DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Relatório Técnico

RT-MAC-9901

A Dynamic Software Certification an Verification Procedure

Carlos Alberto de Bragança Pereira
Fabio Nakano
Julio Michael Stern

março de 1999

A Dynamic Software Certification an Verification Procedure

Carlos Alberto de Bragança Pereira, ***cpereira@ime.usp.br*****, Statistics Department,**

Fabio Nakano, ***nakano@supremum.com*****, Supremum Assessoria e Consultoria,**

Julio Michael Stern, ***jstern@ime.usp.br*****, Computer Science Department,**
Institute of Mathematics and Statistics of the University of São Paulo

ABSTRACT (ACCEPTED SCI'99 ISAS'99)

In Oct-14-1998 ordinance INDESP-104 established the federal software certification and verification requirements for gaming machines in Brazil. The authors present the rationale behind these criteria, whose basic principles can find applications in several other software authentication applications.

Keywords: Software Authentication, Certification, Verification, Gaming, Law Enforcement, Virtual Emulation, Simulation.

1. INTRODUCTION

The certification and validation methodology described in this paper has been the Brazilian standard since Oct-14-1998, by ordinance INDESP-104. The adoption of this standard nationally is the consequence of its successful adoption in the State of São Paulo, since Jan-19-1996, by ordinance CAT-11. In these 4 years, the authors' proposed standard proved to be reliable and enforceable, offering secure technical guidelines for the young Brazilian gaming industry.

2. THE CERTIFICATION AND VERIFICATION PROBLEMS

The gaming regulatory and inspection authority, on behalf of the users, the government and the industry itself, has to make sure that a machine behaves according to the pertinent legislation and also according to the machine's own manual and advertisement.

This responsibility can be divided in two tasks:

- Certification: Check if a specific machine, submitted to certification, is compliant.
- Verification: Check if a given example in the field, is a faithful copy of the specific machine submitted to certification.

Different jurisdictions have established distinct certification and verification methodologies. For the sake of comparison we briefly describe two paradigms:

- Nevada paradigm: The manufacturer has to deliver the source code of the machine, compilation tools, dedicated chips' projects, as well as any emulation hardware requested by the gaming authority. The certification procedure is based on a comprehensive analysis of the software source code and the machine's hardware and its special components' design. The basic verification procedure is based on indirect static comparison by EPROM and hardware checksums and signatures. More extensive tests can be carried out at the laboratory using hardware specific emulation equipment.

- Glib paradigm: The manufacturer has to deliver a machine, its manuals, and little additional information like software fragments. The certification is based on manual tests at the lab, or statistical data collected from machines of the same model in the field. The verification procedure is based mainly on EPROM static checksums and payment records.

None of the paradigms above, nor several other variations adopted by many jurisdictions, seemed appropriate for our use.

The Glib paradigm is flagrantly weak, for several reasons:

- Large prizes imply rare events, for which significant statistics may require huge sample sizes [6] (we give some insights of the statistical difficulties of this methodology in appendix 1).
- Even correct prize frequencies do not guarantee the absence of mechanisms to induce or predict large prizes, what could be used to obtain illegal or unethical benefits.
- Modern machines often have several processors, like CPU and sound and image coprocessors, and custom proprietary VLSI chips large enough to contain a simple processor (like an 8086) and some memory. In these circumstances static checksums are useful to check accidental EPROM damage, but hopelessly impotent against intentional wrongdoing.

The Nevada paradigm is very sound and thorough, but poses other problems like:

- Many manufacturers are unwilling to so completely open their technology and industrial secrets.
- The meticulous analysis of the software (a source code, maybe several thousands lines long, sometimes written in an unfamiliar language) and its platform (an elaborate hardware with exotic components) is a complex, laborious, expensive and time consuming work.
- The absolute technological knowledge and control by the regulating authority creates the risk of power abuse or misuse; an extremely dangerous situation for all parts involved and the society at large.

3. Virtual Machine Dynamic Emulation

In ordinance INDESP-104, we adopted a novel dynamic certification and verification methodology that, we hope, is better suited to the government and the gaming industry needs, and more adapted to the fast evolving technology.

Our dynamic methodology is based on a small portable standalone ANSI-C program emulating the game machine. The emulation program is called the Virtual Machine, VM, and the game machine itself the Real Machine, RM.

The basic requirements over the VM - RM set are:

1.1 The game must be clearly explained in the game machine manual.
1.2 The payoff statistical characteristics must comply with pertinent legislation.
1.3 The game payments table may be dependent on qualified external information (like jackpots), but that information must be easily accessible to the player.
1.4 Unless otherwise stated and clearly specified, any "random choice" in the game assumes a uniform distribution (equal chances).

2.1 The game events must be determined only by:
2.1.1 The player choices or moves in the game.
2.1.2 A special function called the Random Number Generator (RNG).
2.1.3 The machine initial state.
2.2 The game events can not depend or be influenced by any action, signal or information external to the current game, except the information specified at item 2.1.
2.3.1 The RNG shall be implemented in software, and be the only source of "randomness" in a game.
2.3.2 Successive calls to the RNG function shall produce a uniform pseudo-random sequence suited for general statistical simulation [1], [3], [7], [10].

3.1 The manufacturer must provide the following portable ANSI-C programs, well written and documented, as ASCII files:
3.1.1 A portable function emulating of the RNG.
3.1.2 A portable emulation of the game. This program, the Virtual Machine (VM) will take as inputs the player moves from a standard (ASCII) keyboard, and produce ASCII text output describing the game subsequent events.
3.1.3 A synchronization function allowing the synchronization of the VM to any given state of the RM.

4.1 The RM must be able to provide the necessary information for backward (delayed, retroactive) synchronization showing, at the operator's command, the RNG seeds at the beginning of the current period.
4.2 At the beginning of each period, the RM must Rerandomize the seeds. Rerandomize the seeds means calling the RNG function a nondeterministic number of times, based on hardware information (like the machine clock, a measure of elapsed time between events, etc).
4.2.1 The rerandomizations shall not compromise the statistical properties of the RNG as defined in 3.1.1
4.3 A rerandomization shall start a new period at the RM power on, after displaying the current period starting seeds, and at regular intervals between games.
4.3.1 Even knowing the VM program, the information gathered observing the games in a period shall make the determination of the period's starting seeds very unlikely.
4.3.2 The observation of a some consecutive periods shall provide significant statistical evidence that the RM is a faithful implementation of the VM, [9].

4. A Simple Example

File hypogame.cpp in ULR *www.ime.usp.br/~jstern* , [11] illustrates the VM - RM set behavior and interaction for an hypothetical game. Like most manufacturers, hipogame uses a linear congruence generator, in this case a well documented IMSL RNG subroutine, [1], [4], [5], [12].

This RNG has a cycle length of order C = 2^30. At a rerandomization, hypogame calls the RNG function K times, where K is the number of milliseconds remaining in the current second, obtained from the machine's real time clock. Since K <= 1000 < 2^10 << 2^30 = C, hipogame is likely to comply with requirement 4.2.1. Hypogame rerandomizes every N=10 games, and displays the last M=10 periods' starting seeds, so making N*M=100 games available at a time. Appropriately balanced value attributions to constants C, K, M and N achieve requirements 4.3.1 and 4.3.2 compliance [9].

5. Software Certification and Verification Procedures

The RM software certification is based on the VM, for the VM can be used to study all game properties, including statistical simulation of millions of games using the RNG supplied by the manufacturer, and also simulations using other RNG functions.

The RNG and VM functions are usually very small programs and shall be well written and documented. The virtual machine should not contain any special graphics or sound effects of the real machine, for these are useless in the study of the machine properties, and also not portable. Therefore a competent C programmer, that knows nothing about the RM hardware, must be able to read and understand the source code, and compile it in his-her favorite ANSI-C compiler.

These C functions are for exclusive use of the regulatory and inspection authorities, although no great harm should result in the event of a security break. This is why it should not be possible to use the VM to predict (foresee) the outcome of new games. Hence, every time the RM displays the seeds that started the current period, it should start a new period rerandomizing the current seeds.

If the RM offers the player advice for choosing his-her moves in the game, it has to be implemented in the VM and will be used as default; otherwise we may write our own code to simulate a user, with possibly sub-optimal moves. For displaying the seeds the RM can use, for example, a special maintenance screen or provide software to get the information at an RS-232 or ethernet port.

The VM and RNG properties to be analyzed are, among others:
- Conformity of the game to the rules stated at the real machine manual, and the integrity of the information offered to the player.
- Statistical properties of the RNG and the game itself, like payment expected values and higher moments, auto and cross correlation, etc..
- Compliance of the game rules and statistical properties with the pertinent legislation.

After the game properties are analyzed, we must verify that the virtual machine actually emulates the real machine. This is done by sampling event sequences (like cards or numbers drawn) in both machines. In order to be able to match those sequences we have to:

1- Record a sequence of games in the RM, including the player's moves.
2- Access the RM starting period(s) seeds.
3- Enter the starting seeds and replay, at the VM, the recorded sequence.

Any discrepancy between the sequences played at the virtual and the real machine will be considered, by definition, a fraud! The VM and RNG functions must be provided by the company licensing a machine in Brazil, who is liable for any fraud if one is detected.

Field verifications are based on dynamic emulation tests as described in the last two paragraphs, combined with static EPROM and hardware checksums and other proprietary signatures. The inspection agents can use the VMs at a laptop computer remotely connected to a secure server.

6. Conclusions and Further Development

The methodology presented in this paper is now firmly established as the Brazilian national standard for software certification and verification in the gaming industry. Its adoption nationally was a consequence of a very successful 3 year experience in the State of São Paulo, with manufacturers form Australia, Brazil, Canada, China, Israel, Japan, Spain, and the USA.

After some initial apprehension for so radically departing from the standards set at other jurisdictions, most of the parts involved realized the several benefits of this novel methodology. The law enforcement agents, as well as the operators, were specially pleased with the possibility of performing extensive verification tests in loco, requiring simple training and inexpensive equipment.

Most of the complains against the Brazilian standard came from operators trying to use surplus equipment or no longer supported software. This unexpected side effect turned out to be beneficial and in line with Brazilian trade laws prohibiting the import of used equipment and software piracy.

We are currently further developing the cryptology and secure server technology of the project, in a joint research effort of IME-USP - Instituto de Matemática e Estatística da University of São Paulo, and LEARN-PUC-Rio - Laboratório de Engenharia de Algoritmos da Pontifícia Universidade Católica do Rio de Janeiro.

Acknowledgments

We are grateful for the support received from IME-USP - Instituto de Matemática e Estatística da Universidade de São Paulo, CNPq - Conselho Nacional de

Desenvolvimento Científico e Tecnológico, FAPESP - Fundação de Amparo à Pesquisa do Estado de São Paulo, CAT-SF-SP - Coodenação da Administração Tributária da Secretaria da Fazenda do Estado de São Paulo, and from INDESP - Instituto Nacional de Desenvolvimento do Desporto.

We are also grateful to many people for their suggestions, ideas, critics, and support including: Prof. R.Milidiú of LEARN-PUC-Rio -Laboratório de Engenharia de Algoritmos da Pontifícia Universidade Católica do Rio de Janeiro, Prof. J.Pissolato of LAT-UNICAMP –Laboratório de Alta Tensão da Universidade Estadual de Campinas, M.A.Robinson, of The State of Nevada Gaming Control Board -USA, M.Lowell, of Aristocrat Inc. -Australia, J.Ignacio Cases of Sistemas de Televisión -Spain, A.Saffari, of International Game Technology -USA, B.Bruner, of Tekbilt Inc. -USA, J.A.Morales Murga, of Recreativos Franco -Spain, and V.Espíndola, of NVC Eletrônica -Brazil. Finally we thank Prof. N.Callaos, of the International Institute of Informatics and Systemics –USA, for inviting us to present this paper at the International Conference on Information Systems Analysis and Synthesis - ISAS'99, and the World Conference on Systemics, Cybernetics and Informatics - SCI'99.

REFERENCES

[1] P.Bratley, B.L.Fox, L.Schrage. A Guide to Simulation. Springer-Verlag, 1987.

[2] M. DeGroot, Probability and Statistics, Addison Wesley 1986.

[3] G.S.Fishman. Monte Carlo Methods. Springer-Verlag, 1996.

[4] D.E. Knuth. The Art of Computer Programming, vol 2 - Seminumerical Algorithms, Addison Wesley, 1996.

[5] P. L'Ecuyer. Efficient and Portable Combined Pseudo-Random Number Generators. Commun. ACM, 1988.

[6] J.G. Leite, C.A.B. Pereira, F.W. Rodrigues. Waiting Time to Exaust Lottery Numbers. Commun. in Statist. 22, pp. 301-310, 1993.

[7] P.A.W.Lewis, E.J.Orlav. Simulation Methodology. Wadsworth & Brooks Cole, 1989.

[8] C.A.B. Pereira, S. Wechsler. On the concept of P-Value. Brazilian Journal of Probability and Statistics 7, pp. 159-177, 1993.

[9] C.A.B. Pereira, J.M.Stern. Evidence and Credibility: A Full Bayesian Significance Test. Submitted, 1999.

[10] B.D.Ripley. Stochastic Simulation. John Wiley, 1987.

[11] J.M. Stern. Web page, www.ime.usp.br/~jstern

[12] B.A. Wichmann I.D. Hill. An Efficient and Portable Pseudo-Random Number Generator. Appl. Stat. 31, pp. 188-190, 1982.

Appendix 1. Statistical Tests of Compliance

It could be suggested that standard statistical hypothesis test can be used to verify if a machine performs its games accord to its manual. These tests are applied in a large sample of results [6].

Usually, the null hypothesis, H, is the probability distribution described in the manual. The procedure rejects H only if there occurs an event (the sample results) with low probability under this hypothesis. If not, H is not rejected. There could be many other null hypotheses that would not be rejected by the same sample. Many of them could be hypotheses that would be against the player, [8].

Note that, contrary to standard procedures, one needs to control the error of second kind, the probability of accept H when it is false. With so many possibilities of alternative distributions, this control is not possible. If one could consider as null hypothesis the set of all distributions that are against the players, rejecting such hypothesis would guaranty the desired quality of the machine [2], [9] .

APPENDIX 2 - EXCERPTS OF HIPOGAME.CPP

```
/* Hypogame
   Compiler: BORLANDC v3.1 or higher (this one was developed in BC4.52).
   DOS large memory model Application.
*/
#include <dos.h>
#include <conio.h>
#include <stdio.h>
#include <sys\timeb.h>
#include <math.h>
#include <stdlib.h>

/* To generate the real machine, _RM_ macro must be defined and
   _VM_ must not.
   To generate the virtual machine, _VM_ must be defined and _RM_
   must not.
   If both are defined or undefined there will be no compiling error
   but the application will not work correctly.
   Rename the application according to its purpose
   ex.: VM.EXE if _VM_ was defined and RM.EXE if _RM_ was defined.*/

//#define _RM_     RealMachine
#define _VM_  VirtualMachine

#define A1 168071
#define A2 28361
#define M 21474836471
#define P 1277731

#define NUMCORES     4                        /* Number of different
colors. */
#define NUMMESMACOR 13                        /* Number of balls of same
color.*/
#define NUMSORTE     5                        /* Balls in a hand. */
#define NUMBOLAS    (NUMCORES*NUMMESMACOR)    /* All balls in the game. */

/* PRIZE codes */
#define NADA         0         /* nothing */
#define DUQUES       1         /* pair */
#define DOISDUQUES   2         /* two pairs */
#define TRINCA       3         /* three of a kind */
#define TRINCAEDUPLA 4         /* full house */
#define CORES        5         /* flush */
#define SEQUENCIA    6         /* straight */
#define QUADRA       7         /* four */
#define SEQCORES     8         /* straight flush */

long int seed;           /* semente do gerador de numeros aleatorios. */
                         /* RNG seed */
```

```
int bola[NUMBOLAS];      /* As bolas do jogo: cor*NUMMESMACOR+numero. */
                         /* Array of balls in the game */

int sorteada[NUMSORTE]; /* As bolas sorteadas. */
                         /* Array of balls in a hand */

int select[NUMSORTE];    /* selecionada=1, nao selecionada=0. */
                         /* hold balls - hold=1, release=0 */

int ordenada[NUMSORTE]; /* Bolas sorteadas ordenadas, desprezando as
cores.*/
                         /* sorted balls color insensitive. */

int posicao[NUMSORTE];   /* Posicao de cada bola - 0..NUMSORTE. */
                         /* position of each sorted ball */

int cor[NUMCORES];      /* Seleciona as bolas sorteadas por cor. */
                        /* Array of counters of same color balls in the
hand */

int repet[NUMMESMACOR]; /* Conta cartas de numeros repetidos. */
                        /* Array of counters of balls of same number */

int ultimabola;         /* Ultima bola da urna */
                        /* index of the last not in hand ball */

char strpremio[9][32]= { "Sorry, no prize", "pair", "two pairs",
                         "Three of a kind", "Full House", "Flush.",
                         "Straight!", "Four!", "Straight flush!!"};
#ifdef _RM_
long int lastseed[10], lastgame[10];
long int countgames;
#endif

char codcores[4][4]={"Bl","Gr","Cy","Rd"};

long int Rand (void)
/* Gerador de numeros aleatorios */
{
    long int k1;
    k1=seed/P;
    seed=A1*(seed-k1*P) - k1*A2;
    if (seed<=01) seed=seed+M;
    return (seed);
}

void sRand (long int newseed)
/* Sincronizador */
{
    seed=newseed;
}

void scramble (void)
```

```
/* inicializa o vetor (ou enche a urna) e mistura as bolas. */
/* scrambles the balls */
{
    int i, j, k, aux;
    /* inicializa o vetor */
    for (k=0;k<NUMBOLAS;k++) bola[k]=k;
    ultimabola=NUMBOLAS-1;
    for (k=0;k<3000; k++) {
        i=(int) (Rand()&0x7FFF)%NUMBOLAS;
        j=(int) (Rand()&0x7FFF)%NUMBOLAS;
        aux=bola[i];
        bola[i]=bola[j];
        bola[j]=aux;
    }
}

void clearselection (void)
/* libera todas as bolas. */
{
    int i;
    for (i=0;i<NUMSORTE;i++) select[i]=0;
}

void getballs (int select[])
/* sorteia as bolas que nao foram selecionadas, logo
ate' NUMSORTE bolas da urna - remove as bolas da urna. */
{
    int i, pos;
    for (i=0;i<NUMSORTE;i++) {
        if (select[i]==0) {
            pos=(int) (Rand()&0x7FFF)%ultimabola;
            sorteada[i]=bola[pos];
            bola[pos]=bola[ultimabola];
            ultimabola--;
        }
    }
}

void showballs (int linha)
/* mostra as bolas sorteadas na linha da tela */
{
    int i, passo;
    passo=80/NUMSORTE;
    gotoxy (1, linha);
    clreol();
    for (i=0;i<NUMSORTE;i++) {
        gotoxy(passo*i+1, linha);
        textcolor ((sorteada[i]/NUMMESMACOR)+9);
        cprintf ("%2d", sorteada[i]%NUMMESMACOR);
    }
}

void showhold (int linha)
/* Mostra a sugestao - bolas retidas na cartela */
```

```
{
    int i, passo;
    passo=80/NUMSORTE;
    gotoxy (1, linha);
    clreol();
    for (i=0;i<NUMSORTE;i++) {
        if (select[i]) {
            gotoxy(passo*i+1, linha);
            textcolor ((sorteada[i]/NUMMESMACOR)+9);
            cprintf ("XX");
        }
    }
}

void showremainindballs (int linha)
/* Mostra as bolas que ficaram na urna na linha da tela. */
{
    int i;
    textcolor(WHITE);
    window (1, linha, 80, linha+4);
    cprintf ("Urna:\n");
    for (i=0;i<=ultimabola;i++) {
        textcolor ((bola[i]/NUMMESMACOR)+9);
        cprintf ("%2d  ", bola[i]%NUMMESMACOR);
    }
    textcolor(WHITE);
    cprintf ("\n\r=%ld=", seed);
    window (1, 1, 80, 25);
}

int prize (int select[])
/* Reconhece o premio, retornando seu codigo. */
{
    int i, j, aux, paux, codpremio;
    int nquadras,
        ntrincas,
        nduques,
        sequencia,
        cores;

    nquadras=0; ntrincas=0; nduques=0; sequencia=0; cores=0; codpremio=0;

    /* conta bolas sorteadas de mesma cor -
       counts balls of same color */
    for (i=0;i<NUMCORES;i++) cor[i]=0;
    for (i=0;i<NUMSORTE;i++) cor[sorteada[i]/NUMMESMACOR]++;
    for (i=0;i<NUMCORES;i++) {
        if (cor[i]==NUMSORTE) cores=1; /* todas as bolas de mesma cor. */
    }


    /* conta bolas sorteadas de mesmo numero
       counts balls of same number. */
    for (i=0;i<NUMMESMACOR;i++) repet[i]=0;
for (i=0;i<NUMSORTE;i++) repet[sorteada[i]%NUMMESMACOR]++;
for (i=0;i<NUMMESMACOR;i++) {
    if (repet[i]==2) {
        nduques++;
        for (j=0;j<NUMSORTE;j++)
            if ((sorteada[j]%NUMMESMACOR)==i)
                select[j]=1; /* sugestao */
    }
    if (repet[i]==3) {
        ntrincas++;
        for (j=0;j<NUMSORTE;j++)
            if ((sorteada[j]%NUMMESMACOR)==i)
                select[j]=1; /* sugestao */
    }
    if (repet[i]==4) {
        nquadras++;
        for (j=0;j<NUMSORTE;j++)
            if ((sorteada[j]%NUMMESMACOR)==i)
                select[j]=1; /* sugestao */
    }
}

/* ordena bolas desprezando as cores.
   sorts the balls by number. */
for (i=0;i<NUMSORTE;i++) {
    ordenada[i]=sorteada[i]%NUMMESMACOR;
    posicao[i]=i;
}
/* bubble sort */
for (i=0;i<NUMSORTE-1;i++) {
    for (j=NUMSORTE-1;j>i;j--) {
        aux=ordenada[j];
        paux=posicao[j];
        if (aux<ordenada[j-1]) {
            ordenada[j]=ordenada[j-1];
            ordenada[j-1]=aux;
            posicao[j]=posicao[j-1];
            posicao[j-1]=paux;
        }
    }
}
/* premia sequencias - straight */
if ((ordenada[0]==(ordenada[1]-1))&&(ordenada[1]==(ordenada[2]-1))&&
     (ordenada[2]==(ordenada[3]-1))&&(ordenada[3]==(ordenada[4]-1))) {
    sequencia=1;
    for (i=0;i<NUMSORTE;i++) select[i]=1;     /* sugestao */
}
/* duques - pair */
if (nduques==1) codpremio=DUQUES;
/* dois duques two pairs*/
if (nduques==2) codpremio=DOISDUQUES;
/* trinca - three of a kind*/
if (ntrincas==1) codpremio=TRINCA;
/* trinca e dupla - full*/
```

```
    if ((ntrincas==1)&&(nduques==1)) codpremio=TRINCAEDUPLA;
    /* todas de mesma cor - flush */
    if (cores) codpremio=CORES;
    /* sequencia de numeros - straight*/
    if (sequencia) codpremio=SEQUENCIA;
    /* quadra - four*/
    if (nquadras) codpremio=QUADRA;
    /* sequencia de cores - straight flush */
    if (sequencia&&cores) codpremio=SEQCORES;

    return (codpremio);
}

void showlabel (int x, int y)
{
    gotoxy(x, y);
    textcolor (0x8D);
    cprintf ("=============================");
    gotoxy(x, y+1);
#ifdef _RM_
    cprintf ("  This is the Real Machine");
#endif
#ifdef _VM_
    cprintf ("This is the Virtual Machine.\n");
#endif
    gotoxy(x, y+2);
    cprintf ("=============================");
}

void main (void)
{
    int codpremio;
    char c;
#ifdef _RM_
    int i, currentindex, showedseeds;
    FILE *fpOut;
    struct timeb t;
    ftime (&t);
    seed=t.time;
    for (i=1000;i>t.millitm;i--) Rand ();
    countgames=0;
    currentindex=0;
    showedseeds=0;
    for (i=0;i<10;i++) { lastseed[i]=0; lastgame[i]=0; }
    fpOut=fopen ("fotos.txt", "wt");
    if (!fpOut) {
        perror ("fotos.txt");
        exit (0);
    }
#endif

#ifdef _VM_
    long semente;
    /* Sincronizacao */
    printf ("Type the seed to synchronize RM and VM.\n");
    scanf ("%ld", &semente);
    sRand(semente);
#endif

    c='\0';

 /* Jogo - Game*/
    while (c!=27) {
#ifdef _RM_
        if (c==9) {
            gotoxy (1,15);
            for (i=0;i<10;i++)
                printf ("s[%ld]= %ld\n", lastgame[i], lastseed[i]);
            printf ("Press a key to continue. ");
            while (!kbhit());
            c=getch();
            showedseeds=1;
        }
        /* Rerandomize based on internal clock. */
        if (((countgames%10)==0)||(showedseeds)) {
            ftime (&t);
            for (i=1000;i>t.millitm;i--) Rand ();
            lastseed[currentindex]=seed;
            lastgame[currentindex]=countgames;
            currentindex++;
            currentindex%=10;
            showedseeds=0;
        }
        countgames++;
#endif
        clrscr();
        scramble();
        clearselection();
//         showremainindballs(2);
        showlabel (40, 20);
        getballs(select);
        showballs(10);
        codpremio=prize(select);
        showhold(11);
#ifdef _RM_
        fprintf (fpOut, "1st hand\n");
        for (i=0;i<NUMSORTE;i++) {
            fprintf (fpOut, "%s%02d ", codcores[sorteada[i]/13],
                                       sorteada[i]%13);
        }
        fputc ('\n', fpOut);
        for (i=0;i<NUMSORTE;i++) {
            fprintf (fpOut, "  %d  ", select[i]);
        }
        fputc ('\n', fpOut);
#endif
        c='\0';
        while (c!=' ') {
            gotoxy (1, 1);
            clreol ();
#ifdef _RM_
            printf ("1..5= toggle hold, space= continue, tab= show
seeds.\n");
#endif
#ifdef _VM_
            printf ("1..5 to toggle hold, space to continue.\n");
#endif
            c=getch();
            switch (c) {
                case '1':
                    if (select[0]) select[0]=0;
                    else select[0]=1;
                    break;
                case '2':
                    if (select[1]) select[1]=0;
                    else select[1]=1;
                    break;
                case '3':
                    if (select[2]) select[2]=0;
                    else select[2]=1;
                    break;
                case '4':
                    if (select[3]) select[3]=0;
                    else select[3]=1;
                    break;
                case '5':
                    if (select[4]) select[4]=0;
                    else select[4]=1;
                    break;
            }
            showhold(11);
        }
#ifdef _RM_
        fprintf (fpOut, "Hold\n");
        for (i=0;i<NUMSORTE;i++) {
            fprintf (fpOut, "   %d   ", select[i]);
        }
        fputc ('\n', fpOut);
#endif
        getballs(select);
        showballs(12);
        clearselection ();
        codpremio=prize(select);
        showhold(13);

#ifdef _RM_
        fprintf (fpOut, "2nd hand\n");
        for (i=0;i<NUMSORTE;i++) {
            fprintf (fpOut, "%s%02d ", codcores[sorteada[i]/13],
                                              sorteada[i]%13);
        }
        fputc ('\n', fpOut);
        for (i=0;i<NUMSORTE;i++) {
            fprintf (fpOut, "  %d  ", select[i]);
        }
        fputc ('\n', fpOut);
        fprintf (fpOut, "\n%s\n\n", strpremio[codpremio]);
#endif
        printf ("\n%s\n", strpremio[codpremio]);
        gotoxy (1, 1);
        printf ("ESC to end, any other key to continue.\n");
        while (!kbhit());
        c=getch();
    }
#ifdef _RM_
    fclose (fpOut);
#endif
}
```

RELATÓRIOS TÉCNICOS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Instituto de Matemática e Estatística da USP

A listagem contendo os relatórios técnicos anteriores a 1996 poderá ser consultada ou solicitada à Secretaria do Departamento, pessoalmente, por carta ou e-mail(mac@ime.usp.br).

Daniela V. Carbogim and Flávio S. Corrêa da Silva
FACTS, ANNOTATIONS, ARGUMENTS AND REASONING
RT-MAC-9601, janeiro de 1996, 22 pp.

Kunio Okuda
REDUÇÃO DE DEPENDÊNCIA PARCIAL E REDUÇÃO DE DEPENDÊNCIA GENERALIZADA
RT-MAC-9602, fevereiro de 1996, 20 pp.

Junior Barrera, Edward R. Dougherty and Nina Sumiko Tomita
AUTOMATIC PROGRAMMING OF BINARY MORPHOLOGICAL MACHINES BY DESIGN OF STATISTICALLY OPTIMAL OPERATORS IN THE CONTEXT OF COMPUTATIONAL LEARNING THEORY.
RT-MAC-9603, abril de 1996, 48 pp.

Junior Barrera e Guillermo Pablo Salas
SET OPERATIONS ON CLOSED INTERVALS AND THEIR APPLICATIONS TO THE AUTOMATIC PROGRAMMINIG OF MMACH'S
RT-MAC-9604, abril de 1995, 66 pp.

Kunio Okuda
CYCLE SHRINKING BY DEPENDENCE REDUCTION
RT-MAC-9605, maio de 1996, 25 pp.

Julio Stern, Fabio Nakano e Marcelo Lauretto
REAL: REAL ATTRIBUTE LEARNING FOR STRATEGIC MARKET OPERATION
RT-MAC-9606, agosto de 1996, 16 pp.

Markus Endler
SISTEMAS OPERACIONAIS DISTRIBUÍDOS: CONCEITOS, EXEMPLOS E TENDÊNCIAS
RT-MAC-9607, agosto de 1996, 120 pp.

Hae Yong Kim
CONSTRUÇÃO RÁPIDA E AUTOMÁTICA DE OPERADORES MORFOLÓGICOS E EFICIENTES PELA APRENDIZAGEM COMPUTACIONAL
RT-MAC-9608, outubro de 1996, 19 pp.

Marcelo Finger
NOTES ON COMPLEX COMBINATORS AND STRUCTURALLY-FREE THEOREM PROVING
RT-MAC-9609, dezembro 1996, 28 pp.

Carlos Eduardo Ferreira, Flávio Keidi Miyazawa e Yoshiko Wakabayashi (eds)
ANAIS DA I OFICINA NACIONAL EM PROBLEMAS DE CORTE E EMPACOTAMENTO
RT-MAC-9610, dezembro de 1996, 65 pp.

Carlos Eduardo Ferreira, C. C. de Souza e Yoshiko Wakabayashi
REARRANGEMENT OF DNA FRAGMENTS: A BRANCH-AND-CUT ALGORITHM
RT-MAC-9701, janeiro de 1997, 24 pp.

Marcelo Finger
NOTES ON THE LOGICAL RECONSTRUCTION OF TEMPORAL DATABASES
RT-MAC-9702, março de 1997, 36 pp.

Flávio S. Corrêa da Silva, Wamberto W. Vasconcelos e David Robertson
COOPERATION BETWEEN KNOWLEDGE BASED SYSTEMS
RT-MAC-9703, abril de 1997, 18 pp.

Junior Barrera, Gerald Jean Francis Banon, Roberto de Alencar Lotufo, Roberto Hirata Junior
MMACH: A MATHEMATICAL MORPHOLOGY TOOLBOX FOR THE KHOROS SYSTEM
RT-MAC-9704, maio de 1997, 67 pp.

Julio Michael Stern e Cibele Dunder
PORTFÓLIOS EFICIENTES INCLUINDO OPÇÕES
RT-MAC-9705, maio de 1997, 29 pp.

Junior Barrera e Ronaldo Fumio Hashimoto
COMPACT REPRESENTATION OF W-OPERATORS
RT-MAC-9706, julho de 1997, 13 pp.

Dilma M. Silva e Markus Endler
CONFIGURAÇÃO DINÂMICA DE SISTEMAS
RT-MAC-9707, agosto de 1997, 35 pp

Kenji Koyama e Routo Terada
AN AUGMENTED FAMILY OF CRIPTOGRAPHIC PARITY CIRCUITS
RT-MAC-9708, setembro de 1997, 15 pp

Routo Terada e Jorge Nakahara Jr.
LINEAR AND DIFFERENTIAL CRYPTANALYSIS OF FEAL-N WITH SWAPPING
RT-MAC-9709, setembro de 1997, 16 pp

Flávio S. Corrêa da Silva e Yara M. Michelacci
MAKING OF AN INTELLIGENT TUTORING SYSTEM (OR METHODOLOGICAL ISSUES OF ARTIFICIAL INTELLIGENCE RESEARCH BY EXAMPLE)
RT-MAC-9710, outubro de 1997, 16 pp.

Marcelo Finger
COMPUTING LIST COMBINATOR SOLUTIONS FOR STRUCTURAL EQUATIONS
RT-MAC-9711, outubro de 1997, 22 pp.

Maria Angela Gurgel and E.M.Rodrigues
THE F-FACTOR PROBLEM
RT-MAC-9712, dezembro de 1997, 22 pp.

Perry R. James, Markus Endler, Marie-Claude Gaudel
DEVELOPMENT OF AN ATOMIC-BROADCAST PROTOCOL USING LOTOS
RT-MAC-9713, dezembro de 1997, 27 pp.

Carlos Eduardo Ferreira and Marko Loparic
A BRANCH-AND-CUT ALGORITHM FOR A VEHICLE ROUTING PROBLEM WITH CAPACITY AND TIME CONSTRAINTS
RT-MAC-9714, dezembro de 1997, 20 pp.

Nami Kobayashi
A HIERARCHY FOR THE RECOGNIZABLE M-SUBSETS
RT-MAC-9715, dezembro de 1997, 47 pp.

Flávio Soares Corrêa da Silva e Daniela Vasconcelos Carbogim
A TWO-SORTED INTERPRETATION FOR ANNOTATED LOGIC
RT-MAC-9801, fevereiro de 1998, 17 pp.

Flávio Soares Corrêa da Silva, Wamberto Weber Vasconcelos, Jaume Agustí, David Robertson e Ana Cristina V. de Melo.
WHY ONTOLOGIES ARE NOT ENOUGH FOR KNOWLEDGE SHARING
RT-MAC-9802, outubro de 1998, 15 pp.

J. C.de Pina e J. Soares
ON THE INTEGER CONE OF THE BASES OF A MATROID
RT-MAC-9803, novembro de 1998, 16 pp.

K. Okuda and S.W.Song
REVISITING HAMILTONIAM DECOMPOSITION OF THE HYPERCUBE
RT-MAC-9804, dezembro 1998, 17pp.

Markus Endler
AGENTES MÓVEIS: UM TUTORIAL
RT-MAC-9805, dezembro 1998, 19pp.

Carlos Alberto de Bragança Pereira, Fabio Nakano e Julio Michael Stern
A DYNAMIC SOFTWARE CERTIFICATION AN VERIFICATION PROCEDURE
RT-MAC-9901, MARÇO 1999, 21pp.