VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

KONVERZE MODELŮ REGULÁRNÍCH JAZYKŮ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE David Navrkal
AUTHOR

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

KONVERZE MODELŮ REGULÁRNÍCH JAZYKŮ

CONVERSION OF MODELS OF REGULAR LANGUAGES

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE David Navrkal
AUTHOR

VEDOUCÍ PRÁCE Ing. Zbyněk Křivka, Ph.D.
SUPERVISOR

BRNO 2013

Abstrakt

Tato práce se zabývá jak didakticky prezentovat studentům modely regulárních jazyků se zaměřením na jejich vzájemné konverze. Tato technická zpráva je ponejvíce průvodcem návrhu a implementace aplikace, která tyto konverze demonstruje. Výsledná aplikace se snaží o co největší uživatelskou přívětivost a intuitivnost. Je určena jak pro hromadnou demonstraci probírané látky na přednáškách a cvičeních, tak i pro samostatné procvičování látky studenty. Aplikace se snaží demonstrovat teorii regulárních jazyků na konkrétních příkladech a má pomoci studentům tuto někdy na první pokus hůře stravitelnou teorii lépe pochopit a procvičit.

Co v této práci naopak nenaleznete, je podrobná teorie regulárních jazyků. Tato práce se nesnaží být učebnicí, nebo odbornou publikací zabývající se formálními jazyky. (Pro tyto účely jsou určeny přednášky a cvičení, či jiné publikace.) U čtenáře se zároveň předpokládá její znalost z výuky.

Abstract

This thesis deals with how to present to students didactically models of regular languages focusing on their mutual conversions. This technical report is mainly guide of design and implementation of the application that demonstrates these conversions. Final application is trying to be maximum user-friendly and intuitive. It is designed for both public demonstration of subject matter discussed in lectures and seminars, as well as home practicing by students. The application is trying to demonstrate the theory of regular languages that should be sometimes at the first try hard to understand, on concrete examples. This should help the students to better understand and practice theory.

What you don’t find in this work is a detailed theory of regular languages. This thesis is not intended to be a textbook or scientific publications dealing with formal languages. (For these purposes are lectures, exercises, and other publications.) The reader is also assumed that knows this theory.

Klíčová slova

Konečný automat, regulární výraz, regulární gramatika, konverze, regulární modely, formální jazyky, precedenční tabulka, syntaktická analýza, Qt5.

Keywords

Finite automata, regular expression, regular grammar, conversions, regular models, formal languages, precedence table, syntax analysis, Qt5.

Citace

Navrkal David: Konverze modelů regulárních jazyků, bakalářská práce, Brno, FIT VUT v Brně, 2013

Konverze modelů regulárních jazyků

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Zbyňka Křivky, PhD.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
David Navrkal
Datum (31. 7. 2013)

Poděkování

Tímto bych chtěl poděkovat za ochotu, vřelost, lidskost, trpělivost a odborné vedení panu doktoru Křivkovi.

Obsah

1 Úvod

Tato práce vznikla, jako pomoc studentům formálních jazyků s pochopením probírané látky. V této práci se budeme zabývat regulárními jazyky, jejich modely a především pak konverzemi těchto modelů. Tato aplikace se soustřeďuje na didaktickou část, takže nejenom že umí navzájem převádět modely regulárních jazyků, ale navíc se snaží pomoci uživateli pochopit, naučit a procvičit demonstrované algoritmy na konkrétních příkladech.

V aplikaci je předpřipravené množství příkladů, které si uživatel může zkusit vyřešit, zároveň má uživatel možnost si vyzkoušet i příklady vlastní. Demonstrace převodních algoritmů na konkrétních příkladech je klíčová a umožňuje tak uživateli snáze pochopit teorii.

Aplikace umí jeden převod prezentovat třemi různými způsoby, tak že si uživatel může nejprve prohlédnout, jak výpočet probíhá krok po kroku. Tomuto způsobu reprezentace budeme říkat Režim běhu. Poté co si uživatel prohlédl, jak výpočet probíhá na konkrétním příkladu, může si zkusit svůj první převod, tak že mu aplikace průběžně kontroluje, zdali postupuje správně. Tento režim budeme nazývat Režim běhu. Následně pokud si uživatel myslí, že už převod umí, může si další převod zkusit samostatně a aplikace mu pouze zkontroluje výsledek (případě i řekne výsledek správný). Tomuto poslednímu režimu budeme říkat Režim kontroly.

Využití aplikace může být spousta, nejdůležitější případy užití jsou ve výuce přímo ve škole, a to na přednášce či cvičení, kde může vyučující využít Režim běhu, kde může demonstrovat algoritmus na příkladech. Studenti tam mají možnost se v případě jakékoli nejasnosti ptát a vyučující jim může odpověď ukázat na nějakém příkladě. Zvídavý studenti mají totiž často na přednáškách a cvičeních dotazy na mezní chování algoritmů.

Po představení teorie a pár příkladů na přednášce následuje samostatné procvičování studenty. Učitel má tak možnost zadat studentům domácí úkoly. Student si může v Režimu kontroly vyzkoušet samostatně převod, v případě nejasností může využít i Režim běhu, kde si může krokovat celý algoritmus, nebo se jen podívat na správné řešení, či si zkusit výpočet za asistence aplikace.

A na závěr když se student připravuje na blížící se zkoušku, může si vyzkoušet vyřešit každý příklad, který ho napadne a bez nutnosti účasti druhé osoby si nechat zkontrolovat výsledek.

V současné době existují programy, které umí převádět některé modely, neexistuje však žádný, který by uměl převádět navzájem všechny modely regulárních jazyků, zároveň je i didakticky demonstroval a asistoval uživateli při výpočtu. Tato aplikace se o tyto cíle snaží.

Dále se chci zmínit o tom, že předpokládám, že čtenář zná základní pojmy týkající se formálních jazyků, především pak jazyků regulárních, pojmů, definic, modelů a algoritmů jejich vzájemných převodů a další věci s tím související. Přesto musím uvést alespoň základní definice,

protože při studiu teorie jsem narazil na některé mírně odlišné definice a chci, aby tato práce byla srozumitelná i jiným, kteří nestudovali na naší fakultě. Méně důležité pojmy a definice jsem do hlavního těla práce neuváděl a můžete je nalézt pouze jako přílohy.

V této práci jsem se soustředil především na návrh a implementaci.

2 Teorie

V této části si popíšeme teorii týkající se modelů regulárních jazyků. V případě, že dané definice a konverzní algoritmy dobře znáte, můžete tuto část přeskočit a přečíst si konverzní algoritmy mezi těmito modely.

2.1 Konečný automat

Následující definici jsem přebral z (1 str. 101) .

Definice:

Konečný automat (KA) je pětice:

$M = (Q,\Sigma,R,s,F)$, kde:

- *Q je konečná množina stavů*
- *Σ je vstupní abeceda*
- *R je konečná množina pravidel tvaru: $p\ a \rightarrow q$, kde $q, p \in Q$, $a \in \Sigma \cup \{\varepsilon\}$.*
- *$s \in Q$ je počáteční stav*
- *$F \subseteq Q$ je množina koncových stavů*

Tuto definici zde uvádím zejména proto, že jsem se setkal na internetu i v literatuře s odlišnými definicemi, které definovali například *s* ne jako počáteční stav, ale jako množinu počátečních stavů. (Pro KA podle definice z (1 str. 101) se dá nedeterministický výběr počátečního stavu simulovat ε-přechody z počátečního stavu do „simulovaných počátečních stavů“.)

2.2 Regulární výrazy

Následující definici jsem přebral z (1 str. 33)

Definice:

Nechť Σ je abeceda. Regulární výrazy nad abecedou Σ a jazyky, které značí, jsou definovány následovně:

$\emptyset$ je RV značící prázdnou množinu (prázdný jazyk)

ε je RV značící jazyk $\{\varepsilon\}$

a, kde $a \in \Sigma$, je RV značící jazyk $\{a\}$

Nechť r a s jsou regulární výrazy značící po řadě jazyky L_r a L_s, potom:

$(r.s)$ je RV značící jazyk $L = L_r\ L_s$

(r+s) je RV značící jazyk $L = L_r \cup L_s$

(r) je RV značící* $L = L_r^*$

V literatuře můžeme nalézt i definice r^+ značící *rr** nebo *r*r*. My tohle rozšíření nebudeme používat, protože vyjadřovací síla regulárních výrazů podle předešlé definice zůstává stejná. Pro zjednodušení můžeme používat *rs* na místo notace *r.s*.

Kvůli redukci počtu závorek budeme uvažovat následující priority: „* > . > +“ („*“ má větší prioritu než „.“ a to má větší prioritu než „+“).

2.3 Regulární gramatika

Pro jistotu si zde uvedeme i obecnou definici gramatiky a následně uvedeme i definici pravé lineární gramatiky a pravé regulární gramatiky.

Definice gramatiky:

Gramatika G je čtveřice G = (N, Σ, P, S), kde

- *N je konečná množina nonterminálních symbolů.*
- *Σ je konečná množina terminálních symbolů.*
- *P je konečná množina kartézského součinu (N ∪ Σ)*N.(N ∪ Σ)*Σ, nazývaná množinou přepisovacích pravidel.*
- *S ∈ N je výchozí (startovací) symbol gramatiky G.*

Prvek (α, β) ∈ P je přepisovací pravidlo a zapisuje se ve tvaru α → β, kde α je levá strana, β je pravá strana pravidla, α → β.

Viz (2).

V předchozí části jsme si představili definici obecné gramatiky, nás však bude zajímat gramatika regulární, konkrétně nyní pak pravá lineární gramatika.

Definice pravé lineární gramatiky:

Pravá lineární gramatika je taková, která má přepisovací pravidla ve tvaru:

A →xB, nebo A →x kde A, B ∈N, x ∈ Σ (kde Σ* je množina řetězců nad abecedou Σ)*

Viz (2).

Nyní si nadefinujeme pojem pravá regulární gramatika. Jak následně uvidíte, jedná se o speciální typ pravé lineární gramatiky kde x je řetězec délky jedna.

Definice pravé regulární gramatiky:

Pravá regulární gramatika je taková, která má přepisovací pravidla ve tvaru:

$A \rightarrow xB$, *nebo* $A \rightarrow x$ *kde* $A, B \in N, x \in \Sigma$

Viz (2).

3 Specifikace požadavků

Následující požadavky jsem získal opakovanými debatami s vedoucím mé bakalářské práce a mými vlastními úvahami.

Aplikace se má soustředit na grafické uživatelské rozhraní. Uživatelské rozhraní má být pokud možno co nejvíce intuitivní a přehledné. Cílem je, aby se uživatel mohl soustředit na pochopení a následné procvičení demonstrovaných konverzí a ne před použitím aplikace číst dlouhé dokumentace. Aplikace má studentovi pomoci ve studiu, a ne mu brát jeho drahocenný čas.

Aplikace má sloužit jako výukový nástroj, který má pomoci studentům formálních jazyků a teoretické informatiky pochopit a naučit se používat konverzní algoritmy modelů regulárních jazyků.

Aplikace má být určená na konverzi malých objemů dat (řádově do desítek komponent regulárních modelů), hlavní důraz se klade na didaktickou demonstraci a samostatné procvičení převodních algoritmů.

S vedoucím mé bakalářské práce jsme se dohodli na implementaci následujících převodních algoritmů:

- Obecný konečný automat (KA) na KA bez ε-pravidel.
- KA bez ε-pravidel na deterministický KA bez nedostupných stavů (DKA).
- DKA na dobře specifikovaný konečný automat (DSKA).
- DSKA na minimální konečný automat.
- KA na regulární výraz.
- Regulární výraz na KA.
- KA bez ε-pravidel na pravou regulární gramatiku.

Pokud budou výše uvedené převody implementovány, tak i bonusově demonstraci přijímání řetězce konečným automatem.

Dále by pak aplikace měla umět následující módy převodů:

- **Editační režim**, ve kterém si uživatel vybere jednu z konverzí a vytvoří model (regulární výraz, KA, pravou regulární gramatiku), který chce převádět. Uživatel by měl mít možnost kterýkoli z následujících módů přerušit a vrátit se do tohoto módu.
- **Krokový režim**, ve kterém uživatel převádí jeden model na druhý po jednotlivých krocích algoritmu. Tento režim je nejdůležitější, protože v něm se uživatel učí krůček po krůčku používat daný konverzní algoritmus. V tomto režimu se postupně zvýrazňují jednotlivé kroky algoritmu, které má uživatel vykonat.

- **Režim kontroly**, ve kterém uživatel samostatně převede jeden model na druhý a program mu pouze zkontroluje, zda převod provedl správně. Tento mód bude užitečný hlavně uživatelům, kteří převodní algoritmus již pochopili a chtějí si zkontrolovat svou samostatnou práci.
- **Režim běhu**, ve kterém program bez zásahu uživatele převede jeden model na druhý. V tomto režimu si může uživatel algoritmus krokovat a pozorovat jak mají být správně provedeny jednotlivé kroky. Využití má především pro uživatele, kteří si chtějí prohlédnout postup výpočtu, popřípadě srovnávají svůj postup s postupem správným a hledají, kde udělaly chybu.

Aplikace má být využitelná pro hromadnou demonstraci (tj. na přednášce, nebo na demo cvičení), z čehož vyplývá, že mají být vidět použité fonty i ze zadních řad. Zároveň by měla být použitelná na osobních počítačích pro samostatné procvičení přednášené látky studenty. Z obojího plyne, že by aplikace měla mít možnost měnit velikost použitých písmen a změnu měřítka grafických prvků, tak aby byla dobře vidět, jak na data projektoru, tak na obrazovkách osobního počítače.

Primárně má aplikace běžet na operačním systému Microsoft Windows 7. Vítaná by byla možnost provozovat aplikaci i na operačních systémech *Unixového typu*, Linux apod., vzhledem k odbornému zaměření budoucích uživatelů, mezi kterými je vysoké procento těch, kteří právě tyto operační systémy používají.

Výsledný produkt má být hlavně určen pro použití na Fakultě informačních technologii VUT v Brně, proto použité názvosloví, definice a barevné značení bude odpovídat tomu, které se používá v předmětech Formální jazyky a překladače a Teoretická informatika vyučovaných na naší fakultě.

4 Návrh

Kapitolu Návrh jsem záměrně rozdělil do dvou částí, toho co uvidí uživatelé (návrh grafického uživatelského rozhraní) a na návrh implementace (objektový návrh). K návrhu jsem přistupoval s papírem, tužkou a mými nápady. Ne vše se mi povedlo i zdárně implementovat. Jednak kvůli časové tísni a také protože, že ne všechny věci byly v prostředí *Qt 5* standardní a jejich implementace by odporovala zvyklostem při použití tohoto aplikačního rozhraní, přesto jsem se snažil i přes tyto problémy většinu myšlenek, nápadů a konceptů z návrhu implementovat.

4.1 Návrh grafického uživatelského rozhraní

Při návrhu grafického uživatelského rozhraní (GUI) jsem vycházel z klasického rozložení a vzhledu ovládacích prvků. Mým cílem bylo vytvořit aplikaci s chováním a vzhledem, na které je uživatel zvyklý z obvyklých grafických aplikací z prostředí MS Windows a ne aplikaci s experimentální GUI. Aplikace má mít intuitivní grafické uživatelské rozhraní, aby uživatele nerozptylovala od učení se nových algoritmů. Pro usnadnění práce se po najetí myši na ovládací prvek zobrazí nápověda.

Následující obrázky jsem vytvořil pomocí programu Balsamic pro tvorbu návrhu uživatelských rozhraní, tzv. „mockupů“, proto se může vzhled výsledné aplikace mírně lišit.

4.1.1 Návrh zobrazení základních elementů

Při návrhu grafických elementů jsem postupoval od nejobecnějších ke specifičtějším.

4.1.1.1 Hlavní okno

Ze všeho nejdříve se podíváme, jak bude aplikace vypadat, když ji uživatel poprvé spustí. Důležitý je totiž první dojem. Při spuštění programu, aplikace pro lepší první dojem přivítá uživatele a zároveň mu říká, jak může začít pracovat. Není totiž uživatelky moc příjemné, když uživatel poprvé spustí program a neví jak má začít a to i přes to, že muže být jinak aplikace jinak dobře navržená. V uživateli může zanechat dojem, že je nepřehledná.

První dojem je opravdu hodně důležitý. Jak říká známá poučka, „když uděláte špatný první dojem, musíte udělat deset dobrých věcí, abyste tento dojem zvrátili“, samozřejmě platí i naopak. Nejen toto, ale i více o dobrém uživatelském rozhraní se můžete dočíst v (3).

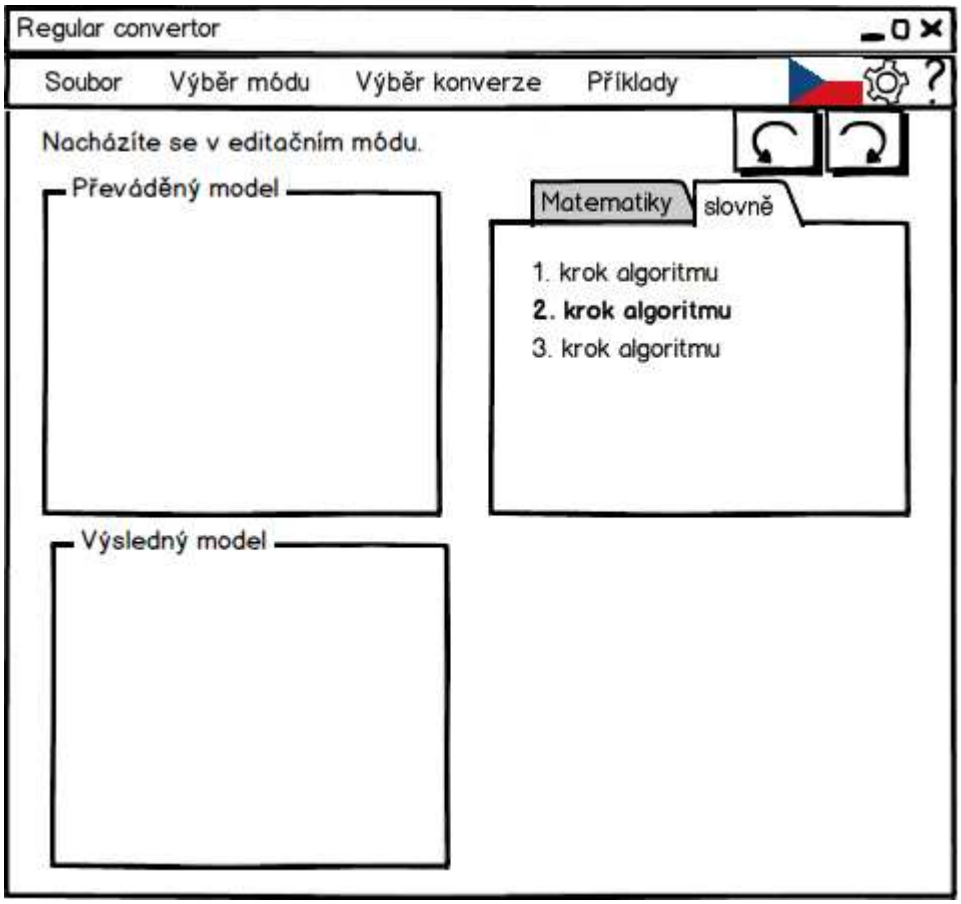
Obrázek 1: Hlavní okno programu před výběrem některé z konverzí

Na obrázku 1 v hlavním menu se popořadě nachází položky **Soubor**, pod kterou se nachází standardní volby jako, uložení výchozího modelu pro převod do souboru a načtení modelu ze souboru. Dále pak **výběr módu** a **výběr konverze**, obojí viz kapitola Specifikace požadavků str. 7. Jako poslední položka menu obsahuje **příklady,** ve kterých najde uživatel předpřipravené příklady modelů a jejich konverzí, které si může sám vyzkoušet. Napravo od příkladů uživatel uvidí vlajku ukazující **aktuální jazyk**, po kliku na něj se zobrazí výběr ostatních lokalizací. Napravo od něj se nachází ikony pro **nastavení** a práci s **nápovědou**. Text v okně informuje uživatele, aby si vybral konverzi, příklad, nebo načetl konverzi ze souboru a následně zvolil mód.

4.1.1.2 Módy

V aplikaci se budou nacházet čtyři módy a to editační mód, krokový mód, režim kontroly a režim běhu. Nyní se pojďme podívat na návrh jednotlivých módů.

Začneme **Editačním režimem**. Na obrázku 2 vpravo nahoře můžeme vidět dvě tlačítka symbolizující předchozí a následující krok v editaci. Jsou výhodná, když byste si náhodou část modelu omylem smazali, tak se můžete vrátit v editaci zpět, anebo vpřed.

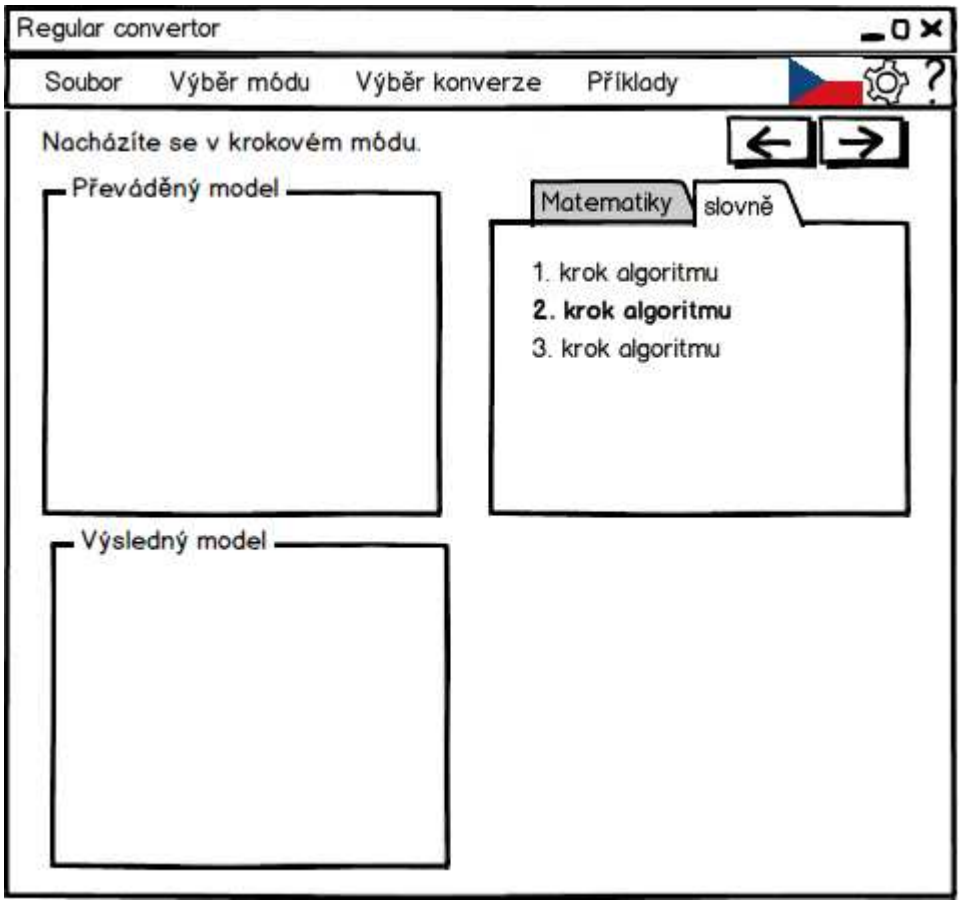


Obrázek 2: Editační režim

Pod těmito tlačítky můžete vidět okno s dvěma záložkami, v obou se nachází algoritmus převodu. Jediný rozdíl je, že v záložce pojmenované **matematicky** se nachází algoritmus v zapsaný v matematické podobě a druhé pojmenované **slovně** jsou instrukce ve formě vět.

Nalevo od tohoto okna je převáděný model, pod kterým se nachází okno na výsledný model. (Všechna následující okna převodu budou mít schéma rozložení takové, že vpravo nahoře je okno s algoritmem, nalevo od něj je převáděný model a v dolní části hlavního okna jsou okna potřebná pro konkrétní převod.)

Další představený je **Krokový režim**. Návrh rozložení hlavního okna pro tento režim je na obrázku 3.



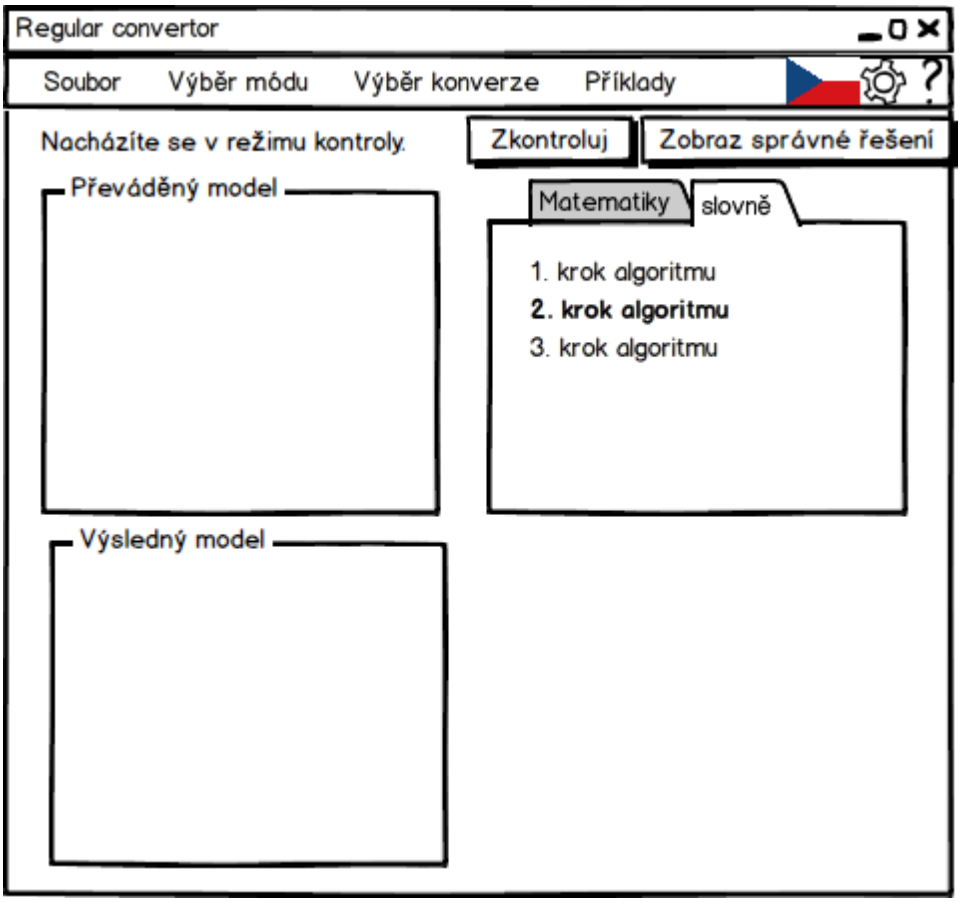
Obrázek 3: Krokový režim

Vpravo nad oknem algoritmu můžeme stejně jako na předchozím obrázku nalézt tlačítka „vzad“ a „vpřed“ s ikonami ve tvaru šipek. Na rozdíl od předchozího režimu mají odlišný význam. Uživatel si pomocí nich může listovat v historii kroků algoritmů, které už provedl. Má tak možnost si prohlédnout celý postup výpočtu a může mu to lépe pochopit daný konverzní algoritmus.

V tomto módu uživatel postupně provádí jednotlivé kroky algoritmu. Pokud se podaří uživateli provést jeden krok, tak se v algoritmu automaticky zvýrazní další krok, informující uživatele, že se mu povedlo úspěšně provést krok předchozí.

Původně jsem zamýšlel, že bude existovat tlačítko, které když uživatel stiskne, tak mu aplikace zkontroluje, zda provedl aktuální krok správně, pokud ano, tak se zvýrazní další krok algoritmu, který má uživatel provést. Následně jsem tento koncept zavrhl, protože by uživatel nedělal skoro nic jiného, nežli mačkal toto tlačítko a myslím, že by ho to přestalo velmi rychle bavit. Jako přirozenější se mi jeví, aby aplikace automaticky kontrolovala postup uživatele a při splnění se automaticky posunula na další krok.

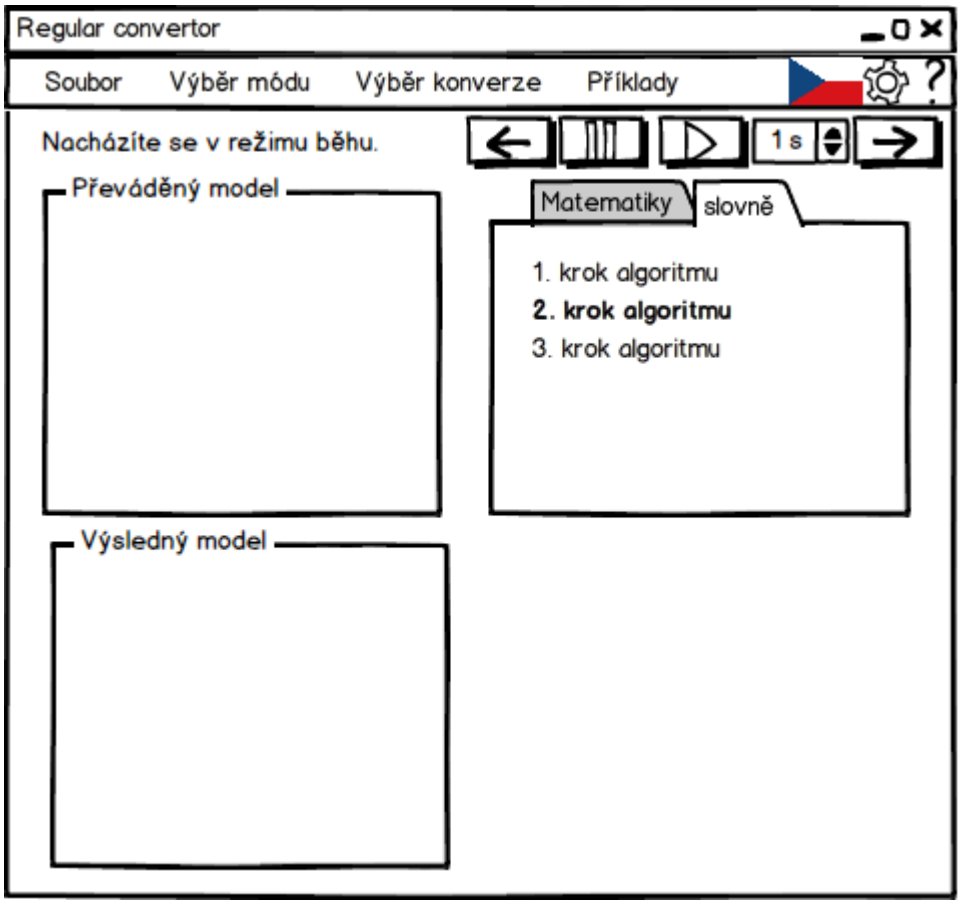
Poté co si uživatel zkusil převést daný model krok po kroku za pomoci krokového režimu, je vhodné, aby si uživatel zkusil převést model úplně sám. Jedině tehdy se uživatel skutečně přesvědčí, zdali se převod správně naučil a rozumí všem krokům. Tohle vše umožňuje **Režim kontroly**, jehož grafický návrh můžete vidět na obrázku 4.



Obrázek 4: Režim kontroly

Nad oknem pro algoritmus jsou vidět dvě tlačítka. První, „Zkontroluj“ řekne uživateli, zdali je výsledný model správný. Druhé, „Zobraz správné řešení“ ukáže uživateli, jak má správný výsledek vypadat, bylo by totiž z didaktického hlediska špatné, kdybychom uživateli řekli, že jeho řešení je nesprávné a neposkytli mu správné řešení.

Poslední a neméně důležitý je **Režim běhu**. Jeho grafický návrh můžete nalézt na obrázku 5. V tomto módu si uživatel může krokovat algoritmus a sledovat tak jak funguje na vlastních příkladech.

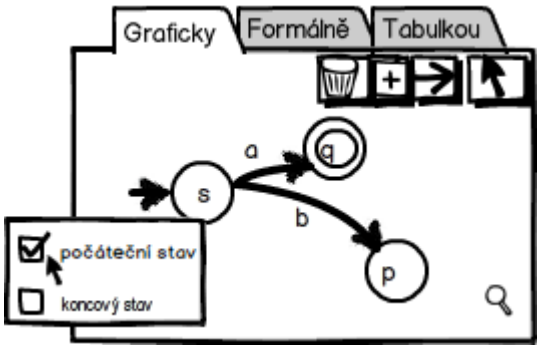


Obrázek 5: Režim běhu

Na rozdíl od ostatních má nejvíce tlačítek na ovládání. Uživatel si může krokovat algoritmus stiskem tlačítek s ikonou šipek nacházejících se úplně vlevo a úplně vpravo. Mezi nimi jsou tlačítka s ikonami pauzy a play. Po navolení prodlevy mezi kroky si pomocí těchto dvou tlačítek můžeme buď algoritmus zastavit, anebo nechat volně běžet s přednastavenou dobou prodlevy mezi jednotlivými kroky.

4.1.1.3 Reprezentace konečného automatu

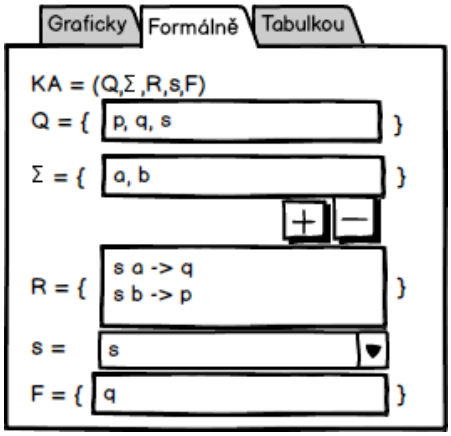
Na konečný automat můžeme reprezentovat třemi různými způsoby, první a to **grafický pohled** můžete vidět na obrázku 6 (ostatní dva si představíme následně).



Obrázek 6: Grafický pohled na KA po kliknutí pravým tlačítkem myši na uzel s

V grafickém pohledu se nachází v pravém horním rohu tlačítka na **mazání** uzlů a hran, dále pak tlačítko na **přidávání uzlů** a **přidávaní hran** a na **přemisťování** hran a uzlů. Pokud chceme zároveň přesouvat více uzlů, můžeme je označit klikem myši spolu s držením klávesy „ctrl“. Po kliknutí pravím tlačítkem na uzel, jak je vidět na obrázku, má uživatel možnost nastavit uzel jako koncový a jako startovní. (Aplikace dovolí uživateli nastavit jako startovní pouze jeden uzel, viz kapitola Teorie str. 4.) Vpravo dole se nachází ikona pro přiblížení, nebo oddálení objektů v grafickém pohledu, stejného efektu se dá dosáhnout i stiskem klávesy „ctrl“ a pohybem kolečka myši.

Na následujícím obrázku 7 můžete vidět druhý a to **formální pohled** na stejný KA. Kde *Q* je konečná množina stavů, *Σ* je abeceda, *R* je množina přechodů, *s* je startovní stav a *F* je konečná množina koncových stavů.



Obrázek 7: Formální popis KA z obrázku 2

Uživatel bude moci pomocí textového řádkového editoru upravovat množiny *Q* a *Σ* a *F*. Jak symboly abecedy, tak názvy stavů musí být odděleny čárkou a libovolným množstvím mezer. Řádkový editor pro symboly abecedy akceptuje pouze jednoznakové symboly s výjimkou znaku mezera, která slouží spolu s čárkami jako oddělovač. Jiné než výše uvedené způsoby zápisů nebudou akceptovány. Po dokončení editace (stisk klávesy „enter“, nebo ztráty zaměření, anglicky focus) jedné z množin *Q*, *Σ* nebo *F* jsou duplicitní prvky odstraněny a ostatní se automaticky seřadí podle abecedy a jako oddělovač se použije čárka následovaná právě jednou mezerou, s výjimkou posledního prvku, za který se čárka nedoplní. Při editaci množiny F se budou navíc při psaní automaticky nabízet prvky z množiny Q, prvky, které nejsou z množiny Q, se po dokončení editace automaticky odstraní.

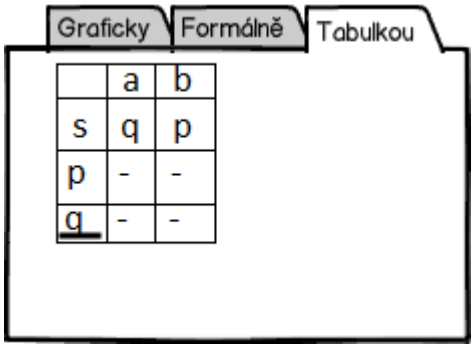
Prvky množiny *R* se přidávají pomocí tlačítka „+“, po jehož stisknutí se objeví dialogové okno, v němž může uživatel nastavit nový přechod. Pokud chce uživatel smazat prvky z množiny *R*, pak si je

označí a s pomocí tlačítka „-“ je odstraní. Dialog pro editaci se zobrazí pomocí dvojkliku na položku z množiny *R*.

Startovní stav si uživatel vybere jeden prvek z množiny Q pomocí rozbalovacího seznamu (anglicky „combo box“). Pokud uživatel nezvolí vlastní počáteční stav, tak se automaticky vybere abecedně první stav.

4.1.1.4 Tabulková reprezentace konečného automatu

Třetím možným zápisem je zapsat KA **tabulkou**. Tohoto zápisu nejvíce využijeme při algoritmu minimalizace KA.



Obrázek 8: Tabulkový popis obrázku 2

Na obrázku 8 můžete vidět zápis KA tabulkou. V prvním sloupci je jméno uzlu, při čemž na prvním řádku je počáteční uzel a všechny uzly, které jsou podtržené, jsou uzly počáteční. V záhlaví tabulky jsou uvedeny symboly abecedy a pod nimi je naznačena množina přechodů. Pokud z místa nevede hrana, se symbolem abecedy, pak je v tabulce uveden znak pomlčky. V tabulce je zapsán opět stejný KA jako z předchozích obrázků.

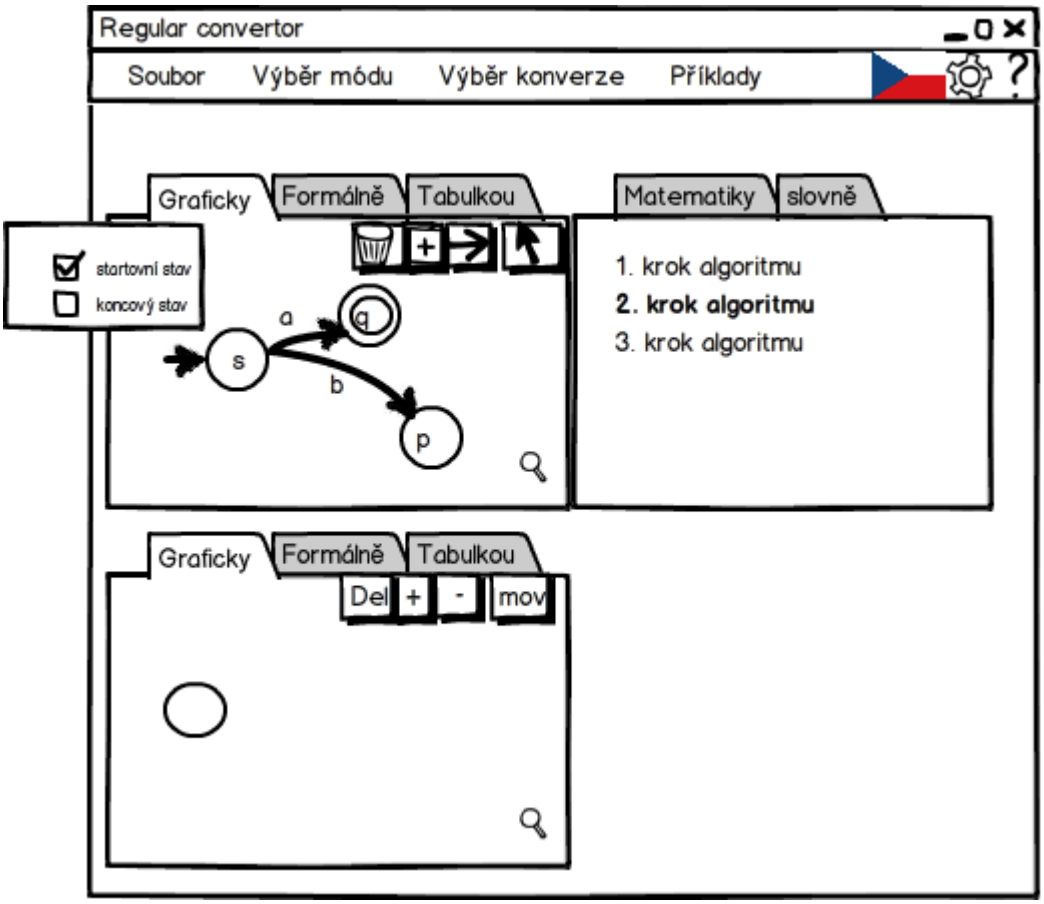
4.1.2 Návrh zobrazení převodů

Při návrhu GUI pro převody jsem se snažil držet schématu, že vždy vpravo nahoře bude převodní algoritmus, nalevo od něj bude výchozí model a v dolní části obrazovky okna potřebná k získání výsledného modelu.

4.1.2.1 Konverze konečných automatů

Následující schéma zobrazené na obrázku 9 znázorňuje převod konečného automatu na jeho speciální typy a to:

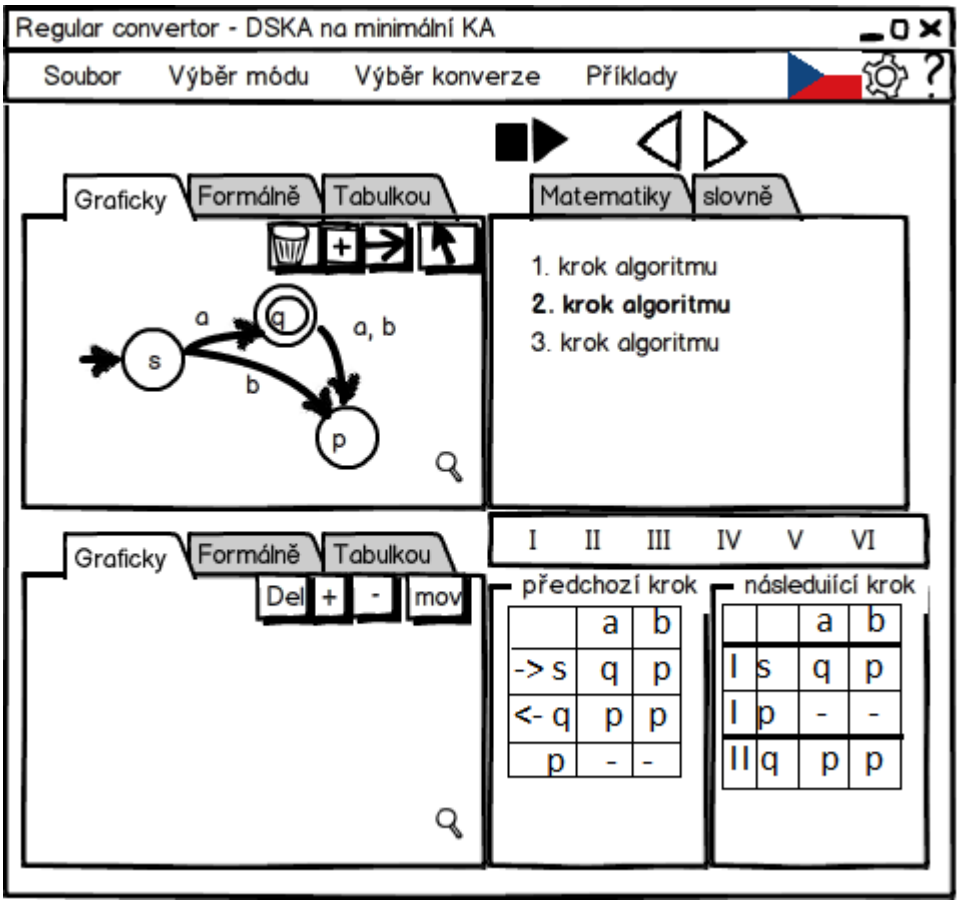
- Obecný konečný automat KA na KA bez ε-pravidel.
- KA bez ε-pravidel na deterministický KA bez nedostupných stavů (DKA).
- DKA na dobře specifikovaný konečný automat (DSKA).



Obrázek 9: Hlavní okno pro převod konečných automatů

Uživatel bude postupovat podle převodního algoritmu a postupně bude v levém dolním okně tvořit výsledný KA. Pokud bude uživatel převádět v *krokovém režimu,* pak nebude moci po dobu jeho spuštění moci upravovat výchozí KA.

Speciálním typem převodu konečných automatů je převod **DSKA** na **minimální konečný automat**. Při tomto převodu potřebujeme dvě pomocná okna, ve kterých bude uživatel hledat rozlišitelné stavy.

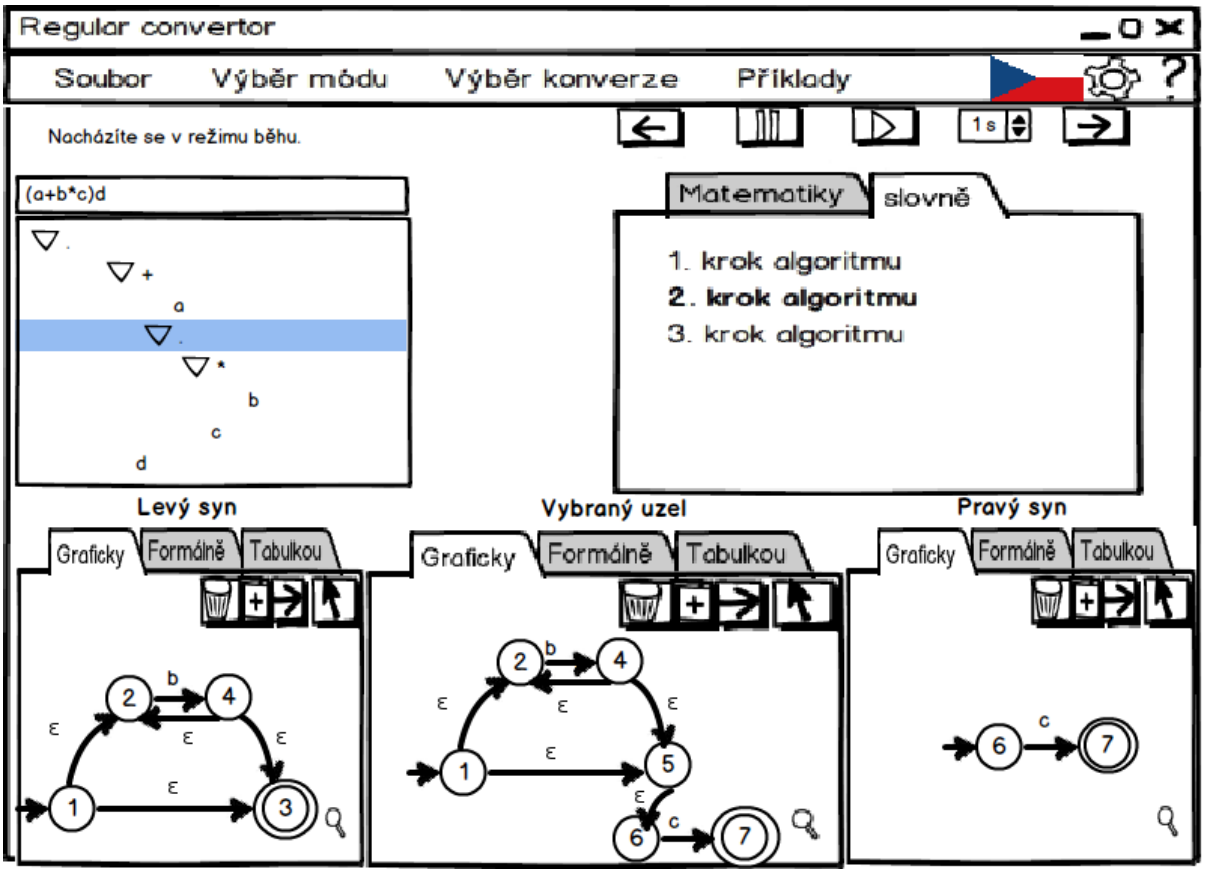


Obrázek 10: Obrazovka pro převod DSKA na minimální KA

Jak můžete vidět na obrázku 10, tyto dvě okna jsem umístil do volného prostoru vpravo dole pod okno s algoritmem. V levém okně pojmenovaném **předchozí krok** uživatel vidí předchozí krok algoritmu. Nový krok, který se následně stane krokem předchozím, tvoří uživatel v levém okně pojmenovaném **následující krok**. Jelikož je konvence v převodu na minimální KA označovat jednotlivé skupiny uzlů římskými číslicemi, je nad těmito dvěma okny prvek, ve kterém si může uživatel vybrat, kterou římskou číslicí chce označit kterou skupinu. Po rozdělení uzlů KA do finálních skupin musí uživatel výsledný automat překreslit do výstupního okna pro minimální automat.
Z tohohle pohledu lze okna *předchozí* a *následující krok* brát jako okna pomocné.

4.1.2.2 Konverze regulárního výrazu na konečný automat

Návrh okna pro převod regulárního výrazu můžete nalézt na obrázku 11.



Obrázek 11: Převod regulárního výrazu na konečný automat

V levé horní části okna se nachází řádkový editor pro regulární výraz. Když uživatel edituje regulární výraz, tak se automaticky vytváří v okně pod řádkovým editorem syntaktický strom. Pokud se nachází v řádkovém editoru neplatný regulární výraz (definici validního regulárního výrazu můžete nalézt na straně 4, tak se jeho barva změní na červenou, informujíc uživatele, že udělal chybu a zmizí i syntaktický strom.

V dolní části obrazovky vidíme tři editory konečných automatů. Prostřední z nich je konečný automat odpovídající regulárnímu výrazu v označeném uzlu. (V našem případě se jedná o regulární výraz „b*c“.) Levý konečný automat odpovídá levému synu (v našem horizontálním zobrazení syntaktického stromu vrchnějšímu potomkovy) a pravý KA je analogický levému KA.

Převod v Krokovém režimu probíhá tak, že uživatel postupuje od listů syntaktického stromu směrem ke kořeni. V nekoncových uzlech se nachází operátory a v jejich synovských uzlech se nacházejí operandy. Takto uživatel pomocí atomických operací skládá výsledný konečný automat, který se nachází v kořeni stromu. Všechny uzly, ke kterým není doplněný KA, jsou označeny žlutě – čekají na vyplnění. Pokud uživatel vyplní konečný automat pro jeden uzel syntaktického stromu správně, změní se jeho barva na zelenou – signalizující úspěch, v opačném případě na červenou – signalizující neúspěch.

Tento přístup je velmi vhodný i pro Režim kontroly, kdy uživatel nemusí vyplňovat všechny uzly, ale jenom některé, například může vynechat triviální reprezentaci KA pro listové uzly syntaktického stromu, anebo i u jednoduššího regulárního výrazu napsat rovnou správný výsledek do kořenového uzlu. Navíc pokud uživatel udělá chybu, tak se mu červeně zvýrazní všechny uzly stromu, pro které vytvořil nesprávný KA.

5 Implementace

Celý projekt jsem implementoval v C++ za použití grafické knihovny Qt 5. Zdrojové kódy byly vyvíjeny pod linuxovým operačním systémem a testovány v MS Windows 7 a XP.

5.1 Implementace editoru konečného automatu

Implementace editoru KA sestává ze tří částí. První je interní třída reprezentující KA programově. Druhá je grafický pohled na KA a třetí je formální pohled na KA.
Pojďme se podívat na druhou část.

Grafický editor KA jsem implementoval ve vlastní režii pomocí QGraphicsView a QGraphicsScene. Uzly jsem implementoval jako vlastní QGraphicsitem. Přechody mezi uzly (šipky) jsem implementoval pomocí oddědění z QGraphicLineItem.

Po výpočet automatického rozmístění uzlů jsem chtěl původně použit knihovnu pro práci s grafy Graphviz. Avšak díky nekvalitní dokumentaci a nedostatku času jsem byl nucen naimplementovat vlastní rozmísťování uzlů. Můj způsob řešení spočíval v tom, že pro každý nový uzel jsem si nechal náhodně vygenerovat pozici a následně zkontroloval, jestli na tomto místě nekoliduje s již umístěným uzlem. Takto má algoritmus *n* pokusů (v mé implementaci jsem stanovil *n* rovno 1000) na to aby umístil uzel na své místo. Pokud ani po *n* náhodných pokusech algoritmus nenalezne místo, kde by mohl umístit nový uzel bez kolize s ostatními, tak jej umístí na takovou pozici z *n* pokusů takovou, kde kolidoval s co nejmenším počtem uzlů. Tento způsob funguje rychle s počtem uzlů v řádu desítek, dokud již není scéna zcela zaplněná. Jelikož se jedná o výukový program, neshledávám toto omezení za zcela zásadní.

5.2 Implementace editoru regulárních výrazů

Při implementaci editoru RV jsem řešil hlavní dva problémy a to kontrolu jeho validity a jak jej vhodně zobrazit uživateli. Pro zobrazení jeho syntaktického stromu jsem se rozhodl použít QListView. Abych mohl získat syntaktický strom, rozhodl jsem se použít syntaktickou analýzu zdola nahoru. Precedenční tabulku můžete vidět v tabulce číslo Tabulka 1.

	+	.	*	(	)	i	$
+	>	<	<	<	>	<	>
.	>	>	<	<	>	<	>
*	>	>	>		>	<	>
(	<	<	<	<	=	<	
)	>	>	>		>		>
i	>	>	>		>		>
$	<	<	<	<		<	

Tabulka 1: Precedenční tabulka

V prvním řádku se nachází další čtený symbol a v levém sloupci se nachází terminál nejblíže vrcholu zásobníku. Uvnitř tabulky se nacházejí symboly „<“, „>“, „=“ a „ “ (prázdná buňka tabulky), které popořadě znamenají „shift”, „redukce“, „rovno“ a „chyba“.

Jelikož se v zápisu regulárního výrazu obvykle vynechává operátor „.“ tak jej bylo nutné pro vytvoření syntaktického stromu interně doplnit. Znak tečky se doplňoval, pokud aktuální znak byl znak abecedy, nebo levá závorka, nebo hvězdička a zároveň pokud následující znak byl znak abecedy, nebo levá závorka.

6 Závěr

Jsem si vědom toho, že odevzdávám práci nedokončenou. Bylo to z důvodu nemoci (díky níž je toto můj první pokus o zvládnutí bakalářské zkoušky), osobních, existenčních a hlavně časových důvodů. Proto nežádám lepší známku, než F. Důvod proč tuto práci odevzdávám nedokončenou je ten, že se chci pokusit zvládnout alespoň ústní část závěrečné zkoušky a příští rok si dodělat jen tuto písemnou část zkoušky. Zároveň se chci omluvit jak mému vedoucímu bakalářské práce za to, že jsem práci nestihl dodělat a pak i oponentovy za čtení nedokončené práce.

Literatura

1. **Meduna, Alexandr.** *Automata and Languages: theory and applications.* London : Springer, 2000. ISBN 81-8128-333-3.

2. **Smrčka, Aleš, Vojnar, Tomáš and Češka, Martin.** *Teoretická informatika - studijní opora.* Brno : s.n., 2011. p. Dostupné na URL: <http://www.fit.vutbr.cz/study/courses/TIN/public/Texty/oporaTIN.pdf>.

3. **Krug, Steve.** *Nenuťte uživatele přemýšlet.* Brno : Computer Press, 2006. ISBN 80-251-1291-8.

Seznam obrázků