PŘÍRODOVĚDECKÁ FAKULTA UNIVERZITY PALACKÉHO
KATEDRA INFORMATIKY

BAKALÁŘSKÁ PRÁCE

Skriptovací jazyk pro programování pokerbotů založených na pravidlech

2013 Radek Janoštík

Anotace

Naprogramovat kvalitního počítačového hráče deskových a karetních her je výzvou pro většinu programátorů. V této práci jsem navrhl programovací jazyk nazvaný Heads Up Script, který umožní vytvořit počítačového hráče pokeru(pokerbota) nejen programátorům, ale také schopným lidským hráčům pokeru. Programátor čitelnou formou specifikuje sadu pravidel, která udává, jak se má pokerbot zachovat v určitých herních situacích. Heads Up Script poskytuje mnoho konstrukcí tak, aby měl dostatečnou vyjadřovací schopnost pro definici pravidel. Heads Up Script je zaměřen na pokerovou herní variantu Heads Up, logicky kopíruje členění hry. Součástí práce je i funkční implementace jazyka Heads Up Script.

Děkuji vedoucímu mé práce Mgr. Janu Konečnému, Ph.D. za vypsání tématu a čas strávený konzultacemi, rodině za gramatickou korekturu a podporu.

Obsah

Seznam obrázků

Seznam tabulek

1. Úvod

Umělá inteligence byla předmětem studií již od zrodu informatiky a byla i jedním z hnacích motorů vývoje nových a rychlejších počítačů. Nedílnou součástí této disciplíny byla snaha vytvořit počítačové hráče k různým deskovým hrám, kteří by byli schopni porazit lidské hráče. Za zmínku stojí souboj tehdejšího šachového velmistra Garryho Kasparova s počítačem Deep Blue v roce 1997. [1]

Snaha porazit člověka počítačem byla i v karetních „hazardních" hrách, kde motivací nebyla jen nehynoucí sláva, ale i vidina zisku peněz. Není divu, že jakmile začali být počítačoví karetní hráči srovnatelní s lidskými, začali používat rozsáhlé databáze herních situací a klasifikaci protihráčů, tak většina internetových kasin začala detekovat tyto hráče a následně jim zakazovat hru či zabavovat finanční prostředky.

V následujících podkapitolách proberu problematiku řešení vybraných deskových a karetních her. Při popisu používám následující pojmy:

Stavová složitost hry – udává celkový počet unikátních kombinací pozic na herní desce.

Složitost herního stromu – udává celkové množství unikátních herních kombinací, které mohou být hrány z počátečního stavu hry, nebo-li celkový počet listů herního stromu při úplném prozkoumání.

1.1. Hry s úplnou znalostí

Do této množiny her patří ty, při kterých oba hráči mají stejnou znalost hry. Například oba hráči vidí všechny figurky na šachovnici, všechny dosud provedené tahy apod. Hru považujeme za *vyřešenou*, pokud jsme schopni předpovědět výsledek z každého stavu hry za předpokladu, že oba hráči hrají tzv. *perfektní hru*. V opačném případě hru považujeme za *nevyřešenou*.

Ke snadno (počítačově) hratelným hrám patří ty, ke kterým existují poměrně jednoduché algoritmy, jak vyhrát nebo alespoň zabezpečit remízu nebo hry s nízkou stavovou složitostí. Typickým příkladem jsou piškvorky, kde můžeme taktiku shrnout do malého množství vět a naprogramování hráče nezabere mnoho času. Dále sem patří hra dáma, známa též jako *checkers*, která má poměrně vysokou stavovou složitost, ale jednoduché algoritmy na bázi *minimax* ji jsou schopny obstojně hrát. Jedná se o hru s nejvyšší stavovou složitostí, která byla *vyřešena*. [2]

Naopak obtížně hratelné jsou ty, které dosud nebyly vyřešeny a mají tak vysokou stavovou složitost, že algoritmy na bázi *minimax* již na kvalitní hru nestačí. Pro hraní se často používá algoritmů založených na metodě *Monte Carlo*, používajících databáze počátečních tahů a koncovek nebo obojího v kombinaci s *Minimax*. Patří sem například Šachy, Go nebo Reversi. Pro srovnání uvádím tabulku 1. [3] stavových složitostí a řešitelnosti her.

Název hry	Stavová složitost	Složitost herního stromu	Vyřešena
Awari	10^{12}	10^{32}	Ano
Dáma	10^{21}	10^{31}	Ano
Šachy	10^{46}	10^{123}	Některé slabší varianty
Go (19×19)	10^{172}	10^{360}	Do velikosti 5×5

Tabulka 1. Přehled složitostí vybraných her a stav jejich řešitelnosti.

Název hry	Počet počátečních stavů
Prší (2 hráči)	$3.986646 \cdot 10^{12}$
Poker Texas Hold'em (2 hráči)	1 624 350
Poker Texas Hold'em (8 hráčů)	$8.469802 \cdot 10^{23}$
Poker Omaha Hold'em (8 hráčů)	$3.011853 \cdot 10^{38}$

Tabulka 2. Přehled možných počátečních stavů vybraných her.

1.2. Hry s neúplnou znalostí

Oproti předchozí kategorii, v těchto hrách je část znalosti o hře druhému hráči zatajena – většinou se jedná o hráčovy karty, skrytý hrací kámen nebo typ hráčovy postavy. Tato skutečnost znesnadňuje předpovědět protihráčův tah. Příslušné hry se vyznačují velkým množstvím počátečních stavů, to ale neznamená, že nejde vytvořit schopného počítačového hráče. Například známá karetní hra Prší má sice ve dvou hráčích $3.986646 \cdot 10^{12}$ počátečních kombinací, ale i přesto lze sestrojit hráče, který bude hrát stylem: „Zbavuj se karet, kterých můžeš. Měň na barvu, kterou máš nejvícekrát. Má-li protihráč málo karet, pokus se zbavit bojové karty.“ Troufám si konstatovat, že takový hráč bude hrát stejně dobře, jako většina jiných lidských hráčů.

Uvažujme nyní karetní hru *Poker Texas Hold'em Heads-Up*. Tedy dvoukartovou variantu pokeru, kde hrají pouze dva hráči. Tato varianta má 1 624 350 počátečních kombinací. (Více viz tabulka 2.) Odmyslíme-li sázení po každém kole, herní strom bude mít velikost $\approx 10^{18}$ uzlů. Avšak právě ono odmyšlené vsázení má ve hře zásadní roli, protože nevíme, jak na naši sázku zareaguje soupeř a navíc vsázet je teoreticky možné až do vyčerpání peněz. Pro tuto skutečnost se tvorba počítačového hráče pokeru (pokerbota) stává netriviální záležitostí a je to také mou hlavní motivací pro tvorbu programovacího jazyka, který usnadní programování pokerbota.

2. Metody konstrukce pokerbotů

Pokerbotem rozumíme počítačový program, který na základě dostatečné znalosti o hře dokáže rozhodnout, jak v dané situaci hrát. Příkladem může být třeba náhodný hráč, ale i komplexní programy s prvky umělé inteligence nebo provádějící složité simulace.

První zmínky o studii pokerbotů sahají do 80. let minulého století [4]. Prudký nárůst zájmu zkoumání možností pokerbotů přišel na přelomu tisíciletí díky rozšíření on-line pokeru. Většina vědeckých článků pochází z této doby. Od roku 2006 se pravidelně pořádá turnaj *The Annual Computer Poker Competition* [5], kde spolu soupeří pokerboti. Za zmínku stojí i turnaj *Man-Machine Poker Championship*, kde soupeří pokerboti proti lidským hráčům.

Programování složitých pokerbotů je pro svou náročnost určeno spíše programátorům, což valná většina pokerových hráčů není. Proto vzniklo několik nástrojů, jak programování usnadnit. Jedním z nich je skriptovací jazyk *OH-Script*, který je součástí *OpenHoldem Project* [6]. Tento skriptovací jazyk je stavěn na hru v deseti hráčích a je velmi obecný. Nevýhodou pro pokerové hráče je velká podobnost s jazykem C a přílišná „nízkoúrovňovost“. Zajímavou alternativou může být komerční *Poker Programming Language* [7] od firmy Shanky Technologies, který umožňuje vytvářet profily definované pravidly (viz. 2.1.1.) pro pokerboty od stejné firmy, nezaměřuje se pouze na variantu *Heads Up*, a proto je poměrně komplikovaný.

Existuje několik základních metod konstrukce pokerbotů. Ty proberu ve zbytku kapitoly.

2.1. Znalostní systémy

Pokerbot na základě předem definované bázi znalostí rozhodne, jak bude hrát. Bázi znalostí musí nadefinovat expert, což v našem případě může být zkušený hráč pokeru. Rozlišujeme dva základní způsoby zadávání znalostí.

2.1.1. Systémy definované pravidly

Jednoduchý znalostní systém je určen sadou pravidel typu *if-then*(pokudpak), která pokryjí většinu možných situací, které mohou nastat. Při vyhodnocení systém jednoduše zkoumá, zda byla splněna podmínka a případně se provede kód v *then*-sekci.

U pokerbotů bývají podmínky často netriviální a výstup bývá pravděpodobnostní trojice (*fold*, *call*, *raise*), tedy s jakou pravděpodobností má pokerbot vykonat jakou akci. Pokerbot, vytvořený jazykem navrženým v této práci, je sice založen na pravidlech, ale od pravděpodobnostní trojice jsem upustil, protože nezachycuje, jaká částka se má zvyšovat.

Efektivitu vyhodnocení lze zvýšit tzv. *líným vyhodnocováním*, tzn. pokud systém narazí na první splněné pravidlo, aplikuje jej a skončí. V nejhorším případě se tedy vyhodnotí všechna pravidla. Kvalitu výsledku lze ovlivnit zavedením vah u pravidel nebo fuzzy přístupem.

2.1.2. Systémy definované formulemi

Jedná se o obecnější přístup než předchozí. Pokerbot je určen formulemi, které udávají způsob, jak získat další veličinu z herní situace a na základě výsledků formulí rozhodne, jak bude hrát. Typickou počítanou veličinou bývá číselná reprezentace síly karty nebo *pot odds*.

Jednou z číselných reprezentací karet je *immediate hand rank*(IHR), tedy okamžitá síla karet. Vypočítá se průchodem přes všechny možné kombinace, které může protihráč mít a zaznamená se, zda bychom vyhráli, prohráli nebo remizovali. Výpočet je poměrně rychlý, protože možných kombinací je po *flopu* $\binom{47}{2} = 1081$, po *turnu* $\binom{46}{2} = 1035$ a po *riveru* $\binom{45}{2} = 990$. IHR udává pravděpodobnost na výhru v aktuální situaci, nezohledňuje ale fakt, že protihráč většinou špatné kombinace pokládá již na začátku hry. IHR se vypočítá pomocí následujícího vzorce:

$$IHR = \frac{vyhry + 0.5 \cdot remizy}{vyhry + remizy + prohry}$$

Dále se často počítá *pot odds*, což je číslo, které udává návrat vkladu oproti vkladu, který musí hráč vložit, aby zůstal ve hře. Vypočítá se pomocí vzorce:

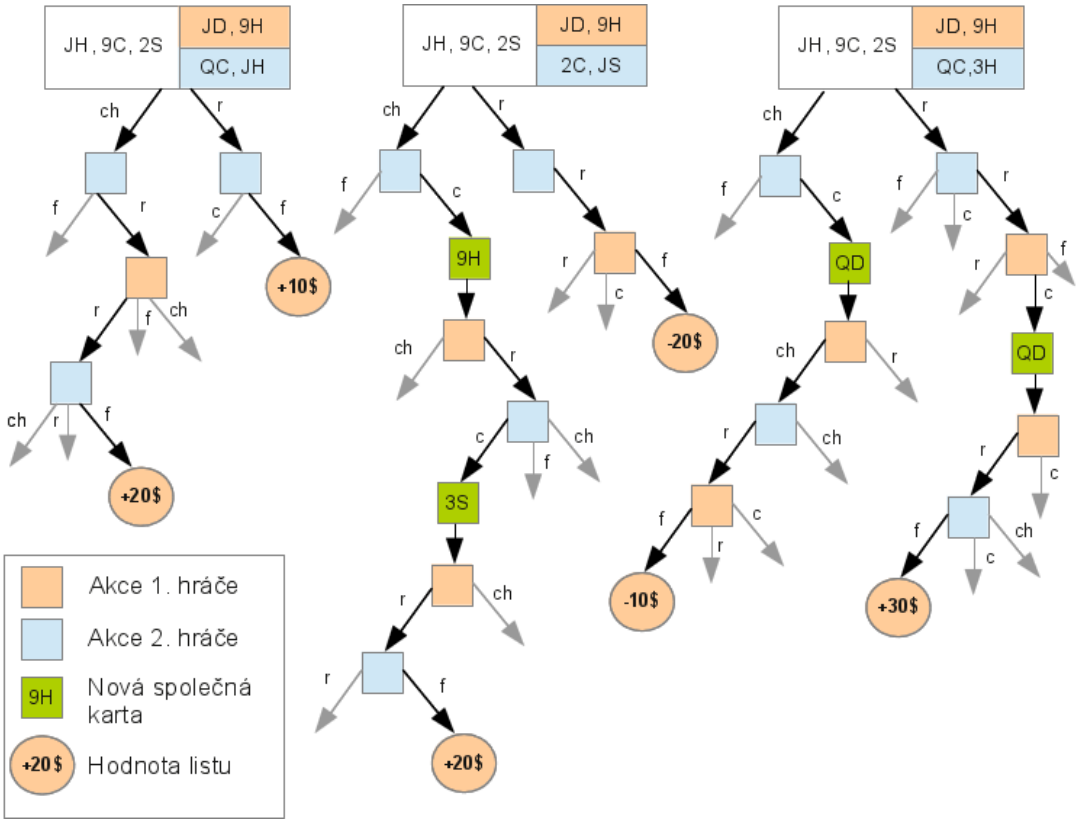
$$potOdds = \frac{c}{p + c}$$

kde c je obnos potřebný k zůstání ve hře a p je velikost *potu*. Mějme situaci, kdy v *potu* je 40$ a musíme dát 10$, abychom zůstali ve hře. *Pot odds* je v tomto případě 0.2. Což znamená, že musíme mít šanci na výhru alespoň 20%, abychom navýšení mohli akceptovat.

2.2. Simulační pokerboti

Simulační pokerboti prochází herní stavový prostor (herní strom) a přiřazují „koncovým uzlům“ hodnotu, která rozhodne, jak bude pokerbot hrát. Patřili by sem i pokerboti používající algoritmy *minimax* a *alfa-beta ořezávání*, pro které je však limitující možná hloubka průchodu – tyto algoritmy jsou použitelné u her s úplnou znalostí. U pokeru tedy selhávají, protože nelze jednoznačně určit stav hry.

Alternativní strategii průchodu stromem nabízí metoda *Monte Carlo*. Ta spočívá v náhodném průchodu herním stromem až do listů stromu, kde můžeme jednoznačně určit hodnotu uzlu. Většinou se jako hodnocení uzlu používá peněžní



Obrázek 1. Příklad simulace hry.

výnos. Pro ilustraci průběhu simulace herním stromem přikládám obrázek, kde jsou zobrazeny tři možné simulace, viz 1.

Aby byl konečný výsledek simulace co nejlepší, musíme provést co nejvíce simulací. Je běžné, že pokerboti založeni na simulaci provádí v každé fázi hry miliony až stamiliony simulací. Proto je také kladen velký důraz na efektivitu ohodnocovacího algoritmu, kde se cení každá milisekunda. Je známo, že podpůrné pokerové programy si chrání své algoritmy, a firmy jsou dokonce ochotné tvůrce nových algoritmů solidně zaplatit. Případem může být třeba *Cactus Kev's Poker Hand Evaluator* [8].

2.3. Pokerboti založení na teorii her

Pokerboti využívají přístup známý z teorie her – snaží se nalézt *vyváženou hru*, známou jako *Nashovo equilibrium*. Což je ekvivalentní s nalezením ideální kombinace pravděpodobností změny herní strategie. Například u jednoduché hry jako kámen-nůžky-papír se jedná o vektor $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$, tedy že máme rovnoměrně měnit strategii, abychom při dlouhodobém hraní dosáhli alespoň remízy.

U složitých her, jako je poker, je nalezení *vyvážené hry* v reálném čase prakticky nemožné, proto se musí použít abstrakcí k redukci herního stromu. Roz-

lišujeme bezztrátové abstrakce, tedy ty, kvůli kterým nepřijdeme o strategicky důležité informace a ztrátové abstrakce, které sice redukují herní strom, ale na úkor ztráty informací.

Nejpoužívanější bezztrátovou abstrakcí je shlukování karetních kombinací na základě toho, zda mají stejnou barvu, či nikoliv. Například ve fázi *preflop* vůbec nerozlišujeme rozdíl mezi $Q\spadesuit J\spadesuit$ a $Q\clubsuit J\clubsuit$, protože ze strategického hlediska bychom s těmito kombinacemi měli hrát stejně. Tím omezíme počet počátečních kombinací z 1 624 350 na 28 561. Tuto abstrakci však již nelze využít v dalších fázích, protože barva karet již hraje roli ve výpočtech síly karet.

Často používanou ztrátovou abstrakcí je tzv. *bucketing*, což je sdružování karetních kombinací do ekvivalentních tříd. Jedna z ekvivalencí může být založena například na pravděpodobnosti výhry proti náhodné kombinaci. Musíme však tyto kombinace simulovat a zjišťovat procenta výher. Třídami ekvivalence pak budou disjunktní intervaly z $[0, 1]$. Další ztrátovou abstrakcí je omezení počtu sázek v sázecím kole nebo vynechání poslední fáze hry.

3. Referenční příručka jazyka Heads Up Script

3.1. Cíle jazyka

Hlavní cíl jazyka Heads Up Script je umožnit naprogramovat plnohodnotného pokerbota založeného na pravidlech. Nejprve bych zmínil vlastnosti, na které jsem velký důraz nekladl:

Mimoherní projevy protihráče – mimika, gesta a verbální projev jistě k pokeru patří a často bývá velmi důležité nedat na sobě nic znát. Většinu těchto projevů eliminuje on-line poker. Mohli bychom třeba zaznamenávat délku přemýšlení či jiné měřitelné neherní projevy, ale i ty mohou být velmi zavádějící, proto nejsou v jazyce nijak podchyceny.

Odhad strategie protihráče bývá pro profesionály to nejdůležitější. Autoři knih o pokeru mu věnují rozsáhlé kapitoly a pokud máme dostatečně velkou databázi her daného hráče, je možné předvídat, jak by asi mohl hrát. Pokerbot, který vznikne z jazyka Heads Up Script, je stavěn na to, aby rozhodl pouze na základě údajů z jedné hry. Profilace hráče je možná pouze pomocí volání externího programu.

Efektivita, rychlost operací je stěžejní u pokerbotů provádějících velké množství simulací. V mém případě se vyhodnocují pouze pravidla pro aktuální fázi, kterých sice může být mnoho, ale ne tolik, aby záleželo na každé ušetřené milisekundě.

Naopak následující vlastnosti jsem se snažil upřednostnit na úkor předchozích.

Vyjadřovací schopnost jazyka musí být dostatečná, abychom mohli přesně vyjádřit, jak má bot v daných situacích hrát. Předlohy pravidel jsem čerpal z [9], [10] a z internetových fór. Heads Up Script umožňuje definovat reakce na specifické karty, rozsahy karet, ale i hru protihráče.

Blízkost definice pravidel přirozenému jazyku je vhodná pro většinu pokerových hráčů, kteří neumí vůbec programovat, ale chtěli by si vytvořit vlastního pokerbota. Proto je většina prvků jazyka Heads Up Script tvořena slovy z pokerového slangu.

Nenáročná syntaxe úzce souvisí s předchozím bodem. Heads Up Script neobsahuje žádné speciální znaky, závorky či apostrofy. Pouze rozlišuje malá a velká písmena, což složí k odlišení konstrukcí jazyka od pravdivostních výrazů a akcí.

3.2. Struktura jazyka

Struktura jazyka kopíruje průběh hry pokeru. Pro upřesnění pravidel nahlédněte do přílohy A.

Hraní v každé fázi hry se natolik liší, že je vhodné definovat pravidla pro každou fázi jednotlivě. Kód je proto rozdělen na 4 části odpovídající fázím hry a fázi *ALWAYS*, ve které se pravidla vyhodnocují vždy, nezávisle na fázi hry. Kdybychom chtěli dané pravidlo použít ve více fázích, museli bychom jej zapsat i do další fáze. Dochází tak sice k redundanci pravidel, ale nepočítám s tím, že by se totožná pravidla hojně vyskytovala ve více fázích.

V pravidlech se nejprve definují podmínky, za kterých se má pravidlo aplikovat, poté následuje definice samotné akce. Dále je možné pro každé pravidlo nadefinovat neomezené množství reakcí na předchozí soupeřovu akci.

3.3. Popis struktury jazyka

V Literatuře se často používá označení *Hero* pro našeho hráče a označení *Villain* pro soupeře. Tohoto označení se držím i v navrženém jazyce. Karty 2-9 jsou reprezentovány svou číselnou hodnotou. Karty 10, kluk, dáma, král a eso jsou po řadě reprezentovány písmeny T, J, Q, K, A.

3.3.1. Fáze hry

Klíčová slova pro označení úseku pravidel pro danou fázi jsou psána velkými písmeny a jsou shodná s názvy fází, tedy `PREFLOP, FLOP, TURN` a `RIVER`. Každý takto započatý úsek musí být ukončen stejným klíčovým slovem s prefixem `END`, tedy `ENDPREFLOP, ..., ENDRIVER`. Dále jsou k dispozici klíčová slova `ALWAYS, ENDALWAYS`, která vymezují pravidla, jež vyhodnotí vždy nezávisle na fázi hry.

Pořadí úseků není určeno, je také dovoleno mít více úseků pro stejnou fázi, avšak to z důvodů větší přehlednosti kódu nedoporučuji.

Příklad rozlišení fází hry:

```
PREFLOP
  pravidlo1
  pravidlo2
ENDPREFLOP
FLOP
  previdlo3
  pravidlo4
ENDFLOP
```

3.3.2. Definice pravidel

Jednotlivá pravidla jsou vymezena klíčovými slovy `RULE` a `ENDRULE`. Poté následuje definice pravdivostních výrazů, při jejichž platnosti se má dané pravidlo aplikovat.

Pravdivostní výrazy zapsané na jednom řádku oddělené čárkami se vyhodnocují ve smyslu „a zároveň“, odřádkujeme-li, vznikne nová větev. Jednotlivé větve se vyhodnocují ve smyslu „nebo“. Odřádkování zde tedy tvoří syntaktický prvek. Pravidlo se tedy aplikuje, pokud budou pravdivé všechny pravdivostní výrazy alespoň v jedné větvi.

Po poslední větvi následuje klíčové slovo `THEN` a akce, která se má provést. Viz 3.3.5.

Dále může následovat nepovinná sekce s definicí reakcí na protihráčovu hru. Sekce začíná klíčovým slovem `VILLAIN`, následuje obdobná definice pravdivostních výrazů jako u pravidla, poté klíčové slovo `THEN` a akce (viz 3.3.5.). Vše je ukončeno klíčovým slovem `ENDVILLAIN`.

K vyhodnocení reakcí na protihráče dochází pouze v případě, že se dané pravidlo aplikuje a protihráč hrál v dané fázi dříve než my. Pokud protihráč hrál před námi a v aplikovaném pravidle není definována reakce na protihráče nebo nebyla splněna žádná podmínka u reakce, neprovede se žádná akce z tohoto pravidla.

Jsou-li splněny podmínky u více pravidel současně, program o tom uživatele informuje a náhodně vybere jedno z platných pravidel. Náhodnost je v tomto případě výhodou, protože znesnadňuje protihráčům zjištění pokerbotové strategie. Výběr výsledné akce by mohl být nahrazen „kombinátorem akcí“, který by dokázal splněná pravidla vhodně zkombinovat do jedné akce. Otázkou je, jak by se tento „kombinátor“ měl zachovat například pro dvě protikladné akce.

Příklad definice pravidla:

```
RULE
  pravdivostniVyraz1, pravdivostniVyraz2
  pravdivostniVyraz3, pravdivostniVyraz4, pravdivostniVyraz5
    THEN akce
  VILLAIN
     pravdivostniVyraz6, pravdivostniVyraz7
    THEN AKCE
  ENDVILLAIN
ENDRULE
```

3.3.3. Konstanty

V jazyce je k dispozici několik slovních pojmenování, představujících číselnou hodnotu charakteristiky. Konstanty se na svou hodnotu vyhodnocují již při syntaktické analýze, což zjednodušuje vyhodnocení složitějších výrazů. Významy jednotlivých konstant nejlépe zachycuje tabulka 3.

Jméno konstanty	Význam
pot	součet všech sázek v herním kole
bb nebo BB	hodnota *velkého blindu*
sb nebo SB	hodnota *malého blindu*
effectiveStack	obnos peněz chudšího hráče
heroStack	náš obnos peněz
villainStack	protihráčův obnos peněz
handRank2652	síla karet podle [6]
handRank169	viz předchozí
allIn	viz heroStack
toCall	velikost sázky, abychom mohli pokračovat
villainPreflopRaise	součet protihráčových sázek v první fázi
villainPreflopRaiseTimes	počet protihráčových navyšování v první fázi

Tabulka 3. Přehled dostupných konstant a jejich významy.

Ke konstantě *villainPreflopRaise* existují ekvivalenty pro všechny ostatní fáze – *villainFlopRaise* apod. Ke konstantě *villainPreflopRaiseTimes* existují ekvivalenty pro ostatní fáze a akce. Např.: *villainTurnCheckTimes, villainRiverCallTimes.*

3.3.4. Pravdivostní výrazy

Jedná se o ty výrazy, které se vyhodnotí buď na *pravdu* nebo na *nepravdu.* Heads Up Script podporuje tři typy pravdivostních výrazů:

Dotazy na karty sledují, zda dvojice karet našeho hráče splňuje danou podmínku. Přehled a ukázku výrazů zachycuje tabulka 4. Není-li uvedeno písmeno „s“ (*suited*) nebo „o“ (*offsuit*), implicitně se použije „o“. Navíc je zřejmé, že u dotazu na pár se vždy použije „o“.

Predikáty umožňují dotazy, nastala-li určitá herní situace. Např.: Držíme-li pár, chybí-li nám pouze jedna karta na postupku nebo zda jsou společné karty odlišných barev. Výčet predikátů naleznete v tabulce 5. Jako ve známých programovacích jazycích lze použít unární operátor „!“ před jménem predikátu pro negaci predikátu.

Číselné porovnávání se zapisuje standardně ve tvaru: *číslo komparátor číslo*, kde výraz *číslo* může být konstanta, násobek konstanty, numerická hodnota nebo výsledek dotazu do historie, který vrací číselnou hodnotu. *Kompará-*

Tvar výrazu	Příklad	Význam
(karta1)(karta2)	T5	Držíme-li karty 10 a 5 v lib. barvě.
(karta1)(karta2)s	Q8s	Držíme-li dámu a osmu ve stejné barvě.
(karta1)(karta2)s+	Q8s+	Držíme-li dámu a osmu nebo vyšší ve stejné barvě.
(karta1)(karta2)o-	Q8s-	Držíme-li dámu a osmu nebo nižší v lib. barvě.
(karta1)(karta1)	KK	Držíme-li daný pár.
(karta1)(karta1)	JJ+	Držíme-li pár kluků nebo vyšší pár.
(karta1)(karta1)	JJ-	Držíme-li pár kluků nebo nižší pár.

Tabulka 4. Přehled dotazů na karty a jejich významy.

tory jsou symboly `>`, `<`, `>=`, `<=` pro porovnání, `==` pro rovnost a `!=` pro nerovnost.

3.3.5. Akce

Výrazy typu akce specifikují, jak má náš hráč v konkrétním případě hrát. Heads Up Script podporuje všechny akce: *FOLD, CHECK, CALL, RAISE* po řadě pro ukončení hry, nenavyšování, souhlas s protihráčovým navýšením a pro navýšení sázky. U navyšování musí být uvedeno číslo, které specifikuje výši částky.

Heads Up Script umožňuje zadat i více hodnot pro navýšení, u každé však musí být uvedena číselná hodnota „pravděpodobnosti“, s jakou se má vsadit daná částka. Program poté náhodně vybere částku pro vsazení s ohledem na váhu. Váhy mohou být libovolné číslo, pro přehlednost však doporučuji, aby bylo v intervalu $(0, 1]$ a součet vah byl 1, není to však podmínkou. Jednotlivé dvojice váha a částka jsou odděleny čárkami.

Příklad použití akcí:

```
THEN FOLD
THEN RAISE 3BB
THEN RAISE 0.25 2BB, 0.5 4BB, 0.25 8BB
THEN RAISE 1 2BB, 3 3BB, 5 6BB
```

3.3.6. Volání externího programu

Heads Up Script umožňuje použít externí program, jehož výstup je ve specifickém tvaru, který je upřesněn v závěru této podsekce. To nám umožňuje použít jakýkoliv již hotový podpůrný program, jako například ohodnocovač karetních kombinací. Volat externí program je povoleno z podmínek pravidel a části, kde se očekává akce.

Název predikátu	**Význam**
air	Nemáme-li vůbec žádnou ohodnocenou kombinaci.
rainbow	Společné karty jsou ve více jak třech barvách.
highPair	Máme-li vysoký pár, tedy kluky až esa.
midPair	Máme-li střední pár, tedy 10–6.
lowPair	Máme-li nízký pár, tedy 5–2
pair	Máme-li jakýkoliv pár.
twoPair	Máme-li dva páry.
threeOfKind	Máme-li trojici stejných karet.
straight	Máme-li postupku v jakýchkoliv barvách.
flush	Máme-li 5 karet ve stejné barvě.
fullHouse	Máme-li *full house*, tedy pár a trojici.
poker	Máme-li 4 stejné karty.
straightFlush	Máme-li postupku ve stejné barvě.
royalFlush	Máme-li postupku 10-A ve stejné barvě.
flushDraw	Máme-li 4 karty ve stejné barvě.
straightDraw	Chybí-li nám do postupky pouze 1 karta.
openEnded	Máme-li postupku délky 4.
desk3SameColor	Jsou-li 3 společné karty se stejnou barvou.
deskStraightDraw	K postupce ve společných kartách chybí jen jedna karta.
deskOpenEnded	Společné karty obsahují postupku délky 4.
deskHighPair	Společné karty obsahují vysoký pár.
deskMidPair	Společné karty obsahují střední pár.
deskLowPair	Společné karty obsahují nízký pár.
deskPair	Společné karty obsahují jakýkoliv pár.
deskThreeOfKind	Společné karty obsahují trojici stejných karet.
deskTwoPair	Společné karty obsahují dva páry.

Tabulka 5. Přehled predikátů a jejich významy.

Volání externího programu je vymezeno klíčovým slovem *external* následované samotným voláním v uvozovkách tak, jako by byl program spouštěn z příkazové řádky/shellu. Volání programu může obsahovat konstanty a speciální výrazy zachycující reprezentaci hry, ty budou textově nahrazeny svou hodnotou, což umožní předat programu všechny potřebné parametry. Speciálními výrazy jsou:

heroCards – vrátí dvojici karet, kterou má náš hráč. Karty jsou reprezentované svou textovou hodnotou a barvou (První písmeno anglického názvu barvy). Například 8H 10D značí osmu srdcovou a desítku károvou.

flopCards – vrátí první tři společné karty.

turnCard – vrátí 4. společnou kartu.

riverCard – vrátí poslední společnou kartu.

communityCards – vrátí všechny společné karty.

wholeGame – vrátí celou reprezentaci hry ve stejné podobě, v jaké ji dostal pokerbot. Tedy obsah celéhé XML souboru.

Je-li volání externího programu použito v podmínkách pravidla, očekává se, že první řádek výstupu externího programu bude obsahovat číslo, které bude bráno jako návratová hodnota. V mé implementaci se hledá první číslo na prvním řádku.

Je-li externí program volán z části za *THEN*, tak výstup volaného programu musí být jednořádkový a musí obsahovat pouze název akce. Je-li danou akcí *raise*, musí název akce následovat číslo oddělené mezerou.

3.3.7. Použití tabulky chování

V první fázi hry se často definují akce přímo podle dvojice našich karet. Jazyk proto umožňuje zadat akci pomocí tabulky podobné s tabulkou, jaká je použita v [10]. Tu jsem mírně upravil a může vypadat jako na obrázku 2. V části tabulky nad hlavní diagonálou jsou uvedeny akce, které se mají provést za předpokladu, že daná karetní kombinace má stejnou barvu. Pod diagonálou včetně, pokud karty mají barvu odlišnou.

Tabulka akcí se zadává pomocí klíčových slov *USE TABLE*, za kterými musí být uvedena cesta k textovému souboru s tabulkou. Textový soubor musí mít alespoň 13 řádků a 13 sloupců. Sloupce musí být odděleny tabulátorem. Je-li v souboru více řádků nebo sloupců, jsou ignorovány.

3.4. Příklad

	Suited											
Offsuit	A	K	Q	J	T	9	8	7	6	5	4	3
A	R5.5	R4.0	R3.0	R2.0	C	C	C	C	C	C	C	C
K	R3.0	R4.0	R3.0	R1.0	C	C	C	C	C	C	C	C
Q	R2.5	R1.0	R3.0	R2.0	C	C	C	C	C	C	C	C
J	R1.5	R1.0	C	R1.5	C	C	C	C	C	C	C	C
T	C	C	C	C	R1.0	C	C	C	C	C	C	C
9	C	C	C	C	C	C	C	C	C	C	C	C
8	C	C	C	C	C	F	C	C	C	C	C	C
7	C	C	C	C	F	F	F	C	C	C	C	C
6	C	C	C	F	F	F	F	F	C	C	C	C
5	C	C	F	F	F	F	F	F	F	C	C	C
4	C	C	F	F	F	F	F	F	F	F	C	F
3	C	C	F	F	F	F	F	F	F	F	F	C
2	C	F	F	F	F	F	F	F	F	F	F	F

Obrázek 2. Tabulka akcí.

3.4.1. Herní situace 1

Nacházíme se v herní fázi *preflop*, vlastníme 1000$, náš protihráč pouze 200$. Velikost *big blindu*(BB) je 20$, *small blindu*(SB) 10$. Bylo nám rozdáno Q♡, T♠, jsme na pozici *small blindu*, tedy začínáme. Popis pravidla v přirozeném jazyce zní: „Máte-li více jak 3krát více peněz než soupeř a dojdou vám vyšší karty nebo pár vyšší než 6, navyšte 3 nebo 6 násobek *BB*. Pokud je na tahu protihráč a nezvedal více než $4BB$, zvyšte sázku o 2BB, zvedal-li mezi 4BB a 6BB souhlaste, jinak položte.“

Takto definované pravidlo bychom do jazyka Heads Up Script přepsali takto:

```
RULE heroStack >= 3 villainStack, JT+
     heroStack >= 3 villainStack, 77+
  THEN RAISE 0.5 3BB, 0.5 6BB
  VILLAIN villainRaise <= 4BB
    THEN RAISE 2BB
  ENDVILLAIN
  VILLAIN villainRaise >= 4BB, villainRaise <=6BB
```

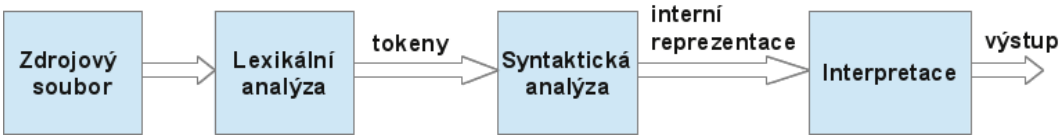
```
    THEN CALL
  ENDVILLAIN
  VILLAIN villainRaise > 6BB
    THEN FOLD
  ENDVILLAIN
ENDRULE
```

3.4.2. Herní situace 2

Nacházíme se v herní fázi *flop*, vlastníme 750$, náš protihráč 1500$. *BB* je 40$, SB je 20$, velikost *potu* je 80$. Dostali jsme karty K♣, 7♣, jsme *big blind* a protihráč před námi *checknul*. Společné karty jsou K♢, 9♡ a 7♠. Popis pravidla v přirozeném jazyce zní: „Máte-li výrazně méně než soupeř a méně než 10 BB a máte dva páry aniž by mezi společnými kartami hrozila *flush* nebo máte *flush*, zvyšte sázku na 5BB. V případě, že váš protihráč zvedal méně než 5BB, vsaďte všechno, jinak nechte rozhodnout externí program.“

Pravidlo bychom mohli přepsat takto:

```
RULE heroStack <= 0.5 villainStack, heroStack<=10BB, twoPair, rainbow
     heroStack <= 0.5 villainStack, heroStack<=10BB, flush
  THEN RAISE 5BB
  VILLAIN villainRaise <= 5BB
    THEN RAISE allIn
  ENDVILLAIN
  VILLAIN villainRaise > 5BB
    THEN external "~/decider wholeGame"
  ENDVILLAIN
ENDRULE
```



Obrázek 3. Schéma procesů při implementaci.

4. Implementace

Pro implementaci jsem zvolil jazyk *Java*, konkrétně Oracle JDK verze 6. [11] Většina pokerbotů je kvůli efektivitě psána v C/C++, ale jak jsem psal v sekci 3.1., na rychlost nekladu hlavní důraz. Java je sice obecně pomalejší [12], ale poskytne mi větší programátorský komfort, a navíc později by mohla být snadná implementace rozhraní pro známý software Poker Academy Pro [13].

K vývoji jsem použil vývojové prostředí *Netbeans IDE 7.2* [14], které patří mezi současnou špičku javových vývojových prostředí. Pro verzování a zálohování kódu jsem použil distribuovaný verzovací systém *Git* [15].

V následující části popíši celý převod zdrojového kódu pokerbota na funkční implementaci. Schematicky jej popisuje obrázek 3.

4.1. Lexikální analýza

Lexikální analýzou rozumíme proces, kdy zpracujeme vstupní zdrojový kód tak, že původní posloupnost symbolů rozdělíme na *lexémy* s určeným *tokenem*, tedy na řetězce s přiřazeným typem, se kterým dále pracuje syntaktický analyzátor.

Lexikální analyzátor je vstupní částí každého překladače. Má za úkol zkontrolovat, zda v kódu nejsou nějaké řetězce chybné, často odstraňuje bílé znaky a komentáře. Výstupem bývá dvojice (*token*, *lexém*), kterou dále zpracovává syntaktický analyzátor.

Jednoduchý lexikální analyzátor bychom mohli vytvořit tak, že bychom pro každý *token* určili množinu možných *lexémů* a poté bychom četli kód a pro každé slovo testovali, zda splňuje nějakou podmínky lexému. Takto implementovaný analyzátor by sice fungoval, ale nebyl by příliš efektivní, navíc jeho programování a případná úprava byla poměrně nekomfortní.

Tvorbu lexikálních analyzátorů značně zjednodušují generátory lexikálních analyzátorů, které na základě zadaných pravidel lexikální analyzátor vytvoří. Nejznámějším generátorem lexikálních analyzátorů pro C/C++ je *Lex* [16] nebo jeho otevřená varianta *Flex* [17], jejichž produkty jsou schopné spolupracovat se syntaktickými analyzátory vytvoření pomocí *Yacc* [18] či *GNU Bison* [19].

Pro vytvoření lexikálního analyzátoru jsem zvolil generátor *JFlex* [20], což je alternativa známého *Flex* naprogramovaná v jazyce Java. Sice již delší dobu není vyvíjen (Poslední verze 1.4.3 vyšla v lednu 2009), ale díky dobré dokumentaci,

podpoře regulárních výrazů, minimalizaci vytvořeného konečného deterministického automatu, podobnosti zadávání pravidel s *Flex* a kompatibilitou s generátory syntaktických analyzátorů *GNU Bison* nebo *CUP* [21] je pro mou práci ideální.

Pro ukázku zadávání pravidel pro *tokeny* uvádím část zdrojového souboru. V první části jsou uvedeny definice *tokenů*, ve druhé jsou definovány akce, které se mají provést při daném *tokenu*.

```
NUM = [0-9]+ ("." [0-9]+)?
CARDPREDICATE = [2-9{A,K,Q,J,T}][2-9{A,K,Q,J,T}][{o,s}]?[{+,-}]?
ACTION = "raise" | "check" | "call"

"RULE" {return PokerParser.RULE;}
{NUM}  {lVal = Double.parseDouble(yytext()); return PokerParser.NUM;}
{CARDPREDICATE} {lVal = (String)(yytext());
                 return PokerParser.CARDPREDICATE;}
{ACTION} {lVal = (String)(yytext()); return PokerParser.ACTION;}
```

4.2. Syntaktická analýza

Syntaktická analýza je proces, jehož vstupem je výstup lexikální analýzy a který kontroluje, zda je vstup větou bezkontextové gramatiky daného jazyka, tedy zda je vstup v předem dohodnuté formě.

Syntaktické analyzátory, někdy též *parsery*, bývají součástí interpretrů nebo kompilátorů jazyků. Vstup převádí do interní reprezentace – derivačního stromu, syntaktického stromu nebo jiných hierarchických struktur, se kterými poté pracuje kompilátor či interpretr pro určení sémantiky.

Konstrukce syntaktického analyzátoru je mnohem složitější než konstrukce lexikálního analyzátoru, proto se i zde používají generátory syntaktických analyzátorů. Tyto generátory dokáží zpracovat bezkontextovou gramatiku, zapsanou v Backus-Naurově formě nebo obdobné notaci, a vytvořit pro ni syntaktický analyzátor. Mezi známé generátory syntaktických analyzátorů patří unixový *YACC* [18], který generuje analyzátor v jazyce C, otevřený *GNU Bison* [19], který je kompatibilní s *YACC* a navíc experimentálně podporuje Javu, dále *Coco/R* [22], *CUP* [21] či *ANTLR* [23].

K vytvoření syntaktického analyzátoru jsem použil JBison [24], což je javový wrapper, který spouští GNU Bison. Oproti C/C++ rozhraní přináší několik rozšíření v deklaracích, ale v definici gramatiky se prakticky neliší, což přináší výhodu ve velkém množství dokumentačních zdrojů.

GNU Bison se umí vypořádat s oběma typy konfliktů jak *přesun/redukce*, tak i *redukce/redukce*. V podobě varování upozorňuje na přebytečné neterminály nebo nepoužitá pravidla, což značně usnadňuje zápis a testování gramatiky. Ve zdrojovém souboru s gramatikou se nejprve deklarují parametry pro jméno vý-

sledné třídy, hlášení chyb a podobně. Poté následuje zápis kódu v jazyce Java definujícího rozhraní, kde se mimo jiné specifikuje napojení na lexikální analyzátor či includy. Nezbytnou částí je definice jednotlivých tokenů, kde je vhodné kromě jména tokenu určit jeho javovou třídu nebo název, který se má zobrazovat při syntaktických chybách. Poté následuje definice samotné gramatiky, která se zadává v *Backus-Naurově formě* s mírně upravenou syntaxí. U jednotlivých pravidel se může do složených závorek uvést javový kód, který se má provést při aplikaci daného pravidla, který definuje převod zdrojového kódu jazyka do interní reprezentace. Například definice větvení podmínek u *RULE* vypadá následovně:

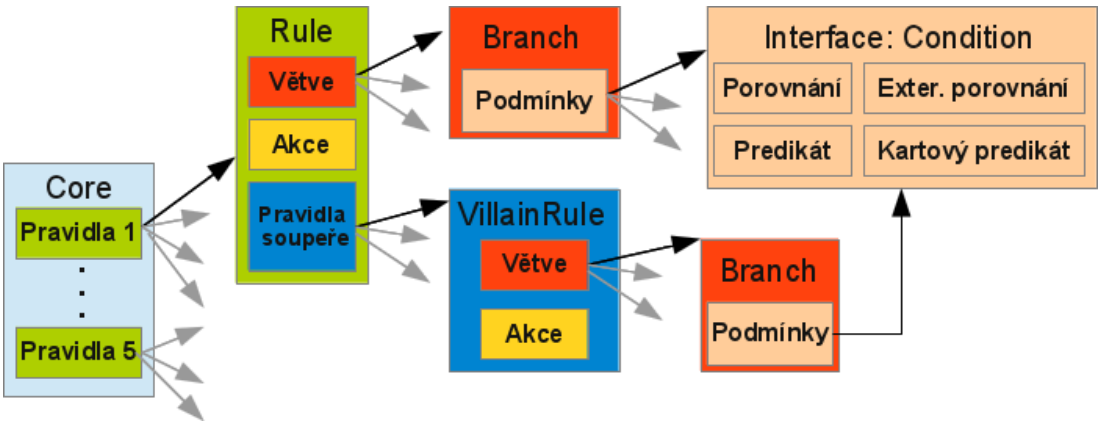
```
Andbranch:
   BranchItem
|  BranchItem COMMA Andbranch
|  BranchItem Andbranch
;
BranchItem:
  PREDICATE {core.addCondition(new Predicate($1, core));}
| NEG PREDICATE {core.addCondition(new Predicate($2, core, true));}
| NEG PREDICATE NL {core.addCondition(new Predicate($2, core, true));}
| PREDICATE NL {core.addCondition(new Predicate($1, core)); core.endBranch();}
| Comparation
| Comparation NL {core.endBranch();}
| CARDPREDICATE {core.addCondition(new CardPredicate($1, core)); }
| CARDPREDICATE NL {core.addCondition(new CardPredicate($1, core)); core.endBranch();}
;
```

Kde slova začínající velkým písmenem jsou neterminály gramatiky a slova psaná velkými písmeny jsou terminály/tokeny. Výraz *core* je hlavní třída implementace jazyka, zde se používá pro vytvoření interní reprezentace programu zadaným zdrojovým kódem.

4.3. Interní reprezentace

Pro interní reprezentaci kódu v jazyce jsem zvolil hierarchickou strukturu objektů, které významově kopírují konstrukce jazyka. Derivační strom je v mém případě nevhodný, protože Heads Up Script umožňuje mít definice pravidel pro fázi hry na různých místech zdrojového souboru, a to v libovolném pořadí. Pro snadné a přímé vyhodnocení je vhodné mít pravidla pro danou fázi na jednom místě, což derivační strom neumožňuje.

Převod do interní reprezentace obstarává syntaktický analyzátor, který volá metody hlavní třídy *Core*. Převod by se na první pohled mohl zdát jednoduchý, ale komplikuje jej fakt, že syntaktický analyzátor volá metody *Core* „z hloubky“. Tedy *Core* musí mít neustále vytvořené instance tříd v hierarchicky nadřazené vrstvě, protože nelze zjistit, zda se náhodou nebude ještě nějaký prvek přidávat. Při ukončování jedné úrovně objektu se musí vždy vytvořit nové instance. Například v kódu 4.2. `core.addCondition(new Predicate($1, core));` přidá nový predikát do aktuální větve aktuální pravidla, které je v aktuální fázi. Při výskytu odřádkování je potřeba ukončit danou větev a do pravidla přidat novou,



Obrázek 4. Interní reprezentace.

kam se budou přidávat další podmínky. Ukončení větve a přidání nové zajišťuje `core.endBranch();`.

Schéma interní reprezentace je znázorněno na obrázku 4. Velkými obdélníky jsou znázorněny třídy s jejich názvem, malé obdélníky značí, že daná třída obsahuje jinou. Šipky značí, že je možná vazba typu 1:N. U rozhraní *Condition* jsou znázorněny implementační třídy.

4.4. Vyhodnocení jazyka

Pro převod zdrojového kódu jazyka jsem mohl použít následující dvě metody:

Kompilaci jazyka – tedy převod kódu pokerbota do jiného nižšího jazyka. Výstupem by byl binární soubor reprezentující program, který by sémanticky splňoval chování jazyka. Výhodou kompilace je, že se lexikální a syntaktická analýza provádí pouze jednou a pokerbot by byl snadněji šiřitelný.

Intepretaci jazyka – tedy vytvoření programu, který by na vstupu bral kód pokerbota a přímo vykonával jeho činnost. Nevýhodou oproti kompilaci je provádění lexikální a syntaktické analýzy při každém spuštění. Velkou výhodou je však snadnější implementace.

Vzhledem k náročnosti vytvoření kompilátoru je pochopitelné, že jsem pro testovací účely a vývoj jazyka sestrojil „pouze“ interpretr jazyka. Samotná tvorba kompilátoru by zabrala mnohem více času a téma tvorby efektivního kompilátoru by vystačilo na další bakalářskou práci.

Výsledkem této práce je program, který dostává na vstupu soubor s kódem pokerbota v navrženém jazyce a XML soubor s reprezentací rozehrané hry. Program zkontroluje správnou syntaxi ve zdrojovém souboru, převede kód do interní reprezentace a interpretuje ji. Výstupem je textová informace „jak hrát“ založená na nadefinovaných pravidlech. Volání programu vypadá takto:

```
$> java -jar ./Parser.jar ./code ./game.xml
raise 25
```

4.4.1. XML vstup

Vstup s údaji o hře jsem volil v podobě XML souboru, protože je ve své podstatě dobře čitelný i pro méně znalé uživatele a i proto, protože většina pokročilejších on-line hráčů pokeru se s tímto formátem setkalo například při ukládání historií her. Formát XML je výhodný i pro svou přenositelnost mezi platformami/jazyky. Když si někdo bude chtít naprogramovat rozhraní pro nějakou herní aplikaci, tak nemusí znát Javu, ale může ve svém oblíbeném jazyce naprogramovat generování XML z herní aplikace a to předávat pokerbotovi na vstup.

Původně jsem chtěl využít nějaké schéma XML souboru, které používají některá známá kasina pro ukládání herní historie, avšak zjistil jsem, že prakticky každé kasino má své vlastní schéma a navíc záznamy z historie obsahují vždy karty obou hráčů. Kvůli nepřebernému množství formátů vznikají dokonce i konvertory mezi formáty. Navrhl jsem tedy vlastní schéma, které přesně vystihuje potřebu pokerbota, částečně jsem se inspirovat v diskuzi o sjednocení pokerových formátů [25]. Například zápis konkrétní hry ve fázi *flop* by vypadal takto:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<game>
  <init>
    <BB>20</BB>
    <SB>10</SB>
    <heroStack>1500</heroStack>
    <villainStack>2500</villainStack>
    <heroCards>
      <card rank="J" color="Hearts"></card>
      <card rank="3" color="Clubs"></card>
    </heroCards>
  </init>
  <posts>
    <post player="Hero" amount="10">SB</post>
    <post player="Villain" amount="20">BB</post>
  </posts>
  <preflop>
    <action no="0" player="Hero" act="call" amount="10"></action>
    <action no="1" player="Villain" act="raise" amount="20"></action>
    <action no="2" player="Hero" act="raise" amount="40"></action>
    <action no="3" player="Villain" act="raise" amount="20"></action>
    <action no="4" player="Hero" act="call" amount="20"></action>
```

```
  </preflop>
  <flop>
    <cards>
      <card rank="Q" color="Spades"></card>
      <card rank="T" color="Clubs"></card>
      <card rank="8" color="Clubs"></card>
    </cards>
    <action no="0" player="Hero" act="check" amount="0"></action>
    <action no="1" player="Villain" act="raise" amount="20"></action>
  </flop>
</game>
```

Uzel `<game>` musí obsahovat uzly `<init>`, `<posts>` a uzly s odpovídající fází hry. Program nijak nekontroluje, zda vstupní XML soubor neobsahuje nějaký sémantický nesmysl – například neplatné zadání karty, akci *fold*(neměla by smysl), špatné množství karet nebo špatnou hodnotu *raise*, předpokládá korektní XML schéma.

V uzlu `<init>` je brán jakýkoliv uzel kromě uzlu `<heroCards>` jako definice číselné konstanty, kde jméno uzlu značí jméno konstanty a obsah její hodnotu. V uzlu `<heroCards>` se nachází informace u pokerbotových kartách. Atribut `rank` udává hodnotu karty (2-9, T, J, K, Q, A) a atribut color její barvu (Hearts, Diamonds, Spades, Clubs). Společné karty jsou definovány stejným způsobem.

V uzlu `<posts>` jsou definovány počáteční vklady, kde uzel `<post>` značí jednotlivý vklad, kde atribut `player` udává hráče a `amount` výši vkladu.

Uzly pro jednotlivé fáze nesou stejné jméno jako fáze a obsahují definici společných karet (pokud v dané fázi nějaké jsou) a jednotlivé akce hráčů. `<action>` mají atribut `no`, který značí pořadí akce, a to proto, protože specifikace XML nezaručuje stejné pořadí uzlů při zpracovaní jako ve zdrojovém, souboru [26]. Dále je přítomen atribut `act` nabývající hodnot check, raise nebo call, udávající samotnou akci, atribut `player` pro jméno hráče a `amount` pro výši vkladu.

4.4.2. Reprezentace karet

Karty kóduji do 64bitového čísla typu *long*, kde každá barva má rezervované své 2 byty a hodnotu karty udává nastavená jednička v daném bitu a to od nejnižší zprava. Tato reprezentace má výhodu v tom, že jedno číslo typu *long* může obsáhnout až všechny karty z balíčku, nicméně pro mou potřebu si vystačím maximálně se 7. Snadná je také implementace predikátů, kde jsem se mohl omezit jen na několik bitových operací a cyklů.

Tuto karetní reprezentaci jsem přebral z [27], pouze jsem ji upravil tak, že jsem bity ve 2 bytech zarovnal místo doleva doprava. Původně jsem totiž chtěl použít ohodnocovač karet *The Pokersource Poker-Eval Evaluator*, který tuto reprezentaci používá, avšak pro použití Heads Up Script jsem si vystačil pouze

s predikáty, tak jsem použití ohodnocovače vypustil, avšak reprezentaci karet jsem zachoval.

Karty T♠, 5♢ v reprezentaci vypadají následovně:

```
      Spades            Clubs            Diamonds          Hearts
XXXAKQJT 98765432 XXXAKQJT 98765432 XXXAKQJT 98765432 XXXAKQJT 98765432
00000001 00000000 00000000 00000000 00000000 00001000 00000000 00000000
```

Kde v prvním řádku je hlavička pro jednotlivé dvojice bytů, ve druhém řádku je hlavička pro význam bitů a ve třetím je samotná bitová reprezentace dvojice karet. Navíc pětice společných karet A♡, Q♣, 7♡, 8♠, T♢ ve fázi *river* má opět 64bitů a vypadá takto:

```
00000000 01000000 00000100 00000000 00000001 00000000 00010000 00100000
```

Pro získání sedmice karet stačí pouze vypočítat bitové *OR* a s výsledkem vyhodnocovat predikáty.

4.4.3. Vyhodnocení pravidel

Vyhodnocují se pouze pravidla, která jsou definovaná pro aktuální fázi, a to následujícím způsobem:

1. Vyhodnotí se větve s podmínkami u pravidla s podporou tzv. *líného vyhodnocení*, tedy po první platné větvi se ostatní přeskočí. Jednotlivé větve jsou označeny za pravdivé, pokud jsou splněny všechny podmínky. V tomto případě stačí jedna nesplněná podmínka a můžeme označit větev za nepravdivou.

2. Pokud hraje *Hero* před *Villainem*, přidáme pravidlo do seznamu splněných pravidel.

3. Pokud hraje dříve *Villain*, postupně vyhodnocujeme pravidla pro reakci na protihráče podobně jako v bodu 1. Je-li splněno alespoň jedno, přidáme pravidlo do seznamu splněných pravidel. V opačném případě pravidlo splněno nebylo.

Pokud je seznam splněných pravidel prázdný, aplikace vypíše chybovou hlášku. Pokud je splněno pouze jedno pravidlo, aplikace vypíše akci nadefinovanou v pravidle, je-li splněných pravidel více, vypíše se varování a náhodně se zvolí jedno ze splněných pravidel.

4.4.4. Konstanty

Konstanty jsou udržovány ve struktuře *HashMap*, kde klíče jsou řetězce a hodnoty jsou čísla typu *double*. U klíčů aplikace nerozlišuje malá a velká písmena.

Všechny konstanty jsou vypočítány ještě před lexikální a syntaktickou analýzou. Konstanty jsou čerpány z několika zdrojů.

Prvním zdrojem je uzel `<init>` ze vstupního XML souboru, ve kterém se vše kromě uzlu `<heroCards>` bere jako definice konstanty. Těchto konstant teoreticky může být neomezené množství, ale problémem by bylo, jak k nim přistupovat z jazyka. Řešením by byla obecná funkce, která by měla jeden parametr typu řetězec a vracela by číselnou hodnotu konstanty. Toto použití zvážím v budoucnu.

Další konstanty jsou čerpány z historie hry. Historie je čerpána taktéž z XML souboru, ale převádím ji do interní reprezentace, se kterou se lépe počítá. Z historie jsou dopočítány také některé herní konstanty jako je *pot* nebo *effectiveStack*.

Zajímavé jsou konstanty *handRank2652* a *handRank169*, pro které jsem nenašel žádný podrobný popis, jak je vypočítat, a tedy mi nezbývalo nic jiného, než hodnoty „zadrátovat“ v podobě tabulky přímo do programu [28].

5. Možná rozšíření

Během navrhování jazyka Heads Up Script a implementace jsem narazil na několik zajímavých rozšíření, některá by značně zkomplikovala implementaci, a proto jsem je z časových důvodů vynechal. Rád bych je však v budoucnu implementoval.

5.1. Cachování výsledků predikátů

Vezměme predikát *air*, který rozhoduje, zda *Hero* nemá vůbec žádnou kombinaci. Je implementován pomocí ostatních predikátů takto:

```
return !(pair() || straight() || flush() || flushDraw() || straightDraw());
```

V případě, že se necachují výsledky predikátů, tak se počítá každý predikát při každém volání. Což je značně neefektivní. Řešením by byla jednoduchá cache, kde by klíčem bylo jméno predikátu a hodnota by byla *pravda/nepravda*. Při volání predikátu by se nejdříve ověřilo, zda již není znám výsledek a podle toho by se buď predikát vyhodnotil nebo vrátil známý výsledek.

5.2. Implementace rozhraní

Doposud hotové volání pokerbota pomocí zadání v XML není moc praktické pro širší pokerovou veřejnost, proto by bylo vhodné naprogramovat rozhraní pro některé pokerové programy, jako je například Poker Academy Pro [13], které má API v jazyce Java nebo pro OpenHoldem Project [6].

Rozhraní jsem neimplementoval, protože podstatou mé práce bylo hlavně navržení jazyka a pro testování implementace jsem si vystačil s ručním voláním.

5.3. Váha pravidel

V dosavadním návrhu jazyka si jsou všechna pravidla rovna, tedy pokud se splní dvě pravidla, vybere se náhodně jedno. Pravidla však mohou být významově a strategicky naprosto odlišná. Například máme-li pravidlo, které obsahuje mimo jiné to, že držíme pár a druhé, že máme čistou postupku. Může se stát, že jsou splněna obě pravidla, většina lidí by ale hrála podle toho silnějšího.

Řešením by bylo u každého pravidla uvádět jeho váhu či prioritu. Tedy číslo, které by ovlivnilo výběr pravidla v případě, že je jich splněno více. Otázkou je, zda by tohle číslo muselo být povinné nebo nepovinné s tím, že kdyby nebylo uvedeno, počítalo by se s nějakou výchozí hodnotou.

Závěr

Vyvinutý jazyk Heads Up Script umožňuje naprogramovat pokerbota pro poker varianty Heads Up i neprogramátorské většině pokerových hráčů. Má jednoduchou syntaxi a dostatečnou vyjadřovací schopnost pro přepis pravidel z přirozeného jazyka. Poskytuje mnoho konstrukcí, díky nimž je možné definovat chování i pro velmi specifické situace.

Vytvořil jsem funkční implementaci jazyka, která zahrnuje lexikální a syntaktický analyzátor a sémantickou část. To vše v jazyce *Java* za pomocí *JFlex* a *JBison*. Vytvořený pokerbot dokáže na základě znalosti o aktuální hře, která mu je předána pomocí XML souboru, rozhodnout podle pravidel o tom, jakou akci má provést.

Podařilo se mi splnit všechny hlavní cíle práce, během návrhu a implementace jsem ale narazil na několik možných rozšíření, která bych v budoucnu rád implementoval.

Reference

[1] Tim Harding. How much longer can man match the computer? *The Kibitzer. Chesscafe.com*. http://www.chesscafe.com/text/kibitz91.pdf, [Cit. 2013-05-05].

[2] Chinook project. http://webdocs.cs.ualberta.ca/ chinook/. [Cit. 2013-04-28].

[3] H. J. van den Herik, J. W. H. M. Uiterwijk, and J. van Rijswijck. Games solved: now and in the future. *Artificial Intelligence*, 134(1–2), 2002.

[4] Jonathan RUBIN and Ian WATSON. Coputer poker: A review. *Artificial Intelligence*, 175:958–987, 2011.

[5] The annual computer poker competition. http://www.computerpokercompetition.org/. [Cit. 2013-04-28].

[6] Openholdem bot. https://code.google.com/p/openholdembot/. [Cit. 2013-04-29].

[7] Shanky Technologies. Poker programming language. http://www.bonusbots.com/PPLguide.pdf. [Cit. 2013-05-06].

[8] Kevin Suffecool. Cactus kev's poker hand evaluator. http://www.suffecool.net/poker/evaluator.html. [Cit. 2013-04-28].

[9] C. Moshman. *Heads-Up No-Limit Hold'em: Expert Advice for Winning Heads-Up Poker Matches*. Two Plus Two Publishing, 2008.

[10] Bertrand Grospellier, Lee Nelson, Tysen Streib, and Tony Dunst. *The raiser's edge*. Huntington press, 2011.

[11] Oracle java jdk verze 6. http://www.oracle.com/technetwork/java/javase /downloads/jdk6downloads-1902814.html. [Cit. 2013-04-28].

[12] Brent Fulgham. The computer language benchmarks game. http://benchmarksgame.alioth.debian.org/u64q/java.php. [Cit. 2013-04-28].

[13] Softonic. Poker academy pro. http://poker-academy-pro.en.softonic.com/. [Cit. 2013-04-28].

[14] Netbeans ide. http://www.netbeans.org. [Cit. 2013-04-28].

[15] Git. http://git-scm.com/. [Cit. 2013-04-28].

[16] Lex online manual. http://dinosaur.compilertools.net/lex/index.html. [Cit. 2013-04-28].

[17] Flex: The fast lexical analyzer. http://flex.sourceforge.net/. [Cit. 2013-04-28].

[18] Yacc: Yet another compiler-compiler. http://dinosaur.compilertools.net/yacc/index.html. [Cit. 2013-04-28].

[19] Gnu bison. http://www.gnu.org/software/bison/. [Cit. 2013-04-28].

[20] Jflex - the fast scanner generator for java. http://www.jflex.de/. [Cit. 2013-04-28].

[21] Cup parser generator for java. http://www.cs.princeton.edu/ appel/modern/java/CUP/. [Cit. 2013-04-28].

[22] The compiler generator coco/r. http://ssw.jku.at/Coco/. [Cit. 2013-04-28].

[23] Another tool for language recognition. http://www.antlr.org/. [Cit. 2013-04-28].

[24] Bison and jbison 2.0. http://www-inst.eecs.berkeley.edu/ cs164/sp05/docs/jbison/bison.html. [Cit. 2013-04-28].

[25] Xml standard format for hh. http://forumserver.twoplustwo.com/45/software/xml-standard-format-hh-351873/. [Cit. 2013-04-29].

[26] Extensible markup language (xml) 1.0 (fifth edition). http://www.w3.org/TR/REC-xml/. [Cit. 2013-04-29].

[27] The pokersource poker-eval evaluator. http://www.codingthewheel.com/archives/poker-hand-evaluator-roundup/#pokersource. [Cit. 2013-04-29].

[28] Texas holdem starting hands, hand rankings. http://www.winholdem.net/handrank.txt. [Cit. 2013-04-29].

[29] Jakub Trmota. Pravidla pokeru. http://poker.forrest79.net/rules/poker-pravidla.pdf. [Cit. 2013-04-29].

A. Pravidla Texas Hold'em pokeru s českými překlady

Zjednodušená pravidla jsem přebral z [29], se souhlasem autora mírně poupravil a doplnil. U většiny pojmů je v závorce uveden anglický originál.

A.1. Obecná fakta o pokeru

- vždy se hraje s balíčkem 52 karet ve čtyřech barvách - kříže (clubs), káry (diamonds), srdce (hearts), piky (spades) - v hodnotách eso (Ace), král (King), královna (Queen), kluk (Jack), 10, 9, 8, 7, 6, 5, 4, 3 a 2
- partie pokeru se dělí na několik sázkových kol, při nichž hráči sází do společného banku (pot) a dostávají nebo si vyměňují karty, počet kol závisí na konkrétní variantě pokeru.
- hra probíhá po směru hodinových ručiček
- vybraní hráči musí na začátku partie umístit do společného banku povinné sázky (blinds či ante)
- na začátku každého kola je rozdán předem určený počet karet, buď do ruky (vidí je jen hráč, tzv. hole cards) nebo na stůl (vidí je všichni, tzv. community cards)
- jakmile je hráč na řadě, může si vybrat z následujících možností - pokud nikdo před ním nevsadil, může zůstat ve hře bez vsazení (check), složit karty (fold) nebo vsadit (bet), pokud některý z hráčů před ním vsadil, může dorovnat sázku (call), zvýšit ji (raise) nebo složit karty (fold) na konci posledního sázkového kola zbylí hráči ukáží své karty a ten s nejlepší výherní kombinací získává bank.

A.2. Sázení

- sázecí kolo končí v momentě, kdy poslední hráč dorovná poslední sázku, to znamená, že všichni hráči, kteří zůstávají ve hře, vložili do banku stejné množství žetonů. Všechna dorovnání nebo navýšení sázek jsou umístěna na stůl před každého hráče. Je tak hned vidět, kdo kolik vsadil. Poté, kdy se kolo sázek ukončí, stáhne dealer všechny sázky do banku.
- v limitních (limit) hrách je velikost první sázky (bet) a případných navýšení (raise) konstantní a počet navýšení (raise) pro jedno kolo může být omezen (př.: na 3 navýšení = první vsazení a dvě navýšení), pokud však zůstanou ve hře už jen dva hráči (heads-up), limit na počet navýšení se nemusí uplatnit

- ve hrách, kdy je sázka omezena aktuální výší banku (pot-limit), můžete, když jste na řadě, vždy vsadit nebo zvýšit sázku až do výše banku
- v bezlimitních hrách (no-limit) můžete kdykoliv během hry vsadit jakoukoliv sázku
- pokud všichni hráči stojí (check), aktuální sázkové kolo končí a pokračuje se dál

A.3. Texas Hold'em poker

Cílem v Texas Hold'em je sestavit co nejlepší výherní kombinaci pěti karet složenou z libovolných sedmi karet - z pěti společných a dvou vlastních a vyhrát tak bank se sázkami všech hráčů (pot).

A.3.1. Stručný popis průběhu hry

Partie začíná povinnými sázkami, poté dostane každý hráč do ruky dvě karty (hole cards), dalších pět je postupně vykládáno na stůl barvou nahoru a jsou společné pro všechny hráče. První tři ze společných karet (flop) jsou vyloženy najednou, čtvrtá (turn) a pátá (river) jednotlivě. Hráč má možnost zůstat ve hře bez vsazení (check), dorovnat sázku (call), vsadit (bet), zvýšit sázku hráče před sebou (raise) nebo složit karty (fold). Partie pokeru končí ukázáním karet zbývajících hráčů a porovnáním výherních kombinací (showdown). Hráč s nejvyšší kombinací vyhrává bank (pot).

V Texas Hold'em se povinná sázka nazývá blind. Small blind (malá sázka naslepo) platí pouze pro prvního hráče po směru hodinových ručiček za rozdávajícím (dealer), big blind (velká sázka naslepo) platí pouze pro druhého hráče po směru hodinových ručiček za rozdávajícím. Small blind má obvykle výši poloviny nižšího limitu hry, big blind odpovídá nižšímu limitu hry.

A.3.2. Krok po kroku

Zahájení hry (1. kolo sázek) – Každá partie Texas Hold'em začíná povinnými sázkami (blind). Poté dostane každý hráč dvě karty (rozdané po jedné, první kartu obdrží hráč na small blindu), které ostatní nevidí (hole cards). První na řadě je hráč následující za big blindem, tedy třetí hráč za dealerem. Hráči mohou buď dorovnat sázku (call), zvýšit sázku (raise) nebo složit karty (fold). Jelikož je na stole sázka (big blind), není možné zůstat ve hře bez vsazení (check) s výjimkou hráče na big blindu (pokud všichni hráči maximálně dorovnali (call), může hráč na big blindu, když na něj přijde poprvé řada, položit (fold), zvýšit (raise) nebo zůstat ve hře bez vsazení (check), což ukončí první kolo sázek).

Flop (2. kolo sázek) – Dealer sejme jednu kartu a na stůl vyloží tři karty barvou nahoru (flop). Sázkové kolo začíná první hráč vlevo od dealera, který zůstává ve hře. Každý z hráčů může karty na stole, spolu s těmi co má v ruce, použít k vytvoření výherní kombinace. Sázení se řídí stejnými pravidly jako v předchozím kole. Dokud nikdo nevsadí, je možno zůstat ve hře bez vsazení (check), po sázce je možno pouze dorovnat sázku (call), zvýšit sázku (raise) nebo složit karty (fold).

Turn (3. kolo sázek) – Dealer sejme kartu a přidá na stůl čtvrtou kartu (turn). I tuto kartu lze použít k vytvoření výherní kombinace. Při a po turnu se sázky zvyšují na horní hranici limitu hry. Sázení se řídí stejnými pravidly jako v předchozím kole.

River (4. kolo sázek) – Dealer sejme kartu a přidá na stůl pátou kartu (river), poslední kartu, kterou lze použít k vytvoření výherní kombinace. Sázení se řídí stejnými pravidly jako v předchozím kole. Po skončení tohoto kola se ukáží karty a vítěz vyhrává bank. Novým dealerem se stává hráč sedící po levici aktuálního dealera.

A.4. Výherní kombinace

Partie pokeru končí dvěma způsoby. Buď jeden hráč donutí sázkami ostatní hráče složit karty a vyhrává bank bez nutnosti ukázat karty, nebo v posledním kole zůstane více hráčů. Ti pak ukážou karty a porovnají své výherní kombinace (showdown). Hráč s nejvyšší kombinací vyhrává bank (pot).

Výherní kombinace jsou řazeny od nejsilnější po nejslabší. Nerozlišují se barvy karet.

Royal flush – nejvyšší výherní kombinace v pokeru – královská barva – postupka s esem, všechny karty ve stejné barvě.

Straight flush – je postupka v barvě – pět po sobě následujících karet, všechny ve stejné barvě, postupka zakončená vyšší kartou vyhrává

Four of a kind (poker) – čtveřice karet se stejnou hodnotou, v případě rovnosti kombinací vyhrává vyšší čtveřice

Full house – trojice (three of a kind) a dvojice (a pair) dohromady, v případě rovnosti kombinací vyhrává hráč s vyšší trojicí

Flush – pět karet stejné barvy, v případě rovnosti kombinací vyhrává hráč s nejvyšší kartou v barvě

Straight – postupka – pět po sobě následujících karet, v případě rovnosti kombinací vyhrává postupka zakončená vyšší kartou

Three of a kind (trips) –trojice – tři karty se stejnou hodnotou, v případě rovnosti kombinací vyhrává vyšší trojice

Two pair – dva páry čili dvě dvojice karet se stejnou hodnotou, v případě rovnosti kombinací vyhrává vyšší pár z obou dvojic, je-li vyšší pár stejný, rozhoduje nižší pár, je-li i nižší pár stejný, rozhoduje zbývající karta

Pair (one pair) – jeden pár čili dvojice karet se stejnou hodnotou, vyšší pár poráží pár nižší, při rovnosti párů rozhoduje zbývající karta v ruce (kicker)

High card – v případě, že nemáte žádnou z výše uvedených kombinací, počítá se pouze nejvyšší karta

Mají-li dva nebo více hráčů stejnou výherní kombinaci složenou ze stejných karet (např. všichni mají postupku zakončenou Q), dělí se bank rovným dílem mezi ně. Nerozlišuje se barva karet. Mají-li dva nebo více hráčů stejnou výherní kombinaci složenou z různých karet (např. trojice es, trojice osmiček atd.), vyhrává bank kombinace nejvyšších karet.

B. Obsah přiloženého CD

`bin/`
: Zkompilovaná verze parseru Heads Up Script v podobě spustitelného jar souboru.

`data/`
: Ukázková testovací data pro běh. Včetně zdrojových kódů pokerbota, externího programu, XML souboru se hrou a ukázkové tabulky akcí.

`doc/`
: Bakalářská práce ve formátu PDF a ZIP soubor se zdrojovými soubory práce a obrázky.

`src/`
: Kompletní zdrojové soubory parseru jazyka včetně nutných souborů pro bezproblémové zkompilování aplikace. Makefile pro snadné zkompilování.

`readme.txt`
: Instrukce pro spuštění parseru,