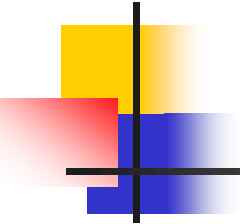
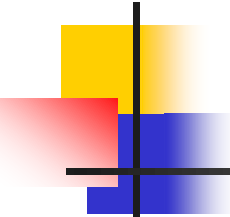


Programski prevodioci 1

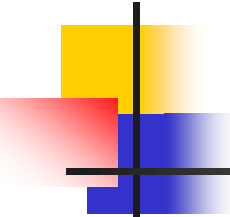
Lekcija 3 – Konačni procesori

- 
-
- Prepoznavaci i procesori
 - Implementacija konačnih automata



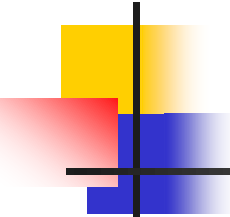
Konačni procesori

- Prepoznavač : binarna odluka – ulazna sekvenca se prihvata ili odbija
- Procesor – pored prepoznavanja ulazne sekvence vrši i neku dodatnu obradu kao bočni efekat,
- Primeri dodatne obrade u kontekstu prevođenja programa:
 - Izračunavanje binarne vrednosti konstante
 - Smeštanje simbola u tabelu simbola



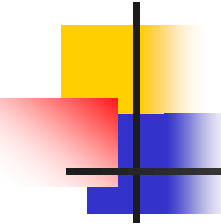
Konstrukcija konačnog procesora

- Prepoznavać pretvaramo u procesor na taj način što svakom (nepraznom) ulazu u tabeli prelaza pridružujemo AKCIJU (proizvoljnu sekvencu programskog koda).
- Ulaznu azbuku automata možemo proširiti simbolom kraja ulazne sekvence (end marker) —| koji se javlja jednom u ulaznoj sekvenci, uvek i isključivo na njenom kraju. Praktični razlozi.
- Praznim ulazima, po potrebi, možemo pridružiti akcije za detekciju i oporavak od leksičkih grešaka.



Primer

- Treba konstruisati deterministički procesor za izračunavanje vrednosti neoznačenih realnih konstanti.
- Primeri dozvoljenih konstanti su:
1257,.0392, 1E + 534, 1., 1.E5, 2.0E10
- a nedozvoljenih -21, .E3, E, E10, 12E.



Primer...

- Procesor izračunava binarnu vrednost konstante postavljajući sadržaje sledeća četiri registra:
- NR (number register) sadrži binarnu vrednost celog broja koji se dobija uklanjanjem decimalne tačke iz mantise;
- ER (exponent register) sadrži binarnu vrednost eksponenta koji je naveden u konstanti (ne vrši se normalizacija mantise);
- CR (count register) sadrži broj cifara iza decimalne tačke;
- SR (sign register) pamti predznak eksponenta (+1 ili -1)
 - Na primer, za konstantu 12.3E4 registri će imati sledeće vrednosti:

NR	ER	CR	SR
123	4	1	+1

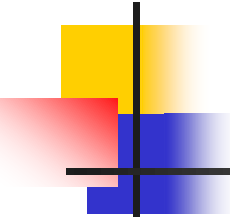
1. korak – Konstruisanje prepoznavaća

- Ulazna azbuka {DIGIT, •, E, SIGN}
- prelazi razmatranjem karakterističnih primera ulaza

3 8 . 7 E - 3
0 1 1 2 3 4 5 6

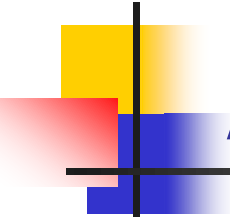
. 9 E 2 1
0 7 3 4 6 6

		DIGIT	E	•	SIGN	
inicijalno stanje	0	1		7		0
cifra pre opcione tačke	1	1	4	2		1
opciona decimalna tačka	2	3	4			1
cifra nakon decimalne tačke	3	3	4			1
slovo E	4	6			5	0
predznak eksponenta	5	6				0
cifra eksponenta	6	6				1
tačka iza koje mora ići cifra	7	3				0



2. korak - dodavanje akcija

	DIGIT	E	•	SIGN	—
0	1a		7		
1	1b	4a	23c		YES1
23	23a	4b			YES2
4	6a			5	
5	6b				
6	6c				YES3
7	23b				



Akcije imaju sledeći izgled

- Operator $\text{val}(\text{cifra})$ daje binarnu vrednost cifre

1a: $\text{NR} := \text{val}(\text{DIGIT})$

1b: $\text{NR} := \text{NR} * 10 + \text{val}(\text{DIGIT})$

23a: $\text{CR} := \text{CR} + 1$

$\text{NR} := \text{NR} * 10 + \text{val}(\text{DIGIT})$

23b: $\text{CR} := 1$

$\text{NR} := \text{val}(\text{DIGIT})$

23c: $\text{CR} := 0$

7: _____

YES1: $\text{ER} := 0$

$\text{CR} := 0$

4a: $\text{CR} := 0$

4b: _____

5: $\text{SR} := \text{val}(\text{SIGN})$

6a: $\text{SR} := +1$

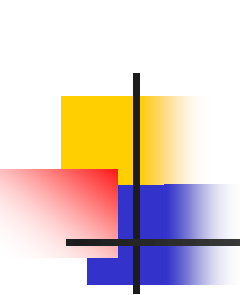
$\text{ER} := \text{val}(\text{DIGIT})$

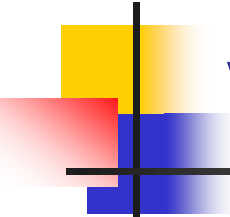
6b: $\text{ER} := \text{val}(\text{DIGIT})$

6c: $\text{ER} := \text{ER} * 10 + \text{val}(\text{DIGIT})$

YES2: $\text{ER} := 0$

YES3: _____

- 
-
- Prepoznavaci i procesori
 - Implementacija konačnih automata



Varijante implementacije KA

- Pod **implicitnom predstavom funkcije prelaza** podrazumeva se da je stanje određeno mestom u programu koji se izvršava.
- U **eksplicitnoj predstavi**, postoji matrica koja pamti funkciju prelaza pri radu automata.
- **Objektno-orijentisana** implementacija – projektni uzorak **State**.

Primer

- Želi se realizovati na C-u konačni automat sa slike koristeći **implicitno** i **eksplicitno** predstavljanje funkcije prelaza.

	a	b	
→A	B		1
B	A	B	0

- Rešenje:
eksplicitno FSM_TT.java
implicitno FSM.java

```

class FSM_TT
{

enum State {
    A( true ),
    B( false ),
    ERROR( false );

    final boolean accepting;

    State( boolean accepting ) {
        this.accepting = accepting;
    }
}

static State state;
static State transition[][] =
{
    { State.B, State.ERROR},
    { State.A, State.B},
    { State.ERROR, State.ERROR}
};

```

```

static void handleInput(char ch) {
    int index;
    switch( ch ) {
        case 'a':
            index=0; break;
        case 'b':
            index=1; break;
        default:
            state=State.ERROR; return;
    }
    state = transition[state.ordinal()][index];
}

public static void main(String[] args)
    throws java.io.IOException {
    int ch;
    state = State.A;
    while ((ch = System.in.read ()) != '\n')
        handleInput((char)ch);
    if (state.accepting)
        System.out.println("Prihvata");
    else
        System.out.println("Odbija");
}
} // class FSM_TT

```

```

class FSM
{
    enum State {
        A( true ),
        B( false ),
        ERROR( false );

        final boolean accepting;

        State( boolean accepting ) {
            this.accepting = accepting;
        }
    }

    public static void main(String[] args)
        throws java.io.IOException {
        int ch;
        state = State.A;
        while ((ch = System.in.read ()) != '\n')
            handleInput((char)ch);
        if (state.accepting)
            System.out.println("Prihvata");
        else
            System.out.println("Odbija");
    } //main

```

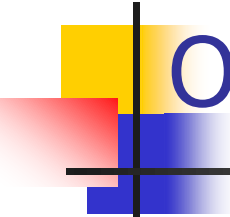
```

static void handleInput(char ch){
    switch( state ) {
        case A:
            if ( ch=='a' )
                state = State.B;
            else
                state = State.ERROR;
            break;

        case B:
            if ( ch=='a' )
                state = State.A;
            else if ( ch=='b' )
                state = State.B;
            else
                state = State.ERROR;
            break;

        case ERROR:
            //state = State.ERROR;
            break;
    } //switch
} //handleInput
} //class FSM

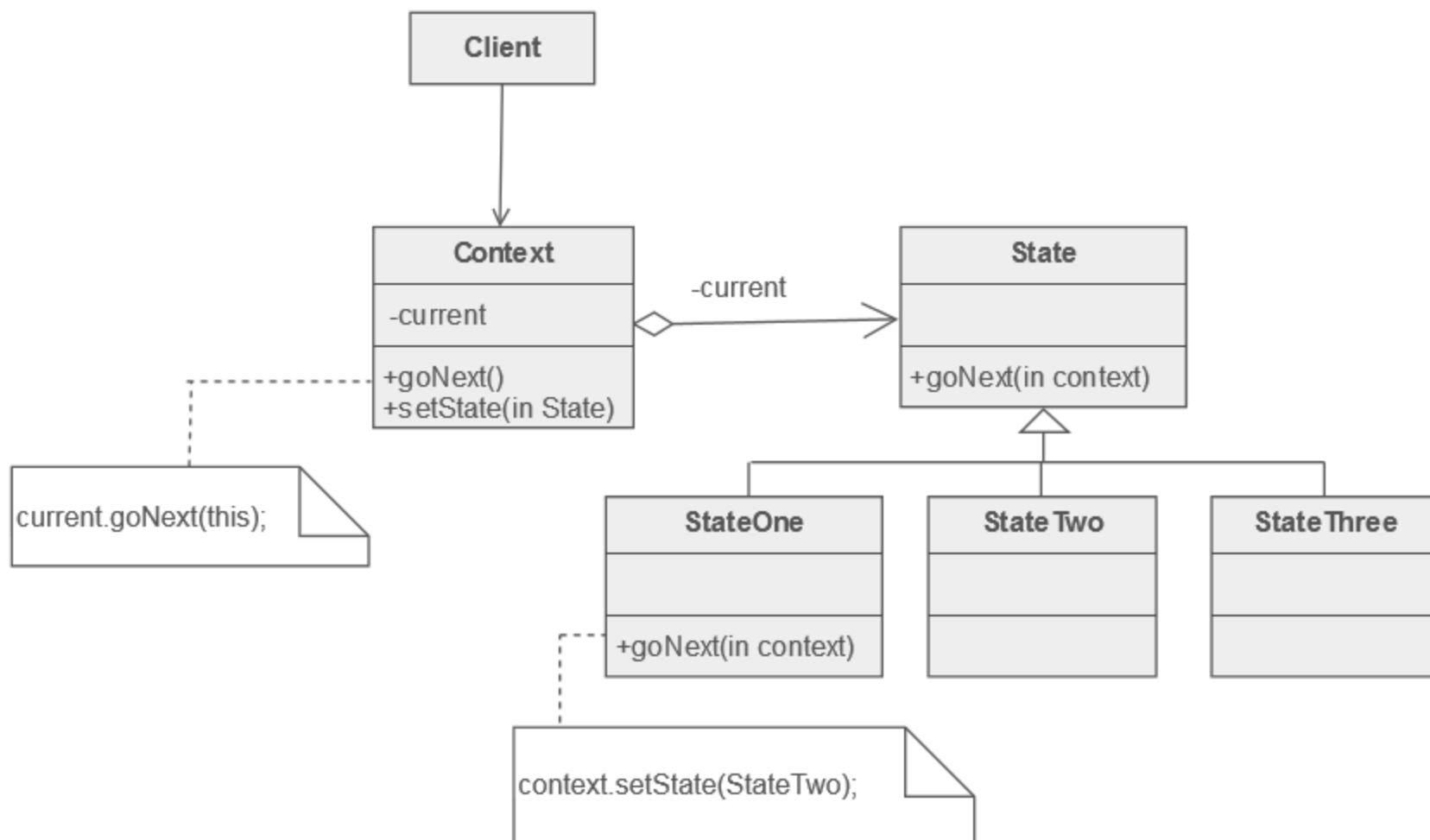
```



Objektno-orijentisana implementacija KA

- Želimo implementirati zadati automat koristeći projektni uzorak **State** iz knjige Gama,... "Design Patterns/Elements of Reusable Object-Oriented Programming Style".

Projektni uzorak Stanje



Rešenje: FSM_OO.java

```

class FSM_OO // "Context"
{
    State A = new StateA();
    State B = new StateB();
    State ERROR = new StateERROR();

    interface State {
        boolean accepting();
        void handleInput(FSM_OO fsm, char ch); // "goNext"
    };

```

```

class StateA implements State
{
    public boolean accepting() { return true;}

    public void handleInput(FSM_OO fsm, char ch){ // "goNext"
        if ( ch=='a' )
            fsm.setState(B);
        else
            fsm.setState(ERROR);
    }
};

```

```

class StateB implements State
{
    public boolean accepting() { return false;}
    public void handleInput(FSM_OO fsm, char ch) // "goNext"
    {
        if ( ch=='a' )
            fsm.setState(A);
        else if ( ch=='b' )
            fsm.setState(B);
        else
            fsm.setState(ERROR);
    }
};

```

```

class StateERROR implements State
{
    public boolean accepting() { return false;}
    public void handleInput(FSM_OO fsm, char ch) // "goNext"
    {
        fsm.setState(ERROR);
    }
};

```

```

class FSM_OO {
    ...
    private State state; // "current"
    public void setState(State state) {
        this.state = state;
    }
    public State getState() {
        return state;
    }
    void handleInput(char ch) { // "goNext"
        state.handleInput(this, ch);
    }
    public static void main(String[] args) throws java.io.IOException { // "client"
        int ch;
        FSM_OO fsm = new FSM_OO();
        fsm.setState(fsm.A);
        while ((ch = System.in.read ()) != '\n')
            fsm.handleInput((char)ch);

        if (fsm.getState().accepting())
            System.out.println("Prihvata");
        else
            System.out.println("Odbija");
    } //main
} //class FSM_OO

```