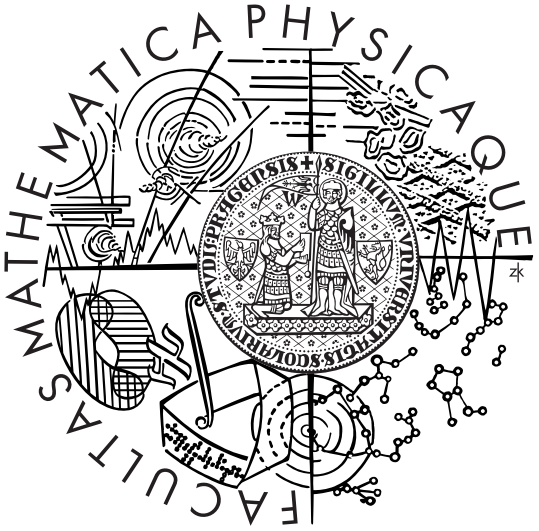
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

Diplomová práce



Jakub Bystroň

Konceptuální modelování neurochirurgických dat

Katedra softwarového inženýrství
Vedoucí diplomové práce: *RNDr. Antonín Říha, CSc.*
Studijní program: *Informatika, Softwarové systémy*

Chtěl bych poděkovat Donaldu E. Knuthovi za typografický systém TEX, autorům projektu PSTricks za soubor maker pro sázení obrázků, Prof. RNDr. Janě Zvárové, DrSc. za vstřícnost z pozice vedoucího oddělení medicínské informatiky Ústavu informatiky Akademie věd ČR a zvláště RNDr. Antonínu Říhovi, CSc. jako vedoucímu diplomové práce.

Prohlašuji, že jsem svou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Karviné dne 10. srpna 2006 Jakub Bystroň

Název práce: Konceptuální modelování neurochirurgických dat
Autor: Jakub Bystroň
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí diplomové práce: RNDr. Antonín Říha, CSc.
e-mail vedoucího: riha@euromise.cz

Abstrakt

V této práci je představen nový minimalistický konceptuální AC-model určený pro modelování hierarchických dat. Při budování modelu syntaxe vycházíme z toho, co máme na logické úrovni k dispozici, tedy z regulárních stromových gramatik. AC-model se skládá ze dvou částí – archetypového modelu (A-model) a sady omezujících podmínek (C-model) vyjádřených pomocí logiky prvního řádu. Dokážeme, že A-model je schopen namodelovat libovolnou regulární stromovou gramatiku. Dále dokážeme, že problém validace podle AC-modelu je NP-těžký a že problém existence validní instance konkrétního AC-modelu je nerozhodnutelný. Součástí práce je rovněž pilotní implementace systému pro validování XML dokumentů na základě C-modelu.

Klíčová slova: konceptuální modelování, regulární stromová gramatika, XML, logika prvního řádu, omezující podmínky

Title: Conceptual modeling of neurosurgical data
Author: Jakub Bystroň
Department: Department of Software Engineering
Supervisor: RNDr. Antonín Říha, CSc.
Supervisor's e-mail address: riha@euromise.cz

Abstract

We present a new minimalistic conceptual model for hierarchical data called AC-model. Syntax part of this model is based upon the regular tree grammar theory. AC-model is composed of two parts – Archetype model and Constraint model. A-model is intended to model arbitrary regular tree grammar, C-model adds a first order logic constraints. We also prove that validity problem for AC-model is NP-hard and conformance problem is undecidable. Along with this thesis C-model based validator of XML documents' semantics is presented.

Keywords: conceptual model, regular tree grammar, XML, first order logic, constraints

Obsah

Předmluva

V celé práci budeme snažit podkládat všechna zásadní rozhodnutí objektivními matematickými fakty. Na některých místech se vyskytují mé subjektivní názory na tu kterou problematiku. Tyto subjektivní názory budou striktně oddělovány od ostatního textu příkladně uvozeními typu „*Dle mého subjektivního názoru...*“ nebo odlišnou sazbou písma. V práci se také budeme snažit u použitých technologií uvádět pouze jejich stručnou specifikaci s odkazem na literaturu a uměle tak nezvětšovat objem práce. K jazykové části uveďme striktní psaní čárky před nebo, pokud se jedná o jakýkoli vylučovací vztah. Konkrétně například ve větě „*Funkce je v bodě x spojitá, nebo nespojitá*“ budeme, pokud nemohou nastat obě možnosti zároveň, vždy psát čárku před nebo. Úroveň formalismu je v různých kapitolách různá a odpovídá aktuální potřebě. Značení je běžné pro matematické texty sázené v systému TEX.

Značení

- Symbol $\mathbb{N}$ označuje přirozená čísla, $\mathbb{Z}$ celá čísla, $\mathbb{Q}$ racionální čísla, $\mathbb{R}$ reálná čísla, $\mathbb{C}$ komplexní čísla
- Zápis $f\colon A \to B$ označuje funkci f z množiny A do množiny B
- Je-li X množina, pak symbolem 2^X budeme označovat potenční množinu množiny X.
- „Veliké“ třídy objektů budeme označovat kaligrafickými písmeny, např. $\mathcal{R}, \mathcal{F}$ apod.
- Je-li $n \in \mathbb{N}$, pak symbolem $[n]$ označujeme množinu přirozených čísel $\{1, 2, \ldots, n\}$.

Kapitola 1

Úvod

„Ve vědách není žádné jistoty tam, kde nelze užít některé matematické vědy, ani v tom, co nemá sepětí s matematikou.“

– Leonardo da Vinci

1.1 Cíle diplomové práce

V této diplomové práci se budeme zabývat diskusí principů tvorby konceptuálních modelů pro systémy založené na datech ve formátu XML s vazbou na systémy pro vedení elektronického zdravotního záznamu. Tyto informační systémy (podobně jako v jiných oblastech použití) umožňují skladování zdravotních karet v přísně strukturovaných formátech, jakož i jejich opětovné efektivní získávání. V současné době existuje několik pokusů o řešení tohoto úkolu, ve světě za všechny jmenujme například projekt *openEHR*.[1] Průnik tohoto řešení s řešením prezentovaným dále v textu je ale víceméně jen v tom, že se jedná o tzv. *schema–based* přístup.

Ideální stav zdravotnictví vzhledem ke zdravotním záznamům by byl takový, že by každý pacient měl v národním zdravotním systému uložen svůj zdravotní záznam. Jeho části by byly přístupné jednotlivým lékařským zařízením, ve kterých by byl pacient léčen. Centrálním skladováním kompletního zdravotního záznamu pacientů by se jednoduše vyřešily problémy s cestováním zdravotních karet při změně ošetřujících lékařů, nedostupností základní anamnézy např. při dopravních nehodách, mnohdy i s kontraindikacemi indikovaných farmak vzniknuvších z neznalosti komplexní léčby pacienta.

[1] `http://www.openehr.org`

1.2 Elektronický medicínský a zdravotní záznam

Terminologie je v dané oblasti výzkumu nejspíše díky rychlosti vývoje značně nejednotná. Může to být dáno i nutností spolupráce při vývoji s lidmi nemajícími informatické nebo matematické vzdělání (zejména s lékaři). Toto nicméně není žádným zásadním problémem. V průběhu práce na tomto textu jsem poměrně s lítostí zjišťoval také značně diferencovanou úroveň obsahu různých vědeckých článků, a bylo proto zpočátku poněkud obtížné z něčeho čerpat. Nyní již přistupme k základním definicím, které ale mají pro další práci poměrně okrajový význam, protože danou úlohu budeme řešit v mnohem obecnějším kontextu.

Nejprve ujasněme terminologii, jež bude použita v tomto textu. V literatuře se vyskytuje několik běžně používaných termínů, mnohdy se různě zaměňují. My tyto termíny budeme používat podle následujících definic.

Definice 1 (Elektronický medicínský záznam)**.** *Elektronický medicínský záznam (Electronic Medical Record – EMR) je strukturovaný počítačem zpracovatelný soubor informací o daném pacientovi v jedné klinické oblasti. EMR zejména obsahuje osobní informace, anamnézy, zápisy všech lékařských zpráv a indikovaných farmak, historii operací, klinických a laboratorních vyšetření, snímků (RTG, CT, MR) apod.*

Definice 2 (Elektronický zdravotní záznam)**.** *Elektronický zdravotní záznam (Electronic Health Record – EHR) je celoživotním souborem všech pacientových EMR.*

Elektronický zdravotní záznam je tedy jakýmsi sjednocením všech zdravotních karet, které jsou o pacientovy vedeny ve všech lékařských zařízeních, ve kterých je daný pacient léčen. To, že je záznam navíc uchován ve strukturované elektronické podobě, umožňuje efektivní algoritmické vyhledávání požadovaných informací. Jen poznamenejme, že v dalším textu budeme obvykle u obou definovaných termínů vynechávat slovo *elektronický*, které je v kontextu této práce víceméně implicitně přítomno. Jelikož jsou osobní zdravotní data velice citlivá, je nutno mít při návrhu systému pro vedení EHR na zřeteli rovněž bezpečnostní hlediska a vytvořit propracovaný systém autentizací a autorizací.

1.3 EHR systémy – možné přístupy

Systémy pro vedení elektronických zdravotních záznamů pacientů je možno z informatického hlediska navrhnout dvěma základními způsoby:

Fixní návrh

Tento způsob volí především většina zakázkových komerčních aplikací. V případě zakázkového vývoje bývá předem známo schéma ukládaných informací. Můžeme proto provést konkrétní návrh databázového schématu, které nejlépe vystihne požadavky zadavatele. V případě, že se v průběhu používání hotové aplikace změní struktura ukládaných dat, kontaktuje obvykle zadavatel projektu softwarovou firmu a ta provede upgrade systému, což obvykle zahrnuje změnu databázového schématu, často změnu komunikačních protokolů a změnu klientských formulářů. Nezřídka se stává, že víceméně prostě formulovatelný požadavek zadavatele implikuje hrubé zásahy do konstrukce celého informačního systému.

- *Výhody:* Výhodou tohoto přístupu je vysoká efektivita databázového řešení. Při znalosti ER schématu databáze jsme schopni navrhnout relační tabulky tak, aby byly typické dotazy uživatele zpracovávány velice efektivně.
- *Nevýhody:* Nevýhody fixního návrhu jsou zřejmé. Sebemenší změnu struktury dat je nutno reflektovat změnou aplikace. Tento proces je časově velice náročný, a tedy i drahý. V případě, že se neustále mění „schéma“ ukládaných informací (jako je tomu právě v medicíně díky rychlému vývoji), je nutný nepřetržitý kontakt uživatele s autorem systému.

Polymorfní návrh (schema–based)

Umožňuje ukládat víceméně „libovolně“ strukturovaná data. V průběhu používání systému již není nutný žádný zásah programátora. Uživatel si sám definuje pouze „schéma“ ukládaných dat a omezující podmínky. To, jakým způsobem se data budou ukládat v databázovém systému, nebo jak se budou dostávat až ke klientskému rozhraní, bude zabezpečovat systém zcela automaticky.

- *Výhody:* Zásadní výhodou tohoto řešení je flexibilita systému. V tomto případě není nutno vytvářet žádné obvyklé typizované systémy např. pro obvodní lékaře, oftalmology, internisty apod. Systém existuje jeden. Každý specialista bude mít jen odlišně definované schéma dat, které používá při výkonu své lékařské praxe.
- *Nevýhody:* Jedinou vážnou nevýhodou tohoto systému je náročnost fáze vývoje. Navrhnout systém tak, aby umožňoval editace, serializaci a následné zobrazení libovolně pevně strukturovaných dat, je netriviální. Další možnou nevýhodou je případná o něco menší efektivita tohoto přístupu nežli je tomu u řešení „na míru“. V případě použití efektivních algoritmů by ale neměla být determinující.

Z výše uvedeného je zřejmé, že chceme-li docílit vyšší míry používání informačních technologií při správě lekařských dat pacientů, je nutno jít cestou *polymorfního řešení*. Je totiž maximálně neefektivní vytvořit desítky fixních systémů pro všechny typy lékařských zařízení a ještě tyto produkty následně stále spravovat.

Ukážeme, že schéma ukládaných dat je jediným vstupem pro vytvoření konkrétní instance polymorfního informačního systému. Jinými slovy, že celý informační systém (zejména jeho perzistenční i prezentační vrstvu) je možno jednoduše vygenerovat algoritmicky jen na základě vstupního konceptuálního modelu dat.

K tomu, abychom mohli ale v budoucnu takovýto polymorfní systém vytvořit, je primárně zapotřebí mít k dispozici konceptuální model umožňující takovéto systémy modelovat. Prezentace vlastního konceptuálního AC-modelu jakož i stručný komentář k modelům existujícím následuje dále v textu.

1.4 Pohled uživatele

Zamysleme se nyní *z pohledu uživatele* nad úlohou formalizace schématu medicínského záznamu daného pacienta, konkrétně nad strukturou dat, typy informací a (většinou myšlených) omezujících podmínkách na data a strukturu v aktuálně používaných zdravotních kartách. To vše v této krátké kapitole provedeme na intuitivní úrovni bez zbytečných formalismů. Zcela formálně budeme budovat až teorii AC-modelu.

Datové typy

Ukládat budeme potřebovat libovolné datové typy běžné z databází jako např. čísla (různé obory), řetězce, veličiny (číslo a jednotka), jakož i multimediální data jako obrázky a video (EKG, RTG, USG, CT, MR) nebo zvuk. V tomto bodu se polymorfní a fixní EHR systémy nijak neliší. S datovými typy nebude žádný problém, protože výše uvedené typy nám umožňují ukládat víceméně všechny běžně používané databázové systémy.

Struktura dat v medicínském záznamu

Informace o jednom konkrétním pacientovi mají v aktuálně používaných „papírových“ zdravotních kartách jednoduchou grafovou strukturu. U pacienta máme uloženy osobní informace, ty obsahují např. adresu, ta se skládá z ulice, čísla popisného a města atp. Například u ambulantních lékařů je pak v kartě typicky už jen chronologicky seřazený seznam lékařských zpráv s případnými odkazy na laboratorní nebo klinická vyšetření. Obecně tedy dostáváme *orientovaný acyklický graf*, v některých případech dokonce strom. To, že samozřejmě

nemusíme dostat vždy strom, ukazuje např. situace, kdy několik lékařských zpráv odkazuje na jedno konkrétní vyšetření.

My ale půjdeme dále. Určitě bychom si přáli, abychom např. v sekci „rodinná anamnéza (RA)“ měli i odkazy na rodiče daného pacienta. Tyto informace v běžně používaných zdravotních kartách typicky v současné době uvedeny nejsou. Odkazy nám samozřejmě mohou vytvořit v grafu cykly.

Analýzou běžně používaných hierarchických konceptuálních modelů (viz dále) zjistíme, že se většina datových položek a odkazů v konceptuálním modelu nevyskytuje samostatně, nýbrž jsou seskupovány do „tříd“. Třídy reprezentujme pomocí grafů podle těchto definic:

Definice 3 (Atributy třídy)**.** *Pro atributy mějme tyto definice:*

(i) Atribut je dvojice $(název, obsah)$*.*

(ii) Obsah atributu je buď jednoduchý, nebo složený.

(iii) Jednoduchý obsah je buď datový typ, nebo reference na třídu.

(iv) Složený obsah je obecně konečná posloupnost atributů.

Definice 4 (Graf atributu)**.** *Mějme atribut* a*. Pokud je obsah atributu* a *jednoduchý, reprezentujícím grafem je pouze jeden kořenový vrchol. V případě, že obsah atributu* a *je složený a atribut* a *obsahuje atributy* $b_1, b_2, \ldots, b_N$*, reprezentujícím grafem je graf tvořený kořenovým vrcholem* x*, ze kterého vedou orientované hrany do kořenových vrcholů podgrafů* $T_1, T_2, \ldots, T_N$*, kde* $T_i, i \in [N]$ *je graf atributu* b_i*.*

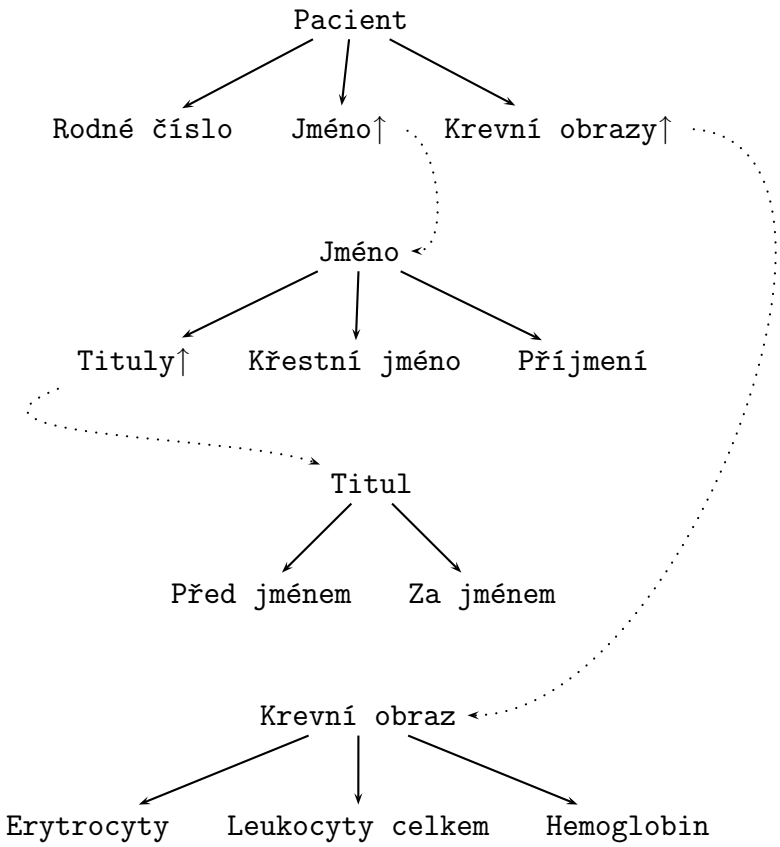
Definice 5 (Graf třídy)**.** *Mějme třídu* X *obsahující atributy* $a_1, a_2, \ldots, a_N$*. Tuto třídu budeme reprezentovat grafem s jedním kořenovým vrcholem* x*, ze kterého povedou orientované hrany do všech kořenových vrcholů podgrafů reprezentujících jednotlivé atributy.*

Třídy provázané referenčními hranami jsou jakýmsi minimální konceptuálním metamodelem. Vidíme základní prvky: hierarchie atributů, zapouzdření těchto hierarchií do tříd a reference mezi třídami.

Uvažujme tento jednoduchý konkrétní model struktury dat:

- Třída „**`Jméno`**" je tvořena tituly, křestním jménem a příjmením.
- Třída „**`Vyšetření krevního obrazu`**" obsahuje např. jen naměřené hodnoty obsahu hemoglobinu, erytrocytů, leukocytů celkem.
- Třída „**`Pacient`**" obsahuje rodné číslo a poté už jen odkaz na jméno a na seznam odkazů na vyšetření krevního obrazu.

Na obrázku 1.1 vidíme 4 grafy tříd **`Pacient`**, **`Jméno`**, **`Tituly`**, **`Krevní obraz`**. Referenční vrcholy jsou označeny šipkou, tečkované hrany odpovídají referencím.



Obrázek 1.1: Provázání tříd

Omezující podmínky

Samostatnou kapitolou při návrhu schématu konkrétní instance informačního systému je specifikování seznamu omezujících podmínek. Omezující podmínky nechť jsou buď globální – platné pro schéma jako celek, nebo lokální pro každou třídu nebo atribut.

Budeme požadovat, abychom mohli definovat různé omezující podmínky jednak na hodnoty atributů (typicky restrikce oboru hodnot datového typu) a jednak také na strukturu a vztahy mezi atributy navzájem.

Příkladem mohou být následující podmínky:

Omezující podmínky na datové typy

U atributů s jednoduchým typovým obsahem budeme přirozeně požadovat existenci omezujících podmínek přiřazujících obor hodnot toho kterého datového typu.

Omezující podmínka	Příklad
Celočíselný interval	Celé číslo od 2 do 20
Reálný interval	Pravděpodobnost $\in [0, 1]$
Řetězec odpovídající regulárnímu výrazu	*neuro*$*$
Výčtový typ	Hemisfera $\in \{\text{Levá}, \text{Pravá}\}$

Tabulka 1.1: Omezující podmínky na datové typy

Omezující podmínky na obsah a strukturu

V případě omezujících podmínek (OP) na obsah (sémantiku) a strukturu (syntax) definujeme složitější vztahy v rámci tříd i mezi nimi. V této sekci definujeme 3 typy rozšířených omezujících podmínek.

Typ	OP	Příklad
Syntax	Kardinality	Atribut *jméno* má alespoň jeden podatribut *křestní* a právě jeden podatribut *příjmení*
Sémantika	Vztahy mezi atributy	Koncové datum hospitalizace je chronologicky větší nebo rovno počátku hospitalizace
Sm $\rightarrow$ Sx	Viditelnost podmnožiny atributů třídy v závislosti na platnosti podmínky	Reference na třídu `Léčba` je přístupná jen pokud je hodnota atributu `diabetik = true`

Tabulka 1.2: Omezující podmínky na obsah a strukturu dat

Algoritmické omezující podmínky

Každá z výše uvedených omezujících podmínek má pouze jeden speciální atomický účel. Jistě by bylo nanejvýše efektní, kdybychom mohli definovat validační omezující podmínky například jako libovolnou boolovskou funkci.

Stačí vyřešit dva problémy. Za prvé musíme zvážit volbu jazyka, ve kterém bude možno takové funkce konstruovat, a za druhé způsob odkazování z dané funkce na validovaná data, resp. vyřešit definiční obor dané funkce.

1.4.1 Shrnutí

V této kapitole jsme si stručně a *neformálně* stanovili základní požadavky na konceptuální model, který by byl vhodný pro polymorfní medicínský infor-

mační systém. Budeme požadovat hierarchickou strukturu a schopnost vyjádřit alespoň výše uvedené omezující podmínky.

Kapitola 2

Konceptální modely pro XML

„Everything should be made as simple as possible, but not simpler.“

– Albert Einstein

Terminologie. V oblasti konceptuálního modelování se používají běžně termíny jako schémata a modely ve víceméně synonymním vztahu. Pokud potřebujeme schémata formalizovat např. do textové podoby, používá se pojem konceptuální jazyky nebo jazyky pro zápis konceptuálních modelů. Modelování je tedy proces vytváření modelů. Model je tedy slovem konceptuálního jazyka.

Právě slovo *model* se ale mnohdy používá nesprávně. Model je konkrétní zástupce nějaké třídy objektů, a tedy např. „E-R model“ je nějaký konkrétní model splňující všechny požadavky, jaké musí E-R model splňovat. Příkladně v logice se modelem teorie T s jazykem L míní libovolná realizace jazyka L, která splňuje všechny axiomy teorie T. Bylo by určitě dobré používat slovo *model* v tomto smyslu. Je tedy speciálně nemožno říkat „E-R je nejznámnějším konceptuálním modelem“ nebo „Musíme vyvinout nový konceptuální model pro XML“ a myslet tím nějaký nový jazyk, ve kterém se mohou modely vytvářet. Tento pojmový non-sense se v literatuře vyskytuje. Správnější by určitě bylo rozlišovat hierarchii

$$\textit{Model} \in \textit{Konceptuální jazyk} \in \textit{Konceptuální jazyky},$$
$$\textit{Konceptuální jazyky pro XML} \subset \textit{Konceptuální jazyky}.$$

Po úvaze jsem se ale nakonec rozhodl na nevhodnost této terminologie jen upozornit a více ji dále nenabourávat.

Problematika konceptuálního modelování byla v minulosti velice intenzivně studována. Zejmená ale pouze v kontextu relačních databázových systémů, pro které byla navržena všeobecně dobře známá ER schémata. Naopak modelování systémů založených na hierarchických datech ve formátu XML je zatím poměrně neprozkoumanou oblastí a výzkumu je věnována spousta úsilí.

2.1 ER schémata

Konceptuální modelování pomocí ER schémat bylo uvedeno pracemi P. Chena v 70. letech. Základem jsou tzv. *entity* a *relace* mezi nimi. Relace mohou mít libovolnou aritu. Entity i relace mohou mít ještě asociovány doprovodné atributy.

Jak již bylo řečeno, ER schémata jsou určena hlavně pro modelování databázových schémat relačních databází. Existuje mnoho známých algoritmů pro překlad z ER schémat do relačních schémat. ER schémata přicházejí se dvěma základními typy omezujících podmínek. Jednak je jsou to tzv. *klíče* a jednak tzv. *kardinality vztahů* (krátce též jen kardinality, pokud bude jasné, že máme na mysli kardinality vztahů).

Nechť S je ER schéma, D je libovolná pevná databázová implementace schématu S včetně obsahu. Je-li E entita schématu S, pak jako $I(E)$ označme množinu instancí entity E v databázi D.

Definice 6 (Klíč)**.** *Je-li E entita, pak její atribut K_E je klíčem, pokud existuje bijekce mezi $I(E)$ a $\{k : \exists o \in I(E), K_E(o) = k\}$.*

Jen poznamenejme, že klíče mohou být i složené (množina atributů je klíčem). V tomto případě lze výše uvedenou definici analogicky upravit.

Definice 7 (Složený klíč)**.** *Je-li E entita, množina atributů $\{A^1_E, A^2_E, \ldots, A^n_E\}$ je klíčem, pokud existuje bijekce mezi $I(E)$ a $\{k = (k_1, k_2, \ldots, k_n) : \exists o \in I(E), A^i_E(o) = k_i\}$.*

Kardinalita vztahu je definována pro entitu a relaci podle této definice.

Definice 8 (Kardinalita)**.** *Nechť E je entita schématu S a R je relace. Pak kardinalita vztahu entity E a relace R je $[a, b]$[1], $a, b \in \mathbb{N}_0 \cup \{+\infty\}, a \leq b$, pokud počet výskytů každého objektu $o \in I(E)$ v projekci instance relace R na typ E je nejméně a a nejvýše b.*

Pro naše účely jsou ale ER schémata nevhodná. Jedním z mnoha omezení je např. nemožnost přímé specifikace pořadí vektorů v relaci (relace jakožto podmnožina kartézského součinu je samozřejmě z principu neuspořádaná), omezené možnosti při vyjadřování kardinalit relací arity vyšší než 2, nemožnost definování, že objekt buď $o \in I(R_1)[E]$, anebo $o \in I(R_2)[E]$[2], kde R_1, R_2 jsou dvě různé relace (`choice` XML schémat).

Oproti XML má i logický model ER schémat, tedy SQL, těžkopádnější dotazování než například XPath nebo XQuery v případě XML. Pro srovnání

[1] Častěji bývá použito značení (a, b). Vzhledem k tomu, že ale čísla a, b definují celočíselný interval mezi a a b včetně, přijde mi matematicky konzistentnější označení hranatými závorkami.

[2] $R[E]$ je projekce na typ E (resp. na souřadnici odpovídající typu E).

uveďme klasický příklad *Člověk → Potomek* s dotazem: *„Jména všech potomků daného člověka X.“*

Nechť mají příslušná schémata tento tvar

Relační schéma	*DTD*
Clovek (**jmeno**, predek)	Clovek → (jmeno, Clovek*)

Tabulka 2.1: Schémata *Člověk → Potomek*

Zde je dotaz *„Jména všech potomků Jakuba“* v SQL

```
WITH Potomek(jmeno)
AS
(
  SELECT jmeno FROM T
    WHERE predek = 'Jakub'
  UNION ALL
  SELECT P.jmeno
    FROM Potomek Q, T P
    WHERE P.predek=Q.jmeno
)
SELECT * FROM Potomek
```

a zde v XPath

```
Clovek[jmeno='Jakub']//Clovek/jmeno
```

Vůbec na tomto místě nebudeme srovnávat asymptotické složitosti dotazování v SQL a v XPath (nebo dokonce XQuery). V této ukázce rekurzivního dotazu nám šlo čistě jen o ukázku jednoduchosti zápisu tohoto konkrétního dotazu v jazyce XPath oproti těžkopádnosti v SQL.

2.2 Konceptální jazyk XML dat

XML schémata samy o sobě nejsou konceptální modely. Jejich primárním úkolem je validace dokumentů. K rozvoji modelování XML systémů je tedy nutné, navrhnout abstraktní jazyk pro modelování na konceptuální úrovni. Jazyk pro zápis konceptálního modelu by přitom neměl být novým jazykem pro zápis XML schémat, nýbrž co nejvíce abstraktním zápisem vztahů mezi objekty s hierarchickými vztahy, referencemi apod., tedy vlastnostmi blízkými XML. Důraz musí být kladen na vyjadřovací sílu, musí být rovněž formalizována syntax jazyka. Okrajový je grafický zápis. Obrázky složené z třiceti druhů geometrických tvarů, různých typů orámování a čar nijak nepřispívají kvalitě

jazyka (E-R diagramy, UML). Nutné pak také bude spolu s tímto konceptuálním jazykem uvést i algoritmus pro jeho překlad do konkrétního jazyka pro zápis XML schémat.

Klíčovými body pro návrh konceptuálního modelu pro XML jsou tedy tyto:

1. *Formální základ* – primární je formálně definovaný jazyk, formálně dokazatelná expresivita. Bez formalizace jazyka, ve kterém je možno modely zapisovat není možno o jazyce nebo konkrétních modelech dokazovat jakákoli tvrzení. Je vybudována brilantní matematická teorie, konceptuální jazyky by se ji měly snažit využít a nerazit si vlastní „intuitivní“ cestu.

2. *Hierarchická struktura objektů, reference mezi objekty* – jazyk musí obsáhnout 2 základní konstrukce XML a to jednak specifikaci vztahu *otec* $\rightarrow$ *syn* a jednak mechanismus odkazování na jiné XML dokumenty.

3. *Systém omezujících podmínek* – naprosto zásadním bodem je specifikace omezujících podmínek, jež specifikují sémantiku obsahu XML dokumentů. Například unikátnost obsahu dané množiny elementů, omezení na datové typy, přislušnost obsahu dané množiny elementů (atributů) nějaké relaci (např. $\leq$) apod.

4. *Překlad do konkrétního XML schéma jazyka* – musí existovat algoritmus, jenž hotový konceptuální model přeloží na konkrétní regulární stromovou gramatiku a poté do nějakého konkrétního XML schéma jazyka. V žádném případě není nutno, jak je často v současné literatuře nesprávně uváděno, bylo možno konkrétní konceptuální model přeložit do většiny aktuálně používaných XML schéma jazyků. Tyto jazyky se liší svojí expresivitou. Pokud je konceptuální jazyk pokročilý a umožňuje modelovat i např. nejednotypové konstrukce (viz dále), není možno požadovat, aby byl přeložitelný např. do jazyka W3C XML Schema nebo dokonce do DTD. Přitom by model neměl generovat jen nějaký omezený typ regulárních stromových gramatik (např. jen lokální).

5. *Nedeterministický obsah* – rozhodně nebudeme požadovat, aby šly vytvořit jen modely vedoucí na deterministické gramatiky. Tento bod často přidávají autoři používající na logické úrovni jazyk W3C XML Schema, jenž není schopen vyjádřit všechny nedeterministické gramatiky.

6. *Data-centric vs. document-centric XML* – pokud má být konceptuální jazyk univerzální, musí umožňovat modelovat jak datově tak i dokumentově orientované XML. Základní rozdíl mezi těmito třídami je ve větší míře strukturovanosti, typické nepřítomnosti složeného obsahu na straně datových dokumentů, a menší míře strukturovanosti, nutnosti fixace pořadí podelementů a naprosto typické přítomnosti složeného obsahu na straně dokumentově orientovaných XML dokumentů.

V případě jazyka určeného primárně k modelování datových dokumentů je třeba zvážit oddělení modelování elementů a atributů. Jediný rozdíl mezi atributy a elementy v XML je ten, že atributy nemohou mít „potomky“, jsou z principu množina a jako takové jsou tedy neuspořádány, od každého jména existuje nejvýše jeden a mají pouze jednoduchý obsah. U datově orientovaných dokumentů nám typicky nezáleží na tom, je-li něco v XML dokumentu jako atribut nebo element s jednoduchým obsahem. Na konceptuální úrovni tedy nemusíme nutně rozlišovat mezi modely atributů a modely elementů s jednoduchým obsahem.

7. *Dekompozice schémat* – konceptuální modely by mělo být možno dekomponovat na menší části a ty opakovaně používat při vytváření jiných modelů. Příklad – uživatel si jednou vytvoří model *poštovní adresa* a ten může pak opakovaně používat jak např. v kontextu *zaměstnance*, tak i třeba v kontextu *firmy*.

8. *Grafická reprezentace* – jak bylo řečeno je o něco méně významná než předchozí body, nicméně je nutná. Kvalitní grafická reprezentace by měla být co nejjednodušší s co nejmenším množstvím různých symbolů. Naprosto zásadní je, aby jednomu obrázku odpovídal nejvýše jeden konceptuální model, neboť modely musí být jednoznačně interpretovatelné.

Kapitola 3

Regulární stromové gramatiky a jazyky

Víme, že idealizovaně každému konkrétnímu XML schématu odpovídá množina XML dokumentů (instancí), které jsou podle tohoto schématu validní. Idealizovaně proto, že některé konkrétní jazyky do tohoto přirozeného principu vnášejí jisté nekonzistence. Relace *schéma* $\rightarrow$ *dokument* samozřejmě velice připomíná relaci *gramatika* $\rightarrow$ *jazyk*, a proto nepřekvapí, že lze na XML schémata pohlížet jako na určité speciálni gramatiky a na příslušné validní XML dokumenty jako na jazyky generované těmito gramatikami. Pro problematiku XML schémat jsou vhodné regulární stromové gramatiky, které si nyní stručně představíme. Uveďme jen, že automatově–gramaticko–jazykový pohled na problematiku XML schémat je naprosto přirozenou aplikací velice propracované matematické teorie a je více než na místě, protože jen pomocí dokazatelných tvrzení můžeme různé XML schéma jazyky mezi sebou například porovnávat. V [1] jsou navíc i představeny různé validovací algoritmy pro námi dále definované podtřídy regulárních stromových jazyků.

V následujících kapitolách používáme toto běžné značení:

- Jsou-li v, w dvě slova jazyka L ($\{v, w\} \subseteq L$), pak $vw, (v, w), v \cdot w$ jsou zřetězení těchto dvou slov. Jsou-li A, B dvě množiny slov, pak výrazu AB odpovídají všechna slova $vw, v \in A, w \in B$.
- Je-li L jazyk na abecedou Σ a $X \subseteq \Sigma$, pak symbol X^* definujeme jako libovolné (i prázdné) zřetězení písmen z množiny X a symbol $X^+ := XX^*$.
- Je-li X konečná neprázdná abeceda, pak jako $Regexp(X)$ označme množinu všech regulárních výrazů nad X.
- Pro regulární výrazy budeme symbol λ používat pro prázdný řetězec, $+$ pro výčet, $a?$ pro nejvýše jeden výskyt a, $a*$ pro libovolný počet výskytů a, $a+$ jako zástupce $aa*$, implicitní operací je, jak je běžné, operace

spojení.

- Je-li G gramatika, pak jako $L(G)$ budeme označovat jazyk generovaný gramatikou G.

Značením a terminologií se budeme držet [2] a [3].

Podejme nyní základní definice a věty týkající se teorie tzv. *regulárních stromových gramatik a jazyků*. Obecně by se dalo říci, že k popisu XML schémat bychom mohli použít třídu bezkontextových gramatik. Připomeňme si, že bezkontextové jazyky jsou druhý stupeň (T2) nad regulárními jazyky (T3) Chomského hierarchie a že bezkontextové gramatiky povolují jen přepisovací pravidla ve tvaru

$$A \rightarrow w, w \in \{T \cup N\}^*, \tag{3.1}$$

kde A je neterminál, T je množina terminálů a N množina neterminálů. Přívlastek bezkontextový pramení, jak je vidět, z tvaru přepisovacích pravidel, které přepisují neterminál na levé straně pravidla bez ohledu na to, kde se daný neterminál vyskytuje.

Bezkontextové gramatiky jsou již poměrně silným vyjadřovacím nástrojem. Syntax většiny běžně používaných programovacích jazyků je popsán právě bezkontextovými gramatikami. Rozpoznávány jsou např. nedeterministickými zásobníkovými automaty (nedeterminismus je narozdíl od konečných automatů v případě regulárních jazyků u zásobníkového automatu nutný).

Jak víme a zmíníme se o tom i dále, XML je textovou reprezentací stromu. Abychom zohlednili to, že budeme pracovat pouze nad stromy, použijeme místo bezkontextových gramatik regulární stromové gramatiky, jež nám budou místo množiny řetězců generovat množinu stromů. Jedním dechem ihned poznamenejme, že strom v XML podobě je samozřemě také řetězec, nicméně bude vhodnější pracovat s gramatikami generujícími stromy. V každém případě je ale souvislost bezkontextových a regulárních stromových jazyků více než těsná, jak ostatně uvidíme z některých tvrzení.

Definice 9 (Regulární stromová gramatika[1])**.** *Regulární stromová gramatika (RSG) je uspořádaná čtveřice $G = (T, N, S, P)$, kde*

- *T je konečná množina terminálů*
- *N je konečná množina neterminálů*
- *$S \subseteq N$ je konečná množina počátečních neterminálů*
- *P je konečná množina přepisovacích pravidel tvaru*

$$A \rightarrow e(C), e \in T, C \in Regexp(N), \tag{3.2}$$

kde terminál e představuje rodičovský vrchol a regulární výraz C pak generuje jeho syny. Regulárnímu výrazu C budeme říkat model obsahu. Prvkům množiny S budeme také říkat axiomy.

[1] Regular Tree Grammar

Jelikož si budujeme teorii regulárních stromových jazyků i za účelem porovnání známých jazyků pro zápis XML schémat, povolme ještě navíc pravidla ve tvaru

$$A \to R, R \in Regexp(N),$$

tedy pravidla negenerující terminál, ale jen za podmínky, že lze toto pravidlo jednoznačně expandovat na pravidlo ve tvaru

$$A \to R', R' \in Regexp(N \backslash \{A\}),$$

kde R' je regulární výraz neobsahující A. Tradičním příkladem špatného pravidla je

$$C \to (ACB) + AB$$

Toto pravidlo je známé z důkazu existence neregulárního jazyka, kde spolu s pravidly

$$A \to a$$
$$B \to b, a, b \in T$$

a axiomem C generují neregulární (konkrétně bezkontextový) jazyk $\{a^n b^n : n \in \mathbb{N}\}$

Nyní ještě přidejme přirozenou definici pro regulární stromový jazyk.

Definice 10 (Regulární stromový jazyk[2]). *Jazyk L je regulární stromový jazyk (RSJ), pokud existuje regulární stromová gramatika G, která jej generuje.*

Obecná regulární stromová gramatika G může obsahovat i neterminály a pravidla, která nemohou být v procesu derivování nikdy použita. Při detailnějším pohledu zjistíme, že existují dvě základní možnosti:

1. Mohou existovat tzv. *nedosažitelné neterminály*. Přirozeně neterminál je dosažitelný, pokud existuje derivační strom s kořenem axiomem obsahující tento neterminál.

2. Druhou možností je, že některé neterminály se nemusí podílet na generování slov jazyka generovaného gramatikou. Např. obsahuje-li gramatika pravidlo
$$B \to x(B), B \in N, x \in T,$$
a žádné jiné pravidlo neobsahuje neterminál B, pak neterminál B je zřejmě tzv. *neproduktivní*. Přesně řečeno, neterminál B je *produktivní*, pokud $L_G(B) \neq \emptyset$.

Definice 11 (Redukovaná RSG). *Regulární stromová gramatika je redukovaná, pokud neobsahuje nedosažitelné a neproduktivní neterminály.*

[2]Regular Tree Language

Nyní se přirozeně naskýtá otázka, zda ke každé regulární stromové gramatice existuje ekvivalentní gramatika, která je redukovaná. Odpověď ano nepřekvapí.

Věta 1. *Ke každé gramatice G existuje regulární stromová gramatika G' taková, že $L(G) = L(G')$ a G' je redukovaná.*

Důkaz. Je dána RSG $G = (T, N, S, P)$. Gramatiku G' určíme jednoduše pomocí tohoto algoritmu.

1. Spočteme množinu $N_1 \subseteq N$ produktivních neterminálů a množinu $P_1 \subseteq P$ takových pravidel, která obsahují jen neterminály z N_1. Položíme $G_1 := (T, N_1, S, P_1)$.

2. Spočteme množinu dosažitelných neterminálů N_2 gramatiky G_1, tedy $N_2 \subseteq N_1$. Analogicky $P_2 \subseteq P_1$ budou jen pravidla obsahující neterminály z N_2. Položíme $G_2 := (T, N_2, S, P_2)$.

Pak gramatika $G' = G_2$ je hledaným reduktem. Rovnosti $L(G) = L(G_1) = L(G_2)$ jsou zřejmé a gramatika G_2 je zřejmě redukovaná. □

Pro úplnost bez důkazu ještě uveďme, že třída regulárních stromových jazyků je zjednodušeně řečeno mezi třídami regulárních a bezkontextových jazyků Chomského hierarchie.

Uveďme praktický příklad regulární stromové gramatiky (RSG). Ve všech dalších příkladech dodržíme konvenci, že názvy neterminálů budou začínat velkými písmeny a názvy terminálů malými.

Příklad 1.

$$
\begin{aligned}
T &= \{\mathit{lide}, \mathit{clovek}, \mathit{jmeno}, \mathit{krestni}, \mathit{prijmeni}, \mathit{string}\} \\
N &= \{\mathit{Lide}, \mathit{Clovek}, \mathit{Jmeno}, \mathit{Krestni}, \mathit{Prijmeni}, \mathit{String}\} \\
S &= \{\mathit{Lide}\} \\
P &= \{\mathit{Lide} \to \mathit{lide}(\mathit{Clovek}*), \\
&\quad\ \mathit{Clovek} \to \mathit{clovek}\ (\mathit{Jmeno}), \\
&\quad\ \mathit{Jmeno} \to \mathit{jmeno}\ (\mathit{Krestni}+, \mathit{Prijmeni}), \\
&\quad\ \mathit{Krestni} \to \mathit{krestni}\ (\mathit{String}), \\
&\quad\ \mathit{Prijmeni} \to \mathit{prijmeni}\ (\mathit{String}), \\
&\quad\ \mathit{String} \to \mathit{string}\ (\lambda)\ \}
\end{aligned}
$$

Tato gramatika, jak je vidět, generuje strom s kořenem *lide*, který obsahuje libovolný počet podelementů člověk (i nula), každý člověk má pak alespoň jedno křestní jméno a právě jedno příjmení.

Nyní přistupme k problému validace. Máme-li schéma zapsáno ve tvaru regulární stromové gramatiky, je otázka validace daného XML dokumentu (stromu) ekvivalentní zjištění, zda je tento strom generován danou gramatikou. Strom $T = (V, E)$ s pojmenovanými vrcholy je generován gramatikou G zřejmě tehdy, pokud existuje „postup“, jak jej lze z některého počátečního neterminálu vytvořit. Každý vrchol stromu T, je-li $T \in L(G)$, vznikl přepisem některého jednoho neterminálu (terminály generující máme jen pravidla ve tvaru $A \to e(C)$). Představme si, že máme funkci

$$g : V \to N \tag{3.3}$$

z množiny vrcholů stromu T do množiny neterminálů gramatiky G splňující tyto dva požadavky:

1. Je-li r kořen stromu T, pak $g(r) \in S$.
2. Je-li x vrchol se syny $y_1, y_2, \ldots, y_n$, pak existuje v gramatice G přepisovací pravidlo $A \to e(Y)$ takové, že
   (a) název vrcholu x je e,
   (b) $g(x) = A$,
   (c) slovo $g(y_1)g(y_2)\ldots g(y_n)$ odpovídá regulárnímu výrazu Y.

Pak existence takové funkce g nám definuje postup, jakým lze strom z dané gramatiky vytvořit. V souladu s běžně používanou terminologií tedy ještě doplněme následující definici.

Definice 12 (Interpretace stromu)**.** *Funkce g je interpretací stromu T vzhledem ke gramatice G, splňuje-li všechny předchozí body.*

Nyní již můžeme vyslovit definici toho, co je validní strom (XML dokument) vzhledem ke gramatice (XML schématu).

Definice 13 (Validní strom)**.** *Strom T je validní vzhledem ke gramatice G, pokud existuje interpretace stromu T vzhledem ke gramatice G.*

V této chvíli již máme potřebný matematický aparát téměř vybudován. Po doplnění poslední potřebné definice budeme připraveni k tomu, abychom zadefinovali hierarchii podtříd třídy regulárních stromových gramatik (jazyků) a zařadili do ní například jazyky DTD, W3C XML Schema a Relax NG.

Poslední potřebnou definicí je definice tzv. konfliktních neterminálů.[3]

[3] V originále zní termín competitive nonterminals. Doslovný český ekvivalent ať už „soutěžící“, nebo „soutěžní“ je dle mého názoru nevhodný. Vyskytuje se rovněž definice, že takové dva neterminály spolu „soutěží“, ale toto je také nevhodné, protože tyto dva terminály rigorózně vzato žádnou činnost nevyvíjejí, jen existují a vyskytují se na levé straně podobných pravidel.

Definice 14 (Konfliktní neterminály)**.** *Řekneme, že dva různé neterminály $X, Y \in N$ gramatiky G jsou konfliktní, pokud existují dvě různá přepisovací pravidla $p_1, p_2 \in P$ ve tvaru*

$$p_1 = \{X \to xR\},$$
$$p_2 = \{Y \to xQ\},$$

kde $X, Y \in N, x \in T, R, Q \in Regexp(N)$.

Tato definice je, jak dále uvidíme, velice užitečná, neboť právě to, zda budeme v gramatikách povolovat nebo zakazovat konfliktní neterminály, nám poměrně jednoduše indukuje hledanou hierarchii.

3.0.1 Lokální regulární stromové gramatiky

Definice 15 (Lokální regulární stromová gramatika[4])**.** *Regulární stromová gramatika G je lokální, pokud neobsahuje konfliktní neterminály.*

Úplné zakázání konfliktních neterminálů vede k velice malé třídě regulárních stromových gramatik. V tomto případě zřejmě pro každý terminál $x \in T$ existuje nejvýše jedno přepisovací pravidlo, které má terminál x na pravé straně. Jednoduchým pozorováním zjistíme, že v kontextu XML to například znamená, že jeden konkrétní element (v gramatice terminál) má jen jeden možný model obsahu (regulární výraz nad neterminály na pravé straně příslušného pravidla).

Tento fakt je v přímém rozporu s tím, co potřebujeme pro popis archetypů. Pro úplnost uveďme, že zástupcem této třídy lokálních RSG je např. jazyk W3C DTD.

3.0.2 Jednotypové stromové gramatiky

Definice 16 (Jednotypová stromová gramatika[5])**.** *O gramatice G, která je regulární stromová, řekneme, že je navíc jednotypová, pokud pro každé přepisovací pravidlo gramatiky G*

$$A \to e(C), A \in N, e \in T, C \in Regexp(N)$$

platí, že všechny neterminály v C obsažené jsou po dvou nekonfliktní, a všechny axiomy jsou rovněž po dvou nekonfliktní.

Podívejme se, jaký má fakt, že jsme zakázali konfliktní neterminály v rámci modelů obsahů, efekt. Pozorování je opět velice triviální. Nechť jsme obdrželi

[4] V originálé local tree grammar.
[5] V originále single-type tree grammar.

aplikací pravidla $A \to x(C)$ vrchol x. Víme, že C neobsahuje konfliktní neterminály. Je-li pak y synem vrcholu x, je model obsahu vrcholu y dán jednoznačně. Indukcí tohoto pozorování dostáváme tvrzení, že model obsahu vrcholu v je jednoznačně určen cestou z kořene do vrcholu v.

Předešleme, že nejznámějším „přibližným“ zástupcem této třídy je právě W3C XML Schema. Přibližným díky tomu, že W3C XML Schema definuje několik prvků, které regulární stromové gramatiky nemohou postihnout, ale na druhé straně nedosahuje zcela úplné síly ani JRSG.

3.0.3 Hierarchie gramatik

Nyní již zbývá jen porovnat expresivitu lokálních ($LRSG$), jednotypových ($JRSG$) a obecných regulárních stromových gramatik (RSG).

Věta 2 ($L(LRSG) \subsetneq L(JRSG)$)**.** *Lokální regulární stromové gramatiky jsou ostře méně expresivní než jednotypové regulární stromové gramatiky.*

Důkaz. Neostrá inkluze $L(LRSG) \subset L(JRSG)$ platí z definice. Gramatika G s axiomy A, B a pravidly

$$\begin{aligned} A &\to \alpha(X) \\ B &\to \beta(Y) \\ X &\to x(\lambda) \\ Y &\to x(Z) \\ Z &\to z(\lambda) \end{aligned}$$

je zřejmým příkladem takové gramatiky, že $L(G)$ nemůže generovat žádná lokální regulární stromová gramatika. Neterminály X a Y jsou konfliktní (oba generují terminál x). Jazyk generovaný touto gramatikou je

$$\{\alpha(x()), \beta(x(z()))\}.$$

U lokální gramatiky je právě jeden možný model obsahu vrcholu x. U tohoto jazyka ale záleží obsah vrcholu x na jeho otci. □

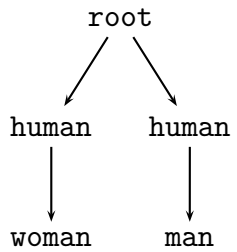
Podobně jednoduše ukažme druhou ostrou inkluzi.

Věta 3 ($L(JRSG) \subsetneq L(RSG)$)**.** *Jednotypové regulární stromové gramatiky jsou ostře méně expresivní než regulární stromové gramatiky.*

Důkaz. Uvažujme regulární stromovou gramatiku G_R

$$
\begin{aligned}
T &= \{root, human, man, woman\} \\
N &= \{Root, Mankind, Womankind, Man, Woman\} \\
S &= \{Root\} \\
P &= \{ \\
&\quad Root \rightarrow root(Mankind*, Womankind*) \\
&\quad Mankind \rightarrow human(Man) \\
&\quad Womankind \rightarrow human(Woman) \\
&\quad Man \rightarrow man(\lambda) \\
&\quad Woman \rightarrow woman(\lambda) \\
&\quad \}
\end{aligned}
$$

Tato gramatika zřejmě generuje stromy s kořenem *root*, které mají jako syny nejprve libovolný počet vrcholů *human* majících syna *man* a pak libovolný počet vrcholů *human* majících syna *woman*.[6] Předpokládejme, že existuje ekvivalentní jednotypová gramatika G_J. Jak již bylo řečeno výše, model obsahu vrcholu v závisí u jednotypových regulárních stromových gramatik jednoznačně na cestě z kořene do vrcholu v. Máme-li tedy cestu $root \rightarrow human$, pak obsah vrcholu *human* je dán jedním pevným modelem obsahu C bez ohledu na pořadí vrcholu *human*. Umí-li tedy gramatika G_J vygenerovat to, co gramatika G_R, pak do jazyka generovaného gramatikou G_J patří i např. strom



Tento strom ale negeneruje gramatika G_R, což je spor. □

3.0.4 Uzávěrové vlastnosti

Nakonec se zmiňme ještě o uzávěrových vlastnostech definovaných podtříd regulárních stromových jazyků. Tyto uzávěrové vlastnosti mají rovněž naprosto fundamentální důležitost. Uzavřenost např. regulárních stromových jazyků na sjednocení, průniky a rozdíly usnadňuje algebraické postupy při práci s těmito

[6] I taková hezká spojení jsou v matematice možná...

jazyky, uzávěrové vlastnosti jsou důležité např. i při zkoumání inkluze dvou daných gramatik. Následující větu uveďme bez důkazu.

Věta 4. *Uzávěrové vlastnosti regulárních stromových, jednotypových regulárních stromových a lokálních regulárních stromových jazyků shrnuje následující tabulka.*

Třída jazyků	*Uzavřenost na* $\cup$	*Uzavřenost na* $\cap$	*Uzavřenost na* $\setminus$
RS jazyky	**ANO**	**ANO**	**ANO**
JRS jazyky	NE	ANO	NE
LRS jazyky	NE	ANO	NE

Tabulka 3.1: Uzávěrové vlastnosti

Poznamenejme, že důkazy negativních tvrzení jsou poměrně snadné a vždy se najdou jednoduché protipříklady. Příkladně, že třídy LRSJ a JRSJ nejsou uzavřené na sjednocení se dokáže najednou tak, že se najdou dvě lokální gramatiky G_1 a G_2 takové, že $L(G_1) \cup L(G_2)$ není ani jednotypový jazyk.

Vidíme tedy, že pouze ryzí regulární stromové gramatiky generují jazyky, které jsou uzavřeny jak na sjednocení, tak i na průnik a rozdíl.

Kapitola 4

A(C)-model

V této kapitole představím svůj vlastní jednoduchý formální konceptuální model XML (hierarchických) dat. Nejprve se zamysleme, jak by měl formální konceptuální model vypadat.

Konceptuální model je, jak již bylo řečeno, vysokoúrovňový pohled na strukturu dat, který si rozdělíme na dvě nezávislé části:

- model struktury dat
- soubor omezujících podmínek na obsah a vztahy[1]

Přitom při vytváření konkrétního konceptuálního modelu postupujeme právě v tomto pořadí. Nejprve navrhneme strukturu dat a až poté doplníme seznam omezujících podmínek.

Při návrhu konceptuálních jazyků je zřejmě třeba mít na paměti jazyk logické vrstvy. Všechny prvky konceptuálního jazyka musí být logickou vrstvou postihnutelné, navíc překlad konceptuálního modelu (tj. modelu struktury a omezujích podmínek) na logický model musí být zcela algoritmizovatelný. V příští kapitole se budeme zabývat i rozhodnutelností problému existence validní instance konkrétního modelu.

V případě XML máme pro model struktury dat na logické úrovni k dispozici regulární stromové gramatiky, které jsou pro jakékoli formální úvahy o XML vynikajícím prostředkem. Pro zápis regulárních stromových gramatik (kvůli zpracování počítačem), potřebujeme vhodný jazyk. V dalším textu bude ukázáno, že jedním z nejsilnějších zástupců je jazyk Relax NG, ve kterém je možno zapsat libovolnou regulární stromovou gramatiku. Navrhnout konceptuální jazyk je ale přitom nutno tak, aby modely neprodukovaly po překladu např. jen lokální nebo jednotypové gramatiky.

Pomocí regulárních stromových gramatik jsme schopni maximálně efektivně postihnout syntax XML dokumentů. Nemáme ale žádné prostředky na

[1] Bude upřesněno dále v textu.

to, definovat například, že textová hodnota elementu *supremum* je jako číslo větší nebo rovna hodnotě elementu *infimum*, nebo že všechny obsahy elementů *ID* musí být po dvou různé.

Z tohoto důvodu je nutno na logické (ale i konceptuální) úrovni uvažovat jazyk, schopný podobné sémantické podmínky vyjádřit. My si jako jazyk pro popis omezujících podmínek zvolíme logiku první řádu, pomocí níž budeme schopni vyjádřit drtivou většinu běžně používaných integritních omezení. Dodejme, že v následujícím textu budeme používat axiom výběru.

Definice 17 (Selektor). *Nechť $T = (V, E)$ je zakořeněný strom. Pak selektor vrcholu $v \in V$, který má syny $w_1, w_2, \ldots, w_k \in V$, je libovolná boolovská formule k-proměnných $\mathcal{B}^k(e_1, e_2, \ldots, e_k)$.*

Definujme několik dále běžně používaných selektorů, $n \in \mathbb{N}$.

- Selektor $0^n(e_1, e_2, \ldots, e_n) \equiv 0$
- Selektor $1^n(e_1, e_2, \ldots, e_n) \equiv 1$
- Selektor $\&^n(e_1, e_2, \ldots, e_n) \equiv \bigwedge_{i=1}^{n} e_i$
- Selektor $\vee^n(e_1, e_2, \ldots, e_n) \equiv \bigvee_{i=1}^{n} e_i$
- Selektor $\oplus^n(e_1, e_2, \ldots, e_n) \equiv \bigvee_{i=1}^{n}(e_i \& \neg \bigvee_{k \in [n]\setminus\{i\}} e_k)$

Definice 18 (Metaneterminál). *Nechť $P_1 = \{@, \#\}$ a $P_2 = \{\cdot, ?, *, +\}$, dále nechť Σ je neprázdná abeceda znaků obsahující alespoň malé a velké znaky abecedy a čísla taková, že $\Sigma \cap (P_1 \cup P_2) = \emptyset$. Označme $\mathcal{L}_M = [P_1^? \Sigma^+ P_2]$ jazyk generovaný regulárním výrazem v hranatých závorkách. Pak libovolné slovo $w \in \mathcal{L}_M$ definujeme jako metaneterminál.*

Metaneterminálům začínajícím na @ budeme říkat reference a metaneterminálům začínajícím na # substituce. Pokud metaneterminál nezačíná na znak z P_1, pak ho označme jako ryzí.

Dále definujme navíc funkci

$$Card : \mathcal{L}_M \to P_2$$

vracející pro daný metaneterminál jeho poslední znak a funkci

$$Term : \mathcal{L}_M \to [\Sigma^+]$$

vracející maximální podslovo z $[\Sigma^+]$.

Definice 19 (Klíčová slova). *Nechť $K = \{\Delta, \Xi, Instance\}$. Rozšiřme množinu Σ alespoň tak, aby $K \subset [\Sigma^+]$. Pak definujme sadu klíčových slov $\mathcal{K}_M = K$.*

Abecedu Σ z předchozí definice si nyní zvolme libovolně a zafixujme ji. Zafixujme i odpovídající jazyk $\mathcal{L}_M$. Dále zafixujme neprázdnou množinu $\mathbb{ID}$ slov z Σ^+. Množině $\mathbb{ID}$ budeme říkat množina identifikátorů, prvkům pak identifikátory.

Definice 20 (Archetyp)**.** *Nechť $T = (V, E)$ je neprázdný zakořeněný strom, jehož hrany jsou orientované od kořene r k listům, s pevně definovaných lineárním uspořádáním na synech libovolného nelistového vrcholu $v \in V$. Nechť F je funkce přiřazují každému vrcholu buď metaneterminál, nebo selektor. Množinu vrcholů stromu T, kterým F přiřazuje ryzí metaneterminál, nazvěme nosič – $\mathcal{N}_F(T)$. Dále mějme funkci D, která každému vrcholu nosiče přiřazuje slovo ze $[\Sigma^+]$. Funkce F a D navíc splňují, že*

$$(Term \circ F)(X) \cap D(\mathcal{N}_F(T)) = \emptyset,$$

kde $X = \{x \in V : F(x)$ je metaneterminál $\}$. Nechť platí všechny následující body.

(i) $F(r)$ *je ryzí metaneterminál a* $Card(r) = \cdot$

(ii) Je-li $x \in V$ list, pak $F(x)$ je metaneterminál.

(iii) Pro každou hranu $(x, y) \in E$ platí, že $F(x)$ je metaneterminál, právě když $F(y)$ není metaneterminál.

(iv) Je-li $F(v)$ substituce nebo reference pro $v \in V$, pak v je list.

(v) Je dán speciální terminál $\tau \in \mathbb{ID}$. Budeme mu říkat typ archetypu.

(vi) Pro každý vrchol $v \in V$ platí, že

$$F(v) \text{ je ryzí metaneterminál} \Rightarrow ((Term \circ F)(v) \neq \Xi \ \& \ D(v) \neq \xi)$$

Pak čtveřici (T, F, D, τ) nazvěme archetyp. Je-li $F(r) = X\cdot$ a $F(v) = \#X, v \in V$, pak metaneterminál $\#X$ nazveme rekurze.[2]

Archetyp bude základním prvkem dále definovaného A-modelu. Nyní pevně definujme interpretaci archetypu, neboli regulární stromovou gramatiku, kterou generuje.

Definice 21 (Převod archetypu na gramatiku)**.** *Nechť $\mathcal{A}$ je třída všech archetypů. Popíšeme funkci*

$$\mathcal{G} : \mathcal{A} \to RSG, \tag{4.1}$$

[2]Rekurze je tedy zvláštním případem substituce.

kde RSG je třída všech regulárních stromových gramatik, převádějící archetyp na regulární stromovou gramatiku.

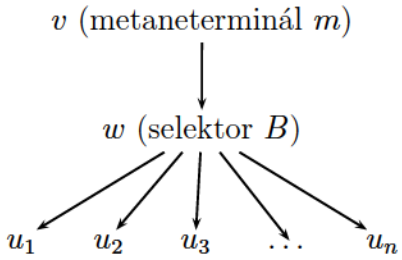
Nechť $A \in \mathcal{A}, A = (T, F, D, \tau)$ je archetyp. Pak gramatikou odpovídající archetypu A je gramatika $G = \mathcal{G}(A), G = (T_G, N_G, S_G, P_G)$ definovaná následujícím algoritmem.

Algoritmus převodu archetypu na gramatiku

Nejprve definujme pomocnou proceduru `pridejPravidla` s jedním parametrem v, který představuje kořenový vrchol (pod)stromu archetypu, jehož prozkoumáním se přidají do „globální“ množiny P_G přepisovací pravidla. Navíc $v \in \mathcal{N}_F(T)$.

`procedura pridejPravidla(vrchol v)` {

1. Není-li v list, pak skoč na bod 3.
2. Do P_G přidáme pravidlo $\{Term(F(v)) \to D(v)(\lambda)\}$ a skončíme.



Obrázek 4.1: Nelistový vrchol v

3. Vrchol v má jediného syna w a ten nechť má syny $u_1, u_2, \ldots, u_n$. Nechť $B := F(w)$. Viz obrázek 4.1.
4. Zkonstrujeme pomocí funkce `naRegexp` (viz dále) regulární výraz C (model obsahu) odpovídající formuli B a konečné posloupnosti metaneterminálů $F(u_1), F(u_2), \ldots, F(u_n)$, tedy

$$C := \texttt{naRegexp}(B, \{F(u_i)\}_{i=1}^{n}). \tag{4.2}$$

5. Do množiny přepisovacích pravidel P_G přidáme pravidlo

$$Term(F(v)) \to D(v)(C). \tag{4.3}$$

6. Rekurzivně se zavoláme na vnucích, kteří patří do nosiče, tedy
$$\forall u \in \{u_1, u_2, \ldots, u_n\} \cap \mathcal{N}_F(T) : \texttt{pridejPravidla}(u).$$

}

Nyní máme definovánu proceduru **pridejPravidla**. Algoritmus pro převedení archetypu na regulární stromovou gramatiku provede tyto kroky:

1. Pokud je funkce $Term \circ F$ na $\mathcal{N}_F(T)$ prostá, skoč na bod 4.
2. Zkonstruuj funkci F' se stejným definičním oborem jako F takovou, že $Term \circ F'$ bude prostá na $\mathcal{N}_F(T)$. Funkci F' vytvoříme tak, že
$$F'(v) = \begin{cases} F(v) & : v \in V \backslash \mathcal{N}_F(T), \\ (w, u(v)) & : v \in \mathcal{N}_F(T), \end{cases}$$
kde $w \in [\Sigma^+]$, u je libovolná prostá funkce z $\mathcal{N}_F(T)$ do $[\Sigma^+ P_2]$ a
$$\begin{aligned} w &= Term(F(v)), \\ Card(u(v)) &= Card(F(v)). \end{aligned}$$
Funkce F' je tedy jen přeznačením všech vrcholů nosiče T. Funkci u můžeme vytvořit například tak, že $u(v)$ bude odpovídat číslu, které přiřadí prostě číslující preorder průchod do hloubky.
3. Polož $F := F'$.
4. Nechť r je kořen stromu T.
5. $T_G := D(\mathcal{N}_F(T)) \cup \{\alpha, \beta, \tau, \iota\}$
6. $N_G := (Term \circ F)(X) \cup \{\Xi, \sqsupset\}$, kde $X = \{x \in V : F(x)\ je\ metaneterminál\}$.
7. $S_G := \{\sqsupset\}$, kde $\sqsupset \notin \Sigma$.
8. Polož $P_G := \emptyset$ a proveď **pridejPravidla**(r).
9. Dále ještě do P_G přidej tato pravidla
$$\sqsupset \to D(r)(I\mathcal{R}), \tag{4.4}$$
$$I \to \alpha(\Xi\Pi), \tag{4.5}$$
$$\#\,Term(F(r)) \to Term(F(r)) \tag{4.6}$$
$$@\,Term(F(r)) \to \beta(\Xi\Pi), \tag{4.7}$$
$$\Xi \to \tau(\lambda), \tag{4.8}$$
$$\Pi \to \iota(\Delta), \tag{4.9}$$
$$\Delta \to \delta(\lambda), \tag{4.10}$$
kde $\mathcal{R}$ v pravidle (4.4) je stejný model obsahu, jako má už přidané pravidlo s $Term(F(r))$ na levé straně.

Symbol $\sqsupset$ je, jak je zřejmé, přidán proto, aby kořen stromu generovaného gramatikou archetypu měl syna α, ale podstromy, které vznikly rekurzí nikoli. Přitom α-podstrom bude dále použit pro uchování identifikátoru instance a β-podstrom pro referenci. Jméno vrcholu τ pak představuje typ instance, nebo typ cíle reference, symbol ι pak její identifikátor. δ-vrcholy budou datové vrcholy. Vše bude objasněno dále. Tuto poznámku jen uvádím pro bližší vysvětlení výše uvedených přepisovacích pravidel.

Nyní zbývá již jen popsat funkci **naRegexp**. Možných algoritmů je více (např. uvažovat jen formule v DNF), uveďme nejprve jednoduchý přímočarý postup. Jde nám totiž pouze o definici. Jak již bylo řečeno vstupem této funkce je boolovská formule n proměnných a konečná posloupnost (!) metaneterminálů $\{F(u_i)\}_{i=1}^{n}$. Výstupem pak je regulární výraz.

funkce naRegexp(B, Fs): Regexp {

1. Nechť E množina všech vektorů ohodnocení proměnných, pro které je formule B splněna, tedy $E = \{E_1, E_2, \ldots, E_K\}, K \in \mathbb{N}_0, E_i \in \{0,1\}^n$,

$$B(E_i) \equiv 1,$$

   kdykoli $i \in [K]$.

2. Pro každé $i \in [K]$ označme m_i počet jedniček ve vektoru E_i. Dále nechť pro pevné $i \in [K]$ je $\{j_p^i\}_{p=1}^{m_i}$ rostoucí posloupnost indexů takových, že $E_{i,j_p^i} = 1$ kdykoli $p \in [m_i]$. Pak položíme

$$R_i = (X_1^{C_1}, X_2^{C_2}, \ldots, X_{m_i}^{C_{m_i}}), \tag{4.11}$$

   kde pro $p \in [m_i]$ je $X_p = Term(F(u_{j_p^i}))$ a $C_p = Card(F(u_{j_p^i}))$.

3. Vrátíme regulární výraz $R = R_1 + R_2 + \cdots + R_K$.

}

Tím je algoritmus převodu archetypu na gramatiku hotov. □

Definice 22 (A-model)**.** *Nechť $\{A_i\}_{i=1}^{n}, n \in \mathbb{N}$ jsou archetypy. Je-li $i \in [n]$, pak jako r_i označme kořen stromu archetypu A_i a jako τ_i identifikátor archetypu A_i. Dále nechť les $\mathcal{L} = \{A_i\}_{i=1}^{n}$ těchto stromů splňuje všechny následující podmínky.*

(i) $i, j \in [n], i \neq j \rightarrow F_i(r_i) \neq F_j(r_j)$ & $\tau_i \neq \tau_j$

(ii) Pro každý vrchol v stromu T_i libovolného archetypu $A_i, i \in [n]$, pro který je $F_i(v)$ substituce nebo reference, existuje index $k \in [n]$ takový, že $(Term \circ F_k)(r_k) = (Term \circ F_i)(v)$.

Potom dvojici $(\mathcal{L}, \tau)$ označíme jako A-model.

A-model je tedy les archetypů s přiřazenými unikátními identifikátory navíc uzavřený na substituce a reference (všechna přepisovací pravidla jsou tedy dobře definovaná). Nyní formálně definujme překlad A-modelu na regulární stromovou gramatiku. Překlad opět budeme definovat pomocí deterministického algoritmu.

Překlad A-modelu na gramatiku

Pro překlad opět definujeme funkci

$$\mathcal{G} : \mathcal{AM} \to RSG,$$

kde $\mathcal{AM}$ je třída všech A-modelů a RSG je třída všech regulárních stromových gramatik. Na konci definice budeme tedy mít k dispozici dvě různé verze funkce $\mathcal{G}$, jednu pro archetypy a druhou pro A-modely.

Nechť $\mathcal{A} = \{A_1, A_2, \ldots, A_n\}$ je A-model. Předefinujeme analogicky jako při překladu jednoho archetypu všechny hodnoty funkce F na vrcholech z nosičů všech archetypů (opět např. DFS na lese) tak, aby byla $Term \circ F$ globálně na nosičích prostá.

Nechť pro každé $i \in [n]$ je $G_i = \mathcal{G}(A_i)$. Nechť BÚNO žádná z gramatik $G_i, i \in [n]$ neobsahuje neterminál *Instance* a terminál *instance*. Pokud ano, provedeme přejmenování. Navíc přejmenujme axiom každé gramatiky $G_i, i \in [n]$ z $\sqsupset$ na $\sqsupset_i$ a odpovídajícím způsobem upravme i příslušné jediné pravidlo mající v gramatice G_i axiom $\sqsupset$ na levé straně. Pak definujme $\mathcal{G}(\mathcal{A}) := G = (T_G, N_G, S_G, P_G)$ následovně:

1. Množinu terminálů výsledné gramatiky G definujme jako

$$T_G := \left(\bigcup_{i=1}^{n} D(\mathcal{N}_{F_i}(T_i)) \right) \cup \{instance\}$$

2. Množinu neterminálů výsledné gramatiky G pak jako

$$N_G := \left(\bigcup_{i=1}^{n} (Term \circ F_i)(X_i) \right) \cup \{Instance\},$$

   kde $X_i = \{x \in V_i : F(x)$ *je metaneterminál* $\}$.

3. Výsledná gramatika G bude mít právě jeden axiom $S_G := \{Instance\}$.

4. A konečně množinu přepisovacích pravidel P_G definujeme jako

$$P_G := \left(\bigcup_{i=1}^{n} P_{G_i} \right) \cup \{Instance \to instance\ (\sqsupset_1^*, \sqsupset_2^*, \ldots, \sqsupset_n^*)\},$$

   kde pro $i \in [n]$ je $\sqsupset_i$ je axiom gramatiky G_i.

Tím je definice algoritmu pro překlad A-modelu na regulární stromovou gramatiku hotova. □

Vyslovme nyní následující klíčovou definici instance A-modelu, ve které už navíc přidáme možný obsah δ-vrcholů. Obsahem bude libovolné slovo z Σ^*. Až po zavedení omezujících podmínek budeme schopni o obsazích uvažovat i v kontextu datových typů.

Instance A-modelu

Definice 23 (Instance)**.** *Instance A-modelu A je dvojice (T, ω), kde T je strom s pojmenovanými vrcholy, $T \in \mathcal{G}(A)$, $T = (V, E, L)$ a ω je libovolná funkce*

$$\omega : \{v \in V : L(v) = \delta\} \to [\Sigma^*].$$

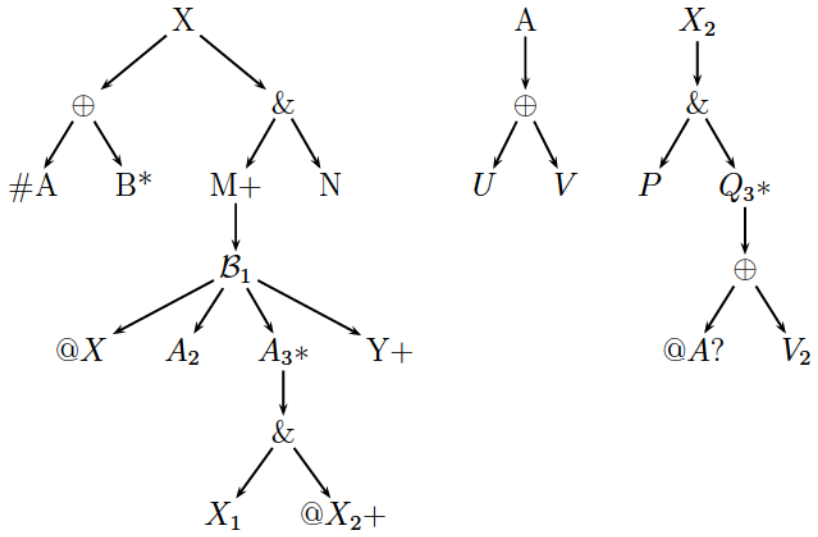
Touto definicí jsme završili budování A-modelu. Hodnota funkce ω v konkrétním vrcholu v, jenž je pojmenovaný δ, je libovolný i prázdný řetězec nad abecedou Σ. Nyní máme tedy formálně zadefinován A-model, určený pro modelování regulárních stromových gramatik, a tedy následně i syntaxe XML dokumentů.

Takto definovaný A-model ale ještě zdaleka není výsledkem. Musíme k němu přidat omezující podmínky. Tedy zejména přidat datové typy a definovat jazyk ekvivalentní logice prvního řádu pro zápis takových omezujících podmínek.

Jelikož A-modely vycházejí z archetypů a archetypy jsou speciálně ohodnocené stromy, je samozřejmě možno A-modely graficky znázorňovat. Na obrázku 4.2 máme příklad jednoduchého A-modelu složeného ze tří archetypů.

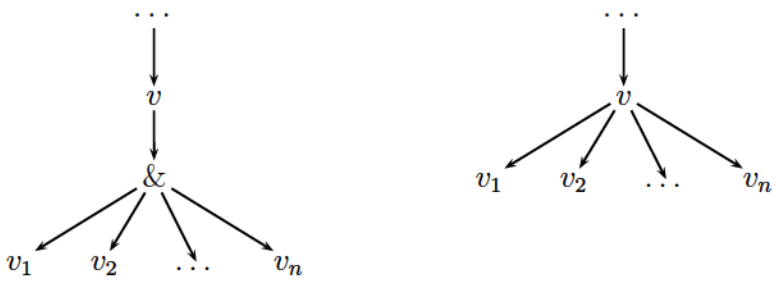
Definice 24 (Grafické konvence)**.** *Pro kreslení archetypů můžeme použít následující konvence:*

- *Každý archetyp je znázorněn zakořeněným stromem s hranami orientovanými k listům (z definice).*
- *Do vrcholů vždy píšeme hodnotu funkce F pro daný vrchol, přitom je-li pro daný vrchol $v \in V, F(v)$ metaneterminál a $Card(F(v)) = \cdot$, pak můžeme tečku v zápise vynechat.*
- *Pokud daný vrchol v patří do definičního oboru funkce D (v je z nosiče) a slovo $w = Term(F(v))$ začíná velkým písmenem, pak můžeme hodnotu funkce D na vrcholu v vynechat a implicitně bude definována jako w, ve kterém se změní počáteční písmeno na malé. Hodnoty funkce D se v daném vrcholu b píší za hodnoty funkce F a od hodnot F se oddělují lomítkem.*



Obrázek 4.2: Jednoduchý A-model

- *Aritu definovaných selektorů* $0, 1, \&, \vee, \oplus$ *je možno v zápise vynechat a implicitně je rovna počtu synů příslušného vrcholu.*
- *Pro zápis obecného n-árního selektoru pro vrchol* $v \in V(T)$ *s n syny se použije sada n libovolných proměnných indexovaných všemi indexy z* $[n]$*, např. pro* $n = 3$ *je použitelná sada* $\{A_1, \alpha_2, \omega_3\}$*. Index pak odpovídá pořadí syna. Tedy např. pokud chceme napsat klauzuli ve tvaru „první syn a zároveň buď druhý, anebo třetí“, napíšeme* $A_1\&(\alpha_2 \oplus \omega_3)$*.*
- *Selektor & je implicitním selektorem. Je jej proto možno libovolně v obrázcích vynechávat a přímo tím spojovat jednotlivé metaneterminály mezi sebou. Přesněji, je-li v vrchol s n syny* $v_1, v_2, \ldots, v_n$*, pak tyto dva obrázky představují totéž.*

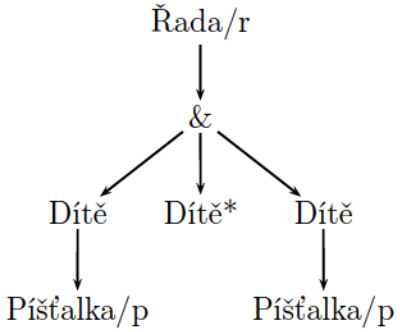


- *Má-li vrchol v s $F(v) = X$ asociovánu hodnotu (syna Δ) a přítomnost tohoto Δ-syna je implikována splněností selektoru obsahu vrcholu v, pak nemusíme graficky Δ vrchol samostatně kreslit a můžeme místo toho symbol Δ připsat k hodnotě $F(v)$ na místo horního indexu.*

- *Všechny výše zmíněné body se týkají pouze grafického znázornění. Ve formálním zápise konkrétního A-modelu nic vynechávat nelze. Zápis musí odpovídat axiomatice A-modelu.*

Příklad 2 (Orientační síla A-modelu). Ukažme, že lze namodelovat jednoduchá situace s řadou dětí na výletě vedoucí na nejednotypovou regulární stromovou gramatiku. Tato situace je tedy v jistém smyslu „složitá“. Chceme, aby právě a jen první a poslední dítě mělo píšťalku za účelem upozornění ostatních dětí. Element *dítě* představuje dítě, pokud má dítě píšťalku, pak obsahuje ještě navíc podelement p.

Na obrázku vidíme její A-model tvořený jedním archetypem *Řada*. Dodejme, že používáme výše uvedenou konvenci o implicitním selektoru.



Obrázek 4.3: Nejednotypový A-model

Překladem tohoto modelu na gramatiku funkcí $\mathcal{G}$ dostáváme gramatiku $G = (T, N, S, P)$ takovou, že

- Množina terminálů $T = \{r, \textit{dítě}, p, \xi, \textit{instance}\}$

- Množina neterminálů

$$N = \{\textit{Řada}, \textit{Dítě3}, \textit{Dítě5}, \textit{Dítě6}, \textit{Píšťalka}, \Xi, \textit{Instance}\}$$

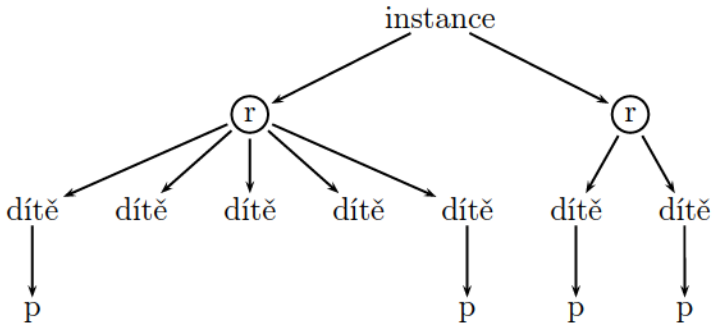
- Jediný axiom $S = \{\textit{Instance}\}$

- Sada přepisovacích pravidel

$$\begin{aligned} P = \{ & \mathit{Instance} \rightarrow \mathit{instance}\ (\mathit{Řada}*), \\ & \mathit{Řada} \rightarrow r\ (\mathit{Dítě3}, \mathit{Dítě5}*, \mathit{Dítě6}), \\ & \mathit{Dítě3} \rightarrow \mathit{dítě}\ (\mathit{Píšťalka}), \\ & \mathit{Dítě5} \rightarrow \mathit{dítě}\ (\lambda), \\ & \mathit{Dítě6} \rightarrow \mathit{dítě}\ (\mathit{Píšťalka}), \\ & \mathit{Píšťalka} \rightarrow p\ (\lambda)\} \end{aligned}$$

Jelikož archetyp obsahoval 3 vrcholy, jimž byl přiřazen metaneterminál, na kterých měla funkce *Term* stejnou hodnotu *Dítě*, došlo k rekonstrukci funkce F pomocí očíslování vrcholů (použito preorder číslování) – viz popis překladového algoritmu.

Tato gramatika generuje například tento strom.



Obrázek 4.4: Příklad instance

Můžeme tedy říci, že výše zobrazený strom je instancí prezentovaného A-modelu. Vrcholy označené kružnicemi odpovídají kořenům instancí archetypu *Řada*. □

4.1 Omezující podmínky

Jak již bylo řečeno, abychom dostali plnohodnotný modelovací jazyk, musíme mít nástroje jednak na modelování hierarchických struktur (A-modely) a jednak též na popis doprovodných omezujících podmínek.

Stromy generované gramatikou $\mathcal{G}(A)$, kde A je libovolný pevný A-model, jsou filosoficky řečeno formou pro data, jež chceme uchovávat. Konkrétní A-model A nyní tedy zafixujeme. Budeme jej používat v dalším textu. Na jazyku

omezujících podmínek je, aby omezil možné kombinace dat a vdechl tak stromům jistou sémantiku.

Jak jsme již rovněž řekli, použijeme v tomto textu logiku prvního řádu a zamyslíme se nad její vyjadřovací silou. Ukážeme, že ani logika prvního řádu nemůže vyjádřit všecha omezení, která můžeme na instance klást.

Datové typy

Prvním typem omezujících podmínek, kterými se budeme zabývat, jsou datové typy. S datovými typy budeme zacházet podle následující definice.

Definice 25 (Datový typ)**.** *Datový typ T je uspořádaná dvojice $(s, \mathcal{M})$, kde $s \in \Sigma^+$ je název datového typu a $\mathcal{M}$ je libovolná spočetná množina.*

Malá poznámka k definici. Uvažujeme v kontextu XML dat. Každý XML dokument je konečná posloupnost znaků z konečné množiny, tedy speciálně obsah každého elementu je konečná posloupnost znaků. Množina obsahující libovolné konečné posloupnosti znaků z konečné abecedy je spočetná. K důkazu můžeme použít klasickou Cantorovu diagonalizační metodu – řetězce seřadíme do skupin podle délky, v každé skupině je seřadíme lexikálně. Množina všech možných znaků je konečná, takže na ní není problém zavést lineární uspořádání. Na výsledek pak použijeme zmíněnou Cantorovu diagonalizační metodu, která nám dokáže, že zmíněná množina všech konečných řetězců je spočetná.

Definice 26 (Obor hodnot datového typu)**.** *Množině M datového typu $T = (s, M)$ budeme říkat obor hodnot datového typu T.*

Regulární cesty

Proměnnými v logických podmínkách budou rovněž podstromy, resp. podlesy instancí. Abychom je mohli specifikovat, musíme vytvořit nástroj absolutních a regulárních cest ve stromu.

Úmluva. Veškeré stromy v této kapitole budou konečné zakořeněné stromy s orientovanými hranami od kořene k listům. Navíc všechny vrcholy stromu jsou pojmenovány řetězci ze Σ^*. Definujme dále na synech každého nelistového vrcholu lineární uspořádání a na jeho základě funkci *pos*,

$$pos : V \to \mathbb{N},$$

jež bude pro vrcholy tohoto stromu vracet jejich pořadí v daném lineárním uspořádání mezi sourozenci. Speciálně, je-li r kořen, pak $pos(r) = 1$.

Nejprve si zadefinujme pojem absolutní cesta ve stromu. Pro tuto podkapitolu definujme

$$\mathcal{P}_\Sigma = (\Sigma \setminus \{/, \cdot, *\})^+.$$

Definice 27 (Absolutní cesta). *Absolutní cesta je libovolné slovo generované následující regulární gramatikou s axiomem P:*

$$P \to EP$$
$$E \to /w, w \in \mathcal{P}_\Sigma$$
$$P \to \lambda$$

Jazyk generovaný touto gramatikou budeme označovat jako AP.

Definice 28 (Popis cesty). *Nechť $T = (V, E, L)$ je zakořeněný strom s pojmenovanými vrcholy a hranami orientovanými od kořene k listům. Pak na tomto stromě definujme funkci*

$$Path : V \to AP$$

rekurzivně tak, že

$$Path(v) = \begin{cases} /L(v) & : v \text{ je kořen stromu } T, \\ (Path(u), /L(v)) & : v \text{ není kořen stromu } T \ \& \ (u, v) \in E. \end{cases}$$

Jsou-li $u, v \in V$ spojeny orientovanou cestou z u do v, pak definujme

$$Path^R(u, v) = Path(v'),$$

kde v' je kořenem stromu T', jenž je podstromem stromu T indukovaným vrcholem v a všemi z něj dosažitelnými vrcholy.

Absolutní cesty budeme používat pro vybírání podstromů daného stromu podle této definice.

Definice 29 (Les generovaný absolutní cestou). *Nechť $p \in AP$ absolutní cesta a $T = (V, E, L)$ je stromem podle úmluvy. Pak jako $[p]_T$ označíme les indukovaných podstromů stromu T, jejichž kořeny jsou prvky množiny*

$$\{v \in V : Path(v) = p\}.$$

Je-li $X \subset AP$, pak definujme

$$[X]_T = \bigcup_{x \in X} [x]_T.$$

Poznamenejme, že aby byla další teorie precizní, je nutno, aby všechny stromy v lese byly po dvou rozlišitelné. Tedy, přestože mohou být dva stromy shodné jak tvarem, tak i pojmenováním, jsou-li jejich kořeny v původním stromu dva různé vrcholy, pak tyto dva stromy nejsou v relaci $=$.

Definice 30 (Regulární cesta). *Regulární cesta je libovolné slovo generované regulární gramatikou s axiomem P a těmito pěti pravidly:*

$$P \to PP \mid \lambda$$
$$P \to /w_1 + w_2 + \cdots + w_k, k \in \mathbb{N}, i \in [k] \to w \in \mathcal{P}_\Sigma$$
$$P \to / \cdot \mid /*$$

Množinu všech regulárních cest (jazyk gramatiky) budeme značit RP.

Příkladem regulárních cest tedy jsou cesty

$$/*/x$$
$$/./x/*/y$$
$$/x/y + z/*/a$$

Definice 31 (Jazyk generovaný regulární cestou). *Nechť r je regulární cesta. Absolutní cesta a je generovaná regulární cestou r, pokud můžeme k cestě r zkonstruovat cestu r' tak, že $a = r'$, a přitom můžeme pro transformaci $r \to r'$ opakovaně používat jen tyto operace:*

- *Nahrazení výskytů znaku $\cdot$ libovolným slovem $w \in \mathcal{P}_\Sigma$*
- *Nahrazení výrazu $w_1 + w_2 + \cdots + w_k$ jedním pevným $w_i, i \in [k]$*
- *Nahrazení výskytů $/*$ libovolnou absolutní cestou z AP*

Symbolem $[r]$ označíme množinu všech absolutních cest generovaných regulárním výrazem r. Zřejmě $[r] \subseteq AP$.

Příklad 3. Pro cestu $/*$ je $[/*] = AP$.

Definice 32 (Les generovaný regulární cestou). *Nechť r je regulární cesta a strom $T = (V, E, L)$ je stromem s pojmenovanými vrcholy generovaný gramatikou $\mathcal{G}(A)$. Les stromů generovaný regulární cestou r je roven výrazu $[r]_T := [[r]]_T$.*

Nyní jsme si vybudovali základní pojmový aparát postačující k pohodlnému zápisu omezujících podmínek.

Zápis omezujících podmínek

Každá omezující podmínka bude formulí logiky prvního řádu. Stručně nyní vybudujme pojem pravdivé formule v logice prvního řádu.

Definice 33 (Jazyk prvního řádu). *Jazyk prvního řádu sestává z následujího*

(i) Logické spojky $\neg, \to$

(ii) *Kvantifikátor* $\forall$

(iii) *Symboly pro proměnné*

(iv) *Binární predikát* $=$

(v) *Pomocné symboly* $(,\ )$

(vi) *Symboly pro predikáty, navíc s každým je asociováno přirozené číslo udávající jeho aritu.*

(vii) *Symboly pro funkce, navíc s každou funkcí je asociováno celé číslo z* $\mathbb{N}_0$ *udávající její aritu. Funkcím arity* 0 *budeme říkat konstanty.*

V dalším textu budeme uvažovat jenom takové jazyky prvního řádu, které mají spočetně mnoho symbolů pro predikáty a funkce.

Definice 34 (Term)**.** *Termy se definují induktivně pomocí těchto tří pravidel:*

(i) *Každá proměnná je term.*

(ii) *Jsou-li* $t_1, t_2, \ldots, t_n$ *termy a* f *je* n*-ární funkce, je i* $f(t_1, t_2, \ldots, t_n)$ *term.*

(iii) *Každý term vznikne konečným počtem použití pravidel (i), (ii).*

Definice 35 (Formule)**.** *Formule se definují induktivně pomocí těchto čtyř pravidel:*

(i) *Nechť* P *je predikátový symbol arity* $n \in \mathbb{N}$ *a* $t_1, t_2, \ldots, t_n$ *jsou termy. Potom* $P(t_1, t_2, \ldots, t_n)$ *je atomická formule.*

(ii) *Nechť* A, B *jsou formule. Potom i* $\neg A, (A \to B)$ *jsou formule.*

(iii) *Nechť* x *je proměnná a* A *je formule. Potom* $(\forall x)A$ *je formule.*

(iv) *Každá formule vznikne konečným počtem opakování pravidel (i)–(iii).*

Pro úplnost poznamenejme, že existenční kvantifikátor resp. formule ve tvaru $(\exists x)P(x)$ je jen zkratka za $\neg((\forall x)\neg P(x))$ a že všechny běžné logické spojky je možno sestavit z implikace.

Výše uvedenými třemi definicemi jazyka prvního řádu, termu a formule jsme zadefinovali syntax logiky prvního řádu. Abychom byli precizní, doplňme ještě definice dalších několika pojmů, které budou dále používány.

Definice 36 (Volný a vázaný výskyt)**.** *Nechť* x *je proměnná jazyka prvního řádu* L *a* φ *je formule* L*. Pak pojem* x *má ve formuli* φ *volný výskyt je definován takto:*

(i) *Je-li* φ *atomická, pak* x *má ve* φ *volný výskyt, právě když se ve formuli* φ *proměnná* x *vyskytuje.*

(ii) Je-li φ je $(\neg\psi)$, pak x má ve φ volný výskyt, právě když má volný výskyt ve ψ.

(iii) Je-li φ je $(\alpha \to \beta)$, pak x má ve φ volný výskyt, právě když má volný výskyt v α nebo v β.

(iv) Je-li φ je $(\forall y)\psi$, pak x má ve φ volný výskyt, právě když jsou x a y různé symboly a x má volný výskyt v ψ.

Pokud se proměnná x ve formuli φ vyskytuje a nemá v ní volný výskyt, pak říkáme, že má ve φ vázaný výskyt.

Definice 37 (Otevřená a uzavřená formule)**.** *Je-li výskyt každé proměnné ve formuli φ volný, říkáme, že je otevřená. Je-li výskyt každé proměnné ve formuli φ vázaný, říkáme, že je uzavřená.*

Nyní již můžeme přistoupit k definování pojmu pravdivost formule v dané realizaci jazyka. Abychom mohli totiž říct, že validní instance je taková, ve které platí daná sada omezujících podmínek, musíme nejprve exaktně říci, co znamená slovo *platí*.

Pravdivost formulí logiky prvního řádu můžeme jednoduše řečeno zkoumat jednak absolutně (věty formálně odvozené z axiomů) např.

$$(\forall x)(P(x) \to P(x))$$

a jednak vzhledem k tzv. *realizaci jazyka*. Např. formule

$$(\forall x)(\forall y)\ x * y = y * x,$$

kde $*$ je funkční symbol arity 2, je zřejmě pravdivá v kontextu každé abelovské grupy $G = (A, e, ^{-1}, *)$, ale v kontextu obecné neabelovské grupy je nepravdivá. Oba zmíněné typy grup představují různé realizace nulární funkce e, unární $^{-1}$ a binární $*$. Definujme tedy poslední potřebný pojem *realizace jazyka*.

Definice 38 (Realizace jazyka 1. řádu)**.** *Nechť L je jazyk prvního řádu. Nechť $\mathfrak{M}$ je struktura obsahující*

(i) neprázdnou množinu M (univerzum),

(ii) pro každý n-ární predikátový symbol $P \in L$ relaci $P^{\mathfrak{M}} \subseteq M^n$,

(iii) pro každou n-ární funkci $f \in L$ zobrazení $f^{\mathfrak{M}} : M^n \to M$.

Pak $\mathfrak{M}$ je realizace jazyka L, prvky univerza M nazýváme individua.

Definice 39 (Ohodnocení proměnných)**.** *Libovolnou funkci e přiřazující každé proměnné jazyka L nějaké individuum univerza realizace jazyka L nazveme ohodnocení proměnných.*

Nyní již máme vybudován minimální logický aparát nutný k definici pravdivé formule logiky prvního řádu v dané realizaci jazyka. Tuto definici poté použijeme při definici validní instance AC-modelu, tedy instance A-modelu splňující navíc daný seznam omezujících podmínek – viz dále.

Definice 40 (Pravdivá formule)**.** *Nechť $\mathfrak{M}$ je realizace jazyka 1. řádu L, e je ohodnocení proměnných (viz předchozí definice). Opět induktivně definujme, že formule φ je pravdivá. To, že je formule φ v dané realizaci při ohodnocení proměnných e pravdivá, pak zapíšeme jako $\mathfrak{M} \models \varphi[e]$.*

1. *Pokud φ je $t_1 = t_2$ pro nějaké termy t_1, t_2, pak $\mathfrak{M} \models \varphi[e]$, pokud $e(t_1) = e(t_2)$.*
2. *Pokud je φ ve tvaru $\neg\psi$, pak $\mathfrak{M} \models \varphi[e]$, pokud není pravda, že $\mathfrak{M} \models \psi[e]$.*
3. *Pokud je φ ve tvaru $(\alpha \to \beta)$, pak $\mathfrak{M} \models \varphi[e]$, pokud není pravda, že $\mathfrak{M} \models \alpha[e]$, nebo $\mathfrak{M} \models \beta[e]$.*
4. *Pokud je φ ve tvaru $(\forall x)\ \psi$, pak $\mathfrak{M} \models \varphi[e]$, pokud pro každé individuum $m \in M$ je $\mathfrak{M} \models \psi[e(x/m)]$.*[3]

Pokud je formule φ pravdivá v dané realizace $\mathfrak{M}$ při jakémkoli ohodnocení proměnných e, píšeme $\mathfrak{M} \models \varphi$.

Poznamenejme, že autorství této definice náleží slavnému polskému logikovi Alfredu Tarskimu (1901-1983). Bližší informace je možno nalézti v [4].

Pomalu se již blížíme k definici omezující podmínky. Omezující podmínky budou schopny mimo jiné specifikovat i příslušnost nějaké hodnoty k datovému typu. Proto dále rovněž umožníme, abychom ke každému datovému typu mohli mít i sadu funkcí a predikátů schopných vyjádřit základní vztahy a operace v příslušném datovém typu. Předešleme tedy ještě tuto definici.

Definice 41 (Systém datových typů)**.** *Nechť $\mathcal{D}$ je množina datových typů. Ke každému datovému typu $D \in \mathcal{D}$ nechť je dána množina predikátových symbolů $\mathcal{P}(D)$ (každý s asociovanou aritou $n \in \mathbb{N}$) a množina funkčních symbolů $\mathcal{F}(D)$ (každá s asociovanou aritou $n \in \mathbb{N}_0$). Realizaci $\mathfrak{D}$ jazyka všech datových typů a příslušných predikátů a funkcí nazveme systém datových typů. Je-li univerzum realizace $\mathfrak{D}$ spočetná množina, pak říkáme, že systém datových typů je spočetný.*

Dále potřebujeme říci, co znamená „hodnota datového vrcholu“. Jak víme, instance daného A-modelu A je uspořádaná dvojice (T, ω), kde T je strom s pojmenovanými vrcholy a ω je funkce přiřazující vrcholům se jménem δ nějaké slovo z Σ^*. Pro naše účely potřebujeme sestrojit funkce, jež budou interpretovat řetězce z Σ^* jako prvky oborů hodnot konkrétních datových typů.

[3] $e(x/m)$ představuje funkci, která x přiřadí m a na všech $y \neq x$ je totéž co $e(y)$.

Definice 42 (Interpretace datového typu)**.** *Interpretace datového typu* $D = (s, M)$ *je zobrazení*

$$\omega^D : \Sigma^* \to M \cup \{\lrcorner\}.$$

Speciální hodnota $\lrcorner$ bude sloužit k označení takových slov ze Σ^*, kterým neodpovídá žádný prvek oboru hodnot daného datového typu. Příkladem funkce odpovídající předchozí definici může být funkce $\omega^{\mathbb{N}} : \Sigma^* \to \mathbb{N}$ přiřazující odpovídající přirozené číslo všem řetězcům složeným jen ze znaků $0, 1, \ldots, 9$, jejichž prvním znakem může být znak $+$, nebo $\lrcorner$ pro všechny ostatní řetězce. Tedy např. $\omega^{\mathbb{N}}(+0003214) = 3214$ a $\omega^{\mathbb{N}}(132a4) = \lrcorner$.

Omezující podmínka

Definice 43 (Omezující podmínka)**.** *Nechť* $\mathfrak{D}$ *je nejvýše spočetný systém datových typů. Nechť* $\mathfrak{M}$ *je realizace jazyka prvního řádu, která rozšiřuje realizaci* $\mathfrak{D}$ *o jazyky AP, RP, jazyk všech instancí (označme* $\mathcal{T}$*), funkci* ω *vracející pro každou jednovrcholovou instanci* $T = (\{v\}, \emptyset, \{(v, \delta)\}, \omega_T)$ *hodnotu* $\omega_T(v)$*, jazyk všech konečných podmnožin* $\mathcal{T}$ *a dvě funkce arity 2*

$$[\cdot]^{AP}_{\cdot} \;:\; AP \times \mathcal{T} \to 2^{\mathcal{T}},$$
$$[\cdot]^{RP}_{\cdot} \;:\; 2^{AP} \times \mathcal{T} \to 2^{\mathcal{T}}.$$

Nechť C *je formule v této realizaci* $\mathfrak{M}$ *obsahující právě jednu volnou proměnnou* I *(validovaná instance). Dále nechť všechny kvantifikované proměnné mají uvedenu doménu a ta je podmnožinou* $\mathcal{T}$*. Pak řekneme, že je formule* C *omezující podmínka.*

Úmluva. Nechť $T = (\{v\}, \emptyset, \{(v, \delta)\}, \omega_T)$ je jednovrcholová instance. Pak budeme psát krátce např. $\omega^{\mathbb{N}}(T)$ a budeme tím myslet $(\omega^{\mathbb{N}} \circ \omega)(T)$.

Pozorování 1. *Univerzum realizace* $\mathfrak{M}$ *z předchozí definice je spočetná množina.*

Důkaz. Jazyky AP, RP jsou množinami konečných posloupností znaků a jako takové jsou tedy spočetné. Jazyk všech konečných stromů s vrcholy pojmenovanými řetězci nad konečnou abecedou a některými vrcholy s konečným řetězcem jako hodnotou je zřejmě spočetný a jazyk všech konečných podmnožin spočetné množiny $\mathcal{T}$ je vždy spočetný (triviální). Navíc víme, že dokonce sjednocením spočetně mnoha spočetných množin dostaneme vždy spočetnou množinu. Tedy i realizace $\mathfrak{M}$ je spočetná. □

Úmluva. Od nynějška formule ve tvaru $(\forall x)\ (x \in A) \to \varphi$ budeme zkracovat jako $(\forall x \in A)\ \varphi$, kde φ je formule, A je „množina“ a x je proměnná. Dále formule ve tvaru $(\forall x_1, x_2, \ldots, x_n \in A)(\forall x_{n+1} \in A)\ \varphi$ budeme zkracovat jako $(\forall x_1, x_2, \ldots, x_{n+1} \in A)\ \varphi$, $n \in \mathbb{N}$. Formule budeme zkracovat opakovaným použitím těchto pravidel.

Příklad 4. Uveďme několik příkladů možných omezujících podmínek. Mějme systém datových typů tvořený jedním datovým typem (*integer*, $\mathbb{N}$) a přirozenou interpretaci datového typu *integer* funkci $\omega^{\mathbb{N}} : \Sigma^* \to \mathbb{N}$. V následující tabulce 4.1 jsou uvedeny příklad formulí a u každé je uvedeno, zda je ($\surd$) omezující podmínka, nebo není ($\times$).

$x + y > 0$	$\times$
$2 * (3 + 5) = 16$	$\surd$
$(\forall n)(\forall x)(\forall y)(\forall z)\ n > 2 \to (x^n + y^n \neq z^n)$	$\times$
$(\exists y)\ y \in [/*/y]_I$	$\surd$
$(\forall x)(\exists y)\ x \in [/g/e/\delta]_I \to (y \in [/g/v/\delta]_I \to \omega^{\mathbb{N}}(x) = \omega^{\mathbb{N}}(y))$	$\surd$
$(\exists x)(\exists y)\ x, y \in [/a/b/\delta]_I\ \&\ x \neq y\ \&\ \omega^{\mathbb{N}}(x) = \omega^{\mathbb{N}}(y)$	$\surd$

Tabulka 4.1: Příklady omezujících podmínek

Formule na třetím řádku je také zářným příkladem osvětlujícím důvody omezení domény kvantifikovaných proměnných na indukované podstromy dané instance.

V této chvíli víme, co je A-model, co je instance A-modelu, co jsou datové typy, jak se interpretují a můžeme přikročit k definici validní instance, tedy instance, na níž bude splněna navíc sada daných omezujících podmínek. Jelikož jsme definovali v A-modelu reference, požadujme, aby struktura validní instance rovněž splňovala uzavřenost na reference. Tomu odpovídá následující definice.

Definice 44 (Podmínky referenční uzavřenosti)**.** *Nechť $\mathcal{A}$ je A-model. Pro každý archetyp $A \in \mathcal{A}$ s typem τ_A definujme formuli $\varphi(A)$ ve tvaru*

$$(\forall r \in [/*/\beta]_I)(\exists m \in [/*/\alpha]_I)\ [/\tau_A]_r = [/\tau_A]_m\ \&\ \omega^{\mathbb{ID}}([/\iota/\delta]_r) = \omega^{\mathbb{ID}}([/\iota/\delta]_m).$$

Nechť $\Omega(\mathcal{A})$ je množina $\{\varphi(A) : A \in \mathcal{A}\}$. Pak množině Ω říkáme podmínky referenční uzavřenosti.

Validní instance

Definice 45 (Validní instance)**.** *Nechť S je instance A-modelu $\mathcal{A}$ a nechť C je nejvýše spočetná množina omezujících podmínek, $\Omega(\mathcal{A}) \subseteq C$. Pak řekneme, že S je validní instancí vůči množině omezujících podmínek C, pokud pro každou formuli $c \in C$ je $M \models c[e]$, kde $e(I) = S$.*

Nyní máme završeno budování formální teorie omezujících podmínek nad instancemi A-modelu. Validní instance jsou přesně ty stromy s pojmenovanými vrcholy a přiřazenými hodnotami vrcholům pojmenovaným δ, na nichž

jsou pravdivé všechny dané omezující podmínky. A-model spolu s konečnou množinou omezujících podmínek budeme nazývat AC-model. Pro úplnost tedy přidejme definici.

Definice 46 (AC-model)**.** *Nechť A je A-model a nechť C je konečná množina omezujících podmínek. Pak dvojici (A, C) nazveme AC-model.*

Závěr

Všimněme si, že teorii AC-modelu jsme vybudovali ze dvou částí – z teorie regulárních stromových gramatik a logiky prvního řádu. Konkrétní regulární stromová gramatika nám generuje jazyk všech možných syntakticky správných stromů, C-model potom pomocí predikátové logiky definuje omezující podmínky, které musí konkrétní stromy s hodnotami splňovat, aby byly označeny za validní. Regulární stromovou gramatiku přitom modeluje A-model, omezující podmínky C-model.

To, že jsou obě teorie na sobě poměrně nezávislé, je velice výhodné. Konkrétní datový obsah stromů navíc definujeme zcela odděleně od formy (syntaxe stromů) pomocí funkce ω definované na datových δ-vrcholech. Tím, že datový vrchol není oproti jiným listům ničím výjimečný, dostáváme naprosto konzistentně na konceptuální úrovni i smíšené obsahy vrcholů. Naprosto svobodně též můžeme zavést interpretace nedelta listů resp. vrcholů s právě jedním delta synem jako atributy nebo jako elementy bez obsahu resp. s jednoduchým obsahem.

Kapitola 5

Příklady a vlastnosti AC-modelů

V této kapitole uveďme několik konkrétních příkladů AC-modelů zejména kvůli demonstraci síly omezovacích podmínek. V klasických relačních konceptuálních modelech jsou některé z uvedených podmínek jen stěží implementovatelné. Dále uvedeme několik teoretických vlastností našeho AC-modelu – zejména zmíníme omezení logiky prvního řádu a nerozhodnutelnost problému existence validní instance, dále se pak podíváme na vyjadřovací schopnosti AC-modelu.

Úmluva. V následujících příkladech budeme vynechávat psaní kořenového elementu *instance*. Zápis $[a], a \in AP \cup RP$ bez udání kontextového stromu znamená $[a]_I$.

Příklad 1 (Datové typy). Chceme-li říci, že libovolný vrchol x kdekoli ve stromu, který má navíc asociovánu hodnotu, musí mít tuto hodnotu interpretovatelnou jako přirozené číslo, pak stačí, když napíšeme formuli

$$(\forall x \in [/*/x/\delta]_I)\ \omega^{\mathbb{N}}(x) \neq \lrcorner.$$

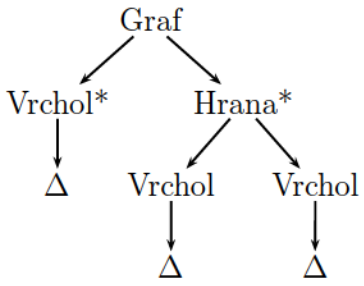
Příklad 2 (Orientovaný graf). Orientovaný graf je dvojice (V, E), $E \subseteq V \times V$. Máme tedy jednak seznam vrcholů a jednak seznam hran, jakožto dvojic vrcholů. A-model této matematické struktury může vypadat například jako na obrázku 5.1.

Přidáním těchto dvou omezujících podmínek obdržíme AC-model orientovaného grafu.

$$(\forall x, y \in [/graf/vrchol/\delta]_I)\ \omega^{\mathbb{N}}(x) = \omega^{\mathbb{N}}(y) \rightarrow x = y$$

$$(\forall x \in [/graf/hrana/vrchol/\delta]_I)(\exists y \in [/graf/vrchol/\delta]_I)\ \omega^{\mathbb{N}}(x) = \omega^{\mathbb{N}}(y)$$

Všimněme si, že první formulí jsme vyjádřili integritní podmínku „primární klíč“, resp. unikátnost a druhou podmínkou jsme vyjádřili „cizí klíč“. Vidíme



Obrázek 5.1: A-model orientovaného grafu

tedy, že primární integritní podmínky z relačních databází jsou v AC-modelu zapsatelné naprosto triviálně.

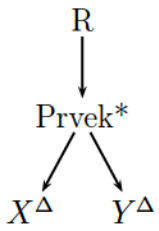
Příklad 3 (Neorientovaný graf obsahující K_3). Mějme stejný A-model jako v předchozím příkladě, vůbec jej nemusíme měnit. Definujme cestu $P_E = /graf/hrana$ a $P_{E1} = /vrchol[0]/\delta$ a $P_{E2} = /vrchol[1]/\delta$. Definujme dále binární predikát *JeHrana* jako

$$JeHrana(x,y) := (\exists e \in [P_E]_I)\ \left(\omega^{\mathbb{N}}([P_{E1}]_e) = \omega^{\mathbb{N}}(x)\ \&\ \omega^{\mathbb{N}}([P_{E2}]_e) = \omega^{\mathbb{N}}(y)\right) \vee \left(\omega^{\mathbb{N}}([P_{E1}]_e) = \omega^{\mathbb{N}}(y)\ \&\ \omega^{\mathbb{N}}([P_{E2}]_e) = \omega^{\mathbb{N}}(x)\right).$$

Jedinou podmínku, kterou stačí přidat, je podmínka

$$(\exists x,y,z \in [/graf/vrchol/\delta]_I)\ x \neq y\ \&\ y \neq z\ \&\ z \neq x\ \& \\ \&\ JeHrana(x,y)\ \&\ JeHrana(y,z)\ \&\ JeHrana(z,x).$$

Příklad 4 (Tranzitivní relace). Namodelujme obecnou tranzitivní binární relaci R racionálních čísel s konečným desetinným rozvojem pouze jako seznam dvojic čísel. A-model je na obrázku 5.2.



Obrázek 5.2: Tranzitivní relace

Omezující podmínky pak můžeme napsat například v tomto tvaru:

$$(\forall r \in [/r/prvek]_I)\ \omega^{\mathbb{Q}}([/x/\delta]_r) \neq \text{ↄ} \ \&\ \omega^{\mathbb{Q}}([/y/\delta]_r) \neq \text{ↄ}$$

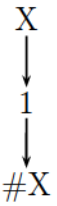
$$VRelaci(x,y) := (\exists r \in [/r/prvek]_I)\ \omega^{\mathbb{Q}}([/\delta]_x) = \omega^{\mathbb{Q}}([/x/\delta]_r)\ \& \\ \omega^{\mathbb{Q}}([/\delta]_y) = \omega^{\mathbb{Q}}([/y/\delta]_r)$$

$$(\forall r_1, r_2 \in [/r/prvek]_I)\ \omega^{\mathbb{Q}}([/y/\delta]_{r_1}) = \omega^{\mathbb{Q}}([/x]_{r_2}) \rightarrow VRelaci([/x]_{r_1}, [/y]_{r_2})$$

Chtěli-li bychom nyní ještě např. říci, že daná relace jsou navíc hrany neorientovaného grafu, pak jednoduše přidáme pouze axiom

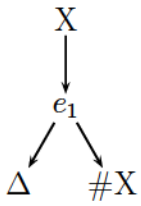
$$(\forall x \in [/r/prvek/x]_I)(\forall y \in [/r/prvek/y]_I)\ VRelaci(x,y) \rightarrow VRelaci(y,x).$$

Příklad 5 (Jednoduchá rekurze). Nejjednodušší příklad rekurzivního A-modelu mající nějakou instanci je na obrázku 5.3.



Obrázek 5.3: Nejjednodušší rekurze

Všimněme si použití selektoru 1, jenž umožňuje libovolný podobsah. Instancí tohoto A-modelu je libovolná orientovaná cesta s vrcholy pojmenovanými x. Jednoduše můžeme tento model rozšířit tak, abychom namodelovali rostoucí posloupnost. Model pak dostane tvar jako na obrázku 5.4.



Obrázek 5.4: Rostoucí posloupnost

Po doplnění omezující podmínky ve tvaru

$$(\forall x \in [/*/x]_I)\ [/x]_x = \emptyset \vee \omega^{\mathbb{N}}([/\delta]_x) < \omega^{\mathbb{N}}([/x/\delta]_x)$$

dostáváme AC-model modelující rostoucí posloupnost. Pokud bychom selektor e_1 nahradili selektorem 1, byl by možnou instancí i strom (v XML zápisu)

```
<x>5
  <x>
    <x>7
      <x>8
        <x/>
      </x>
    </x>
  </x>
</x>
```

V tomto případě kterýkoli element **x** může, nebo nemusí mít obsah. Omezující podmínku tedy musíme přepsat do tvaru

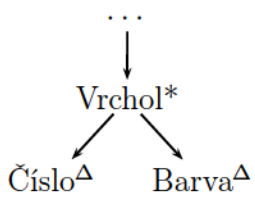
$$(\forall x \in [/*/x]_I)(\forall y \in [/*/x/\delta]_x)\ [/\delta]_x = \emptyset \vee \omega^{\mathbb{N}}([/\delta]_x) < \omega^{\mathbb{N}}([/\delta]_y),$$

neboli libovolný potomek vrcholu x (s hodnotou), který ma asociovánu hodnotu, musí mít tuto hodnotu ostře větší (na přirozených číslech) než je hodnota u vrcholu x.

Příklad 6 (Neorientovaný graf s $\chi(G) \leq 3$). Grafy s chromatickým číslem nejvýše 3 jsou jednoduše popsatelné touto jednoduchou axiomatikou s jedním binárním predikátem $\backsim$ představujícím hranu a třemi unárními predikáty R, G, B takovými, že např. $R(x)$ představuje tvrzení „*x má barvu R*“.

$$(\forall x)(\forall y)\ x \backsim y \rightarrow y \backsim x$$
$$(\forall x)\ \neg(x \backsim x)$$
$$(\forall x)\ (R(x)\&\neg(G(x) \vee B(x))) \vee (G(x)\&\neg(R(x) \vee B(x))) \vee (B(x)\&\neg(R(x) \vee G(x)))$$
$$(\forall x)(\forall y)\ (x \backsim y) \rightarrow (R(x) \rightarrow \neg R(y))$$
$$(\forall x)(\forall y)\ (x \backsim y) \rightarrow (G(x) \rightarrow \neg G(y))$$
$$(\forall x)(\forall y)\ (x \backsim y) \rightarrow (B(x) \rightarrow \neg B(y))$$

Pozměnili-li bychom tedy výše uvedený A-model pro graf tak, že seznam vrcholů bude tohoto tvaru,



Obrázek 5.5: Obarvený vrchol

pak přepsáním posledních čtyř podmínek axiomatiky grafu s $\chi(G) \leq 3$ do C-modelu dostaneme omezení na nejvýše tříbarevné grafy. Tím jsme samozřejmě také mimo jiné ukázali, že problém rozhodnutí, je-li daný strom validní instance AC-modelu, je NP-těžký, neboť jsme na něj redukovali NP-úplný problém třiobarvetelnosti.

Závěr

Vidíme, že pomocí logiky prvního řádu dokážeme zapsat spoustu zajímavých sémantických podmínek, které zcela přesahují výrazový rámec klasických ER diagramů a většiny známých konceptuálních modelů. Speciálně klíče a cizí klíče jsou naprosto triviální. Ale ani logika prvního řádu nedokáže samozřejmě vyjádřit jakoukoli možnou integritní podmínku.

5.1 Omezení logiky prvního řádu

Ačkoli je AC-model již poměrně silným modelovacím nástrojem, přesto v něm nedokážeme namodelovat libovolnou strukturu. Předpokládejme, že máme A-model z příkladu 2, že pomocí něj modelujeme neorientované grafy a že bychom chtěli vyjádřit omezující podmínky tak, aby validní instance byly pouze souvislé neorientované grafy. Ukažme, že to v logice 1. řádu, a tedy ani v AC-modelu není možné.

Věta 5 (Souvislost a logika 1. řádu)**.** *Souvislost neorientovaného grafu není vyjádřitelná formulemi logiky prvního řádu.*

Důkaz. Jazyk prvního řádu neorientovaných grafů má jeden binární predikát $\backsim$, který znamená

$$a \backsim b \equiv \{a, b\} \in E.$$

Neorientovaný graf je pak libovolná struktura, která splňuje tyto dva axiomy:

$$(\forall x)(\forall y)\ x \backsim y \rightarrow y \backsim x \tag{5.1}$$

$$(\forall x)\ \neg(x \backsim x) \tag{5.2}$$

Uvažujme grafy s alespoň dvěma vrcholy. Pak můžeme přidat k jazyku dvě konstanty a, b představující 2 pevné různé vrcholy. Předpokládjme nyní, že existuje formule logiky prvního řádu P, která říká, že graf s vrcholy a, b je souvislý. Zkonstruujme posloupnost formulí logiky prvního řádu $\{P_k\}$ takovou, že P_k bude pro pevné $k \in \mathbb{N}$ formule ekvivalentní slovnímu tvrzení

„*Vrcholy a, b nejsou spojeny cestou délky nejvýše k.*“

Pro libovolné pevné k je konstrukce takovéto formule triviální. Např.

$$P_1 = \neg(a \rightsquigarrow b)$$
$$P_2 = (\forall x)\ (\neg(a \rightsquigarrow x) \vee \neg(x \rightsquigarrow b)) \text{ atd.}$$

Nechť Γ je nekonečná množina formulí

$$\Gamma = \{axiom\ 5.1, axiom\ 5.2, \mathbf{P}, P_1, P_2, P_3, \dots\}$$

Každá konečná podmnožina $\Phi \subset \Gamma$ má model. To je zřejmé, neboť je-li Φ konečná, pak obsahuje konečně mnoho formulí $P_k, k \in \mathbb{N}$. Nechť $n \in \mathbb{N}$ je maximální takový index. Pak cesta délky $n+1$, kde a, b budou oba krajní vrcholy, je modelem teorie Φ. Pak podle věty o kompaktnosti ale dostáváme, že i Γ má model. Poznamenejme, že věta o kompaktnosti logiky prvního řádu je jednoduchým důsledkem známé Gödelovy věty o úplnosti.

Nechť $\mathcal{M}$ je tedy modelem teorie Γ. Pak $\mathcal{M}$ je graf, protože splňuje oba axiomy. Obsahuje vrcholy a, b a ty nejsou spojeny cestou žádné konečné délky. To je ale spor s tvrzením formule P, a tedy žádná taková formule P v logice prvního řádu neexistuje. □

Takto jsme pomocí věty o kompaktnosti dokázali, že vlastnost *být souvislý graf* (v jazyce databází integritní podmínka) není postihnutelná logikou prvního řádu. Tato vlastnost je ale vyjádřitelná například už ve slabé logice druhého řádu, ve které máme navíc k dispozici kvantifikování přes predikátové symboly (a tedy přes množiny).

5.2 Rozhodnutelnost existence instance

Dalším problémem, který musíme při zkoumání našeho konceptuálního AC-modelu zvážit, je problém rozhodnutelnosti existence instance.

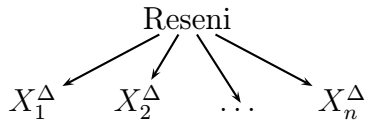
Problém. Je dán AC-model A. Existuje nějaká validní instance modelu A?

Věta 6 (Nerozhodnutelnost AC-modelu)**.** *Problém existence validní instance daného AC-modelu (označme ACE) je algoritmicky nerozhodnutelný.*

Důkaz. Nerozhodnutelnost existence instance AC-modelu ukážeme redukcí desátého Hilbertova problému (HTP), který je známým nerozhodnutelným problémem.

Instancí HTP je libovolná diofantická rovnice. Otázkou je, zda má tato rovnice celočíselné řešení. Formálně máme tedy polynom n proměnných P s celočíselnými koeficienty a ptáme se, existuje-li $\overline{x} \in \mathbb{Z}^n$, pro které $P(\overline{x}) = 0$.

Mějme tedy instanci HTP – polynom $P, deg(P) = n$. Uvažujme A-model z obrázku 5.6.



Obrázek 5.6: HTP→ACE

K němu doplňme jedinou omezující podmínku ve tvaru

$$x_i := \omega^{\mathbb{N}}([/reseni/x_i/\delta]_I), \tag{5.3}$$

$$\mathcal{P}(x_1, x_2, \ldots, x_n) = 0. \tag{5.4}$$

Přitom symbolem $\mathcal{P}(x_1, x_2, \ldots, x_n)$ máme na mysli rozvinutý zápis daného polynomu pomocí výše definovaných x_i. Tímto dostáváme AC-model A. Validní instance AC-modelu A existuje, právě když existuje řešení dané diofantické rovnice. Tedy problém existence validní instance AC-modelu je nerozhodnutelný. □

Nerozhodnutelnost AC-modelu nijak nepřekvapuje. AC-model tak, jak jsme jej definovali, je poměrně silný. Máme k dispozici navíc datové typy a libovolné operace nad nimi. Jednoduchým trikem také můžeme dokázat, že problém ekvivalence dvou daných AC-modelů (označme 2AC) je alespoň tak těžký jako problém ACE.

Věta 7 (Problém ekvivalence AC-modelů (ACE → 2AC))**.** *Problém, zda dva dané AC-modely generují stejný jazyk, je alespoň tak těžký jako problém ACE.*

Důkaz. Máme-li totiž instanci problému ACE, tedy konkrétní AC-model A, pak instancí problému 2AC bude AC-model A a AC-model $\mathcal{A}_\emptyset$ složený s libovolného A-modelu a C-modelu obsahujícího logicky nepravdivou formuli. Pak instance AC-modelu A existuje, právě když AC-modely A a $\mathcal{A}_\emptyset$ neprodukují stejný regulární stromový jazyk. □

5.3 Expresivita A(C)-modelu

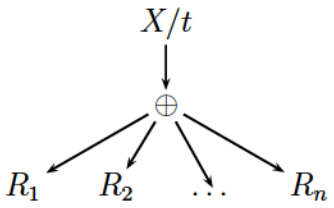
Nejprve se zaměřme na samotný A-model. Expresivitu A-modelu budeme zkoumat v maximálně možné obecné míře. Každý konceptuální model určený pro modelování XML schémat by měl být schopen vyjádřit co nejširší podmnožinu třídy regulárních stromových gramatik. Ideálně by ke každé regulární stromové gramatice měl existovat konceptuální model, jehož překladem je až na isomorfismus právě tato gramatika.

Věta 8 (Vztah RSG a A-modelu)**.** *Je-li G regulární stromová gramatika s jediným axiomem Instance, jenž se nevyskytuje na pravé straně žádného přepisovacích pravidla, pak existuje A-model A takový, že $L(G) \cong L(\mathcal{G}(A))$.*

Důkaz. Vytvořme ke gramatice G gramatiku G_N produkčně ekvivalentní ($L(G) = L(G_N)$) takovou, že množiny terminálů a neterminálů budou v G a G_N identické, ale v G_N bude každý neterminál na levé straně nejvýše jednoho pravidla. Upravme dále všechna pravidla v G_N do tvaru

$$X \to t(R_1 + R_2 + \cdots + R_n), n \in \mathbb{N}_0,$$

kde X je neterminál, t je terminál a pro $i \in [n]$ jsou R_i regulární výrazy nad neterminály neobsahující operátor $+$. To, že je možno pravidla upravit tímto způsobem ukážeme v dodatku k důkazu. Pro každé takovéto pravidlo zkonstruujeme zvláštní archetyp reprezentující pouze toto pravidlo. Archetyp tedy bude vypadat jako na obrázku 5.7.



Obrázek 5.7: Archetyp pravidla $X \to t(\sum_{i=1}^{n} R_i)$

Všechny $R_i, i \in [n]$ navíc rozviňme tak, že všechny výskyty neterminálů v regulárním výrazu R_i budeme prefixovat symbolem #, tedy z nich uděláme substituce.

Každý archetyp bude obsahovat tedy právě jeden ryzí metaneterminál a ten bude přiřazen kořeni příslušného stromu. Má-li gramatika G konečně přepisovacích pravidel, dostáváme takto konečný A-model A. To, že $L(G) = L(\mathcal{G}(A))$, je zřejmé z definice funkce $\mathcal{G}$.

Ještě ukažme, jak lze k danému regulárnímu výrazu R zkonstruovat regulární výraz R' ve tvaru $R' = R_1 + R_2 + \cdots + R_n$ tak, aby jednak oběma výrazům R i R' odpovídal stejný jazyk a jednak regulární výrazy $R_i, i \in [n]$ neobsahovaly operátor $+$. Tvaru, kdy regulární výraz je „součtem“ regulárních výrazů neobsahujících operátor $+$, budeme říkat *rozvinutý*.

To je ale triviální, uvědomíme-li si paralelu, že operaci spojení výrazů můžeme chápat jako násobení, operaci výběru jako sčítání. Samozřejmě s tím, že násobení má v zápise vyšší prioritu než sčítání. Výsledný regulární výraz pak dostaneme „roznásobením“ vstupního regulárního výrazu, tedy odstraněním závorek.

Nyní formálně. Zkonstruujme transformační funkci

$$\pi : Regex \to Regex$$

převádějící regulární výraz R na ekvivalentní rozvinutý regulární výraz.

(i) Je-li R rozvinutý, pak $\pi(R) = R$.

(ii) Jsou-li $A = \sum_{i=1}^{m} a_i, B = \sum_{i=1}^{n} b_i$ rozvinuté, pak

$$\pi(A + B) = \sum_{i=1}^{m} a_i + \sum_{i=1}^{n} b_i$$

(iii) Jsou-li $A = \sum_{i=1}^{m} a_i, B = \sum_{i=1}^{n} b_i$ rozvinuté, pak

$$\pi(A \cdot B) = \sum_{\substack{i \in [m] \\ j \in [n]}} a_i \cdot b_j$$

(iv) Je-li $A = \sum_{i=1}^{m} a_i$ rozvinutý, pak

$$\pi(A^*) = (\prod_{i=1}^{m} a_i^*)^*$$

Takto jsme jednoduše axiomaticky popsali funkci π, která transformuje libovolný regulární výraz na ekvivalentní rozvinutý. A tím je celý důkaz hotov. □

Právě jsme dokázali, že ke každé regulární stromové gramatice G existuje A-model, který ji modeluje. A-model zkonstruovaný podle důkazu této věty má ale tolik archetypů, kolik je v gramatice G_N pravidel. Samozřejmě se dá zkonstruovat i algoritmus, jenž bude produkovat A-modely s co nejméně archetypy. My jsme ale chtěli dokázat jen existenční tvrzení. Například je zřejmé, že sestavíme-li z neterminálů dané gramatiky orientovaný graf, kde z každého neterminálu X povedou hrany do všech neterminálů, které se vyskytují v modelu obsahu pravidla s X na levé straně, a tento graf bude acyklický, pak stačí na modelování této gramatiky jen jeden archetyp.

Kapitola 6

Logická úroveň

V předchozí kapitolách jsme definovali vlastní jednoduchý hierarchický AC-model a ukázali jeho sílu. Zaměřme se nyní na zápis AC-modelu v konkrétních jazycích. Úlohu si, jak se nabízí, rozdělíme na dvě části:

1. Volba jazyka pro zápis A-modelu.
2. Volba jazyka pro zápis omezujících podmínek.

6.1 Reprezentace A-modelu

Součástí definice AC-modelu byla definice funkce

$$\mathcal{G} : \textit{A-modely} \rightarrow \textit{RSG},$$

zobrazující konkrétní A-model na regulární stromovou gramatiku. Na logické úrovni potřebujeme takto vzniklou gramatiku reprezentovat.

Definice 47 (XML schéma jazyk)**.** *Nechť $\mathcal{S}$ je libovolný neprázdný jazyk. Dále nechť existuje funkce $F_{\mathcal{S}}$ zobrazující slova tohoto jazyka na regulární stromové gramatiky. Pak řekneme, že $\mathcal{S}$ je jazykem pro zápis XML schémat a slovům tohoto jazyka budeme říkat XML schémata.*

Pomocí této definice můžeme naprosto přirozeně definovat i instanci schématu resp. XML dokument validní podle konkrétního schématu.

Definice 48 (XML dokument validní podle schématu)**.** *Nechť $\mathcal{S}$ je XML schéma jazyk a nechť $S \in \mathcal{S}$ je konkrétní schéma v tomto jazyce. Pak řekneme, že XML dokument D je validní podle schématu S, pokud je jeho DOM strom bez datových vrcholů prvkem $L(F_{\mathcal{S}}(S))$.*

Stanovme si doplňující podmínku, že když už uvažujeme v kontextu XML, tak budeme chtít, aby i každé XML schéma bylo XML dokumentem.

Volba vhodného jazyka pro zápis XML schémat[1] archetypů je naprosto klíčovou pro celý návrh polymorfního informačního systému, a proto se ji budeme snažit provést na matematických, resp. teoreticko–informatických podkladech teorie regulárních stromových gramatik.

Pro úplnost uveďme ještě pár poznámek k formátu XML a zmiňme, že XML není ničím novým a převratným.

Extensible Markup Language (XML)

Jazyk XML i technologie s ním související jsou v současnosti velice rozšířené. Obecně se dá říci, že stromová struktura XML dokumentů zásadním způsobem usnadňuje výměnu libovolných dat nebo jejich následné zpracování. Často se v literatuře vyskytuje tvrzení, že jazyk XML jako takový je samovysvětlující. To je ale chybný pohled na věc. Jedině existence odpovídajích XML schémat dává XML dokumentu sémantický význam.

Jazyk XML je velice používáný i všude tam, kde potřebujeme uchovávat metadata ve strukturované podobě. Například generátor dokumentace jazyka C++ nejprve vygeneruje XML a z něj je poté možno vytvořit třeba HTML stránky, zdrojový kód systému LaTeX, postscriptový soubor apod. Ve formátu XML můžeme obecně vzato uchovávat dokumenty dvou základních typů:

1. *Document-Centric Documents* – jsou charakterizovány nižší mírou strukturovanosti, většinou obsahují dlouhé nestrukturované texty. Příkladem mohou být knihy ve formátu XML, kde je provedeno jen základní členění na kapitoly a odstavce. Většinou také velice záleží na pořadí podelementů nějakého elementu (např. implicitní řazení kapitol bez udání čísla kapitoly).

2. *Data-Centric Documents* – tyto dokumenty naopak mají velkou míru strukturovanosti a typicky jsou vytvořeny automaticky a určeny pro počítačové zpracování.

Veškeré XML dokumenty uvažované v dalším textu budou náležet do kategorie *Data-Centric Documents*. K dosažení vyšší efektivity dotazování totiž potřebujeme jemnější strukturu XML.

Hlavním důvodem, proč je XML v současnosti tak oblíbeno, je, že většina strukturovaných dokumentů ma hierarchickou (stromovou) strukturu a XML jako takové je víceméně jen dobře uzávorkovaným průchodem do hloubky, kde *preorder* akcí je zapsání otevírací závorky ve tvaru `<Jmeno *>` a *postorder* akcí zápis uzavírací závorky ve tvaru `</Jmeno>`. Na pozici hvězdičky může být

[1] V 1. pádě čísla jednotného budeme jazyku pro zápis XML schémat říkat krátce XML schéma jazyk. Jedná se sice o anglikanismus, který by ale neměl příliš narušit jazykovou stavbu práce.

uveden seznam atributů (parametrů) s hodnotami `par1="val1" par2="val2"` ..., případně i atributy bez hodnot. [2]

XML dokumenty tedy odpovídají výstupu tohoto průchodu do hloubky puštěného na kořen stromu, `dfs(Vrchol v)`:

1. Je-li vrchol `v` datový, vypiš jeho obsah a skonči.
2. Vypiš `<v.Jmeno v.Atributy>`,
3. pro všechny syny vrcholu `v` proveď `dfs(syn)`,
4. vypiš `</v.Jmeno>`.

Jen poznamenejme, že validaci toho, je-li dokument správně uzávorkovaný, lze provést triviálně v čase $O(N)$, kde N je počet znaků XML dokumentu, a s pamětí lineární počtu vrcholů odpovídajícího stromu. Zpětné vytvoření stromu je samozřejmě také lineární jak v čase, tak v paměti. I právě to, že konstrukce parseru pro XML formát jako takový je poměrně triviální záležitostí, způsobuje, že stále více dokumentů se převádí do XML. Příkladem mohou být např. konfigurační soubory, kde přeci jen i samotný parser ne–XML formátu nemusí být zcela jednoduchý.

6.1.1 Expresivita XML schémat

Nejprve opět vyslovme několik definic, které nám umožní porovnávat různé XML schéma jazyky mezi sebou a na matematickém základě vybrat ten nejvhodnější.

Definice 49 (Dokumenty generované XML schématem)**.** *Symbolem $\mathcal{D}$ označme univerzum všech XML dokumentů (verze 1.0) a symbolem $\mathcal{S}$ univerzum všech jazyků pro zápis XML schémat. Nechť $S \in \mathcal{S}$ je konkrétní XML schéma jazyk a $I \in S$ je konkrétní schéma v jazyce S. Pak jako $\Re(I)$ označme podmnožinu třídy $\mathcal{D}$ všech dokumentů validních vůči schématu I. Této množině budeme říkat „dokumenty generované XML schématem“.*

Jedno konkrétní schéma nám tedy okamžitě dělí třídu $\mathcal{D}$ na dvě disjunktní části – validní a invalidní dokumenty.

Definice 50 (Expresivita XML schéma jazyků)**.** *Na množině $\mathcal{S}$ všech XML schéma jazyků zcela přirozeně zaveďme částečné uspořádání $\preccurlyeq$ tak, že pro dva XML schéma jazyky $A, B \in \mathcal{S}$ platí $A \preccurlyeq B$, pokud*

$$\forall P \subseteq \mathcal{D} : (\exists I \in A, \Re(I) = P) \Longrightarrow (\exists J \in B, \Re J = P)\,. \tag{6.1}$$

[2] Obsahem XML jsou ještě tzv. processing instructions a další prvky jazyka, nicméně stromová struktura je zákadní myšlenkou vzniku XML.

Pokud platí $A \preccurlyeq B$ a zároveň $B \preccurlyeq A$ říkáme, že jazyky A a B jsou stejně expresivní (mají stejnou vyjadřovací sílu). Pokud platí $A \preccurlyeq B$, ale neplatí $B \preccurlyeq A$, říkáme, že jazyk A je ostře méně expresivní než jazyk B, a zapisujeme to jako $A \prec B$.

Jinými slovy je tedy XML schéma jazyk A méně nebo stejně expresivní jako jazyk B, pokud to, co se dá vyjádřit v jazyce A, lze vyjádřit i v jazyce B.

Nyní jsme si tedy zadefinovali ideální způsob, jak porovnávat mezi sebou jazyky pro zápis XML schémat. Uvidíme, že přestože jsou aktuálně běžně používané jazyky pro zápis XML schémat velice různorodé, hlavní zástupci jsou porovnatelní výše definovanou relací. Velká spousta článků a jiných vědeckých prací porovnává různé jazyky pro zápis XML schémat většinou bez formálního kontextu a to pouze tím, že upozorňují na ty či ony vlastnosti toho či onoho jazyka. Vyskytují se pak například podobné teze:

> *„Jazyk A umí vyjadřovat libovolné intervalové kardinality, jazyk B zase má objektový prvky a umí dědičnost, což je naprosto fantastická vlastnost, které mu dává obrovskou sílu. Jazyk C má ale zase lepší podporu vestavěných datových typů atp.“* [3]

Porovnávat expresivitu jazyků takovýmito *ad–hoc* metodami by mělo být při výběru vhodného jazyka až sekundární. Na prvním místě *musí* být matematický rozbor expresivity.

Nyní trochu předbíháme, ale příkladně velice často vyzdvihované objektové prvky jazyka W3C XML Schema nikterak *nezvyšují* jeho vyjadřovací schopnosti, jelikož vždy lze zkonstruovat ekvivalentní „neobjektové schéma“. To, že jsou tyto prvky v jazyce W3C XML Schema navíc definovány poměrně nekonzistentně, je již věc druhá a bude to obsahem dalšího textu.

6.1.2 Jazyky pro zápis XML schémat

Pro tvorbu schémat XML dokumentů existuje řada běžně používaných jazyků. Za všechny jmenujme jazyky Document Type Definition (DTD), W3C XML

[3] Již na tomto místě poznamejme, že podobné věty se bohužel vyskytují i v odborných a dokonce publikovaných textech např. ohledně tzv. substitučních skupin jazyka W3C XML Schema. Autoři přitom často nekriticky přebírají tvrzení, místo aby zhodnotili objektivní přínos těchto prvků např. na základě matematické teorie.

Schema [5][4] a RELAX NG[5] [6]. V dalším textu bude řeč ještě o Schematronu.

První jmenovaný nesplňuje podmínku, že schémata v něm napsaná budou XML dokumenty[6], navíc je jeho vyjadřovací síla oproti dvěma zbývajícím velice slabá. Detailnější popis jazyka DTD proto tedy uvádět nebudeme, neboť je pro tuto práci zcela irelevantní.

Podívejme se nyní na hlavní rysy jazyků RELAX NG a W3C XML Schema a zhodnoťme jejich výhody a nevýhody, zejména s důrazem na expresivitu obou zmíněných jazyků. Toto zhodnocení a porovnání provedeme na matematické úrovni, protože jen ta může vést k rigorózně podložitelným tvrzením o uvedených dvou jazycích.

Krátce a stručně nyní představme dva výše uvedené jazyky. Detailní popis je možno nalézti v odkazovaných specifikacích. U obou jazyků zmíníme pouze základní kostru obsahující nejdůležitější prvky těchto dvou jazyků pro zápis XML schémat.

6.1.3 W3C XML Schema

Názvem W3C XML Schema míníme jazyk podle specifikace [5]. Historický vývoj tohoto jazyka rozebírat nebudeme, neboť není pro tuto práci důležitý. Velice stručně si nyní tedy jazyk W3C XML Schema představme a kriticky zhodnoťme jeho nejpodstatnější rysy. V této podkapitole budeme používat pro jazyk W3C XML Schema zkratku XS.

Schémata v jazyce XS jsou XML dokumenty. Základem jazyka jsou jednoduché a složené datové typy.

Jednoduché datové typy

Obor hodnot jednoduchého datového typu je podmnožinou Σ^*, kde Σ je množina přípustných znaků (přesná definice množiny Σ viz [5]). Jednoduché datové typy jsou tedy vždy textové, můžeme ale specifikovat různá omezení. XS obsahuje sadu vestavěných datových typů rozdělenou do dvou skupin *primitivních* a *odvozených* typů. Do množiny primitivních jednoduchých datových typů patří

[4]Většina anglicky psaných článků uvádí vždy vysvětlení, že bude rozlišovat mezi XML schema a XML Schema, kdy první označuje libovolné XML schéma a druhé jazyk konsorcia W3C. My takto v této práci postupovat nebudeme. Budeme-li mít na mysli jazyk XML Schema společnosti W3C, budeme striktně psát W3C XML Schema. Dle mého subjektivního názoru je volba názvu „XML Schema“ konsorciem W3C pro konkrétní jazyk neetická a důvody byly zřejmě vyloženě komerční. V praxi totiž spousta lidí víceméně nerozlišuje mezi XML schema a W3C XML Schema, neboť jiný jazyk ani neznají. Přitom jazyků pro zápis XML schémat existují desítky a W3C XML Schema zdaleka nemá mezi nimi dominantní postavení co do kvality.

[5]Výslovnost je relaxing.

[6]To by nemusel být až takový problém. Kdyby jeho vyjadřovací síla byla vynikající, bylo by snadné vytvořit obálkový XML formát schémat v DTD. Opak je ale pravdou.

typy v tabulce 6.1.

Datový typ	Popis
`string`	Řetězec znaků ze Σ
`boolean`	Logická hodnota z $\{true, false, 1, 0\}$
`decimal, float, double`	Pseudoreálná čísla s různou přesností
`duration`	Doba trvání z $Y \times M \times D \times H \times M \times S$
`dateTime`	Zakódovaný datum a čas
`time, date`	Čas a datum
`gYear, gMonth, gDay`	Pouze rok, měsíc nebo den
`gYearMonth, gMonthDay`	Kombinace předchozích
`hexBinary`	Hexadecimální číslo
`base64Binary`	Binární data kódovaná pomocí Base64 (RFC 2045)
`anyURI`	Uniform Resource Identifier Reference podle RFC 2396
`QName`	Qualified Name ve tvaru `ns:Name`
`NOTATION`	Notace. Prehistorický typ – nepoužívá se

Tabulka 6.1: Primitivní jednoduché datové typy

Vestavěné odvozené jednoduché datové typy jsou odvozeny od typů `string` a `integer`. Příkladem odvozených typů může být `token` od typu `string` pro řetězec bez mezer nebo `negativeInteger` od typu `integer` pro záporné celé číslo. Kompletní seznam je samozřejmě uveden ve specifikaci.

XS umožňuje definovat vlastní odvozené jednoduché typy a to buď *restrikcí*, *sjednocením*, nebo *seznamem*.

1. Odvozováním restrikcí, jak se dá očekávat, omezujeme obor hodnot bázového datového typu. Každý vestavěný datový typ má specifikovánu sadu constraintů, které pro něj můžeme použít – např. pro `string` constrainty `maxLength`, `pattern`, pro typ `integer` constrainty `minInclusive`, `totalDigits` atd.

   Uveďme příklad restrikce z datové typu `string` odpovídající všem řetězcům sudé délky, které začínají malým písmenem a kde se pravidelně střídají malá a velká písmena (regulární výraz $(\{a, \ldots, z\}\{A, \ldots, Z\})^+$).

```
<xs:simpleType name="ZigZagString">
  <xs:restriction base="xs:string">
    <xs:pattern value="([a-z][A-Z])+"/>
  </xs:restriction>
</xs:simpleType>
```

2. Odvození sjednocením odpovídá sjednocení oborů hodnot použitých jednoduchých datových typů.
3. Odvození seznamem vytváří datový typ, který představuje seznam hodnot daného datového typu oddělených whitespacem. Například kód

```
<xs:simpleType>
  <xs:list itemType="xs:integer" />
</xs:simpleType>
```

vytváří jednoduchý datový typ odpovídající seznamu celých čísel.

Složené datové typy

Složenému datovému typu je, v terminologii regulárních stromových gramatik, přiřazen pevný neterminál a jeho obsah odpovídá slovu z $(T \cup N)^*$ Složený datový typ má několik základních druhů obsahu a to konkrétně `simpleContent, sequence, all, choice, complexContent`.

Předně ještě poznamenejme, že elementy (terminály), jsou definovány pomocí elementu `element` ze jmenného prostoru jazyka XS. Můžeme definovat prázdné elementy, elementy s obsahem, který je jednoduchý typ, elementy obsahující pouze další podelementy a elementy se smíšeným obsahem (podelementy i text). Typ elementu se definuje buď atributem `type` elementu `element` (např. `<xs:element name="vek" type="xs:integer" />`), nebo podelementy `simpleType` nebo `complexType`.

1. `simpleContent` – odpovídá obsahu, který je jednoduchý datový typ
2. `sequence` – definuje posloupnost podelementů, každému přitom přiřazujeme typ. K dispozici máme rovněž možnost omezit kardinalitu vztahu (počet výskytů) na celočíselný interval $[min, max]$.

```
<xs:complexType name="Jmeno">
  <xs:sequence>
    <xs:element name="krestni" type="xs:string"
                minOccurs="1" maxOccurs="5" />
    <xs:element name="prijmeni" type="xs:string"
                minOccurs="1" maxOccurs="2" />
  </xs:sequence>
</xs:complexType>
```

3. `all` – specifikuje, že následující elementy se mohou vyskytovat jako množina, tedy neuspořádaně a každý nejvýše jednou. Jedinou volbou je `minOccurs` $\in \{0, 1\}$.

4. `choice` – výběrem specifikujeme, že se může vyskytovat pouze jeden z obsažených elementů
5. `complexContent` – objektově orientovaný rys jazyka XS. Umožňuje jednoduchou dědičnost, a to buď metodou `restriction` – omezení jednoduchého bázového typu, nebo `extension` – rozšíření složeného bázového typu.

Vztah k RSG

Bez důkazu uveďme, že jazyk W3C XML Schema je vzhledem k použití *typů* schopen vyjádřit pouze jednotypové regulární stromové gramatiky. Navíc ale přidává primární klíče a cizí klíče. Tyto sémantické vztahy samozřejmě nemohou regulární stromové gramatiky vyjádřit. Obecně si ale myslím, že na úrovni schématu je mnohem elegantnější a efektivnější striktně oddělit specifikování syntaktických a sémantický podmínek pro XML dokumenty a tyto dva typy podmínek mezi sebou nemíchat. Tak se totiž do XML schéma jazyků přidávají *ad-hoc* konstrukce podle aktuální potřeby vyjádřit ten nebo druhý typ omezujících podmínek.

6.1.4 Relax NG

Jazyk Relax NG pro popis XML schémat vznikl pod patronací The Organization for the Advancement of Structured Information Standards (OASIS), aktuální je verze 1.0 z 3. prosince 2001. Autory jazyka Relax NG jsou James Clark a Murata Mukoto, kteří svou práci založili na svých starších jazycích Trex (James Clark) a Relax (Murata Mukoto). Specifikaci je možno nalézt v [6].

Schéma v jazyce Relax NG je gramatikou generující všechny validní XML dokumenty. V jazyce Relax NG je možno vytvořit schéma pro libovolnou regulární stromovou gramatiku. Schémata v jazyce Relax NG vypadají obecně takto:

```
<grammar>
  <start>
    <!-- pocatecni "slova" nad (N u~T)* -->
  </start>
  <define name="A1">
    ...
  </define>
  <define name="A2">
    ...
  </define>
```

```
  ...
</grammar>
```

Schémata se uzavírají do kořenového element `grammar`, prvním synem bývá element `start` definující počáteční slova nad $\{N \cup T\}^*$. Poté již následují jednotlivá přepisovací pravidla. Atribut `name` elementu `define` specifikuje neterminál, jeho obsah pak pravou stranu přepisovacího pravidla.

Příklad 7. Příklad přepisovacího pravidla Jmeno → jmeno(Krestni$^+$, Prijmeni).

Přepisovací pravidlo

```
<define name="Jmeno">
  <element name="jmeno">
    <oneOrMore>
      <element name="krestni">
        <text />
      </element>
    </oneOrMore>
    <element name="prijmeni">
      <text />
    </element>
  </element>
</define>
```

Odkazování na neterminál se provádí pomocí klíčového elementu `ref` takto

```
<ref name="Neterminal" />
```

Příklad 8. Jednoduché rekurzivní schéma tedy můžeme napsat např. takto:

Rekurzivní schéma

```
<grammar
  xmlns="http://relaxng.org/ns/structure/1.0">
  <start>
    <element name="lide">
      <zeroOrMore>
        <ref name="Clovek" />
      </zeroOrMore>
    </element>
  </start>
  <define name="Clovek">
    <element name="clovek">
      <element name="jmeno">
        <text />
      </element>
      <element name="synove">
```

```
        <zeroOrMore>
          <ref name="Clovek" />
        </zeroOrMore>
      </element>
    </element>
  </define>
</grammar>
```

Ze specifikace ještě uveďme pouze tzv. *Simple syntax* gramatiku, která popisuje všechna schémata v Relax NG, na která se provedou určitá simplifikační pravidla.

Gramatika jednoduché syntaxe

```
grammar ::= <grammar>
              <start>top</start>
              define*
            </grammar>
define  ::= <define name="NCName">
              <element>nameClass top</element>
            </define>
top     ::= <notAllowed/> | pattern
pattern        ::= <empty/>| nonEmptyPattern
nonEmptyPattern ::= <text/>
  | <data type="NCName" datatypeLibrary="anyURI">
      param* [exceptPattern]
    </data>
  | <value datatypeLibrary="anyURI" type="NCName" ns="string">
      string
    </value>
  | <list> pattern </list>
  | <attribute> nameClass pattern </attribute>
  | <ref name="NCName"/>
  | <oneOrMore> nonEmptyPattern </oneOrMore>
  | <choice> pattern nonEmptyPattern </choice>
  | <group> nonEmptyPattern nonEmptyPattern </group>
  | <interleave> nonEmptyPattern nonEmptyPattern </interleave>
param         ::= <param name="NCName"> string </param>
exceptPattern ::= <except> pattern </except>
nameClass     ::= <anyName> [exceptNameClass] </anyName>
| <nsName ns="string"> [exceptNameClass] </nsName>
| <name ns="string"> NCName </name>
| <choice> nameClass nameClass </choice>
exceptNameClass ::= <except> nameClass </except>
```

Jak vidíme, gramatika je velice jednoduchá a leccos vypovídá o kvalitním návrhu jazyka Relax NG.

Jednoduchá dědičnost

Jednoduchá dědičnost – dědění všech vlastností po jednom konkrétním předkovi, se dá poměrně jednoduše pomocí jazyka RELAX NG nasimulovat. Jednoduchá dědičnost tedy nepřináší žádné zvýšení vyjadřovací síly XML schema jazyka.

Mějme uzavřený systém n tříd ($n \in \mathbb{N}$) $C_1, C_2, \ldots, C_n$. Každá třída nechť je buď abstraktní, nebo konkrétní, a nechť buď má definovaného jednoho předka, nebo nemá. Pokud má třída $C_i, i \in [n]$ předka, pak jeho index označme p_i (předek je tedy C_{p_i}). Nechť obsah každé třídy odpovídá obsahu elementu **define**, resp. přesněji řečeno modelu obsahu přepisovacího pravidla. Obsah třídy C_i označme M_i. Za každou třídu vytvoříme nejvýše tři přepisovací pravidla následujícím způsobem. Nechť $i \in [n]$ je libovolné pevné. Pak třídu C_i do gramatiky přidáme tato pravidla:

1. *Instancování.* Je-li třída C_i abstraktní, pak přidej pravidlo $C_i \to$ „N/A“, jinak pravidlo $C_i \to (type = C_i)X_i$.

   Element **notAllowed** zakážeme instancování abstraktních tříd. V případě instancování konkrétní třídy ještě navíc doplníme informaci o typu (runtype).

2. *Obsah potomka.* Má-li třída C_i předka, pak přidej pravidlo $X_i \to X_{p_i}M_i$, jinak pravidlo $X_i \to M_i$.

   Obsah potomka je obsah předka rozšířený o M_i.

3. *Polymorfismus.* Má-li třída C_i předka, pak přidej pravidlo $C_{p_i} \to C_i$.

   Zlaté pravidlo dědičnosti říká, že potomek může kdykoli zastoupit předka. Tedy neterminál předka přepisujeme na neterminál potomka. Jinými slovy, kde se vyskytne neterminál předka, tam jej můžeme rozvinout na neterminál potomka.

Příklad 9. Představme si jednoduchý příklad dědičnosti „*Funkce*“ → „*Spojitá funkce*“. Máme abstraktní třídu **Funkce** s jediným atributem **predpis** a potomka **SpojitaFunkce** mimo jiné navíc obsahující atribut **primitivniFunkce**[7].

Syntax je zřejmý z příkladu. Třídy jsou zabaleny do element **class** s možnými atributy

- **name** – jméno třídy (povinný atribut)
- **abstract** – boolean hodnota = *true*, právě když je třída abstraktní (volitelný atribut)
- **base** – jméno předka (volitelný atribut)

[7] Spojitá funkce na $[a, b]$ má primitivní funkci.

Objektové schéma

```
<class name="AbstraktniFunkce" abstract="true">
  <element name="predpis">
    <text />
  </element>
</class>

<class name="SpojitaFunkce" base="Funkce">
  <element name="a"><data type="integer" /></element>
  <element name="b"><data type="integer" /></element>
  <element name="primitivniFunce">
    <text />
  </element>
</class>
```

Odpovídající sada přepisovacích pravidel vypadá v Relax NG takto:

Neobjektové schéma

```
<!-- Pravidlo 1 -->
<define name="Funkce">
  <notAllowed />
</define>

<!-- Pravidlo 2 -->
<define name="Funkce-X">
  <element name="predpis">
    <text />
  </element>
</define>

<!-- Pravidlo 1 -->
<define name="SpojitaFunkce">
  <attribute>
    <name ns="http://relaxng.org/ObjectOriented-instance">
      type
    </name>
    <value>SpojitaFunkce</value>
  </attribute>
  <ref name="SpojitaFunkce-X" />
</define>

<!-- Pravidlo 2 -->
<define name="SpojitaFunkce-X">
  <ref name="Funkce-X" />
```

```
  <element name="a"><data type="integer /></element>
  <element name="b"><data type="integer /></element>
  <element name="primitivniFunce">
    <text />
  </element>
</define>

<!-- Pravidlo 3 -->
<define name="Funkce" combine="true">
  <ref name="SpojitaFunkce" />
</define>
```

Doplňme, že hodnotou atributu **combine="true"** posledního přepisovacího pravidla víceméně říkáme, že pravidla s neterminálem **Funkce** vlevo jsou různá. Druhá možná hodnota atributu **combine** je **interleave**.

Jak jsme ukázali, vytvořit neobjektový ekvivalent k „objektovému“ schématu je velice jednoduché.

Implementace

Implementací validátorů pro schémata v jazyce RELAX NG je již poměrně dost. Rozšířeností ale nedosahují validátorů pro W3C XML Schema nebo DTD, které jsou přeci jen v praxi používanější. Za všechny validátory jmenujme Sun Multi-Schema XML Validator autora Kohsuke Kawaguchiho. Tento validátor, implementovaný v Javě, jak název napovídá, validuje podle schémat v několika jazycích. Jmenujme RELAX NG, W3C XML Schema, DTD, v dnešní době existuje i vynikající Schematronový add-on Relames.

Pro zajímavost ještě uveďme, že validátor MSV jako vnitřní reprezentaci všech validovaných jazyků používá právě reprezentaci v RELAX NG. Tedy všechna schémata jsou nejprve transformována na schéma v RELAX NG a až poté je provedena validace.

6.1.5 Porovnání

1. **Expresivita** – Jednotypové gramatiky ve W3C XML Schema jsou ostře slabší než regulární stromové gramatiky v RELAX NG. Proto existuje spousta **oprávněných** tříd dokumentů, pro které *nelze* v W3C XML Schema vytvořit schéma.

2. **Atribut vs. elementy** – W3C XML Schema se chová nekonzistentně vůči atributům a elementům (viz dále). Naopak RELAX NG klade elementy a atributy ve svých konstrukcích víceméně na rovnost.

3. **Objektové rysy** – tolik vyzdvihované objektové rysy W3C XML Schema naprosto *žádným* způsobem nezvyšují vyjadřovací sílu schémat. Ke schématu používajícímu dědičnost umožněnou jazykem W3C XML Schema pomocí klíčových elementů `complexContent` vždy existuje ekvivalentní „neobjektové schéma“ v jazyce RELAX NG. Jak bylo ukázáno, dědičnost lze do RELAX NG přidat triviálním způsobem.

   Jako příklad nekonzistence uveďme, že v případě odvozování restrikcí nejsou u odvozovaného typu předpokládány žádné elementy z bázového typu, ale naopak všechny atributy implicitně přítomny jsou.

4. **Slabé choice** – naprosto nedostatečná je specifikace výběrů pomocí `choice`. W3C XML Schema neumožňuje přímo říct, že element `X` obsahuje buď podelementy `P` a `Q`, anebo atribut `a` a element `R`.

5. **Neuspořádané XML dokumenty** – pokud netrváme na pořadí synů v XML stromu, stává se W3C XML Schema opět nepoužitelným. Klíčový element `all` specifikuje pouze množinu. Oproti tomu klíčový element `interleave` jazyka RELAX NG umožňuje definovat libovolně prokládaný obsah.

6. **Datové typy** – W3C XML Schema nabízí pouze omezenou množinu datových typů, i když zřejmě dostatečně velikou. Obecně se dá ale říci, že datové typy definované normami XML schéma jazyků *nejsou* pro ně žádným přínosem. To, že daný jazyk definuje, že obsahuje ty a ty vestavěné datové typy, pouze říká, co musí obsahovat všechny implementace. RELAX NG naopak definuje pouze dva základní datové typy a všechny zbývající nechává na konkrétních implementacích.

7. **Validování** – použití vazby dokumentů na konkrétní schémata je velice nevhodné. Vybírání konkrétního schématu pomocí hodnoty atributu `schemaLocation` ze jmenného prostoru

   ```
   http://www.w3.org/2001/XMLSchema-instance
   ```

   v kořenovém elementu XML dokumentu pomocí

   ```
   xsi:schemaLocation="http://www.badserver.com/src
                       http://www.badserver.com/src.xsd"
   ```

   odporuje základnímu principu validace XML dokumentů jakožto nezávislého procesu se dvěma nezávislými vstupy: XML schématem a XML dokumentem. Aplikace přece může chtít konkrétní XML dokument validovat podle těch schémat, podle kterých to bude aktuálně potřeba. XML dokument jako takový nemá žádné právo na to, aby do tohoto procesu zasahoval. Dalším možným nedostatkem je v případě špatných

implementací validátorů možná *security hole* pramenící z odkazování na obecné URL.

8. **Implicitní hodnoty** – W3C XML Schema umožňuje definici implicitních hodnot atributů, pokud odpovídající atribut v XML dokumentu žádnou hodnotu nemá. To je opět *side-efect* naprosto odporující základním principům validace, neboť pokud dokument obsahuje tyto atributy bez hodnot, dostáváme jiné dokumenty v případě, že validace proběhla, nebo nikoliv.

9. **Namespaces** – W3C XML Schema velice neintuitivně zachází s prostory jmen. Za všechny uveďme následující příklad. Mějme schéma definující, že obsahem elementu `x` může být pouze element `y`, který má jednoduchý obsah – řetězec.

Schéma
```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           targetNamespace="http://uri.org">
  <xs:element name="x">
    <xs:complexType>
      <xs:sequence>
       <xs:element name="y" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

A nyní uvažujme tyto dva XML dokumenty:

Invalidní dokument
```
<x xmlns="http://uri.org">
  <y>abc</y>
</x>
```

Validní validní
```
<ns:x xmlns:ns="http://uri.org">
  <y>abc</y>
</ns:x>
```

W3C XML Schema přitom první dokument považuje za nevalidní a druhý za validní. Oproti tomu má Relax NG naprosto přirozený systém dědění jmenného prostoru pro podelementy.

10. **Syntax** – jednotypovost odpovídajích regulárních stromových gramatik, resp. princip přiřazování typů, vede k velice komplikované syntaxi jazyka W3C XML Schema, která ale bohužel není vykoupena vyjadřovací silou.

    Při návrhu jazyka bývá většinou obvyklá snaha minimalizovat množinu klíčových slov jazyka. Chceme, aby existence každého klíčového slova byla maximálně nutná, jinými slovy, aby každé klíčové slovo přinášelo konstrukty, které by bez něj nešly vytvořit.[8] Jazyk W3C XML Schema je v tomto ohledu velice nehospodárný a mnoho klíčových slov by šlo zcela odstranit a nahradit jejich funkci lepší syntaxí jazyka.

    Perličkou může býti také klíčový element `xs:all` z prostoru jmen jazyka W3C XML Schema. Asi málokoho, kdo by viděl schéma v jazyce W3C XML Schema poprvé, by napadlo, že se jedná o zápis množiny.

    Naopak Relax NG obsahuje velice omezenou množinu klíčových elementů, ve které má každý element svoji specifickou nezastupitelnou funkci. Ke zmíněnému `xs:all` nabízí Relax NG o řád silnější alternativu v podobě elementu `interleave`, který specifikuje libovolné pořadí obsahu (prokládání elementů).

11. **Specifikace** – tento bod je okrajový, nicméně i specifikace jazyka W3C XML Schema je, dle mého subjektivního názoru, jedna z nejhorších specifikací, jaké jsem měl možnost vidět. Naproti tomu jazyk Relax NG má poměrně stručnou, ale maximálně výstižnou specifikaci doplněnou o předhledný zápis gramatiky jazyka Relax NG ať už pro tzv. *Full syntax* nebo *Simple syntax*.

6.1.6 Shrnutí

Z výše uvedeného zcela **rigorózně** bez jakýchkoli pochybností plyne, že jazyk Relax NG je ostře lepší než W3C XML Schema, a to jak co do výrazové síly (důkaz v předchozí kapitole), tak i do snadnosti používání. W3C XML Schema nabízí některé prvky, jež nemohou regulární stromové gramatiky postihnout (např. klíče a cizí klíče). Toto je ale možno odstranit dodáním doplňkového jazyka pro specifikaci omezujících podmínek. Jednoduchým dobře známým zástupcem je například Schematron, jehož symbióza s jazykem Relax NG je vynikající.

Jazyk W3C XML Schema má dle mého názoru tolik objektivních nedostatků, pro které by se v žádném případě neměl stát jazykem číslo 1 pro zápis XML schémat, každopádně alespoň ne na úrovni vědeckých prací. Například pouze jednotypovost odpovídající stromové gramatiky je problém, který již nelze opravit. Zjednodušeně bychom mohli na adresu W3C XML Schema říci asi toto:

[8] Přesnější formalizace těchto slov přesahuje rozsah této práce.

„Proč používat pro zápis regulárních stromových gramatik jazyk, který je syntakticky těžkopádný a ještě je nedokáže zapsat všechny?“

Jazyk W3C XML Schema nenabízí kromě většího rozšíření nic, co by nemohl nabídnout jazyk RELAX NG. Kromě toho, že je výrazově slabší, je také mnohem těžkopádnější. Používání W3C XML Schema pro jeho „objektové“ rysy, je rovněž ve své podstatě naprosto neodůvodněné. U RELAX NG je zřejmý matematický základ, který jej dělá velice konzistentním a elegantním jak při specifikaci, tak při používání. Rozšířenost W3C XML Schema je z velké části zapříčiněna pouze autoritou, které se standardy konsorcia W3C u velké části informatiků těší, a komerční silou, která za W3C stojí a která prosazuje jazyk W3C XML Schema ve svých komerčních produktech. Vytváří tak naprosto mylnou iluzi, že W3C XML Schema je jediným používaným jazykem pro zápis XML schémat. RELAX NG jakožto XML podoba regulárních stromových gramatik přitom nabízí o mnoho více.

Z této kapitoly pro nás tedy plyne rozhodnutí, že jediným dále uvažovaným jazykem pro zápis potřebných XML schémat bude pouze RELAX NG, který nám poskytne maximální komfort ryzích regulárních stromových gramatik. Jazyk RELAX NG tedy bude od této chvíle víceméně pouze formalismem regulárních stromových gramatik. Víme totiž, že jakákoli regulární stromová gramatika lze pomocí jazyka RELAX NG zapsat.

Kapitola 7

Implementace

Součástí této diplomové práce je rovněž pilotní implementace několika modulů založených na představeném AC-modelu. Naimplementovány jsou tyto součásti:

1. Aplikace pro tvorbu AC-modelu
2. Překlad A-modelu na gramatiku a dále na Relax NG schéma
3. Validátor XML dokumentů podle daného C-modelu (`cmv`)

Moduly jsou seřazeny podle rostoucí zajímavosti. Veškerý software dodaný s touto prací je určen pro .NET Framework 2.0, zdrojové kódy jsou v jazyce C#. Volba jazyka C# a prostředí .NETu není nijak zásadní, převedení například do C, C++ nebo Javy je jen rutinní záležitostí.

7.1 Identifikátory

Každý archetyp má přiřazen unikátní identifikátor τ, který při přepisu na gramatiku plní funkci terminálu. Překladová gramatika $\mathcal{G}(\mathcal{A})$ přidává kořenu každé instance I archetypu A α-podstrom obsahující identifikátor τ_A. Díky tomu tedy všechny instance archetypů nesou s sebou informaci o archetypu, jehož jsou instancí. Dále kdekoli se v A-modelu vyskytne vrchol, který je referencí, pak tato reference se přepíše na β-podstrom, který obsahuje identifikátor archetypu a identifikátor instance.

Vše navíc udělejme tak, aby všechny výše zmíněné identifikátory, tedy identifikátory archetypů a identifikátory instancí, byly globálně unikátní. Budou-li totiž všechny identifikátory globálně nezopakovatelné, máme naprosto triviálně vyřešeno globální sdílení archetypů a výměnu instancí.

Nyní se dostáváme k implementaci identifikátoru. Naprosto přirozeně použijeme tzv. univerzálně unikátní identifikátory všeobecně známé spíše pod

zkratkou (UUID). Tyto identifikátory jsou generovány algoritmicky a možnost jejich zopakování v čase je mizivá. Podejme stručnou specifikaci UUID.

Universally Unique Identifier (UUID)

Universally Unique Identifier (někdy také Globally Unique Identifier) je 128-bitové číslo. Účelem vzniku UUID je přinést jednoznačný identifikátor globálně v čase nezopakovatelný. Existují 4 základní verze UUID očíslované čísly 1 až 4.

Typ UUID	*Význam*
1	Time–based
2	DCE security (Leach-Salz)
3	Name–based
4	Pseudorandom

Původní nyní verze 1 byla víceméně jen zřetězením aktuálního času (počet desetin mikrosekundy od přijetí Gregoriánského kalendáře) a MAC adresy síťového rozhraní. Tento typ UUID je ale příliš jednoduchý a nyní se již nepoužívá. Uvedení MAC adresy v UUID je navíc zbytečným zásahem do soukromí.[1] V současné době se používá hlavně typ 4 – pseudonáhodné 128-bitové číslo. UUID se zapisují v této grafické podobě:

```
bfd5fb03-80d5-[4]fc6-b65e-683f07ee3eb4
51f95cf6-392b-[4]041-88cb-ad9f230e9878
5289df73-7df5-[3]326-bcdd-22597afb1fac
```

Čísla v rámečku označují právě typ UUID. Specifikace algoritmů generujících všechny zmíněné typy je dobře popsána např. v RFC 4122. Existuje ještě speciální *Nil–UUID* obsahující samé nuly.

Implementaci UUID nalezneme jak v Javě (třída `java.util.UUID`), tak i v .NETu (třída `System.Guid`), běžně dostupné jsou i samostatné implementace UUID všech typů.

V generátoru regulární stromové gramatiky dodaném s touto prací je použita implementace GUID.

7.2 AC Modeler

AC Modeler je jednoduché GUI pro tvorbu AC modelů. Přímo z GUI je k dispozici možnost exportu A-modelu na regulární stromovou gramatiku a dále ještě na Relax NG schéma. AC Modeler umožňuje i přímo spouštět `cmv` na daný XML dokument. Po kliknutí na položku menu `About` se zobrazí stručná nápověda pro práci s aplikací AC Modeler.

[1] Poznamenejme, že ale právě obsah MAC adresy v UUID pomohl v několika případech i k dopadení autorů různých virů nebo útoků na informační systémy.

Selektory

Zápis selektorů plně odpovídá definovaným konvencím. Selektor je, jak bylo definováno, formule výrokové logiky s proměnnými končícími na číselnou hodnotu (např. `syn1`, `pocet3`). Právě tato hodnota specifikuje pořadí odpovídajícího syna. V aktuální implementaci je možno pro zápis formulí použít tyto operátory:

`&` (and), `|` (or), `^` (xor), `->` (implikace), `!` (not)

Pro jednoduchost použití jsou předdefinovány selektory

Selektor	Formule
`all`	$\bigwedge_{i=1}^{n} e_i$
`one+`	$\bigvee_{i=1}^{n} e_i$
`exactlyone`	$\bigvee_{i=1}^{n}(e_i \& \neg \bigvee_{k\in[n]\setminus\{i\}} e_k)$

V zápise formulí je možno libovolně korektně používat kulaté závorky `(`, `)`.

Příklady selektorů

```
x1 -> (a2 | c3)
jmeno1 & prijmeni2 & (CisloPasu4 ^ CisloOP3)
1
all
```

7.3 Validátor C-modelu (`cmv`)

Program `cmv` je nejzajímavější část implementace. Jedná se o konzolovou aplikaci, jejímž vstupem je C-model a seznam XML dokumentů. Pro každý daný XML dokument se pak zjistí, které z daných omezujících podmínek v něm platí a které nikoli. Aplikace se spouští s těmito parametry:

```
cmv.exe <cmodel.txt> <doc1.xml> <doc2.xml> ...
```

Na standardní výstup je poté vypsán XML dokument obsahující souhrnný report o validacích všech uvedených dokumentů proti danému C-modelu.

Program `cmv` je možno v prostředí .NETu spouštět také pomocí reflexí. Assembly `cmv.exe` obsahuje ve jmenném prostoru `cmv` třídu `Program` se statickou metodou

```
public static string
CreateReport(ArrayList docs, Stream cmodel),
```

která je výkonným jádrem `cmv`. Parametr `docs` je `ArrayList` obsahující instance třídy `XmlDocument`, `cmodel` je pak stream se zápisem omezujících podmínek. Návratovou hodnotou je zápis reportu ve formě řetězce. Mechanismus spuštění metody `CreateReport` pomocí reflexe je pak tento sled kroků:

```
...
Assembly cmvAsm         = Assembly.LoadFile(<cmvpath>);
Type cmvClass           = cmvAsm.GetType("cmv.Program");
MethodInfo CreateReport = cmvClass.GetMethod("CreateReport");
try {
  string result = (string)CreateReport.Invoke(
    null, new object[] { <docs>, <cmodelStream> }
  );
  ...
}
```

Takto spouští program `cmv` aplikace AC Modeler.

Struktura souboru `cmodel.txt`

Libovolný textový soubor. Na každém řádku je uvedena jedna omezující podmínka zapsaná pomocí dále uvedené syntaxe. Řádky začínající dvěma lomítky `//` se interpretují jako komentáře.

Verze

Verze programu `cmv` dodaného s diplomovou prací je označena jako `0.11`.

7.4 Syntax omezujících podmínek

Každá omezující podmínka je, jak víme, formule logiky prvního řádu. Pro účely použití v programu `cmv` pro jednoduchost navíc požadujeme, aby každá formule byla v prenexním tvaru. Jak víme, ke každé formuli existuje formule ekvivalentní v prenexním tvaru. Tento požadavek tedy není žádným omezením.

Kvantifikátory

- Všeobecný kvantifikátor ∀: zapisujeme jako `forall`
- Existenční kvantifikátor ∃: zapisujeme jako `exists`

Kvantifikátor má podobu

`forall` *proměnná* `in` [*regulární cesta*]_*kontext*
nebo
`exists` *proměnná* `in` [*regulární cesta*]_*kontext*

Jednotlivé kvantifikátory se oddělují mezi sebou čárkou. Můžeme tedy napsat například

```
exists x in [/*]_I, forall y in [/x]_x, exists pom in [/z/.]_I
```
atd.

Po uvedení všech kvantifikovaných proměnných s doménami následuje znak dvojtečka `:` a část bez kvantifikátorů. Pokud formule neobsahuje kvantifikátor dvojtečka se na počátku nepíše.

Cesty

Hodnotou výrazu `[/x]_v` je množina synů vrcholu `v`, kteří se jmenují `x`. Cesta v rámci daného kontextu tedy začíná v synovských vrcholech.

Část bez kvantifikátorů

Část bez kvantifikátorů je libovolná formule výrokové logiky.

Proměnné

Formule může obsahovat jen kvantifikované proměnné a volnou proměnnou `I` představující validovaný XML dokument (resp. instanci archetypu).

Konstanty

Pilotní implementace programu `CModelValidator` obsahuje implementaci datových typů celé číslo a řetězec. Celá čísla se zapisují přímo, řetězce, jak je obvyklé, mezi uvozovky `"..."`.

Logické operátory

Pro formulí výrokové logiky máme k dispozici všechny běžně používané operátory s běžnou aritou – viz tabulka 7.1. Operátory jsou v tabulce 7.1 seřezeny podle priority od nejvyšší po nejnižší. Zápis (kromě implikace) odpovídá jazyku C.[2]

Aritmetické operátory

Implementovány jsou všechny běžné aritmetické operátory, konkrétně:

```
+, -, *, /, %, **
```

[2] C nativně operátor implikace neobsahuje.

Operátor	Význam	Zápis
¬	not	!
&	and	&
∨	or	|
→	implikace	->

Tabulka 7.1: Zápis logických operátorů

Poznamenejme, že operátor `%` představuje modulo a operátor `**` mocninu. Aritmetické operátory jsou rozděleny do tří tříd podle priority:

1. `**`
2. `*, /, %`
3. `+, -`

Relační operátory

K disposizi jsou rovněž všechny běžně používané binární relační operátory, zápis kromě `=` jako v případě logických operátorů odpovídá jazyku C.

```
=, !=, >, <, >=, <=
```

V současné implementaci jsou všechny tyto relační operátory definované jak nad dvojicemi celých čísel, tak nad dvojicemi řetězců.

Pomocné symboly

Z pomocných symbolů jsou pro zápis omezujících podmínek k dispozici kulaté závorky `(, )` s očekávanou funkcí.

Funkce

Implementovány jsou funkce pro výběr podstromů podle regulární cesty (viz kapitola o omezujících podmínkách), pomocné funkce a funkce pro interpretaci datových typů (ω-funkce) a funkce simulující *try–catch* blok. Základní implementované funkce shrnuje tabulka 7.2.

Funkce pro simulaci *try–catch* bloků fungují následovně. Představme si, že chceme zapsat, že všechny elementy se jménem `x`, které mají navíc číselnou hodnotu, musí mít tuto číselnou hodnotu větší než 10. Napíšeme tedy

```
forall x in [/*/x]_I: hasvalue(x) -> integer(x) > 10
```

Funkce	Obor hodnot	Příklad použití
`nodename`	Řetězec	`nodename(x) = "Operator"`
`empty`	Logická hodnota	`empty([/x]_I)`
`hasvalue`	Logická hodnota	`hasvalue(x)`
`hasintegervalue`	Logická hodnota	`hasintegervalue(x)`
`integer`	Číslo	`integer(y)*integer(z) > 10`
`string`	Řetězec	`string(jmeno) = "Jakub"`
`count`	Číslo	`count([/*/jmeno]_C) <= 3`

Tabulka 7.2: Základní funkce

Takto zapsaná formule ale na dokumentech s elementy `x` bez hodnoty fungovat nebude, neboť na těchto vrcholech není definována funkce `integer` a nespecifikovali jsme žádné zkrácené vyhodnocování podmínek. Tento problém ale právě řeší unární funkce `true` resp. `false`, které vracejí logické hodnoty *true* resp. *false* v případě, že argument nelze vyhodnotit, nebo samotný argument v případě, že vyhodnotit lze. Výše zmíněnou formuli tedy přepíšeme do tvaru

```
forall x in [/*/x]_I: true(integer(x) > 10)
```

a tím docílíme požadovaného efektu. Pro lepší pochopení uveďme ještě příklady několika omezujících podmínek.

Příklady omezujících podmínek

Všechny elementy pojmenované jednicka musí mít hodnotu 1
```
forall j in [/*/jednicka]_I: integer(j) = 1
```

Všechny číselné hodnoty všech elementů jsou po dvou různé
```
forall x in [/*]_I, forall y in [/*]_I : integer(x) = integer(y)
-> x=y
```

Existuje právě jeden vrchol s nejvyšší číselnou hodnotou
```
exists x in [/*]_I, forall y in [/*]_I : hasintegervalue(x) &
(x=y | true(integer(x) > integer(y)))
```

Hodnoty všech synů jsou větší nebo rovny než hodnota otce (neklesající posloupnosti)
```
forall otec in [/*]_I, forall syn in [/*]_otec :
true(integer(otec) <= integer(syn))
```

Textová hodnota všech elementů je, pokud existuje, stejná jako jejich název
```
forall e in [/*]_I : true(nodename(e) = string(e))
```

Dokument obsahuje element skola
```
!empty([/*/skola]_I)
```

Má-li element cislo hodnotu, pak tato hodnota musí být sudé celé číslo
```
forall cislo in [/*/cislo]_I : true(integer(cislo) % 2 = 0)
```

V kontextu člověka je počet křestních jmen mezi 1 *a* 3 *včetně*
```
forall C in [/clovek]_I: count([/krestni]_C) >=1 &
count([/krestni]_C) <= 3
```

Report

Report je XML dokument obsahující jednak zápis všech formulí použitého C-modelu a jednak pro každý dokument a formuli jednu z hodnot `True`, `False`, nebo `Fail`. V případě varianty `Fail` je ještě doplněna informace o chybě, která při vyhodnocování dané omezující podmínky nastala. Takto může například konkrétní report vypadat:

Příklad reportu

```
<?xml version="1.0" encoding="utf-8"?>
<cmvreport>
  <formulae>
    <formula id="1">count([/*/x]\_I) = 3</formula>
    <formula id="2">2 * 2**4 % 5 = 2</formula>
    <formula id="3">
      false(integer([/*/x]\_I) &gt; 500)
    </formula>
    <formula id="4">
      forall x in [/*]\_I, forall y in [/.]\_x :
        true(integer(x) &lt; integer(y))
    </formula>
    <formula id="5">
      exists x in [/*]\_I, forall y in [/*]\_I :
        false(integer(x) = integer(y)) -&gt; x=y
    </formula>
    <formula id="6">
      forall x in [/*]\_I:
        true(nodename(x) = "z" -&gt; integer(x) = 1)
    </formula>
  </formulae>
  <report>
    <document>
      <validity>
        <formula id="1">False</formula>
        <formula id="2">True</formula>
        <formula id="3">False</formula>
        <formula id="4">True</formula>
        <formula id="5">True</formula>
        <formula id="6">True</formula>
      </validity>
    </document>
  </report>
</cmvreport>
```

Gramatiky jazyků pro zápis selektorů i omezujících podmínek jsou na přiloženém CD. Veškeré gramatiky používané v rámci implementace jsou v EBNF (Extended Backus-Naur form).

7.5 Poznámka k zadání diplomové práce

V zadání je uvedeno „implementace systému pro zadávání dat“. Jen bych chtěl touto cestou upozornit, že otázka vstupu dat je výhradně otázkou validace. Otázka volby úložiště pro XML dokumenty, jakož i algoritmus převodu

$$\text{RSG (Relax NG)} + \text{C-model} \rightarrow \text{databázové schéma}$$

nebyl předmětem této práce. Samozřejmě v souvislosti s vývojem polymorfního systému přijde na řadu i převod na databázové schéma. Poznamenejme jen, že problematika převodu XML schémat např. na relační databázová schémata je poměrně prozkoumaná, leč bohužel ve spoustě případů je převod uvažován *pouze* ze schémat v jazyce W3C XML Schema a vůbec se neuvažuje v kontextu plnohodnotných regulárních stromových gramatik, což je zřejmý nedostatek.

Kapitola 8

Stav ve světě

Veškeré pokusy o navržení konceptuálního jazyka pro XML je možno rozdělit do dvou hlavních skupin, na *rozšíření E-R schémat* a na *stromové (lesní) konceptuální jazyky pro XML*. Osobně bych ještě vyčlenil zvláštní skupinu konceptuálních modelů tvořících nádstavbu jazyka W3C XML Schema autorů lpějících na volbě tohoto jazyka na logické úrovni. V této kapitole si stručně uveďme jen nejznámější zástupce výše zmíněných kategorií. Spousta konceptuálních modelů nepřináší kromě zápisu nebo grafického ztvárnění nic nového, a proto jejich zkoumání by bylo zbytečným zvýšováním rozsahu práce. Některé známější konceptuální modely uvedeme v odkazech na konci kapitoly.

8.1 Rozšíření ER schémat

Zástupci první skupiny modelovacích jazyků se nevzdávají v relačním světě výborně osvědčených E-R schémat, která se snaží rozšířit tak, aby byla použitelná i pro modelování XML dat. Jak již bylo řečeno, primárním úkolem, pokud chceme záplatovat E-R schémata, je vyřešit uspořádání relací, exkluzivní vztahy („buď – anebo“), resp. hierarchičnost.

8.1.1 EReX

Autorem modelu je Murali Mani a základní specifikaci je možno nalézti v [7]. Podejme stručný výklad modelu EReX.

EReX je čistým rozšířením ER modelu o tzv. *kategorie* a omezující podmínky na *pokrytí* a *pořadí*.

Kategorie

Kategorie je binární vztah mezi dvěma entitami vzdáleně připomínající ISA vztah. Formálně entity $\{E_i\}_{i=1}^{n}, n \in \mathbb{N}$ jsou kategorie entity E, pokud pro

každou instanci modelu platí $\forall i \in [n] : I(E_i) \subset I(E)$.

Omezující podmínky na pokrytí

Omezující podmínky se definují na entitách. Specifikují dva základní vztahy: *úplné pokrytí* a *exkluzivní pokrytí*.

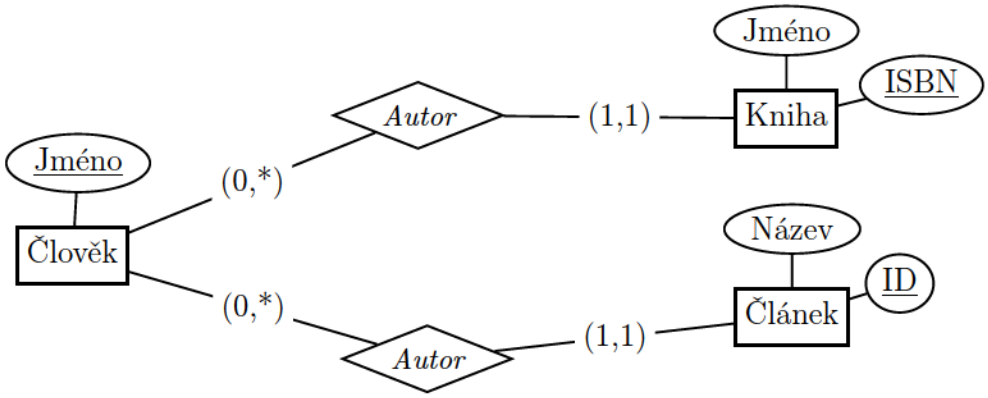
Entity $\{E_i\}_{i=1}^n, n \in \mathbb{N}$ jsou *úplně pokryty* entitou E, pokud pro každou instanci modelu platí $\cup_{i=1}^n I(E_i) = I(E)$. Zapisujeme to pak jako $\cup_{i=1}^n E_i = E$. Analogicky pro role v relacích.

Entity E_1, E_2 jsou *exkluzivně pokryty* entitou E a zapíšeme to jako $E_1 \cap E_2 = \emptyset$, pokud pro každou databázovou instanci tohoto modelu platí, že $I(E_1) \cap I(E_2) = \emptyset$. Analogicky pro role v relacích.

Omezující podmínky na pořadí

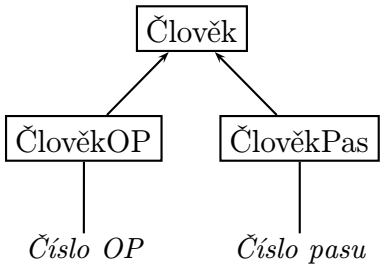
Jelikož v XML dokumentu podle definice záleží na pořadí podelementů daného elementu, řeší EReX částečně i problém pořadí prvků relace. Autorova definice je formálně poněkud nepřesná. Snažil se ale nejspíše říci, že ke každé entitě E označené jako *uspořádané*, která se podílí na relaci R, je přidán číselný atribut *pořadí*. Na prvcích relace je pak tímto atributem indukováno lineární uspořádání.

Příklad EReX schématu



$$\textit{Člověk-Kniha} \cap \textit{Člověk-Článek} = \emptyset$$
$$\textit{Člověk-Kniha} \cup \textit{Člověk-Článek} = \textit{Člověk}$$

Obrázek 8.1: Exkluzivní pokrytí rolí – „buď článek, nebo kniha“



$$\check{C}lov\check{e}kOP \cap \check{C}lov\check{e}kPas = \emptyset$$
$$\check{C}lov\check{e}kOP \cup \check{C}lov\check{e}kPas = \check{C}lovek$$

Obrázek 8.2: Exkluzivní pokrytí entit

Shrnutí

EReX přidává k ER schématům princip hierarchie, exkluzivního vztahu právě dvou entit a pořadí prvků relace. Složitější logické vztahy, např. $(A \rightarrow B) \rightarrow C$ neumí. V [7] je dále uveden překlad konkrétního EReX schématu do gramatiky XGrammar, která oproti regulárním stromovým gramatikám už rozlišuje mezi elementy a atributy a zavádí datové typy. XGrammar navíc přidává omezující podmínky na ID, IDREF, klíč a cizí klíč, tedy prvky známé z relačních databází. Bohužel se tak děje ale poněkud neelegantně. Klíče jsou totiž specifikovány pouze pro typy a ne pro cesty (cesty viz W3C XML Schema). Je tudíž nevyhnutelné, aby existovalo jednoznačné typové přiřazení, neboli, aby odpovídající gramatika byla deterministická. Výstupem algoritmu pro překlad schémat EReX do XGrammar jsou vždy pouze jednotypové regulární stromové gramatiky. To je ale zbytečné omezení. Žádné jiné omezující podmínky EReX specifikovat neumí.

8.2 Stromové modely

8.2.1 Conceptual XML (C-XML)

Tvůrci jazyka Conceptual XML (C-XML) si dali za úkol vytvořit vysoce abstraktní konceptuální model pro modelování obrovských heterogenních databázových systémů výrazově ekvivalentní s XML schéma jazykem W3C XML Schema. V preambuli svých prací tvrdí, že jejich C-XML je nejlepším hierarchickým konceptuálním modelem. Podívejme se na něj tedy zblízka.

Specifikaci C-XML je možno nalézti v [8], kde se snaží ukázat, že jejich

C-XML umí namodelovat „vše reálné“. C-XML leží na pomezí mezi modely založenými na ER a stromovými modely. Přesná taxonomie ale není až tak podstatná.

Každý konkrétní model v C-XML sestává z množiny objektů, množiny relací a množiny omezujících podmínek. Model je hypergraf jehož vrcholy jsou množiny objektů a hrany podmnožiny množiny objektů. Jsou postihnuty základní prvky jako hierarchické vztahy, kardinality, rekurze apod. Autoři víceméně jen na jednom konkrétním příkladu ukazují překlad na W3C XML Schéma. Zajímavým počinem jsou XQuery pohledy významově analogické pohledům v relačních databázích.

Formální úroveň specifikace C-XML je v citovaném článku např. oproti Maniho EReXu velice nízká. Autoři se snaží dokázat ekvivalenci s W3C XML Schema, ale bohužel chybně. Uvádějí definici:

> *For any schema specification S_A of type A (e.g. $S_{C\text{-}XML}$ or $S_{XMLSchema}$ in our discussion here) there is a corresponding valid interpretation I_{SA} (i.e. a valid, populated model instance for a C-XML model instance or a conforming XML document for an XML Schema instance).*
>
> **Definition 1.** *A translation T from schema specification S_A to a schema specification S_B preserves information if there exists a procedure P that for any valid interpretation I_{SA} corresponding to S_A computes I_{SA} from I_{SB} where I_{SB} is the interpretation corresponding to S_B induced by T.*
>
> **Theorem 1.** *Let T be the translation described in Section 3.1 that translates a C-XML model instance SC-XML to an XML Schema instance SXMLSchema. T preserves information and constraints.*
>
> **Theorem 2.** *Let T be the translation described in Section 3.2 that translates an XML Schema instance $S_{XMLSchema}$ to a C-XML model instance $S_{C\text{-}XML}$.* T preserves information and constraints.

Ukažme nesmyslnost výše uvedené definice 1.

1. Každá specifikace schématu je popsána konečně mnoha znaky z konečné množiny. Tedy všech možných specifikací je nejvýše spočetně. Existuje tedy dle Cantor-Bernsteinovy věty prosté surjektivní zobrazení

$$c : \ \textit{specifikace} \to \mathbb{N},$$

2. Nechť T je takové zobrazení, že konkrétní specifikaci S přiřazuje specifikaci s o 1 větším pořadím, tedy

$$T(S) = c^{-1}(c(S) + 1).$$

3. Nechť S_A je libovolně pevně zvolené schéma a $S_B = T(S_A)$. Množinu instancí schémat S_A označíme I_A, množinu instancí schémat S_B označíme

I_B. Množiny I_A a I_B jsou nejvýše spočetné. Nechť bylo schéma S_A BÚNO zvoleno tak, že I_B je právě spočetná. Existuje tedy prostá funkce

$$T' : I_A \rightarrow I_B.$$

Zkonstruujeme nyní funkci P z I_B do I_A tak, že

$$P \equiv T'^{-1}.$$

4. Existence výše definované funkce P je postačující pro to, abychom mohli podle *Definition 1* říci, že T *zachovává informaci*. Vzhledem k tomu, že bijekce c byla ale zvolena libovolně, je výše uvedená definice absurdní. □

Absurdní je tedy i výše uvedený *Theorem 1* a *Theorem 2*. C-XML tedy rozhodně není „foremost conceptual model", jak tvrdí autoři. Zabývat se dále modelem, jenž není formálně specifikován, nemá význam.

8.2.2 Další konceptuální modely

Pro úplnost uveďme, že existuje mnoho a mnoho dalších konceptuálních modelů, které se ale nijak zásadně neliší resp. nedá se říci, že jeden by byl ostře lepší než jiný. Zajímavé však jsou jen hierarchické modely. Záplatování ER schémat vede vždy k matematicky neelegantním modelům. Za všechny známé hierarchické modely jmenujme například Sémantické sítě specifikované v [9] nebo ORA-SS specifikovaný v [10].

8.3 Srovnání s A-modelem

Srovnání A-modelu proveďme následující tabulkou podle vybraných nejdůležitějších bodů z požadavků na XML konceptuální gramatiku.

Model	*Formální*	*Hierachie*	*Constrainty*	*Překlad*
A-model	ANO	ANO	Logika 1. řádu	RSG
EReX	ANO (ER)	ANO (kategorie)	Jen klíče	JRSG
C-XML	NE	ANO	Jen klíče	JRSG

Tabulka 8.1: Srovnání konceptuálních modelů

Kapitola 9

Závěr

V prezentované diplomové práci jsme vyšli ze snahy vytvořit polymorfní informační systém. K vytvoření takového systému je potřeba kvalitní konceptuální model, na jehož základě bude systém upravovat svojí perzistenční a prezentační vrstvu. Lékařská data, stejně jako většina dat v jiných oblastech, mají hierarchickou strukturu, takže je výhodné je reprezentovat pomocí XML. Právě lékařská data ale mají typicky poměrně složitou sémantickou strukturu. Jedním z výsledků práce je systém pro specifikování omezujících podmínek na sémantiku XML dokumentů pomocí formulí logiky prvního řádu.

Předložili jsme vlastní nový formální konceptuální AC-model, jenž umožňuje takovéto hierarchické struktury modelovat. Speciálně jsou tedy všechny definice, věty a příklady týkající se AC-modelu originálními výsledky této diplomové práce.

Narozdíl od velké části výzkumných prací v oblasti konceptuálního modelování XML dat neřešíme nejprve otázku „co chceme modelovat“ a až následně „jaký bude obraz modelu na logické vrstvě“, nýbrž nejprve „co máme k dispozici na logické vrstvě“ a až následně „jak to můžeme modelovat na konceptuální úrovni“. Obrácení tohoto zavedeného špatného klišé vidím jako naprosto zásadní. Teorii budujeme, jak je v matematice obvyklé, od jednodušších (základních) pojmů ke komplikovanějším. Vyjdeme z regulárních stromových gramatik a logiky prvního řádu a budeme hledat vhodný jednoduchý abstraktnější pohled na ně. Budeme postupovat striktně logicky, v žádném případě nebudeme používat vágní spojení jako „entity reálného světa“ a podobně, jež se často v kontextu konceptuálního modelování vyskytují. Tyto pojmy patří do společenských věd. Naším cílem bude vytvořit v jistém smyslu minimalistický konceptuální model schopný namodelovat vše, co dokáží postihnout regulární stromové gramatiky a logika prvního řádu.

AC-model je vybudován na pevných matematických základech, což velice usnadňuje jakékoli zkoumání jeho výrazových možností. Zvláštností AC-modelu oproti některým jiným modelům je striktní oddělení modelování struktury XML dokumentů pomocí abstrakce regulárních stromových gramatik a

popisu sémantických pravidel, jež musí korektní dokumenty splňovat. Tato sémantická pravidla jsou popsána pomocí formulí logiky prvního řádu. Veškerá teorie je vybudována ryze formálně. Poté je na několika příkladech ukázána síla AC-modelu. Logika prvního řádu má svá omezení. S důkazem přes větu o kompaktnosti logiky prvního řádu je předložen konkrétní příklad nemožnosti namodelovat souvislý graf. Dále je dokázána ekvivalence A-modelu s regulárními stromovými gramatikami. Jako tomu bývá u většiny modelů se silnějšími sémantickými schopnostmi, problém existence validní instance konkrétního AC-modelu je nerozhodnutelný. To je dokázáno redukcí Hilbertova desátého problému. Je ukázáno, že problém zjištění, zda je daný XML dokument instancí daného AC-modelu, je NP-těžký.

V další části práce se zaměřujeme na logickou vrstvu, na kterou nemůžeme při tvorbě konceptuálního modelu nemyslet. Na logické úrovni máme k dispozici pro zápis regulárních stromových gramatik několik zavedených jazyků, za všechny jmenujme W3C DTD, W3C XML Schema a Relax NG. V práci je ukázáno, že jazyk Relax NG je výkonějším, matematicky čistším a elegantnějším jazykem pro zápis XML schémat něžli jazyky W3C DTD nebo W3C XML Schema.

To, co ale na logické vrstvě v současné době velice chybí, je standardizovaný jazyk pro zápis omezujících podmínek. V žádném případě nám nestačí jen klíče, na které pamatuje například W3C XML Schema. Potřebujeme mnohem silnější nástroj. Součástí diplomové práce je implementace validátoru XML dokumentů oproti dané sadě omezujících podmínek. Vytvořit na teoretické úrovni vhodný jazyk pro zápis omezujících podmínek, jenž bude i efektivně implementovatelný, v tom osobně vidím obrovskou budoucnost. Syntax XML dokumentů umíme postihnout maximálně efektivně, nyní musí přijít na řadu sémantika.

Literatura

[1] Makoto Murata, Dongwon Lee, Murali Mani, and Kohsuke Kawaguchi. Taxonomy of XML Schema Languages Using Formal Language Theory. 2001. `http://citeseer.ist.psu.edu/murata00taxonomy.html`.

[2] Roman Barták. *Automaty a gramatiky*. MFF UK, 2004.

[3] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications*. 2005.

[4] Petr Štěpánek. *Predikátová logika*. MFF UK, 2000.

[5] W3C. Xml Schema. Website, 2006. `http://www.w3.org/XML/Schema/`.

[6] OASIS. RELAX NG. Website, 2001. `http://www.relaxng.org/`.

[7] Murali Mani. EReX: A Conceptual Model for XML. 2004.

[8] D.W. Embley, S.W. Liddle, and R. Al-Kamha. Enterprise Modeling with Conceptual XML. 2004.

[9] Semantic Networks. 2006. `http://en.wikipedia.org/wiki/Semantic_network`.

[10] G. Dobbie, X. Wu, T.W. Ling, and M.L. Lee. Ora-SS: An Object-Relationship-Attribute Model for Semi-structured Data. 2000.

[11] Kohsuke Kawaguchi. W3C XML Schema Made Simple. 2001.

[12] Byron Long. An algorithm for comparing deterministic regular tree grammars. 2000.

[13] W3C. Extensible markup language (XML) 1.0 (third edition). Website, 2004. `http://www.w3.org/TR/REC-xml/`.

[14] Erik Wilde. Towards Conceptual Modeling for XML. *Proceedings of Berliner XML Tage*, 2005.

[15] Kawaguchi Kohsuke. Ambiguity Detection of RELAX Grammars. *Sun Microsystems, Inc.*, 2001.