3

Kód studenta 15

1 Automaty a gramatiky (3 body)

1. Definujte bezkontextovou gramatiku a Chomského normální formu bezkontextové gramatiky.
2. Mějme gramatiku $G = (\{E, B, I, J\}, \{0, 1, +, (,)\}, \mathcal{P}, E)$, kde $\mathcal{P}$ sestává z následujících pravidel (λ označuje prázdné slovo):

$$\begin{aligned}\mathcal{P} = \{E &\to E \mid B \mid I,\\ B &\to (E + E),\\ I &\to 0 \mid 1J,\\ J &\to 0J \mid 1J \mid \lambda\}\end{aligned}$$

   (a) Sestrojte levou derivaci slova $w = ((0 + 1) + 10)$ z gramatiky G. Zakreslete ji také ve formě derivačního stromu.

   (b) Najděte Chomského normální formu gramatiky G.

v př. 2 je špatné pořadí P a S, tedy gramatika je (V, T, P, S)

1. Bezkontextová gramatika je (V, T, S, P), kde

V, P, T konečné

V je množina neterminálů

T je množina terminálních symbolů

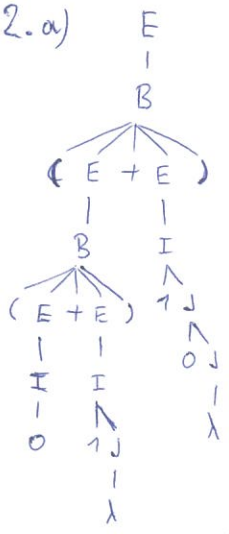
$V \cap T = \emptyset$

$S \in V$ je startovací symbol

P je množina pravidel tvaru $A \to \omega$, kde $A \in V$, $\omega \in (V \cup T)^*$.

Bezkontextová gramatika je v Chomského normální formě, pokud všechna pravidla jsou tvaru $A \to BC$ pro $A, B, C \in V$ nebo $A \to a$ pro $A \in V$, $a \in T$. ✓

2. a)



✓

$E \Rightarrow B \Rightarrow (E + E) \Rightarrow (B + E) \Rightarrow ((E + E) + E) \Rightarrow$

$\Rightarrow ((I + E) + E) \Rightarrow ((0 + E) + E) \Rightarrow ((0 + I) + E) \Rightarrow ((0 + 1J) + E) \Rightarrow$

$\Rightarrow ((0 + 1) + E) \Rightarrow ((0 + 1) + I) \Rightarrow ((0 + 1) + 1J) \Rightarrow ((0 + 1) + 10J) \Rightarrow$

$\Rightarrow ((0 + 1) + 10)$ ✓

DRUHÁ STRANA

2. b) nejprve odstraníme pravidlo $J \to \lambda$

$E \to E \mid B \mid I$
$B \to (E+E)$
$I \to 0 \mid 1 \mid 1J$
$J \to 0 \mid 0J \mid 1 \mid 1J$

dále se zbavíme jednotkových pravidel $E \to B$, $E \to I$, $E \to E$

$E \to (E+E) \mid 0 \mid 1 \mid 1J$
$J \to 0 \mid 0J \mid 1 \mid 1J$

zavedeme nové neterminály $[0]$ a $[1]$

$E \to (E+E) \mid 0 \mid 1 \mid [1]J$
$J \to 0 \mid 1 \mid [0]J \mid [1]J$
$[0] \to 0$
$[1] \to 1$

rozdělíme dlouhé pravidlo $E \to (E+E)$ pomocí nových neterminálů $[(], X, Y, Z, [)], [+]$

$E \to [(]X$
$X \to EY$
$Y \to [+]Z$
$Z \to E[)]$

$[(] \to ($
$[)] \to)$
$[+] \to +$

Dohromady:

$E \to [(]X$
$X \to EY$
$Y \to [+]Z$
$Z \to E[)]$

$E \to [1]J$
$J \to [0]J$
$J \to [1]J$

$E \to 0$
$E \to 1$
$J \to 0$
$J \to 1$

$[(] \to ($
$[)] \to)$
$[+] \to +$

$[1] \to 1$
$[0] \to 0$

neterminály $\{E, X, Y, Z, [1], [0], [(], [)], [+]\}$, terminály stejné jako původní gramatika ✓

3 kop

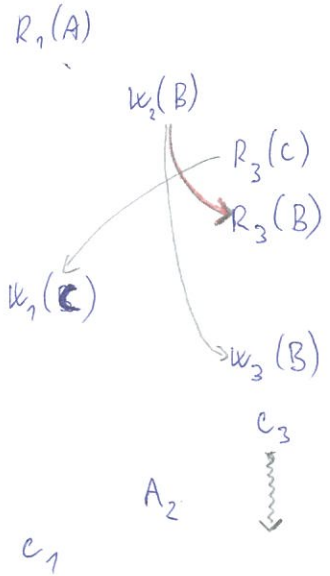
Kód studenta 15

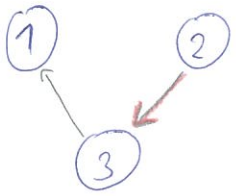
3 Databáze (3 body)

1. Uvažujte transakční rozvrh $T_{123} = R_1(A), W_2(B), R_3(C), R_3(B), W_1(C), W_3(B), COMMIT_3, ABORT_2, COMMIT_1$. V zápisu rozvrhu $R_i(X)$, resp. $W_i(X)$ odpovídá čtení a zápisu proměnné X v i-té transakci.
   - Je rozvrh T_{123} konfliktově uspořádatelný (conflict-serializable)? Odpověď zdůvodněte.
   - Je rozvrh T_{123} zotavitelný (recoverable)? Odpověď zdůvodněte.

   Pokud rozvrh některou z vlastností nesplňuje, opravte jej tak, aby ji splňoval. Pokud to lze, dodržte vzájemné pořadí čtení a zápisů. Pokud to nelze, vysvětlete proč ne.
2. Nad tabulkami *Prodavac*(*ProdavacID*, *Jmeno*, *Pobocka*, *DenNastupu*) a *Prodeje*(*ProdavacID*, *DenProdeje*, *Castka*), najděte pomocí SQL všechny prodavače, kteří v období '1.5.2020' až '31.5.2020' nic neprodali. Pokud v daném dni prodavač nic neprodal, nemá pro daný den v tabulce *Prodeje* žádný řádek.

1.





je konfliktově uspořádatelný v pořadí T_2, T_3, T_1 ✓
– sériový rozvrh s tímto pořadím transakcí má stejné konflikty: $W_2(B) - R_3(B)$, $W_2(B) - W_3(B)$, $R_3(C) - W_1(C)$

není zotavitelný, protože T_3 závisí na T_2 (červená šipka), tedy $COMMIT_3$ nesmí být před $ABORT_2$. Aby byl zotavitelný, musí $COMMIT_3$ proběhnout až po dokončení T_2.
Navíc, protože T_2 končí ABORTem, musíme provést i $ABORT_3$, neboť T_3 závisí na datech zapsaných T_2. ✓

2.

```
SELECT ProdavacID, Jmeno
FROM Prodavac AS P
WHERE (SELECT COUNT(*)
       FROM Prodeje AS R
       WHERE P.ProdavacID = R.ProdavacID
         AND R.DenProdeje ≥ '1.5.2020' AND R.DenProdeje ≤ '30.5.2020')
      = 0
```

✓

3

Kód studenta 15

5 Architektura počítačů (3 body)

Mějme následující úryvek kódu ze třídy implementující spojový seznam:

```
1  struct SimpleListNode {
2      Data           payload;
3      SimpleListNode *next;
4  };
5
6  class SimpleList {
7  public:
8      // insert node n at the beginning of the list
9      void InsertNode(SimpleListNode *n) {
10         n->next = root;
11         root = n;
12     }
13
14     // insert two nodes at the beginning of the list
15     // the nodes must be neighbors, that is,
16     // first->next==second should be true
17     void InsertPair(SimpleListNode *first,
18                     SimpleListNode *second) {
19         InsertNode(second);
20         InsertNode(first);
21     }
22
23     ...
24
25 private:
26     SimpleListNode *root;
27 }
```

Tento úryvek kódu je v C++, pro ostatní jazyky (C# a Java) stačí smazat * a nahradit -> znakem tečky.

1. Může při běhu funkcí v tomto úryvku kódu dojít k jevu zvaném "race condition"? Pokud ano, popište aspoň jednu takovou situaci. Uvažujte skutečně pouze tento úryvek kódu, nikoliv jeho interakci s dalším kódem třídy.
2. Pokud jsou v úryvku nějaké kritické sekce, napište rozsahy řádek, kde se nacházejí.
3. Využijte nějaký synchronizační nástroj a upravte zadaný úryvek kódu tak, aby fungoval korektně i v paralelním prostředí.

1. Pokud bychom pouze spouštěli funkce z úryvku v jedné vlákně, k race condition nedojde. Pokud spustíme funkci InsertNode ze dvou vláken současně, může dojít k race condition, a to konkrétně pokud jedno z vláken bude pozastaveno mezi řádky 10 a 11 a v tu chvíli proběhne druhé vlákno:

InsertNode(n1):
n1->next = root

InsertNode(n2):
n2->next = root
root = n2

root = n1 ← v tuto chvíli máme v seznamu pouze n1 (a to, co nám bylo předtím), n2 jsme přišli o a v seznamu není.

2. kritické sekce: 10–11 (popsáno v odpovědi 1.)

19–20 pokud dojde k podobné situaci jako bylo zmíněno výše, tedy prostřední vlákno mezi řádky 19 a 20 a během jiného vlákna a funkce InsertPair, dojde k umístění páru z druhého vlákna mezi prvky pár z prvního vlákna, nebude tedy splněna podmínka zmíněná v komentáři first->next == second.

3. stačí přidat zámky kolem kritických sekcí

v syntaxi C# přidat: lock (this) { před řádek 10 a před řádek 19

a } za řádek 11 a za řádek 20

Zámek je tady asociovaný s „this", tedy s instancí třídy SimpleList, se kterou pracujeme. V C# je možné použít pro oba tyto zámky stejný objekt, pokud má vlákno nad zámek pro tento objekt, může ho zamknout znovu (a pak musí být dvakrát odemknut). V některých jiných jazycích bychom pro každou kritickou sekci museli mít samostatný zámek.

36

Kód studenta 15

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 uniformní 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. Která z proměnných X, Y, Z má největší entropii? Odpověď přesně zdůvodněte.
2. Určete vzájemnou informaci $I(X;Y)$. Výsledek zdůvodněte.

1. $H(A) = \sum_{a \in A} p(a) \cdot \log \frac{1}{p(a)}$

$H(X) = \sum_{1}^{6} \frac{1}{6} \cdot \log \frac{1}{\frac{1}{6}} = 6 \cdot \frac{1}{6} \cdot \log 6 = \log 6$ ✓

$H(Y) = \frac{1}{2} \cdot \log \frac{1}{\frac{1}{2}} + \frac{1}{2} \cdot \log \frac{1}{\frac{1}{2}} = \log 2 = 1$ ✓

(sudé, liché)

víme, že uniformní rozdělení má největší entropii

$H(Z) = \underbrace{\frac{1}{3} \cdot \log \frac{1}{\frac{1}{3}}}_{>4} + \underbrace{\frac{2}{3} \cdot \log \frac{1}{\frac{2}{3}}}_{<4} = \frac{1}{3} \log 3 + \frac{2}{3} \log 3 - \frac{2}{3} \log 2 = \log 3 - \frac{2}{3} \log 2 = \log 3 - \frac{2}{3} \leq H(Y)$ ✓

$\log 6 > 1 \Rightarrow$ největší entropii má X

2. $I(X,Y) = H(Y) - H(Y|X)$

vidíme, že pro každou z hodnot X je jednoznačně určena hodnota Y (jedna z možných hodnot má pravděpodobnost 1, druhá 0)

$H(Y|X) = \sum_{\substack{x \in X \\ y \in Y}} p(x,y) \cdot \log \frac{1}{p(y|x)} = 6 \cdot \left(1 \cdot \log \frac{1}{1}\right) = 0$

$I(X,Y) = H(Y) - H(Y|X) = 1 - 0 = 1$ ✓

2,5 –

Kód studenta 15

8 Jazykové modelování (otázka studijního zaměření – 3 body)

1. Popište základní fakta o tvorbě jazykových korpusů.
2. Rozdělte jazykové korpusy podle typu značkování.
3. Pro každý typ značkování uveďte alespoň jeden příklad korpusu češtiny a jeden příklad nějakého jiného jazyka.

Korpus sám o sobě nic nedělá, tvůrci se snaží. poskytovat vzorek jazyka

1. Jazykový korpus (se snaží být) reprezentativním vzorkem skutečně používaného jazyka. Jedná se o soubor textů (z různých oblastí* – beletrie, novinové články, odborné texty, ...). Typicky se korpus snaží zachytit současný stav jazyka, proto obsahuje texty z ne příliš dlouhého časového rozpětí (neobsahuje historické, staré texty). *diachronní vs. synchronní korpusy* Korpusy bývají konečné (jakmile jsou vydané, už se do nich nepřidávají další texty, případně je korpus s přidanými texty vydán jako další verze), aby na nich šlo provádět kvantitativní analýzu (četnost výskytů slov a jazykových jevů). Korpus nikdy nezachytí všechny možné jazykové jevy, nicméně může být užitečným nástrojem pro zkoumání jazyka, zejména pro jazykové modelování. Existují i paralelní korpusy (dvou a vícejazyčné s vyznačením toho, které věty si odpovídají), které jsou užitečné pro tvorbu modelů pro překlad.

* texty jsou vybírány tak, aby byl korpus co nejvíce reprezentativní

2. Značkování může probíhat na morfologické úrovni (morfologické kategorie (tagy), lemmata), na syntaktické úrovni (stavba vět), případně může být i jejich kombinace a přidávat i další úrovně.

3. Český národní korpus – morfologické značkování *automatické*

   Pražský závislostní korpus – morfologické i syntaktické (závislostní stromy), značkování vychází z Funkčního generativního popisu, proto má několik rovin

   Penn Treebank – angličtina, syntaktické značkování (složkové stromy)

   Brown corpus – angličtina, bez značek ✗

36

Kód studenta 15

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

Sémantika se v přirozených jazycích uplatňuje na mnoha úrovních, od významu jednotlivých slov až po význam delších textových úseků. Vysvětlete některé základní sémantické pojmy:

1. Vysvětlete pojem „ontologie“ ve zpracování sémantiky přirozeného jazyka.
2. Popište sémantickou síť Wordnet, její strukturu a historii, a uveďte alespoň dva příklady aplikace této sítě v různých oblastech zpracování přirozeného jazyka.
3. Vysvětlete pojem „anafora“ a uveďte základní kategorie anafory v textu.

1. Ontologie je množina kategorií, do kterých slova dělíme podle jejich společného významu. Můžeme mít například kategorii pro slovesa pohybu, pro slova vyjadřující počty, ... Ontologie je potom seznam všech kategorií, které využíváme a které slovům přiřazujeme.

2. Wordnet se snaží popsat sémantické vztahy mezi slovy. Spojuje slova do synsetů = skupin synonym. Mezi synsety potom modeluje další sémantické vztahy – např. antonyma, náležení do množiny („červená“ patří mezi „barvy“), ... Každý synset může být také zařazen do příslušné kategorie z ontologie. Kromě toho můžeme ve Wordnetu popisovat také lexikální vztahy mezi slovy – např. homonymii.
Původní Wordnet obsahoval pouze anglická slova. Později vznikl projekt EuroWordnet, který dokáže sesouvat slova z různých jazyků – proto se EuroWordnet dá použít jako slovník při strojovém překladu. Další využití Wordnetu jsou například pro vyhledávání v textu, kdy můžeme hledat nejen zadané slovo, ale také jeho synonyma, případně příbuzná slova.

3. Anafora je jazykový jev, jehož interpretace závisí na kontextu.
patří sem – odkazování v textu → zpět = anafora (např. nevyjádřený podmět, který se dá doplnit z předchozí věty)
dopředu = katafora
odkazování mimo text – např. na předměty z okolního světa
elipsa = vynechání části věty (ale jde doplnit z kontextu)

Kód studenta 15

3

6 Transportní vrstva v TCP/IP (3 body)

1. Jaké úkoly plní transportní vrstva v architektuře TCP/IP?
2. Uvažme následující vlastnosti přenosů: blokový vs proudový, spojovaný vs nespojovaný, spolehlivý vs nespolehlivý, garantovaný vs negarantovaný, s nebo bez možnosti řešit řízení toku a konečně s nebo bez možnosti předcházet zahlcení sítě. Jaké vlastnosti ve srovnání s IP protokolem na síťové vrstvě v tomto smyslu nabízí protokoly TCP (*Transmission Control Protocol*) a UDP (*User Datagram Protocol*)?
3. Jakou funkci plní porty a jaké konvence používáme pro jejich přidělování – jak například konkrétně víme, že pro SMTP se používá port 25 nebo u PostgreSQL port 5432? Jsme v používání konkrétních portů nějak či někým omezováni? Jaké porty se používají pro označení odesílatele ve spojeních s odpovědí?
4. Jak vypadá relativní a absolutní transportní adresa a jak se identifikují (navzájem rozlišují) jednotlivá transportní spojení (komunikace)?

1. Transportní vrstva zajišťuje spojení mezi dvěma aplikacemi. Kromě samotného posílání dat může navíc nabízet například spolehlivost doručení (TCP při nedoručení dat pokus opakuje, při použití UDP musí tolle řešit aplikace), udržení „trvalého" spojení a další služby.

2.

TCP	UDP	IP
proudový	blokový	blokový
spojovaný	nespojovaný	nespojovaný
spolehlivý	nespolehlivý	nespolehlivý
garantovaný	negarantovaný	negarantovaný
s řízením toku	bez	bez
s předcházením zahlcení	bez	bez

TCP obecně řeší mnoho věcí za aplikaci, např. opakuje odesílání dat, pokud mu nepřišlo od druhé strany potvrzení přijetí. UDP a IP tyto služby nenabízí.

3. Port označuje aplikaci na konkrétním počítači. Díky tomu může na jednom počítači běžet více aplikací a každá z nich dostává pouze data, která jí patří.
SPÍŠ PŘECHODOVÝ BOD

Součástí RFC specifikujícího protokol může být i informace o tom, na jakém portu se má aplikace ve výchozím nastavení provozovat (nicméně je možné aplikaci spustit i s jiným portem). U už dlouho existujících protokolů se při vydání nové verze protokolu zachovává (právě aby bylo možné vědět, na jakém portu aplikace běží). Pro označení těchto „well-known services" se používají nižší čísla portů (< 1024), naopak pro označení odesílatele (tedy portu, na kterém očekávám odpověď) se použijí vyšší čísla portů (> 1024, aby nedošlo ke konfliktu s nějakou jinou aplikací (serverem).
SYSTÉMOVĚ REGISTROVANÉ

OK, ALE CO PROTOKOL? ODESILATEL X PŘÍJEMCE

4. Transportní spojení se identifikují pomocí sockets, což je dvojice (IP adresa, port). IP adresa určuje počítač, port určuje aplikaci na něm. Socket je tedy označení jednoho konce spojení.

IP:PORT VS. PORT

Kód studenta 15 (3)

2 Algoritmy a datové struktury (3 body)

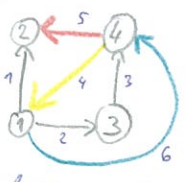
1. Napište pseudokód algoritmu prohledávání orientovaného grafu do hloubky (DFS).
2. Jaká je časová složitost algoritmu DFS, pokud je graf reprezentován maticí sousednosti?
3. Popište DFS klasifikaci hran orientovaných grafů.
4. Rozhodněte, zda v orientovaném grafu může existovat kružnice složená pouze ze zpětných hran.

1. Používáme standardní značení grafů, $uv \in E$ je orientovaná hrana $u \to v$.
Popíšeme rekurzivní DFS z vrcholu v.

Před začátkem DFS je potřeba provést inicializaci:
$\forall u \in V$: state $(u) \leftarrow$ unseen
Pak lze spustit DFS (v). forall $v \in V$

DFS (v):
for $u \in V$: $vu \in E$:
if state (u) = unseen:
state $(u) \leftarrow$ open
DFS (u)
state $(v) \leftarrow$ closed

2. Každý vrchol otevřeme nejvýše jednou a pak [redacted] projdeme po všech hranách vedoucích z něj. Pro matici sousednosti musíme vždy otestovat $\Theta(n)$ možných cílových vrcholů na existující hrany. Celková časová složitost bude $\Theta(n^2)$, kde $n = |V|$.

3.


procházíme graf z vrcholu 1, hrany ve vyznačeném pořadí

a) stromové hrany – po nich DFS postupuje
b) dopředné hrany – do vrcholu, který už je uzavřený, ale byl poprvé otevřený až po odchodem vrcholu = jiná cesta do něho „podstromu"
c) zpětné hrany – zpět do už otevřeného vrcholu (stále je otevřený)
d) příčné hrany – do vrcholu v jiném podstromu (už je uzavřený)

Pokud otevření vrcholu označíme $($ a uzavření $)$ s číslem vrcholu, dají se klasifikace pro hranu $x \to y$ vyjádřit takto:
a) $\ldots (_x \ldots (_y \ldots)_y \ldots)_x \ldots$ b) $\ldots (_x \ldots (_y \ldots)_y \ldots)_x \ldots$ c) $\ldots (_y \ldots (_x \ldots)_x \ldots)_y \ldots$ d) $\ldots (_y \ldots)_y \ldots (_x \ldots)_x \ldots$

4. Nemůže. Pokud $in(v)$ označíme pořadí otevřaných vrcholů, musí pro opětnou hranu $x \to y$ platit $in(y) < in(x)$ ... hrana vede do už otevřeného vrcholu. Není možné, aby pro ~~posloupnost (cyklus)~~ brusnici $x_1, \dots x_n$ platilo $in(x_1) > in(x_2) > \dots > in(x_n) > in(x_1)$.

1) OK
2) OK
3) OK

3

Kód studenta 15

4 Programování (3 body)

Navrhněte objektovou datovou strukturu (implementovanou v jazyce C++, C# nebo Java), reprezentující matematický výraz obsahující konstanty (reprezentované jako double), proměnné (jejichž jména jsou řetězce písmen), operátory $+$, $-$, $*$, $/$ a rozšiřitelnou množinu matematických funkcí (jejichž jména jsou rovněž řetězce písmen). Uveďte veřejná i interní rozhraní umožňující následující funkčnost:

1. Tisk daného výrazu (v programátorské notaci; neřešte přitom přehlednost tištěného výrazu, jako např. nadbytečné závorky).
2. Konstrukce nového výrazu reprezentujícího symbolickou derivaci daného výrazu podle zadané proměnné. (Datová struktura reprezentující originální výraz musí při derivování zůstat nedotčena.) Dbejte na korektnost vytvořené struktury, např. v případě, kdy se při derivování duplikuje podvýraz (např. u derivace podílu), a uveďte v rozhraní i pomocné funkce, které jsou pro zajištění korektnosti nutné. Jako demonstraci použití těchto rozhraní napište implementaci funkce realizující derivaci matematické funkce tg podle vzorce

$$\frac{\mathrm{d}}{\mathrm{d}x} tg(f) = \frac{\frac{\mathrm{d}}{\mathrm{d}x} f}{sqr(cos(f))}$$

3. Mechanismus (bez větších nároků na rychlost, použitelný např. při načítání výrazu z textového vstupu), který pro zadaný řetězec obsahující jméno matematické funkce zkonstruuje objekt reprezentující volání této funkce s danými parametry. Tento mechanismus musí umožňovat snadnou rozšiřitelnost množiny matematických funkcí, s minimálním množstvím zásahů do společného kódu. Kterému návrhovému vzoru se řešení podobá?

Struktura musí umožňovat dostatečně rychlou manipulaci, není přípustné např. vyhledávání názvů funkcí v tabulkách během konstrukce derivace.

výraz můžeme uložit do proměnné ExpressionNode, která je kořenem našeho výrazu = poslední provedená operace

```
public interface ExpressionNode {
    string Print();  // případně lze také void Print(TextWriter output)
    ExpressionNode Clone();  // deep copy
    ExpressionNode Derive(string variable);  // vrací nový výraz reprezentující požadovanou derivaci
}

public class Constant : ExpressionNode {
    double value;
    public Constant(double value) {...}
    ... implementace ExpressionNode
}
public class Variable : ExpressionNode {
    string name;
    public Variable(string name) {...}
    ... implementace
}
```

OK

OK

```
public abstract class BinaryOperator : ExpressionNode {
    protected ExpressionNode left, right; //operandy
    public BinaryOperator (ExpressionNode left, ExpressionNode right) {...} // podobně budou využívat tento konstruktor přes base(...
    ... implementace (rozhraní implementovaná abstraktně)
}
public class AddOperator: BinaryOperator {...} //podobně SubtractOperator, MultiplyOperator, DivideOper.
public abstract class UnaryFunction : ExpressionNode {
    protected ExpressionNode child;
    protected string name
    public UnaryFunction (ExpressionNode child) {...} //podobně jako v BinaryOperator
    ... implementace
}
public abstract class Function : ExpressionNode {
    protected ExpressionNode [] children;
    ...
}
```

1. Implementace Print bude využívat metody Print svých synů ve stromu.

2. předpokládejme existenci SquareFunction : UnaryFunction a CosFunction : UnaryFunction s výše naznačeným konstruktorem

   Derive uvnitř TyFunction : UnaryFunction bude vypadat takto:

```
return new DivideOperator (child.Derive(variable), new SquareFunction(new
    CosFunction (child.Clone()));
```

OK

Pro derivaci výrazu předpokládáme, že všechny funkce mají správný počet parametrů (jinak nahlásíme chybu), kdy listy stromu sestaveného z ExpressionNode jsou konstanty a proměnné. U těch je derivace snadná: je rovna 1 pouze pro proměnnou podle které derivujeme, jinak je 0.

3. Použijeme návrhový vzor Factory Pattern.

```
interface FunctionFactory {
    string GetName();
    ExpressionNode Create(ExpressionNode[] children); //předpokládáme, že parametry máme k dispozici
}
```

MAPA BY MOHLA LEPŠÍ

Pak budeme mít pole factories (tříd implementujících toto rozhraní) a v něm porovnáním jména najdeme správnou funkci a zavoláme Create. Pro přidání nové funkce stačí přidat novou factory do seznamu (pole) — k tomu můžeme mít veřejnou metodu, která to zařídí.