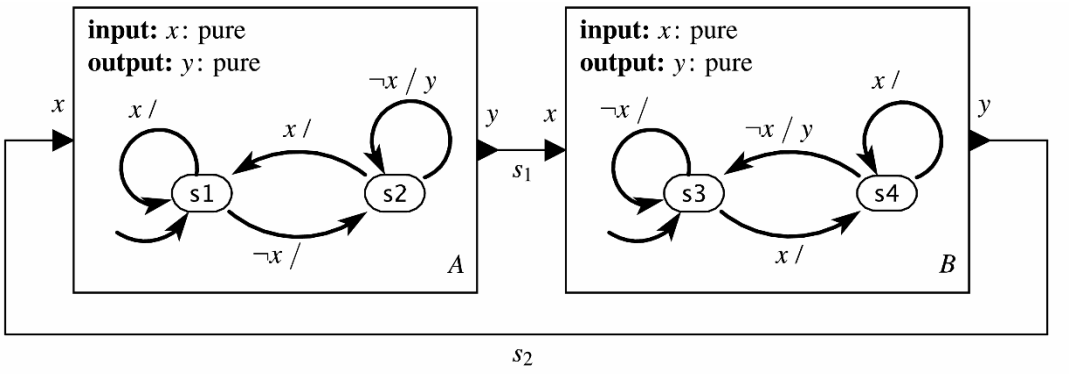
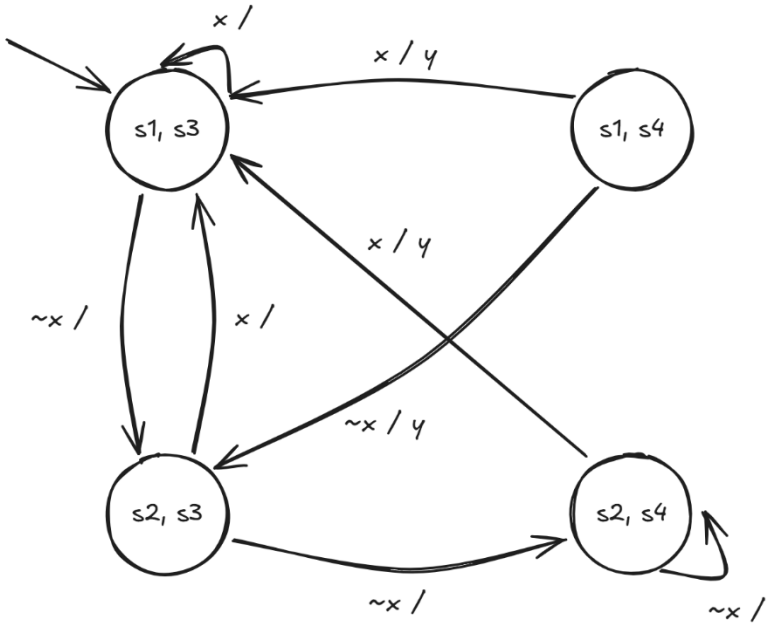
گزارش تمرین ۳

فهرست

بخش ۱:



ماشین حالت ترکیبی معادل آن چیست؟



آیا خوش‌ساخت است؟

برای تشخیص well-fromed یا ill-formed بودن، قدم به قدم حالت‌های ماشین ترکیبی بالا را بررسی می‌کنیم. باید توجه داشته باشیم که تنها حالت‌های دست‌یافتنی[1] برای ما ملاک هستند.

- **حالت s1, s3:** از این حالت دو گذار[2] وجود دارد. در هر دوی این گذارها سیگنال خروجی مدل absent است. پس می‌توان گفت برای این حالت: $s(n) = absent$

[1] Reachable state
[2] Transition

- **حالت s2, s3:** از این حالت دو گذار وجود دارد که در هر دوی این گذارها سیگنال خروجی مدل absent است. پس می‌توان گفت برای این حالت: $s(n) = absent$
- **حالت s2, s4:** از این حالت دو گذار وجود دارد. برای یکی که روی absent بودن سیگنال ورودی، سیگنال خروجی absent تولید می‌کند، می‌توان گفت: $s(n) = absent$ و برای دیگری که روی present بودن سیگنال ورودی، سیگنال خروجی present تولید می‌کند، می‌توان گفت: $s(n) = present$

برای خوش‌ساخت بودن باید تنها یک fixed point وجود داشته باشد. برای این حالت دو fixed point وجود دارد در نتیجه این مدل well-formed نیست.

- **حالت s1, s4:** این حالت دست‌یافتنی نیست.

دنباله مقادیر سیگنال‌های s1 و s2 چیست؟

برای بدست آوردن مقادیر این دو سیگنال به روش جستجوی کامل[3] عمل می‌کنیم.

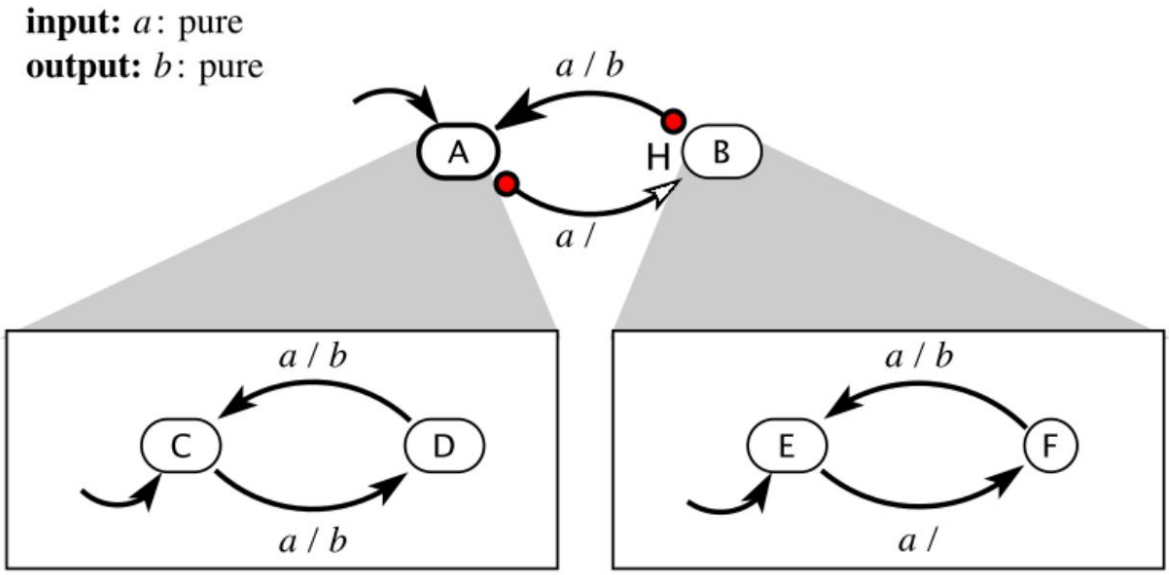
ابتدا در حالت s1 هستیم و تمام سیگنال‌ها unknown هستند	$s1 = \{\ \}$ $s2 = \{\ \}$
می‌توانیم از این فرض که خروجی حالت s1 همیشه absent است استفاده کنیم $s(n) = absent$	$s1 = \{absent\}$ $s2 = \{\ \}$
در حالت s3 با ورودی absent، خروجی absent تولید می‌شود	$s1 = \{absent\}$ $s2 = \{absent\}$
در حالت s2 هستیم و ورودی absent است. پس در حالت s2 می‌مانیم و خروجی present است	$s1 = \{absent, present\}$ $s2 = \{absent\}$
در حالت s3 هستیم و ورودی present است. پس به حالت s4 می‌رویم و خروجی absent است	$s1 = \{absent, present\}$ $s2 = \{absent, absent\}$
در حالت s2 هستیم و ورودی absent است. پس در حالت s2 می‌مانیم و خروجی present است	$s1 = \{absent, present, \text{present}\}$ $s2 = \{absent, absent\}$
در حالت s4 هستیم و ورودی present است. پس در حالت s4 می‌مانیم و خروجی absent است	$s1 = \{absent, present, \text{present}\}$ $s2 = \{absent, absent, absent\}$

[3] Exhaustive search

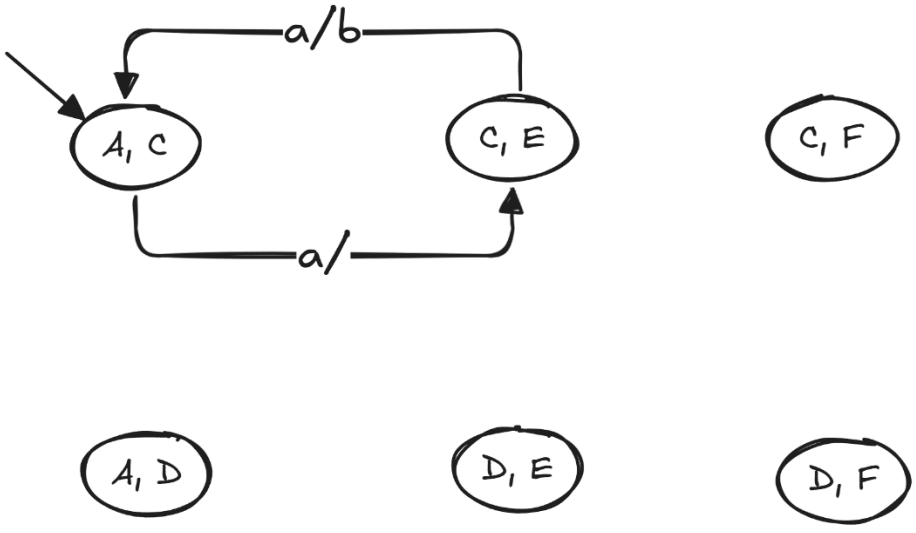
در حالت s2 هستیم و ورودی absent است. پس در حالت s2 می‌مانیم و خروجی present است	$s1 = \{absent, present, \text{present}, present\}$ $s2 = \{absent, absent, absent\}$
در حالت s4 هستیم و ورودی present است. پس در حالت s4 می‌مانیم و خروجی absent است	$s1 = \{absent, present, \text{present}, present\}$ $s2 = \{absent, absent, absent, absent\}$

این حلقه تا بی‌نهایت ادامه پیدا می‌کند. خروجی s1 همواره present و s2 همواره absent خواهد بود.

بخش ۲:



یک FSM مسطح معادل رفتار آن ارائه کنید.



گذار A به B یک گذار ریست است و در هربار ورود به وضعیت B از وضعیت E شروع می‌کند. اما گذار B به A یک گذار تاریخچه‌دار است در هربار ورود به وضعیت A از وضعیت پیشین ادامه می‌دهد. پس به ازای هر وضعیتی که در A نیست ترکیبی از آن‌ها با وضعیت‌های C و D موجود اند. **مجموعا ۴ وضعیت**.

این ماشین چه حالت‌های دسترس پذیری دارد؟

از این ۶ وضعیت تنها ۲ وضعیت قابل دسترس اند. این اتفاق ناشی از preemptive بودن گذار های میان A و B است.

- اگر در وضعیت C باشیم و ورودی a فعال شود، گادر گذار A به B درست است و بدون آپدیت ریفاینمنت وضعیت A به وضعیت B می‌رود.
- اگر در وضعیت E باشیم و ورودی a فعال شود، گادر گذار B به A درست است و بدون آپدیت ریفاینمنت وضعیت B به وضعیت A می‌رود.

$$C \xrightarrow{a/} E \xrightarrow{a/b} C \xrightarrow{a/} \dots$$

رفتار ورودی/خروجی این ماشین را در یک جمله توصیف کنید.

به ازای هر دو رخداد a یک رخداد b رخ می‌دهد.

بخش ۳:

استخراج ماتریس وقوع

ستون‌های این ماتریس بیانگر actor و سطرهای آن کانال‌هایی است که به actorها متصل است. مقادیر داخل این ماتریس نشان می‌دهد که actor مورد نظر روی یک کانال خاص چه تعداد توکن تولید یا مصرف می‌کند.

$$\Gamma = \begin{bmatrix} -3 & 2 & 0 & 0 \\ 0 & 1 & -3 & 0 \\ 0 & 0 & 4 & -1 \\ 0 & -4 & 0 & 3 \end{bmatrix}$$

برای یافتن مرتبه این ماتریس، ابتدا آن را به فرم پلکانی در می‌آوریم.

۱. سطر آخر را با ۴ برابر سطر دوم جمع می‌کنیم.
۲. سطر آخر را با ۳ برابر سطر سوم جمع می‌کنیم.

$$\Gamma = \begin{bmatrix} -3 & 2 & 0 & 0 \\ 0 & 1 & -3 & 0 \\ 0 & 0 & 4 & -1 \\ 0 & -4 & 0 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} -3 & 2 & 0 & 0 \\ 0 & 1 & -3 & 0 \\ 0 & 0 & 4 & -1 \\ 0 & 0 & -12 & 3 \end{bmatrix} \rightarrow \begin{bmatrix} -3 & 2 & 0 & 0 \\ 0 & 1 & -3 & 0 \\ 0 & 0 & 4 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

بلافاصله مشاهده می‌شود که سطر آخر ماتریس به طور کامل صفر می‌شود. پس مرتبه این ماتریس برابر **۳** است.

بدست آوردن بردار آتش

برای یافتن مقدار q دستگاه معادلات زیر را باید حل کنیم.

$$\begin{bmatrix} -3 & 2 & 0 & 0 \\ 0 & 1 & -3 & 0 \\ 0 & 0 & 4 & -1 \\ 0 & -4 & 0 & 3 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

از بخش قبلی ساده شده ماتریس ضرایب را بدست آورده بودیم. ماتریس ضرایب ساده شده را جایگذاری می‌کنیم:

$$\begin{bmatrix} -3 & 2 & 0 & 0 \\ 0 & 1 & -3 & 0 \\ 0 & 0 & 4 & -1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

سپس معادلات را استخراج می‌کنیم:

$$3q_1 = 2q_2 \rightarrow q_1 = \frac{q_4}{2}$$

$$3q_3 = q_2 = \frac{3q_4}{4}$$

$$4q_3 = q_4 \rightarrow q_3 = \frac{q_4}{4}$$

کمترین عدد صحیحی که می‌توان برای q_4 در نظر گرفت تا مقادیر q_1 تا q_3 هم عدد صحیح شوند، ۴ است. پس بدین ترتیب مقادیر زیر بدست می‌آیند:

$$q_1 = 2$$

$$q_2 = 3$$

$$q_3 = 1$$

$$q_3 = 4$$

$$\mathrm{q} = \begin{bmatrix} 2 \\ 3 \\ 1 \\ 4 \end{bmatrix}$$

زمانبندی

پس از یافتن firing vector باید به صورت ترتیبی هر actor را اجرا کنیم به طوری که بافر منفی نشود و کمترین مقدار توکن ممکن داخل یک بافر قرار بگیرد.

بردار b بیانگر تعداد initial token است:

- کانال ۱: a_1 به a_0 = ۴ توکن
- کانال ۲: a_1 به a_2 = ۲ توکن
- کانال ۳: a_2 به a_3 = ۰ توکن
- کانال ۴: a_3 به a_1 = ۴ توکن

$$\mathrm{b} = \begin{bmatrix} 4 \\ 2 \\ 0 \\ 4 \end{bmatrix}$$

حال با داشتن بردار تعداد توکن اولیه و بردار آتش، قدم به قدم اجرا می‌کنیم تا یک زمانبندی بدست بیاید.

پیمایش	actor اجرا شده	بردار آتش	بردار توکن	تغییرات تعداد توکن			
				کانال۱	کانال۲	کانال۳	کانال۴
۰		$\begin{bmatrix}2\\3\\1\\4\end{bmatrix}$	$\begin{bmatrix}4\\2\\0\\4\end{bmatrix}$				
۱	a_0	$\begin{bmatrix}1\\3\\1\\4\end{bmatrix}$	$\begin{bmatrix}1\\2\\0\\4\end{bmatrix}$	-3			
۲	a_1	$\begin{bmatrix}1\\2\\1\\4\end{bmatrix}$	$\begin{bmatrix}3\\3\\0\\0\end{bmatrix}$	+2	+1		-4
۳	a_0	$\begin{bmatrix}0\\2\\1\\4\end{bmatrix}$	$\begin{bmatrix}0\\3\\0\\0\end{bmatrix}$	-3			
۴	a_2	$\begin{bmatrix}0\\2\\0\\4\end{bmatrix}$	$\begin{bmatrix}0\\0\\4\\0\end{bmatrix}$		-3	+4	
۵	a_3	$\begin{bmatrix}0\\2\\0\\3\end{bmatrix}$	$\begin{bmatrix}0\\0\\3\\3\end{bmatrix}$			-1	+3
۶	a_3	$\begin{bmatrix}0\\2\\0\\2\end{bmatrix}$	$\begin{bmatrix}0\\0\\2\\6\end{bmatrix}$			-1	+3
۷	a_2	$\begin{bmatrix}0\\1\\0\\2\end{bmatrix}$	$\begin{bmatrix}2\\1\\2\\2\end{bmatrix}$	+2	+1		-4
۸	a_3	$\begin{bmatrix}0\\1\\0\\1\end{bmatrix}$	$\begin{bmatrix}2\\1\\1\\5\end{bmatrix}$			-1	+3

۹	a_1	$\begin{bmatrix}0\\0\\0\\1\end{bmatrix}$	$\begin{bmatrix}4\\2\\1\\1\end{bmatrix}$	+2	+1		-4
۱۰	a_3	$\begin{bmatrix}0\\0\\0\\0\end{bmatrix}$	$\begin{bmatrix}4\\2\\0\\4\end{bmatrix}$			-1	+3

ترتیب اجرا:

Sequence: $a_0, a_1, a_0, a_2, a_3, a_3, a_1, a_3, a_1, a_3$

حداقل حجم بافر

با داشتن روند تغییرات بردار b می‌توان حجم بافر مورد نیاز را پیدا کرد. برای پیدا کردن حداقل حجم بافر هر کانال باید بیشترین توکنی که در آن کانال در طول اجرا قرار گرفته است را بدست بیاوریم. بیشترین توکن در هر کانال مطابق زیر است:

- کانال ۱: a_1 به a_0 =۴ توکن
- کانال ۲:a_1 به a_2 = ۳ توکن
- کانال ۳: a_2 به a_3 = ۴ توکن
- کانال ۴: a_3 به a_1 = ۶ توکن

کد C

```c
#include <stdio.h>
#include <stdbool.h>
#include <stdlib.h>

#define BUFFER1_SIZE 4 // a1 to a0
#define BUFFER2_SIZE 3 // a1 to a2
#define BUFFER3_SIZE 4 // a2 to a3
#define BUFFER4_SIZE 6 // a3 to a1


#define BUFFER1_INITIAL_TOKEN 4
#define BUFFER2_INITIAL_TOKEN 3
#define BUFFER3_INITIAL_TOKEN 4
#define BUFFER4_INITIAL_TOKEN 6
```

```c
typedef struct {
    int *buffer;
    int size;
    int head;
    int tail;
    int count;
} CircularBuffer;

void init_buffer(CircularBuffer *cb, int size) {
    cb->buffer = (int *)malloc(sizeof(int) * size);
    cb->size = size;
    cb->head = 0;
    cb->tail = 0;
    cb->count = 0;
}

void free_buffer(CircularBuffer *cb) {
    free(cb->buffer);
}

bool push_token(CircularBuffer *cb, int value) {
    if (cb->count < cb->size) {
        cb->buffer[cb->tail] = value;
        cb->tail = (cb->tail + 1) % cb->size;
        cb->count++;
        return true;
    } else {
        return false;
    }
}

int pop_token(CircularBuffer *cb) {
    if (cb->count > 0) {
        int value = cb->buffer[cb->head];
        cb->head = (cb->head + 1) % cb->size;
        cb->count--;
        return value;
    } else {
        return -1;
    }
}

void a0() {
    printf("a0 fired.\n");
}
```

```
void a1() {
    printf("a1 fired.\n");
}

void a2() {
    printf("a2 fired.\n");
}

void a3() {
    printf("a3 fired.\n");
}

int q[4] = {2, 3, 1, 4};

CircularBuffer buffer1, buffer2, buffer3, buffer4;

void fire_actor(int actor) {
    switch (actor) {
        case 0:
            if (q[0] > 0 && buffer1.count >= 3) {
                a0();
                for (int i = 0; i < 3; ++i) {
                    pop_token(&buffer1);
                }
                q[0]--;
            }
            break;
        case 1:
            if (q[1] > 0 && buffer4.count >= 4) {
                a1();
                for (int i = 0; i < 2; ++i) {
                    push_token(&buffer1, 1);
                }
                push_token(&buffer2, 1);
                for (int i = 0; i < 4; ++i) {
                    pop_token(&buffer4);
                }
                q[1]--;
            }
            break;
        case 2:
            if (q[2] > 0 && buffer2.count >= 3) {
                a2();
                for (int i = 0; i < 4; ++i) {
```

```
                    push_token(&buffer3, 1);
                }
                for (int i = 0; i < 3; ++i) {
                    pop_token(&buffer2);
                }
                q[2]--;
            }
            break;
        case 3:
            if (q[3] > 0 && buffer3.count >= 1) {
                a3();
                for (int i = 0; i < 3; ++i) {
                    push_token(&buffer4, 1);
                }
                pop_token(&buffer3);
                q[3]--;
            }
            break;
    }
}

int main() {
    init_buffer(&buffer1, BUFFER1_SIZE);
    init_buffer(&buffer2, BUFFER2_SIZE);
    init_buffer(&buffer3, BUFFER3_SIZE);
    init_buffer(&buffer4, BUFFER4_SIZE);

    buffer1.count = BUFFER1_INITIAL_TOKEN;
    buffer2.count = BUFFER2_INITIAL_TOKEN;
    buffer3.count = BUFFER3_INITIAL_TOKEN;
    buffer4.count = BUFFER4_INITIAL_TOKEN;

    int schedule[] = {0, 1, 0, 2, 3, 3, 1, 3, 1, 3};
    int schedule_length = 10;

    printf("Initial Firing vector and Buffer vector\n");
    printf("q: [%d %d %d %d]\n", q[0], q[1], q[2], q[3]);
    printf("b: [%d %d %d %d]\n", buffer1.count, buffer2.count, buffer3.count,
buffer4.count);

    for (int i = 0; i < schedule_length; ++i) {
        fire_actor(schedule[i]);
        printf("q: [%d %d %d %d]\n", q[0], q[1], q[2], q[3]);
        printf("b: [%d %d %d %d]\n", buffer1.count, buffer2.count, buffer3.count,
buffer4.count);
```

```
    }

    free_buffer(&buffer1);
    free_buffer(&buffer2);
    free_buffer(&buffer3);
    free_buffer(&buffer4);

    return 0;
}
```

خروجی:

```
Initial Firing vector and Buffer vector
q: [2 3 1 4]
b: [4 3 4 6]

a0 fired.
q: [1 3 1 4]
b: [1 3 4 6]

a1 fired.
q: [1 2 1 4]
b: [3 3 4 2]

a0 fired.
q: [0 2 1 4]
b: [0 3 4 2]

a2 fired.
q: [0 2 0 4]
b: [0 0 4 2]

a3 fired.
q: [0 2 0 3]
b: [0 0 3 5]

a3 fired.
q: [0 2 0 2]
b: [0 0 2 6]

a1 fired.
q: [0 1 0 2]
b: [2 1 2 2]

a3 fired.
q: [0 1 0 1]
b: [2 1 1 5]

a1 fired.
q: [0 0 0 1]
b: [4 2 1 1]
```

```
a3 fired.
q: [0 0 0 0]
b: [4 2 0 4]
```