FLP 2021/2022 – funkcionální projekt: Haskell

Daniel Uhříček
`iuhricek@fit.vut.cz`

Úvod

Toto je zadání funkcionálního projektu do předmětu Funkcionální a logické programování 2021/2022. Za projekt zodpovídá Ing. Daniel Uhříček, konzultace jsou možné kdykoliv po předchozí domluvě.

Obecné požadavky

Svá řešení odevzdávejte v předepsaném tvaru (zkomprimovaný archiv zip, viz Závazné pokyny pro projekty) prostřednictvím WIS do datového skladu předmětu FLP. Odevzdaný projekt musí obsahovat zdrojové texty v Haskellu. V hlavní složce musí být soubor `Makefile`, který program přeloží a výsledný binární kód umístí také do hlavní složky. Dále se doporučuje přiložit stručný popis všeho, co nebylo dořešeno nebo naopak bylo implementováno nad rámec zadání.

Například soubor `flp-fun-xlogin00.zip` po rozbalení obsahuje:

```
Makefile
src
├── Main.hs
├── Minimize.hs
├── ParseInput.hs
└── Types.hs
doc
├── README.md
└── test-description.txt
test
├── test01.in
├── test01.out
├── test02.in
└── test02.out
```

Pro překlad použijte kompilátor `ghc` s volbou `-Wall`. Cílový program pojmenujte **flp21-fun**. Můžete využít standardní knihovny z balíku `base`, případně knihovny z balíků `containers`, `parsec`, `vector`, `split`, `directory`. Za referenční stroj je považován server merlin. Před odevzdáním si ověřte, zda lze zip rozbalit a program přeložit a spustit, například:

```
unzip flp-fun-xlogin00.zip
make
./flp21-fun -2 test.in
```

Hodnocení

Za vypracovaný projekt lze získat až 12 bod. Hodnocena bude míra splnění zadání, kvalita řešení, čistota a kvalita kódu. Za obzvláště kvalitní řešení lze získat prémiové body navíc. Vyvarujte se nestandardních funkcí, které obcházejí bezpečné otypování nebo obcházejí zapouzdření vedlejších efektů – nepoužívejte nic, co má ve svém jméně slovo `unsafe`. Snažte se psát čistý kód respektující různé varování a připomínky, které zazněly na přednáškách a cvičeních. Prosím také o dodržení, ať: úvodní řádky zdrojových kódů obsahují název projektu, login, jméno autora a rok řešení; součástí definic na globální úrovni jsou typové anotace a stručný a výstižný komentář; je zřejmé, co a z jakých modulů importujete[1]. Projekt vypracovávejte samostatně.

Následuje popis jednotlivých zadání.

[1] `https://wiki.haskell.org/Import_modules_properly`

1 BKG-2-CNF

Vytvořte program převádějící bezkontextové vlastní gramatiky (dále BKG) na bezkontextové gramatiky v Chomského normální formě (dále CNF). Vlastní gramatika je (dle Def 4.11 Opory TIN) gramatika beze zbytečných symbolů, bez cyklů a bez epsilon-pravidel.

1.1 Rozhraní programu

Program bude možné spustit:

`flp21-fun` *volby* `[`*vstup*`]`

kde

- *vstup* je jméno vstupního souboru (pokud není specifikováno, program čte standardní vstup) obsahujícího BKG
- *volby* jsou parametry ovlivňující chování programu:
  - `-i` na standardní výstup se vypíše pouze načtená a do vaší vnitřní reprezentace převedená BKG. Nevypisujte jen načtený řetězec; tato volba ověřuje, že váš program dokáže gramatiku syntakticky analyzovat a znovu vypsat.
  - `-1` na stdout se vypíše BKG bez jednoduchých pravidel. Postupujte podle algoritmu 4.5 z opory předmětu TIN.
  - `-2` na stdout se vypíše CNF. Postupujte podle Algoritmu 4.7 z Opory TIN.

1.2 Formát vstupu

BKG $G = (N, \Sigma, P, S)$ na vstupu odpovídá standardní definici vlastní BKG (viz oporu předmětu TIN). Pro jednoduchost je množina N podmnožinou množiny velkých písmen `[A-Z]` a abeceda Σ je podmnožinou množiny malých písmen `[a-z]`. Dále můžete předpokládat, že vstupní gramatika neobsahuje pravidlo $S \to \varepsilon$ (tedy jazyk, který vlastní gramatika generuje, neobsahuje prázdné slovo). Vstupní textová reprezentace BKG o n pravidlech má následující formát:

seznam všech neterminálů
seznam všech terminálů
počáteční neterminál

pravidlo$_1$
...
pravidlo$_n$

Symboly jsou v seznamech odděleny čárkou. Levá strana pravidla je od pravé oddělena šipkou `->`. Například reprezentace BKG z příkladu 4.1 Opory TIN je následující:

```
S,A,B
a,b,c,d
S
S->AB
A->aAb
A->ab
B->cBd
B->cd
```

1.3 Formát výstupu

Textová reprezentace BKG na výstupu má stejný formát jako BKG na vstupu. Jelikož není možné rozhodnout, zda jsou dvě BKG ekvivalentní, je nutné, aby vaše řešení generovalo výstupy přesně odpovídající výstupům algoritmů 4.5 a 4.7 z opory předmětu TIN. Pořadí pravidel a pořadí symbolů v seznamech na výstupu není důležité, ale označení symbolů, počet a struktura pravidel musí být 100% shodná. Symbol X' se reprezentujte na výstupu jako `X'`, stav $\langle X_1 \dots X_k \rangle$ jako `<X_1...X_k>` , $\langle ABCD \rangle$ jako `<ABCD>`.

2 DKA-2-MKA

Vytvořte program převádějící deterministické konečné automaty (dále DKA) na minimální deterministické konečné automaty (dále MKA) s totální přechodovou funkcí. Použijte Algoritmus 3.5 z opory TIN. Vstupní automat může obsahovat nedosažitelné stavy a součástí algoritmu je jejich odstranění (Algoritmus 3.4).

2.1 Rozhraní programu

Program bude možné spustit:

`flp21-fun` *volby* `[`*vstup*`]`

kde

- *vstup* je jméno vstupního souboru (pokud není specifikováno, program čte standardní vstup) obsahujícího DKA.
- *volba* je parametr ovlivňující chování programu:

  `-i` na standardní výstup se vypíše načtený a do vaší vnitřní reprezentace převedený DKA. Nevypisujte jen načtený řetězec, tato volba ověřuje, že váš program dokáže DKA analyzovat a uložit, a pokud je syntakticky správně, tak také vypsat.

  `-t` na stdout se vypíše MKA.

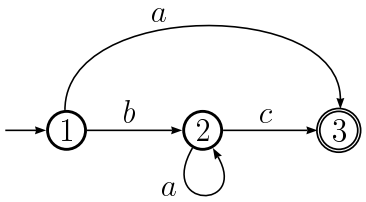
2.2 Formát vstupu

Automat na vstupu odpovídá standardní definici DKA (viz oporu předmětu TIN), ale pro zjednodušení jsou stavy vždy označené nezápornými celými čísly (0, 1, 2, ...) a vstupní abeceda je vždy podmnožinou množiny malých písmen `[a-z]`. Vstupní textová reprezentace automatu o N pravidlech má následující formát:

seznam všech stavů
abeceda
počáteční stav
seznam koncových stavů
pravidlo$_1$
...
pravidlo$_n$

Seznam o M stavech má tvar `stav_1,stav_2,...,stav_M`. Pravidlo pro přechod ze stavu `1` do stavu `2` pomocí symbolu `x` má tvar `1,x,2`.

Například, KA:



bude na vstupu zadán jako:

```
1,2,3
abc
1
3
1,a,3
1,b,2
2,a,2
2,c,3
```

Pozor, vstupní automat nemusí být úplně definovaný (jeho přechodová funkce nemusí být úplná). V krajním případě nemusí být definovaná nikde, tj. vstup neobsahuje žádná pravidla.

2.3 Formát výstupu

Textová reprezentace MKA na výstupu má stejnou syntax jako DKA na vstupu, ale dodržte navíc následující podmínky, zajišťující tzv. *kanonický tvar* minimálního automatu:

- Stavy minimálního automatu číslujte souvisle od nuly a vypisujte je vzestupně seřazené.
- Symboly abecedy seřaďte.
- Vstupní stav bude vždy 0.
- Koncové stavy také seřaďte.
- Pravidla seřaďte lexikograficky podle výchozího stavu (z nějž se přechází) a symbolu abecedy.

- Cílový stav (do nějž se přechází) prvního pravidla může být jen 0 nebo 1. Cílový stav každého dalšího pravidla smí být nanejvýš o 1 větší než největší stav vyskytující se v pravidlech nad ním.

3 DKA-EQ

Vytvořte program rozhodující problém ekvivalence regulárních jazyků pomocí deterministických konečných automatů (dále DKA).

3.1 Rozhraní programu

Program bude možné spustit:

`flp21-fun` *volby* *vstup1* *vstup2*

kde

- *vstup1* je jméno vstupního souboru obsahujícího DKA M_1.
- *vstup2* je jméno vstupního souboru obsahujícího DKA M_2.
- *volby* jsou parametry ovlivňující chování programu:

  `-1` pouze vypíše DKA M_1 na stdout. Nevypisujte jen načtený řetězec, tato volba ověřuje, že váš program dokáže DKA převést do vnitřní reprezentace a znovu vypsat.

  `-2` pouze vypíše DKA M_2 na stdout. Nevypisujte jen načtený řetězec, tato volba ověřuje, že váš program dokáže DKA převést do vnitřní reprezentace a znovu vypsat.

  `-k` vypíše DKA $M = (Q_1 \cup Q_2, \Sigma_1 \cup \Sigma_2, \delta_1 \cup \delta_2, q_0^1, F_1 \cup F_2)$ na stdout. Automat M je vytvořený z M_1 a M_2 podle důkazu věty 3.26 z opory předmětu TIN. Před sjednocením z obou automatů odstraňte nedosažitelné stavy.

  `-d` na stdout se vypíše `True` pokud $L_1(M_1) = L_2(M_2)$, a `False` v opačném případě. Postupujte podle věty 3.26 a jejího důkazu z opory předmětu TIN.

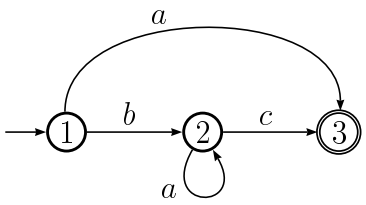
3.2 Formát vstupu a výstupu

Automat na vstupu odpovídá standardní definici DKA (viz oporu předmětu TIN), ale pro zjednodušení jsou stavy vždy označené nezápornými celými čísly (0, 1, 2, ...) a vstupní abeceda je vždy podmnožinou množiny malých písmen `[a-z]`. Vstupní textová reprezentace automatu o N pravidlech má následující formát:

seznam všech stavů
abeceda
počáteční stav
seznam koncových stavů
pravidlo$_1$
...
pravidlo$_n$

Seznam o M stavech má tvar `stav_1,stav_2,...,stav_M`. Pravidlo pro přechod ze stavu `1` do stavu `2` pomocí symbolu `x` má tvar `1,x,2`.

Například, KA:



bude na vstupu zadán jako:

```
1,2,3
abc
1
3
1,a,3
1,b,2
2,a,2
2,c,3
```

Pozor, vstupní automat nemusí být úplně definovaný (jeho přechodová funkce nemusí být úplná). V krajním případě nemusí být definovaná nikde, tj. vstup neobsahuje žádná pravidla.

4 PLG-2-NKA

Vytvořte program, který převádí pravé lineární gramatiky (PLG) bez jednoduchých pravidel (tj. bez pravidel tvaru $A \to B$, v nichž pravou stranu tvoří jediný neterminál) na nedeterministické konečné automaty (NKA).

4.1 Rozhraní programu

Program bude možné spustit:

`flp21-fun` *volby* `[`*vstup*`]`

kde

- *vstup* je jméno vstupního souboru (pokud není specifikováno, program čte standardní vstup) obsahujícího PLG.
- *volby* jsou parametry ovlivňující chování programu:

  `-i` vypíše se načtená a do vaší vnitřní reprezentace převedená PLG na stdout. Nevypisujte jen načtený řetězec, tato volba ověřuje, že váš program dokáže gramatiku přečíst, převést a znovu vypsat. Pokud byla na vstupu validní ale nestandardně zapsaná gramatika, např. se seznamem neterminálů `A,B,A,A`, bude uložena a po volbě `-i` vypsána normálně, tedy s neterminály `A,B`.

  `-1` vypíše se gramatika, která je výsledkem věty 3.2 z opory predmětu TIN, na stdout, s výjimkou pojmenování nových neterminálů. Ty mohou být označeny jménem skládajícím se z velkého písmene následovaného jednou nebo více číslicemi.

  `-2` vypíše se NKA přijímající stejný jazyk, jaký je generován PLG na vstupu. Postupujte podle věty 3.6 z opory předmětu TIN. Stavy automatu označujte nezápornými celými čísly (0, 1, 2, ...).

4.2 Formát vstupu

PLG $G = (N, \Sigma, P, S)$ na vstupu odpovídá standardní definici pravé lineární gramatiky (viz oporu předmětu TIN). Pro zjednodušení je ale abeceda N podmnožinou množiny velkých písmen `[A-Z]`, a abeceda Σ je podmnožinou množiny malých písmen `[a-z]`. Pravá strana tzv. ε-pravidla je tvořena prázdným slovem (ε), na vstupu reprezentovaným znakem `#`. Vstupní textová reprezentace PLG o n pravidlech má následující formát:

seznam všech neterminálů
seznam všech terminálů
počáteční neterminál
pravidlo$_1$
...
pravidlo$_n$

Symboly v seznamech jsou odděleny čárkou. Například, reprezentace PLG z příkladu 2.23 opory předmětu TIN je následující:

```
A,B
a,b,c
A
A->aaB
A->ccB
B->bB
B->#
```

Neterminály, terminály i pravidla tvoří množiny. Pokud se v seznamu neterminálů, terminálů nebo v seznamu pravidel vyskytne stejný prvek víckrát, považujte gramatiku za ekvivalentní gramatice po odstranění duplicit.

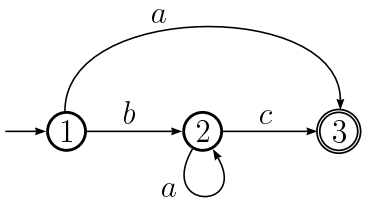
4.3 Formát výstupu

Výstupní textová reprezentace NKA o n pravidlech má následující formát:

seznam všech stavů, očíslovaných $0, 1, \ldots$
abeceda
počáteční stav
seznam koncových stavů
pravidlo$_1$
...
pravidlo$_n$

Symboly v seznamech jsou odděleny čárkou. Pravidlo pro přechod ze stavu `1` do stavu `2` pomocí symbolu `x` má tvar `1,x,2`.

Například, NKA:



bude na výstupu jako:

```
1,2,3
abc
1
3
1,a,3
1,b,2
2,a,2
2,c,3
```

5 RKA-2-DKA

Vytvořte program převádějící rozšířené konečné automaty (dále RKA) na deterministické konečné automaty (dále DKA).

5.1 Rozhraní programu

Program bude možné spustit:

`flp21-fun` *volby* `[`*vstup*`]`

kde

- *vstup* je jméno vstupního souboru (pokud není specifikováno, program čte standardní vstup) obsahujícího RKA.
- *volby* jsou parametry ovlivňující chování programu:
  - `-i` na standardní výstup se vypíše načtený a do vaší vnitřní reprezentace převedený RKA. Nevypisujte jen načtený řetězec; tato volba ověřuje, že váš program dokáže RKA přečíst, převést do vnitřní reprezentace a znovu vypsat.
  - `-t` na stdout se vypíše DKA.

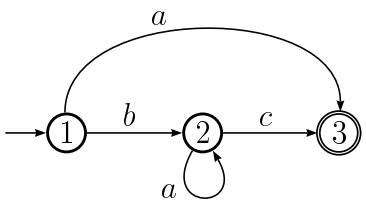
5.2 Formát vstupu

Automat na vstupu odpovídá standardní definici RKA (Def. 3.14 v opoře předmětu TIN), ale pro zjednodušení jsou stavy vždy označené nezápornými celými čísly (0, 1, 2, ...) a vstupní abeceda je vždy podmnožinou množiny malých písmen `[a-z]`. Vstupní textová reprezentace automatu o N pravidlech má následující formát:

seznam všech stavů
abeceda
počáteční stav
seznam koncových stavů
pravidlo$_1$
...
pravidlo$_n$

Seznam o M stavech má tvar `stav_1,stav_2,...,stav_M`. Pravidlo pro přechod ze stavu `1` do stavu `2` pomocí symbolu `x` má tvar `1,x,2`. Epsilon přechod ze stavu `1` do stavu `2` má tvar `1,,2`.

Například, KA:



bude na vstupu zadán jako:

```
1,2,3
abc
1
3
1,a,3
1,b,2
2,a,2
2,c,3
```

5.3 Formát výstupu

Textová reprezentace DKA na výstupu má stejný formát jako RKA na vstupu.

6 RV-2-RKA

Vytvořte program převádějící regulární výrazy (dále RV) na rozšířené konečné automaty (dále RKA). Pro syntaktickou analýzu regulárních výrazů můžete použít knihovnu `Text.Parsec`. Ideu algoritmu převodu čerpejte z Opory TIN, Alg. 3.7.

6.1 Rozhraní programu

Program bude možné spustit:

`flp21-fun` *volby* `[`*vstup*`]`

kde

- *vstup* je jméno vstupního souboru (pokud není specifikováno, program čte standardní vstup) obsahujícího RV.
- *volby* jsou parametry ovlivňující chování programu:

  `-i` dojde pouze k výpisu načteného a do vaší vnitřní reprezentace převedeného RV na stdout. Nevypisujte jen načtený řetězec, tato volba ověřuje, že váš program dokáže RV převést z vnitřní reprezentace.

  `-t` dojde k vypsání RKA na stdout.

6.2 Formát vstupu

RV na vstupu odpovídají standardní definici RV (viz oporu předmětu TIN).

1. Znakový výraz `#` je RV akceptující prázdné slovo.
2. Malé písmeno je RV (akceptující právě toto písmeno).
3. Je-li e RV, pak e`*` je RV (iterace).
4. Jsou-li e_1, e_2 RV, pak $e_1\,e_2$ je RV (zřetězení).
5. Jsou-li e_1, e_2 RV, pak e_1`+`e_2 je RV (sjednocení).
6. Podvýrazy RV lze uzavírat do kulatých závorek. Bez závorek má nejvyšší prioritu iterace, nejnižší sjednocení.
7. RV leží na jednom řádku. Připouštíme v něm mezery.

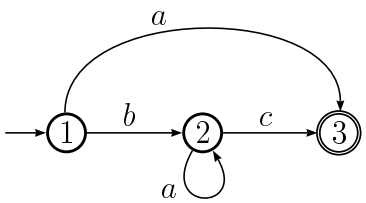
6.3 Formát výstupu

Stavy automatu číslujte souvislou posloupností nezáporných celých čísel (0, 1, 2, ...). Výstupní textová reprezentace automatu o n pravidlech má následující formát:

seznam všech stavů
abeceda
počáteční stav
seznam koncových stavů
pravidlo$_1$
...
pravidlo$_n$

Stavy jsou v seznamu stavů odděleny čárkami, symboly abecedy nikoliv. Předpokládejte, že do abecedy automatu patří jen symboly, které se vyskytují v regulárním výrazu. Pravidlo pro přechod ze stavu `1` do stavu `2` pomocí symbolu `x` má tvar `1,x,2`. Epsilon přechod ze stavu `1` do stavu `2` má tvar `1,,2`.

Například regulárnímu výrazu `a+ba*c` odpovídá konečný automat:



a na výstupu bude jako:

```
1,2,3
abc
1
3
1,a,3
1,b,2
2,a,2
2,c,3
```

7 SIMPLIFY-BKG

Vytvořte program, který odstraňuje zbytečné symboly z bezkontextové gramatiky. Postupujte podle algoritmu 4.3 z opory předmětu TIN, který pracuje ve dvou krocích.

7.1 Rozhraní programu

Program bude možné spustit:

`flp21-fun` *volby* `[`*vstup*`]`

kde

- *vstup* je jméno vstupního souboru (pokud není specifikováno, program čte standardní vstup) obsahujícího BKG.
- *volby* jsou parametry ovlivňující chování programu:
  - `-i` vypíše se pouze načtená a do vaší vnitřní reprezentace převedená BKG na stdout. Nevypisujte jen načtený řetězec, tato volba ověřuje, že váš program dokáže gramatiku převést z vnitřní reprezentace.
  - `-1` vypíše se BKG $\bar{G}$ (po prvním kroku algoritmu 4.3) na stdout.
  - `-2` na stdout se vypíše BKG, která generuje stejný jazyk jako vstupní gramatika, ale neobsahuje žádné zbytečné symboly. Pokud BKG $\bar{G}$ generuje neprázdný jazyk, je vypisovanou gramatikou BKG G' z druhého kroku algoritmu 4.3. Pokud BKG $\bar{G}$ generuje prázdný jazyk, vypište výslednou minimalizovanou gramatiku přímo.

7.2 Formát vstupu

BKG $G = (N, \Sigma, P, S)$ na vstupu odpovídá standardní definici BKG (viz oporu předmětu TIN). Pro zjednodušení je ale abeceda N podmnožinou množiny velkých písmen `[A-Z]`, a abeceda Σ je podmnožinou množiny malých písmen `[a-z]`. Vstupní textová reprezentace BKG o n pravidlech má formát:

seznam všech neterminálů
seznam všech terminálů
počáteční neterminál
pravidlo$_1$

...
pravidlo$_n$

Seznam o m symbolech má tvar `symbol_1,symbol_2,...,symbol_m`. Prázdné slovo je reprezentováno znakem `#`. Epsilon pravidlo má tedy tvar `A->#`.

Například, reprezentace BKG z příkladu 4.1 opory předmětu TIN, s přidaným pravidlem $S \to \varepsilon$, je následující:

```
S,A,B
a,b,c,d
S
S->#
S->AB
A->aAb
A->ab
B->cBd
B->cd
```

7.3 Formát výstupu

Textová reprezentace BKG na výstupu má stejný formát jako BKG na vstupu. V žádném případě ale neměňte označení stavů nebo dokonce sémantiku pravidel. Jen odstraňte zbytečné symboly a pravidla. Ponechané symboly a pravidla musí zůstat stejná, jak byla zadána.

8 FORD-FULKERSON

Vytvořte program počítající maximální tok v síti pomocí algoritmu Ford–Fulkerson.

8.1 Rozhraní programu

Program bude možné spustit:

`flp21-fun` *volby* [*vstup*]

kde

- *vstup* je jméno vstupního souboru (pokud není specifikováno, program čte standardní vstup) obsahujícího síť.
- *volby* jsou parametry ovlivňující chování programu:
  - `-i` na standardní výstup se vypíše síť, která byla předtím načtena a převedena do vaší vnitřní reprezentace. Nevypisujte jen načtený řetězec, tato volba ověřuje, že váš program dokáže síť převést do vnitřní reprezentace a zpět.
  - `-f` na stdout se vypíše jen velikost maximálního toku v síti (tj. výstupem je jedno číslo).
  - `-v` vypíše tok celou sítí.

8.2 Formát vstupu

Síť je na vstupu ve formátu „DIMACS max-flow“, `http://lpsolve.sourceforge.net/5.5/DIMACS_maxf.htm`.

8.3 Formát výstupu

Síť s tokem bude na výstupu ve formátu „DIMACS max-flow“, řádky s tokem hranami začínají písmenem `f`. Výstup nemusí obsahovat komentářové řádky. Vypisujte i hrany s nulovým tokem, pokud byly na vstupu s nenulovou kapacitou.

9 CUSTOM

V případě zájmu o samostatné zadání dle vašich preferencí a specializace prosím kontaktujte cvičícího. Na základě dohody lze vypsat prakticky libovolné (ale vhodně obtížné) zadání na míru. Projekt může být součástí vaší diplomové práce, projektu do jiného předmětu, zaměstnání nebo zájmu. Jedinou podmínkou je, že musí být vypracován v jazyce Haskell.

Jelikož však nebude možné takové zadání automaticky testovat spolu s ostatními, očekávejte své zapojení při prokazování funkčnosti řešení. Konkrétní forma bude záviset na dohodnutém tématu, ale může jít o osobní předvedení nebo vypracování a odevzdání sady automatizovaných testů.