf6 & Rg5 & Rg3 & Rf2 & Rd2 & Rc3).
Rf4 -> (Rd5 & Re6 & Rg6 & Rh5 & Rh3 & Rg2 & Re2 & Rd3).
Rg4 -> (Re5 & Rf6 & Rh6 & Rh2 & Rf2 & Re3).
Rh4 -> (Rf5 & Rg6 & Rg2 & Rf3).

Ra5 -> (Rb6 & Rc6 & Rc4 & Rb3).
Rb5 -> (Ra7 & Rc7 & Rd6 & Rd4 & Rc3 & Ra3).
Rc5 -> (Ra6 & Rb7 & Rd7 & Re6 & Re4 & Rd3 & Rb3 & Ra4).
Rd5 -> (Rb6 & Rc7 & Re7 & Rf6 & Rf4 & Re3 & Rc3 & Rb4).
Re5 -> (Rc6 & Rd7 & Rf7 & Rg6 & Rg4 & Rf3 & Rd3 & Rc4).
Rf5 -> (Rd6 & Re7 & Rg7 & Rh6 & Rh4 & Rg3 & Re3 & Rf4).
Rg5 -> (Re6 & Rf7 & Rh7 & Rh3 & Rf3 & Rg4).
Rh5 -> (Rf6 & Rg7 & Rg3 & Rh4).

Ra6 -> (Rb8 & Rc7 & Rc5 & Rb4).
Rb6 -> (Ra8 & Rc8 & Rd7 & Rd5 & Rc4 & Ra4).
Rc6 -> (Ra7 & Rb8 & Rd8 & Re7 & Re5 & Rd4 & Rb4 & Ra5).
```

```
Rd6 -> (Rb7 & Rc8 & Re8 & Rf7 & Rf5 & Re4 & Rc4 & Rb5).
Re6 -> (Rc7 & Rd8 & Rf8 & Rg7 & Rg5 & Rf4 & Rd4 & Rc5).
Rf6 -> (Rd7 & Re8 & Rg8 & Rh7 & Rh5 & Rg4 & Re4 & Rd5).
Rg6 -> (Re7 & Rf8 & Rh8 & Rh4 & Rf4 & Re5).
Rh6 -> (Rf7 & Rg8 & Rg4 & Rf5).

Ra7 -> (Rc8 & Rc6 & Rb5).
Rb7 -> (Rd8 & Rd6 & Rc5 & Ra5).
Rc7 -> (Ra8 & Re8 & Re6 & Rd5 & Rb5 & Ra6).
Rd7 -> (Rb8 & Rf8 & Rf6 & Re5 & Rc5 & Rb6).
Re7 -> (Rc8 & Rg8 & Rg6 & Rf5 & Rd5 & Rc6).
Rf7 -> (Rd8 & Rh8 & Rh6 & Rg5 & Re5 & Rd6).
Rg7 -> (Re8 & Rh5 & Rf5 & Re6).
Rh7 -> (Rf8 & Rg5 & Rf6).

Ra8 -> (Rc7 & Rb6).
Rb8 -> (Rd7 & Rc6 & Ra6).
Rc8 -> (Re7 & Rd6 & Rb6 & Ra7).
Rd8 -> (Rf7 & Re6 & Rc6 & Rb7).
Re8 -> (Rg7 & Rf6 & Rd6 & Rc7).
Rf8 -> (Rh7 & Rg6 & Re6 & Rd7).
Rg8 -> (Rh6 & Rf6 & Re7).
Rh8 -> (Rg6 & Rf7).

%Section C: negation of the query
-(Ra1 & Rb1 & Rc1 & Rd1 & Re1 & Rf1 & Rg1 & Rh1 & Ra2 & Rb2 & Rc2 & Rd2 & Re2
& Rf2 & Rg2 & Rh2 & Ra3 & Rb3 & Rc3 & Rd3 & Re3 & Rf3 & Rg3 & Rh3 & Ra4 & Rb4
& Rc4 & Rd4 & Re4 & Rf4 & Rg4 & Rh4 & Ra5 & Rb5 & Rc5 & Rd5 & Re5 & Rf5 & Rg5
& Rh5 & Ra6 & Rb6 & Rc6 & Rd6 & Re6 & Rf6 & Rg6 & Rh6 & Ra7 & Rb7 & Rc7 & Rd7
& Re7 & Rf7 & Rg7 & Rh7 & Ra8 & Rb8 & Rc8 & Rd8 & Re8 & Rf8 & Rg8 & Rh8).

end_of_list.
```

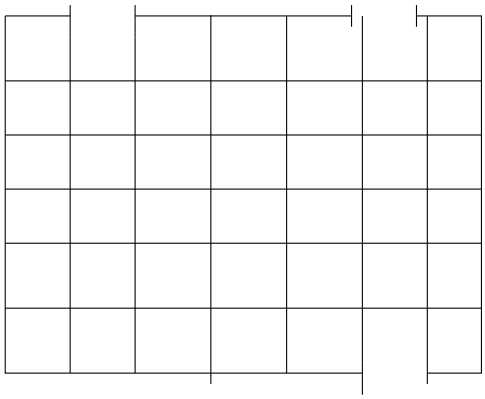
6 A Labyrinth Problem

Otter can be used to prove that there is a way out from a labyrinth. An example labyrinth is in picture 1.



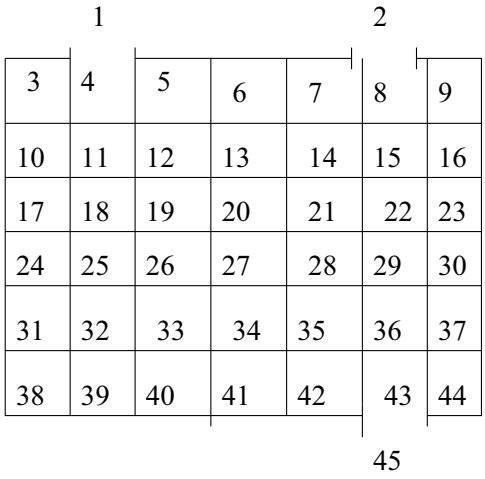
Picture 1. An example labyrinth.

The solving technique proceeds as follows.
1) Generating the squares. The walls of the squares must go along the walls of the labyrinth.



Picture 2. Generating the squares.

2) Numbering the squares



Picture 3. Numbering the squares.

3) Writing the input file. The input file labyrin.in is in the following subsection. Here we explain the contents of this file. Section A of the file has line "R45". This is interpreted as "Square (position) 45 is reachable". This is our entry position. Section B of the file has the definitions. For example line "R3 -> (R4 & R10)." is interpreted as "if square 3 is reachable then squares 4 and 10 are reachable. Here we need to define reachability only to the neighboring squares. In section C of the file we write the query (it's negation). With "-R2" we ask to prove that square (position) 2 is reachable.

6.0.2 The file labyrin.in

```
set(auto).
formula_list(usable).
% Section A: database
R45.

% Section B: definitions
R1 -> (R4).
R2 -> (R8).
R3 -> (R4 & R10).
R4 -> (R1 & R3 & R5).
R5 -> (R4).
R6 -> (R13).
R7 -> (R8).
R8 -> (R2 & R7 & R9).
R9 -> (R8 & R16).
R10 -> (R3 & R17).
```

```
R11 -> (R12).
R12 -> (R11 & R13).
R13 -> (R6 & R12 & R14).
R14 -> (R13 & R15).
R15 -> (R14 & R22).
R16 -> (R9 & R23).
R17 -> (R10 & R18).
R18 -> (R17 & R19).
R19 -> (R18 & R20).
R20 -> (R19 & R21).
R21 -> (R20 & R22).
R22 -> (R15 & R21).
R23 -> (R16 & R30).
R24 -> (R25).
R25 -> (R24 & R26).
R26 -> (R25 & R27).
R27 -> (R26 & R28).
R28 -> (R27 & R29 & R35).
R29 -> (R28 & R30).
R30 -> (R23 & R29).
R31 -> (R32 & R38).
R32 -> (R31 & R33).
R33 -> (R32 & R34).
R34 -> (R33 & R35).
R35 -> (R28 & R34).
R36 -> (R37 & R43).
R37 -> (R36 & R44).
R38 -> (R31 & R39).
R39 -> (R38 & R40).
R40 -> (R39 & R41).
R41 -> (R40 & R42).
R42 -> (R41 & R43).
R43 -> (R36 & R42).
R44 -> (R37).
R45 -> (R43).

%Section C: negation of the query
-R2.

end_of_list.
```

7 Applications of First-order Logic

7.1 About Graph Properties

Let A be a finite alphabet. We denote by $\mathbf{D}(A)$ the class of finite or infinite directed graphs, the edges of which are labelled with labels from A. Here are a few examples [4] of properties of a graph G in the class $\mathbf{D}(A)$ that are first-order expresible:

- G is simple;
- G is k-regular for some fixed integer k;
- G is of degree at most k for some fixed integer k;
- G is edge-colored by A, i.e., any two edges of G having the same label have no vertex in common.

The following properties are not expressible in first-order logic:

- G is connected
- G is planar
- G is Hamiltonian.

Theorem 7.1 *A graph property is in the class P if it is first-order.*[4]

P and other compexity classes are discussed in [8][9].

7.2 Formalizing Some Graph and Relation Properties

- Reflexivity
$$\forall x(G(x,x))$$
where $G(x,y)$ is read "there is an edge from x to y".
- Symmetricity
$$\forall x \forall y(G(x,y) \Rightarrow G(y,x))$$
- Transitivity
$$\forall x \forall y \forall z((G(x,y) \wedge G(y,z)) \Rightarrow G(x,z))$$
- The out-degree of a graph is at most 2:

  The formula $\phi(x)$ defined as:

  $\forall y_1, y_2, y_3[edg(x,y_1) \wedge edg(x,y_2) \wedge edg(x,y_3) \Rightarrow y_1 = y_2 \vee y_1 = y_3 \vee y_2 = y_3]$

  expresses that the vertex x of the represented graph has out-degree at most 2. The closed formula $\forall x \phi(x)$ expresses thus that the considered graph has outdegree at most 2.

7.3 Boolean Circuits

This application of first-order logic is from [12].

A Boolean circuit consists of a set of gates that connect the inputs of the circuit to the outputs. Three types of gates are considered here: and-gates, or-gates and inverters. These gates implement the respective logical connectives (i.e. conjunction, disjunction and negation), but gates handle Boolean values 0 and 1 rather than truth values.
Any Boolean circuit can be described using the following symbols:

- Any set of constants can be introduced as names for the input and output points. Some of these points appear as intermediary points between gates that connect outputs of gates to inputs of others.
- constants 0 and 1 refer to the two Boolean values that a point may possess.
- Predicate $Value(p, v)$: the value of a point p in the circuit is v.
- Predicate $And(x, y, z)$: there is an and-gate in the circuit that connects inputs x and y to an output z. It should be clear that the value of z is 1 only when both x and y have the value 1.
- Predicate $Or(x, y, z)$: there is an or-gate in the circuit that connects inputs x and y to an output z.
- Predicate $Inv(x, y)$: there is an inverter in the circuit that connects an input x to an output y.

Given these, our task was to write a definition for the predicate $Value$ that allows computing the value of the output of the whole circuit given the structure of the circuit as well as values of its inputs. We had to use a language based only on the symbols mentioned above.
As an example, consider a simple circuit with three gates, one of each type; i.e. an and-gate, or-gate and inverter. Suppose that the structure of this circuit is the following: the and-gate connects inputs a and b to output d, the or-gate connects inputs d and e to output f, and the inverter connects input c to output e. In terms of the given predicates, this is expressed as $And(a, b, d)$, $Or(d, e, f)$ and $Inv(c, e)$. Furthermore, suppose that the input values of the circuit are such that the point a is 0 and the points b and c are 1. This is expressed as $Value(a, 0)$, $Value(b, 1)$ and $Value(c, 1)$. Then, the definitions of the value predicate should allow us to infer that e.g. the value of the point d is 0, which is expressed as $Value(d, 0)$.
The definitions should work for arbitrary circuits (without loops) and arbitrary values for inputs.
The input file for Otter is composed of the following sections:

- Section A: description of a particular circuit and the values of inputs given in terms of predicates And,Or,Inv and $Value$.

- Section B: definition of the predicate $Value$.
- Section C: a query involving the $Value$ predicate

7.3.1 The input file for Otter

```
set(auto).
formula_list(usable).

%section A (database)

And(a,b,d).
Or(d,e,f).
Inv(c,e).
Value(a,0).
Value(b,1).
Value(c,1).


%section B (definitions)

all x y z (Value(x,1) & Value(y,1) & And(x,y,z) -> Value(z,1)).
all x y z ( (Value(x,0) | Value(y,0)) & And(x,y,z) -> Value(z,0)).
all x y z ((Value(x,1) | Value(y,1)) & Or(x,y,z) -> Value(z,1)).
all x y z ((Value(x,0) & Value(y,0)) & Or(x,y,z) -> Value(z,0)).
all x y (Value(x,0) & Inv(x,y) -> Value(y,1)).
all x y (Value(x,1) & Inv(x,y) -> Value(y,0)).


%section C (negation of the query)

-(Value(f,0)).
```

7.4 "There Is No Male Barber"

This application of first-order logic is from [13].

An assignment: Prove by resolution, that there is no male barber, when
a) Every barber shaves beards of those men, who do not shave their own beards.
b) No barber shaves beards of those men, who shave their own beards.

Solution: Let's imagine that the universum consists of a set of men. Let's use the next predicates in formalization: $P(x) =$ "x is a barber" and $A(x, y) =$"x shaves y's beard".
a) $\forall x(P(x) \Rightarrow \forall y(\neg A(y, y) \Rightarrow A(x, y)))$,
b) $\forall x(P(x) \Rightarrow \forall y(A(y, y) \Rightarrow \neg A(x, y)))$,

Let's compose the clauses:

a) $\forall x(P(x) \Rightarrow \forall y(\neg A(y,y) \Rightarrow A(x,y)))$
$\forall x(\neg P(x) \vee \forall y(A(y,y) \vee A(x,y)))$
$\forall x \forall y(\neg P(x) \vee A(y,y) \vee A(x,y))$
$\neg P(x) \vee A(y,y) \vee A(x,y)$
$\{\neg P(x_1), A(y_1,y_1), A(x_1,y_1)\}$

b) $\forall x(P(x) \Rightarrow \forall y(A(y,y) \Rightarrow \neg A(x,y)))$
$\forall x(\neg P(x) \vee \forall y(\neg A(y,y) \vee \neg A(x,y)))$
$\forall x \forall y(\neg P(x) \vee \neg A(y,y) \vee \neg A(x,y))$
$\neg P(x) \vee \neg A(y,y) \vee \neg A(x,y)$
$\{\neg P(x_2), \neg A(y_2,y_2), \neg A(x_2,y_2)\}$

We want to prove $\neg \exists x P(x)$ and therefore we compose the negation of the formula: $\exists x P(x)$. This formula is transformed to clause form $\{P(a)\}$.
From clauses $\{\neg P(x_1), A(y_1,y_1), A(x_1,y_1)\}$ and $\{\neg P(x_2), \neg A(y_2,y_2), \neg A(x_2,y_2)\}$ we get $\{\neg P(x_3)\}$ (substitution $\{x_1/x_3, x_2/x_3, y_1/x_3, y_2/x_3\}$). From clauses $\{P(a)\}$ and $\{\neg P(x_3)\}$ we get an empty clause (substitution $\{x_3/a\}$). So the set of clauses is unsatisfiable and $\neg \exists x P(x)$ follows logically from the premises.

8 References

[1] Jean H. Gallier, *Logic for Computer Science: Foundations of Automated Theorem Proving*, Harper & Row Publishers Inc.,1986

[2] Michael R. Genesereth, Nils J. Nilsson, *Logical Foundations of Artificial Intelligence*, Morgan Kaufman Publishers Inc.,1987

[3] Krzysztof R. Apt, Logic Programming, in J. van Leeuwen, ed., *Handbook of Theoretical Computer Science, Volume B*, Elsevier Science Publishers,1990

[4] Bruno Courcelle, Graph Rewriting: An Algebraic and Logic Approach, in J. van Leeuwen, ed., *Handbook of Theoretical Computer Science, Volume B*, Elsevier Science Publishers,1990

[5] Mikko Malinen, *Examples of Using Theorem Prover Otter*, an article published in Internet,http://users.tkk.fi/∼mmalinen, 2004

[6] *Otter: An Automated Deduction System*
http://www.mcs.anl.gov/AR/otter/

[7] Douglas B. West, *Introduction to Graph Theory*,2nd ed., Prentice-Hall Inc.,2001

[8] M.Sipser,*Introduction to the Theory of Computation*,PWS,1997

[9] Christos H.Papadimitriou,*Computational Complexity*,Addison-Wesley Publishing Company Inc.,1995

[10] Tomi Janhunen, Course *T-79.144 Logic In Computer Science: Foundations*, lecture slides (in Finnish), Helsinki University of Technology,2003

[11] Anil Nerode and Richard A.Shore,*Logic for Applications*, Second Edition, Springer-Verlag inc.,New York,1997

[12] Tomi Janhunen, Course *T-79.144 Logic In Computer Science: Foundations*, home assignment 3, Helsinki University of Technology,2003

[13] Tomi Janhunen, Course *T-79.144 Logic In Computer Science: Foundations*, exercises (in Finnish), Helsinki University of Technology,2003

Final Temperature of a Heated Object May Be Predicted

Mikko Malinen

21st January, 2007 (first version 14th May, 2005)

1 Theory

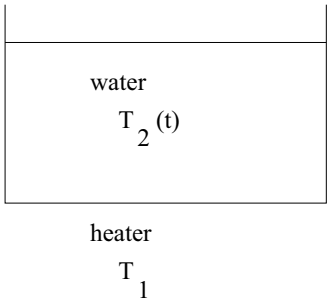
The heat exchange between two objects may be modelled by an equation

$$\frac{dQ}{dt} = kA\frac{T_H - T_C}{L}$$

from [1], where Q is the heat energy of object 2, k is thermal conductivity, A is the cross-section area of the conducting material, T_H is the temperature of object 1, T_C is the temperature of object 2 and L is the distance between objects 1 and 2. This may be written shortly

$$\frac{dQ}{dt} = a(T_1 - T_2(t))$$

where Q is the heat energy of object 2, a is a positive proportionality constant and T_1 and T_2 are temperatures of objects 1 and 2.



Picture 1. An example system.

Heat energy is comparable to temperature

$$Q - Q_0 = cm(T - T_0)$$

where Q_0 is the heat energy of the object just above the melting temperature, T is in Celsius-scale, and T is allowed to get such values that the object is in liquid phase. For example, for water $0 < T < 100^oC$. T_0 is the melting temperature of the object. From this we get

$$T - T_0 = \frac{Q - Q_0}{cm}$$

Remembering also that

$$\frac{d(Q - Q_0)}{dt} = \frac{dQ}{dt}$$

and

$$\frac{d(T - T_0)}{dt} = \frac{dT}{dt}$$

we can write

$$\frac{dT_2(t)}{dt} = \frac{d(T_2(t) - T_0)}{dt} = \frac{1}{cm} \cdot \frac{d(Q - Q_0)}{dt} = \frac{1}{cm}\frac{dQ}{dt} = \frac{1}{cm}a(T_1 - T_2(t)) = b(T_1 - T_2(t)), \quad b > 0, b \neq \infty$$

This is a differential equation, for which solution is

$$T_2(t) = T_1 + e^{-bt} \cdot C \tag{1}$$

Differentiating this we get

$$T_2'(t) = -be^{-bt} \cdot C \tag{2}$$

Differentiating again we get

$$T_2''(t) = b^2 e^{-bt} \cdot C \tag{3}$$

From (2) and (3) we get

$$\frac{T_2'(t)}{-b} = \frac{T_2''(t)}{b^2}$$

When solving b we get

$$b = -\frac{T_2''(t)}{T_2'(t)}, \quad T_2''(t) \neq 0, T_2'(t) \neq 0$$

From (2) and by substituting b we get

$$C = \frac{T_2'(t)}{\frac{T_2''(t)}{T_2'(t)} e^{\frac{T_2''(t)}{T_2'(t)} \cdot t}} = \frac{(T_2'(t))^2}{T_2''(t) e^{\frac{T_2''(t)}{T_2'(t)} \cdot t}}$$

At time $t = 0+$ (just after zero) C becomes

$$C_{t=0+} = \frac{(T_2'(0+))^2}{T_2''(0+)}$$

From (1) and by substituting $C_{t=0+}$ we get (at time $t = 0+$)

$$T_1 = T_2(0+) - \frac{(T_2'(0+))^2}{T_2''(0+)}, \quad T_2'(0+) \neq 0, T_2''(0+) \neq 0 \tag{4}$$

Limit of $T_2(t)$ is (from equation (1)):

$$\lim_{t\to\infty} T_2(t) = \lim_{t\to\infty} T_1 + e^{-bt} \cdot C = T_1$$

This limit is the final temperature. We have already the equation (4) for T_1. The right-hand side of this equation involves only $T_2(0+)$ and its two derivatives:

$$T_{2_{final}} = T_2(0+) - \frac{(T_2'(0+))^2}{T_2''(0+)}$$

Since any time instant may be considered as a new initial time instant, we may write

$$T_{2_{final}} = T_2(t) - \frac{(T_2'(t))^2}{T_2''(t)}, \quad t > 0$$

So the final temperature of an object may be predicted when its temperature and its two derivatives may be measured at time $t > 0$.

2 References

[1] Hugh D.Young and Roger A. Freedman, *University Physics*, 9th edition, Addison-Wesley, 1996

Cooling Down of Coffee

Mikko Malinen

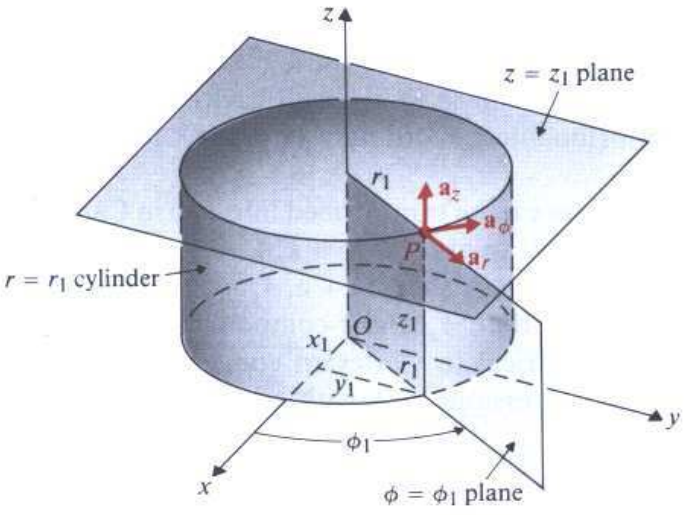
30th March, 2005

Abstract

This article deals with coffee in a cup. It proves, under certain assumptions, that if you pour cream immediately in to coffee, the coffee cools down more slowly than if you pour cream after a certain time. This is done by proving that heat flow out from the coffee is smaller if cream is added than it is for coffee without added cream.

1 Assumptions

1. The shape of the coffee is a cylinder



Picture 1. Cylindrical coordinates and the shape of the coffee [1].

2. The direction of the heat flow is perpendicular to the surface of the cylinder.

3. When the curved shell of the cylinder is considered, $\frac{dT}{dr}$, and when the

top of the cylinder is considered, $\frac{dT}{dz}$, and when the bottom of the cylinder is considered, $-\frac{dT}{dz}$ can be replaced by $-b\Delta T$, where b is a positive constant and ΔT is the temperature difference between the coffee and the room.

4. The heat flow is directly proportional to the area of the coffee

5. Equal volumes of coffee and cream have the same heat energy, $c_{co}m_{co}T = c_{cr}m_{cr}T$. That is $c_{co}V\rho_{co}T = c_{cr}V\rho_{cr}T$.

6. The temperature of the cream to be added is room temperature.

7. The coffee is in thermal equilibrium.

2 The Proof

The energy that crosses per unit time through a unit area placed perpendicular to the direction in which the energy flow takes place will be designated as j_E, the *energy current density*. The direction of the heat flow in the following is the X-axis. This energy flow takes place in a well-defined direction, which is the direction in which the temperature decreases. The change of temperature per unit length (or the *temperature gradient* of the material) is given by $\partial T/\partial x$. It has been found experimentally that (unless the temperature changes very rapidly over a short distance) j_E is proportional to $\partial T/\partial x$; that is,

$$j_E = -\kappa\frac{\partial T}{\partial x}$$

where κ is a coefficient characteristic of each material, and is called *thermal conductivity*. The negative sign indicates that the energy flows in the direction in which the temperature decreases. This statement is known as *Fourier's law* [2].

By multiplying both sides of Fourier's law by A (area) we get

$$j_E \cdot A = -\kappa A\frac{\partial T}{\partial x}$$

Because $\frac{dQ}{dt} = -j_E \cdot A$ we get

$$\frac{dQ}{dt} = \kappa A\frac{\partial T}{\partial x}$$

This may be written for the cylinder

$$\frac{dQ}{dt} = \kappa A_1(\frac{\partial T}{\partial r})_{r=r_{max+}} + \kappa A_2(\frac{\partial T}{\partial z})_{z=z_{max+}} - \kappa A_3(\frac{\partial T}{\partial z})_{z=0-} \qquad (1)$$

A_1 is the area of the cylinder without the top and the bottom. A_2 is the area of the top of the cylinder and A_3 is the area of the bottom of the cylinder. The derivatives are taken just outside the surface of the cylinder. By assumption 3, equation (1) may be written

$$\frac{dQ}{dt} = -\kappa(A_1 + A_2 + A_3)a\Delta T = -\kappa A a \Delta T = -bA\Delta T$$

We want to prove that

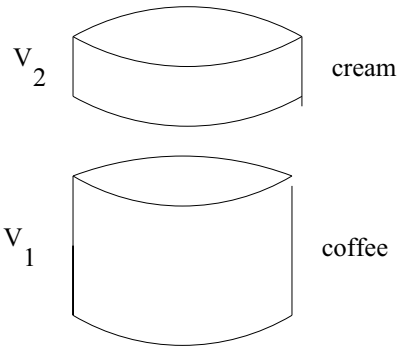
$$\frac{dQ_1}{dt} < \frac{dQ_2}{dt} \tag{2}$$

where Q_1 is the heat energy before adding cream and Q_2 is the heat energy after adding cream. (2) may be rewritten

$$-bA_1\Delta T_1 < -bA_2\Delta T_2$$

That is

$$A_1\Delta T_1 > A_2\Delta T_2 \tag{3}$$



Picture 2. Adding the cream.

Areas and temperature differences are in table 1:

	Before adding cream	After adding cream
area	$A_1 = d \cdot V_1 + 2 \cdot B$	$A_2 = d \cdot (V_1 + V_2) + 2 \cdot B$
temperature difference	ΔT_1	$\Delta T_2 = \frac{V_1 \cdot \Delta T_1}{V_1 + V_2}$

Table 1. d is a constant, B is the area of the top (and bottom) of the cylinder.

When rewriting (3) we get

$$(d \cdot V_1 + 2B) \cdot \Delta T_1 > (d \cdot (V_1 + V_2) + 2B) \cdot \frac{V_1 \cdot \Delta T_1}{V_1 + V_2} \tag{4}$$

We have to prove equation (4). We start by a first row which we can simply verify:

$$2B > \frac{2B \cdot V_1}{V_1 + V_2}$$

$$\Rightarrow d \cdot V_1 + 2B > d \cdot V_1 + \frac{2B \cdot V_1}{V_1 + V_2}$$

$$\Rightarrow d \cdot V_1 + 2B > \frac{d \cdot V_1 \cdot (V_1 + V_2) + 2B \cdot V_1}{V_1 + V_2}$$

$$\Rightarrow d \cdot V_1 + 2B > (d \cdot (V_1 + V_2) + 2B) \cdot \frac{V_1}{V_1 + V_2}$$

$$\Rightarrow (d \cdot V_1 + 2B) \cdot \Delta T_1 > (d \cdot (V_1 + V_2) + 2B) \cdot \frac{V_1 \cdot \Delta T_1}{V_1 + V_2}$$

The last row is the same as (4), so equation (4) is now proved, and it is proved that coffee cools down more slowly, if the cream is added immediately.

3 References

[1] David K. Cheng, *Field and Wave Electromagnetics*, Second Edition, Addison-Wesley, 1989

[2] Marcelo Alonso and Edward J.Finn,*Fundamental University Physics*, Volume One, Second Edition, Addison-Wesley, 1980

Examples of Using Theorem Prover Otter

Mikko Malinen

21st November,2004

Abstract

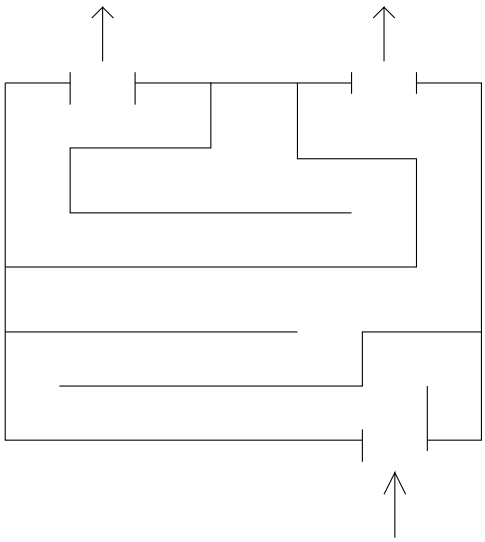
In this paper we introduce how a theorem prover may be used to prove that there exists a solution to a chess related problem and secondly, that there is a way out from a labyrinth.

1 A Chess Related Problem

Description of the problem:
If a knight is placed at the lower left corner of the chess board, can all chess squares be visited using rules of movement for knight? One variation of this problem is called *Knight's Tour* problem. In knights tour every square may be visited only once. It turns out that there is a solution, i.e. all chess squares can be visited. There exists solutions to even knight's tour.
In this presentation we show how the problem may be solved with theorem prover Otter [1]. In this solution any number of visits to chess squares are allowed.
In appendix section you may find listing of file knights.in. Here we explain the contents of this file. In section A of the file there is a line "Ra1.". This is a proposition which is interpreted as "Square a1 is reachable". Square a1 is reachable because there we put the knight initially. In section B of the file we put the definitions. In the beginning of section B there is line "Ra1 -> (Rb3 & Rc2).". This is interpreted as "if square a1 is reachable then squares b3 and c2 are reachable. This is where we write the moving possibilities from each square. The other rows are written in the same way. In section C we put the actual query (it's negation). Here we ask if all the squares are reachable.
The output file of Otter is in the appendix.

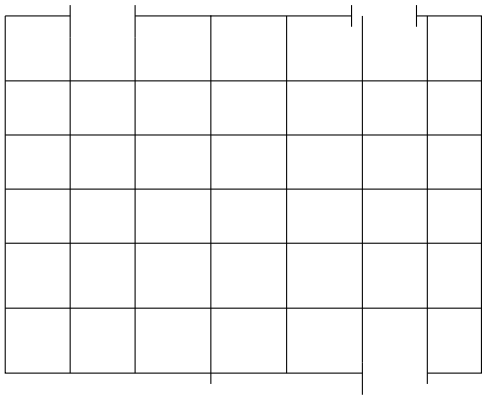
2 A Labyrinth Problem

Otter can be used to prove that there is a way out from a labyrinth. An example labyrinth is in picture 1.



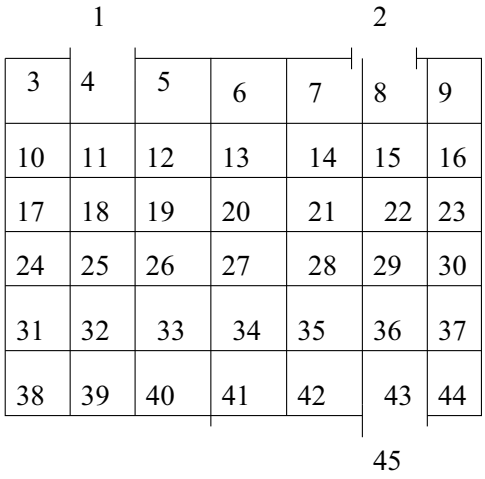
Picture 1. An example labyrinth.

The solving technique proceeds as follows.
1) Generating the squares. The walls of the squares must go along the walls of the labyrinth.



Picture 2. Generating the squares.

2) Numbering the squares



Picture 3. Numbering the squares.

3) Writing the input file. The input file labyrin.in is in appendix. Here we explain the contents of this file. Section A of the file has line "R45". This is interpreted as "Square (position) 45 is reachable". This is our entry position. Section B of the file has the definitions. For example line "R3 -> (R4 & R10)." is interpreted as "if square 3 is reachable then squares 4 and 10 are reachable. Here we need to define reachability only to the neighboring squares. In section C of the file we write the query (it's negation). With "-R2" we ask to prove that square (position) 2 is reachable. The output file (the proof) labyrin1.out is in the appendix.

3 References

[1] Otter: An Automated Deduction System
www.mcs.anl.gov/AR/otter/

4 Appendix

- knights.in
- knights.out
- labyrin.in
- labyrin1.out

4.1 knights.in

```
set(auto).
formula_list(usable).
% Section A: database
Ra1.

% Section B: definitions
Ra1 -> (Rb3 & Rc2).
Rb1 -> (Ra3 & Rc3 & Rd2).
Rc1 -> (Ra2 & Rb3 & Rd3 & Re2).
Rd1 -> (Rb2 & Rc3 & Re3 & Rf2).
Re1 -> (Rc2 & Rd3 & Rf3 & Rg2).
Rf1 -> (Rd2 & Re3 & Rg3 & Rh2).
Rg1 -> (Re2 & Rf3 & Rh3).
Rh1 -> (Rf2 & Rg3).

Ra2 -> (Rb4 & Rc3 & Rc1).
Rb2 -> (Ra4 & Rc4 & Rd3 & Rd1).
Rc2 -> (Ra1 & Ra3 & Rb4 & Rd4 & Re3 & Re1).
Rd2 -> (Rb1 & Rb3 & Rc4 & Re4 & Rf3 & Rf1).
Re2 -> (Rc1 & Rc3 & Rd4 & Rf4 & Rg3 & Rg1).
Rf2 -> (Rd1 & Rd3 & Re4 & Rg4 & Rh3 & Rh1).
Rg2 -> (Re1 & Re3 & Rf4 & Rh4).
Rh2 -> (Rf1 & Rf3 & Rg4).

Ra3 -> (Rb5 & Rc4 & Rc2 & Rb1).
Rb3 -> (Ra5 & Rc5 & Rd4 & Rd2 & Rc1 & Ra1).
Rc3 -> (Ra4 & Rb5 & Rd5 & Re4 & Re2 & Rd1 & Rb1 & Ra2).
Rd3 -> (Rb4 & Rc5 & Re5 & Rf4 & Rf2 & Re1 & Rc1 & Rb2).
Re3 -> (Rc4 & Rd5 & Rf5 & Rg4 & Rg2 & Rf1 & Rd1 & Rc2).
Rf3 -> (Rd4 & Re5 & Rg5 & Rh4 & Rh2 & Rg1 & Re1 & Rd2).
Rg3 -> (Re4 & Rf5 & Rh5 & Rh1 & Rf1 & Re2).
Rh3 -> (Rf4 & Rg5 & Rg1 & Rf2).

Ra4 -> (Rb6 & Rc5 & Rc3 & Rb2).
Rb4 -> (Ra6 & Rc6 & Rd5 & Rd3 & Rc2 & Ra2).
Rc4 -> (Ra5 & Rb6 & Rd6 & Re5 & Re3 & Rd2 & Rb2 & Ra3).
Rd4 -> (Rb5 & Rc6 & Re6 & Rf5 & Rf3 & Re2 & Rc2 & Rb3).
Re4 -> (Rc5 & Rd6 & Rf6 & Rg5 & Rg3 & Rf2 & Rd2 & Rc3).
Rf4 -> (Rd5 & Re6 & Rg6 & Rh5 & Rh3 & Rg2 & Re2 & Rd3).
Rg4 -> (Re5 & Rf6 & Rh6 & Rh2 & Rf2 & Re3).
Rh4 -> (Rf5 & Rg6 & Rg2 & Rf3).

Ra5 -> (Rb6 & Rc6 & Rc4 & Rb3).
Rb5 -> (Ra7 & Rc7 & Rd6 & Rd4 & Rc3 & Ra3).
```

```
Rc5 -> (Ra6 & Rb7 & Rd7 & Re6 & Re4 & Rd3 & Rb3 & Ra4).
Rd5 -> (Rb6 & Rc7 & Re7 & Rf6 & Rf4 & Re3 & Rc3 & Rb4).
Re5 -> (Rc6 & Rd7 & Rf7 & Rg6 & Rg4 & Rf3 & Rd3 & Rc4).
Rf5 -> (Rd6 & Re7 & Rg7 & Rh6 & Rh4 & Rg3 & Re3 & Rf4).
Rg5 -> (Re6 & Rf7 & Rh7 & Rh3 & Rf3 & Rg4).
Rh5 -> (Rf6 & Rg7 & Rg3 & Rh4).

Ra6 -> (Rb8 & Rc7 & Rc5 & Rb4).
Rb6 -> (Ra8 & Rc8 & Rd7 & Rd5 & Rc4 & Ra4).
Rc6 -> (Ra7 & Rb8 & Rd8 & Re7 & Re5 & Rd4 & Rb4 & Ra5).
Rd6 -> (Rb7 & Rc8 & Re8 & Rf7 & Rf5 & Re4 & Rc4 & Rb5).
Re6 -> (Rc7 & Rd8 & Rf8 & Rg7 & Rg5 & Rf4 & Rd4 & Rc5).
Rf6 -> (Rd7 & Re8 & Rg8 & Rh7 & Rh5 & Rg4 & Re4 & Rd5).
Rg6 -> (Re7 & Rf8 & Rh8 & Rh4 & Rf4 & Re5).
Rh6 -> (Rf7 & Rg8 & Rg4 & Rf5).

Ra7 -> (Rc8 & Rc6 & Rb5).
Rb7 -> (Rd8 & Rd6 & Rc5 & Ra5).
Rc7 -> (Ra8 & Re8 & Re6 & Rd5 & Rb5 & Ra6).
Rd7 -> (Rb8 & Rf8 & Rf6 & Re5 & Rc5 & Rb6).
Re7 -> (Rc8 & Rg8 & Rg6 & Rf5 & Rd5 & Rc6).
Rf7 -> (Rd8 & Rh8 & Rh6 & Rg5 & Re5 & Rd6).
Rg7 -> (Re8 & Rh5 & Rf5 & Re6).
Rh7 -> (Rf8 & Rg5 & Rf6).

Ra8 -> (Rc7 & Rb6).
Rb8 -> (Rd7 & Rc6 & Ra6).
Rc8 -> (Re7 & Rd6 & Rb6 & Ra7).
Rd8 -> (Rf7 & Re6 & Rc6 & Rb7).
Re8 -> (Rg7 & Rf6 & Rd6 & Rc7).
Rf8 -> (Rh7 & Rg6 & Re6 & Rd7).
Rg8 -> (Rh6 & Rf6 & Re7).
Rh8 -> (Rg6 & Rf7).

%Section C: negation of the query
-(Ra1 & Rb1 & Rc1 & Rd1 & Re1 & Rf1 & Rg1 & Rh1 & Ra2 & Rb2 & Rc2 & Rd2 & Re2
& Rf2 & Rg2 & Rh2 & Ra3 & Rb3 & Rc3 & Rd3 & Re3 & Rf3 & Rg3 & Rh3 & Ra4 & Rb4
& Rc4 & Rd4 & Re4 & Rf4 & Rg4 & Rh4 & Ra5 & Rb5 & Rc5 & Rd5 & Re5 & Rf5 & Rg5
& Rh5 & Ra6 & Rb6 & Rc6 & Rd6 & Re6 & Rf6 & Rg6 & Rh6 & Ra7 & Rb7 & Rc7 & Rd7
& Re7 & Rf7 & Rg7 & Rh7 & Ra8 & Rb8 & Rc8 & Rd8 & Re8 & Rf8 & Rg8 & Rh8).

end_of_list.
```

4.2 knights.out

```
----- Otter 3.0.6, April 2000 -----
The process was started by mmalinen on borodin.hut.fi, Fri Nov  5 11:34:13 2004
The command was "/p/edu/tik-79.144/bin/otter.linux".  The process ID is 28078.

set(auto).
   dependent: set(auto1).
   dependent: set(process_input).
   dependent: clear(print_kept).
   dependent: clear(print_new_demod).
   dependent: clear(print_back_demod).
   dependent: clear(print_back_sub).
   dependent: set(control_memory).
   dependent: assign(max_mem, 12000).
   dependent: assign(pick_given_ratio, 4).
   dependent: assign(stats_level, 1).
   dependent: assign(max_seconds, 10800).

formula_list(usable).
Ra1.
Ra1->Rb3&Rc2.
Rb1->Ra3&Rc3&Rd2.
Rc1->Ra2&Rb3&Rd3&Re2.
Rd1->Rb2&Rc3&Re3&Rf2.
Re1->Rc2&Rd3&Rf3&Rg2.
Rf1->Rd2&Re3&Rg3&Rh2.
Rg1->Re2&Rf3&Rh3.
Rh1->Rf2&Rg3.
Ra2->Rb4&Rc3&Rc1.
Rb2->Ra4&Rc4&Rd3&Rd1.
Rc2->Ra1&Ra3&Rb4&Rd4&Re3&Re1.
Rd2->Rb1&Rb3&Rc4&Re4&Rf3&Rf1.
Re2->Rc1&Rc3&Rd4&Rf4&Rg3&Rg1.
Rf2->Rd1&Rd3&Re4&Rg4&Rh3&Rh1.
Rg2->Re1&Re3&Rf4&Rh4.
Rh2->Rf1&Rf3&Rg4.
Ra3->Rb5&Rc4&Rc2&Rb1.
Rb3->Ra5&Rc5&Rd4&Rd2&Rc1&Ra1.
Rc3->Ra4&Rb5&Rd5&Re4&Re2&Rd1&Rb1&Ra2.
Rd3->Rb4&Rc5&Re5&Rf4&Rf2&Re1&Rc1&Rb2.
Re3->Rc4&Rd5&Rf5&Rg4&Rg2&Rf1&Rd1&Rc2.
Rf3->Rd4&Re5&Rg5&Rh4&Rh2&Rg1&Re1&Rd2.
Rg3->Re4&Rf5&Rh5&Rh1&Rf1&Re2.
Rh3->Rf4&Rg5&Rg1&Rf2.
Ra4->Rb6&Rc5&Rc3&Rb2.
```

```
Rb4->Ra6&Rc6&Rd5&Rd3&Rc2&Ra2.
Rc4->Ra5&Rb6&Rd6&Re5&Re3&Rd2&Rb2&Ra3.
Rd4->Rb5&Rc6&Re6&Rf5&Rf3&Re2&Rc2&Rb3.
Re4->Rc5&Rd6&Rf6&Rg5&Rg3&Rf2&Rd2&Rc3.
Rf4->Rd5&Re6&Rg6&Rh5&Rh3&Rg2&Re2&Rd3.
Rg4->Re5&Rf6&Rh6&Rh2&Rf2&Re3.
Rh4->Rf5&Rg6&Rg2&Rf3.
Ra5->Rb6&Rc6&Rc4&Rb3.
Rb5->Ra7&Rc7&Rd6&Rd4&Rc3&Ra3.
Rc5->Ra6&Rb7&Rd7&Re6&Re4&Rd3&Rb3&Ra4.
Rd5->Rb6&Rc7&Re7&Rf6&Rf4&Re3&Rc3&Rb4.
Re5->Rc6&Rd7&Rf7&Rg6&Rg4&Rf3&Rd3&Rc4.
Rf5->Rd6&Re7&Rg7&Rh6&Rh4&Rg3&Re3&Rf4.
Rg5->Re6&Rf7&Rh7&Rh3&Rf3&Rg4.
Rh5->Rf6&Rg7&Rg3&Rh4.
Ra6->Rb8&Rc7&Rc5&Rb4.
Rb6->Ra8&Rc8&Rd7&Rd5&Rc4&Ra4.
Rc6->Ra7&Rb8&Rd8&Re7&Re5&Rd4&Rb4&Ra5.
Rd6->Rb7&Rc8&Re8&Rf7&Rf5&Re4&Rc4&Rb5.
Re6->Rc7&Rd8&Rf8&Rg7&Rg5&Rf4&Rd4&Rc5.
Rf6->Rd7&Re8&Rg8&Rh7&Rh5&Rg4&Re4&Rd5.
Rg6->Re7&Rf8&Rh8&Rh4&Rf4&Re5.
Rh6->Rf7&Rg8&Rg4&Rf5.
Ra7->Rc8&Rc6&Rb5.
Rb7->Rd8&Rd6&Rc5&Ra5.
Rc7->Ra8&Re8&Re6&Rd5&Rb5&Ra6.
Rd7->Rb8&Rf8&Rf6&Re5&Rc5&Rb6.
Re7->Rc8&Rg8&Rg6&Rf5&Rd5&Rc6.
Rf7->Rd8&Rh8&Rh6&Rg5&Re5&Rd6.
Rg7->Re8&Rh5&Rf5&Re6.
Rh7->Rf8&Rg5&Rf6.
Ra8->Rc7&Rb6.
Rb8->Rd7&Rc6&Ra6.
Rc8->Re7&Rd6&Rb6&Ra7.
Rd8->Rf7&Re6&Rc6&Rb7.
Re8->Rg7&Rf6&Rd6&Rc7.
Rf8->Rh7&Rg6&Re6&Rd7.
Rg8->Rh6&Rf6&Re7.
Rh8->Rg6&Rf7.
-(Ra1&Rb1&Rc1&Rd1&Re1&Rf1&Rg1&Rh1&Ra2&Rb2&Rc2&Rd2&Re2&Rf2&Rg2&Rh2&Ra3&Rb3&Rc3
&Rd3&Re3&Rf3&Rg3&Rh3&Ra4&Rb4&Rc4&Rd4&Re4&Rf4&Rg4&Rh4&Ra5&Rb5&Rc5&Rd5&Re5&Rf5
&Rg5&Rh5&Ra6&Rb6&Rc6&Rd6&Re6&Rf6&Rg6&Rh6&Ra7&Rb7&Rc7&Rd7&Re7&Rf7&Rg7&Rh7&Ra8
&Rb8&Rc8&Rd8&Re8&Rf8&Rg8&Rh8).
end_of_list.

-------> usable clausifies to:
```

```
list(usable).
0 [] Ra1.
0 [] -Ra1|Rb3.
0 [] -Ra1|Rc2.
0 [] -Rb1|Ra3.
0 [] -Rb1|Rc3.
0 [] -Rb1|Rd2.
0 [] -Rc1|Ra2.
0 [] -Rc1|Rb3.
0 [] -Rc1|Rd3.
0 [] -Rc1|Re2.
0 [] -Rd1|Rb2.
0 [] -Rd1|Rc3.
0 [] -Rd1|Re3.
0 [] -Rd1|Rf2.
0 [] -Re1|Rc2.
0 [] -Re1|Rd3.
0 [] -Re1|Rf3.
0 [] -Re1|Rg2.
0 [] -Rf1|Rd2.
0 [] -Rf1|Re3.
0 [] -Rf1|Rg3.
0 [] -Rf1|Rh2.
0 [] -Rg1|Re2.
0 [] -Rg1|Rf3.
0 [] -Rg1|Rh3.
0 [] -Rh1|Rf2.
0 [] -Rh1|Rg3.
0 [] -Ra2|Rb4.
0 [] -Ra2|Rc3.
0 [] -Ra2|Rc1.
0 [] -Rb2|Ra4.
0 [] -Rb2|Rc4.
0 [] -Rb2|Rd3.
0 [] -Rb2|Rd1.
0 [] -Rc2|Ra1.
0 [] -Rc2|Ra3.
0 [] -Rc2|Rb4.
0 [] -Rc2|Rd4.
0 [] -Rc2|Re3.
0 [] -Rc2|Re1.
0 [] -Rd2|Rb1.
0 [] -Rd2|Rb3.
0 [] -Rd2|Rc4.
0 [] -Rd2|Re4.
```

```
0 [] -Rd2|Rf3.
0 [] -Rd2|Rf1.
0 [] -Re2|Rc1.
0 [] -Re2|Rc3.
0 [] -Re2|Rd4.
0 [] -Re2|Rf4.
0 [] -Re2|Rg3.
0 [] -Re2|Rg1.
0 [] -Rf2|Rd1.
0 [] -Rf2|Rd3.
0 [] -Rf2|Re4.
0 [] -Rf2|Rg4.
0 [] -Rf2|Rh3.
0 [] -Rf2|Rh1.
0 [] -Rg2|Re1.
0 [] -Rg2|Re3.
0 [] -Rg2|Rf4.
0 [] -Rg2|Rh4.
0 [] -Rh2|Rf1.
0 [] -Rh2|Rf3.
0 [] -Rh2|Rg4.
0 [] -Ra3|Rb5.
0 [] -Ra3|Rc4.
0 [] -Ra3|Rc2.
0 [] -Ra3|Rb1.
0 [] -Rb3|Ra5.
0 [] -Rb3|Rc5.
0 [] -Rb3|Rd4.
0 [] -Rb3|Rd2.
0 [] -Rb3|Rc1.
0 [] -Rb3|Ra1.
0 [] -Rc3|Ra4.
0 [] -Rc3|Rb5.
0 [] -Rc3|Rd5.
0 [] -Rc3|Re4.
0 [] -Rc3|Re2.
0 [] -Rc3|Rd1.
0 [] -Rc3|Rb1.
0 [] -Rc3|Ra2.
0 [] -Rd3|Rb4.
0 [] -Rd3|Rc5.
0 [] -Rd3|Re5.
0 [] -Rd3|Rf4.
0 [] -Rd3|Rf2.
0 [] -Rd3|Re1.
0 [] -Rd3|Rc1.
```

```
0 [] -Rd3|Rb2.
0 [] -Re3|Rc4.
0 [] -Re3|Rd5.
0 [] -Re3|Rf5.
0 [] -Re3|Rg4.
0 [] -Re3|Rg2.
0 [] -Re3|Rf1.
0 [] -Re3|Rd1.
0 [] -Re3|Rc2.
0 [] -Rf3|Rd4.
0 [] -Rf3|Re5.
0 [] -Rf3|Rg5.
0 [] -Rf3|Rh4.
0 [] -Rf3|Rh2.
0 [] -Rf3|Rg1.
0 [] -Rf3|Re1.
0 [] -Rf3|Rd2.
0 [] -Rg3|Re4.
0 [] -Rg3|Rf5.
0 [] -Rg3|Rh5.
0 [] -Rg3|Rh1.
0 [] -Rg3|Rf1.
0 [] -Rg3|Re2.
0 [] -Rh3|Rf4.
0 [] -Rh3|Rg5.
0 [] -Rh3|Rg1.
0 [] -Rh3|Rf2.
0 [] -Ra4|Rb6.
0 [] -Ra4|Rc5.
0 [] -Ra4|Rc3.
0 [] -Ra4|Rb2.
0 [] -Rb4|Ra6.
0 [] -Rb4|Rc6.
0 [] -Rb4|Rd5.
0 [] -Rb4|Rd3.
0 [] -Rb4|Rc2.
0 [] -Rb4|Ra2.
0 [] -Rc4|Ra5.
0 [] -Rc4|Rb6.
0 [] -Rc4|Rd6.
0 [] -Rc4|Re5.
0 [] -Rc4|Re3.
0 [] -Rc4|Rd2.
0 [] -Rc4|Rb2.
0 [] -Rc4|Ra3.
0 [] -Rd4|Rb5.
```

```
0 [] -Rd4|Rc6.
0 [] -Rd4|Re6.
0 [] -Rd4|Rf5.
0 [] -Rd4|Rf3.
0 [] -Rd4|Re2.
0 [] -Rd4|Rc2.
0 [] -Rd4|Rb3.
0 [] -Re4|Rc5.
0 [] -Re4|Rd6.
0 [] -Re4|Rf6.
0 [] -Re4|Rg5.
0 [] -Re4|Rg3.
0 [] -Re4|Rf2.
0 [] -Re4|Rd2.
0 [] -Re4|Rc3.
0 [] -Rf4|Rd5.
0 [] -Rf4|Re6.
0 [] -Rf4|Rg6.
0 [] -Rf4|Rh5.
0 [] -Rf4|Rh3.
0 [] -Rf4|Rg2.
0 [] -Rf4|Re2.
0 [] -Rf4|Rd3.
0 [] -Rg4|Re5.
0 [] -Rg4|Rf6.
0 [] -Rg4|Rh6.
0 [] -Rg4|Rh2.
0 [] -Rg4|Rf2.
0 [] -Rg4|Re3.
0 [] -Rh4|Rf5.
0 [] -Rh4|Rg6.
0 [] -Rh4|Rg2.
0 [] -Rh4|Rf3.
0 [] -Ra5|Rb6.
0 [] -Ra5|Rc6.
0 [] -Ra5|Rc4.
0 [] -Ra5|Rb3.
0 [] -Rb5|Ra7.
0 [] -Rb5|Rc7.
0 [] -Rb5|Rd6.
0 [] -Rb5|Rd4.
0 [] -Rb5|Rc3.
0 [] -Rb5|Ra3.
0 [] -Rc5|Ra6.
0 [] -Rc5|Rb7.
0 [] -Rc5|Rd7.
```

```
0 [] -Rc5|Re6.
0 [] -Rc5|Re4.
0 [] -Rc5|Rd3.
0 [] -Rc5|Rb3.
0 [] -Rc5|Ra4.
0 [] -Rd5|Rb6.
0 [] -Rd5|Rc7.
0 [] -Rd5|Re7.
0 [] -Rd5|Rf6.
0 [] -Rd5|Rf4.
0 [] -Rd5|Re3.
0 [] -Rd5|Rc3.
0 [] -Rd5|Rb4.
0 [] -Re5|Rc6.
0 [] -Re5|Rd7.
0 [] -Re5|Rf7.
0 [] -Re5|Rg6.
0 [] -Re5|Rg4.
0 [] -Re5|Rf3.
0 [] -Re5|Rd3.
0 [] -Re5|Rc4.
0 [] -Rf5|Rd6.
0 [] -Rf5|Re7.
0 [] -Rf5|Rg7.
0 [] -Rf5|Rh6.
0 [] -Rf5|Rh4.
0 [] -Rf5|Rg3.
0 [] -Rf5|Re3.
0 [] -Rf5|Rf4.
0 [] -Rg5|Re6.
0 [] -Rg5|Rf7.
0 [] -Rg5|Rh7.
0 [] -Rg5|Rh3.
0 [] -Rg5|Rf3.
0 [] -Rg5|Rg4.
0 [] -Rh5|Rf6.
0 [] -Rh5|Rg7.
0 [] -Rh5|Rg3.
0 [] -Rh5|Rh4.
0 [] -Ra6|Rb8.
0 [] -Ra6|Rc7.
0 [] -Ra6|Rc5.
0 [] -Ra6|Rb4.
0 [] -Rb6|Ra8.
0 [] -Rb6|Rc8.
0 [] -Rb6|Rd7.
```

```
0 [] -Rb6|Rd5.
0 [] -Rb6|Rc4.
0 [] -Rb6|Ra4.
0 [] -Rc6|Ra7.
0 [] -Rc6|Rb8.
0 [] -Rc6|Rd8.
0 [] -Rc6|Re7.
0 [] -Rc6|Re5.
0 [] -Rc6|Rd4.
0 [] -Rc6|Rb4.
0 [] -Rc6|Ra5.
0 [] -Rd6|Rb7.
0 [] -Rd6|Rc8.
0 [] -Rd6|Re8.
0 [] -Rd6|Rf7.
0 [] -Rd6|Rf5.
0 [] -Rd6|Re4.
0 [] -Rd6|Rc4.
0 [] -Rd6|Rb5.
0 [] -Re6|Rc7.
0 [] -Re6|Rd8.
0 [] -Re6|Rf8.
0 [] -Re6|Rg7.
0 [] -Re6|Rg5.
0 [] -Re6|Rf4.
0 [] -Re6|Rd4.
0 [] -Re6|Rc5.
0 [] -Rf6|Rd7.
0 [] -Rf6|Re8.
0 [] -Rf6|Rg8.
0 [] -Rf6|Rh7.
0 [] -Rf6|Rh5.
0 [] -Rf6|Rg4.
0 [] -Rf6|Re4.
0 [] -Rf6|Rd5.
0 [] -Rg6|Re7.
0 [] -Rg6|Rf8.
0 [] -Rg6|Rh8.
0 [] -Rg6|Rh4.
0 [] -Rg6|Rf4.
0 [] -Rg6|Re5.
0 [] -Rh6|Rf7.
0 [] -Rh6|Rg8.
0 [] -Rh6|Rg4.
0 [] -Rh6|Rf5.
0 [] -Ra7|Rc8.
```

```
0 [] -Ra7|Rc6.
0 [] -Ra7|Rb5.
0 [] -Rb7|Rd8.
0 [] -Rb7|Rd6.
0 [] -Rb7|Rc5.
0 [] -Rb7|Ra5.
0 [] -Rc7|Ra8.
0 [] -Rc7|Re8.
0 [] -Rc7|Re6.
0 [] -Rc7|Rd5.
0 [] -Rc7|Rb5.
0 [] -Rc7|Ra6.
0 [] -Rd7|Rb8.
0 [] -Rd7|Rf8.
0 [] -Rd7|Rf6.
0 [] -Rd7|Re5.
0 [] -Rd7|Rc5.
0 [] -Rd7|Rb6.
0 [] -Re7|Rc8.
0 [] -Re7|Rg8.
0 [] -Re7|Rg6.
0 [] -Re7|Rf5.
0 [] -Re7|Rd5.
0 [] -Re7|Rc6.
0 [] -Rf7|Rd8.
0 [] -Rf7|Rh8.
0 [] -Rf7|Rh6.
0 [] -Rf7|Rg5.
0 [] -Rf7|Re5.
0 [] -Rf7|Rd6.
0 [] -Rg7|Re8.
0 [] -Rg7|Rh5.
0 [] -Rg7|Rf5.
0 [] -Rg7|Re6.
0 [] -Rh7|Rf8.
0 [] -Rh7|Rg5.
0 [] -Rh7|Rf6.
0 [] -Ra8|Rc7.
0 [] -Ra8|Rb6.
0 [] -Rb8|Rd7.
0 [] -Rb8|Rc6.
0 [] -Rb8|Ra6.
0 [] -Rc8|Re7.
0 [] -Rc8|Rd6.
0 [] -Rc8|Rb6.
0 [] -Rc8|Ra7.
```

```
0 [] -Rd8|Rf7.
0 [] -Rd8|Re6.
0 [] -Rd8|Rc6.
0 [] -Rd8|Rb7.
0 [] -Re8|Rg7.
0 [] -Re8|Rf6.
0 [] -Re8|Rd6.
0 [] -Re8|Rc7.
0 [] -Rf8|Rh7.
0 [] -Rf8|Rg6.
0 [] -Rf8|Re6.
0 [] -Rf8|Rd7.
0 [] -Rg8|Rh6.
0 [] -Rg8|Rf6.
0 [] -Rg8|Re7.
0 [] -Rh8|Rg6.
0 [] -Rh8|Rf7.
0 [] -Ra1| -Rb1| -Rc1| -Rd1| -Re1| -Rf1| -Rg1| -Rh1| -Ra2| -Rb2| -Rc2| -Rd2|
-Re2| -Rf2| -Rg2| -Rh2| -Ra3| -Rb3| -Rc3| -Rd3| -Re3| -Rf3| -Rg3| -Rh3| -Ra4|
-Rb4| -Rc4| -Rd4| -Re4| -Rf4| -Rg4| -Rh4| -Ra5| -Rb5| -Rc5| -Rd5| -Re5| -Rf5|
-Rg5| -Rh5| -Ra6| -Rb6| -Rc6| -Rd6| -Re6| -Rf6| -Rg6| -Rh6| -Ra7| -Rb7| -Rc7|
-Rd7| -Re7| -Rf7| -Rg7| -Rh7| -Ra8| -Rb8| -Rc8| -Rd8| -Re8| -Rf8| -Rg8| -Rh8.
end_of_list.

SCAN INPUT: prop=1, horn=1, equality=0, symmetry=0, max_lits=64.

The clause set is propositional; the strategy will be
ordered hyperresolution with the propositional
optimizations, with satellites in sos and nuclei in usable.

   dependent: set(hyper_res).
   dependent: set(propositional).
   dependent: set(sort_literals).

------------> process usable:
** KEPT (pick-wt=2): 1 [] -Ra1|Rb3.
** KEPT (pick-wt=2): 2 [] -Ra1|Rc2.
** KEPT (pick-wt=2): 3 [] -Rb1|Ra3.
** KEPT (pick-wt=2): 4 [] -Rb1|Rc3.
** KEPT (pick-wt=2): 5 [] -Rb1|Rd2.
** KEPT (pick-wt=2): 6 [] -Rc1|Ra2.
** KEPT (pick-wt=2): 7 [] -Rc1|Rb3.
** KEPT (pick-wt=2): 8 [] -Rc1|Rd3.
** KEPT (pick-wt=2): 9 [] -Rc1|Re2.
** KEPT (pick-wt=2): 10 [] -Rd1|Rb2.
** KEPT (pick-wt=2): 11 [] -Rd1|Rc3.
```

```
** KEPT (pick-wt=2): 12 [] -Rd1|Re3.
** KEPT (pick-wt=2): 13 [] -Rd1|Rf2.
** KEPT (pick-wt=2): 14 [] -Re1|Rc2.
** KEPT (pick-wt=2): 15 [] -Re1|Rd3.
** KEPT (pick-wt=2): 16 [] -Re1|Rf3.
** KEPT (pick-wt=2): 17 [] -Re1|Rg2.
** KEPT (pick-wt=2): 18 [] -Rf1|Rd2.
** KEPT (pick-wt=2): 19 [] -Rf1|Re3.
** KEPT (pick-wt=2): 20 [] -Rf1|Rg3.
** KEPT (pick-wt=2): 21 [] -Rf1|Rh2.
** KEPT (pick-wt=2): 22 [] -Rg1|Re2.
** KEPT (pick-wt=2): 23 [] -Rg1|Rf3.
** KEPT (pick-wt=2): 24 [] -Rg1|Rh3.
** KEPT (pick-wt=2): 25 [] -Rh1|Rf2.
** KEPT (pick-wt=2): 26 [] -Rh1|Rg3.
** KEPT (pick-wt=2): 27 [] -Ra2|Rb4.
** KEPT (pick-wt=2): 28 [] -Ra2|Rc3.
** KEPT (pick-wt=2): 29 [] -Ra2|Rc1.
** KEPT (pick-wt=2): 30 [] -Rb2|Ra4.
** KEPT (pick-wt=2): 31 [] -Rb2|Rc4.
** KEPT (pick-wt=2): 32 [] -Rb2|Rd3.
** KEPT (pick-wt=2): 33 [] -Rb2|Rd1.
  Following clause subsumed by 0 during input processing: 0 [] -Rc2|Ra1.
** KEPT (pick-wt=2): 34 [] -Rc2|Ra3.
** KEPT (pick-wt=2): 35 [] -Rc2|Rb4.
** KEPT (pick-wt=2): 36 [] -Rc2|Rd4.
** KEPT (pick-wt=2): 37 [] -Rc2|Re3.
** KEPT (pick-wt=2): 38 [] -Rc2|Re1.
** KEPT (pick-wt=2): 39 [] -Rd2|Rb1.
** KEPT (pick-wt=2): 40 [] -Rd2|Rb3.
** KEPT (pick-wt=2): 41 [] -Rd2|Rc4.
** KEPT (pick-wt=2): 42 [] -Rd2|Re4.
** KEPT (pick-wt=2): 43 [] -Rd2|Rf3.
** KEPT (pick-wt=2): 44 [] -Rd2|Rf1.
** KEPT (pick-wt=2): 45 [] -Re2|Rc1.
** KEPT (pick-wt=2): 46 [] -Re2|Rc3.
** KEPT (pick-wt=2): 47 [] -Re2|Rd4.
** KEPT (pick-wt=2): 48 [] -Re2|Rf4.
** KEPT (pick-wt=2): 49 [] -Re2|Rg3.
** KEPT (pick-wt=2): 50 [] -Re2|Rg1.
** KEPT (pick-wt=2): 51 [] -Rf2|Rd1.
** KEPT (pick-wt=2): 52 [] -Rf2|Rd3.
** KEPT (pick-wt=2): 53 [] -Rf2|Re4.
** KEPT (pick-wt=2): 54 [] -Rf2|Rg4.
** KEPT (pick-wt=2): 55 [] -Rf2|Rh3.
** KEPT (pick-wt=2): 56 [] -Rf2|Rh1.
```

```
** KEPT (pick-wt=2): 57 [] -Rg2|Re1.
** KEPT (pick-wt=2): 58 [] -Rg2|Re3.
** KEPT (pick-wt=2): 59 [] -Rg2|Rf4.
** KEPT (pick-wt=2): 60 [] -Rg2|Rh4.
** KEPT (pick-wt=2): 61 [] -Rh2|Rf1.
** KEPT (pick-wt=2): 62 [] -Rh2|Rf3.
** KEPT (pick-wt=2): 63 [] -Rh2|Rg4.
** KEPT (pick-wt=2): 64 [] -Ra3|Rb5.
** KEPT (pick-wt=2): 65 [] -Ra3|Rc4.
** KEPT (pick-wt=2): 66 [] -Ra3|Rc2.
** KEPT (pick-wt=2): 67 [] -Ra3|Rb1.
** KEPT (pick-wt=2): 68 [] -Rb3|Ra5.
** KEPT (pick-wt=2): 69 [] -Rb3|Rc5.
** KEPT (pick-wt=2): 70 [] -Rb3|Rd4.
** KEPT (pick-wt=2): 71 [] -Rb3|Rd2.
** KEPT (pick-wt=2): 72 [] -Rb3|Rc1.
  Following clause subsumed by 0 during input processing: 0 [] -Rb3|Ra1.
** KEPT (pick-wt=2): 73 [] -Rc3|Ra4.
** KEPT (pick-wt=2): 74 [] -Rc3|Rb5.
** KEPT (pick-wt=2): 75 [] -Rc3|Rd5.
** KEPT (pick-wt=2): 76 [] -Rc3|Re4.
** KEPT (pick-wt=2): 77 [] -Rc3|Re2.
** KEPT (pick-wt=2): 78 [] -Rc3|Rd1.
** KEPT (pick-wt=2): 79 [] -Rc3|Rb1.
** KEPT (pick-wt=2): 80 [] -Rc3|Ra2.
** KEPT (pick-wt=2): 81 [] -Rd3|Rb4.
** KEPT (pick-wt=2): 82 [] -Rd3|Rc5.
** KEPT (pick-wt=2): 83 [] -Rd3|Re5.
** KEPT (pick-wt=2): 84 [] -Rd3|Rf4.
** KEPT (pick-wt=2): 85 [] -Rd3|Rf2.
** KEPT (pick-wt=2): 86 [] -Rd3|Re1.
** KEPT (pick-wt=2): 87 [] -Rd3|Rc1.
** KEPT (pick-wt=2): 88 [] -Rd3|Rb2.
** KEPT (pick-wt=2): 89 [] -Re3|Rc4.
** KEPT (pick-wt=2): 90 [] -Re3|Rd5.
** KEPT (pick-wt=2): 91 [] -Re3|Rf5.
** KEPT (pick-wt=2): 92 [] -Re3|Rg4.
** KEPT (pick-wt=2): 93 [] -Re3|Rg2.
** KEPT (pick-wt=2): 94 [] -Re3|Rf1.
** KEPT (pick-wt=2): 95 [] -Re3|Rd1.
** KEPT (pick-wt=2): 96 [] -Re3|Rc2.
** KEPT (pick-wt=2): 97 [] -Rf3|Rd4.
** KEPT (pick-wt=2): 98 [] -Rf3|Re5.
** KEPT (pick-wt=2): 99 [] -Rf3|Rg5.
** KEPT (pick-wt=2): 100 [] -Rf3|Rh4.
** KEPT (pick-wt=2): 101 [] -Rf3|Rh2.
```

```
** KEPT (pick-wt=2): 102 [] -Rf3|Rg1.
** KEPT (pick-wt=2): 103 [] -Rf3|Re1.
** KEPT (pick-wt=2): 104 [] -Rf3|Rd2.
** KEPT (pick-wt=2): 105 [] -Rg3|Re4.
** KEPT (pick-wt=2): 106 [] -Rg3|Rf5.
** KEPT (pick-wt=2): 107 [] -Rg3|Rh5.
** KEPT (pick-wt=2): 108 [] -Rg3|Rh1.
** KEPT (pick-wt=2): 109 [] -Rg3|Rf1.
** KEPT (pick-wt=2): 110 [] -Rg3|Re2.
** KEPT (pick-wt=2): 111 [] -Rh3|Rf4.
** KEPT (pick-wt=2): 112 [] -Rh3|Rg5.
** KEPT (pick-wt=2): 113 [] -Rh3|Rg1.
** KEPT (pick-wt=2): 114 [] -Rh3|Rf2.
** KEPT (pick-wt=2): 115 [] -Ra4|Rb6.
** KEPT (pick-wt=2): 116 [] -Ra4|Rc5.
** KEPT (pick-wt=2): 117 [] -Ra4|Rc3.
** KEPT (pick-wt=2): 118 [] -Ra4|Rb2.
** KEPT (pick-wt=2): 119 [] -Rb4|Ra6.
** KEPT (pick-wt=2): 120 [] -Rb4|Rc6.
** KEPT (pick-wt=2): 121 [] -Rb4|Rd5.
** KEPT (pick-wt=2): 122 [] -Rb4|Rd3.
** KEPT (pick-wt=2): 123 [] -Rb4|Rc2.
** KEPT (pick-wt=2): 124 [] -Rb4|Ra2.
** KEPT (pick-wt=2): 125 [] -Rc4|Ra5.
** KEPT (pick-wt=2): 126 [] -Rc4|Rb6.
** KEPT (pick-wt=2): 127 [] -Rc4|Rd6.
** KEPT (pick-wt=2): 128 [] -Rc4|Re5.
** KEPT (pick-wt=2): 129 [] -Rc4|Re3.
** KEPT (pick-wt=2): 130 [] -Rc4|Rd2.
** KEPT (pick-wt=2): 131 [] -Rc4|Rb2.
** KEPT (pick-wt=2): 132 [] -Rc4|Ra3.
** KEPT (pick-wt=2): 133 [] -Rd4|Rb5.
** KEPT (pick-wt=2): 134 [] -Rd4|Rc6.
** KEPT (pick-wt=2): 135 [] -Rd4|Re6.
** KEPT (pick-wt=2): 136 [] -Rd4|Rf5.
** KEPT (pick-wt=2): 137 [] -Rd4|Rf3.
** KEPT (pick-wt=2): 138 [] -Rd4|Re2.
** KEPT (pick-wt=2): 139 [] -Rd4|Rc2.
** KEPT (pick-wt=2): 140 [] -Rd4|Rb3.
** KEPT (pick-wt=2): 141 [] -Re4|Rc5.
** KEPT (pick-wt=2): 142 [] -Re4|Rd6.
** KEPT (pick-wt=2): 143 [] -Re4|Rf6.
** KEPT (pick-wt=2): 144 [] -Re4|Rg5.
** KEPT (pick-wt=2): 145 [] -Re4|Rg3.
** KEPT (pick-wt=2): 146 [] -Re4|Rf2.
** KEPT (pick-wt=2): 147 [] -Re4|Rd2.
```

```
** KEPT (pick-wt=2): 148 [] -Re4|Rc3.
** KEPT (pick-wt=2): 149 [] -Rf4|Rd5.
** KEPT (pick-wt=2): 150 [] -Rf4|Re6.
** KEPT (pick-wt=2): 151 [] -Rf4|Rg6.
** KEPT (pick-wt=2): 152 [] -Rf4|Rh5.
** KEPT (pick-wt=2): 153 [] -Rf4|Rh3.
** KEPT (pick-wt=2): 154 [] -Rf4|Rg2.
** KEPT (pick-wt=2): 155 [] -Rf4|Re2.
** KEPT (pick-wt=2): 156 [] -Rf4|Rd3.
** KEPT (pick-wt=2): 157 [] -Rg4|Re5.
** KEPT (pick-wt=2): 158 [] -Rg4|Rf6.
** KEPT (pick-wt=2): 159 [] -Rg4|Rh6.
** KEPT (pick-wt=2): 160 [] -Rg4|Rh2.
** KEPT (pick-wt=2): 161 [] -Rg4|Rf2.
** KEPT (pick-wt=2): 162 [] -Rg4|Re3.
** KEPT (pick-wt=2): 163 [] -Rh4|Rf5.
** KEPT (pick-wt=2): 164 [] -Rh4|Rg6.
** KEPT (pick-wt=2): 165 [] -Rh4|Rg2.
** KEPT (pick-wt=2): 166 [] -Rh4|Rf3.
** KEPT (pick-wt=2): 167 [] -Ra5|Rb6.
** KEPT (pick-wt=2): 168 [] -Ra5|Rc6.
** KEPT (pick-wt=2): 169 [] -Ra5|Rc4.
** KEPT (pick-wt=2): 170 [] -Ra5|Rb3.
** KEPT (pick-wt=2): 171 [] -Rb5|Ra7.
** KEPT (pick-wt=2): 172 [] -Rb5|Rc7.
** KEPT (pick-wt=2): 173 [] -Rb5|Rd6.
** KEPT (pick-wt=2): 174 [] -Rb5|Rd4.
** KEPT (pick-wt=2): 175 [] -Rb5|Rc3.
** KEPT (pick-wt=2): 176 [] -Rb5|Ra3.
** KEPT (pick-wt=2): 177 [] -Rc5|Ra6.
** KEPT (pick-wt=2): 178 [] -Rc5|Rb7.
** KEPT (pick-wt=2): 179 [] -Rc5|Rd7.
** KEPT (pick-wt=2): 180 [] -Rc5|Re6.
** KEPT (pick-wt=2): 181 [] -Rc5|Re4.
** KEPT (pick-wt=2): 182 [] -Rc5|Rd3.
** KEPT (pick-wt=2): 183 [] -Rc5|Rb3.
** KEPT (pick-wt=2): 184 [] -Rc5|Ra4.
** KEPT (pick-wt=2): 185 [] -Rd5|Rb6.
** KEPT (pick-wt=2): 186 [] -Rd5|Rc7.
** KEPT (pick-wt=2): 187 [] -Rd5|Re7.
** KEPT (pick-wt=2): 188 [] -Rd5|Rf6.
** KEPT (pick-wt=2): 189 [] -Rd5|Rf4.
** KEPT (pick-wt=2): 190 [] -Rd5|Re3.
** KEPT (pick-wt=2): 191 [] -Rd5|Rc3.
** KEPT (pick-wt=2): 192 [] -Rd5|Rb4.
** KEPT (pick-wt=2): 193 [] -Re5|Rc6.
```

```
** KEPT (pick-wt=2): 194 [] -Re5|Rd7.
** KEPT (pick-wt=2): 195 [] -Re5|Rf7.
** KEPT (pick-wt=2): 196 [] -Re5|Rg6.
** KEPT (pick-wt=2): 197 [] -Re5|Rg4.
** KEPT (pick-wt=2): 198 [] -Re5|Rf3.
** KEPT (pick-wt=2): 199 [] -Re5|Rd3.
** KEPT (pick-wt=2): 200 [] -Re5|Rc4.
** KEPT (pick-wt=2): 201 [] -Rf5|Rd6.
** KEPT (pick-wt=2): 202 [] -Rf5|Re7.
** KEPT (pick-wt=2): 203 [] -Rf5|Rg7.
** KEPT (pick-wt=2): 204 [] -Rf5|Rh6.
** KEPT (pick-wt=2): 205 [] -Rf5|Rh4.
** KEPT (pick-wt=2): 206 [] -Rf5|Rg3.
** KEPT (pick-wt=2): 207 [] -Rf5|Re3.
** KEPT (pick-wt=2): 208 [] -Rf5|Rf4.
** KEPT (pick-wt=2): 209 [] -Rg5|Re6.
** KEPT (pick-wt=2): 210 [] -Rg5|Rf7.
** KEPT (pick-wt=2): 211 [] -Rg5|Rh7.
** KEPT (pick-wt=2): 212 [] -Rg5|Rh3.
** KEPT (pick-wt=2): 213 [] -Rg5|Rf3.
** KEPT (pick-wt=2): 214 [] -Rg5|Rg4.
** KEPT (pick-wt=2): 215 [] -Rh5|Rf6.
** KEPT (pick-wt=2): 216 [] -Rh5|Rg7.
** KEPT (pick-wt=2): 217 [] -Rh5|Rg3.
** KEPT (pick-wt=2): 218 [] -Rh5|Rh4.
** KEPT (pick-wt=2): 219 [] -Ra6|Rb8.
** KEPT (pick-wt=2): 220 [] -Ra6|Rc7.
** KEPT (pick-wt=2): 221 [] -Ra6|Rc5.
** KEPT (pick-wt=2): 222 [] -Ra6|Rb4.
** KEPT (pick-wt=2): 223 [] -Rb6|Ra8.
** KEPT (pick-wt=2): 224 [] -Rb6|Rc8.
** KEPT (pick-wt=2): 225 [] -Rb6|Rd7.
** KEPT (pick-wt=2): 226 [] -Rb6|Rd5.
** KEPT (pick-wt=2): 227 [] -Rb6|Rc4.
** KEPT (pick-wt=2): 228 [] -Rb6|Ra4.
** KEPT (pick-wt=2): 229 [] -Rc6|Ra7.
** KEPT (pick-wt=2): 230 [] -Rc6|Rb8.
** KEPT (pick-wt=2): 231 [] -Rc6|Rd8.
** KEPT (pick-wt=2): 232 [] -Rc6|Re7.
** KEPT (pick-wt=2): 233 [] -Rc6|Re5.
** KEPT (pick-wt=2): 234 [] -Rc6|Rd4.
** KEPT (pick-wt=2): 235 [] -Rc6|Rb4.
** KEPT (pick-wt=2): 236 [] -Rc6|Ra5.
** KEPT (pick-wt=2): 237 [] -Rd6|Rb7.
** KEPT (pick-wt=2): 238 [] -Rd6|Rc8.
** KEPT (pick-wt=2): 239 [] -Rd6|Re8.
```

```
** KEPT (pick-wt=2): 240 [] -Rd6|Rf7.
** KEPT (pick-wt=2): 241 [] -Rd6|Rf5.
** KEPT (pick-wt=2): 242 [] -Rd6|Re4.
** KEPT (pick-wt=2): 243 [] -Rd6|Rc4.
** KEPT (pick-wt=2): 244 [] -Rd6|Rb5.
** KEPT (pick-wt=2): 245 [] -Re6|Rc7.
** KEPT (pick-wt=2): 246 [] -Re6|Rd8.
** KEPT (pick-wt=2): 247 [] -Re6|Rf8.
** KEPT (pick-wt=2): 248 [] -Re6|Rg7.
** KEPT (pick-wt=2): 249 [] -Re6|Rg5.
** KEPT (pick-wt=2): 250 [] -Re6|Rf4.
** KEPT (pick-wt=2): 251 [] -Re6|Rd4.
** KEPT (pick-wt=2): 252 [] -Re6|Rc5.
** KEPT (pick-wt=2): 253 [] -Rf6|Rd7.
** KEPT (pick-wt=2): 254 [] -Rf6|Re8.
** KEPT (pick-wt=2): 255 [] -Rf6|Rg8.
** KEPT (pick-wt=2): 256 [] -Rf6|Rh7.
** KEPT (pick-wt=2): 257 [] -Rf6|Rh5.
** KEPT (pick-wt=2): 258 [] -Rf6|Rg4.
** KEPT (pick-wt=2): 259 [] -Rf6|Re4.
** KEPT (pick-wt=2): 260 [] -Rf6|Rd5.
** KEPT (pick-wt=2): 261 [] -Rg6|Re7.
** KEPT (pick-wt=2): 262 [] -Rg6|Rf8.
** KEPT (pick-wt=2): 263 [] -Rg6|Rh8.
** KEPT (pick-wt=2): 264 [] -Rg6|Rh4.
** KEPT (pick-wt=2): 265 [] -Rg6|Rf4.
** KEPT (pick-wt=2): 266 [] -Rg6|Re5.
** KEPT (pick-wt=2): 267 [] -Rh6|Rf7.
** KEPT (pick-wt=2): 268 [] -Rh6|Rg8.
** KEPT (pick-wt=2): 269 [] -Rh6|Rg4.
** KEPT (pick-wt=2): 270 [] -Rh6|Rf5.
** KEPT (pick-wt=2): 271 [] -Ra7|Rc8.
** KEPT (pick-wt=2): 272 [] -Ra7|Rc6.
** KEPT (pick-wt=2): 273 [] -Ra7|Rb5.
** KEPT (pick-wt=2): 274 [] -Rb7|Rd8.
** KEPT (pick-wt=2): 275 [] -Rb7|Rd6.
** KEPT (pick-wt=2): 276 [] -Rb7|Rc5.
** KEPT (pick-wt=2): 277 [] -Rb7|Ra5.
** KEPT (pick-wt=2): 278 [] -Rc7|Ra8.
** KEPT (pick-wt=2): 279 [] -Rc7|Re8.
** KEPT (pick-wt=2): 280 [] -Rc7|Re6.
** KEPT (pick-wt=2): 281 [] -Rc7|Rd5.
** KEPT (pick-wt=2): 282 [] -Rc7|Rb5.
** KEPT (pick-wt=2): 283 [] -Rc7|Ra6.
** KEPT (pick-wt=2): 284 [] -Rd7|Rb8.
** KEPT (pick-wt=2): 285 [] -Rd7|Rf8.
```

```
** KEPT (pick-wt=2): 286 [] -Rd7|Rf6.
** KEPT (pick-wt=2): 287 [] -Rd7|Re5.
** KEPT (pick-wt=2): 288 [] -Rd7|Rc5.
** KEPT (pick-wt=2): 289 [] -Rd7|Rb6.
** KEPT (pick-wt=2): 290 [] -Re7|Rc8.
** KEPT (pick-wt=2): 291 [] -Re7|Rg8.
** KEPT (pick-wt=2): 292 [] -Re7|Rg6.
** KEPT (pick-wt=2): 293 [] -Re7|Rf5.
** KEPT (pick-wt=2): 294 [] -Re7|Rd5.
** KEPT (pick-wt=2): 295 [] -Re7|Rc6.
** KEPT (pick-wt=2): 296 [] -Rf7|Rd8.
** KEPT (pick-wt=2): 297 [] -Rf7|Rh8.
** KEPT (pick-wt=2): 298 [] -Rf7|Rh6.
** KEPT (pick-wt=2): 299 [] -Rf7|Rg5.
** KEPT (pick-wt=2): 300 [] -Rf7|Re5.
** KEPT (pick-wt=2): 301 [] -Rf7|Rd6.
** KEPT (pick-wt=2): 302 [] -Rg7|Re8.
** KEPT (pick-wt=2): 303 [] -Rg7|Rh5.
** KEPT (pick-wt=2): 304 [] -Rg7|Rf5.
** KEPT (pick-wt=2): 305 [] -Rg7|Re6.
** KEPT (pick-wt=2): 306 [] -Rh7|Rf8.
** KEPT (pick-wt=2): 307 [] -Rh7|Rg5.
** KEPT (pick-wt=2): 308 [] -Rh7|Rf6.
** KEPT (pick-wt=2): 309 [] -Ra8|Rc7.
** KEPT (pick-wt=2): 310 [] -Ra8|Rb6.
** KEPT (pick-wt=2): 311 [] -Rb8|Rd7.
** KEPT (pick-wt=2): 312 [] -Rb8|Rc6.
** KEPT (pick-wt=2): 313 [] -Rb8|Ra6.
** KEPT (pick-wt=2): 314 [] -Rc8|Re7.
** KEPT (pick-wt=2): 315 [] -Rc8|Rd6.
** KEPT (pick-wt=2): 316 [] -Rc8|Rb6.
** KEPT (pick-wt=2): 317 [] -Rc8|Ra7.
** KEPT (pick-wt=2): 318 [] -Rd8|Rf7.
** KEPT (pick-wt=2): 319 [] -Rd8|Re6.
** KEPT (pick-wt=2): 320 [] -Rd8|Rc6.
** KEPT (pick-wt=2): 321 [] -Rd8|Rb7.
** KEPT (pick-wt=2): 322 [] -Re8|Rg7.
** KEPT (pick-wt=2): 323 [] -Re8|Rf6.
** KEPT (pick-wt=2): 324 [] -Re8|Rd6.
** KEPT (pick-wt=2): 325 [] -Re8|Rc7.
** KEPT (pick-wt=2): 326 [] -Rf8|Rh7.
** KEPT (pick-wt=2): 327 [] -Rf8|Rg6.
** KEPT (pick-wt=2): 328 [] -Rf8|Re6.
** KEPT (pick-wt=2): 329 [] -Rf8|Rd7.
** KEPT (pick-wt=2): 330 [] -Rg8|Rh6.
** KEPT (pick-wt=2): 331 [] -Rg8|Rf6.
```

```
** KEPT (pick-wt=2): 332 [] -Rg8|Re7.
** KEPT (pick-wt=2): 333 [] -Rh8|Rg6.
** KEPT (pick-wt=2): 334 [] -Rh8|Rf7.
** KEPT (pick-wt=64): 336 [copy,335,propositional] -Ra1| -Rb3| -Rc2| -Rb1|
-Ra3| -Rc3| -Rd2| -Rc1| -Ra2| -Rd3| -Re2| -Rd1| -Rb2| -Re3| -Rf2| -Re1| -Rf3|
-Rg2| -Rf1| -Rg3| -Rh2| -Rg1| -Rh3| -Rh1| -Rb4| -Ra4| -Rc4| -Rd4| -Re4| -Rf4|
-Rg4| -Rh4| -Rb5| -Ra5| -Rc5| -Rd5| -Re5| -Rf5| -Rg5| -Rh5| -Rb6| -Ra6| -Rc6|
-Rd6| -Re6| -Rf6| -Rg6| -Rh6| -Ra7| -Rc7| -Rb7| -Rd7| -Re7| -Rf7| -Rg7| -Rh7|
-Rb8| -Ra8| -Rc8| -Rd8| -Re8| -Rf8| -Rg8| -Rh8.

------------> process sos:
** KEPT (pick-wt=1): 337 [] Ra1.

======= end of input processing =======

=========== start of search ===========

given clause #1: (wt=1) 337 [] Ra1.

given clause #2: (wt=1) 338 [hyper,337,2] Rc2.

given clause #3: (wt=1) 339 [hyper,337,1] Rb3.

given clause #4: (wt=1) 340 [hyper,338,38] Re1.

given clause #5: (wt=1) 341 [hyper,338,37] Re3.

given clause #6: (wt=1) 342 [hyper,338,36] Rd4.

given clause #7: (wt=1) 343 [hyper,338,35] Rb4.

given clause #8: (wt=1) 344 [hyper,338,34] Ra3.

given clause #9: (wt=1) 345 [hyper,339,72] Rc1.

given clause #10: (wt=1) 346 [hyper,339,71] Rd2.

given clause #11: (wt=1) 347 [hyper,339,69] Rc5.

given clause #12: (wt=1) 348 [hyper,339,68] Ra5.

given clause #13: (wt=1) 349 [hyper,340,17] Rg2.

given clause #14: (wt=1) 350 [hyper,340,16] Rf3.

given clause #15: (wt=1) 351 [hyper,340,15] Rd3.
```

```
given clause #16: (wt=1) 352 [hyper,341,95] Rd1.

given clause #17: (wt=1) 353 [hyper,341,94] Rf1.

given clause #18: (wt=1) 354 [hyper,341,92] Rg4.

given clause #19: (wt=1) 355 [hyper,341,91] Rf5.

given clause #20: (wt=1) 356 [hyper,341,90] Rd5.

given clause #21: (wt=1) 357 [hyper,341,89] Rc4.

given clause #22: (wt=1) 358 [hyper,342,138] Re2.

given clause #23: (wt=1) 359 [hyper,342,135] Re6.

given clause #24: (wt=1) 360 [hyper,342,134] Rc6.

given clause #25: (wt=1) 361 [hyper,342,133] Rb5.

given clause #26: (wt=1) 362 [hyper,343,124] Ra2.

given clause #27: (wt=1) 363 [hyper,343,119] Ra6.

given clause #28: (wt=1) 364 [hyper,344,67] Rb1.

given clause #29: (wt=1) 365 [hyper,346,42] Re4.

given clause #30: (wt=1) 366 [hyper,347,184] Ra4.

given clause #31: (wt=1) 367 [hyper,347,179] Rd7.

given clause #32: (wt=1) 368 [hyper,347,178] Rb7.

given clause #33: (wt=1) 369 [hyper,348,167] Rb6.

given clause #34: (wt=1) 370 [hyper,349,60] Rh4.

given clause #35: (wt=1) 371 [hyper,349,59] Rf4.

given clause #36: (wt=1) 372 [hyper,350,102] Rg1.

given clause #37: (wt=1) 373 [hyper,350,101] Rh2.

given clause #38: (wt=1) 374 [hyper,350,99] Rg5.
```

```
given clause #39: (wt=1) 375 [hyper,350,98] Re5.

given clause #40: (wt=1) 376 [hyper,351,88] Rb2.

given clause #41: (wt=1) 377 [hyper,351,85] Rf2.

given clause #42: (wt=1) 378 [hyper,352,11] Rc3.

given clause #43: (wt=1) 379 [hyper,353,20] Rg3.

given clause #44: (wt=1) 380 [hyper,354,159] Rh6.

given clause #45: (wt=1) 381 [hyper,354,158] Rf6.

given clause #46: (wt=1) 382 [hyper,355,203] Rg7.

given clause #47: (wt=1) 383 [hyper,355,202] Re7.

given clause #48: (wt=1) 384 [hyper,355,201] Rd6.

given clause #49: (wt=1) 385 [hyper,356,186] Rc7.

given clause #50: (wt=1) 386 [hyper,359,247] Rf8.

given clause #51: (wt=1) 387 [hyper,359,246] Rd8.

given clause #52: (wt=1) 388 [hyper,360,230] Rb8.

given clause #53: (wt=1) 389 [hyper,360,229] Ra7.

given clause #54: (wt=1) 390 [hyper,369,224] Rc8.

given clause #55: (wt=1) 391 [hyper,369,223] Ra8.

given clause #56: (wt=1) 392 [hyper,370,164] Rg6.

given clause #57: (wt=1) 393 [hyper,371,153] Rh3.

given clause #58: (wt=1) 394 [hyper,371,152] Rh5.

given clause #59: (wt=1) 395 [hyper,374,211] Rh7.

given clause #60: (wt=1) 396 [hyper,374,210] Rf7.

given clause #61: (wt=1) 397 [hyper,377,56] Rh1.
```

```
given clause #62: (wt=1) 398 [hyper,380,268] Rg8.

given clause #63: (wt=1) 399 [hyper,381,254] Re8.

given clause #64: (wt=1) 400 [hyper,392,263] Rh8.

-----> EMPTY CLAUSE at   0.03 sec ----> 401 [hyper,400,336,337,339,338,364,344,
378,346,345,362,351,358,352,376,341,377,340,350,349,353,379,373,372,393,397,
343,366,357,342,365,371,354,370,361,348,347,356,375,355,374,394,369,363,360,
384,359,381,392,380,389,385,368,367,383,396,382,395,388,391,390,387,399,386,
398] $F.

Length of proof is 64.  Level of proof is 6.

---------------- PROOF ----------------

1 [] -Ra1|Rb3.
2 [] -Ra1|Rc2.
11 [] -Rd1|Rc3.
15 [] -Re1|Rd3.
16 [] -Re1|Rf3.
17 [] -Re1|Rg2.
20 [] -Rf1|Rg3.
34 [] -Rc2|Ra3.
35 [] -Rc2|Rb4.
36 [] -Rc2|Rd4.
37 [] -Rc2|Re3.
38 [] -Rc2|Re1.
42 [] -Rd2|Re4.
56 [] -Rf2|Rh1.
59 [] -Rg2|Rf4.
60 [] -Rg2|Rh4.
67 [] -Ra3|Rb1.
68 [] -Rb3|Ra5.
69 [] -Rb3|Rc5.
71 [] -Rb3|Rd2.
72 [] -Rb3|Rc1.
85 [] -Rd3|Rf2.
88 [] -Rd3|Rb2.
89 [] -Re3|Rc4.
90 [] -Re3|Rd5.
91 [] -Re3|Rf5.
92 [] -Re3|Rg4.
94 [] -Re3|Rf1.
95 [] -Re3|Rd1.
```

```
98 [] -Rf3|Re5.
99 [] -Rf3|Rg5.
101 [] -Rf3|Rh2.
102 [] -Rf3|Rg1.
119 [] -Rb4|Ra6.
124 [] -Rb4|Ra2.
133 [] -Rd4|Rb5.
134 [] -Rd4|Rc6.
135 [] -Rd4|Re6.
138 [] -Rd4|Re2.
152 [] -Rf4|Rh5.
153 [] -Rf4|Rh3.
158 [] -Rg4|Rf6.
159 [] -Rg4|Rh6.
164 [] -Rh4|Rg6.
167 [] -Ra5|Rb6.
178 [] -Rc5|Rb7.
179 [] -Rc5|Rd7.
184 [] -Rc5|Ra4.
186 [] -Rd5|Rc7.
201 [] -Rf5|Rd6.
202 [] -Rf5|Re7.
203 [] -Rf5|Rg7.
210 [] -Rg5|Rf7.
211 [] -Rg5|Rh7.
223 [] -Rb6|Ra8.
224 [] -Rb6|Rc8.
229 [] -Rc6|Ra7.
230 [] -Rc6|Rb8.
246 [] -Re6|Rd8.
247 [] -Re6|Rf8.
254 [] -Rf6|Re8.
263 [] -Rg6|Rh8.
268 [] -Rh6|Rg8.
335 [] -Ra1| -Rb1| -Rc1| -Rd1| -Re1| -Rf1| -Rg1| -Rh1| -Ra2| -Rb2| -Rc2| -Rd2|
-Re2| -Rf2| -Rg2| -Rh2| -Ra3| -Rb3| -Rc3| -Rd3| -Re3| -Rf3| -Rg3| -Rh3| -Ra4|
-Rb4| -Rc4| -Rd4| -Re4| -Rf4| -Rg4| -Rh4| -Ra5| -Rb5| -Rc5| -Rd5| -Re5| -Rf5|
-Rg5| -Rh5| -Ra6| -Rb6| -Rc6| -Rd6| -Re6| -Rf6| -Rg6| -Rh6| -Ra7| -Rb7| -Rc7|
-Rd7| -Re7| -Rf7| -Rg7| -Rh7| -Ra8| -Rb8| -Rc8| -Rd8| -Re8| -Rf8| -Rg8| -Rh8.
336 [copy,335,propositional] -Ra1| -Rb3| -Rc2| -Rb1| -Ra3| -Rc3| -Rd2| -Rc1|
-Ra2| -Rd3| -Re2| -Rd1| -Rb2| -Re3| -Rf2| -Re1| -Rf3| -Rg2| -Rf1| -Rg3| -Rh2|
-Rg1| -Rh3| -Rh1| -Rb4| -Ra4| -Rc4| -Rd4| -Re4| -Rf4| -Rg4| -Rh4| -Rb5| -Ra5|
-Rc5| -Rd5| -Re5| -Rf5| -Rg5| -Rh5| -Rb6| -Ra6| -Rc6| -Rd6| -Re6| -Rf6| -Rg6|
-Rh6| -Ra7| -Rc7| -Rb7| -Rd7| -Re7| -Rf7| -Rg7| -Rh7| -Rb8| -Ra8| -Rc8| -Rd8|
-Re8| -Rf8| -Rg8| -Rh8.
337 [] Ra1.
```

```
338 [hyper,337,2] Rc2.
339 [hyper,337,1] Rb3.
340 [hyper,338,38] Re1.
341 [hyper,338,37] Re3.
342 [hyper,338,36] Rd4.
343 [hyper,338,35] Rb4.
344 [hyper,338,34] Ra3.
345 [hyper,339,72] Rc1.
346 [hyper,339,71] Rd2.
347 [hyper,339,69] Rc5.
348 [hyper,339,68] Ra5.
349 [hyper,340,17] Rg2.
350 [hyper,340,16] Rf3.
351 [hyper,340,15] Rd3.
352 [hyper,341,95] Rd1.
353 [hyper,341,94] Rf1.
354 [hyper,341,92] Rg4.
355 [hyper,341,91] Rf5.
356 [hyper,341,90] Rd5.
357 [hyper,341,89] Rc4.
358 [hyper,342,138] Re2.
359 [hyper,342,135] Re6.
360 [hyper,342,134] Rc6.
361 [hyper,342,133] Rb5.
362 [hyper,343,124] Ra2.
363 [hyper,343,119] Ra6.
364 [hyper,344,67] Rb1.
365 [hyper,346,42] Re4.
366 [hyper,347,184] Ra4.
367 [hyper,347,179] Rd7.
368 [hyper,347,178] Rb7.
369 [hyper,348,167] Rb6.
370 [hyper,349,60] Rh4.
371 [hyper,349,59] Rf4.
372 [hyper,350,102] Rg1.
373 [hyper,350,101] Rh2.
374 [hyper,350,99] Rg5.
375 [hyper,350,98] Re5.
376 [hyper,351,88] Rb2.
377 [hyper,351,85] Rf2.
378 [hyper,352,11] Rc3.
379 [hyper,353,20] Rg3.
380 [hyper,354,159] Rh6.
381 [hyper,354,158] Rf6.
382 [hyper,355,203] Rg7.
383 [hyper,355,202] Re7.
```

```
384 [hyper,355,201] Rd6.
385 [hyper,356,186] Rc7.
386 [hyper,359,247] Rf8.
387 [hyper,359,246] Rd8.
388 [hyper,360,230] Rb8.
389 [hyper,360,229] Ra7.
390 [hyper,369,224] Rc8.
391 [hyper,369,223] Ra8.
392 [hyper,370,164] Rg6.
393 [hyper,371,153] Rh3.
394 [hyper,371,152] Rh5.
395 [hyper,374,211] Rh7.
396 [hyper,374,210] Rf7.
397 [hyper,377,56] Rh1.
398 [hyper,380,268] Rg8.
399 [hyper,381,254] Re8.
400 [hyper,392,263] Rh8.
401 [hyper,400,336,337,339,338,364,344,378,346,345,362,351,358,352,376,341,377,
340,350,349,353,379,373,372,393,397,343,366,357,342,365,371,354,370,361,348,
347,356,375,355,374,394,369,363,360,384,359,381,392,380,389,385,368,367,383,
396,382,395,388,391,390,387,399,386,398] $F.

------------ end of proof -------------


Search stopped by max_proofs option.

============ end of search ============

-------------- statistics -------------
clauses given                 64
clauses generated             64
clauses kept                 399
clauses forward subsumed       2
clauses back subsumed        334
Kbytes malloced              255

----------- times (seconds) -----------
user CPU time          0.03          (0 hr, 0 min, 0 sec)
system CPU time        0.01          (0 hr, 0 min, 0 sec)
wall-clock time        0             (0 hr, 0 min, 0 sec)
hyper_res time         0.00
for_sub time           0.00
back_sub time          0.00
conflict time          0.00
demod time             0.00
```

```
That finishes the proof of the theorem.

Process 28078 finished Fri Nov  5 11:34:13 2004
```

4.3 labyrin.in

```
set(auto).
formula_list(usable).
% Section A: database
R45.

% Section B: definitions
R1 -> (R4).
R2 -> (R8).
R3 -> (R4 & R10).
R4 -> (R1 & R3 & R5).
R5 -> (R4).
R6 -> (R13).
R7 -> (R8).
R8 -> (R2 & R7 & R9).
R9 -> (R8 & R16).
R10 -> (R3 & R17).
R11 -> (R12).
R12 -> (R11 & R13).
R13 -> (R6 & R12 & R14).
R14 -> (R13 & R15).
R15 -> (R14 & R22).
R16 -> (R9 & R23).
R17 -> (R10 & R18).
R18 -> (R17 & R19).
R19 -> (R18 & R20).
R20 -> (R19 & R21).
R21 -> (R20 & R22).
R22 -> (R15 & R21).
R23 -> (R16 & R30).
R24 -> (R25).
R25 -> (R24 & R26).
R26 -> (R25 & R27).
R27 -> (R26 & R28).
R28 -> (R27 & R29 & R35).
R29 -> (R28 & R30).
R30 -> (R23 & R29).
R31 -> (R32 & R38).
R32 -> (R31 & R33).
```

```
R33 -> (R32 & R34).
R34 -> (R33 & R35).
R35 -> (R28 & R34).
R36 -> (R37 & R43).
R37 -> (R36 & R44).
R38 -> (R31 & R39).
R39 -> (R38 & R40).
R40 -> (R39 & R41).
R41 -> (R40 & R42).
R42 -> (R41 & R43).
R43 -> (R36 & R42).
R44 -> (R37).
R45 -> (R43).

%Section C: negation of the query
-R2.

end_of_list.
```

4.4 labyrin1.out

```
----- Otter 3.0.6, April 2000 -----
The process was started by mmalinen on vroomfondel.hut.fi, Tue Nov 16 12:46:27
2004
The command was "/p/edu/tik-79.144/bin/otter.linux".  The process ID is 1730.

set(auto).
   dependent: set(auto1).
   dependent: set(process_input).
   dependent: clear(print_kept).
   dependent: clear(print_new_demod).
   dependent: clear(print_back_demod).
   dependent: clear(print_back_sub).
   dependent: set(control_memory).
   dependent: assign(max_mem, 12000).
   dependent: assign(pick_given_ratio, 4).
   dependent: assign(stats_level, 1).
   dependent: assign(max_seconds, 10800).

formula_list(usable).
R45.
R1->R4.
R2->R8.
R3->R4&R10.
R4->R1&R3&R5.
```

```
R5->R4.
R6->R13.
R7->R8.
R8->R2&R7&R9.
R9->R8&R16.
R10->R3&R17.
R11->R12.
R12->R11&R13.
R13->R6&R12&R14.
R14->R13&R15.
R15->R14&R22.
R16->R9&R23.
R17->R10&R18.
R18->R17&R19.
R19->R18&R20.
R20->R19&R21.
R21->R20&R22.
R22->R15&R21.
R23->R16&R30.
R24->R25.
R25->R24&R26.
R26->R25&R27.
R27->R26&R28.
R28->R27&R29&R35.
R29->R28&R30.
R30->R23&R29.
R31->R32&R38.
R32->R31&R33.
R33->R32&R34.
R34->R33&R35.
R35->R28&R34.
R36->R37&R43.
R37->R36&R44.
R38->R31&R39.
R39->R38&R40.
R40->R39&R41.
R41->R40&R42.
R42->R41&R43.
R43->R36&R42.
R44->R37.
R45->R43.
-R2.
end_of_list.

-------> usable clausifies to:
```

```
list(usable).
0 [] R45.
0 [] -R1|R4.
0 [] -R2|R8.
0 [] -R3|R4.
0 [] -R3|R10.
0 [] -R4|R1.
0 [] -R4|R3.
0 [] -R4|R5.
0 [] -R5|R4.
0 [] -R6|R13.
0 [] -R7|R8.
0 [] -R8|R2.
0 [] -R8|R7.
0 [] -R8|R9.
0 [] -R9|R8.
0 [] -R9|R16.
0 [] -R10|R3.
0 [] -R10|R17.
0 [] -R11|R12.
0 [] -R12|R11.
0 [] -R12|R13.
0 [] -R13|R6.
0 [] -R13|R12.
0 [] -R13|R14.
0 [] -R14|R13.
0 [] -R14|R15.
0 [] -R15|R14.
0 [] -R15|R22.
0 [] -R16|R9.
0 [] -R16|R23.
0 [] -R17|R10.
0 [] -R17|R18.
0 [] -R18|R17.
0 [] -R18|R19.
0 [] -R19|R18.
0 [] -R19|R20.
0 [] -R20|R19.
0 [] -R20|R21.
0 [] -R21|R20.
0 [] -R21|R22.
0 [] -R22|R15.
0 [] -R22|R21.
0 [] -R23|R16.
0 [] -R23|R30.
0 [] -R24|R25.
```

```
0 [] -R25|R24.
0 [] -R25|R26.
0 [] -R26|R25.
0 [] -R26|R27.
0 [] -R27|R26.
0 [] -R27|R28.
0 [] -R28|R27.
0 [] -R28|R29.
0 [] -R28|R35.
0 [] -R29|R28.
0 [] -R29|R30.
0 [] -R30|R23.
0 [] -R30|R29.
0 [] -R31|R32.
0 [] -R31|R38.
0 [] -R32|R31.
0 [] -R32|R33.
0 [] -R33|R32.
0 [] -R33|R34.
0 [] -R34|R33.
0 [] -R34|R35.
0 [] -R35|R28.
0 [] -R35|R34.
0 [] -R36|R37.
0 [] -R36|R43.
0 [] -R37|R36.
0 [] -R37|R44.
0 [] -R38|R31.
0 [] -R38|R39.
0 [] -R39|R38.
0 [] -R39|R40.
0 [] -R40|R39.
0 [] -R40|R41.
0 [] -R41|R40.
0 [] -R41|R42.
0 [] -R42|R41.
0 [] -R42|R43.
0 [] -R43|R36.
0 [] -R43|R42.
0 [] -R44|R37.
0 [] -R45|R43.
0 [] -R2.
end_of_list.

SCAN INPUT: prop=1, horn=1, equality=0, symmetry=0, max_lits=2.
```

```
The clause set is propositional; the strategy will be
ordered hyperresolution with the propositional
optimizations, with satellites in sos and nuclei in usable.

   dependent: set(hyper_res).
   dependent: set(propositional).
   dependent: set(sort_literals).

------------> process usable:
** KEPT (pick-wt=2): 1 [] -R1|R4.
** KEPT (pick-wt=2): 2 [] -R2|R8.
** KEPT (pick-wt=2): 3 [] -R3|R4.
** KEPT (pick-wt=2): 4 [] -R3|R10.
** KEPT (pick-wt=2): 5 [] -R4|R1.
** KEPT (pick-wt=2): 6 [] -R4|R3.
** KEPT (pick-wt=2): 7 [] -R4|R5.
** KEPT (pick-wt=2): 8 [] -R5|R4.
** KEPT (pick-wt=2): 9 [] -R6|R13.
** KEPT (pick-wt=2): 10 [] -R7|R8.
** KEPT (pick-wt=2): 11 [] -R8|R2.
** KEPT (pick-wt=2): 12 [] -R8|R7.
** KEPT (pick-wt=2): 13 [] -R8|R9.
** KEPT (pick-wt=2): 14 [] -R9|R8.
** KEPT (pick-wt=2): 15 [] -R9|R16.
** KEPT (pick-wt=2): 16 [] -R10|R3.
** KEPT (pick-wt=2): 17 [] -R10|R17.
** KEPT (pick-wt=2): 18 [] -R11|R12.
** KEPT (pick-wt=2): 19 [] -R12|R11.
** KEPT (pick-wt=2): 20 [] -R12|R13.
** KEPT (pick-wt=2): 21 [] -R13|R6.
** KEPT (pick-wt=2): 22 [] -R13|R12.
** KEPT (pick-wt=2): 23 [] -R13|R14.
** KEPT (pick-wt=2): 24 [] -R14|R13.
** KEPT (pick-wt=2): 25 [] -R14|R15.
** KEPT (pick-wt=2): 26 [] -R15|R14.
** KEPT (pick-wt=2): 27 [] -R15|R22.
** KEPT (pick-wt=2): 28 [] -R16|R9.
** KEPT (pick-wt=2): 29 [] -R16|R23.
** KEPT (pick-wt=2): 30 [] -R17|R10.
** KEPT (pick-wt=2): 31 [] -R17|R18.
** KEPT (pick-wt=2): 32 [] -R18|R17.
** KEPT (pick-wt=2): 33 [] -R18|R19.
** KEPT (pick-wt=2): 34 [] -R19|R18.
** KEPT (pick-wt=2): 35 [] -R19|R20.
** KEPT (pick-wt=2): 36 [] -R20|R19.
** KEPT (pick-wt=2): 37 [] -R20|R21.
```

```
** KEPT (pick-wt=2): 38 [] -R21|R20.
** KEPT (pick-wt=2): 39 [] -R21|R22.
** KEPT (pick-wt=2): 40 [] -R22|R15.
** KEPT (pick-wt=2): 41 [] -R22|R21.
** KEPT (pick-wt=2): 42 [] -R23|R16.
** KEPT (pick-wt=2): 43 [] -R23|R30.
** KEPT (pick-wt=2): 44 [] -R24|R25.
** KEPT (pick-wt=2): 45 [] -R25|R24.
** KEPT (pick-wt=2): 46 [] -R25|R26.
** KEPT (pick-wt=2): 47 [] -R26|R25.
** KEPT (pick-wt=2): 48 [] -R26|R27.
** KEPT (pick-wt=2): 49 [] -R27|R26.
** KEPT (pick-wt=2): 50 [] -R27|R28.
** KEPT (pick-wt=2): 51 [] -R28|R27.
** KEPT (pick-wt=2): 52 [] -R28|R29.
** KEPT (pick-wt=2): 53 [] -R28|R35.
** KEPT (pick-wt=2): 54 [] -R29|R28.
** KEPT (pick-wt=2): 55 [] -R29|R30.
** KEPT (pick-wt=2): 56 [] -R30|R23.
** KEPT (pick-wt=2): 57 [] -R30|R29.
** KEPT (pick-wt=2): 58 [] -R31|R32.
** KEPT (pick-wt=2): 59 [] -R31|R38.
** KEPT (pick-wt=2): 60 [] -R32|R31.
** KEPT (pick-wt=2): 61 [] -R32|R33.
** KEPT (pick-wt=2): 62 [] -R33|R32.
** KEPT (pick-wt=2): 63 [] -R33|R34.
** KEPT (pick-wt=2): 64 [] -R34|R33.
** KEPT (pick-wt=2): 65 [] -R34|R35.
** KEPT (pick-wt=2): 66 [] -R35|R28.
** KEPT (pick-wt=2): 67 [] -R35|R34.
** KEPT (pick-wt=2): 68 [] -R36|R37.
** KEPT (pick-wt=2): 69 [] -R36|R43.
** KEPT (pick-wt=2): 70 [] -R37|R36.
** KEPT (pick-wt=2): 71 [] -R37|R44.
** KEPT (pick-wt=2): 72 [] -R38|R31.
** KEPT (pick-wt=2): 73 [] -R38|R39.
** KEPT (pick-wt=2): 74 [] -R39|R38.
** KEPT (pick-wt=2): 75 [] -R39|R40.
** KEPT (pick-wt=2): 76 [] -R40|R39.
** KEPT (pick-wt=2): 77 [] -R40|R41.
** KEPT (pick-wt=2): 78 [] -R41|R40.
** KEPT (pick-wt=2): 79 [] -R41|R42.
** KEPT (pick-wt=2): 80 [] -R42|R41.
** KEPT (pick-wt=2): 81 [] -R42|R43.
** KEPT (pick-wt=2): 82 [] -R43|R36.
** KEPT (pick-wt=2): 83 [] -R43|R42.
```

```
** KEPT (pick-wt=2): 84 [] -R44|R37.
** KEPT (pick-wt=2): 85 [] -R45|R43.
** KEPT (pick-wt=1): 86 [] -R2.
86 back subsumes 2.

------------> process sos:
** KEPT (pick-wt=1): 87 [] R45.

======= end of input processing =======

=========== start of search ===========

given clause #1: (wt=1) 87 [] R45.

given clause #2: (wt=1) 88 [hyper,87,85] R43.

given clause #3: (wt=1) 89 [hyper,88,83] R42.

given clause #4: (wt=1) 90 [hyper,88,82] R36.

given clause #5: (wt=1) 91 [hyper,89,80] R41.

given clause #6: (wt=1) 92 [hyper,90,68] R37.

given clause #7: (wt=1) 93 [hyper,91,78] R40.

given clause #8: (wt=1) 94 [hyper,92,71] R44.

given clause #9: (wt=1) 95 [hyper,93,76] R39.

given clause #10: (wt=1) 96 [hyper,95,74] R38.

given clause #11: (wt=1) 97 [hyper,96,72] R31.

given clause #12: (wt=1) 98 [hyper,97,58] R32.

given clause #13: (wt=1) 99 [hyper,98,61] R33.

given clause #14: (wt=1) 100 [hyper,99,63] R34.

given clause #15: (wt=1) 101 [hyper,100,65] R35.

given clause #16: (wt=1) 102 [hyper,101,66] R28.

given clause #17: (wt=1) 103 [hyper,102,52] R29.
```

```
given clause #18: (wt=1) 104 [hyper,102,51] R27.

given clause #19: (wt=1) 105 [hyper,103,55] R30.

given clause #20: (wt=1) 106 [hyper,104,49] R26.

given clause #21: (wt=1) 107 [hyper,105,56] R23.

given clause #22: (wt=1) 108 [hyper,106,47] R25.

given clause #23: (wt=1) 109 [hyper,107,42] R16.

given clause #24: (wt=1) 110 [hyper,108,45] R24.

given clause #25: (wt=1) 111 [hyper,109,28] R9.

given clause #26: (wt=1) 112 [hyper,111,14] R8.

----> UNIT CONFLICT at   0.01 sec ----> 115 [binary,114.1,86.1] $F.

Length of proof is 19.  Level of proof is 19.

---------------- PROOF ----------------

11 [] -R8|R2.
14 [] -R9|R8.
28 [] -R16|R9.
42 [] -R23|R16.
52 [] -R28|R29.
55 [] -R29|R30.
56 [] -R30|R23.
58 [] -R31|R32.
61 [] -R32|R33.
63 [] -R33|R34.
65 [] -R34|R35.
66 [] -R35|R28.
72 [] -R38|R31.
74 [] -R39|R38.
76 [] -R40|R39.
78 [] -R41|R40.
80 [] -R42|R41.
83 [] -R43|R42.
85 [] -R45|R43.
86 [] -R2.
87 [] R45.
88 [hyper,87,85] R43.
```

```
89 [hyper,88,83] R42.
91 [hyper,89,80] R41.
93 [hyper,91,78] R40.
95 [hyper,93,76] R39.
96 [hyper,95,74] R38.
97 [hyper,96,72] R31.
98 [hyper,97,58] R32.
99 [hyper,98,61] R33.
100 [hyper,99,63] R34.
101 [hyper,100,65] R35.
102 [hyper,101,66] R28.
103 [hyper,102,52] R29.
105 [hyper,103,55] R30.
107 [hyper,105,56] R23.
109 [hyper,107,42] R16.
111 [hyper,109,28] R9.
112 [hyper,111,14] R8.
114 [hyper,112,11] R2.
115 [binary,114.1,86.1] $F.

------------ end of proof -------------


Search stopped by max_proofs option.

============ end of search ============

-------------- statistics -------------
clauses given                 26
clauses generated             27
clauses kept                 114
clauses forward subsumed       0
clauses back subsumed         51
Kbytes malloced              127

----------- times (seconds) -----------
user CPU time          0.01          (0 hr, 0 min, 0 sec)
system CPU time        0.00          (0 hr, 0 min, 0 sec)
wall-clock time        0             (0 hr, 0 min, 0 sec)
hyper_res time         0.00
for_sub time           0.00
back_sub time          0.00
conflict time          0.00
demod time             0.00

That finishes the proof of the theorem.
```

```
Process 1730 finished Tue Nov 16 12:46:27 2004
```

On Equality Proving of Mathematical Expressions

Mikko Malinen

26th July, 2004

Abstract

In this paper we introduce two equality proving methods for mathematical expressions: normal form method and proof system method.

1 Introduction

First one definition

Definition 1.1 *Two expressions are similar when their appearance is same.*

Example 1.1 $x+2-1$ *and* $x+1$ *are equal but not similar.*
$x+2-1$ *and* $x+2-1$ *are equal and similar.*

Theorem 1.1 *Two similar expressions are equal.*

2 Normal Form Method

Expression in normal form is similar to every equal expression in normal form. Therefore, when two expressions can be written in normal form, can be noticed if they are equal. For example, let's consider expressions, which consist of terms of the form ax^b, where a and b are integers. Tho expressions of that kind are $x+2+x+3x^2$ and $3+2x+3x^2-1$. If we set the rules of normal form so that terms of equal powers are summed and the sums are arranged in descending powers, then we get the normal form $3x^2+2x+2$ from both expressinons. We can say that both expressions equal. More complicated expressiond need more complicated rules. Some things to be considered are: What is the normal form of expressions like $(x+1)^{1000}$? Is it like $(x+1)^{1000}$ or like $x^{1000}+1000x^{999}+...+1000x+1$? What is the normal form of expressions like $\cosh x$? Is it $\cosh x$ or $\frac{e^x+e^{-x}}{2}$, which equal? What is the normal form of expressions like $\sum_{i=0}^{\infty}\frac{1}{2^i}$? Is it $\sum_{i=0}^{\infty}\frac{1}{2^i}$ or even 2 which is the sum of the geometric series?

3 Proof System Method

In proof system method we introduce rules. One example of a rule could be

$$\frac{a+b}{c} \tag{1}$$

Here a,b and c are integers. Any occurrence of *integer* + *integer* (a+b) may be replaced by their sum c. Applying this rule to 1+2 would yield 3. Rules are equality preserving. Applying these rules to an expression it may be possible to prove an equality to another expression. If the equality can not be found, these expressions may still be equal. For a given expression there are infinite number of equivalent expressions. Any number of rules do not cover all these.

Theorem 3.1 *For a given expression there are infinite number of equivalent expressions.*

Proof. Let's write down equivalent expressions
<*expression*>
<*expression*>+1-1
<*expression*>+2-2
...
This series could be continued infinitely. □

Double checking extends the set of equalities to be found. By double checking we mean that rules are applied to both expressions.

Theorem 3.2 *Double checking extends the set of equalities to be found.*

Proof. By double checking we find all the equalities that we do with single checking. By double checking we find also equalities like following:
Let's consider expressions $x + 1 + 3$ and $x + 2 + 2$. Applying the addition rule (1) to both we get $x + 4$ and $x + 4$. These are similar and thus equal. This we could not have found with single checking. □

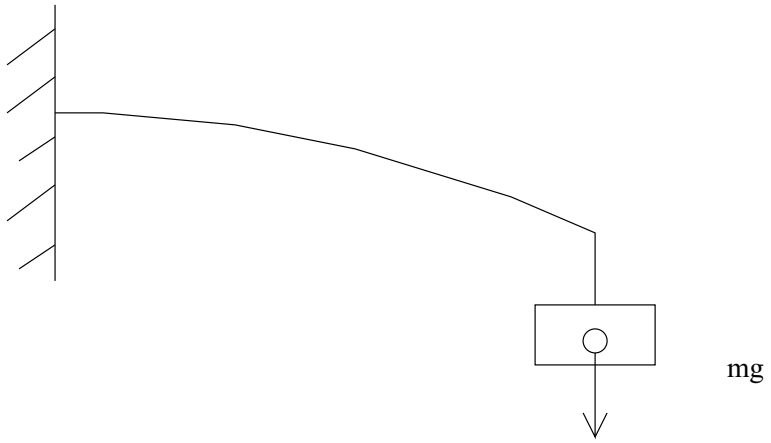
Parametrization of a Bent Rod

Mikko Malinen

15th January, 2003

1 Introduction

A flexible rod is fixed to the wall from the other end. In the other end there is a mass m. The rod, which has initially been horizontal, is now bent. See picture 1.



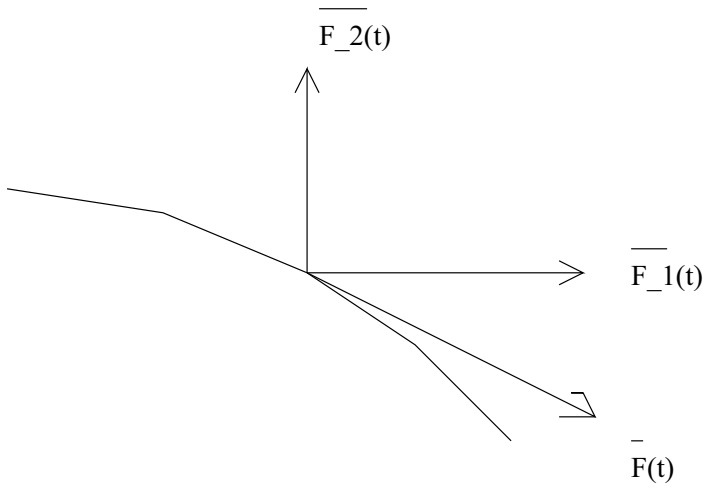
Picture 1.

In this paper we calculate the parametrization for the curve, which goes along the rod.

2 Solution

2.1 Beginning of the Solution

Let us choose directions for the components of $\bar{F}(t) = F_1(t)\bar{i} + F_2(t)\bar{j}$



Picture 2.

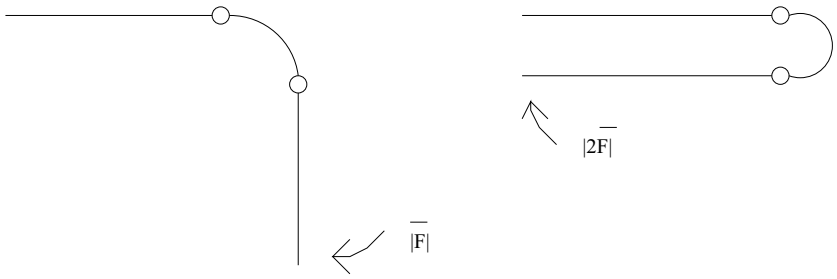
The *curvature* s of $\bar{F}(t)$ is

$$s = |\frac{d\bar{F}(t)}{dt}| = \sqrt{(\frac{dF_1(t)}{dt})^2 + (\frac{dF_2(t)}{dt})^2} \tag{1}$$

2.2 A Proof

THEOREM. The curvature is proportional to the bending momentum.

PROOF. Let us have two inflexible sticks connected together with a piece of flexible material of length l. When bent so that the sticks are in 90^0 angle to each other, we need force $|\bar{F}|$ at the end of the stick. When bent more, so that the sticks are in 0^0 to each other, we need a force $|2\bar{F}|$. (See picture 3.)



Picture 3.

In the first case

$$l = 2\pi R_1/4$$

$$R_1 = \frac{4l}{2\pi}$$

In the other case

$$l = 2\pi R_2/2$$

$$R_2 = \frac{2l}{2\pi}$$

So

$$R_2 = \frac{1}{2}R_1$$

The curvature is

$$s = \frac{1}{R}$$

So

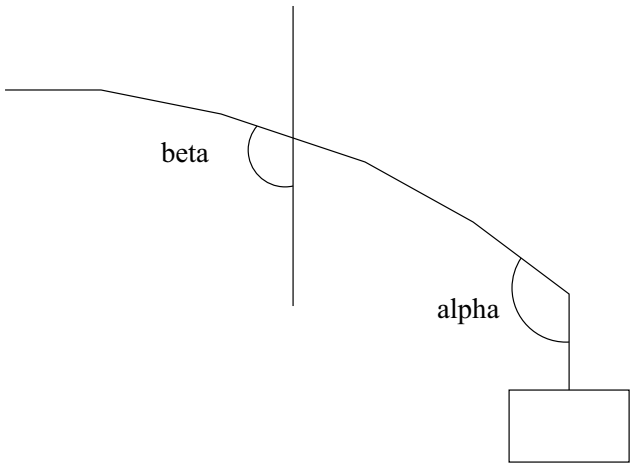
$$\frac{1}{s_2} = \frac{1}{2}\frac{1}{s_1}$$

$$s_1 = \frac{1}{2}s_2$$

The proof is concluded.

2.3 The Solution Continues

So far we know that we can calculate the curvature by knowing the momentum. Since we have a bent rod, the angle of momentum varies with the affecting force. We have to apply a correcting term $\sin\alpha$, which takes in to account the angle between the end of the rod and the affecting force, which is not the 90^0. 90^0 would not need any correcting term.



Picture 4.

$$\sin\alpha = \sin(\cos^{-1}(F_2(t)\cdot -1)) \tag{2}$$

(Because because the definition of the dot product we have

$$F(t)\cdot(0,-1) = |F(t)|\cdot 1\cdot\cos\alpha$$

)
(2) $\Rightarrow$

$$\sin\alpha = \sqrt{1-F_2(t)^2}$$

When we want to calculate the curvature at a point not at the end of the rod, we have to apply another correcting term

$$\sin\beta = \sqrt{1-F_2(t)^2}$$

Now the curvature at the point t, is a proportionality constant a times (total length in x-direction - total length in x-direction at point t) times mg times correcting term $\sin\alpha$ times correcting term $\sin\beta$. t_1 is the value of t at the end of the rod.

$$s = a\cdot(t_1-t)\cdot mg\cdot\sqrt{1-F_2(t)^2}\cdot\sqrt{1-F_2(t_1)^2}$$

Since $\sqrt{1-F_2(t_1)^2}$ is constant we can write

$$b = a\cdot\sqrt{1-F_2(t_1)^2}$$

$\Rightarrow$

$$s = b \cdot (t_1 - t) \cdot mg \cdot \sqrt{1 - F_2(t)^2} \tag{3}$$

$$|\bar{F}| = \sqrt{F_2^2 + F_1^2}$$
$$1 = F_2(t)^2 + F_1(t)^2$$
$$F_2(t)^2 = 1 - F_1(t)^2$$

$\Rightarrow$

$$F_2(t) = \sqrt{1 - F_1(t)^2} \tag{4}$$

Now, combining 1,3 and 4 we get the equation

$$\sqrt{(\frac{dF_1(t)}{dt})^2 + (\frac{d\sqrt{1 - F_1(t)^2}}{dt})^2} = b \cdot (t_1 - t) \cdot mg \cdot \sqrt{1 - (\sqrt{1 - F_1(t)^2})^2}$$

$$(\frac{dF_1(t)}{dt})^2 + (\frac{d\sqrt{1 - F_1(t)^2}}{dt})^2 = b^2 \cdot (t_1 - t)^2 \cdot m^2 g^2 \cdot F_1(t)^2$$

$$(\frac{dF_1(t)}{dt})^2 + (\frac{1}{2} \cdot \frac{1}{\sqrt{1 - F_1(t)^2}} \cdot 2 \cdot F_1(t) \cdot \frac{dF_1(t)}{dt})^2 = b^2 \cdot (t_1 - t)^2 \cdot m^2 g^2 \cdot F_1(t)^2$$

$$(\frac{dF_1(t)}{dt})^2 + \frac{1}{1 - F_1(t)^2} \cdot F_1(t)^2 \cdot (\frac{dF_1(t)}{dt})^2 = b^2 \cdot (t_1 - t)^2 \cdot m^2 \cdot g^2 \cdot F_1(t)^2$$

$$(\frac{1}{1 - F_1(t)^2} \cdot F_1(t)^2 + 1) \cdot (\frac{dF_1(t)}{dt})^2 = b^2 \cdot (t_1 - t)^2 \cdot m^2 g^2 \cdot F_1(t)^2$$

$$\frac{1}{1 - F_1(t)^2} \cdot (\frac{dF_1(t)}{dt})^2 = b^2 \cdot (t_1 - t)^2 \cdot m^2 g^2 \cdot F_1(t)^2$$

$$\frac{dF_1(t)}{dt} = b \cdot (t_1 - t) \cdot mg \cdot F_1(t) \sqrt{1 - F_1(t)^2} \tag{5}$$

Solving the differential equation 5 gives

$$F_1(t) = \frac{2e^{agt_1 mt + \frac{1}{2} agmt^2 + C}}{e^{agmt^2 + e^{2agt_1 mt + 2C}}}$$

Boundary value $F_1(0) = 1 \Rightarrow$

$$1 = \frac{2e^C}{1 + e^{2C}}$$

$$1 + e^{2C} = 2e^{C}$$

$$e^{2C} - 2e^{C} + 1 = 0$$

Let us write $v = e^{C}$

$$v^2 - 2v + 1 = 0$$

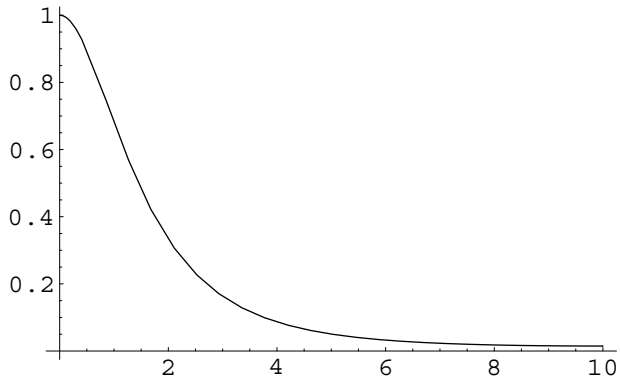
$$v = \frac{2 \pm \sqrt{4-4}}{2} = 1$$

$$e^{C} = 1$$

$$C = ln1 = 0$$

$\Rightarrow$

$$F_1(t) = \frac{2e^{agt_1 mt + \frac{1}{2}agmt^2}}{e^{agmt^2} + e^{2agt_1 mt}}$$



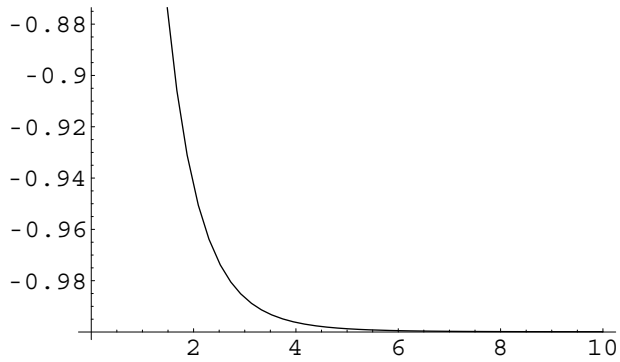
Picture 5. $F_1(t)$. Parameters $a = 0.01$,$g = 9.81$,$t = 10$ and $m = 1$.

Let us calculate $F_2(t)$ with the help of $F_1(t)$.

$$\sqrt{F_1(t)^2 + F_2(t)^2} = 1$$

$$F_2(t)^2 = 1 - F_1(t)^2 = 1 - (\frac{2e^{agt_1 mt + \frac{1}{2}agmt^2}}{e^{agmt^2} + e^{2agt_1 mt}})^2$$

$$F_2(t) = -\sqrt{1 - (\frac{2e^{agt_1 mt + \frac{1}{2}agmt^2}}{e^{agmt^2} + e^{2agt_1 mt}})^2}$$



Picture 6. $F_2(t)$. Parameters as in Picture 5.

3 Conclusion

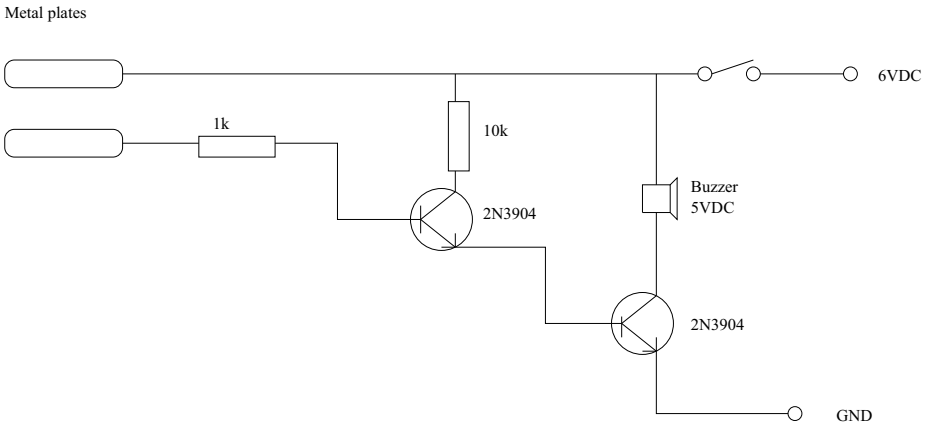
We calculated the expressions for both $F_1(t)$ and $F_2(t)$. It is now shown that these expressions may be calculated. It is impossible to express x or y as a function of time or y as a function of x. In the former case it would need integration of $F_1(t)$ and $F_2(t)$ with respect to t and in the latter case solving of t as a function of x. The writer recommends to use numerical integration when these expressions are needed.

Circuit Diagrams II

Mikko Malinen

6th October,2003

1 Water Level Indicator



Energy of a Lateral Spring

Mikko Malinen

25th October, 2003

1 Introduction

In this paper we discuss lateral springs (picture 1). We try to find the optimal thickness in order to gain maximal energy in a fully stretched spring.

Picture 1. Lateral spring (a) no stretch (b) fully stretched

2 Solution

For a bent rod, the curvature is proportional to the bending momentum. This is proved in [1]. As is known, the curvature s is the inverse of the radius of curvature R

$$s = \frac{1}{R}$$

So, the radius of curvature is inversely proportional to bending momentum and therefore inversely proportional to energy.

Therefore we may reason that the maximum energy of the spring is independent of the thickness of the spring.

3 References

[1] Malinen, M. Parametrization of a Bent Rod. 2003.

Value of a Variable Can Be Read From an Indicator

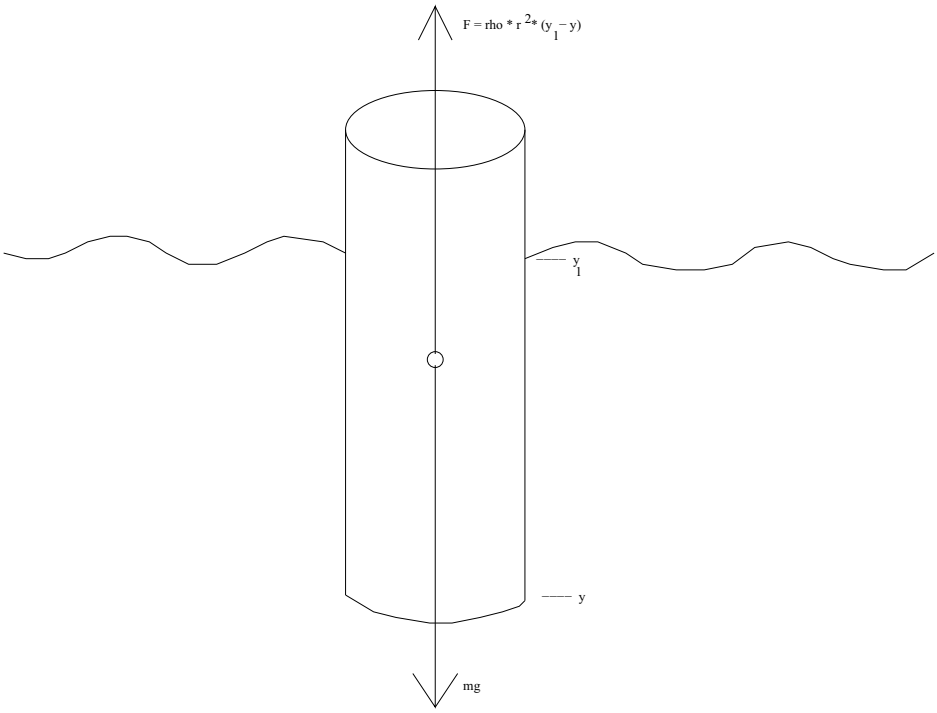
Mikko Malinen

7th January, 2002

1 Abstract

This paper presents a case represented in elementary physics, in which the value of a variable can not be directly measured. Instead, the value can be calculated with the help of another variable and its time derivatives. In the first case we try to measure the water level with a floating float. We will see that we need the float position value and its second derivative to calculate the water level. As another case we could consider a temperarture sensor and try to measure rapid changes in temperature. As we know, it takes a while since the measurement value reaches the actual temperature value.

2 The Float



Picture 1. A float in the water

Variables in picture 1:
ρ density of water
r radius of the float
y_1 water level
y float position (in vertical direction)

Consider a cylindrical float in the water. We want to know the water level and calculate it from the position of the float. The total force affecting the float is the sum of the forces in picture 1:

$$F_1 = \rho \cdot \pi r^2 \cdot (y_1 - y) - mg$$

We do not take in to account the water resistance. Newton's first law states that

$$F_1 = ma$$

(1) ans (2) lead to

$$ma = \rho \cdot \pi r^2 (y_1 - y) - mg$$

Since acceleration a is the second time derivative of y, we can write

$$my'' - \pi r^2 (y_1 - y) - mg = 0$$

Solving this for y_1 leads to

$$y_1 = \frac{my''}{\pi r^2} + y - \frac{mg}{\pi r^2}$$

This may be written

$$y_1 = C_1 y'' + y - C_2$$

for simplicity where C_1 and C_2 are constants. This is our result. The water level depends on the second derivative of the float position and on the actual float position.

3 Conclusions

It has been shown that in some cases the value of a variable may be calculated from another variable. This other variable may be the only one available and the actual variable may be "hidden" but may be found knowing some physical laws. In the case presented the value of a variable was found by using time derivatives of the indicator variable.

A System for Removing the Room Echo

Mikko Malinen

18th July,2002

This paper shows that for a transfer function H there exists always an inverse function H^{-1}. Therefore, a signal that has been altered by a transfer function can always be restored to the original signal.

PROOF.
Let the H be $\frac{1+2z^{-1}+3z^{-2}}{1+3z^{-1}}$
Then

$$H \cdot H^{-1} = \frac{1+2z^{-1}+3z^{-2}}{1+3z^{-1}} \cdot \frac{1+3z^{-1}}{1+2z^{-1}+3z^{-2}} = 1$$

The same for all other Hs.

A Search "Algorithm" for Finding the Sum of a Series

Mikko Malinen

18th July,2002

A series is given

$$\sum_{k=0}^{n} < term >$$

Find a formula for the sum, when $n = 0$

Investigate with the mathematical induction, if the formula is valid for all n.

A Theoretical Sine Wave Oscillator

Mikko Malinen

May 28th, 2001

1 Abstract

This paper introduces a way of making an electrical sine wave oscillator with help of the constant-coefficient homogenous second order differential equation and electrical integrators. Same ideas may be used to solve other differential equations electrically [1].

2 Oscillator

The constant-coefficient homogenous second order differential equation is of the form

$$ay'' + by' + cy = 0 \tag{1}$$

This equation can be solved by solving the characteristic equation

$$a\lambda^2 + b\lambda + c = 0$$

for λ and placing the result to the certain general solution of (1). Now

$$\lambda = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

When $b^2 - 4ac < 0$, $\lambda = \alpha \pm \beta i$, and the general solution is

$$y = e^{\alpha t}(A\cos(\beta t) + B\sin(\beta t)) = e^{\alpha t} \cdot C\sin(\beta t + \phi)$$

For example, if $a = 1$,$b = 0$ and $c = 1$:

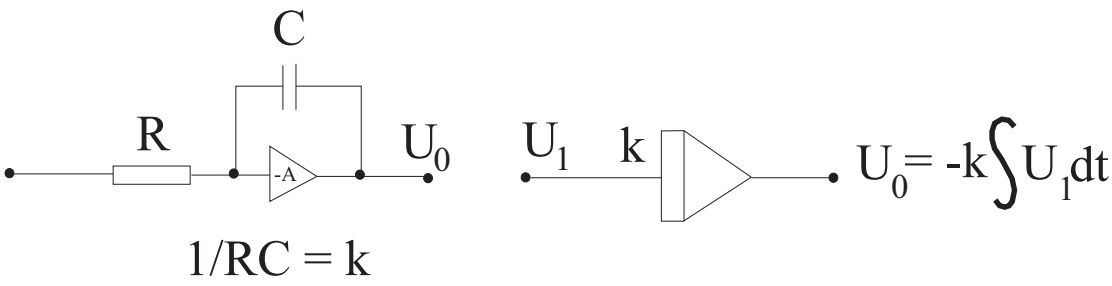
$$y'' + y = 0$$

$$\Rightarrow y = e^{0 \cdot t} \cdot C\sin(t + \phi) = C\sin(t + \phi)$$

Noticing that in this case $2\pi ft = t \Rightarrow f = \frac{1}{2\pi}$ we see that y oscillates as a sine wave with amplitude C and frequency $\frac{1}{2\pi}$.

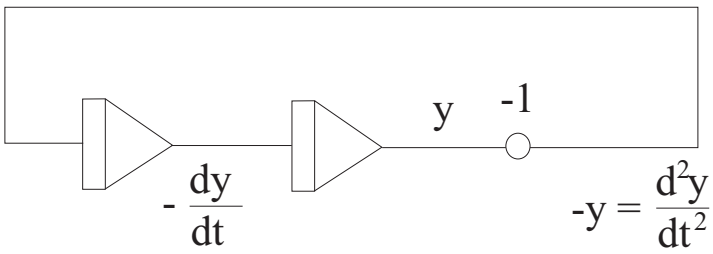
3 The circuit

[1] says: "Differentiating is technically difficult, so the equations to be solved are tried to write so that they do not contain derivatives". For the oscillator we need electrical integrators (picture).



Picture 1. Operation amplifier as an integrator (left:the circuit,right:the symbol)[1]

The complete oscillator is seen in the picture 2.



Picture 2. The electrical sine wave oscillator.

4 Problems

This is a theoretical view only so the actual circuit has not been built by the author. Probably the oscillator works, but with amplitude $C = 0$, since the initial values for y, dy/dt and d^2y/dt^2 are all zero. It is difficult to set the correct initial values to the circuit.

5 References

[1] Uusi tietosanakirja, part 13, topic ”Matematiikkakoneet”,Tietosanakirja Oy,Helsinki 1963

A One-coil Voltage Source And Measurement Of Small Inductances

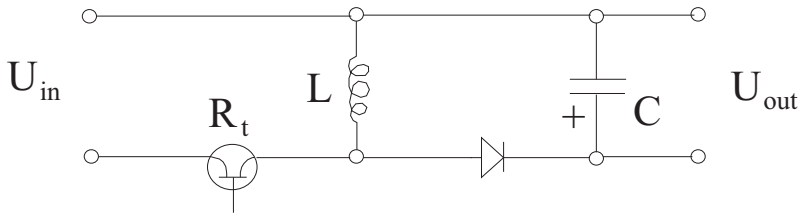
Mikko Malinen

May 28th, 2001

1 Abstract

This paper presents an idea of a one-coil voltage source and one application for it, using it for measurement of an inductance. With this equipment also small inductances may be measured.

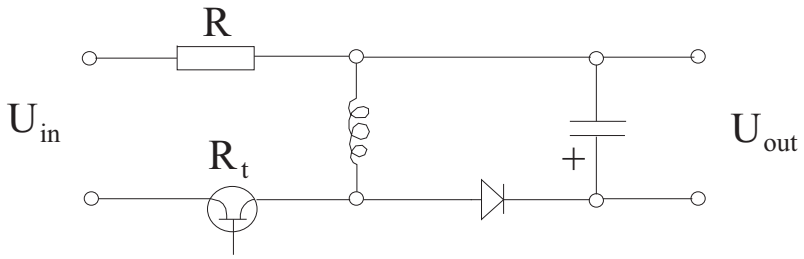
2 One-coil voltage source



Picture 1. A one-coil voltage source.

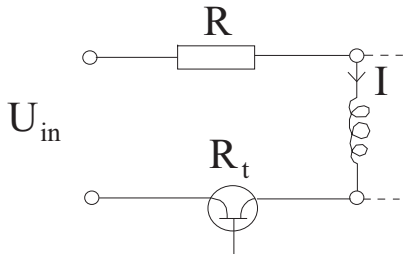
Picture 1 shows the circuit. U_{in} is a DC voltage, the transistor is for example of a MOSFET type. The function of the transistor is to work as a switch. In phase one the transistor is closed, so the current starts to flow through the coil and the transistor. After a short period of time the transistor is opened and the phase two begins. The coil tends to continue the flow of the current and the energy of the coil is transferred through the diode into the capacitor. These two phases are repeated, so the voltage of the capacitor increases and we have a voltage U_{out}.

3 Measurement of small inductances



Picture 2. Equipment for measuring of an inductance.

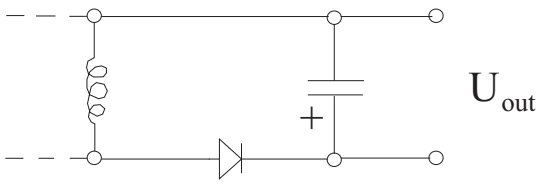
Inductance measurement can be done with the voltage source introduced. One resistor has to be added to the circuit (picture 2).



Picture 3. Phase one.

In phase one the transistor conducts. We have to wait until the current reaches its maximum value. The current is restricted by the resistor R, so the amount of time is not critical. The maximum current and the energy of the inductor are

$$I = \frac{U_{in}}{R + R_T} \qquad W = \frac{1}{2}LI^2$$



Picture 4. Phase two.

In phase two the current can not go through the transistor, and the energy

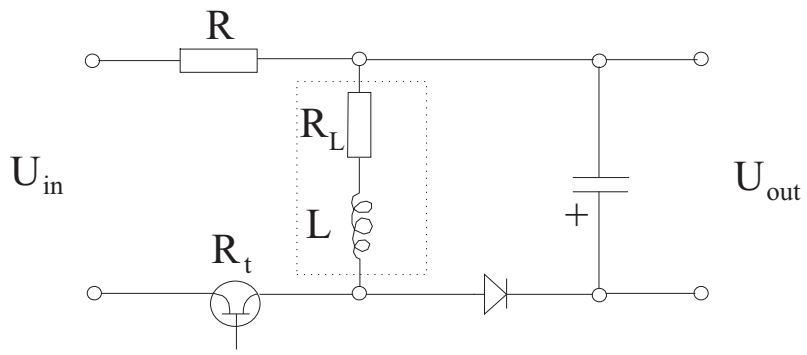
of the inductor goes into the capacitor. The energy of the capacitor at the end of phase two will be the same as the energy of the inductor at the beginning of phase two. From this equation we get the equation for the inductance L:

$$W_1 = \frac{1}{2}LI^2 = \frac{1}{2}CU_1^2 \qquad \Rightarrow \qquad L = \frac{CU_1^2}{I^2}$$

When the inductance is small, the voltage U_1 will be small. Then we can make a loop of f.ex. 100 phase ones and phase twos so that the energy cumulates into the capacitor:

$$W_{100} = 100 \cdot \frac{1}{2}LI^2 = \frac{1}{2}CU_{100}^2 \qquad \Rightarrow \qquad L = \frac{CU_{100}^2}{100I^2}$$

4 The Complete Equivalent Circuit



Picture 5. The complete equivalent circuit.

To make measurements accurate, we have to take into account some additional resistances, look at picture 5. The resistance of the coil R_L affects to both phase one and phase two. Energy lost in phase two is

$$W_{R_L} = \int P(t)\,dt = \int I(t)^2 R_L\,dt$$

$I(t)$ has to be calculated to get W_{R_L}.

5 Caution

The actual circuit has not been built by the author, so it is not tested. Be aware of not to exceed the maximum voltage of the capacitor in the circuit in picture 1. Both the capacitor and the transistor may be damaged.

Moore's Law And Chess Programs

Mikko Malinen

May 28th, 2001

1 Abstract

It is known that the required computer power increases exponentially in chess depending how deep we want to examine the positions. It is also known that the computing power of computers increases according to Moore's law. This paper discusses how Moore's law affects chess programs.

2 Introduction

Moore's law in original form states that **packaging density of transistors double in every 18 months**. This may be understood so that computing power increases as much. On the other hand, it is known that in chess there are on average 37 move possibilities [1]. To examine the game tree one ply (one half-move) deeper we must have 37 times more computing power. Tree search in chess is described in [1] and [2].

3 Calculation

18 months is the same as 1.5 years. The first question is that how much the computing power increases in one year.

$$a^{1.5} = 2$$

$$a = \sqrt[1.5]{2} \approx 1.587$$

In one year the computer power increases 1.587 times. The next question is how many times we must have computing power to search n plys deeper.

$$s = 37^n$$

What we search is y (years) as a function of n. We may write

$$1.587^y = 37^n$$

$$\Rightarrow \log_{1.587} 37^n = y$$

$$\Rightarrow n \log_{1.587} 37 = y \tag{1}$$

$$1.587^x = 37 \quad \Rightarrow \quad x = \frac{\ln 37}{\ln 1.587} = 7.818 \tag{2}$$

From (1) and (2) we get

$$y = 7.818n \approx 8n$$

4 Conclusions

It is evident that adding one ply to the search tree in every 8 years is actually a slow process. It is also evident (if Moore's law is valid), that a complete search tree can be constructed in the future.

5 References

⊏1⊐ Hyvönen-Karanta-Syrjänen: Tekoälyn ensyklopedia, Oy Gaudeamus Ab, Hämeenlinna 1993

⊏2⊐ Malinen,Mikko: A chess-playing computer program, an article published in Internet 2000

An Electrostatic Motor

Mikko Malinen

September 5th, 2001

1 Introduction

The first electrostatic motor was invented in 19th century. This paper presents one construction, which differs from the first of its kind. The motor is based on electrostatic force caused by capacitors. The arms of the motor move inside capacitors.

2 Costruction

Part of the motor is presented in figure 1.

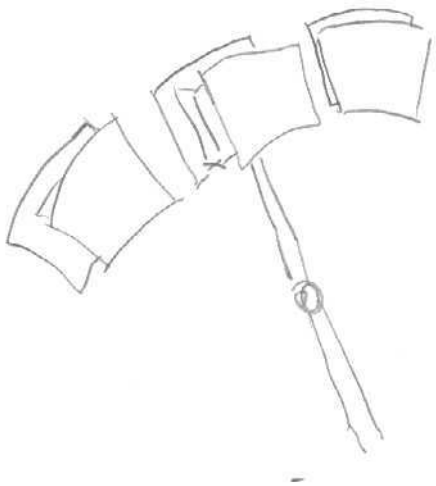


Figure 1.

Capacitance of a plate capacitor is

$$C = \frac{\epsilon A}{d} \tag{1}$$

where ϵ is the permittivity of the material between the plates, A is the area of one plate and d is the distance between the plates.
Capacitance of a capacitor partly filled with air and partly filled with other material can be written

$$C = \frac{\epsilon_0 A_1}{d} + \frac{\epsilon_r \epsilon_0 A_2}{d} = \frac{\epsilon_0}{d}(A_1 + \epsilon_r A_2) \tag{2}$$

Because

$$\frac{\partial A_1}{\partial x} = -\frac{\partial A_2}{\partial x} \tag{3}$$

we get

$$\frac{\partial C}{\partial x} = \frac{\epsilon_0}{d}(\frac{\partial A_1}{\partial x} - \epsilon_r \frac{\partial A_1}{\partial x}) = \frac{\epsilon_0}{d}\frac{\partial A_1}{\partial x}(1 - \epsilon_r) \tag{4}$$

Energy of a capacitor is

$$W = \frac{1}{2} C U^2 \tag{5}$$

and the force is

$$F = -\frac{\partial W}{\partial x} \tag{6}$$

so finally we get (combining 4,5 and 6)

$$F = -\frac{1}{2}\frac{\partial C}{\partial x} U^2 = \frac{1}{2}\frac{\epsilon_0}{d}\frac{\partial A_1}{\partial x}(1 - \epsilon_r) \tag{7}$$

3 Construction of the successive capacitors

In addition to that we have to give the capacitors more energy, we have to transfer the remaining energy of one capasitor to the next at the right moment.

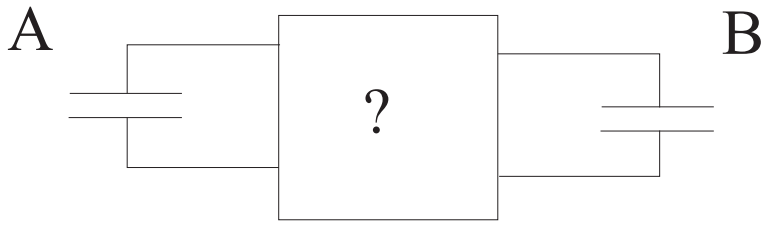


Figure 2. How to transfer the energy from A to B?

First we calculate the current of a parallel CL circuit:

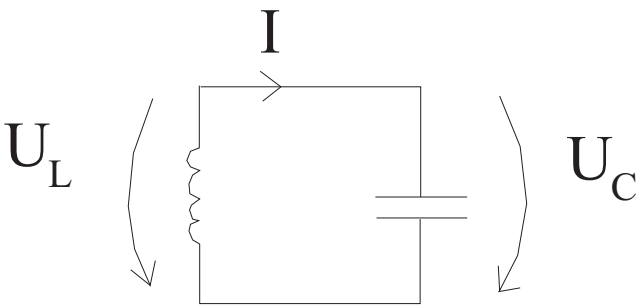


Figure 3. CL circuit. Voltage across L and C.

We know that the voltage across an inductor is

$$U_L = L\frac{dI}{dt} \tag{8}$$

and voltage across a capasitor is

$$U_c = \frac{1}{C}\int Idt \tag{9}$$

Writing these equal we get

$$L\frac{dI}{dt} = \frac{1}{C}\int Idt \tag{10}$$

Differentiating 10 we get

$$\frac{d^2I}{dt^2} - \frac{1}{C}I = 0 \tag{11}$$

This is a second order differential equation

$$LI'' - \frac{1}{C}I = 0 \tag{12}$$

Solution of this equation is of the form

$$I = C_1 \sin\omega + C_2 \cos\omega \tag{13}$$

where

$$w = f(L, C) \tag{14}$$

Actually we know the frequency of the oscillating current. This oscillation can be used to transfer the energy from capasitor C_1 to C_2 (Figure 4.)

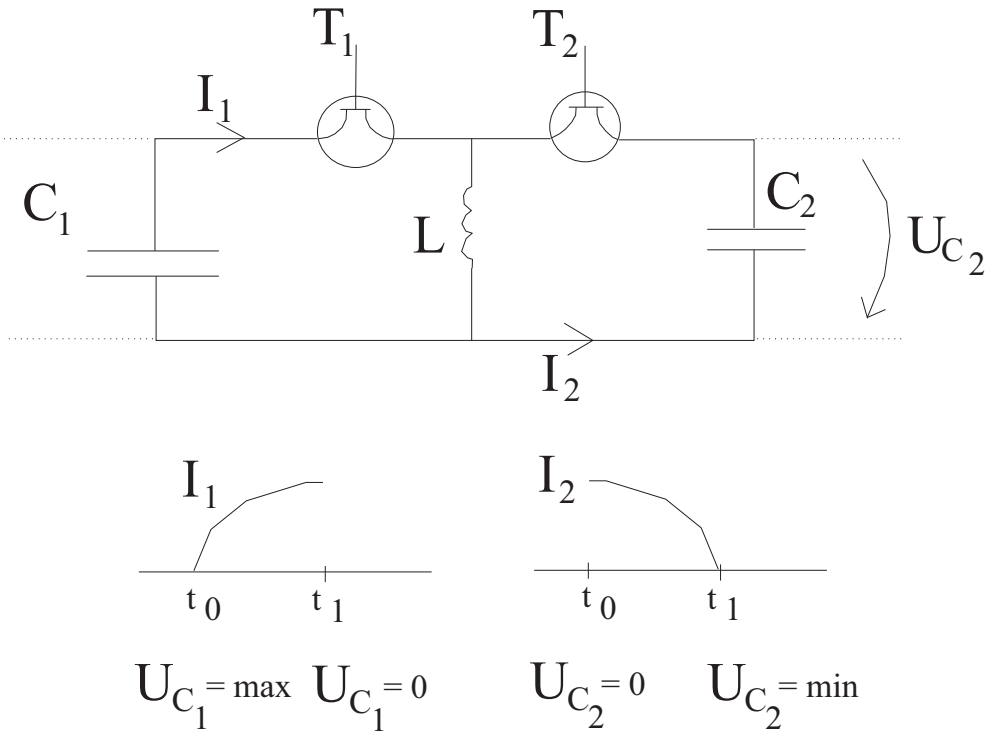


Figure 4. Transferring energy. I_1 and I_2 as a function of time.

At time t_0 all the energy is in C_1. Switch T_1 is closed, switch T_2 is open and the current starts to flow through L. At time t_1 the current is at maximum and voltage $U_{C_1} = 0$. Then T_1 is opened and T_2 is closed and the voltage U_{C_2} starts to decrease from 0. At time t_2 all the energy is transferred to C_2. Nevertheless, there has been resistive losses at T_1 and T_2.

4 Problems

-resistive losses at transistor switches
-high voltage needed $\Rightarrow$ many transistors needed
-maximum power is less than in induction motors

A New Compression Method for Analog Signals?

Mikko Malinen

September 5th, 2001

1 Introduction

Analog transmission lines are often bandlimited. This means that we can transmit only a limited number of channels of a certain bandwidth, for example a limited number of speech channels in a transmission line. This paper suggests a method for compressing a channel of a certain bandwidth to a narrower band channel. Then more channels can be transmitted in the same transmission line. It will be noticed in this paper that the method works mathematically, but it is not realizable in the suggested form. The paper still leaves open the question if a mathematical function and the corresponding realizable circuit can be found.

2 Frequency doubler

The double angle formula

$$\cos^2 t = \frac{1 + \cos 2t}{2} \tag{1}$$

can be written in form

$$\cos 2t = 2\cos^2 t - 1 \tag{2}$$

The circuit realization of this formula is in figure 1.

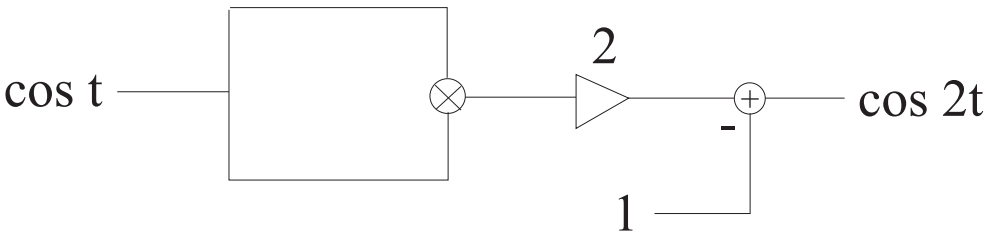


Figure 1. Frequency doubler.

This circuit maps the spectrum of the input signal to a wider spectrum, see figure 2.

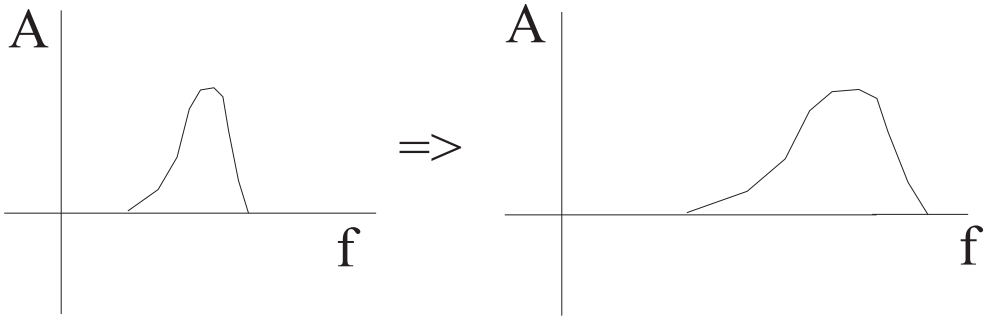


Figure 2. Affect of the frequency doubler to the spectrum.

3 Frequency divider

The above circuit arises a question if there is a circuit opposite to the frequency doubler - a circuit that divides the frequency by two. Mathematically we can solve $\cos t$ from 2:

$$\cos t = \pm\sqrt{\frac{\cos 2t + 1}{2}} \tag{3}$$

Writing the circuit in figure 1 in opposite order we get the circuit in figure 3.

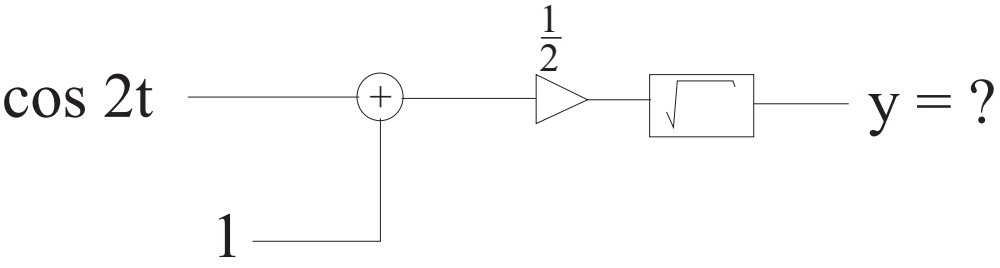


Figure 3. A try for a frequency divider

Because we can not realize the $\pm$ sign in a circuit, it appears that the circuit in figure 3 is not a frequency divider. The actual output of the circuit is the upper curve in figure 4.

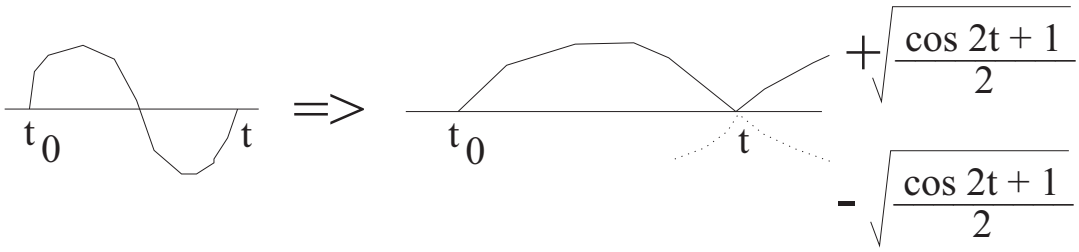


Figure 4. Input and output of the circuit in figure 3.

So there is still open the question if there exists a circuit which divides frequencies. Mathematically written: Does there exist such f that

$$f(\cos 2t) = \cos t \tag{4}$$

and that all operations of f are realizable so that the actual circuit can be built.

Linux c/C++ programming and Latex typesetting

Collected by Mikko Malinen with years of experience
Writing began 13.3.99
Latest modifications 1st July 1999 or later

Parsing of integers and floats

see clib manual, p.213

Timer

There are two possibilities available now: One for counting on a second scale and one for counting more accurate durations (the latter from clib manual p.233).

```
#include<time.h>
tm *aika;
time_t *timep = new time_t   // man ctime
main{
*timep = time(NULL);
aika = gmtime(timep);
cout << aika->tm_min << "   "
     << aika-> tm_sec;     // int
}
```

```
#include<time.h>

clock_t start, end;
main()
{
start = clock();
elapsed = ((double)(end-start))/CLOCKS_PER_SEC;
}
```

Executing a program from a c-program

execve and execvp functions are available, but I could not make them work in Linux yet. The executed program overwrites the calling program.
None of the three test programs works.

Database programming

At the moment there is postgres database for Linux available, not in Linux distribution CDs.
For programmers there is in distributions ready library

```
#include<ndbm.h> // info from book "Beginning Linux programming", database
handling
```

Graphics programming

For X there is complicated libraries, but I prefer svgalib which has f.ex.

```
vga_setmode(int mode);
vga_screenon();
          off();
vga_setcolor(int color);
vga_drawpixel(int x, int y);
```

Parallel (printer) port programming

see /usr/src/linux/include/lp.h
Don't use termios to control printer port. Use ioctl and inb/outb if necessary. There is also ca 20 page document in Internet writted by Zhahai Stewart zstewart@nyx.cs.du.edu.

Pausing a program for seconds

```
#include<unistd.h>
main()
{
sleep(2.5);
}
```

Resquing a corrupted filesystem or hard disk

sometimes there is reading errors which can be recovered by special copying the whole corrupted partition to other by

dd if=/dev/hdb2 of=/dev/hda3 bs=1k conv=noerror,sync

I have recovered with this method both Linux and Dos partitions.

ASCII Character Codes , 001-127

LF = line feed , FF = form feed , CR = carriage return , DEL = rubout

Dec	Chr
000	NUL
001	SOH
002	STX
003	ETX
004	EOT
005	ENQ
006	ACK
007	BEL
008	BS
009	HT
010	LF
011	VT
012	FF
013	CR
014	SO
015	SI
016	DLE
017	DC1
018	DC2
019	DC3
020	DC4
021	NAK

```
022   SYN
023   ETB
024   CAN
025   EM
026   SUB
027   ESC
028   FS
029   GS
030   RS
031   US
032   SP
033   !
034   "
035   #
036   $
037   %
038   &
039   ’
040   (
041   )
042   *
043   +
044   ,
045   -
046   .
047   /
048   0
049   1
050   2
051   3
052   4
053   5
054   6
055   7
056   8
057   9
058   :
059   ;
060   <
061   =
062   >
063   ?
064   @
065   A
066   B
067   C
```

```
068   D
069   E
070   F
071   G
072   H
073   I
074   J
075   K
076   L
077   M
078   N
079   O
080   P
081   Q
082   R
083   S
084   T
085   U
086   V
087   W
088   X
089   Y
090   Z
091   [
092   \
093   ]
094   ^
095   -
096   ^
097   a
098   b
099   c
100   d
101   e
102   f
103   g
104   h
105   i
106   j
107   k
108   l
109   m
110   n
111   o
112   p
113   q
```

```
114   r
115   s
116   t
117   u
118   v
119   w
120   x
121   y
122   z
123   {
124   |
125   }
126   ~
127   DEL
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
```

160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200

abcdefghijklmnopqrstuvwxyz
αβχδεφγηιφκλμνοπθρστυϖωξψζ

§½!"#¤%&/()=?‘*’´+\}][{$£@|<>,.-_:;ÖÄäöÅå¨^~
♣ | !∀#∕%&/()=? ¯*ɘ×+∴ }][{∃≤≅|<>,.−_:;√⊗™)⊕∑♦⊥~

ABCDEFGHIJKLMNOPQRSTUVWXYZ
ABXΔΕΦΓΗΙϑΚΛΜΝΟΠΘΡΣΤΥςΩΞΨΖ

A picture which clears the use of the symbol font characters versus Times New Roman font characters

♦	,	;	J	Y	h	w	½	
4	44	59	74	89	104	119	171	255
♣	-	<	K	Z	i	x	α	
5	45	60	75	90	105	120	224	
§	.	=	L	[	j	y	β	
21	46	61	76	91	106	121	225	
	/	>	M	/	k	z	Γ	
32	47	62	77	92	107	122	226	
!	0	?	N	]	l	{	π	
33	4	63	78	93	108	123	227	
"	1	@	O	^	m	|	Σ	
34	49	64	79	94	109	124	228	
#	2	A	P	_	n	}	σ	
35	50	65	80	95	110	125	229	
$	3	B	Q	´	o	~	µ	
36	51	66	81	96	111	126	230	
%	4	C	R	a	p	ä	τ	
37	52	67	82	97	112	132	231	
&	5	D	S	b	q	å	Φ	
38	53	68	83	98	113	134	232	
´	6	E	T	c	r	Ä	Θ	
39	54	69	84	99	114	142	233	
(	7	F	U	d	s	Å	Ω	
40	55	70	85	100	115	143	234	
)	8	G	V	e	t	ö	δ	
41	56	71	86	101	116	148	235	
*	9	H	W	f	u	Ö	ϕ	
42	57	72	87	102	117	153	237	
+	:	I	X	g	v	£	ε	
43	58	73	88	103	118	156	238	

PC-8 character set. Some characters are omitted because they were not found from either Times New Roman font or Symbol font.

Linux

esim./pseudomalli	toiminto	havaitut puutteet
scanf("int ch; ch = getchar();	lukee yhden merkin ja palauttaa sen arvonaan	vastaava tulostusfunktio putchar(ch); tulostaa va
int a; double b; cin >> a >> b;	lukee a:lle arvoksi kokonaisluvun ja b:lle arvoksi reaaliluvun	
sscanf vrt.scanf	lukee merkkijonosta	

In addition there is more info about reading single characters one at a time in the book gnu libc manual, chapter Low-level terminal I/O

Examining the end-of-file when reading from a file

A few lines which clears the case:

```
istream from(&f1);
if(!from.eof() || to.bad())
error("something ...","");
```

To be done : how to know the date from within a C++ program?

Typesetting a table in Latex

An example which clears the case.

```
\begin{tabular}{|l|l|l|}\hline
\multicolumn{3}{|c|}{Eri bittijonoja vastaavat ulostuloj\"annitteet}
```

```
\\ \hline
input & mitattu $ U_{out} $ & huom. \\ \hline
0000 & -0.001 & bl\"a\"ah \\ \hline
\end{tabular}
```

This gives an out put flom Latex

Eri bittijonoja vastaavat ulostulojännitteet		
input	mitattu U_{out}	huom.
0000	-0.001	blääh

Referances

Stroustrup

GNU libc manual

Korpela-Larmela: C-kieli

Automaatiosovellus Linuxiin

Mikko Malinen

Kesäkuun 20.,2000

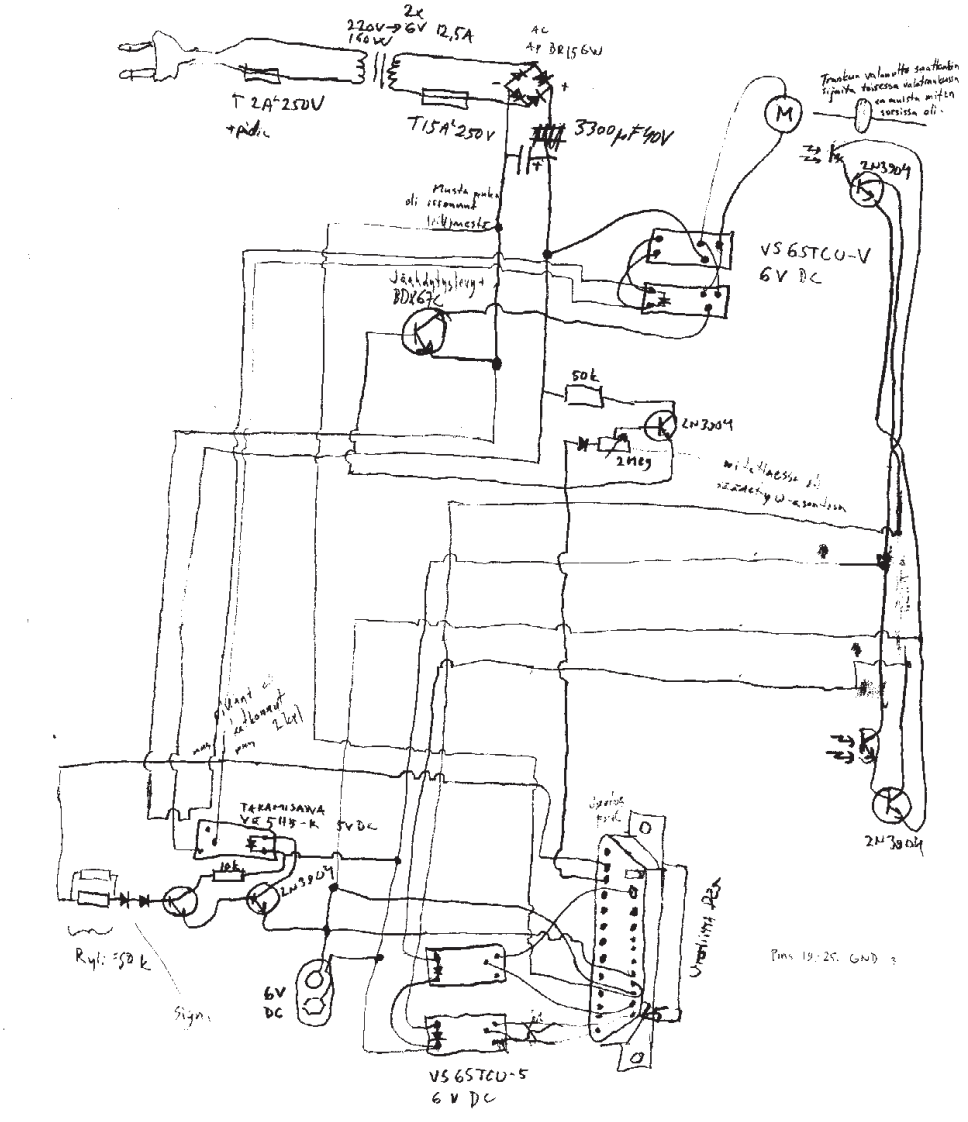
1 Johdanto

Dokumentissa esitetään, miten elektroninen laite ja tietokoneohjelma voidaan liittää toisiinsa rinnakkaisportin kautta. Dokumentissa esitetään, miten nidontakoneen elektroniikkaosaa ohjataan tietokoneohjelman avulla. Laite sisältää sekä tuloja että lähtöjä. Työssä on saatu aikaan valmiiksi mitoitetut kytkennät releen ohjaamiseksi, tasavirtamoottorin ohjaamiseksi päälle ja pois, moottorin pyörimissuunnan ohjaamiseksi sekä valo-ohjatun sisääntulon toteuttamiseksi. Lisäksi on esitetty malli laitteen loogisen toiminnan ohjaamiseksi tietokoneohjelmasta käsin.

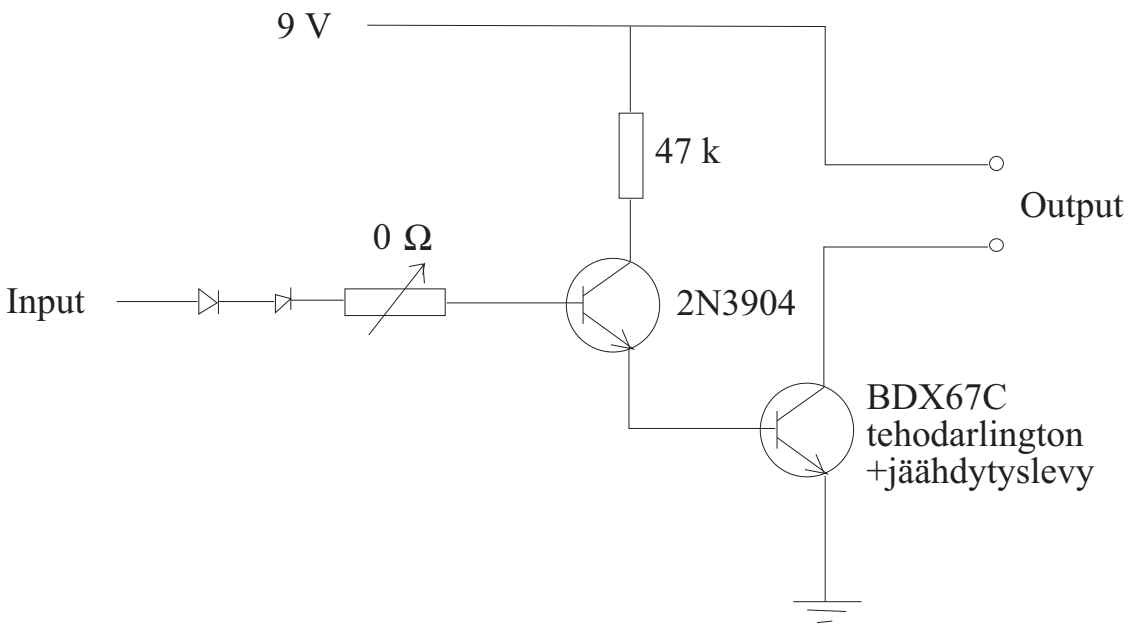
2 Laitteen rakenne

Laite käyttää rinnakkaisportin kahta ulostuloa ja kahta sisäänmenoa. Kaikkiaan ulostuloja on neljä kappaletta, jotka on numeroitu 13-16 ja joita vastaavat rinnakkaisporttiliittimen nastat 1,14,16 ja 17. Sisäänmenoja on neljä, numeroitu 9-12. Niitä vastaavat liittimen nastat 13,12,10 ja 11. Kun ulostulo asetetaan 1:ksi, saa liittimen nasta jännitteen 5V. Kun sisäänmenonasta kytketään liittimen maahan, saa sisäänmeno arvon 0. Muulloin sen arvo on 1.
Kuva 2 esittää moottorin käyntiä kontrolloivan kytkentäkaavion osan. Input tulee rinnakkaisporttiliittimen nastasta 13. Toimivassa kytkennässä trimmeripotentiometrin arvo oli 0Ω, mutta se tulisi säätää esim. 20kΩ:ksi toisaalta rinnakkaisliitännän ylikuormituksen välttämiseksi sekä toisaalta transistoreiden vaurioitumisen estämiseksi. Output menee tasajännitemoottorille suunnanvaihtolaitteen läpi. On huomattava, että kaavion maa on sekä käyttöjännitteen maa että rinnakkaisliittimen maa.
Kuva 3 esittää releohjaimen kytkentäkaavion. Sisääntulo tulee rinnakkaisporttiliittimen ulostulonastasta numero 1. Sisääntulon diodit voivat olla tavallisia piensignaalidiodeja. Rele kytkee jännitteen moottorin suunnanvaihtolaitteeseen, ks. kuva 4. Kuva 4 esittää varsinaisen suunnanvaihtolaitteen kytkentäkaavion. Suunnan ohjausjännite tulee kuvan 3 laitteen ohjaamana. Moottorin

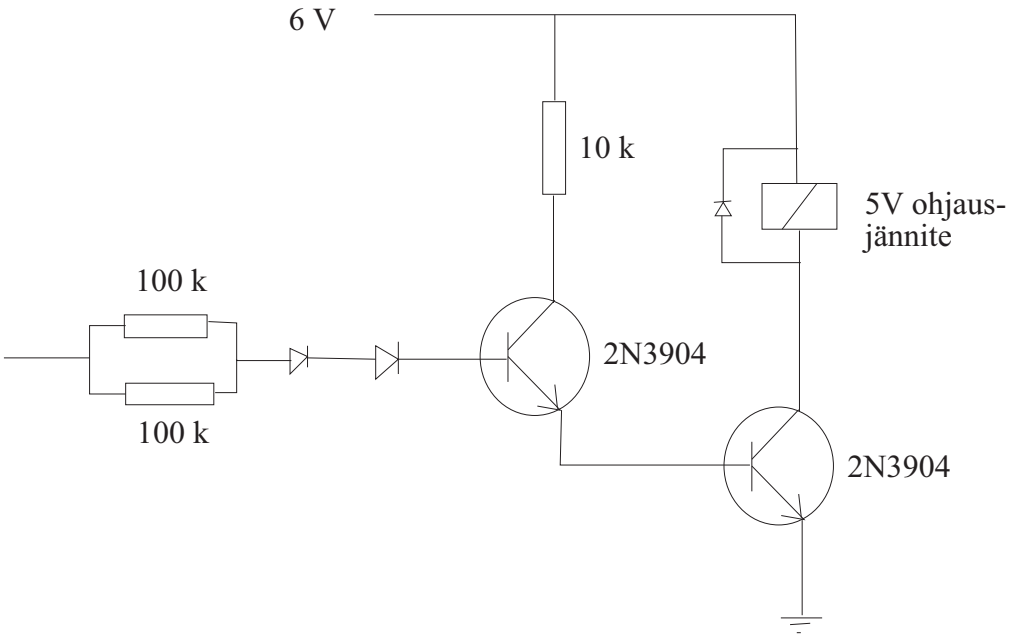
käyttöjännite tulee kuvan 2 laitteelta.
Kuvassa 5 on valo-ohjattujen sisäänmenojen kytkentäkaavio. Valo-ohjaus tapahtuu valotransitoreiden avulla. Kun valotransistori 1 on valossa, kytkeytyy rinnakkaisportin sisäänmenonasta 13 rinnakkaisliittimen maahan eli nastaan 25. (Nastat 19.-25. ovat GND.) Vastaavasti valotransistori 2 kytkee rinnakkaisliittimen nastan 12 maahan.



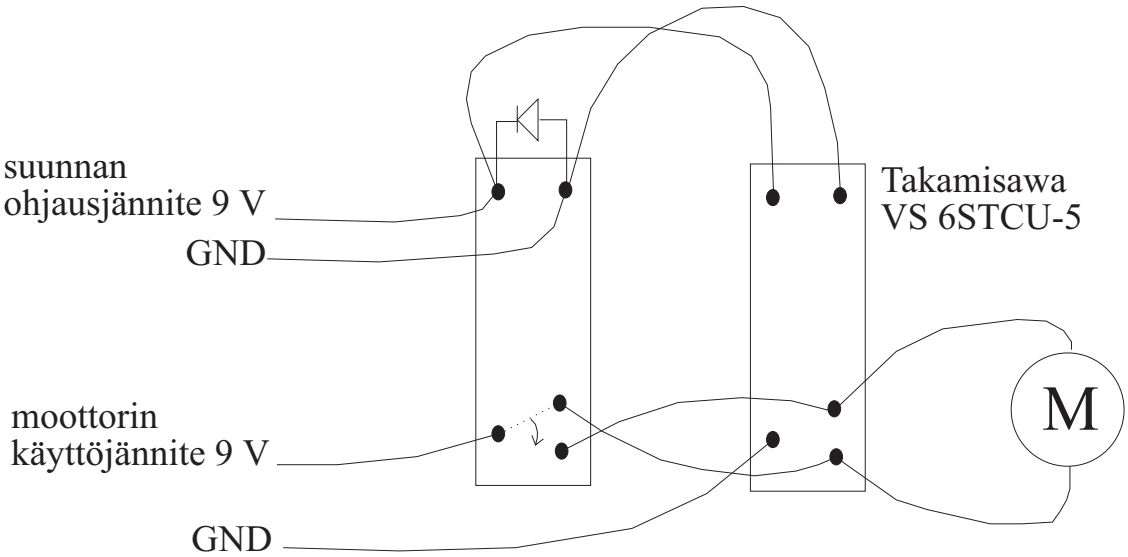
Kuva 1. Laitteen kytkentäkaavio. Osia kaaviosta on kuvattu erillisissä kuvissa. Liittimen kytkennät poikkeavat hieman esitetyistä, ks. teksti.



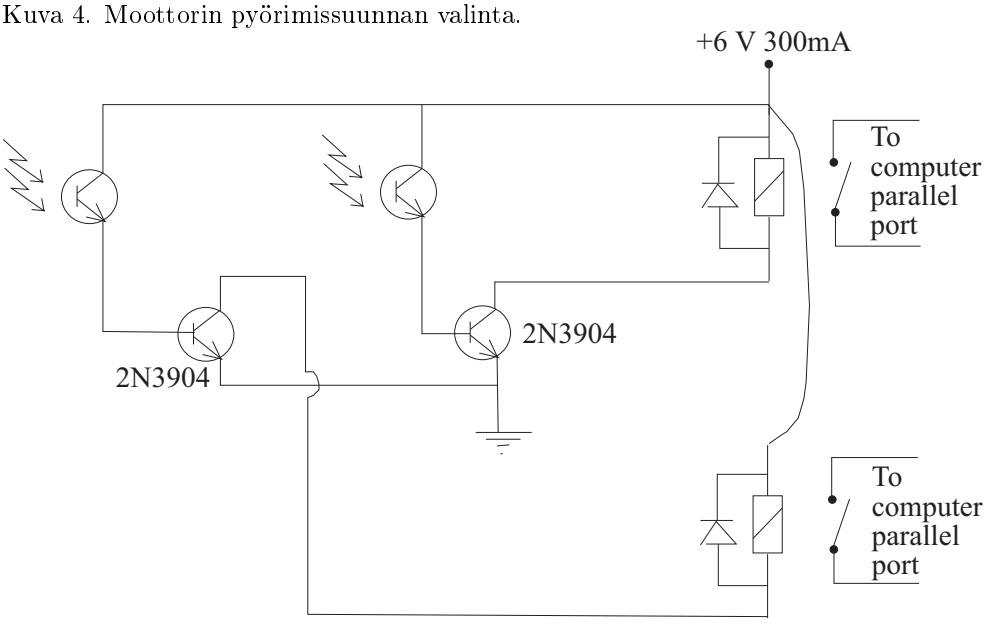
Kuva 2. Moottorin ohjaus päälle/pois.



Kuva 3. Moottorin suunnan vaihto. Ks.myös kuva 4.



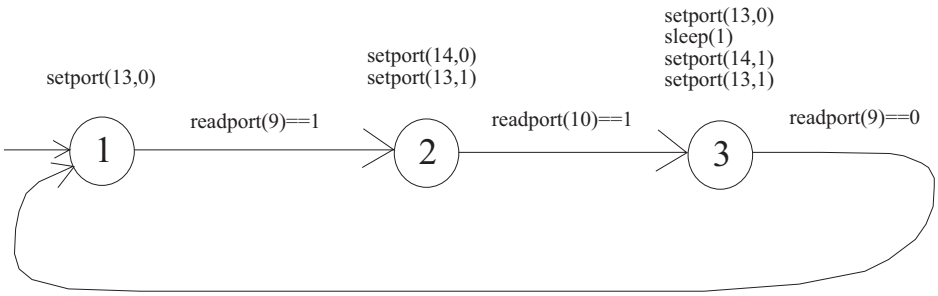
Kuva 4. Moottorin pyörimissuunnan valinta.



Kuva 5. 2 kpl valo-ohjattuja sisäänmenoja.

3 Tietokoneohjelma

Tietokoneohjelma toteuttaa laitteen loogisen osan. Ohjelman periaatekaavio on kuvassa 6. Kaavion tilojen selitykset ovat seuraavat:
1 Paperi ohittanut poistumisaukon/Odotetaan uutta paperia
2 Paperi valmiina
3 Paperi kulkenut niittauskohtaan
On huomattava, että tietokoneohjelman linjan numero ei ole välttämättä sama kuin rinnakkaisporttiliittimen nastanumero. Tietokoneohjelma sisältää varsinaisen logiikkaosan lisäksi toteutukset funktioille setportson(),setportsoff(),readport(int noOfPort),setPort(int noOfPort,bool value),printports() ja oscillate(int hz,double duration). On suositeltavaa, että laitteen virta kytketään päälle vasta kun ohjelma on käynnistetty. Tällöin moottorin virta on katkaistu. Tietokoneohjelma on käännettävä käyttäen optimointioptiota -O makrojen takia.



Kuva 6. Tietokoneohjelman logiikkaosan periaatekaavio.

```
/* The following is some example code. Note that it must be compiled
g++ -O file.C
because some of the hardware control layers are actually macros, which won't
work unless optimisation is performed at compile time. A nasty trap!
-----------------------------------------------------------------------------
*/

/* Additions made 1999 to author's original code:

  At the moment of writing this the functions for setting values easily
  are implemented. Read comments later in the file.

*/


/*
 *  AUTHOR: Sven Goldt (goldt@math.tu-berlin.de)
```

```
 *  Modified by Mikko Malinen
 *
 *  This program is free software; you can redistribute it and/or
 *  modify it under the terms of the GNU General Public License
 *  as published by the Free Software Foundation; either version 2
 *  of the License, or (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
*/
/*
 *
 * Changes in 1.2: amn@bigbang.nfra.nl removed "oldlp.h", converted
                   to "new" LPGETIRQ ioctl(),
 * Changes in 1.1: Fixed bit meaning for low active lines !
 *
*/

#include<stdio.h>
#include<fcntl.h>
#include<linux/lp.h>
#include<iostream.h>
#include<time.h>
#include<math.h>

/* DAMNED ! - Someone crippled that file !!! */
/* amn: Yes, yes, data encapsulation, you know... :-) */
/*      That table is currently hidden inside the kernel. */
/*      For our direct parallel I/O port references, here's a copy */
/*      (from '/usr/src/linux/drivers/char/lp.c'): */
struct lp_struct lp_table[] = {
{ 0x3bc, 0, 0, LP_INIT_CHAR, LP_INIT_TIME, LP_INIT_WAIT, NULL, NULL, },
{ 0x378, 0, 0, LP_INIT_CHAR, LP_INIT_TIME, LP_INIT_WAIT, NULL, NULL, },
{ 0x278, 0, 0, LP_INIT_CHAR, LP_INIT_TIME, LP_INIT_WAIT, NULL, NULL, },
};
#define LP_NO 3

/* The direct I/O port reference macros reside here. */
/* They work _only_ if the source files using them are compiled */
/* with '-O' optimization flag on; they rely on 'inline' code generation. */
#include<asm/io.h>

int i;
```

```
clock_t start,end,start2;
int hz;

void
setportson()
{
//outb( (inb(LP_B(i)+2)&(0xE0))|(0x04),LP_B(i)+2 );
outb(0xE4,LP_B(i)+2);

}

void
setportsoff()
{
//outb( (inb(LP_B(i)+2)|(0x00))&(0xEB),LP_B(i)+2 );
outb(0xEB,LP_B(i)+2);
}

bool
readport(int noOfPort)
{
// noOfPort 1...16

if (noOfPort<9)
  {
 return (inb(LP_B(i))& (char)pow(2,noOfPort-1));
  }
if (noOfPort<12)
  {
 return (inb(LP_B(i)+1)& (char)pow(2,noOfPort-5));
  }
if (noOfPort==12)
 {
 return !(inb(LP_B(i)+1)& (char)(128));
 }
if (noOfPort<15)
  {
 return !(inb(LP_B(i)+2)& (char)pow(2,noOfPort-13));
  }
if (noOfPort==15)
  {
 return (inb(LP_B(i)+2)& (char)(4));
  }
else    // noOfPort = 16
  {
 return !(inb(LP_B(i)+2)& (char)(8));
```

```
  };
}

void
setport(int noOfPort,bool value)
{
// noOfPort 13-16, value 0 or 1
if (noOfPort != 15)
  { if (value == 1) { // next line: 239 = 224+15
           outb(inb(LP_B(i)+2)& (char)(239-pow(2,noOfPort-13)), LP_B(i)+2);
                    }
  else {
           outb(inb(LP_B(i)+2)| (char)(pow(2,noOfPort-13)), LP_B(i)+2);
                    };
 }

else
  { if (value == 1) {   // next line: 4 = pow(2,2)
    outb(inb(LP_B(i)+2)| (char)(4), LP_B(i)+2);

                    }
  else {  // next line: 4 = pow(2,2),  235 = 224+15-4
      outb(inb(LP_B(i)+2)& (char)(235), LP_B(i)+2);
                    };
 };
}

void
printports()
{
  cout << (int)(inb(LP_B(i)));
  cout << " ";
  cout << (int)(inb(LP_B(i)+1)) << " ";
  cout << (int)(inb(LP_B(i)+2)) << " ";

  cout << (bool)(inb(LP_B(i))&0x01);
  cout << (bool)(inb(LP_B(i))&0x02);
  cout << (bool)(inb(LP_B(i))&0x04);
  cout << (bool)(inb(LP_B(i))&0x08);
  cout << (bool)(inb(LP_B(i))&0x10);
  cout << (bool)(inb(LP_B(i))&0x20);
  cout << (bool)(inb(LP_B(i))&0x40);
  cout << (bool)(inb(LP_B(i))&0x80);

  cout << (bool)( inb(LP_B(i)+1)&0x10 );
  cout << (bool)( inb(LP_B(i)+1)&0x20 );
```

```
  cout << (bool)( inb(LP_B(i)+1)&0x40 );
  cout << !(bool)( inb(LP_B(i)+1)&0x80 );     // negative      !   LISATTAVA

  cout << " ";

  //  cout << (int)(!(bool)( inb(LP_B(i)+2)&&(0x01) ));    // negative
  //cout << (int)(!(bool)( inb(LP_B(i)+2)&&(0x02) ));    // negative
  //cout << (int)(( inb(LP_B(i)+2)&&(0x04) ));
  //cout << (int)(!(bool)( inb(LP_B(i)+2)&&(0x08) ));    // negative

  cout << !(bool)(inb(LP_B(i)+2)&(0x01));    // negative
  cout << !(bool)(inb(LP_B(i)+2)&(0x02));    // negative
  cout << (bool)(inb(LP_B(i)+2)&(0x04));
  cout << !(bool)(inb(LP_B(i)+2)&(0x08));    // negative

  cout << endl;
}

void
oscillate(int hz,double duration)
// hz: dont use too big freq!     duration: in seconds
{
//hz=3;
start = clock();
start2= clock();
end= clock();
do
{
  //cout << ((double)(end-start))/CLOCKS_PER_SEC << endl;
  if ( ((double)(end-start2))/CLOCKS_PER_SEC > 0.5/hz )
    {
      //cout << ((double)(end-start))/CLOCKS_PER_SEC << "  " << readport(13) << endl;
      start2=clock();
      if (readport(13) == 1) {setport(13,0);} else {setport(13,1);};
    }
  end = clock();
} while (  ((double)(end-start))/CLOCKS_PER_SEC < duration );
}



main(int argc, char *argv[])
{
  int fd,irq, retCode;
  // int i  moved to global scope
  unsigned char status=0;
```

```
  char printer[10];
  char dummy[7];
  void setport(int noOfPort, bool value);
  bool readport(int noOfPort);
  void setportson();
  void setportsoff();
  void printports();

fprintf(stderr,"mainprogissa ollaan\n");
  for (i=0;i<LP_NO;i++)  /* polling is being used.) */
    {
    sprintf(printer,"/dev/lp%d",i);
    printf("\nchecking %s: ",printer);
    fflush(stdout);


    /*    fd = open(printer, O_WRONLY);   */
    fd = open(printer, O_RDWR);
    if (fd < 0) {
      perror(NULL);
      continue;
    }
    retCode = ioctl(fd, LPGETIRQ, &irq);
    close(fd);
    if (retCode < 0) {
      perror(NULL);
      continue;
    }
    if (irq>0) {
      fprintf(stderr, "irq = %d",irq);
    } else {
      fprintf(stderr, "polling driver used");
    }

    fprintf(stderr, ", I/O base address is 0x%x.\n",LP_B(i));

    /* TAMA ON TARKEA, MUUTEN EI PORTTEIHIN IO:TA (CORE)  : */

    /* Getting the permission to directly access a given range */
    /* of I/O port addresses (below 0x3ff, above that see 'man iopl'). */
    if (ioperm(
      /*from address=>*/         LP_B(i)+0,
      /*number of addresses=>*/  3,
      /*turn on=>*/              1  /* 0 to turn off */
    ) < 0) {
      fprintf(stderr, "access to port 0x%x denied\n",LP_B(i)+1);
```

```
      continue;
    }

    /*    cout << "seuraavaksi luetaan status" << endl;  */

    /* Read the status byte directly from 8255. */
    status=inb(LP_B(i)+1);

/*  Next comes some information:

    Address      bits   direction negatives        comment

    LP_B(i)      D0-D7  input                       tri-stated by setting C5=1
    LP_B(i)+1    S4-S7  input     S7 negative
    LP_B(i)+2    C0-C3  output    C0,C1,C3 negative

    Totally 12 input lines, numbered 1-12 and 4 output lines, numbered 13-16.
    Notice that when switch is open, is input 1. Switch closed input 0!

    Parallel plug:

    Inputs  Pin                 Outputs Pin           Else     Pin

    1       2                   13      1             Gnd      25
    2       3                   14      14
    3       4                   15      16
    4       5                   16      17
    5       6
    6       7
    7       8
    8       9   ^tri-stated
    9       13
    10      12
    11      10
    12      11

*/


    /* C7 C6 C5 C4 C3 C2 C1 C0           */
    /* 1   1  1  0             = E       */
    /* 0   0  1  0             = 2       */
/* C4 = 0 to ensure that interrupt is disabled */
outb(inb(LP_B(i)+2)&&(0xEF), LP_B(i)+2);
/* C5 = 1 to tri-state D0-D7 (used as input) */
outb(inb(LP_B(i)+2)||(0x20), LP_B(i)+2);
```

```
//outb(0xFF,LP_B(i));
//printports();
//outb(0x00,LP_B(i));
//printports();

//outb((inb(LP_B(i)+2)&&(0xF0))||(0x04),LP_B(i)+2);
//outb(0xEF,LP_B(i)+2);
//printports();
//outb(0xE0,LP_B(i)+2);
//printports();

/*
printports();
setportson();
cout << "portson: ";
 printports();
setportsoff();
cout << "portsoff: ";
 printports();
setport(14,1);
cout << "port14on: ";
 printports();
setport(14,0);
cout << "port14off ";
 printports();
setport(15,1);
printports();
setport(15,0);
printports();
*/

// for(;;) {
//printports();
//    }

//setport(13,1);

//oscillate(3,1.5);
//oscillate(10,1.5);
//oscillate(20,1.5);
//for(;;) {
//oscillate(30,1.5);
```

```
//oscillate(40,1.5);
//}

for(;;) {
setport(13,0);
while (readport(9)==0) {};
setport(14,0);
setport(13,1);
while (readport(10)==0) {};
setport(13,0);
sleep(1);
setport(14,1);
setport(13,1);
while (readport(9)==1) {};
}

/*
    / * One could equally well send out desired value to the
       parallel port outgoing data lines: * /

      for(;;) {
      / * do {} while (getch() != 0); * /
    outb(0xFF, LP_B(i));
    cout << "FF ulostulossa, sisaantuloissa " << (int)inb(LP_B(i)+1) << " ja " << (int)i
    cin >> dummy;
    / * do {} while (getch() != 0); * /
    outb(0x00, LP_B(i));
    cout << "00 ulostulossa" << endl;
    cin >> dummy;
    }
*/

#if 0
    /* Verbose output. */
    fprintf(stderr, " printer is%sand %s\n",
      (status & LP_PSELECD) ? " online " : " offline ",
      (status & LP_PBUSY) ? "idle" : "busy"
    );
    fprintf(stderr, " printer has%sacknowledged data sent\n",
      (status & LP_PACK) ? " not " : " "
    );
    fprintf(stderr, " printer is %s paper\n",(status & LP_POUTPA) ? "out of" : "loaded w
    fprintf(stderr, " printer is signalling%serror\n",
      (status & LP_PERRORP) ? " no " : " an "
    );
#else
```

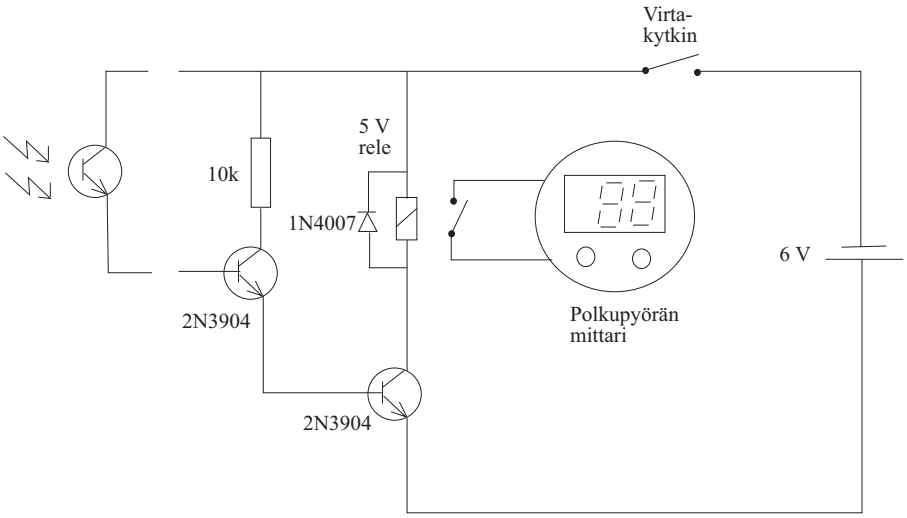
```
    /* Bitwise output. */
    #if 0
    Just repeating stuff from , for docum. purposes:
    LP_PBUSY 0x80  /* inverted input, active high */
    LP_PACK 0x40  /* unchanged input, active low */
    LP_POUTPA 0x20  /* unchanged input, active high */
    LP_PSELECD 0x10  /* unchanged input, active high */
    LP_PERRORP 0x08  /* unchanged input, active low */
    #endif

    fprintf(stderr, " busy=%d, /ack=%d, paper_out=%d, selected=%d, /error=%d\n",
      (status & LP_PBUSY) ? 1 : 0,
      (status & LP_PACK) ? 1 : 0,
      (status & LP_POUTPA) ? 1 : 0,
      (status & LP_PSELECD) ? 1 : 0,
      (status & LP_PERRORP) ? 1 : 0
    );
#endif
  }  /* for each printer */
}  /* main */
```

Kytkentäkaavioita

Mikko Malinen

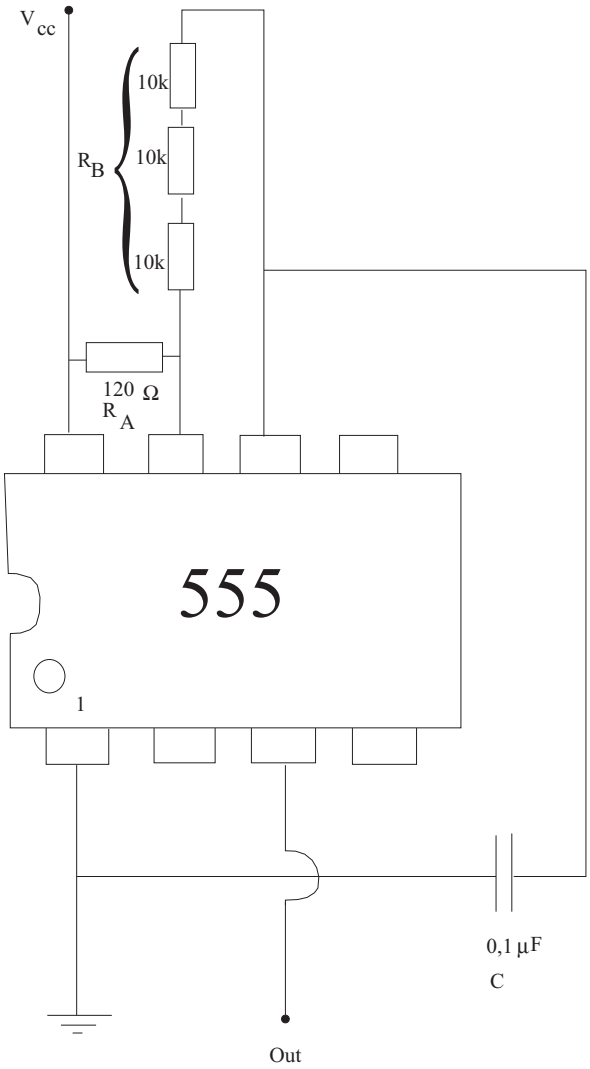
Kesäkuun 30., 2000

1 Askelmittari



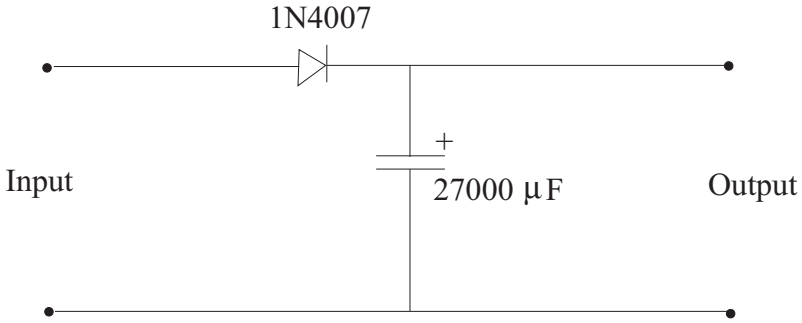
Valotransistori kiinnitetään kengänpohjaan. Polkupyörän mittarin oltava tyyppiä, joka toimii puolaan kiinnitetyn magneetin avulla. Mittarin johto katkaistaan ja se kiinnitetään tähän laitteeseen. Mittarin lukema on skaalattava kunkin kävelytyylin mukaan.

2 Kanttiaalto-oskillaattori

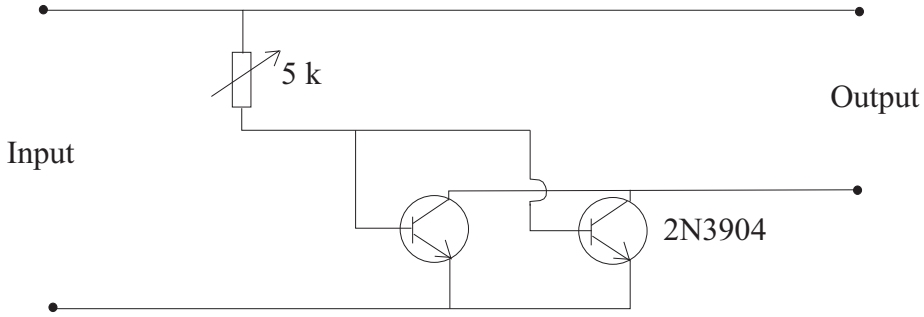


Teoreettinen arvo periodille: $T = 0.69C(R_A + 2R_B)$. R_A valittava paljon pienemmäksi kuin R_B, jotta pulssi on ylhäällä kutakuinkin saman ajan kuin alhaalla. Kuvassa olevan laitteen ulostulon teoreettinen kellotaajuus 241 Hz, mitattu arvo 150 Hz.

3 Jännitteen huippuarvon pitopiiri



4 Latausvirran rajoituslaite



0...10 V pienitehoisille virtalähteille sopiva. Kännykkäakun lataukseen sopiva. Estää virtalähteen sulakkeen palamisen. Varo transistorien ylikuumenemista, säädä trimmeripotentiometrilla. Transistoreita voi kytkeä useampiakin rinnan tehonkeston parantamiseksi. Latausvirta

$$I = \frac{U_{in} - 0.7}{R} \cdot h_{fe} \approx \frac{U_{in} - 0.7}{R} \cdot 120$$

A chess-playing computer program

Mikko Malinen

July 23, 2000

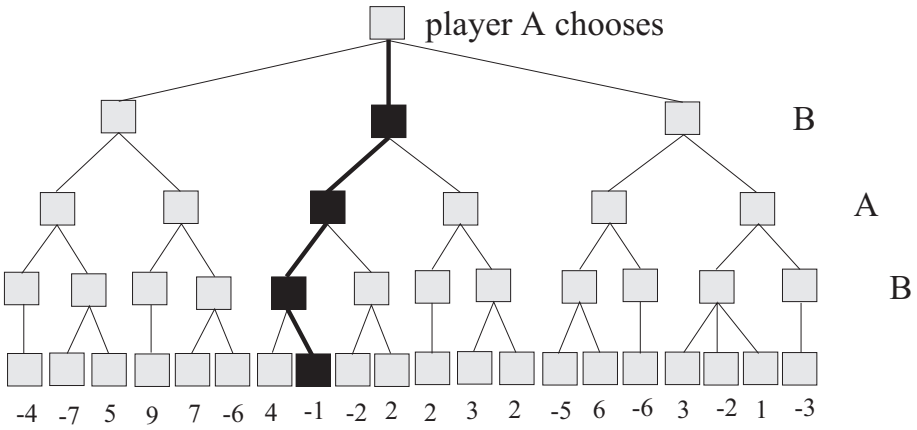
1 Comparison of chess programs

Strength of a chess player is sometimes presented as an elo number. There is a formula for determining player's elo rating:

$$(w-l)/g \cdot 400 + (average \quad of \quad opponents \quad rating) = player's \quad rating$$

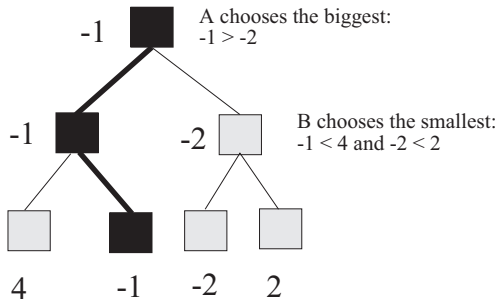
where w = wins, l = losses and g = number of games. If the elo difference between two computer programs is small - let's say that it is 40 - then the better program wins only six out of ten games. This must be taken in consideration when comparing computer programs for example in tournaments. These small differences can not usually be seen there and the winner may be determined by good luck.

2 How chess programs work?



Picture 1. A game tree.

To investigate a specific state of a game chess programs use an evaluation function. Usually it counts the material balance, which chessmen are on the board. It may contain also other factors such as how many squares the chessmen can attack. Chess programs search states which can be obtained after some moves. Look at picture 1. The evaluation function is calculated four half moves (plys) forward. The minimax-algorithm is based on a principle that both players choose the best alternative for them when the tree is traversed from bottom to top. When it is B's turn the smallest is chosen and when it is A's turn the biggest is chosen. The result is a path at picture 2, where A chooses the first move.



Picture 2. Choosing the move in minimax-algorithm.

3 The implemented chessprogram

3.1 The operation of the program

The program is written in C++ and compiled with g++ to allow maximum portability. Program listing mchess.flex has the function generateMove(). This function explains the operation of the program. The C++ class MoveGenerator stores the state of the node at instance C which is of class Board. MoveGenerator has also member functions to find all the move possibilities from a state. When the program has found all the states reachable after a specific number of levels - that is - plys - it finds the optimal move by minimax-algorithm.

3.2 Test results

The game program was let to play against itself. The first game is run on a PC with 64 megabytes of memory. With this equipment four levels - four plys - can be investigated.

Event ["COMPUTER CHESS 1996"]
Date ["1996.11.06"]
Round ["final"]
White ["TITAANIT VI"]
Black ["TITAANIT IV"]
Result ["0-1"]

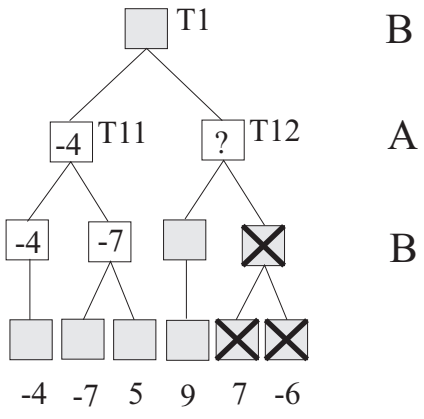
1. h2h3 a7a6 2. h3h4 a6a5 3. h1h3 a8a6 4. h3a3 a6h6 5. g2g3 b7b5 6. e2e3 b8a6 7. f1b5 c7c6 8. b5a6 c8a6 9. g1h3 a5a4 10. b1c3 a6c8 11. f2f4 c6c5 12. h3f2 c5c4 13. a3a4 e7e6 14. a4a8 c8b7 15. a8d8 e8d8 16. g3g4 b7f3 17. d1f3 d7d6 18. f4f5 e6f5 19. f3a8 f5g4

The next example is made with a powerful unix workstation with larger memory capabilities. With this equipment five plys can be investigated. Running the program resulted in segmentation fault at the 14th move from a reason unknown at the moment. However, the quality improvement in moves can be seen.

1. e2e3 e7e6 2. d1g4 d8f6 3. g4f4 f6f4 4. e3f4 f8d6 5. d2d4 b8e6 6. g1f3 d6b4 7. e1d1 a7a6 8. f1d3 h7h5 9. f3e5 c6e5 10. f4e5 h5h4 11. c1d2 b4d2 12. b1d2 h8h5 13. d1e2 d7d6

3.3 How the performance could be improved?

3.3.1 Alpha-beta pruning



Picture 3. The use of alpha-beta -pruning.

In alpha-beta pruning the game tree must be traversed from left to right, not only from bottom to top as in this program. Look at the part of the game tree at picture 3. Minimax-method gives the value -4 for the left part (B has chosen smaller values which are best for him and A has chosen the bigger values). At the point marked by a question mark (T12) A could choose the left part. Then the result would be 9. So the value of T12 is in all cases at least 9. Because A knows that B can reach the value -4 by choosing the left-side alternative (T11), the other states (after T12) need not be investigated. In alpha-beta pruning two variables α and β are used. Alpha is the best value which the opponent (B) may reach. Beta is the worst value, where the program A may go into. With alpha-beta pruning as much as 99,5% may be saved. This means that only every 200th state needs to be investigated.

3.3.2 Decreasing memory usage

At the moment one node reserves memory by three different classes. The memory requirement is as follows:

Class	Memory, bytes
Board	260
OwnMoveGenerator	28
Possibilities	720

The Board -class stores the state of a board. In chess, there are 12 different types of chessmen on the board. 13, if empty is included. This means that one type may be expressed with four bits. There are 64 squares on the board. Places of the chessmen may be expressed with $64 \cdot 4$ bits = 256 bits = 32 bytes. The other classes may be optimized by the similar way.

3.3.3 Finding the optimal parameters

As mentioned earlier, the evaluation function usually takes into account the material balance. It may consider also other factors such as how many move possibilities a player has. The evaluation function may be more complicated than just the material value. One example is

$$value = coefficient_1 \cdot factor_1 + coefficient_2 \cdot factor_2 + ...$$

The coefficients for factors may be determined by playing games against game programs with different evaluation functions.

3.3.4 Estimating the opponents parameters

When there is available a game program that is better than the program under development the coefficients of the evaluation function of the better program may be determined. We do not know the actual evaluation function, but in the best case the function is determined at least approximatively. The coefficients of an evaluation function are to be determined. We know the values of the factors.

The values of the function we do not know but we may give them estimates because we know which moves are selected and which are not. With these information we can apply a linear regression model to determine the coefficients. Of course the predictivity of the estimated evaluation function must be checked by comparing the moves with moves made by a program with the estimated evaluation function.

3.3.5 Increasing computing power

The game program is programmed in PC environment. The first runs were made in PC. The portable code was then run on a high performance UNIX workstation. The next step would be to divide the computing to several workstations. It must be kept in mind that although the huge speed increase we need to make the computing power 37 times bigger to add one ply more. In chess there is on average 37 move possibilities. To add n plys we need 37^n times more computing.

3.4 Program listing

Here are some files from the 1996 version of the program.

3.4.1 Board.h

```
//extern int manAsAmaterial[15] = {0,1,3,4,5,100,12,0,0,-1,-3,-4,-5,-100,-12};
//extern char manAsALetter[16] = {"-STRLKQ  strlkq"};

class Board {


 public:

Board();
~Board();

//unsigned whosTurn : 1; // bit field of length 1: 0=white's turn, 1=black's

 unsigned state[8][8]   // every square is filled with either blanc or
                       // a man of some type.    [x][y]    [0-7][0-7]

= {  { 2,1,0,0,0,0,9,10},
     { 3,1,0,0,0,0,9,11},
     { 4,1,0,0,0,0,9,12},
     { 6,1,0,0,0,0,9,14},
     { 5,1,0,0,0,0,9,13},
     { 4,1,0,0,0,0,9,12},
     { 3,1,0,0,0,0,9,11},
```

```
    { 2,1,0,0,0,0,9,10}
  };


/*
char state[8][4]

     = {  { 33,0,0,154},
  { 49,0,0,155},
  { 65,0,0,156},
  { 97,0,0,158},
  { 81,0,0,157},
  { 65,0,0,156},
  { 49,0,0,155},
  { 33,0,0,154}
     };

*/

void print();
void move(char *m);                                // format of move "e2e4"
void move(char *x1,char *y1,char *x2,char *y2);  // format of move 4,0,4,2
//unsigned whiteInChess :1;   // 0=no,  1=yes
//unsigned blackInChess :1;
int value = 0;


//int manAsAmaterial[15] = {0,1,3,4,5,100,12,0,0,-1,-3,-4,-5,-100,-12};

//protected:

//char manAsALetter[16] = { "-STRLKQ  strlkq" };


};
```

3.4.2 Board.C

3.4.3 MoveGenerator.h

```
class MoveGenerator {

 public:

MoveGenerator(Board *b = 0, int from = 0);
```

```
~MoveGenerator();

Board *C;

/* parameters for this specific move generator */
/* material values of men */
/* ... */

char *generate();


int manAsAmaterial[15] = {0,1,3,4,5,8,100,0,0,-1,-3,-4,-5,-8,-100};
int materialDifference();

/* possibilities to move from x1,y1 to x2,y2 and material after the move */
char x1[100] = {0};
char y1[100] = {0};
char x2[100] = {0};
char y2[100] = {0};
int futureMaterial[100] = {0};

void searchTheOpponentsPossibilities();
void searchThePossibilities();

void addAPossibility(unsigned a1,unsigned b1,unsigned a2,unsigned b2);
void printPossibilities();
int indexOfPossibilities = 0;
int totalPossibilities = 0;
int materialNow;
int fromPrevLevelLeaveNo;
int value;

};
```

3.4.4 MoveGenerator.C

3.4.5 mchess.flex

```
/*    mchess - chess playing program                                */
/*    This program participates COMPUTER CHESS 1996 KONESHAKKI 1996 */
/*    Team TITAANIT                                                 */

/*    The most important objects are Board *B and MoveGenerator *MG */
```

```
%{
/* place any includes here */
#include<iostream.h>
#include<fstream.h>
#include"Board.h"
#include"MoveGenerator.h"
#include<time.h>
#include"binom.h"

int fieldNo = 1;
int fileNo = 1;
int fileRest = 0;
char *fileName = new char [80];
char *filePostfix = new char [80];
char *varString = new char[80]; /* to store the variable read from input */
ofstream outFile;
char *tuple[10];
void writeToFile();

Board *b =new Board;
char *myMove =new char[6];
char *moveFromMoveGenerator;
/* MoveGenerator MG(b); */
char color = 0;   /* 0=white,  1=black  */


int p;
int p2;
int p3;
long int seconds;
int bestIndexSoFar;
char *bestMove = new char[6];

int level = 1;
MoveGenerator *MG[10][100];
int totalNodesAtLevel[10] = {0};


void invertMove(char *move)
{
move[1]=56-(move[1]-49); /* y1 */
move[3]=56-(move[3]-49); /* y2 */
move[0]=105-(move[0]-97); /* x1 */
move[2]=105-(move[2]-97); /* x2 */
}
```

```
char *
generateMove()
{
seconds = time(NULL);
totalNodesAtLevel[0]=1;
MG[0][1] = new MoveGenerator(b,0);
do {
level+=1;
        for (p=1;p<=totalNodesAtLevel[level-1];p++) {
                for (p2=1;p2<=MG[level-1][p]->totalPossibilities;p2++) {
                        MG[level][p2]=new MoveGenerator(MG[level-1][p]->C,p);
                        totalNodesAtLevel[level]++;
                        MG[level][p2]->C->move(&MG[level-1][p]->x1[p2],
                                               &MG[level-1][p]->y1[p2],
                                               &MG[level-1][p]->x2[p2],
                                               &MG[level-1][p]->y2[p2]);
                        if (odd(level)) {MG[level][p2]->searchThePossibilities();}
                                else {MG[level][p2]->searchTheOpponentsPossibilities();}
                }
        }
} while (time(NULL)-seconds < 40 );



/* count move starting from leaves and ending to root */

/* counting will be started just "above" leaves */
for (p=level-1;p>=0;p--) {
if (even(p)) {
for (p2=1;p2<=totalNodesAtLevel[p];p++) {
        bestIndexSoFar = 1;
        for (p3=1;p3<=MG[p][p2]->totalPossibilities;p3++) {
                if (MG[p+1][p3]->value <= MG[p+1][bestIndexSoFar]->value) {
                bestIndexSoFar = p3;
                }
        }
}
if (MG[p][p2]->value > -50) {  // if king is dead then different value
MG[p][p2]->value = MG[p+1][bestIndexSoFar]->value; }
}

else {
for (p2=1;p2<=totalNodesAtLevel[p];p++) {
        bestIndexSoFar = 1;
        for (p3=1;p3<=MG[p][p2]->totalPossibilities;p3++) {
                if (MG[p+1][p3]->value >= MG[p+1][bestIndexSoFar]->value) {
```

```
                bestIndexSoFar = p3;
                }
        }
}
if (MG[p][p2]->value < 50) { // if opponents king is dead then different value
MG[p][p2]->value = MG[p+1][bestIndexSoFar]->value; }
}

} /* for p */



/* preparing for returning the move */

bestMove[0] = (char)(MG[0][1]->x1[bestIndexSoFar]+97);
bestMove[1] = (char)(MG[0][1]->y1[bestIndexSoFar]+49);
bestMove[2] = (char)(MG[0][1]->x2[bestIndexSoFar]+97);
bestMove[3] = (char)(MG[0][1]->y2[bestIndexSoFar]+49);
bestMove[4] = 0;
/* C->move(bestMove); should the move be stored somewhere? */

// deleting of MGs
for (int i=0; i>=level; i--) {
  for (p=1;p<=totalNodesAtLevel[i];p++) {
    delete MG[i][p];
  }
}
for (i=1;i<11;i++) {
totalNodesAtLevel[i] = 0;
}

return bestMove;

}


%}

DIGIT [0-9]
LOWID [a-z][a-z0-9]*
ID [a-zA-Z][a-zA-Z0-9]*
STRING [a-zA-Z0-9]+
EXTSTRING [a-zA-Z0-9.-:]+
UNSIGNINT {DIGIT}
INTEGER [-]*{DIGIT}+
RESTOFLINE [ \t]*\noutFile.open("saate001.tex", ios::out );
```

```
%%

end { exit(0); }

black {
b->state[3][0] = 5;
                        b->state[4][0] = 6;
                        b->state[3][7] = 14;
                        b->state[4][7] = 13;

/*  b->state = { {2,1,0,0,0,0,9,10},
{3,1,0,0,0,0,9,11},
{4,1,0,0,0,0,9,12},
{5,1,0,0,0,0,9,13},
{6,1,0,0,0,0,9,14},
{4,1,0,0,0,0,9,12},
{3,1,0,0,0,0,9,11},
{2,1,0,0,0,0,9,10}
};
*/
/* change has to be made also to MG */
                        /* MG.C->state[3][1] = 5;
                        MG.C->state[4][1] = 6;
                        MG.C->state[3][7] = 14;
                        MG.C->state[4][7] = 13;
*/

color = 1;
}


white {
                        moveFromMoveGenerator = generateMove();
                        b->move(moveFromMoveGenerator);
cout << moveFromMoveGenerator;
                        delete moveFromMoveGenerator;
}

[a-h][1-8][a-h][1-8] {
if (color) {invertMove(yytext);};
b->move(yytext);
         /* MG.C->move(yytext); */
         moveFromMoveGenerator = generateMove();
         b->move(moveFromMoveGenerator);
```

```
if (color) {invertMove(moveFromMoveGenerator);};
cout << moveFromMoveGenerator;
         delete moveFromMoveGenerator;
}

[o][-][o][-][o] {
/* WRITE COLOR-ADDS ! */
                        b->move(yytext);
                        /* MG.C->move(yytext); */
                        moveFromMoveGenerator = generateMove();
                        b->move(moveFromMoveGenerator);
                        cout << moveFromMoveGenerator;
                        delete moveFromMoveGenerator;

/* long castling */
}

[o][-][o] {
/* WRITE COLOR-ADDS !  */
                        b->move(yytext);
                        /* MG.C->move(yytext); */
                        moveFromMoveGenerator = generateMove();
                        b->move(moveFromMoveGenerator);
                        cout << moveFromMoveGenerator;
                        delete moveFromMoveGenerator;

/* short castling */
}

. {   }

<<EOF>>{ exit(1); }


%%

main()
{
yyin = stdin;
cout << "start" << endl;
yylex();

}
```

4 References

Minimax-algorithm and alpha-beta -pruning:
Hyvönen-Karanta-Syrjänen: Tekoälyn ensyklopedia. Gaudeamus.