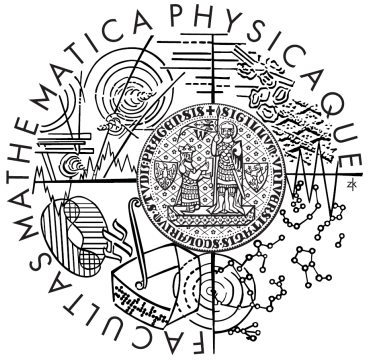
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

# DIPLOMOVÁ PRÁCE




Martin Suda

## Efektivní algoritmy ověřování cílů v počítačových hrách

Katedra teoretické informatiky a matematické logiky

Vedoucí diplomové práce: RNDr. Jan Hric
Studijní program: Informatika, teoretická informatika

Děkuji všem, kteří mě při psaní této práce podpořili, zejména pak Dr. Hricovi za odborné vedení a cenné připomínky a Ondrovi Fialkovi za pomoc s úpravou textu.

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

V Praze dne Martin Suda

# Obsah

Název práce: Efektivní algoritmy ověřování cílů v počítačových hrách
Autor: Martin Suda
Katedra (ústav): Katedra teoretické informatiky a matematické logiky
Vedoucí diplomové práce: RNDr. Jan Hric
E-mail vedoucího: Jan.Hric@mff.cuni.cz

**Abstrakt:** Tato práce se zabývá neuniformními metodami prohledávání stromů her dvou hráčů s úplnou informací. Konkrétně je zkoumána neuniformita založená na hrozbách realizovaná v podobě algoritmů lambda search a dual-lambda search. Hrozby, definované jakožto útočné tahy, po kterých, když se obránce svého tahu vzdá, bude již s jistotou poražen, umožňují zredukovat prohledávaný prostor při zachování korektnosti výsledku.

Práce dále popisuje novou metodu konstrukce tzv. zón relevance, tj. seznamů tahů resp. míst na hrací ploše, které jedině mohou mít vliv na výsledek. S pomocí těchto zón je možné výše jmenované algoritmy značně urychlit.

V práci jsou též popsány tři hry, AtariGo, Hex a Piškvorky, a je rozebrána jejich vhodnost pro aplikaci zkoumané metody. Součástí práce je též implementace metody pro tyto hry s využitím známých technik (transpoziční tabulky, history heuristika).
**Klíčová slova:** hry s úplnou informací, lambda search, zóny relevance

Title: Effective Algorithms for Verifying Goals in Computer Games
Author: Martin Suda
Department: Department of Theoretical Computer Science and Mathematical Logic
Supervisor: RNDr. Jan Hric
Supervisor's e-mail address: Jan.Hric@mff.cuni.cz

**Abstract:** In the present work we study non-uniform methods for searching game trees of two player games with perfect information. Particularly the non-uniformity based on threats as realized in lambda search and dual-lambda search algorithms is investigated. Threats, defined as such attack moves that if followed by a pass from the defender result in his certain loss, allow for a reduction of the search space while guaranteeing correctness at the same time.

The work then describes a new method for construction of so called relevancy zones, a list of moves or places on the game desk that can only have influence on the result of the problem in question. Using these zones it is possible to speed up the mentioned algorithms considerably.

In the work there are also described three games, AtariGo, Hex and Go-Moku, and their appropriateness with respect to the studied methods is analyzed. Part of the work is also an implementation of the methods for these games using known techniques (transposition tables, history heuristic).
**Keywords:** games with perfect information, lambda search, relevancy zones

# Kapitola 1

# Úvod

Hry dvou hráčů s úplnou informací patří k jedné z klasických oblastí zájmu umělé inteligence. Představa o stroji, který by porazil člověka v Šachu, fascinovala mnohé již před několika staletími. S nástupem číslicových počítačů byly otevřeny dveře tvorbě samostatně hrajících programů. V roce 1950 publikoval Shannon článek [16], ve kterém popsal mechanismus pro tvorbu programu hrajícího šachy. Historicky první naimplementovaný herní program, který se konkrétně zabýval hrou dáma, je popsán v [14] z roku 1959. Od té doby pokračuje vytrvalý postup při zlepšování kvality hry počítačových oponentů a v dnešní době se již nejlepší programy pro některé hry mohou měřit se světovými mistry.

Za základní techniku, které se v hrajících programech používá, lze bez pochyby považovat prohledávání stromu hry založené na principu minimaxu ([13]). Prohledávají se typicky všechny možné průběhy hry do předem stanovené hloubky a hodnota dosažených pozic je získávána pomocí heuristické ohodnocovací funkce, která vyjadřuje odhad hráčových šancí na úspěch. Vyšší efektivity se pak dosahuje s použitím metody alfa-beta nebo jejích variant, které dokáží vyloučit z úvah některé části prohledávaného stromu při zachování stejného výsledku.

Ačkoliv byly objeveny mnohé podpůrné techniky, jak výše popsanou metodu hledání nejlepšího tahu dále urychlovat, ne v každé hře ji lze s okamžitým úspěchem použít. Např. u hry Go, prastaré deskové hry původem z Asie, přispívá její vysoký faktor větvení a také obtíže při návrhu spolehlivé ohodnocující funkce k tomu, že dosavadní nejlepší programy nepřesahují svými výkony průměrného hráče ([5]). Je jasné, že musí být objeveny a prozkoumány nové metody a přístupy pro tvorbu počítačových oponentů, aby se nové programy mohly v takových hrách, jako je Go, alespoň vyrovnat svým lidským protějškům.

Jednu z možností, jak problémy způsobené vysokým faktorem větvení

překonávat, představují neuniformní metody prohledávání stromu hry. Při jejich použití se v daném vrcholu stromu hry neprohledávají všechny možnosti, nýbrž se s pomocí jistého pravidla vyberou jen některé významné či perspektivní tahy, kterými se algoritmus dále zabývá. Narozdíl od přístupu nazývaného selective search, jednoho z rozšíření metody alfa-beta, ve kterém je výběr zkoušených tahů řízen heuristikou a potenciálně tedy může vést k nesprávným výsledkům, neuniformní metody si kladou za cíl spočítat skutečnou hodnotu dané pozice, tj. rozhodnout zda může při optimální hře obou hráčů dosáhnout jeden z nich vytyčeného cíle. Je zřejmé, že ne vždy je příslušný algoritmus schopen v daném čase takovou otázku rozhodnout. Neuniformní metody proto nacházejí uplatnění především v herních koncovkách nebo při analýze taktických podcílů.

Tato práce se zabývá neuniformní metodou prohledávání založenou na hrozbách. Za hrozby jsou považovány takové tahy, že když se po jejich zahrání protivník vzdá svého tahu, bude pak již s jistotou poražen. Hrozby umožňují podstatně zmenšit prohledávaný prostor, neboť obráncovy možnosti jak hrozbě uniknout jsou většinou značně omezené. Navíc je možno zaručit, že zúžením výběru tahů jen na hrozby a jim příslušející úniky není ovlivněna korektnost výsledku.

Ne každý tah představuje hrozbu nebo únik před ní. Lokální charakter těchto pojmů však dovoluje stanovit tzv. zóny relevance, seznamy tahů nebo míst na hracím plánu, které jedině mohou mít vliv na výsledek výpočtu. Možnost omezit prohledávání pouze na tahy patřící do zóny významně zvyšuje efektivitu metody.

Kompletní popis algoritmu konstrukce zón relevance pro metodu zkoumanou v této práci je jejím hlavním přínosem.

Tato práce je organizována následujícím způsobem:

- Ve druhé kapitole popíšeme tři hry, AtariGo, Hex a Piškvorky, které jsme si vybrali pro testování zkoumané metody. Na jejich příkladu přiblížíme neformální pojem hrozby a pokusíme se zamyslet nad tím, jaké vlastnosti hra musí mít, aby v ní bylo možné hrozeb využít.

- Ve třetí kapitole představíme relativně novou techniku neuniformního prohledávání stromu hry jménem lambda search. Skrze ni získá pojem hrozby formální obsah, což nám následně umožní dokázat důležité vlastnosti popsané metody, jmenovitě jistou podobu vět o korektnosti a úplnosti.

- Kapitola 4, která se zabývá popisem konstrukce zón relevance, tvoří jádro práce. Nejprve je v ní vybudován formální rámec pro práci s hrami

jistého typu, ve kterých lze zóny relevance snadno definovat. Poté jsou vysloveny a dokázány věty pro konstrukci zón relevance v metodě lambda search.

- Tématem kapitoly 5 je implementace popsaných metod. Teoretické úvahy předchozích dvou kapitol v ní dostanou podobu algoritmu a bude naznačeno, jakých výsledků je s jeho pomocí možno dosáhnout.

- Poslední kapitola podá celkové shrnutí práce a zamyslí se nad možností jejího dalšího rozšiřování.

- Implementaci popisovaných metod napsanou v jazyce Java, která vznikla jako součást této práce, lze nalézt společně se stručnou dokumentací na přiloženém CD.

# Kapitola 2

# Zkoumané hry

Pro testování zkoumaných metod byly vybrány hry AtariGo, Hex a Piškvorky. V následujících sekcích jednotlivé hry nejprve stručně představíme, popíšeme jejich pravidla a zajímavé vlastnosti. Zmíníme se o stavu těchto her z pohledu umělé inteligence, o problematice jejich řešení. Též se pokusíme zdůvodnit, proč byly právě tyto hry pro testování vybrány. Na závěr kapitoly identifikujeme rysy společné všem těmto hrám, které byly důležité pro aplikaci námi zkoumaných metod v nich.

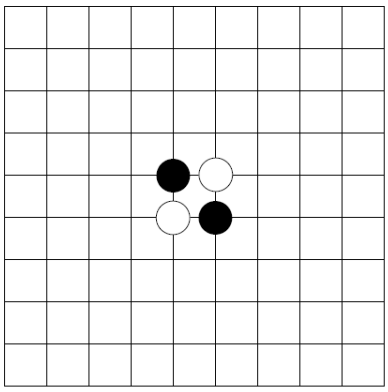
## 2.1 AtariGo

Go je prastará strategická desková hra dvou hráčů. Pochází již ze starověké Číny. První písemná zmínka o ní se dochovala z pátého století před naším letopočtem a můžeme ji proto právem považovat za nejstarší hru svého druhu vůbec. V současné době nachází své příznivce po celém světě, zejména však ve Východní Asii.

V klasické verzi hry Go se dva hráči, černý a bílý, střídají v pokládání kamenů své barvy na prázdné průsečíky devatenácti svislých a vodorovných čar. Kamen nebo skupina kamenů může být zajata a odstraněna z hrací plochy, pokud je úplně obklíčena kameny opačné barvy. Cílem hráče je kontrolovat větší území než soupeř tím, že pokládá kameny takovým způsobem, aby je nebylo možné zajmout. Hra končí v okamžiku, kdy se oba hráči po sobě dobrovolně vzdají tahu, čímž dají najevo, že ani jedna strana již svými tahy nemůže zvětšit své území, ani zmenšit to protivníkovo. Následně jsou velikosti území spočítány a hráč s větším kontrolovaným územím se stává vítězem.

AtariGo je zjednodušená verze hry Go, používaná někdy učiteli Go při výuce této hry. Cíl hráče v AtariGo je jednoduchý: zajmout libovolnou pro-

tivníkovu skupinu. Hráč, který toho první dosáhne, se stává vítězem a hra končí. AtariGo může být hráno na desce libovolné velkosti, většinou se dává přednost deskám čtvercovým. Na obrázku 2.1 je znázorněna typická počáteční pozice v AtariGo. Formace kamenů uprostřed se někdy nazývá křížový střih (anglicky crosscut) a předznamenává nestabilní partii. Možné jsou ale i jiné počáteční pozice lišící se tím, který jev ve hře chce učitel svému žákovi vysvětlit.



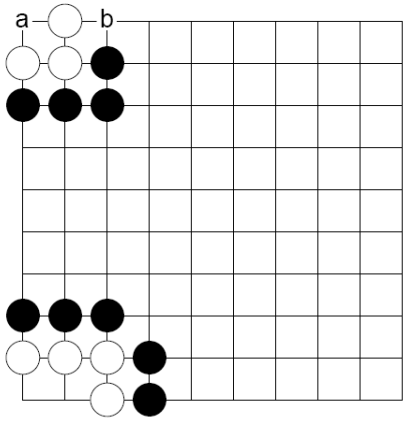
Obrázek 2.1: Typická počáteční pozice pro AtariGo na desce $10 \times 10$.

Nyní si podrobněji vyložíme pravidla hry. Vodorovné a svislé čáry vyznačují na hrací ploše průsečíky, na které je možno umísťovat kameny. Těmto průsečíkům budeme říkat *body*. Dva body nazýváme sousední, jestliže jsou vedle sebe ve svislém nebo vodorovném směru, nikoli však ve směru diagonálním. Dva kameny jednoho hráče umístěné na sousední body jsou spojené a skupiny navzájem spojených kamenů nazýváme *řetězce*. Volné body sousedící s řetězcem nazýváme jeho *svobody*. K zajmutí řetězce a tedy i k ukončení hry dojde, pokud řetězec ztratí poslední svobodu, jinými slovy pokud je obsazen poslední volný bod, se kterým sousedil. Pokud řetězci zbývá poslední svoboda, říkáme, že je *v atari*. Odtud také pochází jméno hry.

Když hráč zahraje takový tah, že nějaký jeho řetězec přijde o všechny svobody, jedná se o „sebevraždu“ a hráč prohrává. Toto však neplatí, pokud by takovým tahem zároveň obsadil poslední svobodu protivníkova řetězce. V takovém případě by se naopak jednalo o vítězný tah.[1] Právě popsanou situaci znázorňuje obrázek 2.2. Pokud černý zahraje na bod označený písmenem a, jedná se o sebevražedný tah a černý prohraje hru. Pokud naproti tomu

[1]Motivace pro toto pravidlo pochází z Go. Protivníkův řetězec je zde totiž v takové situaci odstraněn z hrací plochy a zajímající kámen tím získá svobodu.

nejprve obsadí bod označený `b` a teprve později bod `a`, dosáhne výhry. Ani jeden z hráčů na našem obrázku nechce hrát na body v levém dolním rohu hrací plochy, neboť druhý hráč by v zápětí vyhrál tahem na druhý z bodů. Bílý řetězec je proto v bezpečí, dokud jsou na hrací ploše jiná volná místa.




Obrázek 2.2: Demonstrace situace se sebevraždou v AtariGo.

Jelikož pravidla AtariGo narozdíl od Go nepřipouštějí možnost hráče vzdát se svého tahu, neboli *pasovat*, nemůže tato hra skončit remízou. To si snadno uvědomíme, při pohledu na plně obsazenou hrací plochu. Na ní totiž neexistuje řetězec, který by měl svobody, a tedy dříve, než hra dosáhne takového stavu, nutně skončí výhrou jednoho z hráčů.

V posledních desetiletích se hra Go těší stále větší pozornosti výzkumníků z oblasti umělé inteligence [5]. Navzdory veškerému úsilí jsou i nejlepší programy hrající Go stále velmi slabé a nepřesahují svou úrovní průměrného amatérského hráče. Zdá se, že hlavní překážkou kvalitní počítačové hry je velké množství přípustných tahů typické herní pozice v Go[2], které znemožňuje algoritmům typu alfa-beta prohledávat do dostatečné hloubky. Navíc ani ohodnocování nekoncových pozic nemůže být tak přímočaré jako v některých jiných hrách. Je to způsobeno tím, že prostý výpočet velikosti území pod kontrolou každého z hráčů dává spolehlivé výsledky až při téměř zaplněné hrací ploše. Do takového stavu by se ale hra typicky dostala až mnoho desítek tahů poté, co by ji lidští hráči ukončili a o příslušnosti jednotlivých území se dohodli na základě zkušenosti.

AtariGo je výrazně jednodušší než Go, neboť nepřipouští většinu komplexnějších aspektů svého složitějšího protějšku a má jasně definované ukon-

[2]Toto číslo zřídka klesá pod 50 přípustných tahů (např. u Šachu se uvádí v průměru 37 přípustných tahů) a po většinu hry se pohybuje v rozmezí 150-250.

čení (zajmutí řetězce). Přesto je AtariGo jednou z důležitých podúloh v Go a efektivní algoritmy pro její řešení budou jistě zajímavé i pro počítačové Go.

Řešením problému AtariGo na malé hrací ploše se zabývali van der Werf et al. [19]. Použili metodu prohledávání založenou na algoritmu alfa-beta s několika vylepšeními a speciálně navrženou ohodnocovací funkci. Vyřešen byl problém AtariGo na prázdné hrací ploše do velikosti $5 \times 5$. Ukázalo se, že při sudých velikostech hrací plochy ($2 \times 2$ a $4 \times 4$) má vyhrávací strategii druhý hráč, když nejprve dojde z větší části k zaplnění hrací plochy a první hráč je pak nucen oslabit svou pozici, neboť v AtariGo není dovoleno pasovat. Na hrací ploše s lichým počtem bodů ($3 \times 3$ a $5 \times 5$) vyhrává první hráč poté, co se ujme iniciativy a obsadí střed hrací plochy. Prohledávání pro velikost $6 \times 6$ se autorům nepodařilo dokončit, výsledky ale naznačují, že řešení se nalézá nejméně v hloubce 24 půltahů.

Na ploše velikosti $6 \times 6$ byla vyřešena alespoň pozice s již obsazeným středem, kde kameny tvoří výše zmiňovaný křížový střih. Hodnota této pozice je vítězství prvního hráče a to nejvýše za 15 tahů. Stejný výsledek je možno najít i v [8], kde bylo k jeho spočtení použito metody gradual abstract proof search, která je založena na hrozbách podobně jako algoritmus, který předvedeme v této práci.

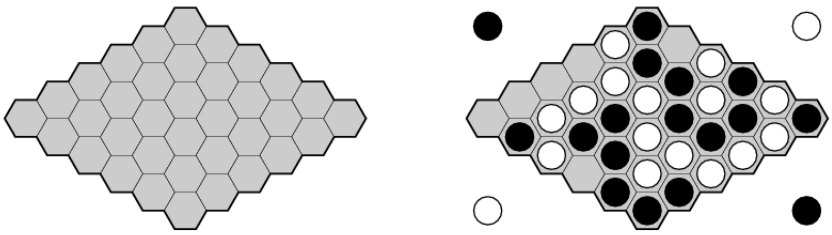
AtariGo jsme si vybrali pro testování, neboť je to svými pravidly sice poměrně jednoduchá, významem však velmi důležitá podúloha Go, hry, která teprve čeká na nějaký průlomový algoritmus či metodu, jež by kvalitu počítačových programů pro Go alespoň přiblížili úrovni profesionálních hráčů. Navíc se ukazuje, že v této hře zastávají důležitou roli hrozby, což je pro úspěšnou aplikaci námi zkoumaných metod podstatné. Za poslední důvod by mohl být považován fakt, že hra AtariGo v sobě obsahuje prvek tzv. zugzwangu, což je vlastnost, kterou se budeme podrobně zabývat v příští kapitole. Zugzwang naopak použití metod založených na hrozbách komplikuje, a tak nám AtariGo poslouží jako zdroj příkladů demonstrujících omezení naší metody.

## 2.2 Hex

Desková hra jménem Hex se hraje na šestiúhelníkové síti ve tvaru kosočtverce o různých rozměrech. Dva hráči, říkejme jim opět černý a bílý, se ve hře pravidelně střídají v umisťování kamenů své barvy na volná šestiúhelníková políčka hrací plochy. Protilehlé strany kosočtverce jsou vždy přiděleny jednomu z hráčů a úkolem hráče je, aby tyto dvě své strany pomocí kamenů své barvy propojil.

Hex je hra poměrně mladá. Pochází z dvacátého století a byla vymyšlena nezávisle dvěma osobami. Jejími autory jsou dánský matematik Piet Hein

a matematik John Nash z university v Princetonu. Brzy po svém vzniku roku 1942 byla hra v Dánsku známa pod názvem Polygon, Nashovi kolegové v Americe jí říkali prostě Nash. V roce 1952 vydala firma Parker Brothers verzi hry pro trh. Hra tehdy dostala jméno Hex a tento název se používá dodnes.



Obrázek 2.3: Hrací plocha pro Hex o rozměrech $6 \times 6$: prázdná počáteční pozice (vlevo) a pozice, ve které zvítězil bíly (vpravo).

Na obrázku 2.3 si můžeme prohlédnout počáteční pozici hry Hex a jednu z možných pozic koncových. V té druhé se bílému podařilo propojit jemu přidělené dvě strany kosočtverce (označené bílými kameny ležícími mimo hrací plochu) nepřerušeným řetězcem kamenů a tím splnil cíl hry.

Ačkoliv jsou pravidla Hexu velmi jednoduchá, dávají vzniknout hře, o jejíž strategii byly již napsány knihy a jejíž některé zajímavé vlastnosti přilákaly i pozornost matematiků. Jednou z takových vlastností je fakt, že hra nemůže skončit remízou. Skutečně, jediným způsobem jak zabránit soupeři, aby propojil své dvě strany kosočtverce, je vytvořit propojení vlastní. Toto tvrzení, ač intuitivně zcela přijatelné, musí být dokazováno s rozvahou. Dá se totiž ukázat (viz [9]), že je ekvivalentní s Brouwerovou větou o pevném bodě.

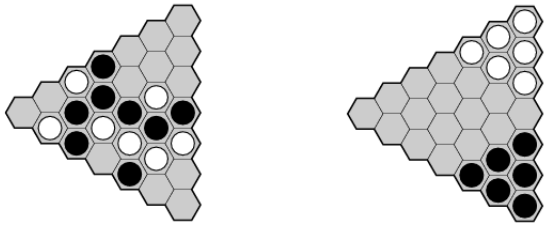
Další zajímavou vlastností, které si všiml již jeden z autorů hry John Nash, je existence vyhrávací strategie pro prvního hráče. Důkaz tohoto tvrzení je nekonstruktivní a využívá myšlenku tak zvaného kradení strategie (anglicky *strategy stealing argument*).

Základní idea této úvahy je následující. Jelikož Hex je konečná hra s úplnou informací, která nemůže skončit remízou, je zřejmé, že buď první nebo druhý hráč má vyhrávací strategii. Předpokládejme, že existuje vyhrávací strategie pro druhého hráče (tedy bílého). Za tohoto předpokladu může první hráč (černý) hrát následujícím způsobem: prvním tahem obsadí libovolné políčko hrací plochy; následující tahy ignoruje svůj první tah (pohlíží na hru jako kdyby začínal až druhý) a hraje podle (vyhrávací) strategie bílého. Pokaždé, když bude černý na tahu, poradí mu takový postup, aby obsadil políčko, které je buď prázdné, nebo již obsazené černým kamenem. Ve druhém

případě si černý opět náhodně zvolí prázdné políčko. Je vidět, že černý může postupovat podle vyhrávací strategie bílého a tedy černý i bílý mají vyhrávací strategii. Pouze jeden z hráčů však může hru vyhrát. Tímto sporem je důkaz ukončen.

V popsaném důkazu je implicitně používána skutečnost, že hráč Hexu si nemůže svým vlastním tahem nijak uškodit. Upozorněme, že tato vlastnost her bude významná při dalším popisu, zejména v následující kapitole.

Existuje několik způsobů, jak nevýhodu druhého hráče kompenzovat. Jedním z nich je uvažovat místo kosočtvercové hrací plochy kosodélník, ve kterém jsou protilehlé strany příslušející druhému hráči o jedno políčko blíže k sobě. Lze ale snadno ukázat, že v takovém případě si výhru může vynutit druhý hráč. Stačí mu pouze „kopírovat“ protivníkovi tahy podle vhodně zvolené symetrie. Druhou možností, která se často používá, je zavedení tzv. pravidla výměny (anglický swap rule). Při jeho použití je druhému hráči dána možnost zvolit si, za kterou barvu bude hrát, poté, co hráč jedna zahrál první tah. Pravidlo výměny ve svém důsledku motivuje prvního hráče zahájit hru neutrálním tahem, neboť druhý hráč si pravděpodobně zažádá o výměnu barev, pokud se mu zahajující tah bude jevit jako výhodný.

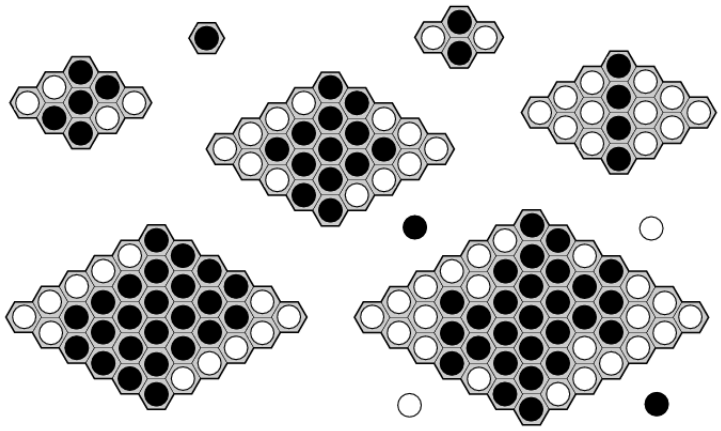


Obrázek 2.4: Hra Y: pozice vyhraná černým (vlevo) a Hex jako speciální případ hry Y (vpravo).

Hrou, která byla naimplementována za účelem výzkumu v této práci, ve skutečnosti není Hex, ale její obdoba známá pod názvem Y. V té se každý ze dvou hráčů snaží o propojení tří stran trojúhelníkové sítě jedním řetězcem (viz obrázek 2.4 vlevo). Hra Y má podobné vlastnosti jako Hex: nemůže skončit remízou a ví se o ní, že první hráč má vyhrávací strategii. Podobně jako v Hexu se ale tuto strategii zatím nepodařilo nikomu explicitně popsat. V jistém smyslu se dá Y považovat za zobecnění Hexu. Snaha o propojení tří stran ve speciálně upravené pozici (obrázek 2.4 vpravo) je totiž ekvivalentní se snahou o propojení protilehlých stran vzniklého kosočtverce tak, jak ji známe z Hexu.

Pro řešení Hexu pomocí počítače byly vyvinuty poměrně sofistikované metody. Jedna z nich je založena na představě hrací plochy Hexu jako elektrické sítě [4], kdy hráči pokládáním kamenů snižují elektrický odpor, který síť klade elektrickému proudu, jenž skrz ní prochází. Tento přístup může být kombinován s výpočtem tzv. virtuálních propojení (virtual connections – viz [11]), neboli míst na hrací ploše, která sice fakticky propojena nejsou, ale jeden z hráčů si dokáže propojení vynutit a to i v případě, že právě není na tahu.

Pro malé hrací plochy Hexu je známo mnoho výsledků. V lidských silách je řešit pozice na ploše do velikosti $5 \times 5$. Postupným úsilím několika týmů byla s pomocí počítače nedávno rozpoznána všechna vyhrávací a prohrávací zahájení pro hrací plochy velikosti $6 \times 6$ a $7 \times 7$ (viz např. [10]). Dosavadní dosažené výsledky shrnuje obrázek 2.5.



Obrázek 2.5: Vyhrávací a prohrávací zahájení. Barva políčka značí vítěze partie, která začne tahem černého na ono políčko.

Ačkoliv hra Hex může díky společnému pojmu řetězce kamenů připomínat Go, strategie jsou v obou hrách značně odlišné. I v Hexu jsou sice neustále přítomny hrozby, jejich vliv by se ale dal označit za méně přímý. Například svým prvním tahem na prázdnou hrací plochu o rozměrech $n \times n$ černý jistě hrozí, že své protilehlé strany propojí, bude k tomu ale potřebovat ještě minimálně dalších $n - 1$ tahů, a to i při neustálém pasování bílého. (Pro srovnání v AtariGo s křížovým střihem uprostřed může černý již svým prvním tahem dostat bílého do atari, a tedy hrozit vítězstvím v tahu následujícím.) Srovnání přístupu založeného na hrozbách s výsledky získanými pomocí metod tak říkajíc „šitých Hexu na míru“ by mělo být jistě přínosné.

## 2.3 Piškvorky

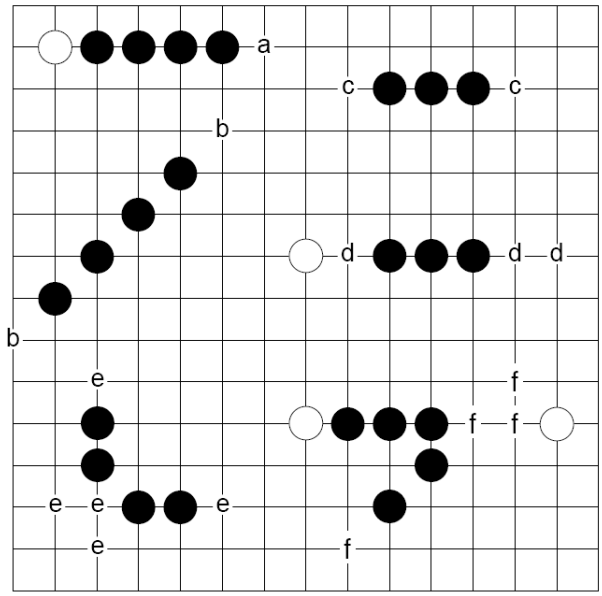
Hru Piškvorky, ve světě známou spíše pod jejím japonským názvem Go-moku, nejspíš není třeba dlouze představovat. V našich končinách se většinou hraje na čtverečkovaném papíře a hráči se v ní střídají v umisťování dvou druhů značek (jeden hraje s kolečky a druhý s křížky) na prázdné čtverečky papíru. V oblastech světa, kde je více zakořeněna tradice hry Go, tedy především v Asii, bývá hra hrána na hrací ploše Go s využitím černých a bílých kamenů umísťovaných na křížení vodorovných a svislých čar. Ačkoliv jsou herní pomůcky různé, úkol hráče zůstává stále stejný: vytvořit na hrací ploše nepřerušenou pětici svých značek či kamenů v řadě a to buď vodorovně, svisle nebo diagonálně.

Podobně jako v Hexu i v Piškvorkách platí, že hráč si nemůže umístěním svého kamene uškodit a dostat se do horší situace, než byla ta před tahem. I v této hře lze tedy využít triku s kradením strategie a ukázat, že existuje neprohrávací strategie pro prvního hráče. Výsledek je oproti Hexu oslaben tím, že v Piškvorkách může nastat i remíza, a to konkrétně v případě, že hráči vyplní kameny celou hrací plochu, aniž by některý z nich dosáhl vytvoření oněch pěti značek v řadě.

Variant pravidel Piškvorek, které se snaží výhodu prvního hráče omezit, je několik. Především se dává přednost hrací ploše velikosti $15 \times 15$ před tradiční velikostí $19 \times 19$ původně převzaté z Go. Další možností je zakázat jednomu nebo oběma hráčům zahrát některé vzory kamenů, jako na příklad tzv. přesah (anglicky overline), což je formace kamenů stejné barvy v řadě o délce šest. Třetí kategorii omezení představuje pravidlo o výměně, se kterým jsme se již setkali u Hexu. Jde opět o to, že po několika prvních tazích dostane druhý hráč možnost zvolit si stranu, za kterou bude ve hře pokračovat, což motivuje prvního hráče připravit pozici s pokud možno nerozhodnou bilancí, aby jeho protivník nemohl svou volbou získat navrch. Všechna tato pravidla s přesným vymezením v sobě zahrnuje profesionální varianta Piškvorek jménem Renju. V této práci se ale popsanými obměnami pravidel nebudeme zabývat.

Piškvorky jsou typickým příkladem hry, ve které hrají důležitou roli hrozby, tedy typy tahů jednoho hráče, na které musí jeho protivník okamžitě specifickým způsobem reagovat, aby zabránil hrozící prohře. Základní druhy hrozeb mají mezi hráči Piškvorek svá jména; některé z nich jsou zachyceny na obrázku 2.6.

Hrozbám označeným v obrázku písmeny `a` a `b` se říká *čtyřka*. Vidíme, že v případě čtyřky `a`, musí bílý okamžitě reagovat a hrozbu blokovat, jinak černý následujícím tahem vytvoří pět kamenů v řadě a zvítězí. V případě čtyřky označené písmenem `b` (někdy se jí též říká nebezpečná čtyřka), přišel již bílý pozdě. Ať se pokusí zabránit čtyřce z kterékoliv strany, černý zví-




Obrázek 2.6: Různé typy hrozeb v Piškvorkách.

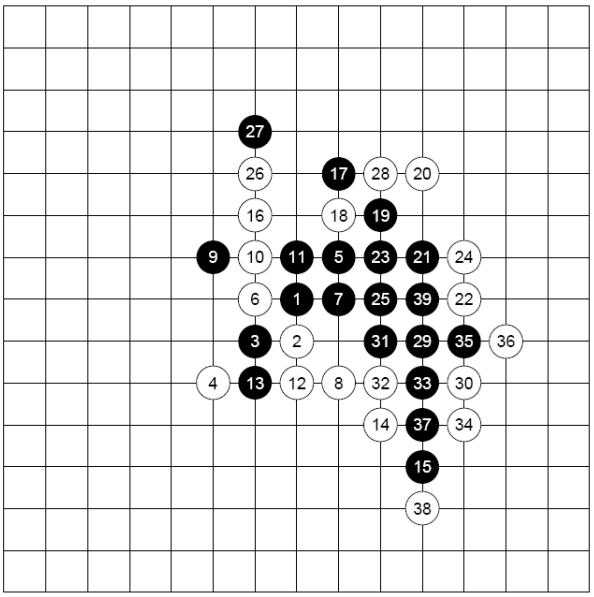
těží tahem na straně druhé. Písmeny `c` a `d` jsou označeny konfigurace známé jako *trojka*. Jedná se o hrozby méně přímé, přesto se nevyplácí je přehlédnout. Přidáním dalšího kamene dokáže totiž černý z trojky vytvořit nebezpečnou čtyřku, a potom vyhrát v následujícím tahu. Posledním druhem hrozeb v Piškvorkách, které zde nyní představíme, jsou tzv. *vidličky* (písmena `e` a `f` na obrázku). Jejich síla spočívá v tom, že při nich jeden z hráčů zkombinuje některé z předchozích hrozeb ve dvou nezávislých směrech a jeho protivník pak již nedokáže všem jednoduchým hrozbám včas zabránit.

Rychlé a přesné vyhodnocování jednotlivých hrozeb na hrací ploše nebo i celých posloupností skládajících se z vynucených tahů je jednou z klíčových vlastností charakterizující profesionální hráče Piškvorek. Toto tvrzení je mj. podporováno skutečností, že každá vítězná partie musí nutně skočit situací, kdy hráč mající převahu vytvoří dvě hrozby najednou (dvě trojky, dvě čtyřky, nebo trojku a čtyřku), a tím si vynutí vítězství.

Na hrozbách je též podstatně založena metoda, kterou implementovali V. Allis et al. ve svém programu Victoria, s pomocí nějž se podařilo v devadesátých letech minulého století Piškvorky vyřešit [3]. Zmíněný program kombinoval prohledávací strategii proof-number search [2] se speciálním modulem pro ohodnocování pozic založeným na hrozbách. Tento modul se o každé pozici snažil dokázat, že je vyhrávací pro hráče na tahu, a to pouze

s využitím hrozeb. Lokální charakter uvažovaných hrozeb umožnil rychle vyloučit z úvah mnohé kombinace tahů, které spolu nesouvisí v tom smyslu, že k případné výhře není potřeba zahrát oba dva. Takový modul byl proto schopen se velmi rychle v situaci hrozeb zorientovat, a pokud vrátil jako výsledek vítězství hráče na tahu, bylo již o dané pozici s jistotou rozhodnuto. V opačném případě pokračovalo prohledávání proof-number search do větší hloubky.

S využitím programu Victoria se podařilo dokázat, že hodnota hry Piškvorky je vítězství pro prvního hráče (tj. černého). Na obrázku 2.7 si můžeme prohlédnout jednu z nejdelších variací hry, která byla při tomto důkazu prohledávána. Vidíme, že i při optimální strategii může bílý odvracet útok černého maximálně 39 půltahů[3].




Obrázek 2.7: Jedna z nejdelších variací při řešení Piškvorek.

Z předchozího je zřejmé, že Piškvorky jsou pro aplikaci metod založených na hrozbách obzvláště vhodné. Ačkoliv by se dalo namítnout, že nemá smysl se zajímat o hru, která již byla vyřešena, domníváme se, že v našem případě není námitka na místě. Námi zkoumaná metoda se totiž dá chápat jako zobec-

[3]Zároveň si můžeme všimnout, že černý v této hře vítězí pomocí přesahu. Allis ve své práci zkoumal i variantu hry, kde tato forma vítězství není přípustná, a i v tomto případě byla výsledkem jistá výhra černého. Nejdelší prohledávaná variace obměněné hry byla ale o několik půltahů delší.

nění Allisova přístupu, které navíc nevyužívá žádných specifických vlastností testované hry. Proto bude srovnáním se specializovaným přístupem možno uvažovat o obecné aplikovatelnosti metody. Navíc by implementaci naší metody mělo být velmi snadné upravit i pro nějaký druh zobecněných piškvorek, hraných např. ve vícerozměrné síti, pro které zatím řešení nebylo nalezeno.

## 2.4 Shrnutí

Ačkoliv byly vybrány hry s různým původem, historií i vlastnostmi, spadají tyto hry do společného rámce, který dovolil zavést všechny klíčové pojmy jednotným způsobem a umožnil tak ve vzniklém programu čistě oddělit logiku hry od prohledávacího algoritmu. Společným rysem těchto her je střídavé kladení kamenů na hrací desku hráči dvou barev, přičemž kameny jednou položené již během hry nemění své místo. Tato „nepohyblivost kamenů“ není pro použití námi zkoumané metody nezbytně nutná, výrazně ale zjednodušuje všechny úvahy související s pojmem zóna relevance.

Druhou vlastností, které tyto hry sdílí, je přítomnost prvku náhlé smrti – pokud by jeden z hráčů několikrát za sebou vynechal svůj tah, dosáhne jeho oponent snadno vítězství. Tato vlastnost je klíčová při použití algoritmu založeného na hrozbách. Pouze v takových hrách, kde lze, byť jen při uvažování tahů jednoho hráče, „dohlédnout do konce“, bude totiž algoritmus moci hrozby nacházet, a pomocí nich směřovat k cíli.

# Kapitola 3

# Lambda prohledávání

Lambda prohledávání (lambda search) je neuniformní technika prohledávání herních stromů založená na hrozbách. Připomeňme, že pod pojmem hrozba si neformálně představujeme takový tah jedno z hráčů, že kdyby po něm jeho protivník tah vynechal, byl by pak již hráč snadno schopen dosáhnout svého cíle. Hrozbám je při lambda prohledávání přiřazena úroveň a jsou tak uspořádány od nejpřímějších k méně přímým. Hrozby nižší úrovně jsou pak využity při definici hrozeb vyšší úrovně a algoritmus pracuje postupným zkoušením hrozeb v pořadí určeným jejich úrovní, což vlastně odpovídá iterativnímu zvyšování složitosti pohledu na problém.

V této kapitole se seznámíme nejen s vlastní metodou, ale i s důležitými vlastnostmi zavedených pojmů. Podstatný je například fakt, že ačkoliv se prohledávání soustředí pouze na hrozby, garantuje nalezení minimaxové hodnoty, a to za předpokladu, že pravidla hry, kterou analyzujeme, dovolují hráči vzdát se svého tahu, neboli pasovat, a nebo hra alespoň nepřipouští tzv. zugzwang. Oba předpoklady tvrzení budou podrobně rozebrány.

Všechny tři hry, které jsme si vybrali pro testovaní, mají symetrickou povahu. Pravidly je dáno, jakého cíle se hráči mají snažit dosáhnout, aby zvítězili. Zároveň v nich ale je implicitně skryta podmínka dosáhnout cíle dříve, než se to podaří soupeři. Speciální druh obranných tahů, kdy soupeř může zabránit splnění cíle vlastním útokem anebo celkovou výhrou, bývá často označován jako *inverze*.

Ve druhé části kapitoly se budeme věnovat obměně lambda prohledávání nazývané dual-lambda search, ve které se, narozdíl od původního přístupu, zkoumají najednou hrozby obou hráčů, čímž je zajištěno včasné odhalování inverzí a jejich odvracení. Ukážeme, že dual-lambda search si zachovává důležité vlastnosti platné pro lambda search a může navíc v některých případech poskytnout i více informací.

## 3.1 Upřesnění problému

Lambda search slouží k prohledávání dvouhodnotových herních stromů určených nějakým cílem. V drtivé většině případů budeme jako cíl uvažovat úspěšné ukončení hry (tedy vítězství ve hře) dané jejími pravidly. Ve složitějších hrách si ale lze snadno představit úvahy o splňování rozličných taktických podcílů, např. získání nějaké důležité soupeřovy figury v Šachu nebo zajmutí konkrétního řetězce v Go. Pro zjednodušení výkladu budeme hráči, který se snaží o dosažení cíle, říkat útočník a jeho oponentovi obránce.

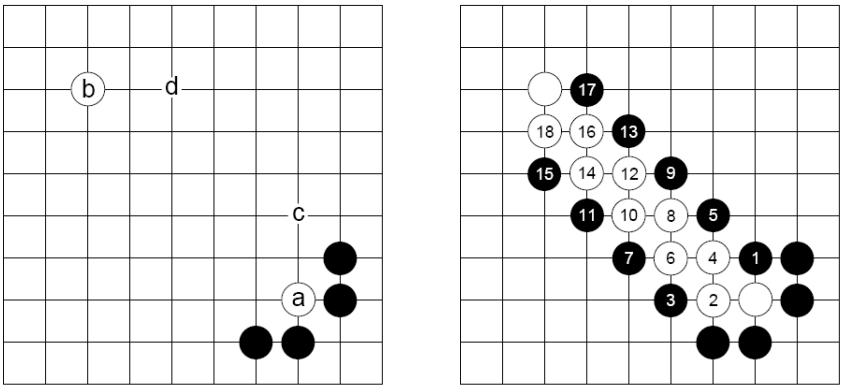
Úplný strom hry upravené vzhledem ke zkoumanému cíli je konstruován přirozeným způsobem. Pokud útočník některým svým tahem cíle dosáhne, je vzniklá pozice označena jako list s hodnotou 1 – tj. úspěch. Pokud v pozici není útočníkova cíle dosaženo a pravidla již nedovolují dále hrát, nebo se dá ukázat, že ani dalšími tahy již nebude možné cíle dosáhnout, je daná pozice označena jako list s hodnotou 0 – neúspěch. Všechny ostatní pozice jsou vnitřními vrcholy stromu a jejich hodnota je definována v závislosti na hodnotě jejich následníků pomocí principu minimaxu.

## 3.2 Motivační příklad

Abychom si myšlenku lambda prohledávání přiblížili na příkladě, představme si nejprve Šachy s cílem dát protivníkovi mat. Přímá hrozba na krále se nazývá šach a posloupnosti šachů končících matem říkejme například úspěšná posloupnost šachů. Všimněme si, že hrozit úspěšnou posloupností šachů nemusí být to samé jako přímo šachovat. Taková „meta-hrozba“, vlastně hrozba druhého řádu pokud šach označíme jako hrozbu řádu prvního, může být realizována méně nápadným a snadněji přehlédnutelným tahem.

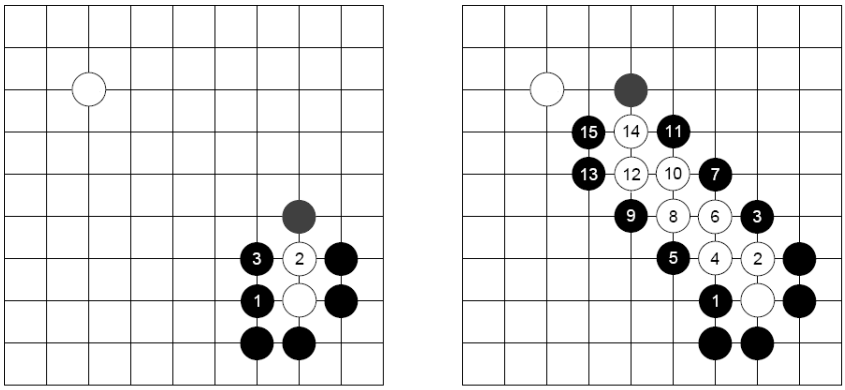
Podobně lze uvažovat i v AtariGo. Pokud obráncově řetězci po posledním tahu útočníka zbývá jen jedna svoboda, říkáme, že je řetězec v atari. Vynucená posloupnost tahů, ve které útočník dostává obráncův řetězec do atari a která nakonec skončí zajmutím řetězce, se nazývá úspěšné *schody*. Ve hře AtariGo můžeme tedy za hrozby prvního řádu považovat ty tahy, které dostávají řetězce do atari, a hrozbami řádu druhého pak budou tahy, po kterých útočník hrozí vítězstvím pomocí úspěšných schodů.

Podívejme se nyní na obrázek 3.1. Černému se nepodaří zajmout bílý kámen označený písmenem `a`, neboť jsou postupující schody zastaveny druhým bílým kamenem (písmeno `b`). Představme si nyní, že černý dostane k dobru jeden tah navíc (tah, po kterém bude bílý pasovat). Existuje hned několik způsobů jak zajistit, aby schody začaly fungovat. Černý toho může dosáhnout hraním tahu navíc např. na `c`, na což naváže úspěšnými schody z obrázku




Obrázek 3.1: Neúspěšný pokus o zajmutí bílého kamene pomocí schodů.

3.2 (vlevo), nebo na `d`, čímž si vynutí úspěšné schody zobrazené na tomtéž obrázku (vpravo).




Obrázek 3.2: Dvoje možné úspěšné schody po tahu černého navíc.

Tahy na `c` či `d` lze tedy chápat jako méně přímé hrozby útoku na kámen `a`, než je přímé ohrožení atari. Na zobecnění právě nastíněného principu je založeno lambda prohledávání.

## 3.3 Formální definice

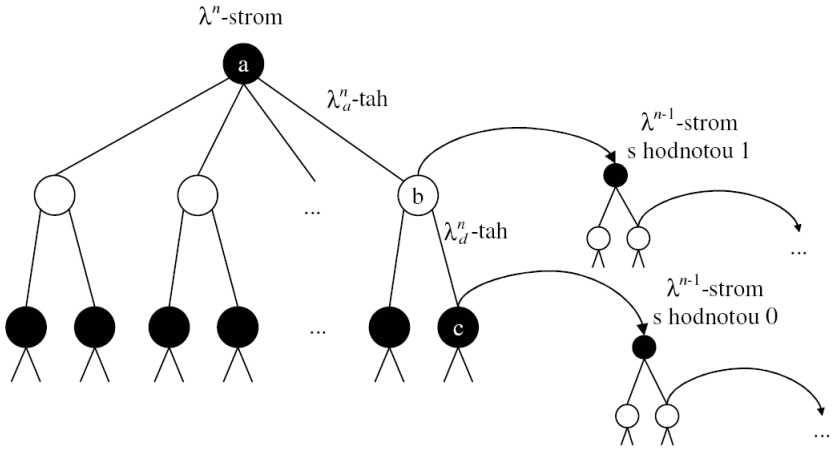
Je na čase přistoupit k definici lambda tahů a lambda stromů, které tvoří formalizaci myšlenek představených v předešlých odstavcích. Jedná se o de-

finici rekurzivního charakteru, přičemž jsou oba pojmy zaváděny najednou. Není ale obtížné se přesvědčit o tom, že definice je korektní, neboť se vždy odvolává jen na pojmy nižších úrovní, které již mohly být zavedeny dříve.

**Definice 1.** [18] Lambda tahy a lambda stromy:

- Útok: $\lambda_a^0$-tah je takový tah, kterým útočník dosáhne přímo svého cíle. Pozice vzniklá zahráním takového tahu je tedy koncová s hodnotou 1.
- Obrana: $\lambda_d^0$-tah neexistuje, neboť v prohrané pozici již nelze nic změnit.
- $\lambda^n$-strom ($n \geq 0$) je strom skládající se výhradně z $\lambda^n$-tahů (tedy z $\lambda_a^n$-tahů a $\lambda_d^n$-tahů). V kořenu stromu je na tahu útočník. Útočník hraje ve svých pozicích $\lambda_a^n$-tahy, obránce $\lambda_d^n$-tahy. Minimaxovou hodnotou stromu je buď 1 (úspěch útočníka), nebo 0 (úspěch obránce). Listu v takovém stromě odpovídá pozice, ve které nejsou k dispozici žádné $\lambda^n$-tahy. V takovém případě je hodnotou listu 1, pokud je v pozici na tahu obránce, a 0, pokud je na tahu útočník.
- Útok: $\lambda_a^n$-tah pro ($n > 0$) je takový tah, že když se v pozici vzniklé jeho zahráním obránce svého tahu vzdá, má útočník k dispozici $\lambda^k$-strom s hodnotou 1 pro nějaké $k < n$.
- Obrana: $\lambda_d^n$-tah pro ($n > 0$) je takový tah, že po jeho zahrání nemá útočník k dispozici žádný $\lambda^k$-strom s hodnotou 1 pro $k < n$.

Metoda lambda prohledávání je znázorněna na obrázku 3.3. Zkoumané pozici, ve které je na tahu černý, odpovídá kořen vyobrazeného stromu (`a`). Tah do pozice označené `b` je $\lambda_a^n$-tahem, neboť pokud po něm bílý pasuje, naváže černý $\lambda^{n-1}$-stromem s hodnotou 1 (viz menší strom napravo od pozice `b`). Když se dále podíváme na tah bílého do pozice `c`, vidíme, že se jedná o $\lambda_d^n$-tah, neboť v pozici `c` je hodnota $\lambda^{n-1}$-stromu rovna 0 (což je opět znázorněno menším stromem napravo od dané pozice). Všimněme si, že všechny uvažované stromy začínají tahem útočníka (tedy černého) a že algoritmus pracuje rekurzivně v „horizontálním“ směru, neboť menší $\lambda^{n-1}$-stromy jsou samy generovány pomocí $\lambda^{n-2}$-stromů a tak dále, až nakonec dospějeme ke stromům úrovně $\lambda^0$ (tento sestup je naznačen třemi tečkami v pravé části obrázku). Stojí též za povšimnutí, že $\lambda^{n-1}$-strom typicky obsahuje mnohem méně vrcholů, než strom $\lambda^n$.




Obrázek 3.3: $\lambda^n$-strom.

## 3.4 Dokončení příkladu

Pokud se ještě jednou vrátíme k našemu příkladu z obrázku 3.1, můžeme v něm snadno identifikovat právě zavedené pojmy. Každý tah, který dostává řetězec bílého do atari, je $\lambda^1_a$-tah a každá jeho záchrana (tedy prodloužení řetězce v místě poslední svobody)[1] je $\lambda^1_d$-tah. Celkově zachycuje obrázek 3.1 (vpravo) jednu z větví neúspěšného $\lambda^1$-stromu, tedy stromu s hodnotou 0. Tahy černého na místa `c` a `d` jsou ukázkou $\lambda^2_a$-tahů, neboť když po nich bílý vynechá, dokáže černý zvítězit pomocí úspěšných schodů, neboli $\lambda^1$-stromu s hodnotou 1.

## 3.5 Základní vlastnosti

Je dobré si uvědomit, jaký je vztah mezi úplným stromem hry pro danou pozici a jejím $\lambda^n$-stromem. Následníky určitého vrcholu v $\lambda^n$-stromě nejsou narozdíl od úplného stromu hry všechny pozice dosažitelné jedním tahem hráče na tahu, ale pouze pozice některé, konkrétně ty dosažitelné pomocí $\lambda^n$-tahu. $\lambda^n$-strom je tedy podstrom úplného stromu hry a některé herní variace v něm chybí. Speciálně konkrétnímu listu v $\lambda^n$-stromě nemusí nutně odpovídat koncová pozice hry.

Účinnost lambda prohledávání spočívá v tom, že přípustné tahy nejsou

[1] Obecně má obránce ještě jednu možnost, jak z atari unikat, a to konkrétně zajmutím některého z útočníkových řetězců jedním tahem. Pokud je takový tah k dispozici, jedná se o typický příklad inverze (viz dále).

jen slepě generovány a probírány jeden po druhém. Vybíráním tahů se metoda soustředí na zakládání a odvracení (více či méně přímých) hrozeb, čímž vlastně navádí prohledávání k cíli. Ačkoliv tedy lambda prohledávání nedostává předem žádné explicitní znalosti o zkoumané hře, dalo by se říci, že $\lambda$-tahy často vykazují alespoň minimální známky účelnosti nebo záměru narozdíl od často náhodně vypadajících posloupností tahů generovaným algoritmem alfa-beta.

Ačkoliv se o lambda prohledávání někdy mluví jako o algoritmu, nepředvádíme zde záměrně žádný konkrétní postup např. ve formě pseudokódu. Ve skutečnosti je totiž potřeba lambda search chápat především jako teoretický rámec, ve kterém je definována určitá třída stromů se specifickými vlastnostmi a ze kterého se teprve dosazením nějaké techniky pro prohledávání stromů algoritmus stane. Lambda stromy je tak možno vyhodnocovat nejen pomocí prohledávání do hloubky, což byl postup zvolený v této práci, ale např. i s použitím techniky proof-number search [20].

## 3.6 Korektnost a úplnost

Nejprve si uvědomme jednoduchou vlastnost zavedených pojmů, která plyne přímo z definice. Platí, že každý $\lambda_a^n$-tah je zároveň $\lambda_a^m$-tahem pro $m > n$ a že každý $\lambda_d^n$-tah je zároveň $\lambda_d^m$-tahem pro $m < n$. Proto, je-li hodnota $\lambda^n$-stromu rovna 1 pro nějaké $n$, bude v dané pozici 1 hodnotou i všech $\lambda^m$-stromů pro $m > n$.

Dvě důležité věty, které jsou v [18] dokázány, objasňují vztah mezi hodnotou $\lambda$-stromu dané pozice a hodnotou vlastní hry definovanou pomocí minimaxu. První z nich, věta o jistotě (confidence), říká, že je-li hodnota $\lambda^n$-stromu rovna 1, tj. úspěch útočníka, je 1 též hodnotou takové pozice. Argumentace při důkazu věty respektuje induktivní ráz pojmu lambda tahů a lambda stromů. Má-li totiž nejprve útočník zajištěnu hodnotu 1 pro $\lambda$-strom úrovně 0, neznamená to nic jiného, než že může vyhrát následujícím tahem, čili hodnota pozice je zřejmě 1. Když již nyní tvrzení platí pro každé $k < n$, uvažujme situaci, kdy hodnota $\lambda^n$-stromu je rovna 1. To znamená, že v jakékoliv variaci hry si útočník dokáže vynutit pozici, ve které obránce nemá k dispozici žádné $\lambda_d^n$-tahy. Ať v takovém momentě zahraje obránce jakýkoliv tah, může útočník navázat $\lambda$-stromem ostře nižší úrovně než $n$. Nyní již stačí jen použít indukční předpoklad a věta je dokázána.

Věta vlastně tvrdí, že na pozitivní výsledek lambda prohledávání se můžeme spolehnout. Zdůrazněme však, že vypovídá pouze o úspěšných $\lambda$-stromech a v situaci, kdy hodnotou $\lambda$-stromu určité úrovně je naopak 0, nám neumožňuje nic určitého o hodnotě pozice říci. V takové situaci věta totiž

pouze naznačuje, že pro případné vítězství útočníka bude potřeba použít lambda tahů vyšší úrovně. Přesto i takový výsledek přináší informaci. Říkám nám, že úloha vyřešit danou pozici je v jistém smyslu složitá.

Druhá věta, o konvergenci, odpovídá na otázku, co se bude dít, když budeme prohledávat $\lambda$-stromy stále vyšší a vyšší úrovně. Pokud předem víme, že v optimální hře může útočník dosáhnout výhry za nejvýše $d$ tahů, a pokud naše hra dovoluje jako legální možnost hráče vzdát se svého tahu (neboli pasovat), nebo hra alespoň nepřipouští tzv. zugzwang, věta říká, že tuto skutečnost odhalíme pomocí $\lambda^n$-stromu (s hodnotou 1) pro nějaké $n \leq (d-1)/2$.

*Poznámka.* Termín zugzwang si zasluhuje podrobnější vysvětlení. Pochází z němčiny, kde znamená „nucení táhnout“, a popisuje situaci ve hře, kdy se určitý hráč dostává do nevýhodné pozice, neboť musí zahrát nějaký tah – nejlepší by pro něj bylo se svého tahu vzdát a nehrát vůbec. Přesné vymezení toho pojmu se v různých kontextech liší. Pro nás budou hry *nepřipouštějící zugzwang* takové, ve kterých libovolný tah hráče navíc pouze zlepšuje hráčovu situaci, tedy hry, ve kterých lze aplikovat argumentaci s kradením strategie, jak byla popsána v minulé kapitole.

Důkaz věty o konvergenci probíhá opět indukcí, tentokrát podle hodnoty čísla $d$. Je-li $d = 1$, může útočník dosáhnout výhry jedním tahem a tento tah je jistě $\lambda_a^0$. Nechť nyní tvrzení platí pro všechna $d' < d$ a nechť může útočník dosáhnout výhry za nejvýše $d$ tahů ($d \geq 3$ a liché)[2]. Útočník má tedy nyní k dispozici tah $A$ takový, že za každým obráncovým protitahem $D_i$ následuje pozice, ve které útočník zvítězí za nejvýše $d-2$ tahů. Nyní se argumentace větví. Buď naše hra připouští vzdát se tahu a pas tedy musel být uvažován mezi některou z obráncových možností $D_i$, nebo nepřipouští zugzwang, a tedy po útočníkově tahu $A$ a obráncově vynechání je obráncova situace ještě horší než po libovolném z tahů $D_i$ a i zde tedy útočník zvítězí za nejvýše $d-2$ tahů. Podle indukčního předpokladu v takové pozici najdeme $\lambda^k$-strom s hodnotou 1 pro $k \leq (d-3)/2$ a tah $A$ byl tedy $\lambda_a^n$-tahem pro $n \leq (d-1)/2$. Tím je důkaz hotov.

*Poznámka.* Ve hrách jako je Šach, které připouštějí zugzwang, věta o konvergenci neplatí. Avšak kromě vzácných situací s zugzwangem nastávajících především v pozdní fázi hry v koncovkách není tento fakt na obtíž, máli obránce k dispozici alespoň jeden neškodný (nesebevražedný) tah, který může zahrát místo pasu. Z praktického hlediska lze tedy větu o konvergenci chápat jako platnou např. i při analyzování matových situací ve střední hře.

[2]Můžeme se omezit pouze na $d$ lichá, neboť ve hře bez zugzwangu, jak jsme si ji zavedli, nemůže hráč prohrát vlastním tahem.

Větami o jistotě a konvergenci jsme vlastně dostali jisté opodstatnění pro implementaci lambda prohledávání iterativním přístupem. Máme-li odpovědět na otázku, zda útočník může v dané pozici dosáhnout svého cíle, budeme za předpokladu, že cíle nemůže být dosaženo jediným tahem (hodnota $\lambda^0$-stromu je 0), postupně zkoušet $\lambda^1$-strom, $\lambda^2$-strom atd. dokud nenarazíme na první $\lambda$-strom, jehož hodnota je 1. Pak bude možno s jistotou uzavřít, že útočníkova cíle může být dosaženo. Navíc je podle věty o konvergenci (alespoň pro jistou třídu her[3]) zaručena úplnost takového postupu, a pokud cíle může být dosaženo, postupným zkoušením lambda stromů vzrůstající úrovně se to jednou dozvíme.

## 3.7 Prohledávání dual-lambda

Již jsme si všimli, že algoritmus lambda search je ve své podstatě nesymetrický, soustředí se pouze na splňování cílů jednoho hráče. To může být v určitých případech dostačující řešení. V koncovkách mnoha her ale často nastávají situace, kdy musí hráč před útokem pečlivě zvažovat, zda-li sám není ohrožen, a pouze takové útočné tahy, které berou v potaz i obráncův případný protiútok, mohou být úspěšné.

Efektivní přístup pro analýzu takových situací nabízí modifikace lambda prohledávání nazvaná dual-lambda search. V ní jsou do úvah zapojeny hrozby obou hráčů, čímž se zajistí, že hráč každým svým útočným tahem zároveň odvrací všechny protivníkovy hrozby, které by mohly být realizovány dříve. Navíc se předem vyloučí takové naivní pokusy o útok, kterými by hráč sám sebe oslabil.

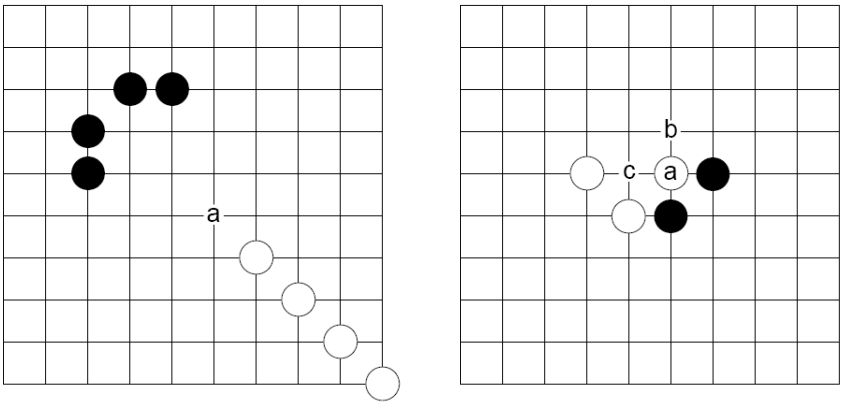
Idea metody spočívá v použití lambda prohledávání vzájemně rekurzivně z pohledu obou hráčů a k původnímu sledování hrozeb se tak přidává i detekce inverzí. V popsaných situacích s možností náhlých zvratů získáváme efektivnější algoritmus, který navíc dokáže v jednom důležitém případě vrátit více informací než jeho předchůdce. Pokud se totiž zjistí, že žádný z možných tahů jako útok neuspěje, neboť po něm protivník dosahuje vítězství dříve, než hráč na tahu, můžeme prohledávání ukončit a pozici označit za prohranou.

Přístup dual-lambda byl poprvé představen v [17], kde je algoritmus používán pro řešení koncovek v japonské hře Shogi. V této práci mírně upřesníme definici zavedenou v citovaném článku a dokážeme navíc pro dual-lambda search obdobu vět o jistotě a konvergenci.

[3]Konkrétně jde o konečné hry, které nepřipouštějí zugzwang nebo je v nich dovoleno pasovat.

## 3.8 Příklad

Pro ilustraci problému uvažujme nejprve pozici ze hry Piškvorky na obrázku 3.4 vlevo. Algoritmus lambda search se bude snažit využít skupiny kamenů v levém horním rohu hrací plochy pro zakládání hrozeb. Jedná o skupinu typu vidlička, která by sama o sobě stačila černému na výhru (dokázanou pomocí $\lambda^2$-stromu) nebýt bílých kamenů v pravém dolním rohu. Ve skutečnosti vidíme, že bílý hrozí vítězstvím v příštím tahu a jedinou možností černého, jak tomu zabránit, je hrát na místo označené písmenem `a`. Lambda prohledávání bude ale v takovéto pozici značně neefektivní, neboť protiútok bílého nebude brát přímo v potaz.




Obrázek 3.4: Příklady situací, kde hrají důležitou roli hrozby obou hráčů. Na tahu je vždy černý. Vlevo hra Piškvorky, vpravo AtariGo.

Druhý příklad (na obrázku 3.4 vpravo) znázorňuje situaci ze hry AtariGo, kdy hráč může svým neopatrným útokem dokonce sám sebe ohrozit. Černý je v této pozici na tahu a snaží se zajmout bílého kámen označený písmenem `a`. Tahy černého na místa `b` i `c` jsou $\lambda^1_a$-tahy, neboť dostávají bílý kámen `a` do atari. Ve skutečnosti je ale tah na místo `c` nepřijatelný, neboť se po něm sám útočící kámen dostává do atari a bude v příštím tahu bílého zajat. V následující sekci zavedeme $\mu$-tahy a $\mu$-stromy jakožto obdobu lambda-pojmů adaptovaných pro dual-lambda search. Uvidíme, že zatímco místo `b` zůstane i nadále místem útočného tahu úrovně 1, tedy $\mu^1_a$-tahu, o místu `c` to již platit nebude[4].

[4]Volba hry AtariGo pro tento příklad nebyla náhodná. Tato hra jako jediná z námi zkoumaných totiž připouští zugzwang, a pouze v ní šlo proto takový příklad předvést.

## 3.9 Teorie

Jak již bylo naznačeno, zavedeme za účelem formalizace metody dual-lambda search obdobu lambda tahů a lambda stromů. Následující definice je velmi podobná definici 1. Jelikož se v ní ale navíc mísí role útočníka a obránce, bude nutné pro sledování tohoto faktoru přidat ke každému pojmu ještě jeden údaj. Tento údaj bude v případě, že se jedná o tah, určovat, kterému z hráčů tah přísluší, v případě stromu bude indikovat hráče, který je na tahu v jeho kořeni. Jednoho z hráčů budeme značit písmenem $p$ a druhého, jeho oponenta, symbolem $o$.

**Definice 2.** [17] Mí tahy a mí stromy:

- Útok: $\mu_a^0(p)$-tah je takový tah, kterým útočník $p$ dosáhne přímo svého cíle. Pozice vzniklá zahráním takového tahu je tedy koncová s hodnotou 1.
- Obrana: $\mu_d^0(o)$-tah neexistuje, neboť v prohrané pozici již nelze nic změnit.
- $\mu^n(p)$-strom ($n \geq 0$) je strom skládající se výhradně z $\mu^n$-tahů (přesněji z $\mu_a^n(p)$-tahů a $\mu_d^n(o)$-tahů). V kořenu stromu je na tahu útočník $p$. Útočník hraje ve svých pozicích $\mu_a^n(p)$-tahy, obránce $o$ hraje $\mu_d^n(o)$-tahy. Minimaxovou hodnotou stromu je buď 1 (úspěch útočníka), nebo 0 (úspěch obránce). Listu v takovém stromě odpovídá pozice, ve které nejsou k dispozici žádné $\mu^n$-tahy zmiňovaných druhů. V takovém případě je hodnotou listu 1, pokud je v pozici na tahu obránce, a 0, pokud je na tahu útočník.
- Útok: $\mu_a^n(p)$-tah pro ($n > 0$) je takový tah, že existuje číslo $k < n$ takové, že když se v pozici vzniklé zahráním tahu obránce $o$ svého tahu vzdá, má útočník $p$ k dispozici $\mu^k(p)$-strom s hodnotou 1 a zároveň, když se svého tahu obránce nevzdá, nemá v této pozici k dispozici $\mu^l(o)$-strom s hodnotou 1 pro žádné $l \leq k$.
- Obrana: $\mu_d^n(o)$-tah pro ($n > 0$) je takový tah, že po jeho zahrání nemá útočník $p$ k dispozici žádný $\mu^k(p)$-strom s hodnotou 1 pro $k < n$.

Rozdíl obou definic je nutno hledat především ve čtvrtém bodě, kde se definuje $\mu_a^n(p)$-tah pro ($n > 0$). Kromě již známého požadavku, aby se jednalo o hrozbu zprostředkovanou nějakým úspěšným stromem úrovně $k < n$, navíc žádáme, aby ve výsledné pozici neměl úspěšný strom útočníkův oponent (a to na žádné úrovni $l \leq k$). Tím je zajištěno, že po zahrání takového tahu nebude sám útočník (alespoň z pohledu úrovně $k$) ohrožen.

## 3.10 Korektnost a úplnost pro dual-lambdu

Uvědomit si, že pro dual-lambda search platí obdoba vět o jistotě a konvergenci, není obtížné. Věnujme se nejprve první z nich a zopakujme, nyní v řeči $\mu$-pojmů, co věta o jistotě říká: Je-li hodnota $\mu^n(p)$-stromu v dané pozici rovna 1 pro nějaké $n$, může hráč $p$ s jistotou dosáhnout svého cíle (a 1 je tedy hodnotou dané pozice i v úplném stromě hry). Důkaz této věty se odvíjí naprosto stejným způsobem jako u předchozí verze (indukcí podle úrovně $n$), a proto se zde pouze pokusíme přeformulovat jeho hlavní myšlenku. Faktem, že se při procházení úplného stromu hry v pozicích, ve kterých je na tahu útočník, algoritmus omezujeme jen na některé tahy (konkrétně na $\mu_a^n$-tahy), nemůže být ovlivněna korektnost výsledku. Každý konkrétní důkaz tvrzení, že hodnota takové pozice je 1, totiž potřebuje pouze jednoho „svědka“ – jednoho jejího následníka ve stromě, jehož hodnota je též rovna 1 a který zajistí stejnou hodnotu i pro spočtené maximum. Problém s korektností by mohl nastat pouze kvůli vynechání některých následníků pozic, ve kterých je na tahu obránce (a ve kterých se počítá minimum). Vynechané (neprohledávané) jsou v těchto pozicích ale pouze tahy, které nejsou obrannými $\mu_d^n$-tahy, a proto po nich má útočník k dispozici $\mu^k$-strom s hodnotou 1 pro nějaké $k < n$. Na tomto místě se v důkazu použije indukční předpoklad a věta je dokázána.

Věta o konvergenci tvrdí, že je-li hodnota herní pozice 1, tedy úspěch útočníka, a tohoto úspěchu lze dosáhnou za nejvýše $d$ tahů, bude pro objevení této skutečnosti potřeba spočítat hodnotu $\mu^n$-stromu pro $n \leq (d-1)/2$ (samozřejmě opět za předpokladu, že hra nepřipouští zugzwang, kterýžto předpoklad byl již dříve podrobně diskutován). V duchu idejí popsaných v minulém odstavci se při důkazu této věty pro dual-lambda search můžeme odvolat na její obdobu z lambda prohledávání. Stačí si totiž uvědomit, jaký je rozdíl mezi stromy prohledávanými oběma metodami, jmenovitě že ve stromě prohledávaném metodou dual-lambda chybí pouze oproti stromu od metody lambda někteří následníci vrcholů, ve kterých je na tahu útočník. Vynechané jsou právě ty útočné tahy, po jejichž zahrání by útočník byl sám ohrožen (viz definice 2, čtvrtý bod). Takové tahy ale zcela jistě nemohou vést k výhře, neboť vedou do pozic vyhraných obráncem podle věty o jistotě.

## 3.11 Poznámka k implementaci

Stejně jako u lambda prohledávání je i v případě implementace metody dual-lambda search výhodné použít iterativní přístup a postupně testovat pozici ve fázích pomocí $\mu$-stromů vzrůstající úrovně. Nižší fáze prohledávání totiž mohou (i v případě, že skončí neúspěchem a nenaleznou vítězný strom) předávat

těm vyšším informace, které celý postup urychlí. Jmenovitě se vyplatí rozlišovat důvod, kvůli kterému určitý tah není při výběru označen jako $\mu_a^n(p)$-tah. V případě, že je to tím, že po zahrání daného tahu útočník obránci nehrozí a nemá tedy k dispozici $\mu^k(p)$-strom s hodnotou 1 pro žádné $k < n$, může tah stále představovat hrozbu na vyšší úrovni (což bude objeveno v dalších fázích výpočtu). Pokud je ale důvodem neúspěšného pokusu o útok fakt, že útočník tahem ohrožuje sám sebe, a obránce má tedy ve vzniklé pozici k dispozici $\mu^l(o)$-strom s hodnotou 1 (pro určité $l < n$), bude tato skutečnost pochopitelně překážkou i na vyšších úrovních a takový tah je možno předem vyloučit i pro všechny následující fáze výpočtu. V extrémním případě se pak může stát, že nižší fáze výpočtu takto vyloučí jako nepřijatelné vůbec všechny možnosti, kam lze táhnout, a výpočet dual-lambda může být ukončen a pozice označena jako (pro útočníka) prohraná.

# Kapitola 4

# Zóny relevance

Jednou z překážek efektivního prohledávání stromů her dvou hráčů často bývá vysoký faktor větvení těchto stromů, neboli veliký počet možných tahů, který má hráč typicky k dispozici a které je nutno vyzkoušet. Přitom je zřejmé, že mnoho vlastností prohledávaných pozic má lokální charakter, a že tedy mnohé ze zkoušených tahů nemohou takovou vlastnost ovlivnit. Seznamům tahů (nebo míst na hracím plánu), které jedině mohou mít vliv na výsledek prohledávání budeme říkat zóny relevance. Pokud se nám podaří pro daný problém definovat zóny relevance tak, že budou ve srovnání s množinou všech možných tahů relativně malé a navíc je bude možno efektivně počítat, můžeme jejich použitím prohledávání značně urychlit.

V této kapitole se budeme zabývat zónami relevance pro lambda search a dual-lambda search. Představíme si postupně zóny pro obranu a pro útok. První z nich, zóny pro obranu budeme definovat pro pozice, ve kterých je hodnota $\lambda^n$-stromu rovna 1 a které vznikly útočníkovým tahem, jenž představuje pro obránce hrozbu. V obranné zóně takové pozice nalezne obránce místa, kde jedině je možné hrozbě čelit a mezi kterými má tedy pouze zkoušet hledat své obranné lambda tahy. Zóny pro útok budou definovány pro pozice, ve kterých je hodnota $\lambda^n$-stromu rovna 0. Útočník jich využije při hledání svých $\lambda_a^{n+1}$-tahů. Ukážeme, že jedině v útočné zóně je možno tyto tahy nalézt.

Tematikou zón relevance pro algoritmy založené na hrozbách se zabývá článek [18] a také [7]. Oba citované přístupy jsou navrženy specificky pro hru Go, přičemž nedostatek prvního vidíme především v neúplnosti konstruované zóny, ve které mohou některé, pro statut testované vlastnosti podstatné, tahy chybět. Přístup z [7] je sice v tomto smyslu úplný, ale omezuje se pouze na některé druhy hrozeb[1]. Domníváme se, že naše nová metoda konstrukce zón

[1]Konkrétně tyto hrozby odpovídají $\lambda$-tahům do úrovně tři navíc s omezenou hloubkou

relevance tyto přístupy překovává, zejména díky následujícím vlastnostem:

- Metoda není přímo vázána na konkrétní hru. Třídu her, na které ji lze aplikovat, podrobně vymezíme v následujícím výkladu a ukážeme, že do ní patří AtariGo, Hex i Piškvorky.
- Metoda je „stejnoměrná“ v tom smyslu, že poskytuje zóny pro hledání útočných i obranných lambda tahů *libovolné úrovně* $n$. Takové zóny jsou vypočítávány během prohledávání stromů nižší úrovně $n - 1$.
- Metoda je úplná. Dokážeme, že útočné resp. obranné $\lambda$-tahy dané pozice se mohou vyskytovat pouze ve spočtených zónách.

Na závěr výčtu dodejme, že metoda vypočítává zóny relevance, které jsou netriviální – tj. typicky neobsahují všechny přípustné tahy dané pozice. To je ale vlastnost samozřejmá pro každou metodu pro výpočet zón, která má být užitečná.

Cesta k popisu konstrukce zón relevance v této kapitole nebude zcela přímočará. Nejprve bude třeba stanovit si formální rámec úvah a zavést některé pojmy a značení, které nám později usnadní vyjadřování. Našemu formálnímu rámci budeme říkat *hry s kameny* a představíme jej v sekci 4.1.

Ačkoliv jsou tématem kapitoly zóny relevance, ve skutečnosti budeme pracovat s abstraktnějšími objekty než jsou zóny neboli oblasti na hrací ploše. Takovým objektům budeme říkat *schémata zón relevance* a jejich obecné definici a vlastnostem se budeme podrobně věnovat v sekci 4.2.

Speciálním případem obecně definovaných schémat zón relevance jsou schémata zón relevance pro útok a obranu v lambda prohledávání. Přesná definice toho pro nás právě důležitého případu a její důsledky se objeví v sekci 4.3.

Metoda konstrukce zón relevance (resp. jejich schémat) pro lambda search má induktivní charakter. Nejprve budeme ukazovat, jak počítat zóny pro lambda stromy a tahy úrovně 0, později pak jejich vhodnou kombinací sestrojíme zóny pro úrovně vyšší. Prvním krokem induktivního postupu se budou zabývat sekce 4.4 a 4.5, z nichž první bude spíše specifická pro námi zkoumané hry a ve druhé se již na obecné rovině konstrukce pro úroveň 0 dokončí.

Konstruováním zón relevance pro lambda stromy a tahy obecné úrovně $n$ s pomocí již spočtených zón pro stromy s menší úrovní nebo alespoň s menším počtem vrcholů se zabývá sekce 4.6. Vyslovíme v ní dvě věty, zvlášť pro

příslušných $\lambda$-stromů.

útok a pro obranu, ve kterých zmíněnou konstrukci popíšeme a dokážeme její korektnost. Abychom se vyhnuli komplikacím, které pro důkaz korektnosti představují speciální typy ukončení hry v podobě inverzí, remízy a sebevražedných tahů, nebudeme nejprve tyto možnosti uvažovat. Ihned po vysvětlení základních idejí se ale k nim ale vrátíme a v sekcích 4.7, 4.8 a 4.9 bude jejich vliv na platnost zmiňovaných vět podrobně rozebrán.

Na závěr kapitoly se pokusíme ještě jednou zopakovat a shrnout, za jakých předpokladů dokázané věty o zónách relevance platí, a zejména jaké požadavky je třeba klást na hry, ve kterých o zónách uvažujeme.

## 4.1 Hry s kameny

Cílem následujících sekcí bude formalizovat pojem zóna relevance pro lambda search a dual-lambda search. Za tímto účelem bude nejprve nutné omezit se pouze na hry, ve kterých pojem zóna nebo oblast bude vůbec dávat smysl. Zavedeme proto tzv. *hry s kameny*, ve kterých půjde myšlenku polohy tahu či oblasti snadno podchytit.

**Definice 3.** Ve *hře s kameny* se dva hráči, černý a bílý, střídají v kladení kamenů černé resp. bílé barvy na hrací plochu. Jedním tahem se tedy myslí umístění kamenu hráčovy barvy na určité *místo* hrací plochy. Hráč může ve svém tahu umístit kámen na libovolné neobsazené místo na ploše. S položenými kameny se již nijak nehýbe a ty zůstávají na svém místě až do konce hry[2].

Jelikož se při úvahách o hrozbách často hráči nestřídají striktně podle pravidel hry (dochází k vynechávání tahů na jedné či druhé straně), nebude námi zavedený pojem herní pozice tuto informaci kódovat a perspektivu hráče na tahu budeme k dané pozici vždy přidávat zvlášť.

**Definice 4.** Další pojmy a značení:

- *Herní pozice* nese informaci o tom, která místa herní plochy jsou obsazena kameny které barvy a která jsou volná. Budeme je obvykle značit $Pos$, $Pos'$, ...
- *Hráčem* je buď černý $\bullet$, nebo bílý $\circ$. Hráče značíme symbolem $\pi$ a jeho oponenta $o(\pi)$.

[2] Ačkoliv jsme explicitně stanovili barvu kamenů obou hráčů, nemusí konkrétní hra barevnost vyžadovat. Potom stačí, když bude při vyhodnocování pozice pravidly dané hry tato informace ignorována.

- *Místa na hrací* ploše budeme značit malými písmeny: $x$, $y$,... a jejich množiny, tedy *oblasti*, písmeny velkými: $X$, $Y$, ...
- *Informace o tahu* se skládá z určení hráče, který tah provede (tedy barvy jeho kamene), a z místa, na které je kamen umístěn. Tyto dvě informace kóduje symbol $\pi/x$.
- Skutečnost, že pozice $Pos$ přejde tahem $\pi/x$ v pozici $Pos'$, zkrátíme zápisem $Pos \xrightarrow{\pi/x} Pos'$. Mimochodem to znamená, že místo $x$ bylo v pozici $Pos$ volné, a také, že pozice $Pos$ nebyla koncová (viz dále).

Je dobré již na tomto místě upozornit na jednu vlastnost her s kameny, která, ač velmi snadno přijatelná a názorná, není obecně v hrách samozřejmostí. Jedná se o vlastnost, které budeme říkat *komutativita* pokládání kamenů a jež spočívá v tom, že pozice vzniklá položením kamenů na určitá místa hrací plochy nezávisí na pořadí, v jakém byly kameny na svá místa umístěny. Formálně bychom mohli komutativitu vyjádřit např. následovně:

**Pozorování** (Komutativita)**.** *Nechť pro libovolné pozice* $Pos$, $Pos_1$, $Pos_2$, $Pos_{12}$ *a* $Pos_{21}$, *libovolná místa* $x_1$ *a* $x_2$ *a hráče* $\pi_1$ *a* $\pi_2$ *platí* $Pos \xrightarrow{\pi_1/x_1} Pos_1$, $Pos \xrightarrow{\pi_2/x_2} Pos_2$, $Pos_1 \xrightarrow{\pi_2/x_2} Pos_{12}$ *a* $Pos_2 \xrightarrow{\pi_1/x_1} Pos_{21}$. *Pak* $Pos_{12}$ *a* $Pos_{21}$ *je jedna a ta samá pozice, neboli* $Pos_{12} = Pos_{21}$.

Vlastnost komutativity není potřeba ve hrách s kameny nijak dokazovat. Plyne prostě z faktu, že objekt pozice v sobě nenese žádnou informaci o historii hry. Důležité ale je si uvědomit, že v námi zkoumaných hrách taková informace není podstatná a že naopak možnost ji ignorovat nás ušetří mnoha komplikací.

Dostáváme se k popisu pojmů týkajících se ukončení hry s kameny, neboli jejích pravidel. Ačkoliv by se zdálo přirozené definovat pravidla hry jako relaci, která rozdělí pozice na koncové a nekoncové a koncovým přiřadí hodnoty např. způsobem: 1 pro vítězství hráče •, 0 pro remízu a $-1$ pro vítězství hráče ∘, musíme v našem případě postupovat opatrněji. Hodnota koncové pozice v hrách s kameny totiž obecně závisí i na tom, který hráč zahrál poslední tah. Dále je často výhodné mít možnost relativizovat výsledek hry vůči jednomu z hráčů v tom smyslu, že výhra jednoho je chápána jako prohra druhého.

**Definice 5.** Zakončení hry:

- *Hodnotou koncové pozice* je číslo $h \in \{1, 0, -1\}$. Čísla vyznačují postupně výhru, remízu a prohru prvního hráče, tedy hráče •.

- Pozice dělíme na *koncové* a *nekoncové*. Pokud je pozice $Pos$ koncová, píšeme $\mathcal{T}(Pos)$.
- Pro koncové pozice je definována jejich hodnota, která navíc obecně závisí na tom, kdo zahrál poslední tah. Skutečnost, že koncová pozice $Pos$, která vznikla zahráním tahu hráče $\pi$, má hodnotu $h$, zapíšeme jako $\mathcal{V}(Pos, \pi) = h$.
- *Relativizující funkce pro hráče* jsou definovány předpisem: $\bullet(h) = h$ a $\circ(h) = -h$. Pokud symbol $\pi$ značí jednoho z hráčů, dovolíme si jej použít v dalším významu i pro označení jemu příslušené relativizující funkce.
- Pokud pozice $Pos$ přejde tahem $\pi/x$ do nějaké pozice $Pos'$, která je koncová a má hodnotu $h$ (vzhledem k poslednímu tahu hráče $\pi$), neboli

  $$(\exists Pos')(Pos \xrightarrow{\pi/x} Pos' \,\&\, \mathcal{T}(Pos') \,\&\, \mathcal{V}(Pos', \pi) = h),$$

  budeme tuto skutečnost zkracovat zápisem $Pos \xrightarrow{\pi/x} h$.

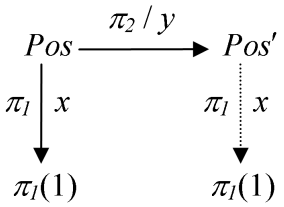
*Příklad.* Použití zavedených symbolů:

- Výhra hráče $\pi$ v pozici $Pos$ tahem na místo $x$: $Pos \xrightarrow{\pi/x} \pi(1)$.
- Sebevražda hráče $\pi$ v pozici $Pos$ na místě $x$: $Pos \xrightarrow{\pi/x} \pi(-1)$.

Při úvahách na dalších stránkách budeme někdy potřebovat silnější předpoklady o hrách, se kterými pracujeme, než jen pouhý fakt, že se jedná o hry s kameny. Takový předpoklad bude typicky tvrdit, že vlastnost tahu ukončit hru v nějaké pozici se zachová i po přechodu do pozice jiné, a budeme mu říkat *stabilita hry*.

**Definice 6.** Hra s kameny se nazývá *stabilní*, pokud pro libovolné pozice $Pos$ a $Pos'$, hráče $\pi_1$ a $\pi_2$ a místa $x$ a $y$ navzájem různá platí: Pokud $Pos \xrightarrow{\pi_1/x} \pi_1(1)$, $Pos \xrightarrow{\pi_2/y} Pos'$ a $\neg\mathcal{T}(Pos')$, pak $Pos' \xrightarrow{\pi_1/x} \pi_1(1)$.

V principu bychom mohli ještě rozlišit dva případy, a to, zda se $\pi_1 = \pi_2$ či nikoliv. Při první možnosti by pak stabilitu šlo interpretovat jako tvrzení, že si útočník nemůže umístěním vlastního kamene škodit či překážet při cestě k vítězné pozici. V případě druhém bychom se dozvěděli, že jedinou šancí obránce, jak odvrátit přímou hrozbu, pokud nemůže sám svým příštím tahem vyhrát, je obsadit místo, na které se útočník chystá umístit svůj vítězný kámen. Stabilitu ilustruje obrázek 4.1.




Obrázek 4.1: Stabilní hra.

## 4.2 Posloupnosti oblastí a schémata zón relevance

Již v úvodu bylo naznačeno, že v naší metodě ve skutečnosti nekonstruujeme zóny relevance, ale složitější objekty, jimž říkáme schémata zón. Pokud by intuitivnímu pojmu zóna ve hrách s kameny měla odpovídat jistá oblast, neboli množina míst na hrací ploše se speciálními vlastnostmi, bude naše schéma zón tvořit celá posloupnost takových oblastí, navíc navzájem do sebe vnořených inkluzí. Užitečnost takového přístupu spočívá v možnosti jemněji rozlišovat vliv jednotlivých míst na platnost vlastnosti pozice, která nás zajímá.

**Definice 7.** Posloupnosti oblastí a operace s nimi:

- *Posloupnosti oblastí* říkáme posloupnostem množin míst na hrací ploše, kteréžto množiny jsou do sebe rostoucím způsobem řazeny inkluzí, a značíme je symboly $\mathbf{X}$, $\mathbf{Y}$, ... $i$-tou množinu posloupnosti $\mathbf{X}$ budeme značit $X_i$. Celkově tedy můžeme shrnout: $\mathbf{X} = (X_1, X_2, \ldots)$ s tím, že platí $X_s \subseteq X_{s+1}$ pro každé $s \geq 1$.

- Na posloupnostech oblastí definujeme unární operaci $u$ – *odvinutí* a binární $\cup$ *sjednocení* následujícím způsobem: Nechť $\mathbf{X} = (X_1, X_2, \ldots)$ a $\mathbf{Y} = (Y_1, Y_2, \ldots)$, pak

$$\begin{aligned} u(\mathbf{X}) &= (X_2, X_3, \ldots), \\ \mathbf{X} \cup \mathbf{Y} &= (X_1 \cup Y_1, X_2 \cup Y_2, \ldots). \end{aligned}$$

  Odvinutí lze tedy chápat jako „zapomenutí“ první oblasti v posloupnosti, zatímco sjednocení je zavedeno prostě po složkách.

- Po složkách zavedeme i predikát $\subseteq$ nad posloupnostmi oblastí: Nechť opět $\mathbf{X} = (X_1, X_2, \ldots)$ a $\mathbf{Y} = (Y_1, Y_2, \ldots)$, pak

$$\mathbf{X} \subseteq \mathbf{Y} \equiv (\forall s \geq 1)(X_s \subseteq Y_s).$$

- Pokud použijeme oblast v kontextu, ve kterém je vyžadována posloupnost oblastí, máme tím na mysli posloupnost konstantní. Budeme tedy někdy psát $X$ ve významu $(X, X, \ldots)$.

Nekonečná posloupnost v definici je samozřejmě jen užitečnou abstrakcí, která nám zjednoduší vyjadřování. Jelikož totiž vždy uvažujeme hry s konečnou hrací plochou a také díky tomu, že jsou jednotlivé oblasti v posloupnosti do sebe vnořeny inkluzí, lze posloupnost oblastí reprezentovat konečným objektem. V paměti počítače může k tomuto účelu sloužit např. celočíselné pole o velikosti hrací plochy, kde číslo na pozici odpovídající určitému místu vyjadřuje index nejmenší oblasti z posloupnosti, do které místo patří.

Konečně se dostáváme k definici pojmu schéma zón relevance. Vyslovíme jej nejprve v plné obecnosti pro libovolnou vlastnost pozice a konkrétním případům zóny pro útok a pro obranu se budeme věnovat později.

**Definice 8.** Schéma zón relevance:
Buď $\Phi$ libovolná vlastnost pozic obecně závisející i na perspektivě hráče. Dále nechť $Pos$ je určitá pozice a $\pi$ hráč. Pak řekneme, že posloupnost oblastí $\mathbf{X} = (X_1, X_2, \ldots)$ je *schématem zón relevance řádu* $r$ pro $(Pos, \pi, \Phi)$, pokud

- platí $\Phi(Pos, \pi)$ a
- buď $r = 1$ a pro libovolný tah $\pi/y$ a pozici $Pos'$ takové, že $Pos \xrightarrow{\pi/y} Pos'$, platí: když $y \notin X_1$, pak $\Phi(Pos', \pi)$,
- nebo $r > 1$ a
  - posloupnost oblastí $\mathbf{X}$ je schématem zón relevance řádu $r'$ pro $(Pos, \pi, \Phi)$ pro každé $r' < r$ a
  - pro libovolný tah $\pi/y$ a pozici $Pos'$ takové, že $Pos \xrightarrow{\pi/y} Pos'$, platí: když $y \notin X_r$, pak $\mathbf{X}$ je schématem zón relevance řádu $r - 1$ pro $(Pos', \pi, \Phi)$.

Intuitivně vzato oblast $X_1$ schématu „chrání“ pozici před ztrátou platnosti vlastnosti $\Phi$ při tazích hráče $\pi$. Oblasti s vyšším indexem chrání oblasti s indexem nižším před ztrátou jejich ochranné funkce (viz též lemma 9 bod (c)). Řád schématu zón relevance potom vyjadřuje, kolik členů posloupnosti je ve skutečnosti platných a ochrany se účastní. V ideálním případě bude naše schéma zón úplné, tj. bude schématem zón relevance řádu $r$ pro každé $r \geq 1$.

Uvědomme si, že ochrana platnosti predikátu zajišťovaná zónou funguje jen jedním směrem. Obecně tedy neplatí, že by tah hráče $\pi$ do oblasti $X_1$ musel nutně platnost predikátu $\Phi$ v nové pozici zrušit. Zóna je jen jakousi horní aproximací skutečné oblasti míst, která vlastnost $\Phi$ ovlivňují, a proto se vždy budeme snažit konstruovat naše zóny co nejmenší.[3]

Při práci s konkrétními realizacemi zón relevance pro různé vlastnosti pozic je nutné si uvědomit vztah predikátu $\Phi$ v definici a účelu použití zóny, které jsou v jistém smyslu protichůdné. Například zóna pro hledání úspěšných útočných tahů musí být totiž zavedena pomocí predikátu charakterizujícího neúspěch útoku, neboť pouze hraním do zóny bude možné úspěšný útok založit.

Následující lemma ukazuje vztah schémat zón relevance a dříve zavedených operací nad příslušnými posloupnostmi.

**Lemma 9.** *Obecné vlastnosti schémat zón relevance:*

*Nechť* $\mathbf{X}$ *a* $\mathbf{Y}$ *jsou schémata zón relevance řádu* $r$ *pro* $(Pos, \pi, \Phi)$ *resp. pro* $(Pos, \pi, \Psi)$.

*(a) Každá posloupnost oblastí* $\mathbf{X}'$ *taková, že* $\mathbf{X} \subseteq \mathbf{X}'$*, je schématem zón relevance řádu* $r$ *pro* $(Pos, \pi, \Phi)$.

*(b)* $\mathbf{X} \cup \mathbf{Y}$ *je schématem zón relevance řádu* $r$ *pro* $(Pos, \pi, \Phi \& \Psi)$.

*(c) Nechť* $k < n$*. Definujeme-li predikát* $Meta(\mathbf{X}, \Phi, k)(Pos, \pi)$ *jako „*$\mathbf{X}$ *je schématem zón relevance řádu* $k$ *pro* $(Pos, \pi, \Phi)$*“, pak* $u^k(\mathbf{X})$ *je schématem zón relevance řádu* $r - k$ *pro* $(Pos, \pi, Meta(\mathbf{X}, \Phi, k))$[4].

Formální důkaz lemmatu spočívá v jednoduchém ověření všech bodů definice schématu zóny relevance. Pro ověřování třetího bodu je nutno používat indukce podle $r$, v případě tvrzení (b) se navíc využije předem dokázaný bod (a).

## 4.3 Zóny pro útok a obranu – úvod

Dříve než přistoupíme k popisu zón relevance pro útok a obranu, zavedeme užitečné zkratky pro pojmy související s lambda prohledáváním v kontextu her s kameny. Připomeňme, že tyto pojmy byly definovány v minulé kapitole.

**Definice 10.** Zkratky pro lambda-pojmy:

[3] Na neformální úrovni budeme v textu termín schéma zón relevance někdy zkracovat na zóna relevance či jen zóna při zachování významu.

[4] Kde $u^k(\mathbf{X})$ je $k$-násobná aplikace operátoru odvinutí.

- Hodnotu $\lambda$-stromu úrovně $n$ vystavěného v pozici $Pos$, kdy za hráče na tahu a tedy útočníka bereme hráče $\pi$, budeme zkracovat zápisem $\lambda(Pos, \pi, n)$.
- Množinou míst útočných $\lambda$-tahů pro daného hráče v dané pozici je $\Lambda_a^n(Pos, \pi)$.
- Množinou míst obranných $\lambda$-tahů pro daného hráče v dané pozici je $\Lambda_d^n(Pos, \pi)$[5].

Schémata útočných a obranných zón zavedeme podle očekávání jako zóny relevance pro určité specifické predikáty. Predikát útoku úrovně $n$ budeme značit $A_n$ a predikát obrany, též parametrizovaný úrovní $n$, symbolem $D_n$. Definujeme je takto:

$$\begin{aligned} A_n(Pos, \pi) &\equiv \lambda(Pos, \pi, n) = 0 \\ D_n(Pos, \pi) &\equiv \lambda(Pos, o(\pi), n) = 1. \end{aligned}$$

*Poznámka.* Již dříve jsme si uvědomili, jaký je vztah definujícího predikátu s účelem příslušné zóny. Při hraní mimo nejvnitřnější oblast schématu zůstává platnost predikátu nezměněna, a tedy pouze tím, že hráč zahraje naopak do této oblasti, má šanci platnost predikátu změnit. V případě predikátu $A_n$ to znamená, že by v nové pozici mohl založit $\lambda$-strom s hodnotou 1, v případě predikátu $D_n$, že se mu možná podaří zastavit útok, kterým protivník hrozí, a způsobit, že v nové pozici bude hodnota protivníkova $\lambda$-stromu 0.

Ve světle předchozí poznámky je nyní již zřejmé následující lemma:

**Lemma 11.** *Útočné a obranné zóny pro lambda prohledávání:*

*(a) Nechť posloupnost oblastí* $\mathbf{X} = (X_1, X_2, \ldots)$ *je schématem zón relevance libovolného řádu* $r \geq 1$ *pro* $(Pos, \pi, A_n)$*, pak* $\Lambda_a^{n+1}(Pos, \pi) \subseteq X_1$.

*(b) Nechť posloupnost oblastí* $\mathbf{X} = (X_1, X_2, \ldots)$ *je schématem zón relevance libovolného řádu* $r \geq 1$ *pro* $(Pos, \pi, D_n)$*, pak* $\Lambda_d^{n+1}(Pos, \pi) \subseteq X_1$.

Můžeme si všimnout, že se v lemmatu využívá jen nejvnitřnější oblasti zóny. Dalo by se proto namítnou, že jsme složitý pojem schématu zón relevance definovali zbytečně. Tak tomu ale není. Význam schémat se totiž projeví v dalších sekcích kapitoly, kdy budeme kombinací dílčích schémat vytvářet schémata nová a kde oblasti „vyšší úrovně“ sehrají svou důležitou roli.

[5]Hráč $\pi$ je zde tedy obráncem a $o(\pi)$ útočníkem – na rozdil od definice utocnych míst.

## 4.4 Predikáty ukončení hry

Nyní si představíme dva důležité predikáty, s jejichž pomocí budeme konstruovat zóny relevance pro útok a obranu lambda úrovně 0. Oba predikáty se budou týkat možnosti bezprostředního ukončení hry a jim příslušná schémata zón budeme umět popsat za předpokladu stability uvažované hry.

První predikát, který nazveme *LosingInNext*, vyjadřuje, že uvažovanému hráči v dané pozici hrozí „prohra v příštím tahu“, ledaže by pozice sama již byla koncová[6], neboli formálně:

$$LosingInNext(Pos, \pi) \equiv (\exists x) Pos \xrightarrow{o(\pi)/x} \pi(-1) \vee \mathcal{T}(Pos).$$

O konstrukci schématu zón relevance pro tento predikát hovoří následující věta, kterou je velmi snadné dokázat.

**Věta 12.** *Nechť ve stabilní hře v určité pozici Pos hráč $\pi$ prohraje v příštím tahu v místě $x$. Pak pro libovolné $r \geq 1$ je konstantní posloupnost oblastí $\mathbf{X} = (\{x\}, \{x\}, \ldots)$ schématem zón relevance řádu $r$ pro $(Pos, \pi, LosingInNext)$.*

*Důkaz.* První požadavek definice schématu zón relevance, tedy platnost příslušného predikátu v dané pozici, je splněn podle předpokladu. Druhá vlastnost, kterou ověřujeme, tedy fakt

$$Pos \xrightarrow{\pi/y} Pos' \,\&\, x \neq y \,\&\, \neg\mathcal{T}(Pos') \;\Rightarrow\; Pos \xrightarrow{o(\pi)/x} \pi(-1),$$

je přímým důsledkem stability hry. Nakonec si už zbývá jen uvědomit, že stabilitu lze znovu aplikovat i ve vzniklé pozici $Pos'$ a naše posloupnost oblastí $\mathbf{X}$ je tedy schématem zón relevance libovolného řádu $r \geq 1$ pro $(Pos, \pi, LosingInNext)$. □

Druhým stavebním kamenem pro tvorbu schémat zón relevance, jak pro útok, tak i pro obranu, bude v našich hrách predikát *NotWonYet*, který o pozici a hráči říká, že v dané pozici hráč (ještě) není vítězem. Formálně:

$$NotWonYet(Pos, \pi) \equiv \neg\mathcal{T}(Pos) \vee \mathcal{V}(Pos, \pi) \neq \pi(1).$$

Zabývejme se nyní otázkou, jak by mohlo vypadat schéma zón relevance pro tento predikát, a uvědomme si, že je žádoucí, hledat takové schéma co nejmenší. Je zřejmé, že nás budou zajímat především místa hrací plochy, kde lze výhry dosáhnout za málo tahů, navíc se omezíme jen na taková, že jejich obsazení je pro dosažení výhry opravdu podstatné. Tím se dostáváme k následujícím pojmům:

[6] Tato druhá podmínka má technický charakter. Dovoluje nám prozatím odhlédnout od případné možnosti inverzí.

**Definice 13.** Vítězné množiny pozic a $PPV$:

- Zápisem $Pos \xrightarrow{\pi/X} \pi(1)$, kde $X$ je množina míst, budeme vyjadřovat, že pro nějaké uspořádání $x_1, \ldots, x_k$ prvků množiny $X$, přechází pozice $Pos$ tahy hráče $\pi$ v místech množiny $X$ v určeném pořadí postupně až do koncové pozice s hodnotou výhra pro hráče $\pi$[7].

- Řekneme, že místo $x$ se v pozici $Pos$ *podstatně podílí na výhře* za $k$ tahů, pokud

$$(\exists X)[x \in X \,\&\, |X| = k \,\&\, Pos \xrightarrow{\pi/X} \pi(1) \,\&\, (\forall X' \subset X)\neg(Pos \xrightarrow{\pi/X'} \pi(1))].$$

- Množinu všech míst, která se v pozici $Pos$ podstatně podílí na výhře hráče $\pi$ za *nejvýše* $k$ tahů budeme značit $PPV(Pos, \pi, k)$.

Vlastnost místa $x$ podstatně se podílet na výhře říká, že místo je prvkem nějaké oblasti $X$, která při obsazení poslouží útočníkovi k vítězství, a že se navíc jedná o minimální oblast (vzhledem k inkluzi) s touto charakteristikou. Říkáme, že oblast $X$ tento fakt místu $x$ dosvědčuje.

Je dobré si uvědomit, že v úvahách o predikátu $NotWonYet$ hrají roli pouze tahy jednoho hráče. Můžeme to chápat tak, že druhý hráč se neustále svého tahu vzdává, a pokud se ani v takto zjednodušených podmínkách nepodaří útočníkovi svého cíle dosáhnout, tím spíše by selhal ve skutečné hře.

Následuje základní věta této sekce, která svazuje pojem podstatně se podílet na výhře s predikátem $NotWonYet$. Jejím předpokladem je opět stabilita hry, kterou jsme zavedli na straně 36.

**Věta 14.** *Konstrukce zón relevance pro predikát NotWonYet:*
*Nechť ve stabilní hře v určité pozici $Pos$ hráč $\pi$ ještě nedosáhl výhry. Pak pro libovolné $r \geq 1$ je posloupnost oblastí $\mathbf{X} = (X_1, X_2, \ldots)$, kde $X_i = PPV(Pos, \pi, i)$, schématem zón relevance řádu $r$ pro $(Pos, \pi, NotWonYet)$.*

*Důkaz.* Nejprve si uvědomíme, že přímo z definice pojmu $PPV$ získáváme monotonii $X_i \subseteq X_{i+1}$, a $\mathbf{X}$ je tedy korektně definovaná posloupnost oblastí.

Za druhé, podle předpokladu v pozici $Pos$ hráč $\pi$ ještě nedosáhl výhry, a proto platí $NotWonYet(Pos, \pi)$.

Dále postupujeme indukcí podle $r$. Nechť nejprve $r = 1$, $Pos \xrightarrow{\pi/y} Pos'$ a $y \notin X_1$. Vidíme, že $y$ se nepodílí na výhře za jeden tah, což především znamená, že tah $\pi/y$ není vítězný, a máme tedy snadno $NotWonYet(Pos', \pi)$.

[7] K přechodu do vítězné pozice nemusí nutně dojít až po zahrání posledního tahu v místě $x_k$.

Nyní dokazujme tvrzení pro $r+1$ s tím, že již platí pro $r$. Opět vyjdeme z předpokladu $Pos \xrightarrow{\pi/y} Pos'$ a dále budeme postupovat nepřímo. Není-li naše posloupnost oblastí $\mathbf{X}$ schématem zón relevance řádu $r$ pro $(Pos', \pi, NotWonYet)$, musí se podle indukčního předpokladu oblast $X_r = PPV(Pos, \pi, r)$ lišit od $PPV(Pos', \pi, r)$. Máme tedy buď $PPV(Pos, \pi, r) \setminus PPV(Pos', \pi, r) \neq \emptyset$ nebo $PPV(Pos', \pi, r) \setminus PPV(Pos, \pi, r) \neq \emptyset$. Až z jedné i druhé možnosti odvodíme $y \in X_{r+1}$, bude důkaz hotov[8].

Nechť nejprve nastává první možnost a zvolme $a \in PPV(Pos, \pi, r) \setminus PPV(Pos', \pi, r)$ a oblast $X$, která to dosvědčuje, tj.

$$a \in X \,\&\, |X| = r \,\&\, Pos \xrightarrow{\pi/X} \pi(1) \,\&\, (\forall X' \subset X) \neg (Pos \xrightarrow{\pi/X'} \pi(1)).$$

Kdyby bylo $y \in X$, máme $y \in PPV(Pos, \pi, r) \subseteq PPV(Pos, \pi, r+1)$ a jsme pro tento případ hotovi. Jinak víme, že vzhledem ke stabilitě naší hry je i $Pos' \xrightarrow{\pi/X} \pi(1)$, ale díky předpokladu není v pozici $Pos'$ množina $X$ minimální, tj. máme $X' \subset X$ takovou, že $Pos' \xrightarrow{\pi/X'} \pi(1)$. Vrátíme-li se zpět do pozice $Pos$, víme, že pro $X'' = X' \cup \{y\}$ platí $Pos \xrightarrow{\pi/X''} \pi(1)$. Můžeme nyní mezi podmnožinami $X''$ hledat minimální vítěznou množinu $X'''$ pro pozici $Pos$. Jisté je, že tato množina $X'''$ bude obsahovat místo $y$, jinak by to byla vlastní podmnožina $X$ a nemohla by být vítězná. Dále vidíme, že $|X'''| = r_0 \leq r$ a tedy $x \in PPV(Pos, \pi, r_0) \subseteq PPV(Pos, \pi, r+1)$.

Pro druhou možnost zafixujme $a \in PPV(Pos', \pi, r) \setminus PPV(Pos, \pi, r)$ a oblast $X$, která to dosvědčuje, tj.

$$a \in X \,\&\, |X| = r \,\&\, Pos' \xrightarrow{\pi/X} \pi(1) \,\&\, (\forall X' \subset X) \neg (Pos' \xrightarrow{\pi/X'} \pi(1)).$$

Uvažme nyní množinu $\bar{X} = X \cup \{y\}$ v kontextu pozice $Pos$. Určitě platí $y \in \bar{X}$, $|\bar{X}| = r+1$ a též $Pos \xrightarrow{\pi/\bar{X}} \pi(1)$. Zbývá si uvědomit, že žádná oblast $\bar{X}'$ z vlastních podmnožin $\bar{X}$ na vítězství v pozici $Pos$ nestačí. Ve stabilní hře by taková oblast totiž musela stačit na výhru i v pozici $Pos'$, a proto by musela $\bar{X}' = X$. Jenže to by znamenalo, že $a \in PPV(Pos, \pi, r)$, což není možné. □

Hledáním množin $PPV$ v námi zkoumaných hrách se budeme zabývat v kapitole 5, kde také vyložíme několik příkladů, které snad pomohou ještě více objasnit tento pojem.

[8] V důkazu se implicitně předpokládá, že pozice $Pos'$ není koncová. Kdyby tomu tak totiž bylo, je buď hodnotou pozice $Pos'$ výhra hráče $\pi$, a tím pádem $y \in X_1 \subseteq PPV(Pos, \pi, r+1)$, nebo svým tahem $\pi/y$ hráč remizuje, či dokonce prohrává, a pak je schématem zón relevance řádu $r$ pro $(Pos', \pi, NotWonYet)$ jakákoliv posloupnost oblastí, tedy speciálně i ta naše $\mathbf{X}$.

## 4.5 První krok

Nyní již máme k dispozici všechny prostředky k tomu, abychom dokázali zkonstruovat schémata zón relevance pro útok a obranu na nejnižší úrovni, tedy schémata pro predikáty $A_0$ a $D_0$. Zabývejme se nejprve prvním z nich a uvědomme si ekvivalenci:

$$\begin{aligned} A_0(Pos, \pi) &\equiv \lambda(Pos, \pi, n) = 0 \\ &\equiv \text{„}\emptyset \text{ je schématem zón relevance řádu 1} \\ &\qquad \text{pro } (Pos, \pi, NotWonYet)\text{“}. \end{aligned}$$

Pro implikaci „shora dolů“ stačí přečíst tvrzení $A_0(Pos, \pi)$ jako „hráč $\pi$ nemůže v pozici $Pos$ vyhrát za jeden tah“ a následně probrat definici schémat zón relevance (nejkomplikovanější třetí bod definice se ani neuplatní, neboť mluvíme o schématu řádu 1). Pro implikaci opačnou využijeme lemmatu 11. S jeho pomocí získáme $\Lambda_a^0(Pos, \pi) = \emptyset$, a tudíž zřejmě $\lambda(Pos, \pi, n) = 0$.

Vyjdeme-li nyní z právě dokázaného výsledku, můžeme použít lemma 9 (c) a uzavřít: Je-li $\mathbf{X}$ schématem zón relevance řádu $r$ pro $(Pos, \pi, NotWonYet)$, je $u(\mathbf{X})$ schématem zón relevance řádu $r - 1$ pro $(Pos, \pi, A_0)$.

V případě predikátu pro obranu je naše strategie obdobná. Máme totiž:

$$\begin{aligned} D_0(Pos, \pi) &\equiv \lambda(Pos, o(\pi), n) = 1 \\ &\equiv NotWonYet(Pos, \pi) \,\&\, LosingInNext(Pos, \pi). \end{aligned}$$

Uvědomit si, že ekvivalence platí, je opět velmi jednoduché. Skutečnost $\lambda(Pos, o(\pi), n) = 1$ prostě znamená, že hráč $\pi$ zatím ještě nevyhrál a hrozí mu prohra v příštím tahu. Jinými slovy obránce může nyní buď zkusit sám vyhrát (realizuje inverzi), nebo se pokusit hrozbu blokovat.

Nyní opět využijeme lemmatu 9, tentokrát bodu (b), abychom mohli shrnout: Je-li $\mathbf{X}$ schématem zón relevance řádu $r$ pro $(Pos, \pi, NotWonYet)$ a $\mathbf{Y}$ schématem zón relevance řádu $r$ pro $(Pos, \pi, LosingInNext)$, je posloupnost oblastí $\mathbf{X} \cup \mathbf{Y}$ schématem zón relevance řádu $r$ pro $(Pos, \pi, D_0)$.

Ačkoliv jsme se při zavádění zón relevance pro predikáty $A_0$ a $D_0$ odvolávali na postupy tvorby zón pro *LosingInNext* a *NotWonYet*, které umíme sestrojit pouze v kontextu stabilních her, nebyly naše úvahy nyní již na stabilitě závislé. Pokud by tedy byl vyvinut jiný postup konstrukce zón pro predikáty ukončení hry, který by fungoval i bez předpokladu stability, šlo by stále myšlenky předvedené v této sekci použít.

## 4.6 Indukční krok

Až doposud neměla naše práce se schématy zón relevance velké praktické opodstatnění, neboť ačkoliv jsme manipulovali s celými posloupnostmi oblastí, nakonec byla vždy využita jen jejich první složka (viz lemma 11). Nyní, při popisu konstrukce zón relevance pro útok a obranu na obecné $\lambda$-úrovni $n$, se situace změní, neboť bude nutné využít schémata v jejich plném rozsahu.

Dříve, než vyslovíme a dokážeme věty o konstrukci schémat zón relevance pro predikáty $D_n$ a $A_n$, větu 15, resp. 16, bude dobré upozornit na jejich induktivní ráz. Zavádíme v nich obrannou, resp. útočnou, zónu pozice pro určitou úroveň lambda pomocí obranných, resp. útočných, zón pozic nižší úrovně a nebo zón pozic s menším důkazovým stromem. Jelikož jsou uvažované stromy konečné, lze se vždy takovým sestupem dopracovat až k $\lambda$-stromům úrovně 0, pro něž již příslušné zóny umíme konstruovat přímo.

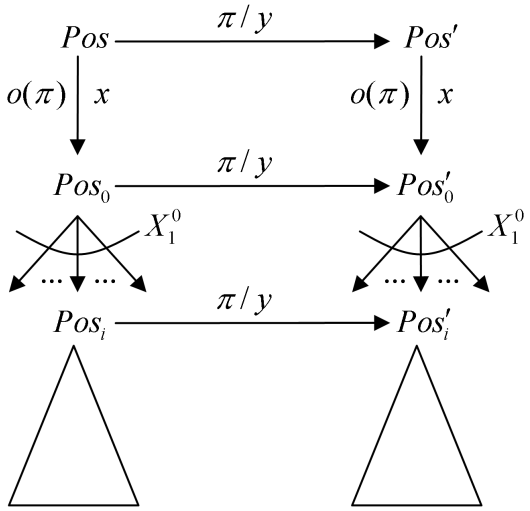
Abychom zbytečně nekomplikovali následující demonstrace mnoha odbočkami a rozbory speciálních případů, budeme až do konce této sekce předpokládat, že hra, o které uvažujeme, má velmi specifická pravidla. Při těchto pravidlech nedovolíme obránci zvítězit (bude jen odrážet hrozby), zakážeme v nich remízu a také v nich nepřipustíme tahy typu sebevražda, kterými by hráč, jenž je provádí, mohl prohrát. V dalších sekcích, poté, co nám tato omezení dovolí jasně vyložit základní ideje důkazu, se k problému vrátíme v plné obecnosti a zdůvodníme, jak lze věty upravit a zanedbané eventuality do nich zahrnout.

**Věta 15.** *Nechť platí $D_n(Pos, \pi)$, neboli $\lambda(Pos, o(\pi), n) = 1$, a nechť $o(\pi)/x$ je příslušný vítězný $\lambda_a^n$-tah, tj. $Pos \xrightarrow{o(\pi)/x} Pos_0$ a $\lambda(Pos_0, o(\pi), n-1) = 1$. Dále nechť $\mathbf{X}^0$ je schématem zón relevance řádu $r+1$ pro $(Pos_0, \pi, D_{n-1})$, přičemž $X_1^0 = \{x_1, x_2, \ldots, x_k\}$. Nechť $Pos_0 \xrightarrow{\pi/x_i} Pos_i$, kde $(1 \leq i \leq k)$. Pro každou pozici $Pos_i$ platí, že $\lambda(Pos_i, o(\pi), n) = 1$, a můžeme tedy předpokládat, že $\mathbf{X}^i$ jsou schémata zón relevance řádu $r$ pro $(Pos_i, \pi, D_n)$. Pak posloupnost oblastí $\mathbf{X}$ definovaná jako $\{x\} \cup u(\mathbf{X}^0) \cup \bigcup_{i=1}^{k} \mathbf{X}^i$ je schéma zón relevance řádu $r$ pro $(Pos, \pi, D_n)$.*

*Důkaz.* Víme, že platí $D_n(Pos, \pi)$, tedy první požadavek z definice schématu zón relevance je splněn podle předpokladu.

Dále se zabývejme druhým požadavkem a zvolme místo $y \notin X_1$ tak, že $Pos \xrightarrow{\pi/y} Pos'$. Ukážeme, že $\lambda(Pos', o(\pi), n) = 1$. Nejprve si uvědomíme, že $x \neq y$. Místo $x$ je tedy v pozici $Pos'$ volné a útočník na něj může zahrát: $Pos' \xrightarrow{o(\pi)/x} Pos'_0$. Zřejmě též $Pos_0 \xrightarrow{\pi/y} Pos'_0$, neboť na pořadí pokládání kamenů nezáleží. Jelikož $y \notin X_2^0$, platí dále, že $\mathbf{X}^0$ je schématem zón relevance řádu 1 pro $(Pos'_0, \pi, D_{n-1})$, z čehož za prvé dostáváme, že $\lambda(Pos'_0, o(\pi), n -$

1) = 1 a tah $o(\pi)/x$ je tedy $\lambda_a^n$-tahem i v pozici $Pos'$, a za druhé podle lemmatu 11 vidíme, že $\Lambda_d^n(Pos'_0, \pi) \subseteq X_1^0$. Ať ale použije obránce v pozici $Pos'_0$ tah na jakémkoliv z míst $x_i \in X_1^0$, aby dosáhl pozice $Pos'_i$, tedy aby $Pos'_0 \xrightarrow{\pi/x_i} Pos'_i$, pak fakt, že opět platí i $Pos_i \xrightarrow{\pi/y} Pos'_i$ a navíc $y \notin X_1^i$ zaručí, že $\lambda(Pos'_i, \pi, n) = 1$. Můžeme proto uzavřít, že $\lambda(Pos', \pi, n) = 1$.




Obrázek 4.2: Ilustrace k důkazu věty 15.

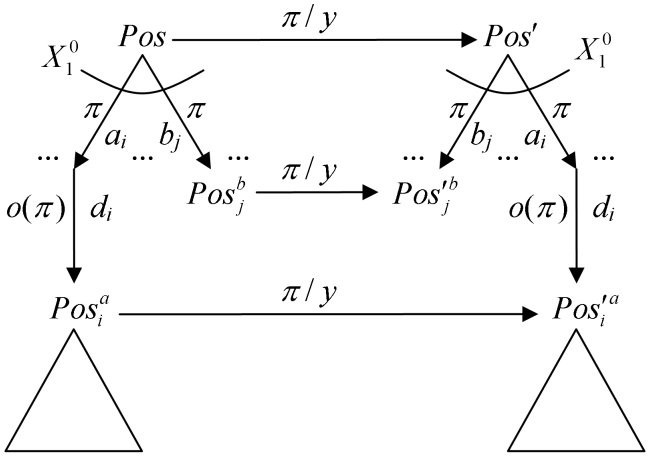
Důkaz třetího požadavku na schéma zón relevance provedeme indukcí podle $r$ a uvědomíme si, že první krok jsme vlastně dokazovali v minulém odstavci. Máme-li nyní $Pos \xrightarrow{\pi/y} Pos'$ za předpokladu $y \notin X_r$, zřejmě platí i $y \notin X_1$ a podle již dokázaného tedy stále $\lambda(Pos', \pi, n) = 1$, což lze navíc dosvědčit tím samým stromem jako v pozici $Pos$ (viz obrázek 4.2). Jelikož $y \notin X_{r+1}^0$, je $\mathbf{X}^0$ schéma zón relevance řádu $r$ pro $(Pos'_0, \pi, D_{n-1})$. Podobně $y \notin X_r^i$, a tedy $\mathbf{X}^i$ jsou schémata zón relevance řádu $r-1$ pro $(Pos'_i, \pi, D_n)$ pro každé $(1 \leq i \leq k)$. Vidíme tedy, že v pozici $Pos'$ lze opakovat veškerou argumentaci věty znovu (právě jsme ověřili její předpoklady) pouze s novým řádem $r' = r - 1$. Tím je důkaz indukcí dokončen. □

**Věta 16.** *Nechť platí $A_n(Pos, \pi)$, neboli $\lambda(Pos, \pi, n) = 0$, a tím pádem též $\lambda(Pos, \pi, n-1) = 0$. Nechť $\mathbf{X}^0$ je schéma zón relevance řádu $r+1$ pro $(Pos, \pi, A_{n-1})$. Rozdělme si oblast $X_1^0$ na $a_1, a_2, \ldots, a_k$, místa útočných $\lambda^n$-tahů pozice $Pos$, a na $b_1, b_2, \ldots, b_l$, místa ostatní. Pro každý útočný tah $a_i$ $(1 \leq i \leq k)$ existuje obráncův tah $d_i$ tak, že $Pos \xrightarrow{\pi/a_i} Pos_i \xrightarrow{o(\pi)/d_i} Pos_i^a$ a $\lambda(Pos_i^a, \pi, n) = 0$. Nechť $\mathbf{Y}^i$ jsou schémata zón relevance řádu $r$*

*pro* $(Pos_i^a, \pi, A_n)$. *Dále máme* $Pos \xrightarrow{\pi/b_j} Pos_j^b$ *pro každé* $(1 \leq j \leq l)$ *a* $\lambda(Pos_j^b, \pi, n-1) = 0$. *Nechť* $\mathbf{Z}^j$ *jsou schémata zón relevance řádu* $r$ *pro* $(Pos_j^b, \pi, A_{n-1})$. *Pak posloupnost oblastí* $\mathbf{X} = u(\mathbf{X}^0) \cup \bigcup_{i=1}^{k} (\{d_i\} \cup \mathbf{Y}^i) \cup \bigcup_{j=1}^{l} \mathbf{Z}^j$ *je schéma zón relevance řádu* $r$ *pro* $(Pos, \pi, A_n)$.

*Důkaz.* Platnost $A_n(Pos, \pi)$, jakožto prvního z požadavků definice schématu zón relevance, je zajištěna podle předpokladu.

Dále dokazujme, že když $Pos \xrightarrow{\pi/y} Pos'$ a $y \notin X_1$, pak $\lambda(Pos', \pi, n) = 0$. Jelikož v takovém případě $y \notin X_2^0$, je posloupnost $\mathbf{X}^0$ schématem útočných zón řádu 1 pro $(Pos', \pi, n-1)$ a i v nové pozici $Pos'$ budeme podle lemmatu 11 místa útočných $\lambda^n$-tahů hledat pouze mezi $X_1^0$. Vzhledem k monotonii $X_1^0 \subseteq X_2^0$ není $y$ rovno žádnému z míst $a_i$, ani $b_j$ a navíc ani žádnému $d_i$. Můžeme proto podobně jako v důkazu věty 15 vytvořit „očárkovanou verzi" naší části stromu hry a ve vzniklém diagramu využívat vlastnosti komutativity pokládání kamenů – viz obrázek 4.3. Nyní už si stačí pouze uvědomit, že pro každé $(1 \leq i \leq k)$ platí $y \notin Y_1^i$ a tedy $\lambda(Pos_i'^a, \pi, n) = 0$ a také, že pro každé $(1 \leq j \leq l)$ platí $y \notin Z_1^j$ a tedy $\lambda(Pos_j'^b, \pi, n-1) = 0$ a žádné z míst $b_j$ se nestane místem $\lambda_a^n$-tahu pozice $Pos'$. Celkově tedy $\lambda(Pos', \pi, n) = 0$.




Obrázek 4.3: Ilustrace k důkazu věty 16.

K dokončení důkazu indukcí si stačí uvědomit, že při přechodu $Pos \xrightarrow{\pi/y} Pos'$ za předpokladu $y \notin X_r$ máme nejprve z monotonie $\lambda(Pos', \pi, n) = 0$ a tento fakt lze dosvědčit tím samým stromem jako v pozici $Pos$. Dále v takové situaci $y \notin X_{r+1}^0$ a tedy $\mathbf{X}^0$ je schéma zón relevance řádu $r$ pro $(Pos', \pi, A_{n-1})$, taktéž $y \notin Y_r^i$ a tedy $\mathbf{Y}^i$ jsou schémata zón relevance řádu

$r-1$ pro $(Pos'^a_i, \pi, A_n)$ pro každé $(1 \leq i \leq k)$ a konečně $y \notin Z^j_r$ a proto $\mathbf{Z}^j$ jsou schémata zón relevance řádu $r-1$ pro $(Pos'^b_j, \pi, A_{n-1})$ pro každé $(1 \leq j \leq l)$. Vidíme tedy, že v pozici $Pos'$ lze opakovat veškerou argumentaci znovu pouze s novým řádem $r' = r-1$. □

## 4.7 Ošetření inverzí a úprava pro dual-lambda search

Věty 15 a 16 jsme zatím kvůli názornosti dokázali pouze za jistých omezujících předpokladů týkajících se možného zakončení hry. Tento nedostatek budeme nyní postupně napravovat a začneme zamyšlením nad tím, jak důkazy obou vět ovlivní přidání možnosti obránce dosáhnout vlastní výhry.

Abychom si usnadnili vyjadřování, budeme se v následujících odstavcích odkazovat na symboly zavedené v důkazech zmiňovaných vět. Někdy proto bude nejspíš pro čtenáře nutné, aby se vrátil na předchozí strany a aby si symboly, spolu s jejich kontextem, připomněl.

Věta 15 se zabývá konstrukcí schématu zón relevance pro predikát $D_n$ a její hlavní myšlenka spočívá v předvedení, že obránce $\pi$ svým tahem v místě $y$ nemůže narušit útočníkův vyhrávací $\lambda$-strom, pokud $y$ nepatří do zkonstruované zóny. Problémem, který byl zatím opomíjen, je, zda nemůže obránce svým tahem $\pi/y$ sám vyhrát a tím argumentaci znehodnotit. Pokusíme se nyní zdůvodnit, že to není možné, a to konkrétně proto, že by takový tah musel být již dříve zahrnut mezi některou z dílčích zón účastnících se konstrukce. To zcela jistě platí v případě tahů $Pos_i \xrightarrow{\pi/y} Pos'_i$, neboť kdyby $\mathcal{T}(Pos'_i) \,\&\, \mathcal{V}(Pos'_i, \pi) = \pi(1)$, nemohlo by $y \notin X^i_1$ vzhledem k tomu, že $\mathbf{X}^i$ je podle předpokladu schématem zón relevance pro $(Pos_i, \pi, D_n)$. Podobné zdůvodnění funguje i u tahu $Pos_0 \xrightarrow{\pi/y} Pos'_0$ s odvoláním na schéma zón relevance $\mathbf{X}^0$.

Posledním tahem, který nám zbývá probrat, je $Pos \xrightarrow{\pi/y} Pos'$. Pokud je hra, o které uvažujeme, stabilní, nemůže ani zde nastat problém. Kdyby totiž $Pos \xrightarrow{\pi/y} \pi(1)$, ze stability bychom dostali též $Pos_0 \xrightarrow{\pi/y} \pi(1)$, což, jak již víme, není možné. Není-li naše hra stabilní, nutí nás dosavadní úvahy pozměnit znění věty 15 a rozšířit výslednou zónu $\mathbf{X}$ o zónu relevance pro $(Pos, \pi, NotWonYet)$, což pochopitelně diskutovanou eventualitu vyloučí.

Můžeme tedy uzavřít, že ve stabilních hrách platí věta 15 beze změny i v případě, že obránce může svým tahem vyhrát. V nestabilní hře je potom pro zachování její platnosti třeba definovat $\mathbf{X}$ jako $\{x\} \cup u(\mathbf{X}^0) \cup \bigcup_{i=1}^{k} \mathbf{X}^i \cup \mathbf{Y}$, kde $\mathbf{Y}$ je schéma zón relevance příslušného řádu pro $(Pos, \pi, NotWonYet)$.

Nyní je na čase zkontrolovat za nových podmínek, které umožňují obránci zvítězit, i platnost věty 16. Využijeme této příležitosti také k tomu, abychom vysvětlili, jak konstrukci zón relevance přizpůsobit pro variantu prohledávání dual-lambda search. Uvědomíme-li si totiž, že právě a pouze rozpoznáváním inverzí, tj. tahů, kterými obránce svým vítězstvím maří založený útok, se přístupy lambda prohledávání a dual-lambda liší, bude vyložení následujících myšlenek obzvláště snadné.

V důkazu věty 16 se útočníkovi $\pi$ nepodařilo v pozici $Pos$ zvítězit pomocí $\lambda$-stromu úrovně $n$, neboť obránce $o(\pi)$ všechny hrozby odvrátil. Doposud jsme neuvažovali případ, ve kterém by odvrácení hrozby spočívalo ve vlastním vítězství obránce, tedy, v řeči symbolů zavedených ve větě, existenci míst $a_0$ a $d_0$ a pozice $Pos_0$ takových, že $Pos \xrightarrow{\pi/a_0} Pos_0$ a $Pos_0 \xrightarrow{o(\pi)/d_0} \pi(-1)$.

Útok v místě $a_0$ selhal, neboť obránce sám zvítězil v následujícím tahu. To je speciální případ situace, která nastává v metodě dual-lambda search, když se určitý tah nestane $\mu_a$-tahem, neboť po jeho zahrání má obránce k dispozici $\mu$-strom s hodnotou 1 (viz sekce 3.9) – na pokus o útok naváže obránce svým úspěšným protiútokem. Otázka nyní zní, jak zajistit, aby zkonstruovaná zóna pro $(Pos, \pi, A_n)$ brala právě pospanou skutečnost na vědomí a aby hráč $\pi$ tahem mimo ni nemohl statut svého neúspěšného útoku změnit.

Řešení se nabízí v podobě použití výsledků spočtených již dříve. Víme-li totiž, že v pozici $Pos_0$ má obránce úspěšný strom pro útok, platí tedy vlastně $D_k(Pos_0, \pi)$ pro nějaké $k$, a můžeme předpokládat, že jsme dostali také příslušnou zónu, přesněji řečeno schéma zón relevance pro $(Pos_0, \pi, D_k)$. Přidáme-li toto schéma do námi konstruovaného výsledku $\mathbf{X}$, bude tím zaručeno, že i „v očárkované“ verzi stromu bude vlastnost úspěšného protiútoku obránce zachována. Jinými slovy ve výsledné zóně se vyskytnou taková místa, kde má útočník naději překazit obráncův protiútok, což je pochopitelně nutný předpoklad k tomu, aby mohl sám úspěšně zaútočit.

Právě popsané myšlenky formálně shrnuje následující úprava věty 16, kterou lze použít ke konstrukci zón pro útok jak pro lambda search tak i pro dual-lambda search.

**Věta 17.** *Nechť platí $A_n(Pos, \pi)$, neboli $\lambda(Pos, \pi, n) = 0$, a tím pádem též $\lambda(Pos, \pi, n-1) = 0$. Nechť $\mathbf{X}^0$ je schéma zón relevance řádu $r+1$ pro $(Pos, \pi, A_{n-1})$. Rozdělme si oblast $X_1^0$ na $a_1, a_2, \ldots, a_k$, místa útočných $\lambda^n$-tahů pozice $Pos$, na $b_1, b_2, \ldots, b_l$, místa, kde se útočníkovi nedaří hrozit, a na $c_1, c_2, \ldots, c_t$, kde je útočník sám ohrožen. Pro každý útočný tah $a_i$ $(1 \leq i \leq k)$ existuje obráncův tah $d_i$ tak, že $Pos \xrightarrow{\pi/a_i} Pos_i \xrightarrow{o(\pi)/d_i} Pos_i^a$ a $\lambda(Pos_i^a, \pi, n) = 0$. Nechť $\mathbf{Y}^i$ jsou schémata zón relevance řádu $r$ pro $(Pos_i^a, \pi, A_n)$. Dále máme $Pos \xrightarrow{\pi/b_j} Pos_j^b$ pro každé $(1 \leq j \leq l)$ a $\lambda(Pos_j^b, \pi, n-1) = 0$. Nechť $\mathbf{Z}^j$ jsou*

*schémata zón relevance řádu r pro* $(Pos_j^b, \pi, A_{n-1})$. *Navíc platí* $Pos \xrightarrow{\pi/c_s} Pos_s^c$ *pro každé* $(1 \leq s \leq t)$ *a* $\lambda(Pos_s^c, o(\pi), m_s) = 1$ *pro nějaké* $m_s < n$.[9] *Nechť* $\mathbf{W}^s$ *jsou schémata zón relevance řádu r pro* $(Pos_s^c, \pi, D_{m_s})$. *Pak posloupnost oblastí* $\mathbf{X} = u(\mathbf{X}^0) \cup \bigcup_{i=1}^{k} (\{d_i\} \cup \mathbf{Y}^i) \cup \bigcup_{j=1}^{l} \mathbf{Z}^j \cup \bigcup_{s=1}^{t} \mathbf{W}^s$ *je schéma zón relevance řádu r pro* $(Pos, \pi, A_n)$.

## 4.8 Ošetření remízy a omezování hloubky prohledávání

Další způsob ukončení hry, kterým jsme se zatím nezabývali, je remíza. Úplně obecně vzato je ukončení hry remízou rovnocenné výhře či prohře a může nastat po jakémkoliv tahu. Ve většině her ale nastává remíza až v situacích, kdy je jasné, že žádný z hráčů si svou výhru nebude moci vynutit, nebo když již jednoduše nelze hrát dále, přičemž o vítězi zatím nebylo rozhodnuto. Pro naše účely bude proto naprosto dostačující, vypořádáme-li se s možností ukončení hry remízou jen ve speciálním případě, a to konkrétně, když remíza nastává po určitém předem daném počtu tahů $d$ od začátku hry, pokud do té doby nebylo rozhodnuto o vítězi.[10]

Podívejme se opět na věty 15 a 16, abychom zjistili, jak jejich platnost přidání možnosti remízy ovlivní. U věty 16 je situace velmi jednoduchá. Pokud je totiž neúspěch útočníkova $\lambda$-stromu vystavěného v pozici $Pos$ zčásti způsoben také tím, že byla v určité hloubce hra ukončena remízou, bude strom stavěný v pozici $Pos'$ neúspěšný tím spíše, neboť remíza nastane ještě o tah dříve. (Limit se o 1 sníží kvůli tahu $Pos \xrightarrow{\pi/y} Pos'$.) V případě věty 15, kdy se snažíme zachovávat $\lambda$-strom naopak úspěšný, tento argument ale nelze použít. Ve skutečnosti není těžké si uvědomit, že tato věta za uvedených podmínek ani neplatí. Řešení tentokrát spočívá v tom, že přestaneme limit $d$ chápat, tak jako doposud, jako číslo vyplývající z podstaty pravidel hry[11], a místo toho jej přidáme mezi parametry prohledávací procedury. Když potom v pozici $Pos$ nalezneme útočníkův úspěšný strom při limitu $d-1$, nebude již nic bránit zachování tohoto stromu v pozici $Pos'$ při limitu $d$.

Je dobré si uvědomit, že výše popsané úvahy lze snadno aplikovat i v případě, že naše hra sice nemůže skončit remízou, ale my bychom rádi omezili hloubku prohledávání našeho algoritmu např. proto, že bez takového ome-

[9] V případě metody lambda search bude vždy $m_s = 0$, neboť tento algoritmus inverze vyšší úrovně nehledá.

[10] Připomeňme, že v námi testovaných hrách může remíza nastat pouze v Piškvorkách a to po celkovém zaplnění hrací plochy.

[11] Např. v Piškvorkách vyplývá hodnota limitu z velikosti hrací plochy.

zení není šance se v rozumném čase dočkat výsledku. Za tímto účelem si můžeme práh $d$ stanovit uměle a po zahrání $d$ tahů oběma hráči hru vždy ukončit remízou. Vzdáme se tím sice úplnosti prohledávání v tom smyslu, že se algoritmu již nepodaří odhalit hodnotu zkoumané pozice, pokud by potřebný důkaz vyžadoval prohledat strom o hloubce větší než $d$, úvahy této sekce nám ale zaručí, že nevynecháme žádný $\lambda$-strom s malou hloubkou a že tedy prohledávání bude úplné alespoň vzhledem k pozměněným herním pravidlům.

## 4.9 Ošetření tahů typu sebevražda

Posledním speciálním případem ukončení hry, který ještě nebyl diskutován, je případ tahů typu sebevražda. Sebevražedné tahy připouští z námi testovaných her pouze hra AtariGo, a to navíc za velmi specifických podmínek. Nebudeme se zde proto pouštět do rozboru podobného dvěma předcházejícím, abychom přidáním dalších předpokladů případně mírnou úpravou konstruovaných zón zdůvodnili platnost vět 15 a 16 i při možnosti sebevražedných tahů. Místo toho na tomto místě pouze prohlásíme, že při implementaci zkoumaných metod pro hru AtariGo není třeba postup konstrukce zón relevance z důvodů přítomnosti sebevražedných tahů nijak upravovat. Důvodem je fakt, že místa na hrací ploše, která by kvůli přítomnosti sebevraždy mohla působit potíže, budou vždy již předem zahrnuta do příslušných zón z jiného důvodu, konkrétně se bude jednat o svobody ohrožených řetězců. Přesné zdůvodnění tohoto naznačeného tvrzení není sice složité, je ale poněkud zdlouhavé, a proto jej zde nebudeme uvádět.

## 4.10 Shrnutí

Shrňme nyní výsledky o konstrukci zón relevance dosažené v této kapitole. Zabývali jsme se v ní specifickým druhem her s jednoduchou strukturou, které jsme nazvali hry s kameny. Pro některá tvrzení jsme navíc potřebovali klást na hru další předpoklady, jimž jsme říkali stabilita. Nakonec bylo nutné opatrně zacházet s atypickými způsoby ukončení hry, jmenovitě s remízou a tahy typu sebevražda.

Dosáhli jsme následujících výsledků:

- Ve stabilních hrách umíme konstruovat zóny relevance pro predikáty *LosingInNext* a *NotWonYet*.

- S pomocí zón relevance pro predikáty *LosingInNext* a *NotWonYet* umíme vytvářet zóny pro predikáty $A_0$ a $D_0$, tedy pro útok a obranu v lambda prohledávání na úrovni 0.
- Zóny relevance pro predikáty $A_n$ a $D_n$ umíme konstruovat pomocí zón týchž predikátů s nižší úrovní $n$ nebo predikátů dosvědčených menšími stromy a to s následujícími doplněními:
  - Při konstrukci zóny pro predikát $D_n$ musíme předpokládat stabilitu hry, nebo vznikající zónu vždy obohacovat o zónu relevance pro predikát *NotWonYet* v dané pozici.
  - Při konstrukci zóny pro predikát $A_n$ využíváme v případě obráncových inverzí zóny pro predikát $D_k$ $(k < n)$, což je navíc způsob, jak definovat zóny relevance pro příslušné pojmy i v prohledávání dual-lambda.
  - Uvažujeme jen speciální druh remízy, která nastává po předem daném počtu tahů od začátku hry, pokud do té doby nedošlo k výhře jednoho z hráčů.
  - Tahy typu sebevražda jsme z našich úvah úplně vyloučili. Přesto lze popsanou metodu použít beze změny i ve hře AtariGo, o korektnosti je ale třeba se přesvědčit zvlášť.

# Kapitola 5

# Implementace prohledávací metody

V předchozích kapitolách jsme se seznámili s technikou lambda prohledávání a konstrukce zón relevance na teoretické úrovni. Ukázali jsme, jak jsou $\lambda$-stromy vyšší úrovně induktivně definovány s pomocí stromů úrovně nižší a jak lze také při konstrukci zón relevance sledovat tento induktivní ráz a vhodnou kombinací zón pro základní predikáty sestrojovat zóny pro predikáty odvozené. Úkolem této kapitoly bude zaměřit se na popsané techniky z implementačního hlediska a popsat fungování programu, který vznikl jako součást této práce za účelem testování zkoumaných metod.

Nejprve představíme kostru prohledávacího algoritmu lambda search, který jsme implementovali pomocí techniky prohledávání do hloubky a podrobně budeme demonstrovat, jak se v průběhu výpočtu zachází se zónami relevance. V sekci 5.2 se stručně zmíníme o úkolu tvorby objektu pro reprezentaci herní pozice v každé ze tří námi zkoumaných her. Technikám urychlování prohledávacího algoritmu se pak bude věnovat sekce 5.3. Na závěr kapitoly uvedeme výsledky, kterých se s pomocí našeho programu podařilo dosáhnout v jednotlivých hrách, a popíšeme přednosti a slabiny zkoumaného přístupu, jež se během testovaní podařilo identifikovat.

## 5.1 Prohledávací algoritmus

Při implementaci algoritmů lambda search a dual-lambda search jsme jako prohledávací paradigma zvolili metodu prohledávání do hloubky. Ačkoliv je možné prohledávat stromy her i jiným způsobem, je tento přístup využíván nejčastěji, a to nejen díky svým nízkým paměťovým nárokům, ale též proto, že jej lze velmi přímočaře realizovat pomocí rekurze. Navíc je k dispozici celá

řada technik jak prohledávání herních stromů do hloubky urychlovat.

Úkolem následujících odstavců bude podrobně rozebrat naši implementaci algoritmu lambda search[1] především se zaměřením na uplatnění zón relevance. Dříve, než ale přistoupíme k prezentaci samotného pseudokódu, je na místě několik poznámek.

Prohledávací algoritmus využívá při své práci služeb objektu *game*. Ten v sobě zapouzdřuje informace o jedné konkrétní pozici zkoumané hry. Dokáže odpovídat na dotazy, zda pozice je, či není koncová (metoda isGameOver), zda daný hráč dokáže dosáhnout výhry v příštím tahu (metoda hasImmediateWin), a umí vracet schémata zón relevance pro základní predikáty $A_0$ a $D_0$ (metody basicAZone resp. basicDZone). Neméně důležité jsou metody doMove a undoMove, které simulují zahrání resp. odčinění určitého tahu a s jejichž pomocí prohledávací algoritmus prochází strom hry. Nutno dodat, že objekt *game* je v algoritmu chápán jako globální a algoritmus vlastně analyzuje tu pozici, kterou objekt *game* reprezentuje v okamžiku jeho spuštění.

Již v kapitole 4 jsme se zmínili o tom, jak v počítači reprezentovat schémata zón relevance, tedy určité posloupnosti oblastí na hrací ploše řazené do sebe inkluzí. K tomuto účelu využíváme celočíselné pole o velikosti hrací plochy, kde číslo na pozici odpovídající určitému místu vyjadřuje index nejmenší oblasti z posloupnosti, do které místo patří. Nad schématy zón relevance (říkejme jim dále jen zóny), jsme definovali binární operaci sjednocení (které znamená sjednocení jednotlivých oblastí v posloupnostech po složkách) a unární odvinutí (které si lze představit jako zapomenutí první oblasti v posloupnosti). Lze si snadno uvědomit, že při výše zmíněné reprezentaci je možno operaci sjednocení implementovat jednoduše pomocí minima a odvinutí jako odčítání jedničky.[2]

Podobně jako stav hry chápeme i zóny relevance v pseudokódu jako objekty. Při manipulaci s nimi se uplatní následující metody. Operaci odvinutí odpovídá metoda unwind. Sjednocení jedné zóny s druhou spolu s uložením výsledku na místo první z nich lze provádět pomocí metody merge. Metoda add přidá předaný prvek do všech oblastní posloupnosti a funkce singleton vytvoří konstantní posloupnost složenou z jednoprvkových oblastí obsahujících předaný prvek. Nakonec popišme důležitou metodu select, která z posloupnosti vydělí první složku, tedy onu množinu tahů, která nás nejvíce zajímá.

Nyní již můžeme konečně přistoupit k popisu vlastního algoritmu ve formě

[1] Verzi dual-lambda search se zde věnovat nebudeme. Využívá stejných myšlenek jen v poněkud komplikovanější podobě, neboť se musí postupně zabývat hrozbami obou hráčů.

[2] Přesněji: Reprezentuje-li pole $X$ posloupnost oblastí $\mathbf{X}$ tak, že číslo $X[i]$ určuje $\min_j i \in \mathbf{X}_j$ a podobně pro pole $Y$ a posloupnost $\mathbf{Y}$, pak pro pole $Z$ reprezentující posloupnost $\mathbf{Z} = \mathbf{X} \cup \mathbf{Y}$ platí $Z[i] = \min(X[i], Y[i])$ a pro pole $W$ reprezentující posloupnost $\mathbf{W} = u(\mathbf{X})$ máme $W[i] = max(1, X[i] - 1)$.

pseudokódu. Uvědomme si ale, že je v něm kvůli přehlednosti vynecháno několik podstatných detailů, např. kód pro omezování hloubky prohledávání nebo ošetření ukončení hry remízou či sebevražedným tahem.

```
LAMBDA(player, level)
 1  if game.hasImmediateWin(player) then
 2        return (1, game.basicDZone(player))
 3  aZone ← game.basicAZone(player)
 4  if level = 0 then
 5        return (0, aZone)
 6  for curLevel ← 1 to level do
 7        (result, zone) ← ATTACK(player, curLevel, aZone)
 8        if result = 1 then
 9              return (1, zone)
10        else
11              aZone ← zone
12  return (0, aZone)
```

Pro názornost je kód prohledávacího algoritmu rozdělen do tří procedur. Hlavní procedura LAMBDA je vstupním bodem celého prohledávání a využívá pomocných procedur ATTACK a DEFEND. Všechny tři procedury vrací jako svůj výstup dvojici hodnot. První složkou dvojice je vypočtený výsledek z oboru $\{0, 1\}$, druhou pak zóna, která výsledku odpovídá.

Procedura LAMBDA provádí lambda prohledávání iterativním způsobem. V podobě parametrů jí jsou předány hráč na tahu *player*, kterému budeme také říkat útočník, a číslo *level*, což je maximální $\lambda$-úroveň, do které má iterace postupovat. Úkolem procedury je rozhodnout, zda má v dané pozici útočník k dispozici úspěšný $\lambda$-strom úrovně nejvýše *level*. Pokud je tomu tak, vrátí procedura hodnotu 1 a zónu relevance pro predikát $D_{level}$, v opačném případě vrací 0 a zónu pro predikát $A_{level}$. Vrácené zóny poslouží volajícímu v prvním případě k hledání obranných tahů proti přítomné hrozbě, v případě druhém pro zakládání potenciálního útoku na vyšší úrovni, tedy k hledání tahů útočných.

Procedura začíná pokusem o útok na $\lambda$-úrovni 0 (řádky 1 a 2) – může-li útočník dosáhnout výhry v příštím tahu, a má-li tedy k dispozici $\lambda_a^0$-tah, vrací procedura 1 (úspěch) a zónu pro predikát $D_0$. Pokud není možno dosáhnout výhry přímo, a v dané pozici tedy platí naopak $A_0$, získáme od objektu *game* příslušnou zónu (řádka 3), a je-li 0 nejvyšší úrovní, kterou jsme se měli

zabývat, je taky tato zóna společně s číslem 0, příznakem neúspěchu, vrácena a procedura končí (řádky 4 a 5).

Dále zkouší procedura LAMBDA hledat úspěšný $\lambda$-strom na vyšších úrovních, k čemuž slouží **for**-cyklus na řádkách 6 až 11. Využívá k tomu proceduru ATTACK, jejímž úkolem je prohledat $\lambda$-strom zadané úrovně (viz parametr *curLevel*). Před zahájením $i$-té iterace cyklu vždy platí, že je ve zkoumané pozici splněn predikát $A_{i-1}$ a proměnná *aZone* obsahuje jemu příslušnou zónu relevance. Tuto zónu používá právě procedura ATTACK pro hledání útočných tahů. Je-li výsledkem volání procedury ATTACK v $i$-té iteraci cyklu úspěch, což znamená, že se nám podařilo nalézt $\lambda^i$-strom s hodnotou 1, ukončíme provádění cyklu a vracíme taktéž příznak úspěchu společně se získanou zónou, což je zóna relevance pro predikát $D_i$. V opačném případě využijeme vrácenou zónu k aktualizaci proměnné *aZone*, která tak bude obsahovat zónu pro predikát $A_i$, čímž se zajistí platnost invariantu cyklu pro další iteraci. Pokud procedura LAMBDA neskončí během provádění cyklu nebo dříve, uplatní se kód na poslední řádce a vrácena je hodnota 0 spolu s obsahem proměnné *aZone*, což je v tomto okamžiku zóna relevance pro predikát $A_{level}$.

Je možno si všimnout, že iterace v proceduře LAMBDA neslouží pouze k tomu, aby byl příslušný úspěšný strom nalezen na nejnižší možné úrovni a tedy s využitím minimálního množství výpočetních prostředků. Neméně důležitý je také fakt, že neúspěšná fáze vždy připraví zónu pro útok pro fázi následující, a z tohoto pohledu – chceme-li využívat zóny relevance pro útok – je tedy iterace nezbytná.

Jak již bylo naznačeno, úkolem procedury ATTACK je prohledávat $\lambda$-strom hráče *player* úrovně *level* s využitím zóny relevance *aZone*. Zároveň je během výpočtu připravována zóna nová pro případ, že by prohledávání skončilo neúspěchem. Zaměřme se ale nejprve na samotný průběh výpočtu s tím, že se ke konstrukci nové zóny vrátíme později.

Poté, co procedura ATTACK získá kandidáty na útočné tahy (řádka 1), jsou tito postupně v cyklu (3–13) testováni, zda se skutečně jedná o $\lambda_a^{level}$-tahy, či nikoliv. Test se provádí na řádce 5 pomocí rekurzivního volání procedury LAMBDA. Přesně podle definice je totiž testovaný tah $\lambda_a^{level}$-tahem, pokud má po jeho zahrání útočník k dispozici úspěšný $\lambda$-strom úrovně $level - 1$. Tahy, které testem neprojdou, se dále nezabýváme. Naopak za každý úspěšný je potenciálně volána procedura DEFEND, aby se zjistilo, zda obránce dokáže hrozbu odvrátit. Proceduře DEFEND je mj. předávána zóna spočtená při volání LAMBDA, což je obranná zóna příslušející dané hrozbě, a tu procedura DEFEND využije při hledání obranných tahů. Pokud se obránci hrozbu odvrátit nepodaří a DEFEND vrátí 1, je ukončena celá metoda ATTACK a vrácen úspěch (řádka 11). Jinak cyklus postupně prochází další kandidáty a skončí-li bez nalezení úspěšného útočného tahu, procedura ATTACK skončí vrácením

```
ATTACK(player, level, aZone)
 1  attackMoves ← aZone.select
 2  aZone.unwind
 3  for each a ∈ attackMoves do
 4        game.doMove(a, player)
 5        (result, zone) ← LAMBDA(player, level −1)
 6        if result = 1 then
 7              (result, zone) ← DEFEND(player, level, zone)
 8              if result = 1 then
 9                    zone.add(a)
10                    game.undoMove
11                    return (1, zone)
12        aZone.merge(zone)
13        game.undoMove
14  return (0, aZone)
```

příznaku neúspěchu (na řádce 14).

Proměnná *aZone* je během výpočtu procedury ATTACK aktualizována takovým způsobem, aby v případě, že procedura skončí neúspěchem, obsahovala zónu relevance pro predikát $A_{level}$. Konstrukce postupuje plně v souladu se zněním věty 16 kapitoly 4, a výsledná zóna se proto skládá z odvinuté zóny pro predikát $A_{level-1}$ (viz řádka 2) a z příspěvků jednotlivých nepovedených pokusů o úspěšný útočný tah (ty jsou k zóně připojovány na řádce 12).

Zóna vrácená v případě úspěchu, tedy zóna pro predikát $D_{level}$, je konstruována podle věty 15 a její hlavní část připraví již úspěšné volání procedury DEFEND. Úkolem procedury ATTACK je pak již jen připojit k zóně vítězný útočný tah (řádka 9).

Procedura DEFEND je volána na pozice, ve kterých útočník *player* hrozí obránci úspěšným $\lambda$-stromem úrovně $level-1$. S pomocí obranné zóny pro tuto hrozbu *dZone* zjišťuje, zda má ještě obránce šanci útok odvrátit. Zároveň se podílí na přípravě zóny nové, která bude využita, pokud prohledávání skončí úspěchem (útočníka).

Postup v proceduře DEFEND je velmi podobný tomu z procedury ATTACK. Nejprve jsou s využitím parametru *dZone* získáni kandidáti na obranné tahy (řádka 1) a poté je v cyklu (3–14) zkoušeno, zda některý z nich obránce nezachrání. Pokud takovou záchranou není vlastní vítězství obránce – nazývané v tomto kontextu inverze – které je testováno na řádkách 5 až 7, spustí se prostě procedura LAMBDA, aby rekurzivně prozkoumala příslušný

```
DEFEND(player, level, dZone)
 1  defensiveMoves ← dZone.select
 2  dZone.unwind
 3  for each d ∈ defensiveMoves do
 4        game.doMove(d, oponent(player))
 5        if game.isGameOver then
 6              game.undoMove
 7              return (0,singleton(d))
 8        (result, zone) ← LAMBDA(player, level)
 9        if result = 0 then
10              zone.add(d)
11              game.undoMove
12              return (0, zone)
13        dZone.merge(zone)
14        game.undoMove
15  return (1, dZone)
```

podstrom (řádka 8).[3] Pokud toto volání vrátí 0, tedy příznak neúspěchu útočníka, je možno proceduru DEFEND ukončit – obránce se ubránil (viz řádka 12). V opačném případě jsou v cyklu postupně zkoušeny další obráncovy alternativy, a teprve když jsou vyčerpány, končí procedura DEFEND s příznakem úspěchu útočníka (řádka 15).

Úpravy proměnné *dZone* v průběhu výpočtu procedury DEFEND, jmenovitě její odvinutí (řádka 2), a také připojování zón vrácených úspěšnými voláními procedury LAMBDA (řádka 13), zajistí, že je na konci výpočtu vrácena zóna relevance pro predikát $D_{level}$.[4] Podobně zóny vrácené v případě neúspěchu útočníka (řádky 7 a 12) pomáhají volající proceduře ATTACK vyrábět zónu pro predikát $A_{level}$. Ověření obou těchto tvrzení se odvolává na již zmiňované věty 15 a 16 kapitoly 4.

Na závěr si všimněme, že jsme v jistém smyslu v procedurách ATTACK a DEFEND realizovali prohledávání herního stromu metodou alfa-beta. Jelikož totiž vrcholy našeho stromu mohou nabývat pouze dvou hodnot (0 a 1), redukuje se vlastně alfa-beta prořezávání stromu hry na jednoduché pra-

[3] Je dobré si všimnout, že na tomto místě není třeba hledat obranné $\lambda_d$-tahy podle definice. Pro účely algoritmu není totiž třeba rozlišovat, zda zkoumaný tah není $\lambda_d$-tahem, nebo jím sice je, ale neúspěšným.

[4] Zatím ale ještě bez místa úspěšného útočného tahu, které k ní připojí volající procedura ATTACK.

vidlo: Nesetrvávat při vyhodnocování vrcholu stromu, ve kterém se počítá maximum, déle než do objevení první hodnoty 1 a podobně pro minimum a hodnotu 0. Je snadné si uvědomit, že se tímto pravidlem naše procedury řídí.

## 5.2 Implementace objektů hry

Prohledávací algoritmus popisovaný v minulé sekci používá při procházení stromu hry objekt *game*, s jehož pomocí generuje jednotlivé herní pozice a získává o nich potřebné informace. Při přípravě testovacího programu bylo třeba implementovat jednu verzi objektu *game* za každou ze tří námi zkoumaných her. Pokusíme se zde krátce nastínit, co taková implementace obnáší a jaké problémy je třeba vyřešit, má-li být rozumně efektivní.

Objekt *game* reprezentuje v jednom okamžiku vždy jednu pozici dané hry. Základní strukturou pro uložení pozice ve hře s kameny je pochopitelně pole, které pro každé místo na hrací ploše nese informaci o tom, zda je zrovna místo volné či obsazené kamenem jednoho nebo druhého hráče. Pro efektivní rozpoznávání koncových pozic a především pro rychlý výpočet zón relevance je ale většinou nutné přidružit k tomuto poli ještě další pomocné struktury, které tyto výpočty usnadní. Při návrhu těchto struktur je výhodné využít inkrementální charakter změn prováděných při manipulaci s objektem *game*. Pozice reprezentovaná tímto objektem se totiž během prohledávání mění pouze při volání metod doMove resp. undoMove, čili změna se vždy týká pouze jednoho místa hrací plochy. Navíc jsou volání těchto dvou metod „dobře uzávorkována“, a tak je možné uchovávat si v zásobníku změny provedené na struktuře při volání doMove a během undoMove tyto informace používat pro uvedení struktury do původního stavu.

Dříve než se pustíme do stručného popisu naší implementace zmiňovaných struktur v námi zkoumaných hrách – tedy v AtariGo, Hexu a Piškvorkách – je dlužno vyslovit jedno tvrzení, které již čtenář jistě dlouho očekával. Jmenovitě že všechny tři tyto hry lze popsat jako hry s kameny, ve kterých je navíc splněna vlastnost stability.[5] Jak jsme viděli v kapitole 4, je za předpokladu stability hry jediným obtížným krokem při konstrukci zón relevance pro základní predikáty hledání množin $PPV$, tedy množin tahů podstatně se podílejících na výhře, a proto se v následujícím výkladu zaměříme právě na ně.[6]

[5] První část tvrzení je zřejmá. Stabilita se pak ověří pro každou hru zvlášť rozborem vlastností koncových pozic.

[6] Připomeňme, že místo $x$ se v pozici $Pos$ podstatně podílí na výhře za $k$ tahů, pokud $(\exists X)[x \in X \,\&\, |X| = k \,\&\, Pos \xrightarrow{\pi/X} \pi(1) \,\&\, (\forall X' \subset X)\neg(Pos \xrightarrow{\pi/X'} \pi(1))]$.

### 5.2.1 AtariGo

Úkolem hráče ve hře AtariGo je obsadit svými kameny všechny svobody oponentova řetězce. Není proto těžké si uvědomit, že v této hře bude udržování informace o jednotlivých řetězcích a počtu jejich svobod hrát zásadní roli i při hledání míst podstatně se podílející na výhře. Poněkud formálněji můžeme říci, že místa podstatně se podílející na výhře hráče $\pi$ za nejvýše $k$ tahů se vyskytují pouze mezi svobodami řetězců jeho oponenta $o(\pi)$ s nejvýše $k$ svobodami.

Návrh struktury pro sledování řetězců a jejich svobod pro náš program jsme převzali z [1]. Jedná se o seznam záznamů o všech řetězcích dané pozice doplněný několika pomocnými poli a ukazateli tak, aby bylo možné tuto strukturu při změně pozice zahráním tahu inkrementálně měnit a s co nejmenšími nároky na čas aktualizace udržovat i počet svobod každého řetězce. Uvědomme si, že to není zcela jednoduchý úkol, neboť zahráním tahu se mohou dva řetězce (nebo i více) sloučit do jednoho, přičemž výsledný počet svobod sloučeného řetězce není prostým součtem příspěvků jeho jednotlivých částí, neboť některé své svobody mohly tyto části sdílet. Bližší informace o struktuře lze získat v citovaném zdroji. My zde již pouze poznamenáme, že v okamžiku, kdy je tato struktura k dispozici, spočívá výpočet množin $PPV$ již pouze v procházení seznamu řetězců a sjednocování příspěvků v podobě svobod některých z nich (podle kritéria popsaného na konci minulého odstavce).

### 5.2.2 Hex

Ve hře Hex, lépe řečeno v námi zvolené modifikaci jménem Y, se hráči snaží propojit pomocí svých kamenů tři strany hexagonové sítě ve tvaru trojúhelníku. Jelikož je zde, podobně jako v AtariGo, motiv propojování velmi význačný, rozhodli jsme se využít při implementaci Hexu podobné struktury pro správu řetězců (tedy navzájem propojených kamenů téže barvy) jako ve hře předchozí, pouze s malými změnami. Místo počtu svobod každého řetězce náš totiž v Hexu zajímá, zda-li je daný řetězec spojen s některou ze tří stran hrací plochy. Naštěstí se jedná o informaci, kterou je možno během inkrementálních aktualizací struktury snadno přepočítávat.

Přechod od struktury k algoritmu pro výpočet míst podstatně se podílejících na výhře není v Hexu tak snadný jako u AtariGo. Úkolem je totiž hledat minimální množiny míst, na které když hráč umístí své kameny, dosáhne propojení všech tří stran hrací plochy. Není obtížné si uvědomit, že jde vlastně o instanci optimalizačního problému, kterému se říká hledání

Steinerova stromu[7] a jehož rozhodovací verze je NP-úplná ([12]). Ve světle této skutečnosti jsme se u Hexu při výpočtu $PPV$ spokojili s přístupem založeným na zkoumání jednotlivých množin „hrubou silou“. Aby ale doba na výpočet jedné odpovědi nepřesáhla rozumnou mez, byly tímto způsobem hledány pouze množiny velikosti 1 a 2. Pak bylo kvůli zachování korektnosti nutné všechna ostatní místa (tedy ta, která se nepodílí na výhře za jeden či dva tahy) zahrnout do třetí oblasti vracené posloupnosti.[8]

### 5.2.3 Piškvorky

Třetí zkoumanou hrou je hra Piškvorky. Pravidla v ní stanoví, že výhry dosáhne ten hráč, kterému se dříve podaří umístit na hrací plochu pět svých kamenů v řadě a to buď vodorovně, svisle nebo diagonálně. Uvážíme-li množinu všech možných pětic míst, na kterých lze umístěním kamenů výhry dosáhnout, jsme již jen krůček od popisu inkrementální struktury, která byla použita v této hře. Za každou takovou pětici si totiž stačí pamatovat, kolika je obsazena kameny a které barvy, a pak po zahrání tahu vždy aktualizovat záznamy těch pětic, které jsou s místem tahu incidentní. Při výpočtu $PPV$ je pak třeba projít záznamy pětic, které jsou obsazené „mnoha“ kameny příslušného hráče, a každé volné místo takové pětice přidat do množiny. Platí totiž, že místa podstatně se podílející na výhře hráče $\pi$ za nejvýše $k$ tahů se vyskytují pouze mezi volnými místy pětic obsazených pouze hráčem $\pi$ a to alespoň $5 - k$ kameny. Tyto úvahy vedly k implementaci poměrně efektivní struktury pro objekt *game* ve hře Piškvorky.

## 5.3 Další použité techniky urychlení

V sekci 5.1 byla popsána základní kostra prohledávacího algoritmu spolu s vysvětlením, jak se při prohledávání uplatňují zóny relevance. Nyní je na místě se v krátkosti zmínit o dalších urychlovacích technikách, které byly použity v našem programu. Jelikož se jedná o klasické techniky užívané při prohledávání herních stromů, jejichž základní idea fungování by měla být všeobecně známá, zaměříme výklad především na popis specifických vlastností naší implementace.

[7] Vstupem problému je vážený graf a skupina jeho vrcholů nazvaných terminály, úkolem pak najít minimální (ve smyslu váhy) souvislý podgraf, který všechny terminály obsahuje.

[8] O všech místech, o kterých se nepodaří dokázat, že se podílí na výhře za nejvýše dva tahy, tedy program konzervativně předpokládá, že by se mohla podílet na výhře za tahy tři. Tím není ohrožena korektnost výsledku (viz první bod lemmatu 9 kapitoly 4), ale omezí se účinnost konstruovaných zón, které pak mohou být zbytečně veliké.

### 5.3.1 Transpoziční tabulky

Transpoziční tabulky šetří práci prohledávacího algoritmu v případě výskytu tzv. *transpozic*, neboli v situacích, kdy je určitá herní pozice dosažitelná z počáteční pozice stromu hry[9] více než jedním způsobem. Přítomnost transpozic se projeví, když algoritmus postupně uvažuje o dvou nebo více tazích zahraných v různém pořadí. Potom je výhodné uložit po prvním vyhodnocení dané pozice spočtené výsledky do tzv. *transpoziční tabulky* a v okamžiku příštího výskytu pozice – místo opakovaného prohledávání příslušného podstromu – uložené hodnoty rovnou použít. Transpoziční tabulky bývají implementovány jako hašovací tabulky určité předem dané velikosti, čímž se zajistí konstantní doba pro vyhledávání. Nutné je pak zvolit přijatelný způsob ošetření kolizí, jejichž výskytu se není možno obecně vyhnout.

Hašovací technika pro ukládání pozic implementovaná v našem programu pochází z [21] a její použití je velmi jednoduché. Před zahájením prohledávání se pro každé místo na hrací ploše vygeneruje náhodný kód za každý stav, ve kterém se místo může nacházet (tj. prázdné, obsazené černým, obsazené bílým). Celkový kód pozice je pak vždy spočten z kódů jednotlivých míst podle jejich současného stavu v pozici pomocí operace XOR. Výhodou této techniky jsou přijatelná odolnost vůči kolizím, snadná implementace a především možnost rychlého inkrementálního udržování kódu pozice při jejích lokálních změnách.

Vzhledem k tomu, že při lambda prohledávání je často pozice vyhodnocována postupně z pohledu obou hráčů, připojuje program ke kódu pozice informaci o tom, který hráč je právě na tahu. Do transpoziční tabulky pak program ukládá nejen výslednou hodnotu pozice (pro danou úroveň lambda a hloubku prohledávání), ale i spočtenou zónu relevance, která je potřebná pro další fáze prohledávání. Součástí záznamu je konečně také kompletní popis pozice, který je narozdíl od kódu používaného pro adresaci do tabulky pro každou pozici jiný, a jehož použitím je zajištěna stoprocentní spolehlivost spočtených výsledků.

V případě výskytu kolize při hašování, tedy pokud jsou dvě různé pozice namapovány pomocí svých kódů na stejné místo v tabulce, je třeba rozhodnout, který ze dvou záznamů na místě ponechat a kterého se zbavit.[10] Pravidlu, pomocí nejž takové rozhodnutí učiníme, se anglicky říká *replacement scheme* (viz [6]). V našem programu bylo použito pravidlo, které je

[9] Označení „strom hry“ je tedy vlastně nesprávné, neboť v přítomnosti transpozic se obecně jedná o orientovaný graf bez cyklů.

[10] Typicky totiž nepřipouštíme, aby jednomu místu příslušel proměnný počet záznamů, jak tomu někdy bývá u jiných aplikací hašování, neboť takové řešení není v případě transpozičních tabulek praktické.

v citovaném článku nazváno TwoBig. Jedná se o přístup, při kterém jsou na jednom místě v tabulce uchovávány vždy dva záznamy a v případě vyhledávání potenciálně oba testovány, zda neobsahují data zkoumané pozice. Při ukládání je potom vždy jeden ze starých záznamů nahrazen záznamem nově spočtené pozice, a to konkrétně ten, jehož výpočet byl méně náročný (odpovídá stromu s menším počtem vrcholů). Tím, že pravidlo vždy zachovává výsledky od naposledy zkoumané pozice, u níž je velká šance, že bude brzy zkoumána znovu, a zároveň výsledky od pozice, jejíž ohodnocení bylo náročné, a jejíž záznam má proto šanci později ušetřit mnoho práce, dosahuje s jeho použitím technika transpozičních tabulek nejlepších možných výsledků, (což bylo experimentálně potvrzeno právě v [6]).

### 5.3.2 History heuristika

Je známým faktem, že pořadí, ve kterém jsou při prohledávání stromu hry metodou alfa-beta zkoušeny jednotlivé přípustné tahy, může značně ovlivnit velikost prohledávaného stromu a tím i dobu výpočtu. Podaří-li se totiž co nejdříve vyhodnotit podstrom, který dokazuje, že hodnota vrácená právě zkoumaným vrcholem bude nutně ležet mimo „prohledávací okno“ stanovené parametry $\alpha$ a $\beta$, zbylými kandidáty již není třeba se zabývat a jsou takzvaně ořízнuty (anglicky se jim říká *cutoff*).

Pro určovaní pořadí, ve kterém budou jednotlivé tahy zkoušeny při prohledávání stromu hry, byla v našem programu použita tzv. *history heuristika* ([15]). Jedná se o metodu, která je zcela nezávislá na zkoumané hře a využívá pro uspořádávání pouze informace získané v předchozích fázích prohledávání. Za každý možný tah (tedy za každé místo na hrací ploše) je uchovávána v tabulce informace o jeho dosavadní úspěšnosti a v nově zkoumaném vrcholu jsou pak tahy zkoušeny v pořadí určeném právě touto hodnotou. Úspěšnost tahu se přitom měří tím, kolikrát se s jeho pomocí v dosavadní historii prohledávání podařilo způsobit cutoff (za každou takovou událost je k hodnotě tahu přičítána jistá váha). Heuristika vlastně využívá pozorování, že mnoho zkoumaných pozic je velmi podobných, a tak je u každého tahu, který byl úspěšný dříve, velká šance, že se vyplatí ho přednostně vyzkoušet i v nové pozici.

Pro výpočet váhy, která je v případě úspěchu tahu přidělena, používá náš program předpis $2^{depth}$, kde *depth* je hloubka prohledávaného podstromu. Za důležitější jsou tak považovány ty úspěchy, které se odehrají blíže ke kořeni celkového stromu. Na závěr podotkněme, že tabulka s hodnotami tahů v našem programu nerozlišuje, který hráč zrovna tah provádí. Spoléháme se při tom na empiricky ověřené tvrzení, že tah, který je důležitý pro mého oponenta, muže být důležitý i pro mne.

## 5.4 Dosažené výsledky

Nyní stručně představíme výsledky, kterých se s pomocí vzniklého programu podařilo dosáhnout. Předem je nutno upozornit na skutečnost, že vyřešit některou z testovaných her nebylo cílem této práce. Výsledky zde proto uvádíme především pro ilustraci úspěšnosti zkoumané metody, uvedená čísla pak jako referenční body pro její případné další zdokonalování. Poté, co předvedeme, jak si program vedl při řešení malých verzí námi zkoumaných her, se pokusíme zhodnotit obecné vlastnosti metody, které se během testovaní podařilo identifikovat.

### 5.4.1 Řešení malých verzí zkoumaných her

Pro každou ze tří her, kterými jsme se zabývali, lze přirozeným způsobem zavést „menší" verze, a tak bylo možno se s pomocí našeho programu pokusit o jejich řešení, i když u původních her tak, jak jsou tradičně popisovány, by takový úkol v rozumném čase nebyl zvládnutelný. Změna spočívala především ve zmenšení velikosti hrací plochy, v případě hry Piškvorky též v redukci počtu kamenů v řadě, které je potřeba umístit pro dosažení vítězství, a to z pěti na čtyři.

Pro testování byl použit algoritmus dual-lambda search, který vykazoval ve všech třech hrách lepší výsledky, než jednodušší lambda search.[11] Testovali jsme na počítači s procesorem Celeron o frekvenci 1.0 GHz. Výsledky[12] shrnuje tabulka 5.1.

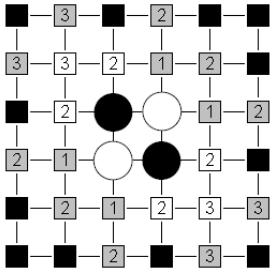
| Malá hra | $\lambda$-úroveň | hloubka | čas(s) | #vrcholů |
|---|---|---|---|---|
| AtariGo $6 \times 6$ s kříž. střihem | 3 | 15 | 5 | 370545 |
| Hex $4 \times 4$ | 3 | 9 | 3 | 24552 |
| Piškvorky $7 \times 7$ na 4 vítězné | 3 | 11 | 13 | 903414 |

Tabulka 5.1: Výsledky testu řešení malých verzí zkoumaných her.

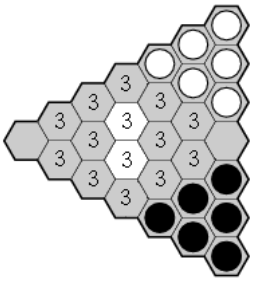
Pro vyřešení každé z popsaných úloh bylo třeba využít prohledávání až do $\lambda$-úrovně 3. Porovnáním hodnoty času a počtu zkoumaných vrcholů stromu hry si lze všimnout několikanásobně pomalejšího vyhodnocování pozic v případě hry Hex, o kterém jsme již mluvili v sekci 5.2.2. Výstupy programu pro jednotlivé úlohy jsou znázorněny na obrázcích 5.1–5.3. Vidíme, že program během svého výpočtu nenašel pouze jediný vítězný tah, který by stačil

[11] Tento jev byl velmi výrazný u hry AtariGo a u Piškvorek, zatímco u Hexu spíše zanedbatelný. Ukazuje nám, jaký význam mají v jednotlivých hrách inverze.

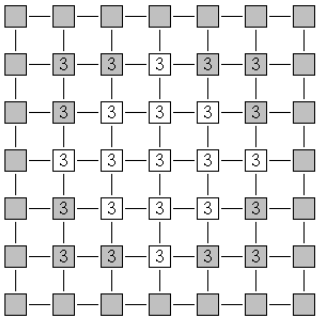
[12] Uvádíme čísla získaná bez použití transpoziční tabulky.



Obrázek 5.1: Rozbor pozice malé verze hry AtariGo při jejím řešení.



Obrázek 5.2: Rozbor pozice malé verze hry Hex při jejím řešení.



Obrázek 5.3: Rozbor pozice malé verze hry Piškvorky při jejím řešení.

pro vyřešení hry, spíše by se dalo říci, že jeho výstupem je podrobný rozbor počáteční pozice obsahující i další informace.

Interpretace obrázků je následující: V každé ze tří pozic je na tahu černý a může zvítězit tahem na libovolné z bíle označených míst. Šedou barvou jsou označena místa, kterými se program také zabýval, ale výhru se po příslušeném tahu dokázat nepodařilo. Černé značky pak indikují, že hraním na takové místo první hráč ohrozí sám sebe a partii prohraje. Co se týče čísel, jež doplňují barevné značky, ta určují $\lambda$-úroveň příslušného útočného tahu. U tahů bez čísla se programu za daných výpočetních omezení nepodařilo existenci hrozby prokázat.

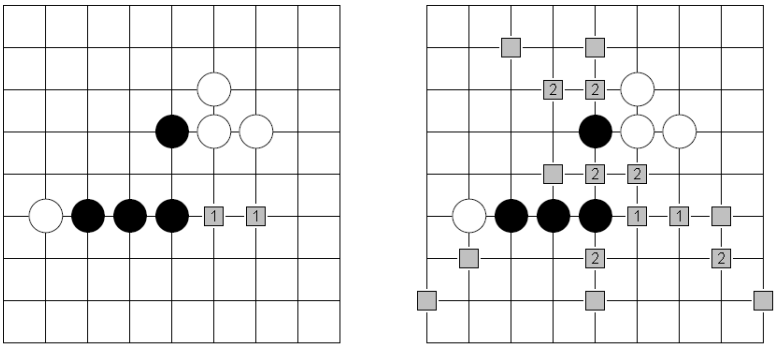
Srovnáme-li naše výsledky s jejich obdobou prezentovanou v odborné literatuře (viz kapitola 2), můžeme provést toto zhodnocení: Výsledky získané při řešení hry AtariGo na hrací ploše $6 \times 6$ s křížovým střihem uprostřed jsou přinejmenším srovnatelné s těmi z [8], kde autor uvádí, že se mu podařilo stejnou pozici vyřešit na počítači s procesorem Athlon 1.7 GHz za 21 sekund. U her Hex a Piškvorky je situace méně příznivá. Pomocí technik lépe využívajících specifických vlastností každé z her se již podařilo vyřešit Hex pro velikost hrací plochy $7 \times 7$ ([11]) a Piškvorky na hrací ploše $15 \times 15$ při hře na 5 vítězných kamenů v řadě ([3]), což jsou výsledky s naším programem bez dalších vylepšení nedosažitelné.

### 5.4.2 Obecné postřehy

Při testování našeho programu jsme si uvědomili některé rysy zkoumané metody, na které je dobré zvlášť upozornit. Prvním z nich je fakt, že efektivita použití zón relevance postupně klesá se vzrůstající úrovní lambda. Na obrázku 5.4 si můžeme prohlédnout jednu pozici ze hry Piškvorky analyzovanou nejprve na $\lambda$-úrovni 1, a pak 2. Je vidět, že zatímco v prvním případě se množina zkoumaných míst, jak ji určuje zóna relevance, přesně shoduje s množinou $\lambda_a^1$-tahů dané pozice[13], v případě druhém jsou prohledávána i místa, která $\lambda_a^2$-tahy nejsou (viz šedě označená místa bez čísla). Kdybychom se později pokoušeli hledat $\lambda_a$-tahy úrovně 4, musela by již být testována vůbec všechna volná místa dané pozice.

Nelze tvrdit, že by právě popsaná vlastnost byla nedostatkem navržené metody. Dává však nahlédnout, že se vzrůstající $\lambda$-úrovní též nutně stoupá i složitost zkoumaného problému a zóny relevance, jakožto horní aproximace míst, která na statut problému mohou mít vliv, musí tento fakt svým nárůstem reflektovat.

[13] Lze dokázat, že to není náhoda a že rovnost $\Lambda_a^1(Pos, \pi) = PPV(Pos, \pi, 2)$ platí v každé pozici $Pos$, ve které hráč $\pi$ nemůže vyhrát jedním tahem.




Obrázek 5.4: Pozice v Piškvorkách analyzovaná na $\lambda$-úrovni 1 a 2.

Druhá vlastnost, na kterou zde upozorníme, souvisí více se zvoleným způsobem prohledávání než se zónami relevance. Jakkoliv přínosné pro rychlost výpočtu se může zdát uspořádání tahů určené history heuristikou, někdy se při testování stávalo, že byl pro vyzkoušení vybrán nejprve tah, který například neodvracel hrozbu tím úplně nejjednodušším způsobem. Pokud rozhodnutí o hodnotě pozice vzniklé zahráním takového tahu bylo obtížné (např. proto, že její hodnota „ležela blízko remízy“), strávil program mnoho času zkoumáním velkého podstromu, i když bylo k dispozici i jednoduché odvrácení hrozby.

Nezbývá než konstatovat, že pro vyřešení popsaného problému, a tím pro celkové zrychlení programu, by mohlo být užitečné zvolit jiný způsob uspořádávání zkoušených tahů. Jako alternativa se též nabízí myšlenka volby jiného způsobu procházení stromu hry, který by nebyl na správném pořadí tahů tolik závislý – např. nějaké varianty prohledávání typu best-first.

### 5.4.3 Shrnutí

Program, který vznikl v rámci testování metod zkoumaných v této práci, by se dal označit jako taktický analyzátor pozic. Jeho síla spočívá především v analýze koncovek nebo lépe řečeno pozic, které jsou bohaté na hrozby jednoho či obou hráčů. Při dalším vylepšování programu, např. za účelem vyřešení větších verzí zkoumaných her, by mohlo být přínosné vyzkoušet jiný způsob prohledávání $\lambda$-stromů, než je prohledávání do hloubky, např. metodu konstrukce zón relevance pro lambda search kombinovat s algoritmem proof-number search ([2]).

# Kapitola 6

# Závěr

V této práci jsme se zabývali metodami neuniformního prohledávání herních stromů založenými na hrozbách. Popsali jsme algoritmy lambda search a dual-lambda search, které hrozby využívají při směřování prohledávání k rychlému nalezení cíle. Dále jsme ukázali, jak lze pro urychlování popsaných algoritmů konstruovat tzv. zóny relevance – množiny míst na hrací ploše, které jedině mohou ovlivnit výsledek prohledávání. Tyto zóny jsme definovali v teoretickém rámci tzv. her s kameny, speciální třídě her, ve kterých je možno s pojmy místa či zóny snadno pracovat. Na závěr jsme předvedli, jak lze pomocí strategie prohledávání do hloubky studované algoritmy naimplementovat, a ukázali některé výsledky programu, který vznikl při testování popsaných metod.

Je zřejmé, že úspěšnost či použitelnost námi zkoumaných metod se bude u různých her lišit. Především je potřeba se omezit na hry, ve kterých se vyskytují hrozby. Pro ně bývá charakteristická přítomnost prvku „náhlé smrti“, což znamená, že jeden z hráčů by dokázal vždy během několika málo tahů zvítězit, kdyby se jeho oponent neustále vzdával svého tahu neboli pasoval.[1] Přesto i u her, které charakter náhlé smrti postrádají, se dá o možnosti aplikovat popsané metody stále uvažovat, např. při analýze koncovek či určitých taktických podcílů.

Teoretickou překážkou použití metody lambda search může býti u her zugzwang. Viděli jsme, že vlastnost hry nepřipouštět zugzwang je jedním z předpokladů pro úplnost této techniky. Na druhou stranu experimenty s hrou AtariGo ukázaly, že i ve hře, ve které se zugzwang v některých pozicích vyskytuje, může lambda search dosahovat dobrých výsledků. Zdá se, že míra vlivu zugzwangu na úspěšnost metody není zcela jasná a měla by být podrobena dalšímu výzkumu.

[1] Zdůrazněme, že není podstatné, zda je pas pravidly hry přímo povolen, spíše se jedná o hypotetickou možnost jednoho z hráčů zahrát více tahů za sebou.

Za prozkoumání zcela jistě také stojí možnost definovat zóny relevance i v jiných hrách než v takových, které lze popsat jako hry s kameny. Např. v Šachu, kde figury mění svou pozici na hrací desce a možnost provést určitý tah je složitě podmiňována přítomností ostatních figur, není přímá aplikace zde popsané konstrukce zón relevance možná. Naznačené obtíže by mohly být překonány zavedením složitějšího rámce, ve kterém by místo jednoduchého popisu pozice pomocí míst, která jsou buď volná nebo obsazená, musela ke slovu přijít obecnější teorie chápající jednotlivé pozice jako stavy přetvářené pomocí tahů – operátorů.

# Literatura


[1] GNU Go online dokumentace. `http://www.gnu.org/software/gnugo/gnugo_15.html#SEC180`.

[2] Allis L. V., van der Meulen M., van den Herik H. J. (1994): Proof number searcs. *Artificial Intelligence*, **66**(1):91–124.

[3] Allis L. V., van den Herik H. J., Huntjens M.P.H. (1996): Go-moku solved by new search techniques. *Computiational Intelligence*, **12**:7–23.

[4] Anshelevich V. V. (2000): The game of hex: An automatic theorem proving approach to game programming. In *Proc. 17th National Conf. on AI (AAAI-2000)*, pages 189–194.

[5] Bouzy B., Cazenave T. (2001): Computer go: An AI oriented survey. *Artificial Intelligence*, **132**(1):39–103.

[6] Breuker D., Uiterwijk J., van den Herik H. J. (1994): Replacement schemes for transposition tables. *ICCA Journal*, **17**(4):183–193.

[7] Cazenave T. (2000): Abstract proof search. *Lecture Notes in Computer Science*, **2063**:39–54.

[8] Cazenave T. (2002): Gradual abstract proof search. *ICGA*, **25**(1):3–16.

[9] Gale D. (1979): The game of hex and brouwer fixed-point theorem. *The American Mathematical Monthly*, **86**:818–827.

[10] Hayward R., Bjornsson Y., Johanson M., Po N., van Rijswijck J. (2005): Solving 7x7 hex with domination, fill-in, and virtual connections. *Theoretical Computer Science*, **349**:123–139.

[11] Hayward R., van Rijswijck J. (2006): Hex and combinatorics. *Discrete Math*, **306**:2515–2528.

[12] Karp R. M. (1972): Reducibility among combinatorial problems. *Complexity of Computer Computations*, pages 85–103. Plenum Press.

[13] Von Neumann J., Morgenstern O. (1944): *Theory of Games and Economic Behavior*. Princeton University Press.

[14] Samuel A. L. (1959): Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, **3**(3):210–229.

[15] Schaeffer J. (1989): The history heuristic and alpha-beta search enhancements in practice. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-**11**(11):1203–1212.

[16] Shannon C. E. (1950): Programming a computer for playing chess. *Philosophical Magazine*, **41**(4):256–275.

[17] Soeda S., Kaneko T., Tanaka T. (2005): Dual lambda search and its application to shogi endgames. In *Proceedings of Advances in Computer Games 11 (ACG11)*.

[18] Thomsen T. (2000): Lambda-search in game trees — with application to go. *ICGA Journal*, **23**(4):203–217.

[19] van der Werf E., Uiterwijk J., van den Herik H. J. (2002): Solving ponnuki-go on small boards.

[20] Yoshizoe K., Kishimoto A., Müller M. (2007): Lambda depth-first proof number search and its application to go. In *20th International Joint Conference on Artificial Intelligence (IJCAI-07)*, pages 2404–2409.

[21] Zobrist A. L. (1970): A new hashing method with application for game playing. Technical report #88, Computer Science Department, The University of Wisconsin, Madison WI, USA.