University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitni 8
30614 Pilsen
Czech Republic

Transformace XML dokumentu do podoby logického programu

Martin Zíma

Technical Report No. DCSE/TR-2009-13
Prosinec, 2009

Distribution: public

Technical Report No. DCSE/TR-2009-13
Prosinec, 2009

Transformace XML dokumentu do podoby logického programu

Martin Zíma

Abstrakt

Zpráva ukazuje postup, jak převést vstupní XML dokument na odpovídající logický program napsaný v jazyce Datalog. Důraz byl kladen na reprezentaci seznamové struktury, která je pro transformaci nezbytná, kterou námi implementovaná rozšířená verze jazyka Datalog nepodporuje. Bude též ukázáno, že výsledný logický program obsahuje množinu univerzálních pravidel, obsažených v každém výsledném logickém programu a dále specifickou množinu faktů reprezentujících vstupní XML dokument.

This work was supported by Ministry of Education, Youth and Sports of the Czech Republic – project 2C06009

Copies of this report are available on
http://www.kiv.zcu.cz/publications/
or by surface mail on request sent to the following address:

University of West Bohemia in Pilsen
Department of Computer Science and Engineering
Univerzitni 8
30614 Pilsen
Czech Republic

Obsah

1. Motivace

V současnosti je snaha uchovávat data na internetu v podobě sémantického webu. To znamená, že internetové dokumenty již nejsou psány v jazycích HTML resp. XHTML, ale jsou použity jazyky typu RDF [7], RDFS [6], OWL [5] apod. Všechny výše uvedené jazyky jsou založeny na jiném značkovacím jazyce, kterým je jazyk XML [8]. Pro vyhledání požadované informace v XML dokumentu se využívá dotazovacího jazyka XQuery [10], který je procedurální nadstavbou jazyka XPath [9]. Prostřednictvím jazyka XPath je možné definovat vybranou část XML dokumentu.

Zajímavým problémem současnosti je hledání způsobu, jak vyhodnotit dotaz nad XML dokumentem s využitím aparátu logických pravidel. Logický program může být napsán např. v jazyce Prolog nebo Datalog. Logické programování dovolí vyhodnocovat dotazy, které vyžadují výpočet tranzitivního uzávěru. Na našem pracovišti proběhl vývoj a realizace experimentálního deduktivního databázového systému [11], který umožní transformaci logického programu napsaného v rozšířené verzi jazyka Datalog do programového kódu v jazyce PL/SQL [4], který používá SŘBD Oracle.

S možným řešením, jak převést XML dokument na logický program, přišel Jesús M. Almendros-Jiménez [1]. Uvádí postup, jak převést XML dokument do množiny příslušných logických faktů a pravidel. K podchycení dat XML dokumentu používá strukturu seznam, která reprezentuje výsledek příslušného XPath dotazu. Logická pravidla programu definují strukturu XML dokumentu, tj. které značky jsou vnořené do jiné značky v konkrétním případě. Autor k definici struktury XML dokumentu definuje specifické funkce s různým počtem argumentů, bohužel nepopisuje způsob, jak tyto funkce vyhodnocovat. Výsledný logický program obsahuje veškerá data vstupního XML dokumentu v množině faktů, logická pravidla respektují strukturu daného XML dokumentu. To znamená, že pro každý XML dokument bude transformovaný logický program obsahovat jinou množinu pravidel.

Ve zbývající části dokumentu bude navržen a aplikován transformační postup, kterým bude vytvořen jiný logický program, který bude obsahovat univerzální pravidla. To jest

dva různé logické programy (získané transformací dvou různých XML dokumentů) budou obsahovat stejná logická pravidla.

2. Reprezentace seznamu

Logický program v jazyce Datalog, jehož rozšířená varianta byla vyvíjena na našem pracovišti, umožní používat v predikátech konstanty (číselné i textové), proměnné a složené argumenty v podobě aritmetického výrazu a agregované funkce. Bohužel, argument v podobě seznamu není podporován.

V minulosti byl na pracovišti řešen jiný problém, který také vyžadoval datovou strukturu seznam. Jednalo se o návrh logického programu vyhledání minimální cesty v orientovaném grafu. Výsledná minimální cesta nemohla být uložena do seznamu, ale byla reprezentována několika n-ticemi, které popisovaly nalezenou minimální cestu. Počet těchto n-tic udával, kolika uzly vyhledaná minimální cesta procházela. Z důvodu jednoduchosti nebyla řešena situace, kdy mezi dvěma uzly grafu existovalo více minimálních cest. Každá tato n-tice obsahovala tyto prvky:

- identifikaci uzlu, kde minimální cesta začíná,
- identifikaci uzlu, kde minimální cesta končí,
- identifikaci uzlu, kterým minimální cesta prochází – prvek seznamu
- další dva argumenty, které vychází z použitého Dijkstrova algoritmu [2, 3], určující pořadí popisovaného uzlu ležícího na minimální cestě.

Uvedený přístup je možné v modifikované míře aplikovat pro reprezentaci požadovaného seznamu, který je svázán s konkrétním výskytem značky jazyka XML. Tento seznam obsahuje značky jazyka XML, kterými je třeba v XML dokumentu projít, abychom se dostali ke kořenové značce (není vnořená do žádné značky). Je třeba zajistit, aby tento seznam měl definované pořadí prvků a také identifikaci, že tyto prvky patří do stejného seznamu. K dalšímu výkladu je vhodné použít jako příklad fragment XML dokumentu z [1], který je v levé části doplněn o čísla řádků:

```
 1.  <xml version="1.0">
 2.  <knihy>
 3.    <kniha rok="2003">
 4.      <autor>Abiteboul</autor>
 5.      <autor>Buneman</autor>
 6.      <autor>Suciu</autor>
 7.      <nazev>Data on the Web</nazev>
 8.      <recenze>Výborná kniha.</recenze>
 9.    </kniha>
     ...
80.  </knihy>
```

V [1] je zapsána informace, že Buneman je autorem dané knihy, následujícím faktem:

```
autor('Buneman', [2, 1, 1], 3, 'knihy.xml').
```

První argument popisuje hodnotu značky `autor`, druhý argument definuje formou seznamu strukturu XML dokumentu. Konkrétně první prvek seznamu, číslo 2, říká, že výskyt značky `autor` je druhý v pořadí uvnitř jiné značky (v našem případě značka `kniha`). Druhý prvek, číslo 1, definuje, že tato značka (`kniha`) je první v pořadí uvnitř další značky (značka `knihy`) a ta je první kořenovou značkou XML dokumentu (poslední prvek seznamu). Zbývající argumenty nejsou pro další výklad potřebné.

Výše popisovaný seznam lze zapsat do podoby XPath dotazu takto:

```
/knihy[1]/kniha[1]/autor[2]
```

K reprezentaci následujícího seznamu bude definován predikát `xml`, který bude použit k zápisu jednotlivých prvků seznamu v podobě faktů. Například, pro výše uvedeného autora bude logický program obsahovat 3 nové fakty. Společným a zároveň prvním argumentem těchto faktů (tj. prvků seznamu) bude číslo řádku vstupního XML dokumentu, kde je daná informace zapsána. Druhý argument bude obsahovat jméno značky, která se vyskytuje v zápisu XPath dotazu. Pořadí těchto značek v příslušném seznamu (stejné číslo řádku) bude definováno dvojicí čísel. Druhé číslo bude udávat, jaká je úroveň zanoření dané značky v XML dokumentu. První číslo udává, kolikátá je to značka v pořadí na dané úrovni. Pro výše uvedený zápis v jazyce XPath zapíšeme tuto trojici faktů:

```
xml(5, 'autor', 2, 3).
xml(5, 'kniha', 1, 2).
xml(5, 'knihy', 1, 1).
```

Ze zápisu uvedených faktů je patrné, že neobsahují textovou informaci v podobě jména autora, který je uveden na pátém řádku zdrojového XML dokumentu. K ukládání těchto informací je nutné definovat další predikáty.

3. Informace z XML dokumentu

V předchozí kapitole definovaný predikát `xml` podchycuje pouze strukturu vstupního XML dokumentu, ale nezahrnuje v něm uložené informace. Mezi tyto informace řadíme názvy atributů značek a jejich hodnoty a také textový obsah uzavřený danou značkou, např. obsah značky `autor`. V našem případě budeme předpokládat, že vstupní XML dokument bude dobře formátovaný a značky nebudou mít smíšený obsah. To znamená, že každá značka bude obsahovat buď textovou informaci, nebo jiné vnořené značky. Nebude dovolena kombinace obou přístupů.

Všechna data XML dokumentu jsou svázána s konkrétním výskytem dané značky. V našem případě se jedná o číslo řádku, na kterém je daná značka nesoucí příslušnou

informaci v XML dokumentu zapsána. Definujme predikát `data`, který bude obsahovat:

- číslo řádku,
- název značky, která obsahuje příslušnou textovou informaci,
- hodnota, tj. obsah této značky.

Řekneme, že Buneman je autorem první knihy takto:

```
data(5, 'autor', 'Buneman').
```

Protože některé značky XML dokumentu mohou mít atributy (a jejich hodnoty), je třeba také tuto informaci podchytit v logickém programu do podoby faktů. Různé značky mohou mít různý počet atributů, nebo také žádný. Proto je třeba každý atribut a jeho hodnotu zapsat do odpovídajícího faktu. Definujme predikát `atribut`, který bude obsahovat:

- číslo řádku,
- název značky, kde je definován daný atribut,
- název atributu, jehož hodnota nás zajímá,
- hodnotu atributu.

Ukázkový XML dokument má definován jediný atribut `rok` ve značce `kniha`, výsledný fakt bude mít tento zápis:

```
atribut(3, 'kniha', 'rok', '2003').
```

I když hodnota atributu `rok` je číslo, obecně jsou všechny informace v XML dokumentu považovány za textové. Z tohoto důvodu je číslo 2003 uzavřeno do apostrofů.

4. Struktura XML dokumentu v pravidlech

Výše definovaný predikát `xml`, který reprezentuje strukturu vstupního XML dokumentu, je možné použít k definici pravidel určujících, která značka je vnořena do které. K získání této informace je třeba definovat pomocný rekurzivní predikát `prunik`. Predikát `prunik` bude hledat průnik dvou množin definovaných dvěma různým čísly řádků vstupního XML dokumentu. Definice predikátu `prunik` je následující:

```
prunik(Radek1, Radek2, Znacka, N, 1) :-
  xml(Radek1, Znacka, N, 1),
  xml(Radek2, Znacka, N, 1),
  Radek1 < Radek2.
```

```
prunik(Radek1, Radek2, Znacka2, N2, P2) :-
  xml(Radek1, Znacka2, N2, P2),
  xml(Radek2, Znacka2, N2, P2),
  P1 := P2 - 1,
  prunik(Radek1, Radek2, Znacka1, N1, P1).
```

První dva argumenty jsou čísla dvou různých řádků vstupního XML souboru a zbylé tři argumenty reprezentují prvek seznamu, který je oběma řádkům společný. Rekurzivní pravidlo respektuje pouze ty řádky, které mají společné prvky na nižších úrovních zanoření. Celý výpočet začíná na kořenové značce, jak ukazuje nerekurzivní pravidlo.

Výsledné dvojice čísel řádků, které obdržíme výpočtem predikátu `prunik` nezaručují, že na těchto řádcích v XML dokumentu jsou zapsány dvě značky, které jsou bezprostředně v sobě vnořené. To je možné říci pouze o těch řádcích, na kterých jsou definovány dvě různé množiny. Ta menší je shodná s průnikem obou množin a ta větší množina obsahuje jeden prvek navíc. Příslušný výpočet zajišťuje predikát `bezprostredne_vnorene_radky`:

```
bezprostredne_vnorene_radky(Radek1, Radek2) :-
  prunik(Radek1, Radek2, Znacka, N, P1),
  P2 := P1 + 1,
  not xml(Radek1, _, _, P2),
  xml(Radek2, _, _, P2),
  P3 := P2 + 1,
  not xml(Radek2, _, _, P3).
```

Zjištění, které dvě dané značky vstupního XML dokumentu jsou vnořené (tj. rozdíl úrovně zanoření obou značek je větší než 1) lze snadno získat aplikací rekurzivního výpočtu, který definuje predikát `vnorene_radky`:

```
vnorene_radky(Radek1, Radek2) :-
  bezprostredne_vnorene_radky(Radek1, Radek2).

vnorene_radky(Radek1, Radek3) :-
  bezprostredne_vnorene_radky(Radek1, Radek2),
  vnorene_radky(Radek2, Radek3).
```

V této kapitole uvedená logická pravidla jsou univerzální, to znamená, že nejsou nikterak závislá na vstupním XML souboru. K formulaci dotazu je ještě nutné definovat predikát `znacka_na_radku`, který poskytne informaci, jaká značka je ve vstupním XML dokumentu zapsána na daném řádku:

```
znacka_na_radku(Radek, Znacka) :-
  xml(Radek, Znacka, N, P1),
  P2 := P1 + 1,
  not xml(Radek, _, _, P2).
```

Predikát `znacka_na_radku` bude ve formulaci dotazu použit pro ty značky vstupního XML dokumentu, které nemají žádný atribut a žádný textový obsah nebo pouze obsahují jiné vnořené značky.

5. Formulace dotazu

Nyní je vše připraveno k formulaci dotazu, který bude definován novým predikátem `autori_knih_2003`. Pro definici tohoto predikátu si vystačíme s jedním pravidlem. Následující dotaz zjišťuje, jací autoři se podíleli na psaní knih, které byly vydány v roce 2003.

```
autori_knih_2003(Autor) :-
  atribut(Radek1, 'kniha', 'rok', '2003'),
  data(Radek2, 'autor', Autor),
  vnorene_radky(Radek1, Radek2).
```

V dotazu průběžně zjišťujeme čísla řádků XML dokumentu, kde jsou zapsány knihy vydané v roce 2003. Predikát `data` nám na jiných řádcích poskytuje jména autorů. Aby výčet autorů byl ten požadovaný, musíme dodat, že značka `autor` je vnořená ve značce `kniha`, jak ukazuje výskyt predikátu `vnorene_radky`.

Dotaz `autori_knih_2003` poskytne množinu autorů (někteří se mohou opakovat), ale není možné určit, kteří autoři se podíleli na psaní které knížky. Definujme proto nový predikát, který bude získávat nejen jména autorů, ale také název knihy, na které se podíleli. Opět nás budou zajímat knihy vydané v roce 2003:

```
knihy_2003_a_jejich_autori(Nazev, Autor) :-
  atribut(Radek1, 'kniha', 'rok', '2003'),
  data(Radek2, 'autor', Autor),
  data(Radek3, 'nazev', Nazev),
  vnorene_radky(Radek1, Radek2),
  vnorene_radky(Radek1, Radek3).
```

Změna v tomto dotazu spočívá pouze v tom, že potřebujeme zjistit na jiném řádku, jaký má daná kniha název. Že se jedná o název a autora/y téže knihy zaručují oba výskyty predikátu `vnorene_radky`.

Třetím dotazem chceme zjistit názvy všech knih, které jsou uvedeny ve vstupním XML dokumentu. Definujme následující predikát `nazev_knih` takto:

```
nazvy_knih(Nazev) :-
  znacka_na_radku(Radek1, 'knihy'),
  data(Radek2, 'nazev', Nazev),
  vnorene_radky(Radek1, Radek2).
```

Všechny knihy, jejichž název je hledán, jsou vnořeny do značky `knihy`. Aby byl tento nadřazený vztah zajištěn, bude použit predikát `znacka_na_radku`, který poskytne číslo řádku, kde je daná značka ve vstupním XML dokumentu zapsána.

Poslední dotaz doplňuje předchozí o rok vydání všech evidovaných knih:

```
nazvy_knih_a_rok(Nazev, Rok) :-
  znacka_na_radku(Radek1, 'knihy'),
  atribut(Radek2, 'kniha', 'rok', Rok),
  data(Radek3, 'nazev', Nazev),
  vnorene_radky(Radek1, Radek2),
  vnorene_radky(Radek2, Radek3).
```

6. Závěr

Tato zpráva ukazuje možnost, jak přepsat XML dokument do podoby logického programu napsaného v jazyce Datalog. Výsledný logický program je univerzální, protože výsledná logická pravidla lze použít na různé XML dokumenty. Dva logické programy se liší pouze množinou faktů, která definuje jak strukturu vstupního XML dokumentu, tak také data v něm uložená.

Navržený postup transformace XML dokumentu na logický program má jisté omezení. Toto omezení se týká XML dokumentu, jehož některé značky mají smíšený obsah. To znamená, že značka obsahuje na stejné úrovni text a jiné vnořené značky. Pro popisovaný transformační postup nesmí vstupní XML dokument obsahovat značky se smíšeným obsahem. Z tohoto důvodu byl mírně upraven z [1] převzatý vstupní XML dokument, ve kterém je osmý řádek zapsán takto:

```
8.       <recenze><em>Výborná</em> kniha.</recenze>
```

V tomto případě značka `recenze` obsahuje jak text tak také vnořenou značku `em`. Takto složenou informaci není možné zaznamenat do podoby příslušných faktů, jak je uvedeno v kapitolách 2 a 3. Popisované omezení není významné, jeho hlavním přínosem je lepší návrh transformace XML dokumentu na logický program, než který je uveden v [1].

V blízké budoucnosti se očekává návrh a implementace nástroje, který využije navržené postupy transformace XML dokumentu na logický program psaný v jazyce Datalog. Transformovaný logický program zpracuje experimentální deduktivní databázový systém [11], který využívá SŘBD Oracle pro vyhodnocování dotazů. Předpokládá se, že navržený systém umožní zpracovat rozsáhlý vstupní XML soubor, kde vyhodnocení položeného dotazu bude vyžadovat výpočet tranzitivního uzávěru, který experimentální deduktivní databázový systém podporuje. Vhodným kandidátem zpracovávaného souboru může být rozsáhlá *The DBLP Computer Science Bibliography* na adrese

`http://dblp.uni-trier.de`, kde autor poskytuje export této databáze do XML formátu.

Literatura

[1] Jesús M. Almendros-Jiménez: *An RDF Query Language based on Logic Programming.* Electronic Notes in Theoretical Science, Volume 200, str. 67-85, 2008.

[2] Edsger W. Dijkstra: *A Note on Two Problems in Connexion with Graphs.* Numerische Mathematik, Volume 1, str. 269-271, 1959.

[3] Moshe Sniedovich: *Dijkstra's algorithm revisited: the dynamic programming connexion.* Journal of Control and Cybernetics, Volume 35(3), str. 599-620, 2006.

[4] Oracle: *Oracle Database PL/SQL User's Guide and Reference 10g Release 2 (10.2).* Manuál společnosti Oracle, www.oracle.com, 2005.

[5] W3C: *OWL Web Ontology Language Reference.* Technický report, www.w3.org, 2004.

[6] W3C: *RDF Vocabulary Description Language 1.0: RDF Schema.* Technický report, www.w3.org, 2004.

[7] W3C: *Resource Description Framework (RDF): Concepts and Abstract Syntax.* Technický report, www.w3.org, 2004.

[8] W3C: *Extensible Markup Language (XML) 1.0 (Fifth Edition).* Technický report, www.w3.org, 2008.

[9] W3C: *XML Path Language (XPath) 2.0.* Technický report, www.w3.org, 2007.

[10] W3C: *XQuery 1.0: An XML Query Language.* Technický report, www.w3.org, 2007.

[11] Martin Zíma: *Experimentální deduktivní databázový systém s neurčitostí.* Disertační práce, Západočeská univerzita v Plzni, 2002.

A. Ukázkový XML dokument

Příloha obsahuje fragment datové kolekce studijních programů a oborů vysokých škol a univerzit v ČR. Z datové kolekce byly vybrány dvě univerzity, z každé univerzity byla vybrána jedna fakulta a z každé fakulty jeden, společný studijní program. Studijní programy obou fakult byly zredukovány na jeden studijní obor.

```
<?xml version="1.0?>
<skoly>
  <programy>
    <program kod="B1801">Informatika</program>
  </programy>
  <obory>
    <obor kkov="1801R008">Obecná informatika</obor>
    <obor kkov="1801R025">Umělá inteligence a zpracování přirozeného jazyka</obor>
  </obory>
  <formy>
    <forma id="bp">bakalářský prezenční</forma>
    <forma id="bk">bakalářský kombinovaný</forma>
  </formy>
  <skola id="MU" nazev="Masarykova univerzita" typ="veřejná">
    <url>www.muni.cz</url>
    <adresa>
      <ulice>Žerotínovo náměstí 9</ulice>
      <mesto>Brno</mesto>
      <psc>601 77</psc>
    </adresa>
    <fakulta id="FI" nazev="Fakulta informatiky">
      <program kod="B1801">
        <obor kkov="1801R025">
          <forma id="bp"/>
        </obor>
      </program>
    </fakulta>
  </skola>
  <skola id="UK" nazev="Univerzita Karlova" typ="veřejná">
    <url>www.cuni.cz</url>
    <adresa>
      <ulice>Ovocný trh 3/5</ulice>
      <mesto>Praha 1</mesto>
      <psc>116 36</psc>
    </adresa>
    <fakulta id="MFF" nazev="Matematicko-fyzikální fakulta">
      <program kod="B1801">
        <obor kkov="1801R008">
          <forma id="bp"/>
          <forma id="bk"/>
        </obor>
      </program>
    </fakulta>
  </skola>
</skoly>
```

B. Výsledný logický program

Tato příloha obsahuje výsledný logický program napsaný v jazyce Datalog, který byl ručně transformován podle postupu navrženého ve zprávě. Tímto postupem byl transformován XML dokument z přílohy A. Logický program je doplněn o jedno pravidlo definující predikát `kde_uci_informatiku`, který reprezentuje vhodný dotaz nad transformovaným XML dokumentem.

```
xml(2, 'skoly', 1, 1).
xml(3, 'programy', 1, 2).
xml(3, 'skoly', 1, 1).
xml(4, 'program', 1, 3).
xml(4, 'programy', 1, 2).
xml(4, 'skoly', 1, 1).
xml(6, 'obory', 2, 2).
xml(6, 'skoly', 1, 1).
xml(7, 'obor', 1, 3).
xml(7, 'obory', 2, 2).
xml(7, 'skoly', 1, 1).
xml(8, 'obor', 2, 3).
xml(8, 'obory', 2, 2).
xml(8, 'skoly', 1, 1).
xml(10, 'formy', 3, 2).
xml(10, 'skoly', 1, 1).
xml(11, 'forma', 1, 3).
xml(11, 'formy', 3, 2).
xml(11, 'skoly', 1, 1).
xml(12, 'forma', 2, 3).
xml(12, 'formy', 3, 2).
xml(12, 'skoly', 1, 1).
xml(14, 'skola', 4, 2).
xml(14, 'skoly', 1, 1).
xml(15, 'url', 1, 3).
xml(15, 'skola', 4, 2).
xml(15, 'skoly', 1, 1).
xml(16, 'adresa', 2, 3).
xml(16, 'skola', 4, 2).
xml(16, 'skoly', 1, 1).
xml(17, 'ulice', 1, 4).
xml(17, 'adresa', 2, 3).
xml(17, 'skola', 4, 2).
xml(17, 'skoly', 1, 1).
xml(18, 'mesto', 2, 4).
xml(18, 'adresa', 2, 3).
xml(18, 'skola', 4, 2).
xml(18, 'skoly', 1, 1).
xml(19, 'psc', 3, 4).
xml(19, 'adresa', 2, 3).
xml(19, 'skola', 4, 2).
xml(19, 'skoly', 1, 1).
xml(21, 'fakulta', 3, 3).
xml(21, 'skola', 4, 2).
xml(21, 'skoly', 1, 1).
xml(22, 'program', 1, 4).
xml(22, 'fakulta', 3, 3).
xml(22, 'skola', 4, 2).
xml(22, 'skoly', 1, 1).
xml(23, 'obor', 1, 5).
xml(23, 'program', 1, 4).
xml(23, 'fakulta', 3, 3).
xml(23, 'skola', 4, 2).
xml(23, 'skoly', 1, 1).
xml(24, 'forma', 1, 6).
xml(24, 'obor', 1, 5).
xml(24, 'program', 1, 4).
```

```
xml(24, 'fakulta', 3, 3).
xml(24, 'skola', 4, 2).
xml(24, 'skoly', 1, 1).
xml(29, 'skola', 5, 2).
xml(29, 'skoly', 1, 1).
xml(30, 'url', 1, 3).
xml(30, 'skola', 5, 2).
xml(30, 'skoly', 1, 1).
xml(31, 'adresa', 2, 3).
xml(31, 'skola', 5, 2).
xml(31, 'skoly', 1, 1).
xml(32, 'ulice', 1, 4).
xml(32, 'adresa', 2, 3).
xml(32, 'skola', 5, 2).
xml(32, 'skoly', 1, 1).
xml(33, 'mesto', 2, 4).
xml(33, 'adresa', 2, 3).
xml(33, 'skola', 5, 2).
xml(33, 'skoly', 1, 1).
xml(34, 'psc', 3, 4).
xml(34, 'adresa', 2, 3).
xml(34, 'skola', 5, 2).
xml(34, 'skoly', 1, 1).
xml(36, 'fakulta', 3, 3).
xml(36, 'skola', 5, 2).
xml(36, 'skoly', 1, 1).
xml(37, 'program', 1, 4).
xml(37, 'fakulta', 3, 3).
xml(37, 'skola', 5, 2).
xml(37, 'skoly', 1, 1).
xml(38, 'obor', 1, 5).
xml(38, 'program', 1, 4).
xml(38, 'fakulta', 3, 3).
xml(38, 'skola', 5, 2).
xml(38, 'skoly', 1, 1).
xml(39, 'forma', 1, 6).
xml(39, 'obor', 1, 5).
xml(39, 'program', 1, 4).
xml(39, 'fakulta', 3, 3).
xml(39, 'skola', 5, 2).
xml(39, 'skoly', 1, 1).
xml(40, 'forma', 2, 6).
xml(40, 'obor', 1, 5).
xml(40, 'program', 1, 4).
xml(40, 'fakulta', 3, 3).
xml(40, 'skola', 5, 2).
xml(40, 'skoly', 1, 1).

data(4, 'program', 'Informatika').
data(7, 'obor', 'Obecná informatika').
data(8, 'obor', 'Umělá inteligence a zpracování přirozeného jazyka').
data(11, 'forma', 'bakalářský prezenční').
data(12, 'forma', 'bakalářský kombinovaný').
data(15, 'url', 'www.muni.cz').
data(17, 'ulice', 'Žerotínovo náměstí 9').
data(18, 'mesto', 'Brno').
data(19, 'psc', '601 77').
data(30, 'url', 'www.cuni.cz').
data(32, 'ulice', 'Ovocný trh 3/5').
data(33, 'mesto', 'Praha 1').
data(34, 'psc', '116 36').

atribut(4, 'program', 'kod', 'B1801').
atribut(7, 'obor', 'kkov', '1801R008').
atribut(8, 'obor', 'kkov', '1801R025').
atribut(11, 'forma', 'id', 'bp').
atribut(12, 'forma', 'id', 'bk').
atribut(14, 'skola', 'id', 'MU').
atribut(14, 'skola', 'nazev', 'Masarykova univerzita').
atribut(14, 'skola', 'typ', 'veřejná').
atribut(21, 'fakulta', 'id', 'FI').
```

```
atribut(21, 'fakulta', 'nazev', 'Fakulta informatiky').
atribut(22, 'program', 'kod', 'B1801').
atribut(23, 'obor', 'kkov', '1801R025').
atribut(24, 'forma', 'id', 'bp').
atribut(29, 'skola', 'id', 'UK').
atribut(29, 'skola', 'nazev', 'Univerzita Karlova').
atribut(29, 'skola', 'typ', 'veřejná').
atribut(36, 'fakulta', 'id', 'MFF').
atribut(36, 'fakulta', 'nazev', 'Matematicko-fyzikální fakulta').
atribut(37, 'program', 'kod', 'B1801').
atribut(38, 'obor', 'kkov', '1801R008').
atribut(39, 'forma', 'id', 'bp').
atribut(40, 'forma', 'id', 'bk').


prunik(Radek1, Radek2, Znacka, N, 1) :-
  xml(Radek1, Znacka, N, 1),
  xml(Radek2, Znacka, N, 1),
  Radek1 < Radek2.

prunik(Radek1, Radek2, Znacka2, N2, P2) :-
  xml(Radek1, Znacka2, N2, P2),
  xml(Radek2, Znacka2, N2, P2),
  P1 := P2 - 1,
  prunik(Radek1, Radek2, Znacka1, N1, P1).

bezprostredne_vnorene_radky(Radek1, Radek2) :-
  prunik(Radek1, Radek2, Znacka, N, P1),
  P2 := P1 + 1,
  not xml(Radek1, _, _, P2),
  xml(Radek2, _, _, P2),
  P3 := P2 + 1,
  not xml(Radek2, _, _, P3).

vnorene_radky(Radek1, Radek2) :-
  bezprostredne_vnorene_radky(Radek1, Radek2).

vnorene_radky(Radek1, Radek3) :-
  bezprostredne_vnorene_radky(Radek1, Radek2),
  vnorene_radky(Radek2, Radek3).

znacka_na_radku(Radek, Znacka) :-
  xml(Radek, Znacka, N, P1),
  P2 := P1 + 1,
  not xml(Radek, _, _, P2).


kde_uci_informatiku(Mesto) :-
  data(Radek1, 'program', 'Informatika'),
  data(Radek2, 'mesto', Mesto),
  atribut(Radek1, 'program', 'kod', Kod),
  atribut(Radek3, 'program', 'kod', Kod),
  znacka_na_radku(Radek4, 'skola'),
  znacka_na_radku(Radek5, 'skoly'),
  vnorene_radky(Radek4, Radek2),
  vnorene_radky(Radek4, Radek3),
  vnorene_radky(Radek5, Radek1),
  vnorene_radky(Radek5, Radek4).
```