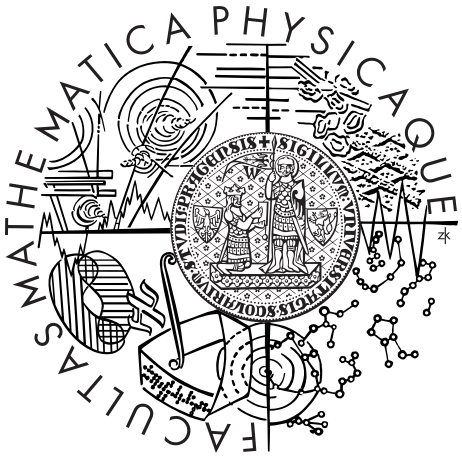
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Stanislav Nowak

Formální popis deskových her

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: Prof. RNDr. Petr Štěpánek, DrSc.
Studijní program: Informatika
Studijní obor: Obecná informatika

Praha 2011

Na tomto místě bych chtěl poděkovat zejména vedoucímu své práce profesoru Petru Štěpánkovi a rodině za podporu.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Název práce: Formální popis deskových her

Autor: Stanislav Nowak

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: prof. RNDr. Petr Štěpánek DrSc., Katedra teoretické informatiky a matematické logiky

Abstrakt: Cílem bakalářské práce bylo navrhnout matematický formalismus umožňující popis a zkoumání vlastností deskových her. Práce těží především z poznatků teorie automatů a logického programování. Z teorie automatů se práce zabývá konečnými automaty a jejich možnou aplikací pro potřeby deskových her. Výsledkem je rozšíření konečného automatu pokrývající specifika deskových her nazvané herní automat. Deskové hry jsou komplexní doménou a nabízí se využít k jejich popisu prostředků vyšší úrovně. Takovým prostředkem je i logické programovaní. S využitím logického programování byl navržen nástroj, který umožňuje deklarativní popis her, pojmenovaný herní systém. Tento deklarativní popis tvoří rámec pro implementaci interaktivních her. Propojení obou světů bude demonstrováno převodem herního systému na herní automat.

Klíčová slova: hry, automaty, gramatiky, logické programování

Title: Formal description of board games

Author: Stanislav Nowak

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: prof. RNDr. Petr Štěpánek DrSc., Department of Theoretical Computer Science and Mathematical Logic

Abstract: The aim of thesis was to design a mathematical formalism that allows describing and exploring the properties of board games. The work benefits from the findings of automata theory and logic programming. First part of thesis deals with finite automata and their possible applications for the needs of board games. The result is an extension of finite-state automaton covering the specifics of board games called game automaton. Board games are a complex domain hight level tools should be used. Such a tools is a logical programming. Using logic programming was designed instrument that allows declarative descriptions of games, named gaming system. This declarative description is used as an framework for the implementation of interactive games. Linking the two worlds will be demonstrated by conversion of the gaming system to the game automaton.

Keywords: games, grammars, automata, logic programming

Obsah

Úvod

Deskové hry byly součástí kultury mnoha civilizací často i předtím, než ovládly písmo. Nejstarší záznam věnovaný deskovým hrám se datuje do doby 3500 let před naším letopočtem. V posledních letech se deskové hry těší čím dál větší oblibě, zejména díky hrám moderního typu, někdy také označované jako hry německého typu. Tento typ her přináší řadu inovativních a zábavných mechanismů, ale za ceny značného zvýšení komplexnosti herních pravidel.

Návrh, vývoj a zejména testování těchto her se stává složitější a méně přehledný. Hrozí, že výsledná pravidla budou obsahovat různé vady. V bakalářské práci navrhneme formalismus nazvaný *herní systém*, kterého bude možné využít k deklarativnímu popisu her, ke zkoumání struktur a vlastností her.

Na oblibě získávají i různé online systémy pro provoz deskových her. Nabízí se možnost využití herního systému pro deklarativní popis her do těchto systémů. S jeho pomocí pak můžeme ověřovat platnosti hráčských tahů a vyhodnocovat herní situace. Toto užití je však závislé na výpočetních vlastnostech formalismu. Budeme se jimi rovněž zabývat, ale pouze okrajově.

Běžně se matematika zajímá o hry především prostřednictvím disciplíny nazývané teorie her. Ta se však věnuje hlavně výherním strategiím a podmínkám výhry jednotlivých hráčů. Náš cíl je odlišný, neboť se zabýváme čistě formálním popisem. Přesto má tato práce pro teorii her určitý význam. Příprava modelu pro teorie her může být zejména u her se složitými pravidly velmi náročná. Pomocí metod, které si představíme, lze tento úkol zjednodušit. Obor, který se zabývá tvorbou univerzální umělé inteligence pro deskové hry, se nazývá general game playing. V tomto oboru se věnují formálnímu popisu her, ale žádný z nich nezachází do takové hloubky jako ten představený v této práci.

V první kapitole se podíváme na deskové hry z pohledu teorie automatů. Konečné automaty nabízejí intuitivní model k popisu průběhu a vyhodnocení hry. Zavedeme variantu konečného automatu nazývanou herní automat. Tuto variantu dále zobecníme na stochastický herní automat, abychom postihli širší škálu her. Přes svou jednoduchost přinášejí konečné automaty řadu problémů, a proto zavedeme formalismus nazvaný herní systém.

Budeme se jím podrobně zabývat ve druhé kapitole. Při návrhu herního systému jsme těžili především z poznatků logického programovaní. Umožní nám deklarativně popisovat hry. Zkusíme si nastínit použití herního systému jako rámce pro tvorbu interaktivních her. V závěru kapitoly si předvedeme převod herního systému na herní automat.

Ve čtvrté kapitole uvedeme několik zpracovaných her. K tomu nebudeme používat přímo představený formalismus, ale dáme přednost jazyku prolog. Ten obsahuje řadu vlastností, které nám usnadní práci a navíc jeho syntax je povědomější než jazyk herního systému. Budeme popisovat klasické hry, jako třeba pasiáns nebo člověče nezlob se, které jsou podstatně jednodušší na analýzu než moderní hry.

Následně provedeme srovnání herních systémů s podobnými formalismy. V závěru zhodnotíme práci jako celek.

V této práci budeme volně zaměňovat pojmy deskové, stolní a společenské hry. Všechna tři označení jsou pro naše potřeby ekvivalentní.

1. Herní automaty

K popisu nejrozmanitějších fyzikálních, biologických, ekonomických systémů, které se mohou vyskytovat v konečně mnoha stavech a reagují deterministicky na konečně mnoho druhů podmětů či vstupů přechodem do dalšího stavu, často popisujeme formalismem nazývaný konečný automat. Ani desková hra nebude výjimkou. K jejímu popisu budeme používat rozšíření a zobecnění konečného automatu nazvané *herní automat*.

Herní automat se skládá ze všech možných herních situací označovaných jako herní stavy. Platí, že automat se vždy nachází pouze v jednom z možných herních stavů. Mezi herními stavy se přechází tahem aktuálního hráče.

Průběh hry je jednoznačně určen posloupností tahů. Hlavním účelem herního automatu je pro danou posloupnost tahů určit, zda některý z hráčů nedosáhl vítězství, případně, zda nenastala remíza.

1.1 Herní automat

Nyní již přistoupíme k definici herního automatu.

Definice (Herní automat)**.** *Herní automat je uspořádaná osmice:*

$$A = (S, P, T, W, s, \omega, \tau, \lambda), kde$$

S *je neprázdná konečná množina herních stavů*

P *je neprázdná konečná množina hráčů*

T *je neprázdná konečná množina tahů*

W *je neprázdná konečná množina vítězných stavů,* $W \subseteq S$

s *je počáteční stav,* $s \in S$

ω *je funkce výběru hráče,* $\omega : S \to P$

τ *je tahová funkce,* $\tau : S \times T \to S$

λ *je vítězná funkce,* $\lambda : W \to P \cup \{\varepsilon\}$

Množina herních stavů **S** reprezentuje všechny možné herní situace.

V množině **P** jsou obsaženi hráči účastnící se hry.

Množina tahů **T** představuje akce, které mohou hráči ve hře vykonat. Odpovídá vstupní abecedě u konečného automatu.

Množina vítězných stavů **W** obsahuje herní situace, v nichž vyhrává některý z hráčů nebo bylo dosaženo remízy. Ve vítězném stavu končí automat svoji činnost.

Počáteční herní situace je reprezentována stavem **s**.

Aktuální hráč na tahu v daném stavu je určen funkcí výběru hráče ω.

Tahová funkce τ stanovuje tahy, jež může hráč v příslušném stavu provést pro přesun do dalšího stavu. V konečném automatu odpovídá přechodové funkci.

Vítězná funkce λ určuje hráče, který vyhrál v daném vítězném stavu. Výslednou hodnotou ε je označena remíza.

Některé hry umožňují výhru několika hráčů v jednom stavu. Klasické hry důsledně dbají na to, aby taková situace nenastala, nebo ji řeší remízou. Řada moderních her však tuto situaci umožňuje. V naší práci se těmito případy nebudeme zabývat.

Všimněme si, že tento automat není deterministický. Hráč nemůže v jakémkoliv stavu zahrát libovolný tah. Tahová funkce tedy není totální. Standardně budeme vstup, který nespadá do definičního oboru tahové funkce ignorovat. Problém lze vyřešit i tak, že zavedeme zvláštní stav reprezentující chybný tah, do kterého bude tahová funkce zobrazovat všechny neplatné tahy.

1.1.1 Popis činnosti automatu

Automat začíná činnost v počátečním stavu **s**.

V každém stavu se pomocí funkce ω určí hráč na tahu. Tento hráč vykoná některý z tahů z **T** přípustný v daném stavu. Automat se tak převede do dalšího stavu.

Výsledný stav po hráčově tahu tedy odpovídá aplikaci tahové funkce τ, kde parametrem je aktuální stav a zvolený tah.

Pokud výsledný stav spadá do množiny vítězných stavů **W**, hra a činnost automatu je ukončena. Použitím vítězné funkce λ se zjistí vyhrávající hráč. V případě, že výsledkem vítězné funkce je ε, vyhlásí se remíza.

Může nastat i taková situace, že hráč nemůže v daném stavu provést žádný tah. Taková situace však svědčí o chybném návrhu hry.

1.1.2 Pomocné definice

Zavedeme tahovou funkci, která umí vyhodnotit posloupnost tahů.

Definice (Rozšířená tahová funkce)**.** *Nechť τ je tahová funkce, $\tau : S \times T \to S$. Pak rozšířená tahová funkce τ^* je*

$$\tau^*(q, t) = \tau(q, t)$$
$$\tau^*(q, wt) = \tau(\tau^*(q, w), t), t \in T, w \in T^*, q \in S$$

Definice (Vítězná posloupnost tahů pro hráče)**.** *Řekneme, že posloupnost tahů a je vítězná pro hráče p, jestliže*

$$\tau^*(s, a) \in W \wedge \lambda(\tau^*(s, a)) = p$$

Definice (Remízová posloupnost tahů)**.** *Řekneme, že posloupnost tahů a je remízová, jestliže*

$$\tau^*(s, a) \in W \wedge \lambda(\tau^*(s, a)) = \varepsilon$$

Množiny všech vítězných posloupností pro hráče společně s množinou remízových posloupností budeme nazývat *množinou všech partií*.

Nabízí se po vzoru formálních jazyků definovat hru jako množinu všech partií. Zkusme si však představit, že by nám skutečně někdo popsal hru jako množinu všech možných partií. Taková informace nám prakticky žádnou představu o hře nepřinese. Ke hře tedy budeme přistupovat skrze herní automat.

Uvedeme několik definic popisujících vlastnosti her.

Definice (Ukončitelná)**.** *Řekneme, že hra popsaná herním automatem je ukončitelná, jestliže každá nekonečná posloupnost tahů začínající v počátečním stavu skončí po konečném množství kroků vítěstvím některého z hráčů nebo remízou.*

Definice (Totálně definovaná)**.** *Řekneme, že hra popsaná herním automatem je totálně definovaná, jestliže v každém dosažitelném stavu, který není vítězný pro nějakého hráče ani remízový, existuje pro aktuálního hráče alespoň jeden možný tah.*

Definice (Silně vítězná)**.** *Řekneme, že hra popsaná herním automatem je silně vítězná, jestliže existuje vítězná posloupnost tahů hráče p, taková, že každý tah je prováděn pouze tímto hráčem.*

Definice (Slabě vítězná)**.** *Řekneme, že hra popsaná herním automatem je slabě vítězná, jestliže existuje vítězná posloupnost tahů pro každého hráče tak, že tahy jsou prováděny i ostatními hráči.*

Definice slabě vítězné hry říká, že žádný hráč nemůže ve hře zvítězit, aniž by ostatní hráči měli možnost zasáhnout svými tahy. Slabě vítězné hry jednoho hráče jsou zároveň silně vítězné.

Definice (Dobře definovaná)**.** *Řekneme, že hra popsaná herním automatem je dobře definovaná, jestliže je slabě vítězná, ukončitelná a totálně definovaná.*

Dobře definovaná hra zaručuje, že neobsahuje žádné vážné vady, které by znemožnily její dohrání, nebo aby některý z hráčů měl výraznou převahu. Je možné přidávat další podmínky zaručující lepší vyváženost, ale těmi se v této práci nebudeme zabývat.

Definice (Hra je symetrická pro hráče)**.** *Řekneme, že hra popsaná herním automatem je symetrická pro hráče, jestliže všichni hráči mají k dispozici stejnou sadu tahů.*

Většina klasických her je symetrická.

Definice (Herní automat s normovanými vítěznými stavy)**.** *Řekněme, že herní automat má normované vítězné stavy, jestliže existuje nejvýše jeden vítězný stav pro každého hráče a nejvýše jeden remízový.*

Věta (Normalizace vítězných stavů herního automatu)**.** *Libovolný herní automat lze převést na herní automat s normovanými vítěznými stavy.*

Důkaz. Popíšeme algoritmus převodu. Pro každého hráče přidáme do automatu nový vítězný stav, a pokud hra umožňuje i remízu, tak i jeden remízový. Budeme postupně procházet původní vítězné stavy. Pro každý stav zjistíme, jaký hráč v něm vítězí. Uvědomíme si, že ve všech vítězných stavech automat končí svoji činnost, a z tohoto stavu se již nelze přejít do dalšího stavu. Tento stav tedy můžeme vymazat. Upravíme tahovou funkci tak, že všechny stavy, které do něj vedly, nasměrujeme do nově přidaného vítězného stavu odpovídajícího hráče. Stejně budeme zacházet i s remízovými stavy. □

Poznámka (O párových hrách). Existuje kategorie her, kde sice tahy vykonávají jednotlivý hráči, ale vítězství je přiřazeno páru, případně skupině hráčů. Mezi takové hry patří například whist nebo bridge. Na první pohled se zdá, že takovou hru neumíme pomocí herního automatu popsat kvůli omezení na vítěství jednoho hráče v jednom stavu.

Problém lze ale vyřešit tak, že pro každý pár zavedeme nového hráče. Vítězné stavy, ve kterých měl vyhrát pár, označíme jako vítězné pro nově zavedeného hráče.

1.2 Stochastický herní automat

U některých her není stav hry určen pouze deterministickým rozhodováním hráče, ale i prvkem náhody, a následující stav je dán výsledkem náhodného pokusu. Mezi takové hry patří člověče nezlob se nebo vrhcáby. Například ve hře člověče nezlob se je pole pro přesun figurky určeno výsledkem hodu šestistěnnou kostkou. Herní automat tak, jak jsme jej definovali, se nedokáže s takovým typem her vypořádat, a proto pro něj zavedeme stochastický herní automat.

U stochastického automatu se hra vyskytne v jistém stavu s určitou pravděpodobností.

Definice vychází ze základního herního automatu a umožňuje pracovat pouze s jedním generátorem náhody. Budeme také požadovat, že výsledkem náhodného pokusu může být pouze elementární jev.

Definice (Stochastický herní automat)**.** *Nechť* (Ω, F, P_Ω) *je pravděpodobnostní prostor. Stochastický herní automat je:*

$$A = (S, D, N, P, T, W, s, \omega, \tau, \kappa, \lambda), kde$$

S *je neprázdná konečná množina herních stavů*

D *je množina deterministických stavů,* $D \subseteq S$

N *je množina náhodných stavů,* $N \subseteq S$

P *je neprázdná konečná množina hráčů*

T *je neprázdná konečná množina tahů*

W *je neprázdná konečná množina vítězných stavů,* $W \subseteq S$

s *je počáteční stav,* $s \in S$

ω *je funkce výběru hráče,* $\omega : S \to P$

τ *je tahová funkce,* $\tau : D \times T \to S$

κ *je randomizační tahové funkce,* $\kappa : N \times \Omega \to S$

λ *je vítězná funkce,* $\lambda : W \to P \cup \{\varepsilon\}$

Množina deterministických stavů D reprezentuje stavy, ve kterých o přechodu do dalšího stavu rozhoduje hráč svým tahem.

Množina náhodných stavů N představuje stavy, ve kterých o následujícím stavu rozhoduje výsledek náhodného pokusu.

Platí, že $D \cup N = S$

Randomizační tahové funkce τ určují další stav podle výsledku náhodného pokusu. Výsledkem náhodného pokusu musí být elementární náhodný jev.

1.2.1 Popis činnosti automatu

Automat začíná činnost v počátečním stavu **s**.

Pomocí ω se určí hráč na tahu.

Zjistíme, zda stav patří do N nebo do D. Pokud stav spadá do D, hráč provede svůj tah a výsledný stav se určí aplikací tahové funkce τ stejně jako u základního herního automatu.

Jestliže stav spadá do N, hráč vykoná náhodný pokus. Náhodný pokus ve většině her odpovídá hodu kostkou nebo mincí. Podle výsledného náhodného jevu získáme aplikací náhodné přechodové funkce κ další stav.

Je možné, že aktuální stav spadá do množiny D i N. Pak si hráč vybere, zda provede tah nebo náhodný pokus.

Podmínky ukončení běhu automatu jsou stejné jako u základního varianty.

1.2.2 Pomocné definice

Definice (Pravděpodobnost stavu)**.** *Nechť je dána posloupnost tahů pro stochastický herní automat $wt \in \{T \cup \Omega\}^*$ začínající v počátečním stavu. Pak můžeme pomocí funkce spočítat τ_Ω pravděpodobnost, s jakou herní automat dojde do výsledného stavu.*

$$\tau_\Omega(q, \varepsilon) = 1$$
$$\tau_\Omega(q, wt) = P_\Omega(t)\tau_\Omega(q, w), t \in \{T \cup \Omega\}, w \in \{T \cup \Omega\}^*, q \in S$$

Pokud je $t \in T$, pak $P_\Omega(t) = 1$.

1.3 Zhodnocení herního automatu

Herní automat poskytuje jednoduchý a intuitivní model pro popis her. Přes svou jednoduchost se však vyznačuje řadou nepříjemných vlastností, které omezují jeho reálné použití.

Popis hry pomocí herního automatu je příliš zevrubný a neumožňuje bližší zkoumání struktury hry.

Ruční konstrukce herního automatu není triviální. Popis složitější hry může být značně nepřehledný a nezvládnutelný.

Největší slabinou herního automatu jsou jeho paměťové nároky. Automat pro triviální hru tic-tac-toe bude obsahovat celkem 19683 stavů. Komplexní hry mohou nabývat více něž stovky milionů stavů.

Herní automat tak zůstává především teoretickou konstrukcí.

2. Herní systém

V této kapitole se budeme zabývat dalším formalismem pro popis deskových her nazvaný herní systém. Deskové hry jsou obecně složitá doména a k jejich popisu je vhodné použít jazyk se silnou vyjadřovací schopností. Tímto jazykem bude logické programovaní. Výsledný formalismus pak bude možné použít k deklarativnímu popisu her.

2.1 Definice herního systému

Při formulaci definice herního systému vycházíme z pozorování, že libovolná desková hra se skládá ze dvou druhů komponent, které budeme nazývat tokeny T a herní zóny Z.

Tokeny, někdy také nazývané herní díly, jsou pohyblivé herní prvky a hráči je svými tahy umisťují do herních zón. Tokeny odpovídají například figurkám ve hře člověče nezlob se, kolečkům a křížkům ve hře piškvorky, figurám ve hře šachy nebo kartám v nejrůznějších karetních hrách.

Herní zóny slouží jako prostor pro umístění tokenů. Standardně jsou herní zóny nehybné, ale zejména v moderních hrách mohou plnit úlohu tokenů a být umístěny do jiné herní zóny.

Ve hře šachy a piškvorky jsou herní zóny jednotlivá hrací pole. Karetní hry často obsahují několik druhů herních zón. Příkladem může být hra prší, kde herní zóny odpovídají hráčově ruce s kartami, společné vykládací hromádce na stole a místu s balíčkem karet pro lízání.

Platí, že token musí být vždy umístěn v nějaké zóně a nemůže být umístěn v několika zónách najednou. Vyžadujeme, aby herní zóny byly po dvou disjunktní. Vztah tokenů a herních zón nám tedy umožňuje definovat *rozmisťovací funkci*. Ta bude spolu s dalšími údaji (aktuální hráč, etc.) sloužit jako hlavní údaj pro popis a identifikaci stavu hru.

Definice (Rozmisťovací funkce)**.** *Nechť Z je množina zón a T tokenů. Pak rozmisťovací funkce je definována následovně: $\rho : T \to Z$.*

S matematickou funkcí se neoperuje příliš pohodlně, zejména, co se množinových operací týče. Zavedeme proto alternativní definici k rozmisťovací funkci s podobnou sémantikou. Využijeme množinový systém, kde jeho prvky budou odpovídat zónám. O zónách budeme uvažovat jako o unárních relacích nad množinou tokenů. Zároveň budeme požadovat, aby pro každý token existovala právě jedna zóna, ve které je umístěn. Formálně zapíšeme následovně.

Definice (Rozmístění tokenů do zón)**.** *Nechť T je množina tokenů a Z množina zón. Pak množinový systém*

$$Z_T \equiv \{r_z | z \in Z, r_z \subseteq T\}$$

splňující

$$\forall t \in T \; \exists! r \in Z_T : t \in r$$

budeme nazývat rozmístění tokenů T do zón Z.

Aktuální herní situace je dána hráčem na tahu a rozmístěním tokenů do zón.

Definice (Herní stav)**.** *Nechť P je množina hráčů, Z_T je rozmístění tokenů T do zón Z. Pak herní stav s je určen jako následovně:*

$$s \equiv (p, Z_T),\ p \in P.$$

Obecně lze pravidla deskových her rozdělit na dva základní typy. První typ pravidel popisuje strukturu hry. Určují počet hráčů ve ve hře, jaké se budou při hře používat tokeny a zóny, do kterých budou umisťovány. Těmito pravidly jsme se zabývali doposud.

Druhý typ pravidel vymezuje, jak můžou hráči pomocí svých tahů měnit strukturu hry. Tato pravidla se v logice obvykle označují jako odvozovací nebo strukturální. Pro popis strukturálních pravidel budeme používat klauzule logického programování. Budeme je nazývat tahové klauzule. Pro inferenci pravidel rezoluční metody logického programování využijeme SLD-rezoluci.

Krátce si připomeneme syntax klauzule.

Definice (Syntax klauzule)**.** *Formule je definována induktivně:*

1. *term*
   (a) *každá proměnná je term.*
   (b) *je-li f n-ární funkční symbol a $t_1, t_2, \ldots, t_n$ jsou termy, potom $f(t_1, t_2, \ldots, t_n)$ je term.*
2. *formule*
   (a) *je-li p n-ární predikátový symbol a $t_1, t_2, \ldots, t_n$ jsou termy, potom $p(t_1, t_2, \ldots, t_n)$ je atomická formule, říkáme jí atom.*
   (b) *jsou-li $H, B_1, B_2, \ldots, B_n$ atomy, potom $H \leftarrow B_1, B_2, \ldots, B_n$ je formule, říkáme jí definitní klauzule.*
   Atomu H říkáme hlava a n-tici atomů $B_1, B_2, \ldots, B_n$ říkáme tělo klauzule.
   (c) *jsou-li $A_1, A_2, \ldots, A_n$ atomy, říkáme této n-tici dotaz.*

V knize [Russell – Norvig(2010)Russell, Norvig] je uveden situační kalkulus, kde je každá akce popsána dvěma axiomy: *axiom možnosti (possibility axiom)* a *axiom efektu (effect axiom)*. Axiom možnosti říká, kdy lze použít akci, a axiom efektu určuje, co se stane po použití této akce.

V podobném duchu budeme definovat i tahové klauzule. Tělo tahových klauzulí bude složeno z klauzulí předpokladu a z klauzulí efektu.

Klauzule předpokladu určují herní stavy, ve kterých lze uplatnit tahovou klauzuli. Stanovují podmínky, jako vlastnosti tokenů a zón a relace mezi nimi, které musí být splněny, aby hráč mohl provést tah.

V hře piškvorky si to lze představit například tak, že podmínková klauzule hlídá, že nebude možné zapsat křížek či kolečko do obsazeného pole.

Klauzule efektu popisují změnu herního stavu, tedy jak se změní stav hry po aplikaci hráčova tahu.

Takové klauzule budou definovány tak, že první proměnná bude vstupní herní stav a druhý bude změněný výstupní herní stav. Klauzule předpokladu pracují pouze se vstupním herním stavem. Klauzule efektu mají opět dvě proměnné odpovídající vstupnímu a výstupnímu hernímu stavu.

Definice (Tahová klauzule)**.** *Taková klauzule je klauzule následujícího tvaru:*

$$T(s_{in}, s_{out}) \leftarrow P_1(s_{in}), \ldots, E_n(s_{in}), E_1(s_{in}, s_1), \ldots, P_m(s_m, s_{out})$$

kde

$P_1 \ldots P_n$ *jsou klauzule předpokladu,* $n \in N$

$E_1 \ldots E_m$ *jsou klauzule efektu,* $m \in N$

$s_{in}, s_{out}, s_1 \ldots s_m$ *jsou herní stavy*

Budeme vyžadovat, aby všechny tahové klauzule měly stejný tvar hlavy.

Tahové klauzule nikdy nemění aktuálního hráče v herním stavu, ale pouze rozmístění tokenů do zón. Střídání hráčů bude popsáno pomocí ternárních klauzulí, které podle vstupního stavu upraví aktuálního hráče na dalšího hráče v pořadí a výsledek vrátí ve výstupním stavu.

Definice (Klauzule střídání hráčů)**.** *Klauzule střídání hráčů je klauzule následujícího tvaru:*

$$N(s_{in}, s_{new}, s_{out}) \leftarrow K_1, \ldots, K_n$$

kde

s_{in} *je původní vstupní herní stav*

s_{new} *je výstupní herní stav z tahové klauzule*

s_{out} *je je upravený výstupní herní stav z tahové klauzule o aktuálního hráče*

$K_1 \ldots K_n$ *jsou klauzule,* $n \in N$

Rovněž budeme pro herní stav potřebovat ověřit, zda v něm některý z hráčů nedosáhl vítězství, nebo nenastala remíza. I k tomuto úkolu poslouží klauzule označované jako vítězné. Prvním proměnná vítězné klauzule bude herní stav, druhá bude odpovídat vítěznému hráči. Pokud se jedná o remízu, bude místo hráče vrácen zvláštní symbol.

Definice (Vítězná klauzule)**.** *Vítězná klauzule je klauzule následujícího tvaru:*

$$V(s_{in}, p) \leftarrow K_1, \ldots, K_n$$

kde

$p \in P$*, kde* P *je množina hráčů*

$K_1 \ldots K_n$ *jsou klauzule,* $n \in N$

Předpokládáme, že každý hráč má k dispozici stejnou sadu tahových i vítězných klauzulí.

Počáteční herní stav bude složen z počátečního rozmístění tokenů do zón a začínajícího hráče.

Poznámka (Variabilní počet hráčů). Některé hry nabízejí variabilní počet hráčů. Pokud však máme stejnou hru, ale hrajeme ji s různým počtem hráčů, z formálního hlediska se jedná o odlišné hry. Například pro hru člověče nezlob se hráno s dvěma a čtyřmi hráči bude vypadat popis herním systémem zcela jinak.

Poznámka (Rozšíření jazyka logického programovaní). Všimněme si, že herní stav se skládá z n-tice a systému množin. S těmito objekty neumí čisté logické programování pracovat a pohybujeme se tedy v jeho rozšíření. Vlastní aplikaci pak budeme demonstrovat v jazyce prolog a místo těchto dvou objektů použijeme seznam.

Nyní jsme již připraveni uvést definici herního systému.

Definice (Herní systém)**.** *Herním systémem budeme rozumět uspořádanou osmici*

$$G = (K, P, T, Z, L, V, N, s), kde$$

K *je neprázdná konečná množina komponent*

P *je neprázdná konečná množina hráčů*

T *je neprázdná konečná množina tokenů,* $T \subset K$

Z *je neprázdná konečná množina zón,* $Z \subset K$

L *je konečná neprázdná množina tahových klauzulí*

N *je konečná neprázdná množina klauzulí střídání hráčů*

V *je konečná neprázdná množina vítězných klauzulí*

s *je počáteční herní stav*

2.1.1 Použití herního systému

Pokud chce hráč v herním stavu vykonat tah použije k tomu tahovou klauzuli následovanou klauzulí střídání hráče. Podmínkové klauzule zajišťují legálnost tahu v dané situaci. Klauzule efektu převedou herní stav.

Takto však můžeme zpracovat pouze jeden hráčův tah. Hodil by se nám nástroj, který dokáže provést celou posloupnost tahů a dokázal by ukončit vyhodnocení, pokud je herní stav vítězný pro některého z hráčů. Tím bude predikát $Proved$.

Definice (Predikát Proved)**.**

$$Proved(l, s_{in}, p) \leftarrow V(s_{in}, p)$$
$$Proved([T|l], s_{in}, p) \leftarrow T(s_{in}, s), N(s_{in}, s, s_{out}), Proved(l, s_{out}, p)$$

Pokud bychom chtěli pomocí herního systému implementovat interaktivní hru, museli bychom uvedený predikát $Proved$ rozšířit o možnost uživatelského vstupu a zlepšit uživatelskou přívětivost. Nyní by hráč musel ručně zadávat klauzule a navíc ani neví, jaké tahy má k dispozici.

Zadefinujeme interaktivní predikát $Hrej$. Ten pomocí $Print$ vypíše aktuální herní stav. Následně použitím $Bagof$ vygeneruje všechny možné legální herní stavy, do nichž lze přejít ze vstupního herního stavu pomocí některého z tahů. Tyto herní stavy jsou navíc opatřeny identifikátorem tahové klauzule, která k tomu byla využita, získané pomocí $Hash$. Přípustné herní stavy jsou spolu s identifikátory vypsány hráči na výstup. Ten na vstupu zvolí některý z identifikátorů a podle toho se vybere následují herní stav.

Definice (Predikát hrej)**.**

$$
\begin{aligned}
Hrej(s_{in}, p) \leftarrow & V(s_{in}, p) \\
Hrej(s_{in}, p) \leftarrow & Print(s_{in}), \\
& Bagof([s_t, id_t], T(s_{in}, s_{tmp}), N(s_{in}, s_{tmp}, s_t), \\
& Hash(T(s_{in}, s_{tmp}), id_t), list_t), \\
& Print(list_t), \\
& Read(id), \\
& Select(id, list_t, s_{out}), \\
& Hrej(s_{out}, p)
\end{aligned}
$$

Predikát $Hrej$ tvoří rámec pro tvorbu interaktivních her v logickém programování.

2.1.2 Převod herního systému na herní automat

Pomocí herního systému umíme pro danou posloupnost tahů rozhodnout, zda je vítězná pro hráče. Herní systém tedy disponuje stejnou schopností jako herní automat a je na čase zjistit, jaký je mezi nimi vztah. Uvedeme si větu, která propojí svět herních systémů a herních automatů.

Věta (Převod herního systému na herní automat)**.** *Libovolný herní systém lze převést na herní automat*

Algoritmus převodu

Popíšeme si postup převodu herního systému na herní automat.

Nechť $G = (K, P, T, Z, F, L, V, s)$ je herní systém. Sestrojíme k němu herní automat $A = (S_G, P_G, T_G, W_G, s_G, \omega_G, \tau_G, \lambda_G)$, který bude popisovat stejnou hru.

Množiny T_G, W_G budou na počátku prázdné. Převod množiny hráčů a počátečního stavu je triviální: $P_G \leftarrow P$ a $s_G \leftarrow s$. Do množiny S_G vložíme počáteční herní stav s. Tím je inicializace ukončena.

Budeme definovat rekurzivní proceduru $conversion(s_{in})$, kde proměnná s_{in} je vstupní herní stav.

Dále budeme potřebovat nástroj, který nám pro danou herní situací zjistí všechny možné herní situace, do nichž se lze pomocí tahů hráče dostat. Pro tento účel zavedeme funkci $generate(s_{in})$. Ta pro vstupní herní stav vrátí seznam dvojic,

obsahující instanci tahové klauzule a výstupní herní stav získaný aplikací příslušné tahové klauzule a klauzule střídání hráče. V jazyce prolog k podobnému účelu slouží procedura $bagof$.

V prvním kroku procedura $conversion$ určí pro vstupní herní stav hráče na tahu. Pro stav $s_{in} \equiv (p, Z_T)$ doplníme funkci výběru hráče ω_G tak, že $\omega_G(s_{in}) \leftarrow p$.

Užitím funkce $winnable$ ověříme, zda stav s_{in} nesplňuje některou vítěznou klauzuli. V kladném případě přidáme herní stav do W a funkcí $getWinner$ zjistíme hráče, který v tomto stavu vyhrává. Jestliže hráč není uveden, jedná se o remízový stav. Podle výsledku funkce $getWinner$ upravíme vítěznou funkci $\lambda_G(s_{in}) = getWinner(s_{in})$.

Následně pomocí funkce $generate(s_{in})$ vygenerujeme seznam dvojic tvaru $[t(s_{in}, s_{out}), s_{out}]$. Tento seznam budeme postupně procházet a pro každou dvojici provedeme následující operace. Instanci tahové klauzule přidáme do množiny T_G a výstupního herní stav s_{out} do S_G . Tahovou funkci τ_G doplníme pro dvojici tak, že $\tau_G(s_{in}, t(s_{in}, s_{out})) \leftarrow s_{out}$. Nakonec rekurzivně zavoláme proceduru $conversion(s_{out})$.

Listing 2.1: Procedura převodu

```
procedure conversion(s_in):
    ω_G(s_in) ← getPlayer(s_in)

    if winnable(s_in):
        λ_G(s_in) ← getWinner(s_in)
        return

    list ← generate(s_in)

    for [t(s_in, s_out), s_out] in list:
        T_G ← t(s_in, s_out)
        S_G ← s_out
        τ_G(s_in, t(s_in, s_out)) ← s_out
        conversion(s_out)
```

Důkaz. Indukcí bychom ověřili, že mají stejnou množinu všech partií. □

Věta (Dobrá definovatelnost převodu)**.** *Pokud hra popsaná herním systémem byla dobře definovaná, je i po převodu na herní automat dobře definovaná.*

Důkaz. Bez újmy na obecnosti předpokládáme, že všechny stavy jsou dosažitelné. Pokud byla hra popsaná herním systémem ukončitelná, znamená to, že v každé nekonečné posloupnosti musel být tah, jehož výstupem byl herní stav, pro něhož platila vítězná klauzule. Algoritmus projde všechny možné herní stavy a v těch, pro které platí vítězná klauzule, ukončí svou činnost a označí je jako vítězné v herním automatu.

Jestliže byla hra popsaná herním automatem totálně definovaná, musela být v každé situaci uplatnitelná nějaká tahové klauzule. Algoritmus induktivně prochází herní stavy a pro každý z nich vygeneruje skrze tahové klauzule všechna možná pokračovaní a ty také projde.

Pro slabě vítěznou hru platí, že muselo během vítězné posloupnosti tahů dojít ke střídání hráčů. Střídání hráčů v herních stavech zajistí funkce $generate$ a funkcí

getPlayer ji zapíšeme do vítězného herního automatu. Hra popsaná výsledným herním automatem je tedy dobře definovaná. □

Poznámka (O převodu). Obrácená věta neplatí, neboť herní systém je popsán prostředky mnohem vyšší úrovně, než kterými disponuje herní automat, a nejsme schopni zajistit jednoznačný převod.

Praktické využití tohoto převodu je kvůli paměťovým nárokům herního automatu minimální. Z teoretického hlediska je převod o poznání zajímavější a můžeme tak aplikovat naše poznatky o herních automatech i na hry popsané herním systémem.

2.2 Definice stochastického herního systému

Abychom byli důslední, zavedeme si ve zkratce i stochastickou variantu herního systému. Budeme uvažovat pouze jeden generátor náhody, který bude provádět náhodný pokus nad danou množinou náhodných jevů. Omezíme výsledky náhodného pokusu pouze na elementární jevy.

Stochastický herní systém musí umět uchovat výsledek náhodného pokusu a dále se jím řídit. K tomu využijeme tokenů a zón. Pro každý elementární náhodný jev zavedeme jednu zónu, jeden společný značkovací token, který bude umístěn do příslušné zóny podle výsledku náhodného pokusu, a jednu výchozí zónu, kde bude token umístěn před provedením náhodného pokusu. Šlo by to udělat i obráceně. Zavést značkovací token pro každý elementární jev a jen dvě pomocné zóny.

Náhodný pokus budeme realizovat pomocí klauzulí. Budeme je nazývat randomizační. Tyto klauzule budou stejně jako tahové klauzule obsahovat podmínkové klauzule, ale jedna klauzule efektu bude vykonávat náhodný pokus. Podle výsledku náhodného pokusu přesune klauzule značkovací token z výchozí zóny do zóny odpovídající náhodnému jevu.

Z hlediska predikátů $Proved$ a $Hrej$ není potřeba mezi randomizačními a tahovými klauzulemi rozlišovat. To je důležité až při převodu na herní automat, abychom věděli, zda stav spadá do množiny D nebo N. Uvedeme modifikovaný algoritmus převodu.

Funkce *generate_randomize* pro daný stav vrátí všechny možné výsledky náhodných pokusů, které lze v daném stavu vykonat užitím randomizačních klauzulí získat.

Listing 2.2: Procedura převodu

```
procedure conversion(s_in):
    ω_G(s_in) ← getPlayer(s_in)

    if winnable(s_in):
        λ_G(s_in) ← getWinner(s_in)
        return

    list_T ← generate(s_in)

    if not empty(list_T):
        D_G ← s_in

    for [t(s_in, s_out), s_out] in list_T:
        T_G ← t(s_in, s_out)
        S_G ← s_out
        τ_G(s_in, t(s_in, s_out)) ← s_out
        conversion(s_out)


    list_R ← generate_randomize(s_in)

    if not empty(list_R):
        N_G ← s_in

    for [ν, s_out] in list_R:
        S_G ← s_out
        κ_G(s_in, ν)) ← s_out
        conversion(s_out)
```

3. Analýzy her

Abychom demonstrovali použitelnost herního systému, provedeme rozbor několika klasických her. K popisu her nebudeme přímo používat prostředků formalismu herního systému, ale místo něj použijeme jazyk prolog. Ten obsahuje řadu vlastností, které nám usnadní práci a navíc jeho syntax je povědomější než jazyk herního systému.

3.1 Od herního systému k prologu

Hráči budou uložení v unárním predikátu *players*. Zóny budou definovány jako unární predikáty obsahující seznam, do kterého se budou vkládat tokeny.

Herní stav *state* zadefinujeme jako binární predikát obsahující aktuálního hráče a seznam zón. Seznam zón odpovídá množinovému systému rozmístění tokenů do zón. Klauzule budou uloženy ve vnitřní databázi prologu a není pro ně potřeba zavádět žádný zvláštní predikát. Tahové klauzule budeme označovat predikátem *turn*, vítězné klauzule predikátem *win* a klauzule střídání hráčů *next*.

Z prostředků jazyka prolog budeme často využívat nedeterminismu a pattern-matching. Dále nebudeme v rámci tahových klauzulí pevně dodržovat rozdělení na podmínkové klauzule a klauzule efektu a budeme je libovolně míchat, kde nám to syntax dovolí.

3.2 Návod na analýzu her

Pravidla her bývají nejčastěji dostupná a zaznamenaná v přirozeném jazyce. Krátce si popíšeme, jak takto zapsaná pravidla převést do formálního popisu, a upozorníme na některá úskalí.

Jak již bylo uvedeno dříve, pravidla deskových her mají dvojí úlohu. Jednak popisují strukturu hry. Říkají, kolik hráčů se hry účastní, s jakými tokeny a na kterých zónách budou hrát. V druhém případě pravidla definují tahy, pomocí nichž hráč mění herní situaci a přesouvá tokeny mezi zónami.

Všimněme si, že herní systém tak, jak jsme jej zavedli, nepoužívá pravidel prvního typu k popisu vlastností tokenů a zón a relací mezi nimi. Ty jsou ve skutečnosti nepřímo uvedeny až v rámci podmínkových klauzulí. Podmínkové klauzule hlídají, aby po změně herního stavu zůstaly zachovány všechny požadované vlastnosti a relace. Samozřejmě musí platit, že tyto požadavky jsou splněny již v počátečním stavu.

Například u hry piškvorky platí vlastnost, že v jednom políčku nesmí být víc než jeden křížek nebo kolečko. Na začátku hry jsou všechna pole prázdná a podmínka je splněna. Při tazích podmínkové klauzule nepřipustí, aby bylo možné do obsazeného pole vložit další token. Tím zůstane zachována požadovaná vlastnost.

Tento fakt je potřeba zohledňovat při návrhu tahových klauzulí a doplnit je korektními podmínkovými klauzulemi, aby po aplikaci zůstala hra v konzistentním stavu.

Při psaní vítězných klauzulí musíme být opatrní, zejména při popisu remíz. Remízy patří k nejkomplikovanějším částem her, a to nejen ve formálním popisu,

ale už při vlastním návrhu a tvorbě pravidel deskové hry. Musí být zajištěno, aby prohrávající hráč nezneužil remízy k zmaření očividného vítězství jiného hráče, ale zároveň musí zaručit ukončení hry, která nevede k vítězství žádného hráče. Například ve hře v šachy se stále objevují nové poznatky, kdy za použití počítačů byly vytvořeny šachové koncovky, v nichž příprava k matu trvala déle než padesát tahů, aniž by byl sebrán kámen nebo taženo pěšcem. Při formálnímu popisu hry nesmíme zapomínat na vítězné klauzule představující remízu, neboť hrozí, že hru nebude možné ukončit.

Řada skutečností nebývá v pravidlech explicitně uvedena, ale při důsledné analýze je musíme vzít v potaz a zahrnout je do formálního popisu.

Prvním takovým příkladem jsou zdrojové a vyřazovací zóny. Od rozmístění tokenů do zón požadujeme, aby mělo sémantiku funkce a každý token byl umístěn v nějaké zóně. To znamená, že ve hře piškvorky musí být křížky a kolečka na začátku hry uloženy v nějaké zvláštní zóně předtím, než budou zapsány na hrací desku. Těmto zónám se často říká zdrojové. Podobná situace platí například i pro vyřazené šachové figury, které se přesouvají do vyřazovací zóny.

Tokenů a zón se často využívá pro uložení nějaké stavové informace ve hře. Například ve hře v šach si budeme chtít zapamatovat, zda již hráč použil rošádu. Pro každého hráče zavedeme dvě speciální zóny a jeden token. Pokud hráč použije rošádu, přesuneme token z první zóny do druhé. Stejný princip můžeme využít, když si chceme v nějaké karetní hře zapamatovat trumfovou barvu. Pro každou barvu zavedeme jednu zónu, plus navíc jednu zdrojovou a budeme mezi nimi přesouvat značkovacím tokenem. Takto lze například počítat i bodové hodnocení ve hře. Do nějaké vyhrazené zóny budeme přesouvat počet tokenů odpovídající počtu získaných bodů.

3.3 Příklady her

Základní principy formálního popisu předvedeme na jednoduché poziční hře tic tac toe. Ze světa karetních her jsme připravili pasiáns a hry obsahující náhodu budou zastoupeny člověče nezlob se.

3.3.1 Tic tac toe

Jedná se o poziční hru dvou hráčů probíhající na čtvercové sítí o rozměrech tři krát tři. Hráči se střídají po tazích a každý z nich vkládá do neobsazeného herního pole jeden svůj token. Hra končí, jakmile jeden z hráčů složí ze svých tokenů řádku, sloupec nebo diagonálu.

Zadefinujeme hráče.

```
player(p_x).
player(p_o).
```

Popíšeme počáteční herní stav. Budeme potřebovat devět zón, kam se umisťují tokeny a jednu zdrojovou (z_x a z_o) zónu pro každého hráče. Každý hráč bude mít k dispozici pět tokenů.

```
state(p_x, [z_1_1([]), z_1_3([]) ,z_1_3([]),
    z_2_1([]), z_2_3([]) ,z_2_3([]),
    z_3_1([]), z_3_3([]) ,z_3_3([]),
```

```
    %zdrojove zony
    z_x(x_1, x_2, x_3, x_4, x_5),
    z_o(o_1, o_2, o_3, o_4, o_5)]):-player(p_x).
```

Nyní potřebujeme popsat možné tahy hráče. Ze zdrojové zóny vybereme první token a přidáme ho do prázdné zóny na hrací desce.

```
turn(state(p–P, Z_in), state(p–P, Z_out)):-
    %vybereme cilovou prazdnou zonu
    select(z_X_Y([]), Z_r, Z_s),

    %vybereme zdrojovou zonu
    select(z–P(T|L), Z_in, Z_r),

    %pridame token do cilove zony
    Z_out=[z_X_Y([T])|z–P(L)|Z_s].
```

Střídaní hráčů je v tomto případě triviální.

```
next(S, state(p–x, Z_out), state(p–o, Z_out)).
next(S, state(p–o, Z_out), state(p–x, Z_out)).
```

Nakonec musíme ještě zadefinovat vítězné podmínky. K tomu budeme potřebovat predikáty *row*, *column*, *diagonal* a *full*.

```
%vitezne klauzule
win(state(p–P, Z_in), p–P):-row(Z_in).
win(state(p–P, Z_in), p–P):-diagonal(Z_in).
win(state(p–P, Z_in), p–P):-column(Z_in).

%remizova klauzule
win(state(p–P, Z_in), p–P):-full(Z_in), \+ row(Z_in),
                            \+ column(Z_in), \+ diagonal(Z_in).
```

3.3.2 Pasians

Uvedeme variantu bez počítaní bodů. Jedná se o hru jednoho hráče s poměrně komplexním systém zón.

Pravidla hry

Krátce se seznámíme s variantou pravidel, která byla v programu implementována. Při pojmenování struktur a predikátů jsme vycházeli z anglické terminologie.

Hrajeme s 52 kartami - s 13 figurami (ranks: 2-10, jack, queen, king, ace) a 4 barvami (suits: heart, diamont, clubs, spades).

Herní plocha je rozdělena do 4 zón. Jsou to: stack, waste, foundation a tableau.

Ve stack jsou uloženy karty, které nám zbyly po rozdávaní lícem dolů. Ze stack obracíme vrchní kartu do waste. Foundation obsahuje čtyři hromádky karet, a to pro každou barvu jednu. V hromádce jsou karty uspořádány podle hodnot figur s tím, že se začíná od ace, následuje 2 a končíme king. Uspořádání karet ve foundation bude popisovat predikát *foundat_order*.

Tableau obsahuje 7 hromádek karet. Každá hromádka se skládá ze dvou částí. V horní jsou karty skryté a v dolní jsou viditelné a uspořádány sestupně podle

hodnot figur a musí se střídat červená a černá barva karet. Uspořádání karet ve foundation bude popisovat predikát *tableau_order*.

Hra končí, když umístíme všech 52 karet do foundation.

Ve hře je celkem 11 pravidel popisujích přesuny karet mezi hromádkami.

1. Přesun jedné karty ze stack do waste.
2. Pokud je stack prázdný, můžeme do něj překlopit waste.
3. Přesun karty ze skryté části tableau do viditelné části tableau. Viditelná část musí byt prázdná.
4. Přesun karty z waste do foundation. Musí být zachováno uspořádání karet.
5. Přesun z tableau do foundation. Musí být zachováno uspořádání karet.
6. Přesun z waste do prázdné tableau. Takto lze přesunout pouze krále.
7. Přesun z foundation do prázdné tableau. Takto lze přesunout pouze krále.
8. Přesun z foundation do tableau. Musí být zachováno uspořádání karet.
9. Přesun z waste do tableau. Musí být zachováno uspořádání karet.
10. Přesun skupiny karet z jedné hromádky v tableau do jiné hromádky v tableau. Musí být zachováno uspořádání karet.
11. Přesun skupiny karet z jedné hromádky v tableau do prázdné hromádky v tableau. Skupinka musí končit kartou king.

Analýza hry

Zavedeme pomocné predikáty.

suit popisuje barvy karet.

```
suit(c). suit(d). suit(h). suit(s).
```

rank popisuje figury/hodnoty karet.

```
rank(2). rank(3). rank(4). rank(5). rank(6). rank(7).
rank(8). rank(9). rank(10). rank(j). rank(q). rank(k). rank(a).
```

red/black popisuje barvy barev karet

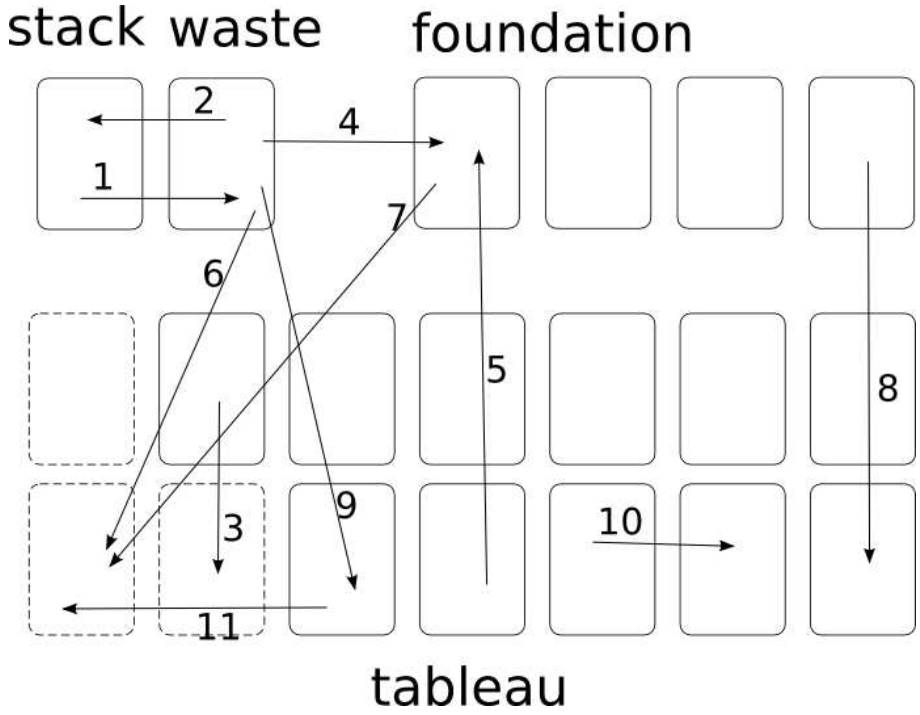
```
red(h). red(d).
black(c). black(s).
```

cards popisuje hrací kartu jako dvojici barvy a figury

```
cards(S–R):- suit(S), rank(R).
```

new_deck vygeneruje 52 karet potřebných ke hře, tedy seznam tokenů.

```
new_deck(Deck):- bagof(C, cards(C), Deck).
```



Obrázek 3.1: Přesuny karet mezi hromádkami

inverse_color zjisti, zda barvy jsou inverzni

```
inverse_color(A,B):-red(A),black(B);red(B),black(A).
```

tableau_order je predikát uspořádání karet v tableu. Abychom si ušetřili práci, implementovali jsme relaci pomoci seznamu. Matematicky hezčí by bylo, kdybychom popsal vzájemnou relaci prvků pomoci predikátu.

```
tableau_order(S1-R1,S2-R2,[R1,R2|_]):-
    cards(S1-R1),cards(S2-R2),inverse_color(S1,S2).

tableau_order(S1-R1,S2-R2,[X,Y|F]):-
    R1\==X,R2\==Y,inverse_color(S1,S2),
    tableau_order(S1-R1,S2-R2,[Y|F]).

tableau_order(A,B):-
    tableau_order(A,B,[2,3,4,5,6,7,8,9,10,j,q,k,a]).
```

foundat_order je predikát uspořádání karet ve foundation.

```
foundat_order(S-R,Sf-Rf,[Rf,R|_]):-
    cards(S-R),cards(Sf-Rf),suit(S)==suit(Sf).

foundat_order(S-R,Sf-Rf,[X,Y|F]):-
    Rf\==X,R\==Y,suit(S)==suit(Sf),
    foundat_order(S-R,Sf-Rf,[Y|F]).

foundat_order(S-R,Sf-Rf):-suit(S)==suit(Sf),
    foundat_order(S-R,Sf-Rf,[a,2,3,4,5,6,7,8,9,10,j,q,k]).
```

Ve hře je pouze jediný hráč.

```
player(p).
```

Popíšeme počáteční herní stav. Budeme definovat celkem dvacet herních zón, do kterých rozmístíme padesát dva karet.

```
state(p,Zin):-
    new_deck(Deck),

    %pripravime si karty do top tableau
    [TB1|Deck1]=Deck,[TB2|Deck2]=Deck1,
    [TB3|Deck3]=Deck2,[TB4|Deck4]=Deck3,
    [TB5|Deck5]=Deck4,[TB6|Deck6]=Deck5,
    [TB7|Deck7]=Deck6,

    %pripravime si karty do bottom tableau
    [TT2|Deck8]=Deck7,
    length(TT3,2),
    append(TT3,Deck9,Deck8),
    length(TT4,3),
    append(TT4,Deck10,Deck9),
    length(TT5,4),
    append(TT5,Deck11,Deck10),
    length(TT6,5),
    append(TT6,Deck12,Deck11),
    length(TT7,6),
    append(TT7,Deck13,Deck12),

    Zin=[w([]),1-tt([]),2-tt([TT2]),3-tt(TT3),
         4-tt(TT4),5-tt(TT5),6-tt(TT6),7-tt(TT7),
         1-tb([]),2-tb([TB2]),3-tb(TB3),
         4-tb(TB4),5-tb(TB5),6-tb(TB6),7-tb(TB7),
         s(Deck13),
         1-f([]),2-f([]),3-f([]),4-f([])]:-player(p).
```

Uvedeme pravidla pro přesun karet mezi hromádkami.

```
%1. presun karty z Stack do Waste
turn(state(p,Zin),state(p,[w([S|L])|s(Ss)|Zs])):-
    select(s([S|Ss]),Zin,Zr),
    select(w([L]),Zr,Zs).

%2. Waste na empty stack
turn(state(p,Zin),state(p,[w([])|s(Ls)|Zs])):-
    select(w(L),Zin,Zr),
    select(s([]),Zr,Zs),
    reverse(L,Ls).

%3. top tableau -> bottom tableau
turn(state(p,Zin),state(p,Zout)):-
    select(N-tt([D|L]),Zin,Zr),
    select(N-tb(K),Zr,Zs),
    Zout=[N-tt(L)|N-tb([D|K])|Zs].
```

```
%4. presun karty z waste na foundation
turn(state(p,Zin),state(p,[w(L)|N-f([W|X])|Zs])):-
    select(w([W|L]),Zin,Zr),
    select(N-f(X),Zr,Zs),
    foundat_check(W,X).

%5. presun karty z tableau na foundation
turn(state(p,Zin),state(p,[N-tb(Ds)|K-f([D|Fx])|Zs])):-
    select(N-tb([D|Ds]),Zin,Zr),
    select(K-f(Fx),Zr,Zs),
    foundat_check(D,Fx).

%6. presun z waste na prazdne tableau (jen kral)
turn(state(p,Zin),state(p,[N-tb([Su-k])|w(W)|Zs])):-
    select(w([Su-k|W]),Zin,Zr),
    select(N-tb([]),Zr,Zs),
    select(N-tt([]),Zs,_).

%7. presun foundation na prazdne tableau (jen kral)
turn(state(p,Zin),state(p,[N-tb([Su-k])|K-f(Fx)|Zs])):-
    select(N-tb([]),Zin,Zr),
    select(N-tt([]),Zin,_),
    select(K-f([Su-k|Fx]),Zr,Zs).

%8. presun z foundation na tableau
turn(state(p,Zin),state(p,[N-tb([Fh|D|Ds])|K-f(Ft)|Zs])):-
    select(K-f([Fh|Ft]),Zin, Zr),
    select(N-tb([D|Ds]),Zr,Zs),
    tableau_order(Fh,D).

%9. presun z waste na tableau
turn(state(p,Zin),state(p,[w(Ws),N-tb([W|D|Ds]),Zs])):-
    select(w([W|Ws]),Zin,Zr),
    select(N-tb([D|Ds]),Zr,Zs),
    tableau_order(W,D).

%10. preneseni hromadky z tableau na tableau
turn(state(p,Zin),state(p,[N-tb(D1n),K-tb(D2n)|Zs])):-
    select(N-tb(D1),Zin,Zr),
    select(Crd,D1,_),
    select(K-tb([D2|D2s]),Zr,Zs),
    tableau_order(Crd,D2),
    nth1(Nn,D1,Crd),
    split(Nn,D1,R),
    append(R,D1n,D1),
    append(R,[D2|D2s],D2n).

%11. presun tableau hromadky na prazdne tableau (jen krale)
turn(state(p,Zin),state(p,[N-tb(D1n),K-tb(R)|Zs])):-
    select(N-tb(D1),Zin,Zr),
```

```
    select(Su–k,D1,_),
    select(K–tb([]),Zr,Zs),
    select(K–tt([]),Zin,_),
    nth1(Nn,D1,Su–k),
    split(Nn,D1,R),
    append(R,D1n,D1).
```

Zadefinujeme vítěznou klauzuli. Hra končí, když jsou všechny karty ve foundation, tedy když jsou všechny ostatní zóny prázdné.

```
win(state(p, Zin), p):-
    select(1–tb([]),Zin,_),select(2–tb([]),Zin,_),
    select(3–tb([]),Zin,_),select(4–tb([]),Zin,_),
    select(5–tb([]),Zin,_),select(6–tb([]),Zin,_),
    select(7–tb([]),Zin,_),
    select(1–tt([]),Zin,_),select(2–tt([]),Zin,_),
    select(3–tt([]),Zin,_),select(4–tt([]),Zin,_),
    select(5–tt([]),Zin,_),select(6–tt([]),Zin,_),
    select(7–tt([]),Zin,_),
    select(w([]),Zin,_), select(s([]),Zin,_).
```

3.3.3 Člověče nezlob se

Analýzu stochastické poziční hry budeme demonstrovat na populární hře člověče nezlob se.

Pravidla hry

Krátce si popíšeme pravidla varianty hry, kterou budeme analyzovat. Uvažujeme variantu se čtyřmi hráči pojmenovaných podle barev *r, g, b, y*.

Ve hře existují celkem tři typy zón: počáteční domeček, cílový domeček a běžné pole. Některá běžná pole mají speciální účel. Jsou to startovací pole a cílové pole. Každý z hráčů má k dispozici svoji vlastní sadu zón typu: počáteční domeček, startovací pole, cílový domeček a cílové pole.

V popisu stochastického herního systému jsme uvedli, že je vhodné mít pro každý náhodný elementární jev speciální zónu a značkovací token. V případě člověče nezlob se jsou elementární jevy počty ok na kostce. Zjednodušíme si práci a zavedeme pouze jednu pomocnou zónu a do ní budeme přímo ukládat výsledek hodu s tím, že nulou označíme, že ještě nebyl proveden hod.

Pole spadající do počátečního domečku budeme reprezentovat jednou zónou. Ve hře bude celkem 41 zón (16 zón cílových domečků, 4 počáteční domečky, 20 standardních polí a 1 pomocná zóna pro zapamatování hodu).

Každý hráč má k dispozici čtyři figurky, které jsou na počátku hry umístěny v počátečním domečku.

Hráčův tah se bude skládat ze dvou částí: hodu kostkou a přesunu figurkou. Pokud hráči padla šestka, provede přesun figurkou a pak provede další tah.

Pro přesun figurky platí, že ji hráč nesmí posunout na pole, na kterém stojí jeho vlastní figurka. Dále platí, že pokud na cílovém poli stojí soupeřova figurka, je zajata a přesouvá se do svého počátečního domečku.

Podle výsledku hodu kostkou může hráč provést různé přesuny figurkou.

1. Pokud padlo šest, může přesunout figurku ze svého počátečního domečku na své startovní pole.

2. Posunout figurku o počet následujících standardních polí odpovídajících počtu ok na kostce. Pokud hráč projde přes svoje cílové pole, pokračuje do cílového domečku. Nesmí pokračovat za poslední pole svého cílového domečku.

Cílem hry je přesunout všechny své figurky do cílového domečku jako první.

Analýza hry

Zadefinujeme hráče.

```
player(r). player(g). player(b). player(y).
```

Popíšeme počáteční herní stav.

```
state(r,Zin):-
  Zin=[
    %pocatecni domecky
    r-home([r-1,r-2,r-3,r-4]),g-home([g-1,g-2,g-3,g-4]),
    b-home([b-1,b-2,b-3,b-4]),y-home([y-1,y-2,y-3,y-4]),

    %standardni pole
    field-1([]),field-2([]),field-3([]),field-4([]),
    field-5([]),field-6([]),field-7([]),field-8([]),
    field-9([]),field-10([]),field-11([]),field-12([]),
    field-13([]),field-14([]),field-15([]),field-16([]),
    field-17([]),field-18([]),field-19([]),field-20([]),

    %cilove domecky
    field-21([]),field-22([]),field-23([]),field-24([]),
    field-25([]),field-26([]),field-27([]),field-28([]),
    field-29([]),field-30([]),field-31([]),field-32([]),
    field-33([]),field-34([]),field-35([]),field-36([]),

    %pomocna zona
    pom([0])]:-player(r).
```

Zavedeme pomocné predikáty.

Označíme některá běžná pole jako startovací, konečná a jako konečné domečky.

```
r-start(1).g-start(5).b-start(10).y-start(15).

r-finish(20).g-finish(4).b-finish(9).y-finish(14).

r-end-1(21),r-end-2(22),r-end-3(23),r-end-4(24),
g-end-1(25),g-end-2(26),g-end-3(27),g-end-4(28),
b-end-1(29),b-end-2(30),b-end-3(31),b-end-4(32),
y-end-1(33),y-end-2(34),y-end-3(35),y-end-4(36),
```

Predikát **adjacent** popisuje, jak jsou navzájem propojeny různé zóny.

```
%bezna pole
adjacent(field-1(_),field-2(_)).adjacent(field-2(_),field-3(_)).
adjacent(field-3(_),field-4(_)).adjacent(field-4(_),field-5(_)).
adjacent(field-5(_),field-6(_)).adjacent(field-6(_),field-7(_)).
adjacent(field-7(_),field-8(_)).adjacent(field-8(_),field-9(_)).
adjacent(field-9(_),field-10(_)).adjacent(field-10(_),field-11(_)).
adjacent(field-11(_),field-12(_)).adjacent(field-12(_),field-13(_)).
adjacent(field-13(_),field-14(_)).adjacent(field-14(_),field-15(_)).
adjacent(field-15(_),field-16(_)).adjacent(field-16(_),field-17(_)).
adjacent(field-17(_),field-18(_)).adjacent(field-18(_),field-19(_)).
adjacent(field-19(_),field-20(_)).adjacent(field-20(_),field-1(_)).

%propojeni cilovych domecku
adjacent(field-21(_),field-22(_)).adjacent(field-22(_),field-23(_)).
adjacent(field-23(_),field-24(_)).

adjacent(field-25(_),field-26(_)).adjacent(field-26(_),field-27(_)).
adjacent(field-27(_),field-28(_)).

adjacent(field-29(_),field-30(_)).adjacent(field-30(_),field-31(_)).
adjacent(field-31(_),field-32(_)).

adjacent(field-33(_),field-34(_)).adjacent(field-34(_),field-35(_)).
adjacent(field-35(_),field-36(_)).
```

Predikát **move** popisuje pohyb figurky po polích podle hodu na kostce.

```
%F - oznaceni vstupniho pole
%H - hod na kostce
%P - hrac
%Fo vystupni pole

%pohyb nepokracuje do domecku
move(F,H,P,Fo):-\+ P-finish(F),H1 is H - 1,
    adjacent(field-F,field-FX),
    move(FX,H1,P,Fo).

move(F,H,P,Fo):-P-finish(F),H1 is H - 1,P-end-1(FX),
    move(FX,H1,P,Fo).

move(F,0,P,F).
```

Popíšeme klauzule na střídání hráčů. Jak již bylo napsáno, hráčský tah je rozdělen na hod kostkou a přesun figurky. Po hodu nesmí dojít k výměně hráče, ale až po přesunu figurky. Dále při střídání hráčů musíme hlídat, zda padlo šest, protože pak má stávající hráč tah navíc.

```
%probehl hod, hrac se nestrida
next(state(P, Zold), state(P, Znew), state(P, Znew)):-
    select(pom([0]),Zold,_),
    select(pom([X]),Znew,_),X\==0.

%probehl presun, ale nepadlo sest, hrac se strida
next(state(P, Zold), state(P, Znew), state(Ps, [pom([0])|Zs])):-
```

```
    select(pom([X]),Zold,_),
    select(pom([X]),Znew,Zs),X\==0,X\==6,
    order(P,Ps,[r,g,b,y]).

%padlo sest, hrac je znova na tahu
next(S, state(P, Znew), state(P, [pom([0])|Zs])):-
    select(pom([6]),Znew,Zs).

%vyber dalsiho hrace v poradi
order(A,B,[X|Y]):-A\==X,order(A,B,Y).
order(A,B,[X|Y]):-A==X,[B|_]=Y.
order(A,B,[A|[]]):-B=r.
```

Dále popíšeme vlastní hráčské tahy. Simulaci hodu kostkou budeme provádět pomocí funkce *random/1*.

```
%hod kostkou
turn(state(P,Zin),state(P,[pom([X])|Zs])):-
    X is random(6),
    select(pom([0]),Zin,Zs).

%nasazeni figurky na prazdne pole
turn(state(P,Zin),state(P,[P–home(Ts)|field–F([T])|Zss])):-
    select(pom([6]),Zin,_),
    select(P–home([T|Ts]),Zin,Zs),
    P–start(F),
    select(field–F([]),Zs,Zss).

%nasazeni figurky na pole obsazene spoluhracovou figurkou
turn(state(P,Zin),
    state(P,[P–home(Ts)|field–F([T])|R–home([R–N|Rs])|Zsss])):-
    select(pom([6]),Zin,_),
    select(P–home([T|Ts]),Zin,Zs),
    P–start(F),
    select(field–F([R–N]),Zs,Zss),R\==P,
    select(R–home(Rs),Zss,Zsss).

%presun figurky na volne pole
turn(state(P,Zin),state(P,[field–F([])|field–Fo([P–N])|Zss])):-
    select(pom([H]),Zin,_),H\==0,
    select(field–F([P–N]),Zin,Zs),
    move(F,H,P,Fo),
    select(field–Fo([]),Zs,Zss).

%presun figurky na obsazene pole
turn(state(P,Zin),
    state(P,[field–F([])|field–Fo([P–N])|R–home([R–K|Rs])|Zsss])):-
    select(pom([H]),Zin,_),H\==0,
    select(field–F([P–N]),Zin,Zs),
    move(F,H,P,Fo),
    select(field–Fo([R–K]),Zs,Zss),R\==P,
    select(R–home(Rs),Zss,Zsss).
```

Zadefinujeme vítěznou klauzuli. Hra končí, když některý z hráčů přesune všechny své figurky do cílového domečku.

```
win(state(P, Zin), P):-
    P-end-1(FA),select(field-FA(A),Zin,_),A\=[],
    P-end-2(FB),select(field-FB(B),Zin,_),B\=[],
    P-end-3(FC),select(field-FC(C),Zin,_),C\=[],
    P-end-4(FD),select(field-FD(D),Zin,_),D\=[].
```

4. Srovnání s dalšími systémy

4.1 Game Description Language

Nejznámějším počinem v oblasti formalizace deskových her je Game Description Language (GDL). GDL je součástí projektu General Game Playing Stanfordské University v Kalifornii.

General Game Playing je návrh univerzálního programu umělé inteligence, který by dokázal úspěšně hrát rozmanité hry. Řada dostupných programů je orientovaná pouze na jednu hru a obsahuje speciální jednoúčelové algoritmy, které nemohou být přenášeny mezi různými hrami. Například program pro hrání šachu nelze použít pro hraní dámy. General Game Playing by měl, pokud je dobře navrhnut, poskytnout univerzální řešení.

V naší práci využíváme pro reálnou aplikaci jazyk prolog. GDL je rozšíření deklarativního logického jazyka datalog. Datalog byl navržen jako dotazovací jazyk pro deduktivní databáze. Vznikl spojením světů logického programovaní a databází. Jeho syntax je podmnožinou prologu. Od prologu se liší tím, že zakazuje složené termy v argumentech predikátu, každá proměnná uvedená v hlavě klauzule musí být uvedena i v těle a obsahuje určitá omezení na použití negace a rekurze. Díky těmto omezením je datalog výrazně rychlejší než prolog a je zaručeno, že každý program skončí.

Syntax jazyka GDL je podobná jazyku scheme. GDL využívá pro popis stavového modelu hry následující relace: *role*, *init*, *true*, *does*, *next*, *legal*, *goal*, a *terminal*. Relace *role* se používá pro definování hráčů ve hře. Pomocí *init* se určí fakta, která platí v počátečním stavu hry. Užitím relace *true* stanovíme, že nějaký fakt platí v aktuálním stavu hry. Relace *next* stavuje fakt, který bude platit v následujícím herním stavu. Pomocí *legal(player,move)* ověříme, zda hráč muže vykonat pohyb v daném stavy hru. Tah aktuálního hráče a změna stavu hry se provede predikátem *does(player, move)*. Relace *goal* se používá ke stanovení a ohodnocení cílů, které hráči během hry získávají. Konečné stavy hry jsou popsány relací *terminal*.

GDL je určen především pro strojové použití a umělou inteligenci. Pro využití na formální analýzu her však není tak intuitivní a expresivní jako jazyk herního systému. Nerozlišuje herní díly na zóny a tokeny a střídání hráču je součástí hráčských tahů. Jako velmi zajímavou volbou se jeví výběr datalogu místo prologu, zejména kvůli rychlosti a zaručenému ukončení všech programů.

Více se lze o tomto systému dozvědět na stránce: `http://games.stanford.edu/`.

4.2 Zillions of Games

Zillions of Games je komerční herní systém systém vyvinutý Jeffem Malletetem a Markem Leflerem v roce 1998.

Pravidla her nejsou popsána logickým jazykem jako v předchozím případě, ale funkcionálním jazykem podobném LISP, založeném na s-výrazech, pojmenovaný Zillions rule language. Součástí tohoto jazyka je i popis grafického znázornění hry.

Zillions of Games rovněž poskytuje prostředky umělé inteligence a umožňuje agentům zhostit se role hráče. Ohodnocovací funkce se odvozuje automaticky

z pravidel hry podle pohyblivosti dílu, ze struktury hrací desky a z herních cílů. Používá podobné algoritmy jako šachové programy: alfa-beta ořezávaní s přerovnáváním tahů, transpoziční tabulky atd.

Nás však zajímá, jak vypadá formální popis hry v tomto systému. Důležité části popisu jsou: *Players*, *Turn order*, *Board definition*, *Piece definition*, *Board setup* a *Goal of the game*. *Players* definuje seznam jmen hráčů, kteří se budou účastnit hry. *Turn order* specifikuje pořadí střídání hráčů jako posloupnost hráčských jmen a operátorů pro opakování. *Board definition* popisuje strukturu a vlastnosti hrací desky. *Piece definition* popisuje herní díly a jejich vlastnosti. Součástí definice jsou i možné pohyby a tahy těchto dílů. *Board setup* udává počáteční rozestavení hry. *Goal of the game* stanovuje vítězné podmínky hry.

V tomto systému už jsou odděleny definice hrací desky a herních dílů. Přesto však princip zón v našem formalismu zaručuje větší volnosti při popisování deskové hry. Náš herní systém má jinou filozofii k udržení konzistence hry. Explicitně nepopisuje vlastnosti herních dílů a herní desky, ale stanovuje podmínky, za kterých mohou být provedeny tahy tak, aby hra zůstala v konzistentním stavu. Další slabinou formátu Zillions rule language je popis střídání hráčů, který se nedokáže vypořádat s hrami, kde není předem dané střídání hráčů, jako třeba v bridge nebo whist.

Více na `http://www.zillions-of-games.com/`.

4.3 Metagame

Metagame je jeden z prvních počinů na poli General Game Playing. Byl vytvořen v roce 1992 panem Barney Pell. Zmiňujeme jej zde pouze z historických důvodů. Více na `http://www.barneypell.com/games-research/`.

Závěr

Naším cílem bylo vytvoření formalismu, který bude vhodný pro zachycení specifických rysů deskových her, přimočarý na použití a dostatečně výstižný, aby dokázal popsat široké spektrum her.

Podařilo se vytvořit intuitivní aparát umožňující formální analýzu široké škály deskových her, využívající poznatků matematiky a teoretické informatiky, zejména z oblasti logického programování. Byla představena původní myšlenka pro zachycení struktury deskových her spočívající v rozdělení herních prvků na tokeny a zóny a v popisu jejich vzájemného vztahu. V rámci práce jsme studovali i články z oblasti general game playing. Žádný z formálních popisu uvedených v těchto materiálech nebyl tak pružný a nezachycoval detaily hry do takové hloubky jako právě herní systém.

Použitelnost herního systému byla demonstrována na několika klasických hrách. Aby se dále prověřily vlastnosti formálního modelu, je zapotřebí provést rozbor dalších a hlavně komplexnějších her. Věříme, že za pomoci herního systému lze analyzovat libovolnou hru splňující požadované předpoklady.

Popsaný formalismus umožňuje deklarativní popis her a v práci je představen základ aplikačního rámce pro tvorbu interaktivních her v prologu.

Nabízí se několik směrů, kterými lze práci dále rozšiřovat. Po vzoru konkurenčních systémů je možné deklarativní popis obohatit o prvky umělé inteligence (ohodnocovací funkce, alfa-beta ořezávání), aby bylo možné hrát nejen proti lidským protivníkům, ale i proti agentům.

Dále je možné v reálných aplikacích místo prologu navrhnout a použít nějaký doménově specifický jazyk, který byl optimalizovaný na konkretní potřeby deskových her. Dalo by se uvažovat o rozšíření jazyka datalog.

V práci je popsaný návod pro převod her z přirozeného jazyka do formálního popisu. Je přímo nasnadě využít obrácený převod a z formálního systému generovat pravidla v různých přirozených jazycích. Pokud bychom se posunuli ještě o úroveň výše, mohli bychom jazyk formálního systému rozšířit tak, aby uměl generovat grafické znázornění hry a uživatelské rozhraní.

Rovněž se nabízí možnost zkoumat tvar klauzulí a zjišťovat z nich vlastnosti her.

Věříme, že poznatky z práce ocení i vývojáři a návrháři her. Nabídli jsme jim nástroj, který jim umožní získat vhled do komplexní struktury deskových her, formálně uvažovat nad pravidly a odhalovat vady v návrhu hry. Pomocí navrženého formalismu lze navrhnou systém na automatické testování her.

Seznam použité literatury

[Genesereth – Love(2005)Genesereth, Love] GENESERETH, M. – LOVE, N. General game playing: Overview of the AAAI competition. *AI Magazine*. 2005, 26, s. 62–72.

[Hopcroft – Ullman(1999)Hopcroft, Ullman] HOPCROFT, J. – ULLMAN, J. *Introduction to automata theory, languages, and computation*. Addison-Wesley series in computer science. : Addison-Wesley, 1999. ISBN 9780201029888.

[Love et al.(2008)Love, Hinrichs, Haley, Schkufza„ Genesereth] LOVE, N. et al. General Game Playing: Game Description Language Specification. 2008.

[Russell – Norvig(2010)Russell, Norvig] RUSSELL, S. – NORVIG, P. *Artificial intelligence: a modern approach*. : Prentice Hall, 2010. ISBN 9780136042594.

[Salen – Zimmerman(2004)Salen, Zimmerman] SALEN, K. – ZIMMERMAN, E. *Rules of play: game design fundamentals*. : MIT Press, 2004. ISBN 9780262240451.

[Spivey(1996)] SPIVEY, M. *An introduction to logic programming through Prolog*. : Prentice-Hall, Inc., 1996. ISBN 0-13-536047-1.