Universidad de Buenos Aires

Facultad de Ingeniería

66.20 - Organización de Computadoras

Trabajo Practico 1

Primer cuatrimestre de 2021

Hojman de la Rosa, Joaquin Guido	102264	jhojman@fi.uba.ar
Giampieri Mutti, Leonardo	102358	lgiampieri@fi.uba.ar
Giardina, Fernando	103732	fgiardina@fi.uba.ar

Índice

1. Introducción

Los autómatas celulares son un caso de modelo matemático para un sistema dinámico que evoluciona en pasos discretos. Originalmente creados en el contexto de la física computacional, se utilizan en varios otros campos, como la teoría de la computabilidad. Mientras que muchos autómatas celulares, como en el caso del Juego de la Vida de Conway, evolucionan en una matriz bidimensional, otros, como los autómatas celulares elementales o unidimensionales, lo hacen en un array unidimensional. Esto nos permite almacenar la evolución en el tiempo del autómata como una segunda dimensión. En los autómatas celulares elementales, las celdas en las que está dividido nuestro universo pueden contener una célula o no, y el estado de cada celda en la iteración i depende del estado de esa celda y sus vecinos inmediatos a izquierda y derecha en la iteración anterior.

Por ejemplo, si expresamos el estado de una celda y sus dos celdas adyacentes como dígitos binarios, donde 0 representa una celda vacía y 1 una ocupada, podemos determinar unívocamente el comportamiento de un autómata celular elemental asignando un dígito binario a cada combinación de tres bits, donde 1 representa que una celda con ese contenido y esos vecinos está ocupada en la siguiente iteración, y 0 representa que no.

1.1. Objetivos

El objetivo del primer trabajo practico de la materia es familiarizarse con el conjunto de instrucciones MIPS32 y el concepto de ABI, escribiendo un programa portable que resuelva un problema determinado. Usaremos el programa QEmu para simular el entorno de desarrollo de una máquina MIPS corriendo una versión reciente del sistema operativo Debian.

Pese a contenter fragmentos en assembler MIPS32, la implementación desarrollada proveera un grado de portabilidad. Es decir que se incluyiran dos versiones de la funcion proximo() que calcula el estado siguiente de una celda, una en MIPS32 y una en C pensada para dar soporte genérico a aquellos entornos que carezcan de una versión más específica.

1.2. ABI

El pasaje de parámetros entre el código C (main(), etc) y la rutina proximo(), en assembler, se hará usando la ABI explicada en clase: los argumentos correspondientes a los registros $a0-a3$ serán almacenados por el callee, siempre, en los 16 bytes dedicados de la sección "function call argument area".

2. ABI

La ABI es una interfaz que nos permite intercambiar información entre funciones, siendo ésta almacenada en una sección específica de memoria.

En nuestro caso, lo que se tiene es la función llamada proximo() que es una función hoja, ya que no llama a ninguna otra función. El pasaje de parámetros entre el código C (start()) y la rutina proximo(), en assembler, se hará usando la ABI vista en clase: los argumentos correspondientes a los registros $a0-a3$ serán almacenados por el callee, siempre, en los, por lo menos, 16 bytes dedicados de la sección "function call argument area". Hacemos aquí un gráfico para poder explicar mejor lo anterior.

2.1. Gráfico stack proximo() genérico

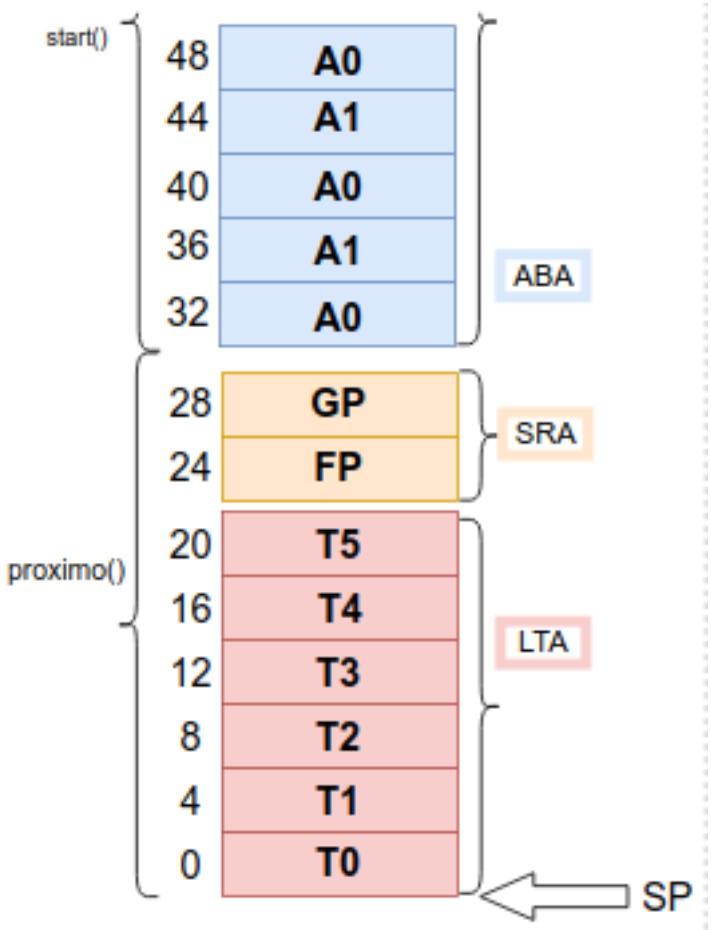


Figura 1: *Gráfico genérico del stack de la función proximo().*

En el gráfico, podemos observar cómo se organiza el stack para la función proximo(), la cual está dividida principalmente en dos secciones: SRA y LTA. Además, tenemos el ABA del caller que es un segmento específico de las funciones no hojas, los cuales consisten en una cantidad mayor o igual a 16 bytes en donde se almacenan los registros adicionales (si los hubiera) a los ai con i entre 0 y 3. Luego tenemos otra área, Local Temporary Area, que es en la cual se almacenan las variables locales y temporales, en la cual tenemos en este caso 8 bytes para

variables temporales y 8 bytes para variables locales. Por otro lado, tenemos que guardar, en toda función el frame pointer y el global pionter. En este caso, al ser proximo() una función hoja, no debemos preservar el register address(ra).

2.2. Gráfico stack proximo() con nombres de variables

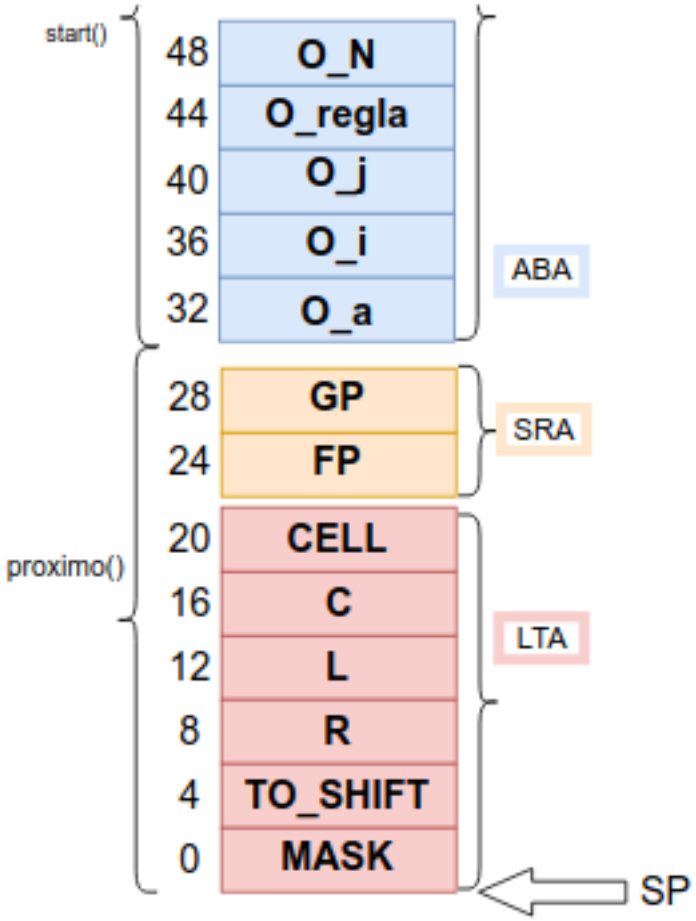


Figura 2: *Gráfico del stack de la función proximo() con nombres de variables.*

En el gráfico superior se presenta, con mayor detalle, un esquema en donde observamos cada una de las variables que componen el área de memoria del stack de la función proximo().

3. Implementación

En esta sección se describen los detalles de implementación del programa. En este trabajo, el desarrollo se hizo parte en C y parte en lenguaje Assembler. Los programas escritos fueron compilados o ensamblados según el caso, y posteriormente enlazados, utilizando las herramientas de GNU disponibles en el sistema Debian utilizado. Como resultado del enlace, se genera la aplicación ejecutable.

3.1. Funcionamiento general

El programa en C inicia interpretando los argumentos de entrada. Luego pide memoria para un arreglo que tendra dimension NxN chars. El motivo de esto es que cada fila tendra N celdas, y el arreglo debe tener N filas, por eso NxN. Luego se llama a una función getcellvalues() que lo que hace es leer el archivo de entrada, validando que exista, que contenga solo ceros y unos y que tenga N celdas. Luego esta función carga al arreglo con la fila inicial que aparece en el archivo de entrada. Cada celda es mapeada como un char que puede valer 0 o 1.

Luego se realiza una iteración n*(n-1) veces (esto debido a que la primera iteración ya se realizo cuando se cargo ella rreglo). En cada iteración se llama a la función proximo (detallada mas adelante) con los siguientes parametros: “a” como el arreglo, n como el numero de celdas, i/n como la fila I, y i mod n como la columna J. Los valores de I y J garantizan el correcto funcionamiento de la función proximo. Luego el carácter obtenido es guardado en a[i+n]. Es decir que nuestro programa funciona como un vector largo, antes que como un vector de vectores (una matriz).

Por ultimo se mapea el arreglo a una imagen con formato PBM, representando las celdas encendidas con color blanco y las apagadas con color negro. Recordemos que nuestro arreglo es un vector, no una matriz, y por ende cuando (i+1) mod n = 0, es decir cada N guardados en el archivo PBM, se imprime un salto de linea.

3.2. Función Proximo()

El siguiente estado para cada celda se obtiene llamando a la funcion

unsigned char proximo(unsigned char *a, unsigned int i, unsigned int j, unsigned char regla, unsigned int N);

Donde a es un puntero a la posición [0, 0] de la matriz, i y j son la fila y la columna respectivamente del elemento cuyos vecinos queremos calcular, y N es la cantidad de filas y columnas de la matriz A. El valor de retorno de la función proximo es el valor de ocupación de la celda [i+ 1, j], o sea el estado de la celda j en la iteración i+1.

Entonces J es la celda a la que quiero acceder e I es la fila. Pero como A es un arreglo, habra que moverse I*N bytes, lo que equivaldria a A[i][j], y en cada celda obtendremos un 1 o un 0. Entonces char c = a[j + i*N] es nuestro carácter, de manera analoga calculamos el carácter de su derecha (R) y de su izquierda (L), pero debemos tener en cuenta las condiciones de borde, que son las siguientes.

- Si J es el bit 0, es la celda de mas a la izquieda, por lo que L= a[j + i*N + N – 1], es decir que salta hasta el ultimo bit de esa fila, y R = a[j + i*N + 1], la celda derecha sin problemas.

- Si J es el bit N-1, es decir la fila de mas a la derecha, R=a[j + i*N - N + 1], es decir que salta hasta el primer bit de esa fila, y L = a[j + i*N – 1], la celda izquierda sin problemas.
- En cualquier otro caso, L=a[j + i * N – 1] y R=a[j + i * N + 1], es decir las celdas izquierda y derecha del carácter en cuestión.

Ahora ya obtuvimos el carácter de la posición que corresponde, y los caracteres de la izquierda y derecha, pero aun resta calcular en base a estos valores el valor que tendrá el proximo. Para eso aplicamos un shift a L de 2 posiciones y un shift a C de 1 posición. No hace falta aplicar un shift sobre R. Luego aplicamos un or entre L, C y R , y hacemos un shift de tipo 0x1 de la siguiente forma: char mask = 0x1 « L | C | R. Por ultimo realizamos la operación Regla & mask y devolvemos el valor del bit, como 1 o 0.

»4. Conclusiones

»Como conlusión, se puede afirmar que se alcanzo el objetivo del trabajo que era implementar el algoritmo de de la función Proximo en MIPS 32. El trabajo sirvió como una buena fuente de aprendizaje para nosotros en el uso del lenguaje. Además cabe destacar que el algoritmo fue implementado enteramente en C para luego pasar al Assembly de MIPS para guiarnos en por dónde llevar el programa (el algoritmo en C se encuentra tanto en el apéndice como en el release de git entregado) y que, por otro lado, nos sirvió para entender cómo implementar código Assembly en un entorno de C.

»Adquirimos conocimiento tanto en ciertas instrucciones MIPS, como también en la ABI. Ésta suponía un cambio radical a lo que sabíamos de Assembly ARC, y en cierto punto pudimos aprender los lineamientos principales que se proponen con dicha interfaz. También nos sirvió para visualizar, a partir de los gráficos que se han realizado en el trabajo, cómo queda organizado finalmente el stack, luego del linkeo de distintos programas. Por último destacamos que el trabajo fue sometido a pruebas y los resultados fueron muy positivos al poder responder como se esperaba frente a casos generales y también casos bordes.

»5. Corridas de prueba

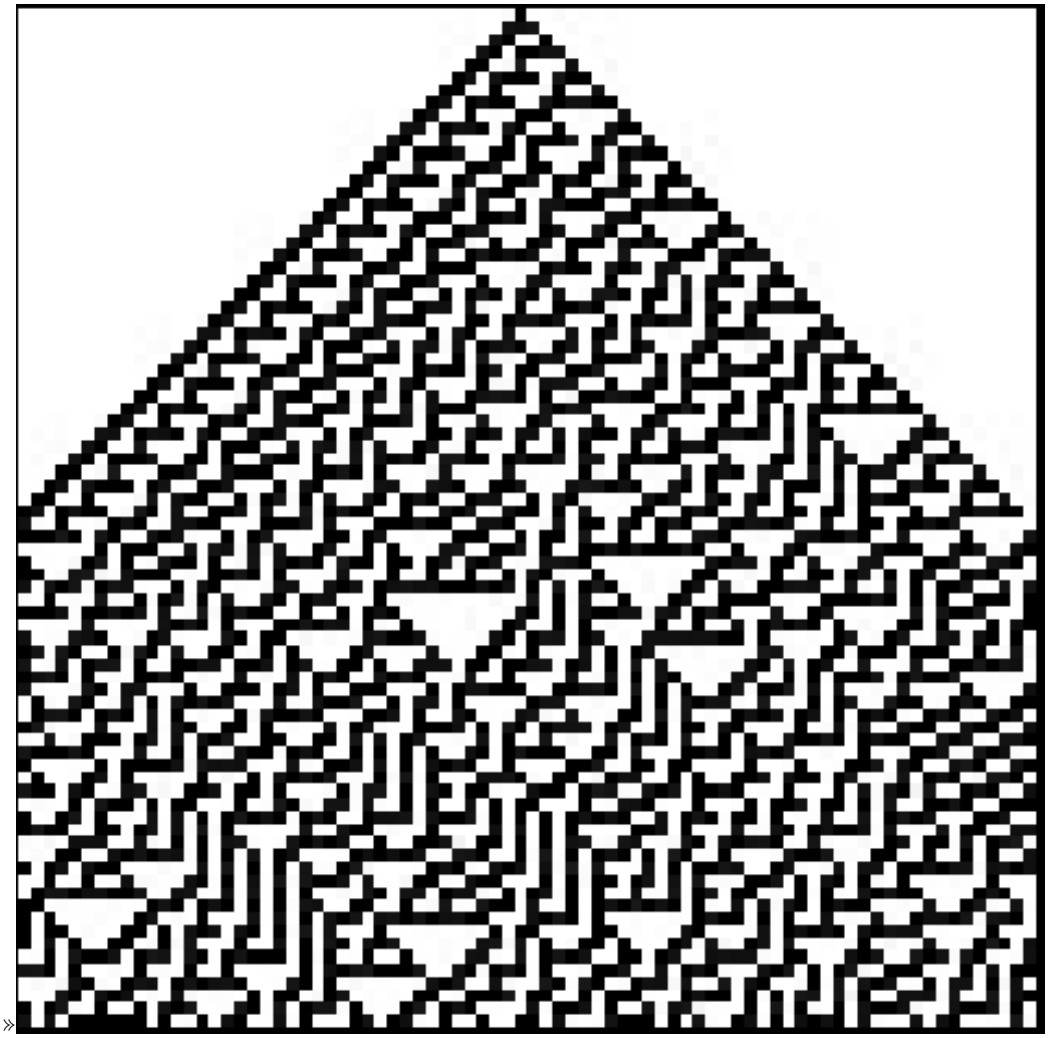
»A continuación se incluyen las corridas de prueba solicitadas, es decir el caso de una matriz de lado 80, con un estado inicial conformado por una única celda. A este estado inicial se le aplicaran las reglas 30, 110 y 126, y podrán observarse los patrones que se forman a partir de ellos.

»Las imágenes que se mostraran son los archivos .pbm que el programa arma a partir de las 80 iteraciones con las reglas indicadas:

»Estado Inicial:

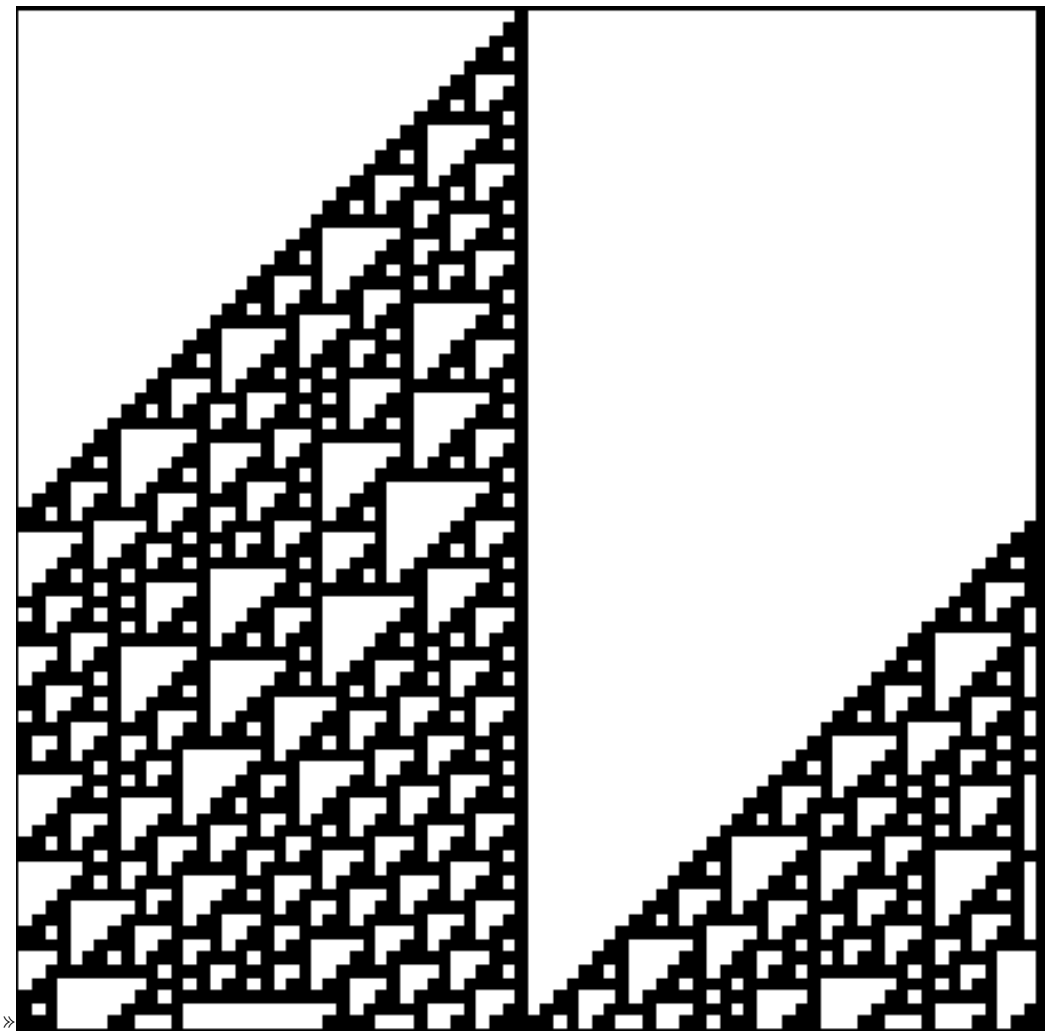
»00000000000000000000000000000000000000010000000000000000000000000000000000000000

»5.1. Regla 30



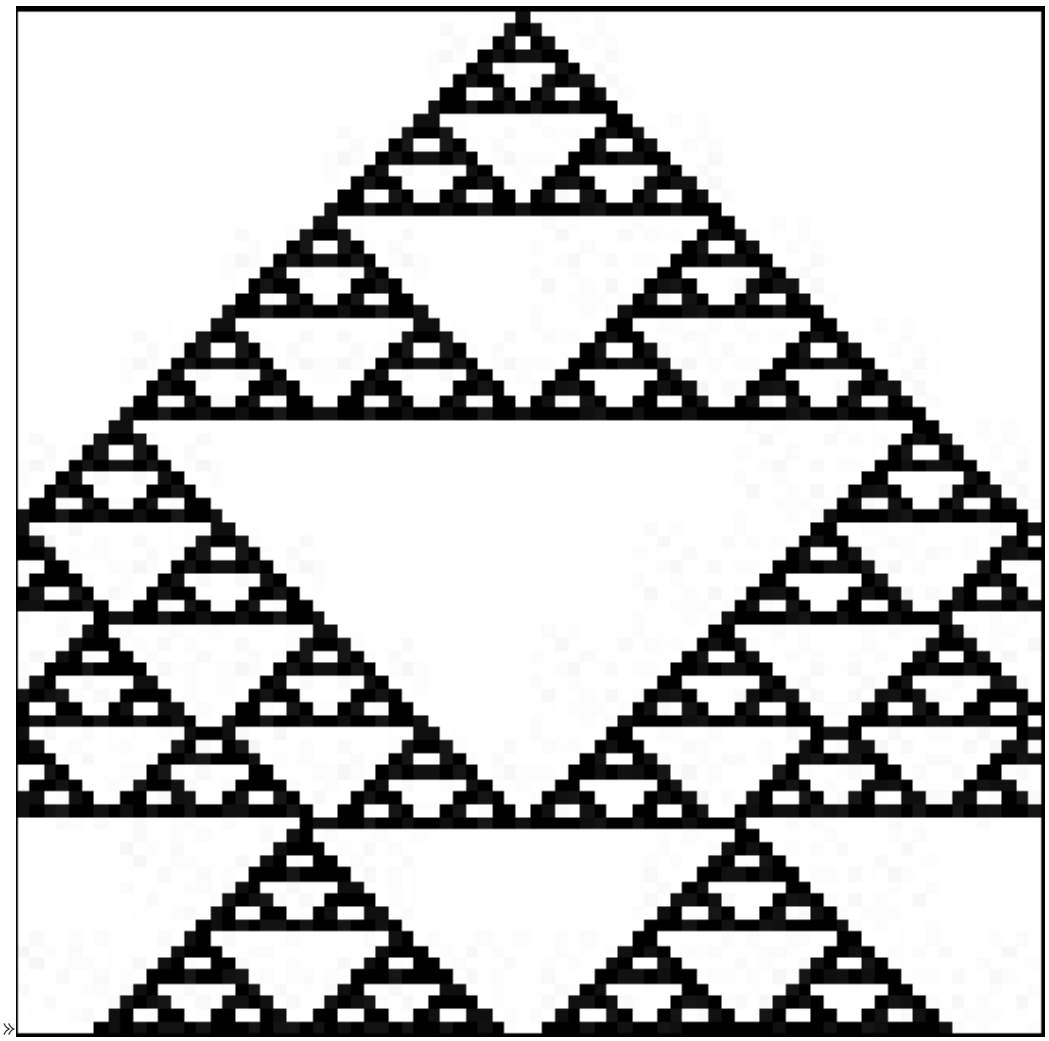
» »Figura 3: Regla 30

»5.2. Regla 110



» »Figura 4: Regla 110

»5.3. Regla 126



»Figura 5: Regla 126

6. Apendice

6.1. main.c

```
#include <stdio.h>
#include <string.h>
#include "autcel.h"

int main(int argc, char* const* argv){
    int opt;
    while (1){
        static struct option options[] = {
            {"help", no_argument, 0, 'h'},
            {"version", no_argument, 0, 'v'},
            {0,0,0,0}
        };
        opt = getopt_long(argc, argv, "hvo::", options, 0);
        if (opt == -1){
            break;
        }
        switch (opt){
            case 'h':
                printf("Uso:\n");
                printf(" autcel -h\n autcel -V\n autcel R N
                inputfile [-o outputprefix]");
                printf(" Opciones:\n");
                printf(" -h --help Imprime este mensaje.\n");
                printf(" -V --version Da la version de este
                programa.\n");
                printf(" -o Prefijo de los archivos de
                salida.\n");
                break;
            case 'v':
                printf("version %s\n", V);
                break;
            case 'o':
                        if (argc != 5 && argc != 6){
                    printf("%d\n", argc);
                            fprintf(stderr,"%s", "-o received
                            wrong number of arguments,
                            expected 6\n");
                    return -1;
                        }
                unsigned char* a = start(atoi(argv[1]),
                atoi(argv[2]), argv[3]);
                if (!a){
```

```
»                    return -1;
»                }
»                        build_pbm((argc == 6 ? argv[5] :
»                        argv[3]), a, atoi(argv[2]));
»                break;
»            default:
»                printf("Error\n");
»                return -1;
»        }
»    }
»
»    return 0;
»}
```

»6.2. autcel.h

```
»#include <stdio.h>
»#include <stdlib.h>
»#include <unistd.h>
»#include <getopt.h>
»#include <string.h>
»#include "file_reader.h"
»#define V "0.0.1"
»
»extern unsigned char proximo(unsigned char*,
»                            unsigned int, unsigned int,
»                            unsigned char, unsigned int);
»
»int get_cells_values(unsigned int N, unsigned char *a,
»const char* input);
»unsigned char* start(unsigned char regla, unsigned int N,
»const char* filename);
»int build_pbm();
```

»6.3. autcel.c

```
»#include "autcel.h"
»#include "file_reader.h"
»#define N_LIM 1000 // 1 mb
»#define MAX_FILE_LENGTH 40
»
»int get_cells_values(unsigned int N, unsigned char *a,
»const char* input){
»    unsigned char bit;
»    unsigned int tope_a = 0;
```

```
»
»    file_reader_t file;
»
»    if (file_reader_init(&file, input) == FR_ERROR)
»        return FR_ERROR;
»    while(((bit = file_reader_number(&file)) != FR_ERROR)){
»        a[tope_a++] = bit;
»    }
»
»    if (tope_a != N) return -1;
»    file_reader_uninit(&file);
»
»    return 0;
»}
»
»unsigned char* start(unsigned char regla, unsigned int N,
»const char* filename){
»  printf("%s\n","Leyendo estado inicial...");
»  if (N > N_LIM){
»    fprintf(stderr, "%s", "N exceded max limit\n");
»    return NULL;
»  }
»
»  unsigned int array_size = N * N * sizeof(unsigned char);
»  unsigned char* a = malloc(array_size);
»  if(!a){
»    fprintf(stderr, "%s", "Could not allocate memory\n");
»    return NULL;
»  }
»
»  if(get_cells_values(N, a, filename) != 0) {
»      fprintf(stderr, "%s", "Bad file format\n");
»      free(a);
»      return NULL;
»  }
»
»  for (size_t i = 0; i < N*(N-1); ++i){
»       unsigned char caracter = proximo
»       (a, i/N, i % N, regla, N);
»       a[i + N] = caracter;
»  }
»  return a;
»}
»
»int build_pbm(const char* outputprefix,
»              unsigned char* a, unsigned int N){
```

```
»  //Guardo el arreglo en una imagen pbm
»  printf("%d\n", N);
»  char outformat[5] = ".pbm";
»  char filename[MAX_FILE_LENGTH] = { 0 };
»
»  strncpy(filename, outputprefix, MAX_FILE_LENGTH);
»  strncat(filename, outformat, 5);
»  printf("Grabando %s\n", filename);
»
»  size_t i, temp = 0;
»  FILE* pbmimg;
»  pbmimg = fopen(filename, "wb");
»  fprintf(pbmimg, "P1\n");
»
»  fprintf(pbmimg, "%d %d\n", N, N);
»
»  for (i = 0; i < N * N; i++) {
»      temp = a[i];
»      fprintf(pbmimg, "%d ", (int)temp);
»      if ((i+1)%N==0) fprintf(pbmimg, "\n");
»  }
»
»  fclose(pbmimg);
»  printf("Listo\n");
»
»  free(a);
»  return 0;
»}
```

»6.4. proximo.c

```
»// pensado como un arreglo donde cada char es una sola celda
»#include <stdio.h>
»
»unsigned char proximo(unsigned char* a,
»                      unsigned int i, unsigned int j,
»                      unsigned char regla, unsigned int N){
»    size_t bytes = i * N;
»    char c = a[j + bytes];
»    char l, r;
»
»    if (!j){
»        l = a[j + bytes + N - 1];
»        r = a[j + bytes + 1];
»    } else if (j == N - 1){
»        l = a[j + bytes - 1];
```

```
»         r = a[j + bytes - N + 1];
»     } else {
»         l = a[j + bytes - 1];
»         r = a[j + bytes + 1];
»     }
»
»     l <<= 2;
»     c <<= 1;
»     char to_shift = l | c | r;
»     char mask = 0x1 << to_shift;
»     return (regla & mask) ? 1 : 0;
»}
```

»6.5. proximo.s

```
»#include <sys/regdef.h>
».eqv FRAME_SIZE, 32
».eqv O_a, 32
».eqv O_i, 36
».eqv O_j, 40
».eqv O_regla, 44
».eqv O_N, 48
».eqv GP, 28
».eqv FP, 24
».eqv CELL, 20
».eqv C, 16
».eqv L, 12
».eqv R, 8
».eqv TO_SHIFT, 4
».eqv MASK, 0
»
».text
».align 2
»/*
»unsigned char proximo(unsigned char* a,
»                      unsigned int i, unsigned int j
»                      unsigned char regla, unsigned int N);
»*/
»
».ent    proximo
».globl  proximo
»
»proximo:
»    subu    $sp, $sp, FRAME_SIZE
»    sw      $gp, GP($sp)
»    sw      $fp, FP($sp)
```

```
»
»     move    $fp, $sp
»
»     sw      $a0, 0_a($fp) # matriz a
»     sw      $a1, 0_i($fp) # fila i
»     sw      $a2, 0_j($fp) # col j
»     sb      $a3, 0_regla($fp) # regla -> 1 byte
»     lw      $t0, 0_N($fp) # N esta guardado en el aba del caller
»
»     multu   $zero, $zero  # nukeamos lo y hi
»     multu   $a1, $t0      # i * N
»     mflo    $t1           # low en t1
»     mfhi    $t2           # hi en t2
»
»     bne     $t2, $zero, overflow  # checkeamos qeu la
»     multiplicacion no haya dado of
»
»     addu    $t1, $t1, $a2 # j + i * N -> lo guardamos en CELL
»     sw      $t1, CELL($fp) # nos guardamos la variable -> indexa
»     en el array en c
»
»     addu    $t2, $a0, $t1 # &a[j + i * N]
»     lbu     $t3, 0($t2)   # t3 = a[j + i * N] loadeo un byte
»     sin signo (unsigned char)
»     sb      $t3, C($fp)   # c = a[j + i * N]  guardo un byte
»
»     subu    $t3, $t0, 1   # t3 = N-1
»
»     beq     $a2, $zero, first_cell  # if(!j)
»     nop
»
»     beq     $a2, $t3, last_cell
»     nop
»
»     b       no_border
»
»
»# if(!j)
»first_cell:
»     addu    $t3, $t1, $t3  # t3 = j + i * N + N - 1
»     addu    $t3, $t3, $a0  # t3 = &a[j + i * N + N - 1]
»     lbu     $t4, 0($t3)    # t4 = a[j + i * N + N - 1]
»     sb      $t4, L($fp)    # l = a[j + i * N + N - 1]
»
»     addiu   $t3, $t1, 1    # t3 = j + i * N + 1
»     addu    $t3, $t3, $a0   # t3 = &a[j + i * N + 1]
```

```
»    lbu     $t4, 0($t3)
»    sb      $t4, R($fp)
»
»    b       shift_seg
»
»# else if(j == N - 1)
»last_cell:
»    subu    $t2, $t1, 1   # t2 = j + i * N - 1
»
»    addu    $t4, $a0, $t2 # t4 = &a[j + i * N - 1]
»    lbu     $t5, 0($t4)   # t5 = a[j + i * N - 1]
»    sb      $t5, L($fp)   # l = a[j + i * N - 1]
»
»    negu    $t3           # t3 = -N + 1
»    addu    $t1, $t1, $t3 # t1 = j + i * N - N + 1
»    addu    $t1, $t1, $a0 # t1 = &a[j + i * N - N + 1]
»    lbu     $t4, 0($t1)   # t4 = a[j + i * N - N + 1]
»    sb      $t4, R($fp)
»
»    b       shift_seg
»
»# else
»no_border:
»    subu    $t1, $t1, 1   # t1 = j + i * N - 1
»
»    addu    $t3, $a0, $t1 # t3 = &a[j + i * N - 1]
»    lbu     $t4, 0($t3)   # t4 = a[j + i * N - 1]
»    sb      $t4, L($fp)   # l = a[j + i * N - 1]
»
»    addiu   $t1, $t1, 2   # t1 = j + i * N + 1
»    addu    $t3, $a0, $t1 # t3 = &a[j + i * N + 1]
»    lbu     $t4, 0($t3)   # t4 = a[j + i * N + 1]
»    sb      $t4, R($fp)   # r = a[j + i * N - 1]
»
»    b       shift_seg
»
»
»shift_seg:
»    lbu     $t0, L($fp)
»    lbu     $t1, C($fp)
»    lbu     $t2, R($fp)
»    sll     $t0, $t0, 2   # t0 = l << 2
»    sll     $t1, $t1, 1   # t1 = c << 1
»    or      $t3, $t0, $t1 # t3 = l | c
»    or      $t3, $t3, $t2 # t3 = l | c | r
»    sb      $t3, TO_SHIFT($fp) # to_shift = l | c | r
```

```
»    li      $t4, 1
»    sllv    $t4, $t4, $t3 # t4 = 0x1 << to_shift
»    sb      $t4, MASK($fp) # mask = 0x1 << to_shift
»    b       and_seg
»
»
»and_seg:
»    lbu     $t0, 0_regla($fp)   # t0 = regla
»    lbu     $t1, MASK($fp)      # t1 = mask
»    and     $t2, $t0, $t1
»    sltu    $v0, $zero, $t2     # v0 == 1 si
»    0 < regla & mask sino 0
»    b       proximo_end
»
»proximo_end:
»    lw      $fp, FP($sp)
»    lw      $gp, GP($sp)
»    addiu   $sp, $sp, FRAME_SIZE
»    jr      $ra
»
»overflow:
»    add $v0, $0, $0
»    b proximo_end
».end proximo
```

»6.6. file_reader.h

```
»#ifndef FILE_READER_H
»#define FILE_READER_H
»
»#include <stdio.h>
»#include <stdbool.h>
»#include <stdlib.h>
»
»#define FR_ERROR 5
»
»typedef struct file_reader{
»        FILE* file;
»} file_reader_t;
»
»size_t file_reader_init(file_reader_t* self, const char*
»file_name);
»
»size_t file_reader_uninit(file_reader_t* self);
»
»// Lee del archivo correspondiente un bit. Devuelve FR_ERROR al
```

```
»terminar.
»unsigned char file_reader_number(file_reader_t* self);
»
»#endif
```

»6.7. file_reader.c

```
»#include "file_reader.h"
»
»#define SUCCESS 0
»#define OPEN_MODE "r"
»
»size_t file_reader_init(file_reader_t* self, const char*
»file_name){
»       self->file = file_name != NULL ? fopen
»       (file_name,OPEN_MODE) : stdout;
»
»       return self->file == NULL ? FR_ERROR : SUCCESS;
»}
»
»size_t file_reader_uninit(file_reader_t* self){
»       size_t result = 0;
»
»       if(self->file != stdout)
»               result = (size_t) fclose(self->file);
»
»       return result == SUCCESS ? SUCCESS : FR_ERROR;
»}
»unsigned char file_reader_number(file_reader_t* self){
»       unsigned char bit;
»       int read_bytes = fread(&bit, sizeof(unsigned char), 1,
»       self->file);
»       return ((read_bytes == 1) && ((bit == '0')
»       || (bit == '1'))) ? bit - '0' : FR_ERROR;
»}
```