3 body

Kód studenta 11

4 OO návrh GUI frameworku (3 body)

Vaším úkolem je v jazyce C++, C# nebo Java navrhnout základ jednoduchého okenního frameworku, který bude moci být využit k tvorbě aplikací s GUI. Vaše implementace by měla vycházet z pravidel vhodného objektového návrhu tak, aby byla do budoucna snadno rozšiřitelná a udržovatelná – nicméně, soustřeďte se na požadované vlastnosti a návrh si zbytečně nekomplikujte.

1. Váš framework by měl splňovat následující základní vlastnosti:
   - Podporovat koncept okna (Window), které může obsahovat libovolné množství ovládacích prvků.
   - Každý ovládací prvek je v okně umístěn na nějaké souřadnici [X, Y] a má danou šířku a výšku (tyto 4 hodnoty seskupuje předpřipravená struktura `Rectangle`, která má navíc i metodu `bool IsPointInside(Point p)`, která testuje, zda se bod `p` nachází uvnitř obdélníku).
   - Připravte ovládací prvky tlačítko (Button) a uživatelem editovatelné textové pole (TextBox). Oba tyto prvky mají mít vlastnost `Text`, která reprezentuje text v nich zobrazený (jakým způsobem probíhá samotné zobrazení neřešte).
2. Každý ovládací prvek má metodu `void HandleMouseClick(Point p)`, která má být zavolána, pokud nad prvkem došlo ke kliknutí myši (metoda bude obsahovat logiku chování specifickou pro daný typ prvku; samotnou logiku ale neimplementujte; také pro jednoduchost předpokládejte, že se ovládací prvky nepřekrývají). Pro získání informace od operačního systému o tom, zda a na jakých souřadnicích došlo ke kliknutí myši, máte k dispozici hotovou třídu `OS` (vrácený bod je ve stejném souřadném systému jako souřadnice určující umístnění ovládacích prvků):

```
enum UserEvent { MouseClick, /* ... */ };
static class OS {
    public static UserEvent WaitForNextEvent() { /* ... */ }
    public static Point ReadLastMouseClick() { /* ... */ }
}
```

3. Framework musí umožňovat aplikacím potřebným způsobem reagovat na stisknutí tlačítka (detekované logikou tlačítka např. v jeho metodě `HandleMouseClick`), programátoři aplikací však nesmějí zasahovat do kódu frameworku. Aplikační kód obsluhující stisk tlačítka přitom musí být schopen komunikovat i s jinými ovládacími prvky daného okna, např. přečíst text z textového pole.

   Jako ilustraci použití vašeho frameworku napište klíčové součásti aplikace, která bude mít okno s jedním tlačítkem "Smazat" a jedním TextBoxem. Pokud uživatel do TextBoxu zadá jméno souboru a stiskne tlačítko "Smazat" (klikne na něj), tak aplikace daný soubor smaže pomocí metody `void Delete(string path)` na předpřipravené třídě `File`.

Návrh zpracován na separátním papíře, zde uvedu podotázku 3 na druhé straně →

```
void main(){
    Window w = new Window();
    Textbox t = new Textbox();
    Button b = new Button(() => {File.delete(t.text)});
    t.text = "napište jméno souboru"; { b.setLocation(...)
    w.start();                     { t.setLocation(...)
}
```

lze udělat přes t.locking kvůli closure

kód 11
otázka 4

```
class Window {
~~class~~ list<Control> controls;   ~~~~ rectangle window;
private ~~~~  Thread uithread; private control inFocus;
public addControl (Control c) { // kontrola pozice control
    controls.add(c);
}
                        public start { uithread.start()
                        }  // render everything

~~~~~~~~~~~~()
internal void handleevent (OS.userevent e) { Point r;
  ~~Point r = ~~ OS. if (e = Mouseclick) { r = OS.ReadLastMouseclick();
}                         c = list.firstordefault(x => x.rec.
                        }                          isPoint
 else { // call handle key press on focused control
  }}
abstract class Control {
    abstract void handlemouseclick (Point r); // mohlo by být
                                                virtual
    virtual void handlekeypress (keycode) { }
    ~~~~ Rectangle rectangle;

    public setLocation(Rectangle rectangle) {
    this.rectangle = rectangle;
    }

}
```

pozn: ~~~~ uithread bude čekat na userevents z OS a bude volat handle metody na classe window

(red annotations:) běžně spíš volající vlákno, ale OK ✓ (takhle to jde též :)) ✓ kód?

```
class button : Control {
  string Text { get; set; }  // + implementace zobrazení
  action a;
  void handleMouseClick(point x) {
      a.invoke();
  }

  // construction
  public button(action a) {
     this.a = a.
  }
  public setAction(action a) { this.a = a; }
}
class textBox : Control {
  string Text { get; set; }

  void handleMouseClick(Point x) {
    // show cursor
  }

  override void handleKeyPress(keycode) {
    // add to text / modify
  }
```

2

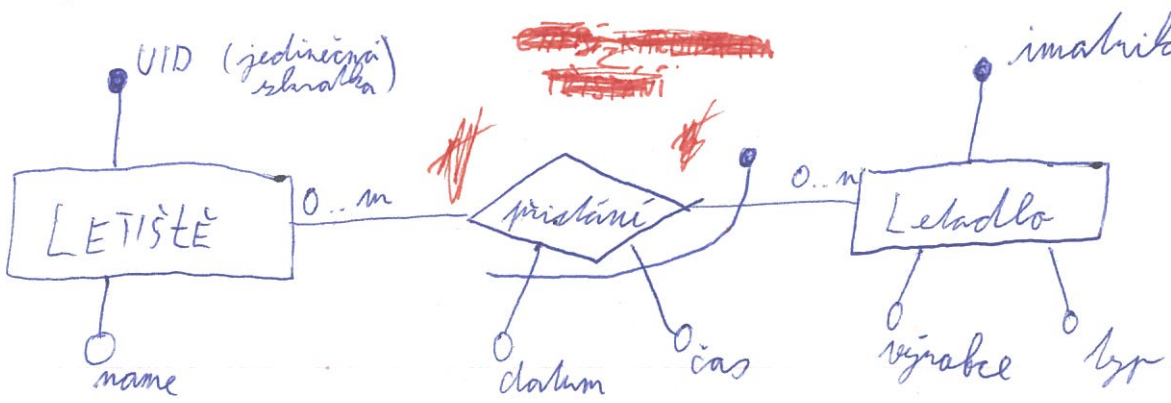
Kód studenta 11

3 ER diagramy a dotazy v SQL (3 body)

Navrhněte (velmi) zjednodušený systém evidence letadel na letištích. Pro každé letiště chceme evidovat jeho jméno a jedinečnou zkratku. Dále chceme evidovat data a časy přistání letadel, jejich výrobce, typ a imatrikulaci (jedinečný kód letadla). Neřešte mezní situace, jako například změnu imatrikulace letadla, změnu zkratky či jména letiště, či slučování výrobců letadel.

1. Pro daný příklad navrhněte ER diagram.
2. ER diagram převeďte na relační schéma (UML diagram tříd či SQL DDL) splňující 3NF. Vyznačte klíče a cizí klíče.
3. Napište dotaz v SQL, který pro dané letiště a den vypíše žebříček výrobců letadel dle počtu přistání jejich strojů.

1)



2)

```
table letiště {
    varchar UID primary key
    varchar name
}

table typletadlavyrobce {
    varchar typ primary key
    varchar vyrobce
}

table letadlo {
    varchar imatrikulace prim. key
    varchar typ foreign key(..)
}

table přistání {
    varchar imatrikulace foreign key(letadlo)
    date čas, date den
    varchar UID foreign key(letiště)
}
```

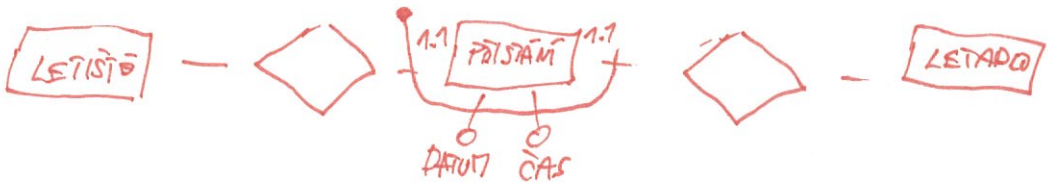
alter přistání primary key (imatrikulace, čas, den)

BEZPEČNĚJŠÍ BY BYLO MÍT ZDE I UID LETIŠTĚ
TATO SPRÁVNOST ZÁVISÍ NA REPREZENTACI
ČASOVÝCH ZÓN

3)

```
Select výrobce, count(výrobce)
from přistání p
join letadlo l on p.imatrikulace = l.imatrikulace
join typletadlavyrobce t on t.typ = l.typ
where den = :den and mid = :mid
group by výrobce
order by count(výrobce);
```

1) PŘISTÁNÍ MÁ BÝT SLABÝ ENTITNÍ TYP. V ŘEŠENÍ JE KOMBINACE SLABÉ IDENTITY SE VZTAHOVÝM TYPEM, CO2 V E-R NEEXISTUJE. VZTAH JE IDENTIFIKOVÁN ENTITAMI, KTERÉ V NĚM VYSTUPUJÍ. V TOMTO PŘÍPADĚ 'LETADLO' A 'LETIŠTĚ', CO2 NESTAČÍ. JE TŘEBA DIAGRAM ROZŠÍŘIT: A DOPLNIT KARDINALITY VZTAHŮ.



7

Kód studenta 11

5 Překlad (3 body)

Popišme si hypotetickou architekturu procesoru. Má tři speciální registry velikosti 32 bitů:
- **SP** (Stack Pointer) - ukazatel zásobníku, ukazuje na první volné místo (prázdný zásobník má SP=0)
- **PC** (Program Counter) - ukazatel následující zpracovávané instrukce
- **FP** (Frame Pointer) - ukazatel na záznam (stack frame) funkce, viz instrukce CALL pro přesné chování

Jedinou pamětí, se kterou tento procesor pracuje, je zásobník. Rozsah platných adres je [0, SP-1], jakýkoliv přístup mimo tento rozsah je chybný. K reprezentaci čísel používá procesor celočíselný znaménkový typ o velikosti 32 bitů reprezentovaný dvojkovým doplňkem.

Instrukce procesoru jsou uvedeny v následující tabulce. V popisu používáme tyto operace a operandy:
- *imm32* - celočíselná znaménková 32-bitová konstanta
- *label* - symbolické návěští do kódu
- *PUSH(X)* - uložení hodnoty X na zásobník
- *POP(N)* - odebrání hodnoty ze zásobníku, tato operace se provádí jako N-tá v pořadí
- * - dereference odkazu

instrukce	popis	operace
ADDI4	sečte dvě celá čísla z vrcholu zásobníku a výsledek uloží zpět na zásobník	PUSH(POP(1)+POP(2))
ADDSP imm32	přičte znaménkovou konstantu k registru SP	SP=SP+imm32
SHL imm32	posune vlevo o imm32 bitů 32-bitovou hodnotu z vrcholu zásobníku	PUSH(POP(1) SHL imm32)
LLD4 imm32	uloží načtenou 32-bitovou hodnotu ze záznamu funkce na vrchol zásobníku	PUSH(*(FP+imm32))
LST4 imm32	uloží 32-bitovou hodnotu z vrcholu zásobníku do záznamu funkce	*(FP+imm32)=POP(1)
JMP label	nepodmíněný skok	PC=label
JLEI4 label	podmíněný skok, který se provede, pokud je vrchní hodnota zásobníku menší nebo rovna než hodnota pod ní	if POP(1)<=POP(2) then PC=label
LIC4 imm32	uloží 32-bitovou znaménkovou konstantu na zásobník	PUSH(imm32)
RET	návrat z funkce	FP=POP(1), PC=POP(2)
CALL label	volání funkce	PUSH(PC), PUSH(FP), FP=SP

Uvažujme dále překladač z jazyka Pascal. Typ INTEGER odpovídá 32-bitovému celočíselnému typu. Klíčové slovo VAR znamená předání parametru odkazem. Předání pole odkazem je řešeno předáním odkazu na první položku pole. Jako příklad vezměme následující proceduru:

```
PROCEDURE incvar(VAR a:INTEGER);
BEGIN
  a := a + 1;
END;
```

Tu by překladač mohl přeložit například takto:

```
LLD4    -12     ; read A reference
LLD4            ; read A
LIC4    1       ; const 1
ADDI4           ; A + 1
LLD4    -12     ; read A reference
LST4            ; store A + 1
RET
```

1. Přeložte následující proceduru v Pascalu do instrukcí našeho CPU.

```
PROCEDURE addvar(a:INTEGER; b:INTEGER; VAR c:INTEGER);
BEGIN
  c := a + b;
END;
```

2. Přeložte následující proceduru v Pascalu do instrukcí našeho CPU.

```
PROCEDURE incitem5(VAR a:ARRAY[0..9] OF INTEGER);
VAR index:INTEGER;
BEGIN
  index := 5;
  a[index] := a[index] + 1;
END;
```

3. Přeložte následující proceduru v Pascalu do instrukcí našeho CPU.

```
PROCEDURE incarr(VAR a:ARRAY[0..9] OF INTEGER);
VAR i:INTEGER;
BEGIN
  FOR i := 0 TO 9 DO
    a[i] := a[i]+1;
END;
```

1)

```
LLD4  -16
LLD4  -20
ADDI4
LLD4  -12
LST4
RET
```

2)

```
LLD4  -12
LIC4   20
ADDI4
LLD4
LIC4    1
ADDI4
LLD4  -12
LIC4   20
ADDI4
LST4
ret
```

reference a[5]

read a[5]

reference a[5]

store

3)

```
LIC4    0
cycle: LDD4   0
LDD4  -12
ADDI4
LDD4
LIC4    1
ADDI4
LDD4    0
LDD4  -12
ADDI4
LST4
```

copy i

get reference a[i]

get a[i]+1

get reference a[i]

store to a[i]

```
LIC4    4
ADDI4
LIC4   36
LDD4    0
JLEI4  cycle
LIC4  -40
ADDI4
ADDI4
RET
```

increment

copy i

clean stack

Kód studenta 11

6 Optimalizace (3 body)

Poznámka: Bez dalšího vysvětlování můžete ve svých odpovědích použít následující větu (neformálně): Každý tok je možné dekomponovat na toky po cestách a cyklech.

1. Nechť $G = (V, E)$ je orientovaný graf a $s, t \in V$ jsou jeho dva různé vrcholy. Formulujte problém nalezení nejkratší cesty v grafu G z vrcholu s do vrcholu t jako úlohu celočíselného lineárního programování tak, že pro každou hranu $e \in E$ použijte binární proměnnou x_e a žádné další proměnné nebudete používat.
2. Uvažte lineární relaxaci Vašeho lineárního programu z bodu 1 a porovnejte hodnoty optimálních řešení původního celočíselného LP a jeho relaxace. Co o nich platí? Vysvětlete.
3. Formulujte duální program k lineárnímu programu z bodu 2; pokuste se duální program formulovat tak, aby nepoužíval více než $|V|$ proměnných.

1) pro hrany : $\forall x_{v_i,v_j} : 1 \geq x_{v_i,v_j} \geq 0 \quad \in [0,1]$

$\forall v_i \in V / \{s,t\} \left(\sum x_{v_i,v_j}\right) = 2$

$v_i \in \{s,t\} \left(\sum x_{v_i,v_j}\right) = 1$

$v = \{v_i, v_j\} \; x_e \sim x_{v_i,v_j}$

přes co sčítáte?

$\min \left(\sum_{v_1,v_2 \in V} x_{v_1,v_2} \right)$

2)

3) $Ax \leq b \rightarrow A^T y \leq c$

$\max c^T x \qquad \min b^T y$

$\sum_i a_{ij} x_j \leq b_j \Rightarrow y_i \geq 0$ (≤ 0 pokud P minimal.)

— ||— = —||— $\Rightarrow y_i \in R$

$x_i \in R \Rightarrow \sum_i a_{ji} y_j = c_j$

$x \geq 0 \Rightarrow \sum_i a_{ji} y_j \geq c_j$ ($\leq$ pokud P minimal)

2+

Kód studenta 11

1 Automaty a gramatiky (3 body)

1. Uveďte dvě definice (či charakterizace), kdy je jazyk L nad abecedou Σ *regulární.*
2. Uvažme dvě následující gramatiky G_1 a G_2 nad abecedou (terminály) $\Sigma_1 = \{a, b, \neg, \vee, \wedge, (,)\}$ resp. $\Sigma_2 = \{a, b, \neg, \vee, \wedge\}$ s jediným (startovním) neterminálem S_1 resp. S_2 a pravidly

$$G_1: \quad S_1 \to a \mid b \mid (\neg S_1) \mid (S_1 \vee S_1) \mid (S_1 \wedge S_1),$$
$$G_2: \quad S_2 \to a \mid b \mid \neg S_2 \mid S_2 \vee S_2 \mid S_2 \wedge S_2.$$

Určete, zda G_1 resp. G_2 generuje regulární jazyk. Svá tvrzení zdůvodněte.

Pozn.: pumping lemma je implikace, nedokazuje reg. jazyky.

1) přijímá ho deterministický konečný automat

~~2)~~ platí o něm pumping lemma pro reg. jazyky

3) je generován gramatikou s pravidly:

$V \to TR$ $\quad V, R \in$ neterminály

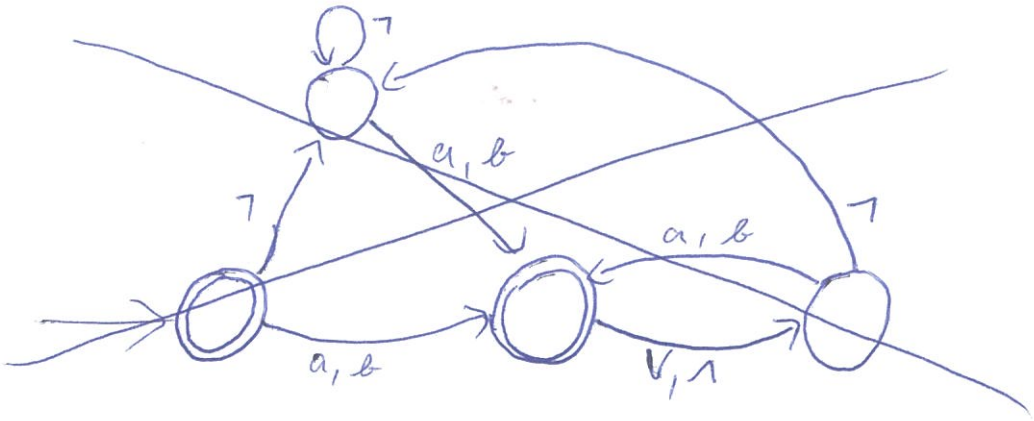
$T \in$ terminály *

(musí taková existovat) ↑ to nestačí, je třeba: $V \to T$

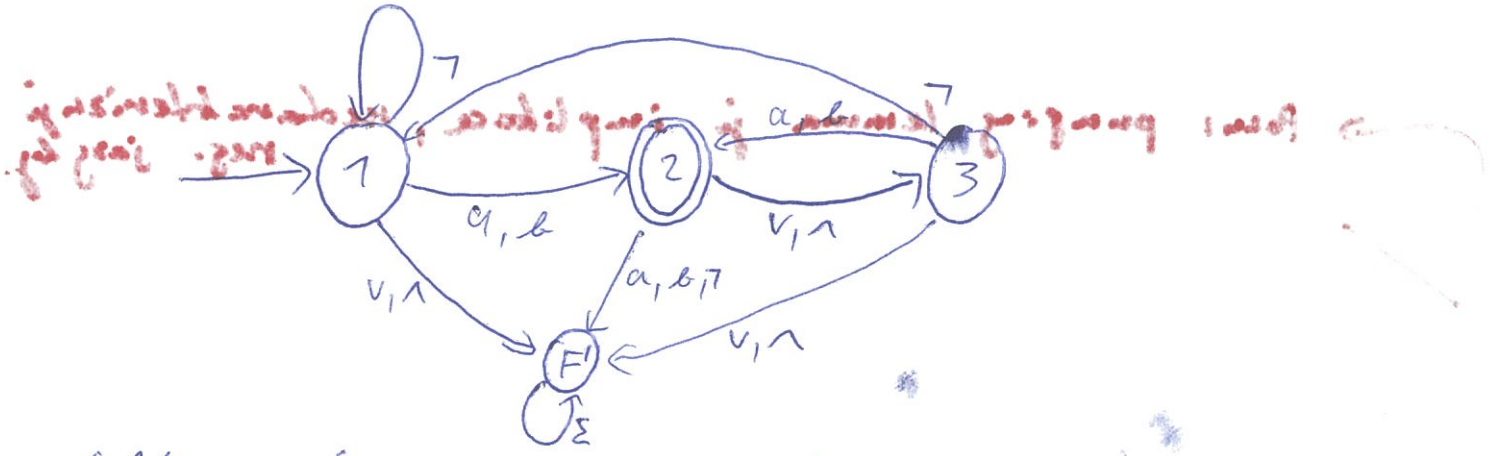
(popř. lze pravidla zrcadlově obrátit)

2) $L(G_1)$ není regulární, protože nelze (a proč ne?) vybrat konečný deterministický automat, který by byl schopen zkontrolovat správné uzávorkování.

$L(G_2)$ ~~[illegible]~~ je regulární, protože k němu lze sestrojit konečný determ. automat (druhá strana) ↙



DFA (G)



$(Q, \Sigma, \delta, F, S)$

$Q = \{1, 2, 3, F'\}$

$\Sigma = \{a, b, \neg, \vee, \wedge\}$

$F = \{2\}$

$S = \{1\}$

δ je označeno výše

F' je failed stav

3-

Kód studenta 11

2 Prohledávání do šířky (3 body)

1. Popište datovou strukturu seznamu sousedů pro reprezentaci neorientovaného grafu.
2. Napište pseudokód algoritmu prohledávání neorientovaného grafu $G = (V, E)$ do šířky počínaje v zadaném vrcholu $s \in V$.
3. Určete časovou složitost prohledávání do šířky při použití seznamu sousedů pro reprezentaci vstupního grafu.
4. Definujte pojem bipartitního grafu a popište způsob použití průchodu do šířky pro určení toho, zda je daný neorientovaný graf bipartitní.

1)

```
Vertex {
    id; // popř. nutná data
    Vertex[]; // seznam sousedů
}
```

sousedi

~~Celá~~ tudíž datová struktura sama je typu Vertex[] indexovaná číslem vrcholu

Tato datová struktura je vhodná pro procházení grafu, ale musí být správně inicializována (pokud v má v_1 v seznamu sousedů, musí v_1 mít v v seznamu sousedů)

V indexované struktuře by šlo pracovat i se seznamem vrcholů a dvojic vrcholů pro hrany (což zanáší neorientovanost)

```
queue = {s}
visited = {s}
while (queue is not empty) {
    v = queue.pop()
    foreach v.sousedi {          // obdoba pro BFS (jako porovnání hledaným hodnotou)
        if soused isn't in visited {
            add soused to visited
            add soused to queue
}}}
```

ok.

3) prohledávání do šířky má složitost $O(N+M)$ pro $N=|V|$ a $M=|E|$

napsaný pseudokód by takové složitosti nedosahoval, bylo by potřeba místo seznamu visited přidat příznak pro každý vrchol nebo visited indexovat podle unikátního id vrcholu. ok.

4) bipartitní graf lze rozdělit do dvou podmnožin $V_1, V_2 \subseteq V$ $V_1 \cap V_2 = \emptyset$ $V_1 \cup V_2 = V$ tak, že pro obě množiny neexistuje hrana $\{v_1, v_2\}$ taková, že $v_1, v_2 \in V_1 \vee v_1, v_2 \in V_2$.

při průchodu do šířky inicializujeme pro každý vrchol příznak, kterým rozlišíme vrchol pro levou partitu, pravou nebo žádnou. při procházení sousedů vrcholu kontrolujeme, zda má opačnou nebo nepřiřazenou známku, pak mu přiřadíme opačnou, pokud by měl stejnou, nejednalo by se o bipart. graf ok.

25

Kód studenta 11

7 Morfologická, syntaktická a sémantická analýza přirozeného jazyka (otázka studijního zaměření – 3 body)

1. Uveďte alespoň 4 základní požadavky, které by měl splňovat dobrý program na kontrolu překlepů, a zdůvodněte je.
2. Uveďte dvě nejčastěji používané metody (pro kontrolu překlepů) založené na využití slovníku daného jazyka a vysvětlete, pro jaké typy jazyků se hodí a proč.
3. Nedílnou součástí kontroly překlepů je také nabízení vhodných oprav. Uveďte alespoň dvě prakticky použitelné metody, pomocí kterých se uživatelům budou nabízet vhodné opravy.

1) - mít vysokou precision
- mít málo false positives (nejlépe žádné)
- správní chybu dle úrovně komunikoval s uživatelem ✓
- nebýt pomalý, uživatel nechce čekat

2) Asmul ?

WORDLIST
SLOVNÍK LEMMAT

~~Dosvidk~~

Mosaic ?

další je morfologická disambignace - pokud se slovo vyskytuje všechny ? značky, musí být vevitě chyba

3) ~~pot~~ - podtržení a nabídka opraveného slova
- automatické nahrazení (u jistých případů jako jsou dvojitý klik písmena, prohození sousedních písmen) ✓

25

Kód studenta 11

8 Základní formalismy pro popis přirozeného jazyka (otázka studijního zaměření – 3 body)

1. Unifikační gramatiky využívají speciální datový typ, tzv. Sestavu rysů (feature structure). Uveďte jeho základní vlastnosti.
2. Vysvětlete, jak funguje mechanismus unifikace dvou sestav rysů. Zdůvodněte, proč je při tvorbě unifikační gramatiky přirozených jazyků nutné používat typované sestavy rysů.
3. Nejvíce rozšířenou unifikační gramatikou byla Head Driven Phrase Structure Grammar (HPSG). Uveďte základní vlastnosti tohoto typu gramatiky.

1) sestava rysu je struktura atribut: hodnota
pomocí ~~kter~~ které můžeme unifikovat, unifikace
probíhala pro slova se stejnými hodnotami, a proto
se zavedly typování sestav (aby šlo unifikovat např. podst. jm
a slovesa)

2) unifikujeme taková sestavy rysů, které odpovídají
hodnotám vzájemných rysů, problém byl unifikace
hodnoty pádu a osoby (a podobných)

→ zavedení typových rysů
- pro určité typy slov jsou dány typové sestavy tak,
aby se nemohly unifikovat dvě různé vlastnosti
se stejnou hodnotou
- určité typy slov mají podobné vlastnosti, proto
se hodí mít typování rysy (např. pro slovesa
osoba, čas, způsob ...)

3) ✓

26+

Kód studenta 11

9 Základy teorie informace (otázka studijního zaměření – 3 body)

Házíme hrací kostkou a hozené číslo z množiny $\{1, 2, 3, 4, 5, 6\}$ interpretujeme jako hodnotu náhodné proměnné X. Předpokládejme, že X má rovnoměrné rozdělení. Dále uvažujme náhodnou proměnnou Y s hodnotami *sudé/liché* a náhodnou proměnnou Z s hodnotami *true* (pokud padne číslo větší než 4) nebo *false* (pokud nepadne číslo větší než 4). Obory hodnot náhodných proměnných jsou shrnuty v tabulce

náhodná proměnná	hodnoty
X	$\{1, 2, 3, 4, 5, 6\}$
Y	$\{sudé, liché\}$
Z	$\{true, false\}$

1. Jsou proměnné X a Y statisticky nezávislé? Zdůvodněte.
2. Která z proměnných X, Y, Z má největší entropii? Odpověď přesně zdůvodněte.
3. Určete vzájemnou informaci $I(X;Y)$. Výsledek zdůvodněte.

1) nejsou, aby byly, muselo by platit $P(X|Y) = P(X) \quad \forall X \in X$

a pro $P(X=2 \mid Y=liché) = 0$ ale $P(X=2) = \frac{1}{6}$ ✓

2) $E(X) = -\sum_{x \in X} P(x) \cdot \log_2(P(x)) = 6 \cdot \frac{1}{6} \cdot \log_2(6) = \log_2(6)$

ENTROPY $E(Y) = -\sum_{y \in Y} P(y) \cdot \log_2(P(y)) = 2 \cdot \frac{1}{2} \cdot \log_2(2) = 1$ ✓

$E(Z) = -\sum_{z \in Z} P(z) \cdot \log_2(P(z)) = \frac{2}{3} \cdot \log\left(\frac{3}{2}\right) + \frac{1}{3} \cdot \log(3)$

$\Rightarrow$ X má největší entropii, ✓

což dává smysl, protože X má více hodnot a má je rovnoměrně rozdělené

$$I(X;Y) = E(X) - E(X|Y) = \underset{\uparrow \frac{1}{12}}{E(Y)} - E(Y|X)$$

$$E(Y|X) = -\sum_{\substack{y \in Y \\ x \in X}} p(x,y) \cdot \log_2(\quad)$$

$P(y|x) \to \frac{1}{3}$

$$= \frac{1}{12} \cdot \left(6 \cdot \log(3)\right) = \frac{1}{2}\log(3)$$

$$\Rightarrow$$

$$I(X;Y) = 1 - \frac{\log_2(3)}{2}$$