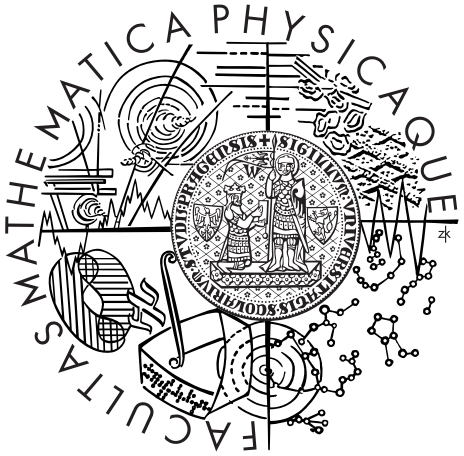
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Martin Koutecký

Optimalizace na grafech s omezenou stromovou šířkou přes vlastnosti vyjádřitelné v MSOL

Katedra aplikované matematiky

Vedoucí bakalářské práce: doc. Mgr. Petr Kolman Ph.D.

Studijní program: Informatika

Studijní obor: Obecná informatika

Praha 2011

Nejprve děkuji vedoucímu práce doc. Mgr. Petru Kolmanovi Ph.D. za cenné rady, doporučení a připomínky k práci a především všechen čas, který takto obětoval.

Dále děkuji doc. RNDr. Jiřímu Fialovi Ph.D., jehož přednášky mě přivedly k zájmu o tematiku práce a položily potřebné základy.

V neposlední řadě patří obrovský dík mým rodičům, kteří mi poskytují potřebné zázemí a bez jejichž podpory bych se nemohl studiu plně věnovat.

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

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

V Praze dne 27. 5. 2011 Martin Koutecký

Název práce: Optimalizace na grafech s omezenou stromovou šířkou přes vlastnosti vyjádřitelné v MSOL

Autor: Martin Koutecký

Katedra: Katedra aplikované matematiky

Vedoucí bakalářské práce: doc. Mgr. Petr Kolman Ph.D.

Abstrakt: Courcellova věta mluví o výpočetní složitosti rozhodovacích problémů definovaných formulemi monadické logiky druhého řádu nad relačními strukturami s omezenou stromovou šířkou. Pro pevnou stromovou šířku a vstupní formuli dává Courcellova věta algoritmus, který formuli rozhodne v lineárním čase nad strukturou dané stromové šířky. Práce podává samostatný důkaz Courcellovy věty pomocí metod teorie konečných modelů. Dále obsahuje důkazy všech potřebných prerekvizit hlavního důkazu, zejména v teorii konečných modelů široce využívané Ehrenfeuchtovy-Fraïssého věty. Práce též obsahuje implementaci algoritmu plynoucího z tohoto důkazu. Nakonec nastiňuje aktuální stav výzkumu dané oblasti a z něj plynoucí možnosti.

Klíčová slova: Courcellova věta, stromová šířka, monadická logika druhého řádu, logické hry

Title: Optimization in graphs with bounded treewidth through properties expressible in MSOL

Author: Martin Koutecký

Department: Department of Applied Mathematics

Supervisor: doc. Mgr. Petr Kolman, Ph.D.

Abstract: Courcelle's theorem speaks about computational complexity of decision problems defined by formulae in monadic second order logic over relational structures with bounded treewodth. For a fixed treewidth and a fixed formula, Courcelle's theorem gives an algorithm, which decides the formula over a structure of said treewidth in linear time. This thesis provides a self-contained proof of Courcelle's theorem using methods of finite model theory. Furthermore it contains the proofs of all propositions and theorems upon which the main proof depends, notably the Ehrenfeucht-Fraïssé theorem widely used in finite model theory. The thesis also contains an implementation of an algorithm which follows from the main proof. Finally a sketch of the current state of the art of the area of research is given, as well as the possibilities following from it.

Keywords: Courcelle's theorem, treewidth, monadic second order logic, logic games

Obsah

1 Úvod **6**

2 Prerekvizity **8**
2.1 Konečné relační struktury 8
2.2 Stromová šířka 9
2.2.1 Stromový rozklad a stromová šířka 9
2.2.2 Hezký rozklad a normalizovaný rozklad 11
2.2.3 Indukované podstruktury 12
2.3 Logické formule 13
2.4 Logické hry 16
2.4.1 Hintikka hry 17
2.4.2 k-typy a elementární ekvivalence 18
2.4.3 Ehrenfeucht-Fraïssé hry 19
2.4.4 Ehrenfeuchtova-Fraïssého věta 21
2.4.5 Aplikace Ehrenfeuchtovy-Fraïssého věty 22

3 Courcellova věta **24**
3.1 Kompoziční lemma 24
3.2 Courcellova věta a algoritmus 26
3.3 Rozšíření a dodatky 30
3.3.1 Optimalizační varianty a kvantifikace v MSO_k 30
3.3.2 Speciální Ehrenfeucht-Fraïssé hry 30
3.3.3 Zobecněné kvantifikátory 32

4 Kam dál **33**
4.1 Datalog a logické programování 33
4.2 Rozšířené Hintikka hry 34
4.3 Jiné typy rozkladů 35

5 Implementace **36**
5.1 Struktura knihovny `pycourcelle` 36
5.2 Instalace a použití 37
5.3 Výkon a omezení 39
5.4 Možné optimalizace 40

6 Závěr **42**

1 Úvod

Už v 70. letech minulého století si někteří matematikové všimli, že mnoho těžkých grafových kombinatorických problémů lze efektivně řešit dynamickým programováním na grafech s omezenou *dimenzí* [BB73]. Mnohem vhodnějším parametrem se později ukázala být *stromová šířka*, zavedená Robertsonem a Seymourem při práci na *Graph Minor Project* [RS84, RS86]. Algoritmy pak mají podobu dynamického programování na *stromovém rozkladu* vstupního grafu. Dalším postupem času se objevily jiné možné parametry a vznikl celý obor nazývaný *parametrizovaná složitost* [DF99], často též spojovaný se zkratkou FPT[1].

Do stejného období lze vysledovat kořeny jiného zajímavého a dnes aktivního oboru, *deskriptivní složitosti*. Ta se zabývá vztahem různých logických jazyků a tříd výpočetní složitosti. V roce 1974 Fagin dokázal, že problémy třídy NP přesně odpovídají sentencím existenční logiky druhého řádu ($\exists$SO[2] – obsahuje formule začínající existenčními kvantifikátory přes proměnné druhého řádu následovanými formulí prvního řádu) [Fag74], v roce 1982 pak ukázal Immerman, že problémy třídy P odpovídají sentencím logiky prvního řádu s přidaným operátorem nejmenšího pevného bodu (FO(LFP)[3] – LFP operátor intuitivně odpovídá induktivní definici relace a lze s ním například popsat tranzitivní uzávěr grafu) [Imm82]. Mnoho dalších výsledků lze najít v Immermanově rozsáhlé knize, která se stala základem tohoto oboru [Imm99].

Na jistém pomezí zmíněných dvou oborů se nachází tzv. *Courcellova věta*. V roce 1990 dokázal Courcelle, že každou grafovou vlastnost popsatelnou v monadické logice druhého řádu (MSO[4] – proměnné druhého řádu jsou pouze množiny prvků univerza, nikoliv relací vyšších arit) lze na relačních strukturách s omezenou stromovou šířkou rozhodnout v čase lineárním vzhledem k velikosti dané struktury [Cou90]. Je důležité zdůraznit, že závislost složitosti na velikosti formule popisující danou vlastnost, a na stromové šířce struktury, je obecně velmi rychle rostoucí funkce (mocninná věž), přesnými odhady složitostí se zabývají Frick a Grohe [FG04]. Krátce po zveřejnění Courcellova důkazu byla věta rozšířena na optimalizační varianty problémů (EMSOL[5]) [ALS91].

Courcellovu větu lze dokázat několika způsoby. Obvyklým postupem bývá konstrukce tzv. stromového automatu, který rozhoduje, zda je struktura $\mathcal{A}$ modelem formule φ, ať už je konstrukce explicitní [ALS91], nebo implicitní [Cou90] pomocí kompozičních vět (v duchu Fefermanovy-Vaughtovy věty, viz Makowského

[1] *Fixed Parameter Tractability*

[2] *Existential Second Order*

[3] *First Order (Least Fixed Point)*

[4] *Monadic Second Order*

[5] *Extended Monadic Second Order*

přehledový článek [Mak04]). Čas potřebný pro konstrukci automatu a jeho velikost závisí jen na stromové šířce struktury a velikosti φ. Tento postup s sebou v praxi nese značné nevýhody, protože velikost automatu je obrovská, a tak algoritmus typicky pro nedostatek paměti selže dříve, než se vůbec dostane ke vstupnímu grafu. To vedlo ke hledání jiných cest, o čemž svědčí řada publikací, např. [Fri01, GPW10, KLR11].

Práce čtenáře nejprve v Kapitole 2 podrobněji uvede do problematiky. Hlavním bodem je samostatný důkaz Courcellovy věty založený na konceptu logických her, kterým se zabývá Kapitola 3. Pak práce nabízí v Kapitole 4 pohled na současný výzkum možností praktického využití této věty a poskytuje množství ukazatelů na směry, kterými je možné se v této oblasti vydat. Myšlenky obsaženého důkazu byly uvedeny do praxe v demonstračním programu, implementovaném v jazyce Python a blíže popsaném v Kapitole 5.

Courcellova věta je široce známým výsledkem v oblasti teorie grafů a algoritmů. Přesto se její ucelený důkaz v učebnicích prakticky nevyskytuje. Tato práce takový důkaz poskytuje – seznámí čtenáře se všemi potřebnými prerekvizitami a větu dokáže bez nutnosti odkazovat se na jiné (zde nedokázané) výsledky. To je také jejím hlavním přínosem.

Na závěr by bylo vhodné podotknout, že pohled práce vychází především ze článku od Gottloba et al. [GPW10], který na první pohled sliboval zajímavý a praktický přístup vhodný pro implementaci. Jak se ale ukázalo později, hlavní přínos tohoto článku pro praxi netkví v důkazu obecné Courcellovy věty, nýbrž v novém a zajímavém způsobu tvorby specifických algoritmů. Z toho také plyne spíše demonstrační hodnota vytvořeného programu. Ten sice implementuje algoritmus popsaný v důkazu Courcellovy věty, ale neklade si za cíl být v tom obzvlášť efektivní. Hlavní možnosti vylepšení vidíme v oblasti teorie, spíše než samotné implementace. K přístupu zmíněného článku se ještě vrátíme v Kapitole 4.

2 Prerekvizity

Než přistoupíme k samotnému důkazu Courcellovy věty, potřebujeme zavést základní pojmy, učinit několik pozorování a dokázat jednu větu, na níž náš důkaz Courcellovy věty stojí.

2.1 Konečné relační struktury

Definice 1. Konečnou množinu symbolů $\tau = \{c_1, \ldots, c_k, R_1, \ldots, R_K\}$ budeme nazývat *abecedou*. Symboly $c_1, \ldots, c_k$ nazýváme *konstanty*, $R_1, \ldots, R_K$ nazýváme *relace*.

Často pro přehlednost dělíme abecedu na část obsahující konstanty $\tau_c = c_1, \ldots, c_k$ a část obsahující relace $\tau_R = R_1, \ldots, R_K$. Pak $\tau = \tau_c \cup \tau_R$.

Definice 2. *Konečná relační struktura* $\mathcal{A}$ nad τ je dána konečnou *doménou* $A = dom(\mathcal{A})$ a *realizací* konstant a relací. Realizací konstant myslíme přiřazení prvků domény konstantám abecedy τ, tedy pro každé i existuje nějaké $a \in A$ takové, že $c_i^{\mathcal{A}} = a$. Realizací relací myslíme přiřazení množin $R_i^{\mathcal{A}} \subseteq A^{\alpha_i}$ relacím τ, kde α_i označuje arity $R_i \in \tau$. Používáme značení $\mathcal{A} = (A, c_1^{\mathcal{A}}, \ldots, c_k^{\mathcal{A}}, R_1^{\mathcal{A}}, \ldots, R_n^{\mathcal{A}})$. Protože se nebudeme zabývat jinými než konečnými relačními strukturami, budeme dále říkat jednoduše *struktura*.

Přidání konstant a relací do struktury je potřeba reflektovat i změnou abecedy τ. To obvykle neděláme explicitně, nýbrž jen značením $(\mathcal{A}, a_1, \ldots, a_l, A_1, \ldots A_L)$, které znamená přidání nových konstant a relací do abecedy a určení jejich realizací $c_i^{\mathcal{A}} = a_i$ a $R_i^{\mathcal{A}} = A_i$. Pojmem *konstanty struktury* myslíme realizace konstant abecedy struktury, obdobně *relace struktury* jsou realizace relací abecedy dané struktury.

Definice 3. *Velikostí struktury* $\mathcal{A}$ myslíme součet velikostí její domény $|dom(\mathcal{A})|$, realizací jejích konstant $|c_i^{\mathcal{A}}| = k$ a realizací jejích relací $|R_i^{\mathcal{A}}|$. Tedy $|\mathcal{A}| = |dom(\mathcal{A})| + k + \sum_{R_i \in \tau} |R_i^{\mathcal{A}}|$.

Příklad 4. Již od začátku je zřejmé, že nejčastěji zkoumanou strukturou budou na následujících stránkách grafy. Graf $\mathcal{G}$ je (popsáno slovníkem definic výše) struktura nad abecedou $\tau = \{E\}$, kde E je binární relace, daná doménou $dom(\mathcal{G}) = V$ a realizací relace E, kterou značíme $E^{\mathcal{G}}$. Velikost $\mathcal{G}$ je součet počtu jeho vrcholů a hran, $|\mathcal{G}| = |V| + |E^{\mathcal{G}}|$. Budeme-li chtít například rozhodnout problém dosažitelnosti mezi dvěma konkrétními vrcholy s a t, musíme je v grafu vyznačit, tedy zavést je jako konstanty. Mluvíme pak o struktuře $\mathcal{G}$ s konstantami s a t a píšeme $(\mathcal{G}, s, t)$. Kdybychom chtěli testovat správnost nějakého obarvení grafu třemi barvami, zavedeme barvy do grafu jako trojici množin R, G a B a píšeme $(\mathcal{G}, R, G, B)$.

2.2 Stromová šířka

2.2.1 Stromový rozklad a stromová šířka

Definice 5. *Stromový rozklad* $\mathcal{T}$ τ-struktury $\mathcal{A}$ je taková dvojice $(T, (A_t)_{t \in T})$, kde T je strom (jehož vrcholy $t \in T$ nazýváme *uzly*) a každá z A_t, kterým říkáme *kapsy*, je podmnožina A s následujícími vlastnostmi:

1) Každé $a \in A$ je obsaženo v nějaké A_t.

2) Pro každou $R_i \in \tau$ a každou α-tici $(a_1, \ldots, a_\alpha) \in R_i^{\mathcal{A}}$ existuje uzel $t \in T$ takový, že $\{a_1, \ldots, a_\alpha\} \subseteq A_t$.

3) Pro každé $a \in A$ indukuje množina $\{t \mid a \in A_t\}$ souvislý podstrom ve stromu T.

Definice 6. *Šířka rozkladu* $\mathcal{T}$ je $\max\{|A_t| \mid t \in T\} - 1$, neboli velikost největší kapsy rozkladu minus 1.

Definice 7. *Stromová šířka struktury* $\mathcal{A}$ je nejmenší šířka přes všechny možné rozklady $\mathcal{T}$. Značíme ji $tw(\mathcal{A})$.

Definice 8. *podmínka 3')*: pro každé dva uzly t_i a t_j a uzel t_k, který leží na cestě mezi nimi, platí $A_i \cap A_j \subseteq A_k$.

Tvrzení 9. *Podmínka 3) v definici stromového rozkladu je ekvivalentní podmínce 3') výše [Klo94].*

Co říkají definice výše pro grafy? Podmínka 1) zajišťuje, že každý vrchol je obsažený v nějaké kapse. Podmínka 2) zaručuje něco podobného pro hrany – kapsy sice žádné hrany neobsahují (jsou to jen podmnožiny vrcholů), ale tato podmínka říká, že pro každé dva vrcholy spojené hranou existuje kapsa, ve které se vyskytují oba dva. Z podmínky 3) pak plyne většina vlastností stromových rozkladů, díky nimž jsou vhodné pro dynamické programování. Tou nejdůležitější pro nás je, že kapsy stromového rozkladu jsou tzv. *separátory*:

Definice 10. *Separátor* (nebo též *vrcholový řez*) grafu G je taková množina vrcholů $S \subseteq V$, jejíž odebrání zvýší počet komponent podgrafu G indukovaného množinou $V \setminus S$.

Tvrzení 11. *Každá kapsa A_t stromového rozkladu $\mathcal{T}$ je separátorem původního grafu.*

Důkaz. Podívejme se v T na dva uzly i a j takové, že t je na cestě mezi nimi. Z podmínek 3') a 2) už lze vidět, že žádné dva $u \in A_i \setminus A_t$ a $v \in A_j \setminus A_t$ mezi sebou nemají hranu, a tedy A_t je separátor G. □

Příklad 12. Stromová šířka stromů je 1. Menší být nemůže, protože každé dva vrcholy spojené hranou se musejí nacházet společně v nějaké kapse. Větší také není, jak dokážeme. Ať je G strom. G zakořeníme a hrany zorientujeme směrem od kořene, a postupným průchodem od kořene k listům vytvoříme stromový rozklad. Každá hrana G představuje kapsu v $\mathcal{T}$ a hranou spojíme uzly t_i a t_j, jsou-li odpovídající orientované hrany tvaru (u, v), (v, w).

Důvodem přítomnosti „-1“ v definici stromové šířky $tw(\mathcal{A})$ je právě snaha, aby stromy měly šířku 1. Stromy (a lesy) jsou navíc **právě** grafy šířky 1, což plyne například z pozorování, že topologické kontrakce hran stromovou šířku nezvětší a že šířka úplného grafu K_n je $n-1$. Přesný důkaz těchto tvrzení je snadný, ale mimo rozsah této práce. Více je možné najít např. v knize od Klokse [Klo94].

Získat intuici pro to, co vlastně stromová šířka znamená, nemusí být úplně snadné. Existuje hezká charakterizace pomocí jednoduché hry na „lupiče a policisty“ [LaP93, ST93], která příhodně zapadá mezi další charakterizace hrami, s nimiž se setkáme později.

Představujme si, že graf znázorňuje město, vrcholy jsou křižovatky a hrany ulice mezi nimi. Ve městě honí policisté ve vrtulnících lupiče na motorce. V každém tahu se policisté přemístí nad libovolné křižovatky, načež se lupič snaží zabránit obklíčení a přesunout se (nějakou nestřeženou cestou) na jinou křižovatku, z níž bude mít lepší možnosti útěku. Formálně definujeme hru následovně:

Definice 13. Hru *lupič a k policistů* na grafu $G = (V, E)$ hrají dva hráči, L a P, představující lupiče a policisty. *Pozice* v této hře je dvojice (C, r), kde $C \in \binom{V}{k}$ je k vrcholů G obsazených P a $r \in V$ je vrchol, na kterém se nachází L. V tahu 0 si nejprve P libovolně zvolí $C_0 \in \binom{V}{k}$ a následně si L zvolí $r_0 \in (V \setminus C_0)$. V tahu i pro $i > 0$ si nejprve P vybere $C_i \in \binom{V}{k}$ a pak si L zvolí r_i takový, že mezi r_{i-1} a r_i existuje v $G \setminus (C_i \cap C_{i-1})$ cesta.

P *vyhraje*, pokud se v konečném počtu tahů dostane hra do pozice, ze které L nemá tah. V opačném případě vyhraje L.

Stromová šířka je pak hrou charakterizována takto [LaP93, ST93]:

Tvrzení 14. *P umí vždy vyhrát s $k+1$ policisty právě tehdy, když je G stromové šířky nejvýše k.*

Nalezení stromového rozkladu minimální šířky je sice v obecném případě NP-úplná úloha, pro pevné k ale lze v lineárním čase rozhodnout, zda $tw(G) = k$, a v případě kladné odpovědi algoritmus vrátí příslušný stromový rozklad [Bod96]. Navíc se ukazuje, že na většině grafů určí i heuristické algoritmy stromovou šířku skoro přesně a především podstatně rychleji než výše zmíněný lineární algoritmus [BK10].

Courcellovu větu dokážeme nejen pro grafy, ale pro obecné relační struktury. K nalezení stromového rozkladu struktury budeme používat *Gaiffmanův graf*:

Definice 15. Pro strukturu $\mathcal{A}$ získáme její *Gaiffmanův graf* následovně. Vrcholy jsou prvky z $dom(\mathcal{A})$ a dva prvky a_i, a_j jsou incidentní, nacházejí-li se společně v nějaké relaci (nebo-li pokud existuje k tž. $a_i, a_j \in \bar{a} \in R_k^{\mathcal{A}}$).

Většina algoritmů konstruujících stromové rozklady se zaměřuje pouze na grafy. Je ale snadné si uvědomit, že pro nalezení rozkladu struktury $\mathcal{A}$ stačí zkonstruovat její Gaiffmanův graf a najít jeho rozklad. Ten bude přesně splňovat podmínky pro rozklad původní struktury $\mathcal{A}$ a bude mít i stejnou šířku [GPW10].

Pohledem na Gaiffmanův graf můžeme také zobecnit definici separátoru.

Definice 16. Buď $\mathcal{A}$ struktura a $S \subseteq dom(\mathcal{A})$ množina jejích prvků. Nechť je $\mathcal{A}' = (A', R_1^{\mathcal{A}'}, \ldots, R_n^{\mathcal{A}'})$ struktura získaná odstraněním prvků S z $dom(\mathcal{A})$ a odstraněním relací obsahujících prvky z S.

Pak je S *separátor struktury* $\mathcal{A}$, pokud lze $dom(\mathcal{A}')$ rozdělit na k disjunktních $A_1 \uplus \cdots \uplus A_k = dom(\mathcal{A}')$ tak, že pro všechny relace R_i platí ekvivalence $(a_1, \ldots, a_{\alpha_i}) \in R_i^{\mathcal{A}'} \Leftrightarrow \exists j : \{a_0, \ldots, a_{\alpha_i}\} \subseteq A_j$ a navíc je maximální takové k striktně větší, než u $\mathcal{A}$.

Z pohledu na Gaiffmanův graf struktury lze také nahlédnout, že dříve vyslovené tvrzení pro grafy ("vrcholy kapsy jsou separátor") platí i pro struktury.

2.2.2 Hezký rozklad a normalizovaný rozklad

Pro snazší konstrukci algoritmů pro struktury omezené stromové šířky se často zavádí různé speciální stromové rozklady. Uvedeme dva z nich. Známější *hezký* rozklad [Klo94] a pak také *normalizovaný* [GPW10] rozklad, který budeme používat v hlavním důkazu.

Definice 17. Stromový rozklad $\mathcal{T} = (T, (A_t)_{t \in T})$ nazveme *hezkým*, pokud je T zakořeněný binární strom s hranami zorientovanými směrem od kořene a pro každý vnitřní uzel i platí, že:

- buď má dva syny j, j' s kapsami A_j, $A_{j'}$ a platí $A_j = A_{j'} = A_i$,
- nebo má právě jednoho syna j s kapsou A_j, pak buď A_i obsahuje všechny prvky z A_j a jeden navíc, nebo v ní naopak jeden prvek chybí.

První případ (uzel se dvěma syny) nazýváme *spojovací* uzel. V druhém případě jde o *zaváděcí* (resp. *zapomínací*) uzel, pokud kapsa uzlu i obsahuje jeden vrchol navíc (resp. o jeden vrchol méně).

Definice 18. Stromový rozklad $\mathcal{T} = (T, (\bar{a}_t)_{t \in T})$ šířky w nazveme *normalizovaným*, pokud je T zakořeněný binární strom s hranami zorientovanými směrem od kořene a platí následující podmínky:

- všechny kapsy jsou $(w+1)$-tice $(a_0, \ldots, a_w)$ různých prvků z $dom(\mathcal{A})$. Obvykle tedy píšeme $\bar{a}$, přesto nevede-li to ke zmatení, povolujeme i zápis A_t a s kapsou pracujeme jako s množinou.
- vnitřní uzly jsou jednoho ze tří druhů:
  1) *permutační* uzel t s kapsou $(a_0, \ldots, a_w)$ má jediného syna s kapsou $(a_{\pi(0)}, \ldots, a_{\pi(w)})$ pro π nějakou permutaci na $w+1$ prvcích (kapsy se liší jen pořadím prvků),
  2) *nahrazovací* uzel t s kapsou $(a_0, a_1, \ldots, a_w)$ má jediného syna s kapsou $(a_0', a_1, \ldots, a_w)$ (kapsy se liší jen na prvním prvku, zbytek je stejný),
  3) *větvící* uzel má dva syny a všechny tři uzly mají identické kapsy.

Pro takové speciální rozklady se již algoritmy sestrojují podstatně lépe – typicky pro jednotlivé druhy uzlů zavedeme pravidla, podle nichž výpočet postupuje a průchodem od listů ke kořeni se úloha vyřeší. Abychom ale měli jistotu, že si můžeme vždy dovolit normalizovaný rozklad předpokládat, budeme potřebovat následující tvrzení. Obdobné tvrzení platí i pro hezký rozklad.

Tvrzení 19. *Každý stromový rozklad $\mathcal{T}$ struktury $\mathcal{A}$ lze v lineárním čase převést na normalizovaný stromový rozklad $\mathcal{T}'$ stejné šířky w a velikosti lineární vzhledem k velikosti původního rozkladu [GPW10].*

Důkaz. Bez újmy na obecnosti předpokládáme, že doména $dom(\mathcal{A})$ má alespoň $w+1$ prvků. Následujících pět kroků převede $\mathcal{T}$ na $\mathcal{T}'$; každý krok zachovává všechny požadavky na stromový rozklad a provede se v lineárním čase:

1) Všechny kapsy doplníme na plnou velikost $w+1$ zkopírováním prvků ze sousedních uzlů. Nechť je s uzel a jemu příslušná kapsa A_s obsahuje $w+1$ prvků – alespoň jeden takový uzel musí v rozkladu existovat. Nechť je s' soused s, jehož kapsa obsahuje $w'+1$ prvků pro $w' < w$. Pak $|A_s \setminus A_{s'}| \geq (w - w')$ a přidáme $(w - w')$ prvků z $A_s \setminus A_{s'}$ do $A_{s'}$.

2) Chceme, aby měl každý uzel nejvýše dva syny. Nechť je s nějaký vnitřní uzel s $k+2$ syny $t_1, \ldots, t_{k+2}$ pro $k > 0$. Standardní postup v takovém případě je doplnit tuto část stromu k kopiemi s, ke kterým připojíme jednotlivé $t_1, \ldots, t_k$. Přesněji: postupně vytváříme kopie s, uzly $s_1, \ldots, s_k$ s $A_{s_i} = A_s$ tak, že druhý syn s je s_1, druhý syn s_1 je s_2 atd. Dále t_1 zůstává prvním synem s, zatímco t_2 učiníme prvním synem s_1, t_3 prvním synem s_2 a tak dále, až t_{k+1} prvním synem s_k. Nakonec připojíme t_{k+2} jako druhého syna s_k.

3) Nyní je již rozklad binární, větvení ale nemusí nutně splňovat podmínku, aby byly kapsy otce s a synů t_1, t_2 identické. V takovém případě pro syna t_i nesplňujícího tuto podmínku vložíme mezi něj a s uzel s_i, který je kopií s.

4) Dále bychom chtěli, aby rozklad splňoval podmínku na nahrazovací uzly. Nechť je s otec s' a jejich kapsy se liší o $k > 0$ prvků ($|A_s \setminus A_{s'}| = k$). Pak „proložíme“ s a s' novými uzly $s_1, \ldots, s_{k-1}$ tak, že s_{k-1} je novým otcem s', s_{k-2} je otcem s_{k-1} a tak dále, až s je otcem s_1. Kapsy A_{s_i} definujeme tak, že pro dva sousední uzly se liší právě v jednom prvku, např. $|A_s \setminus A_{s_1}| = |A_{s_1} \setminus A_s| = 1$.

5) Sousední uzly se teď liší vždy nejvýše v jednom prvku, je ale potřeba zajistit, aby to byl první prvek kapsy (připomeňme, že kapsy jsou uspořádané). Mějme dva uzly s a s', jejichž kapsy se liší právě v jednom prvku, tedy $\exists! a \in A_s$ s $a \notin A_{s'}$ a $\exists! a' \in A_{s'}$ s $a' \notin A_s$. Pak mezi s a s' vložíme dva uzly t a t' tak, že A_t obsahuje stejné prvky, jako A_s, ale a je na pozici 0 a obdobně $A_{t'}$ má stejné prvky, jako $A_{s'}$, ale a' je na pozici 0.

□

2.2.3 Indukované podstruktury

Pro hlavní důkaz budeme potřebovat ještě jeden pojem týkající se stromových rozkladů. Díváme-li se na zakořeněný rozklad a nějaký vnitřní uzel t s příslušnou kapsou A_t, prvky struktury obsažené v podstromu tohoto uzlu přirozeně odpovídají jedné z komponent, na které se $\mathcal{A}$ rozpadne po odebrání A_t (připomeňme, že A_t je separátor).

Definice 20. Pro zakořeněný strom T s hranami zorientovanými směrem od kořene a vnitřní vrchol $t \in T$ značíme T_t podstrom T zakořeněný v t s hranami orientovanými od kořene. Obdobně pro $\mathcal{T}$ normalizovaný stromový rozklad struktury $\mathcal{A}$ a vnitřní uzel $t \in T$ značíme $\mathcal{T}_t = (T_t, (A_s)_{s \in T_t})$. Kapsa u uzlu t je $A_t = \bar{a} = (a_0, \ldots, a_w)$.

Potom strukturu $(\mathcal{A}[\mathcal{T}_t], \bar{a})$ nazýváme *indukovaná podstruktura uzlu t*, její doménou $dom(\mathcal{A}[\mathcal{T}_t])$ je sjednocení prvků kapes $\mathcal{T}_t$, jejími relacemi restrikce původních $R_i^{\mathcal{A}}$ na prvky domény a $\bar{a}$ označuje prvky kapsy A_t (neboli prvky $a_0, \ldots, a_w$ jsme přidali do struktury jako konstanty). Pro značení domény $dom(\mathcal{A}[\mathcal{T}_t])$ používáme pro zkrácení zápis $A[\mathcal{T}_t]$.

2.3 Logické formule

Nyní bychom chtěli formálně definovat jazyk, kterým budeme popisovat grafové vlastnosti – monadickou logiku druhého řádu, MSO – a povšimnout si několika jeho vlastností, které budou důležité v následující podkapitole o logických hrách. Začneme lépe známou (predikátovou) logikou prvního řádu a posléze ji rozšíříme na MSO. Následující definice a pozorování vycházejí především z Libkinovy výborné knihy [Lib04].

Definice 21. *Formule logiky prvního řádu nad abecedou τ.* Předpokládáme spočetně nekonečnou množinu *proměnných*, které značíme malými písmeny $x, y, z, \ldots$. Pracujeme nad konečnou abecedou symbolů τ pro relace a konstanty (viz Kapitolu 2.1). Formule tedy zavedeme induktivně následovně:

- jsou-li x a y proměnné nebo konstanty, pak $x = y$ je (atomická) formule (zkracujeme *atom*)
- jsou-li $x_1, \ldots, x_k$ proměnné nebo konstanty a R je k-ární relační symbol, je $R(x_1, \ldots, x_k)$ (atomická) formule
- jsou-li φ_1 a φ_2 formule, jsou i $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$ a $\neg\varphi_1$ formule
- je-li φ formule, jsou i $\exists x \varphi(x)$ a $\forall x \varphi(x)$ formule.

Zkratkou FO značíme množinu formulí logiky prvního řádu, spojením *FO formule* formuli z této množiny.

Poznamenejme, že jsme se omezili pouze na abecedy s relacemi a konstantami, tedy takové, které neobsahují funkce. FO se obvykle zavádí s nimi, ale my pracujeme s konečnými strukturami, kde se i funkce dají reprezentovat relacemi; zavedení definic by pak přítomnost funkcí jen zbytečně komplikovala.

Definice 22. *Volné proměnné formule* definujeme induktivně takto:

- Volné proměnné formule $x = y$ jsou proměnné z $\{x, y\}$,
- Volné proměnné formule $R(x_1, \ldots, x_k)$ jsou proměnné z $\{x_1, \ldots, x_k\}$,
- Negace ($\neg$) nemění množinu volných proměnných formule, volné proměnné $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$ jsou volné proměnné φ_1 a φ_2,

- Volné proměnné $\exists x\varphi(x)$ a $\forall x\varphi(x)$ jsou volné proměnné φ bez x.

Sentence je formule bez volných proměnných.

Chceme-li zdůraznit, že $\bar{x} = (x_1, \ldots, x_n)$ jsou volné proměnné (ne nutně všechny) formule φ, píšeme $\varphi(\bar{x})$.

Definice 23. Pro danou τ-strukturu $\mathcal{A}$, FO formuli $\varphi(\bar{x})$ a n-tici prvků domény $\bar{a} = (a_1, \ldots, a_n) \in A^n$, určující *ohodnocení* volných proměnných formule $\varphi(\bar{x})$, definujeme *platnost formule při ohodnocení* $\bar{a}$ *ve struktuře* $\mathcal{A}$ (značíme $\mathcal{A} \models \varphi(\bar{a})$) opět induktivně. Mluvíme-li o platnostech atomických formulí, myslíme u konstant ohodnocením jejich realizace ve struktuře.

- Pro $n = 2$ a $\varphi(\bar{x}) \equiv x_1 = x_2$ platí $\mathcal{A} \models \varphi(\bar{a})$ právě tehdy, když ohodnocení přiřazuje x_1 a x_2 tentýž prvek z A.
- Pokud $\varphi(\bar{x}) \equiv R(x_1, \ldots, x_n)$, pak $\mathcal{A} \models \varphi(\bar{a})$ právě tehdy, když $\bar{a} \in R^{\mathcal{A}}$.
- $\mathcal{A} \models \neg\varphi(\bar{a})$, právě když $\mathcal{A} \not\models \varphi(\bar{a})$. $\mathcal{A} \models \varphi_1(\bar{a}) \vee \varphi_2(\bar{a})$, právě když $\mathcal{A} \models \varphi_1(\bar{a})$ nebo $\mathcal{A} \models \varphi_2(\bar{a})$. $\mathcal{A} \models \varphi_1(\bar{a}) \wedge \varphi_2(\bar{a})$, právě když $\mathcal{A} \models \varphi_1(\bar{a})$ a zároveň $\mathcal{A} \models \varphi_2(\bar{a})$.
- Pokud $\psi(\bar{x}) \equiv \exists y\varphi(y, \bar{x})$, pak $\mathcal{A} \models \psi(\bar{a})$, právě když pro nějaké $a' \in A$ platí $\mathcal{A} \models \varphi(a', \bar{a})$. Pokud $\psi(\bar{x}) \equiv \forall y\varphi(y, \bar{x})$, pak $\mathcal{A} \models \psi(\bar{a})$, právě když pro všechna $a' \in A$ platí $\mathcal{A} \models \varphi(a', \bar{a})$.

Hezký algoritmický popis této definice pomocí tzv. *Hintikka her* uvedeme v následujícím Oddíle 2.4 o logických hrách.

V mnoha případech nemusíme rozlišovat mezi formulemi s volnými proměnnými a sentencemi, známe-li ohodnocení formule. O tom mluví následující tvrzení [Lib04].

Tvrzení 24. *Buď $\varphi(\bar{x})$ formule nad abecedou $\tau = \tau_c \cup \tau_R$ s n volnými proměnnými $\{x_1, \ldots, x_n\}$. Nechť je Φ sentence nad abecedou $\tau = \tau'_c \cup \tau_R$ vzniklou z $\varphi(\bar{x})$ nahrazením všech výskytů proměnných x_i novými konstantními symboly c_i ($\tau'_c = \tau_c \cup \{c_1, \ldots, c_n\}$).*

Pak pro ohodnocení formule $\varphi(\bar{x})$ prvky $\bar{a} = \{a_1, \ldots, a_n\}$ platí $\mathcal{A} \models \varphi(\bar{a}) \leftrightarrow (\mathcal{A}, a_1, \ldots, a_n) \models \Phi$.

Definice 25. O dvou formulích $\varphi(\bar{x})$ a $\psi(\bar{y})$ řekneme, že jsou *ekvivalentní*, mají-li na všech strukturách při všech ohodnoceních stejnou platnost. *Neekvivalentní* jsou právě ty formule, pro něž existuje struktura a ohodnocení takové, že jedna platí a druhá ne.

Definice 26. Formule $\varphi(\bar{x})$ je v *prenexní normální formě*, skládá-li se nejprve z počátečního úseku kvantifikátorů, následovaného bezkvantifikátorovou formulí $\psi(\bar{y}, \bar{x})$.

Definice 27. Formule $\varphi(\bar{x})$ je v *negační normální formě*, nacházejí-li se v ní negace jen na atomických podformulích, tedy buď jako $\neg(x = y)$, nebo $\neg R(x_1, \ldots, x_k)$, a nikde jinde.

Je známým faktem [Har09], že pro každou FO formuli existuje ekvivalentní formule, která vyhovuje jak definici prenexní, tak i normální formy. Lze ji získat pomocí rekurzivní aplikace několika přepisovacích pravidel, které vycházejí ze syntaktických ekvivalencí formulí.

Definice 28. *Kvantifikátorá hloubka formule* φ je počet kvantifikátorů na začátku formule, převedeme-li ji do prenexního tvaru. Značíme ji $qr(\varphi)$.

Neekvivalentních formulí FO je zjevně nekonečně mnoho. Situace začíná být zajímavější, omezíme-li svůj pohled na formule s m volnými proměnnými a kvantifikátorové hloubky nejvýše k. Pro ně dovedeme vyslovit jedno důležité tvrzení.

Definice 29. FO[k] je množina všech FO formulí nad konečnou abecedou τ kvantifikátorové hloubky nejvýše k.

Tvrzení 30. *FO[k] obsahuje jen konečně mnoho neekvivalentních formulí s m volnými proměnnými.*

Důkaz. Tvrzení dokážeme indukcí přes k. Základní krok indukce je FO[0]. Ta obsahuje právě atomické formule v m volných proměnných a jejich booleovské kombinace. Atomické formule jsou syntaktické kombinace proměnných $x_1, \ldots, x_m$ a relačních symbolů z τ. Tyto pak můžeme spojovat logickými spojkami $\neg$, $\wedge$ a $\vee$ a takových kombinací je až na logickou ekvivalenci konečně mnoho.

Indukční krok od i k $i+1$ využívá faktu, že každá $\varphi(x_1, \ldots, x_m) \in \text{FO}[k]$ je booleovskou kombinací formulí $\exists x_{m+1}\psi(x_1, \ldots, x_m, x_{m+1})$ kde $\psi \in \text{FO}[i]$. Protože je z indukčního předpokladu neekvivalentních formulí FO[i] s $m+1$ volnými proměnnými (až na ekvivalenci) jen konečně mnoho, je jen konečně mnoho i jejich booleovských kombinací, tedy formulí z FO[$i+1$] s m volnými proměnnými. □

Nyní jsme již připraveni formálně zavést monadickou logiku druhého řádu a povšimnout si, že pro ni platí analogická tvrzení.

Definice 31. *Formule monadické logiky druhého řádu* definujeme následovně. Každá FO formule je i MSO formulí. Zavádíme nový typ proměnných, představující množiny prvků domény struktury, a značíme je velkými písmeny $X, Y, Z, \ldots$. Abychom mohli tyto proměnné rozumně používat, zavádíme též nové atomické formule tvaru $x \in X$ pro x proměnnou prvního řádu a X proměnnou druhého řádu. Induktivní aplikací logických spojek ($\neg$, $\wedge$ a $\vee$) a množinových kvantifikátorů ($\exists Y\varphi(\bar{x}, Y, \bar{X})$ a $\forall Y\varphi(\bar{x}, Y, \bar{X})$) získáme všechny MSO formule.

MSO formule s *volnými proměnnými* prvního řádu $\bar{x}$ a druhého řádu $\bar{X}$ se označuje $\varphi(\bar{x}, \bar{X})$. Volné proměnné prvního řádu budeme též nazývat jako *prvkové* a proměnné druhého řádu jako *množinové*. Obdobně $\exists x$ je *prvkový* kvantifikátor a $\exists X$ *množinový*.

Definice *platnosti* MSO formule je též analogická – atomické formule $x \in X$ i množinové kvantifikace interpretujeme zjevným způsobem.

Příklad 32. Za příklad MSO formule, se kterou budeme pracovat, uveďme formuli popisující, že graf je obarvitelný třemi barvami. Je to tedy MSO formule nad abecedou $\tau = \{E\}$ pro E binární relační symbol incidence dvou vrcholů.

$$\begin{aligned}\varphi_{3col} \equiv &\exists R \exists G \exists B[Partition(R, G, B) \wedge \forall v_1 \forall v_2[E(v_1, v_2) \rightarrow \\ &\neg(v_1 \in R \wedge v_2 \in R) \wedge \neg(v_1 \in G \wedge v_2 \in G) \wedge \neg(v_1 \in B \wedge v_2 \in B)]\end{aligned}$$

kde

$$\begin{aligned}Partition(R,G,B) \equiv &\forall v\ [[(v\ \in R) \vee (v\ \in G) \vee (v\ \in B)] \wedge \\ &\neg(v \in R \wedge v\ \in G) \wedge \neg(v \in R \wedge v\ \in B) \wedge \neg(v \in G \wedge v\ \in B)]].\end{aligned}$$

Je snadné si uvědomit, že všechna tvrzení a definice výše uvedená pro FO se přenášejí i na MSO, především pak následující tři:

Definice 33. MSO[k] je množina všech MSO formulí nad konečnou abecedou τ kvantifikátorové hloubky nejvýše k.

Tvrzení 34. *Pro formuli $\varphi(\bar{x}, \bar{X})$ s n volnými prvkovými proměnnými a m volnými množinovými proměnnými mějme sentenci Φ nad abecednou τ' vzniklou nahrazením výskytů x_i novými konstantami c_i a nahrazením výskytů X_j novými unárními relacemi R_j (tedy $\tau' = \tau'_c \cup \tau'_R$ pro $\tau'_c = \tau_c \cup \{c_1, \ldots, c_n\}$ a $\tau'_R = \tau_R \cup \{R_1, \ldots, R_m\}$).*

Pak pro ohodnocení formule $\varphi(\bar{x}, \bar{X})$ prvky $\bar{a} = \{a_1, \ldots, a_n\}$ a množinami $\bar{A} = (A_1, \ldots, A_m)$ platí $\mathcal{A} \models \varphi(\bar{a}) \leftrightarrow (\mathcal{A}, a_1, \ldots, a_n, A_1, \ldots, A_m) \models \Phi$.

Tvrzení 35. *MSO[k] obsahuje jen konečně mnoho neekvivalentních formulí s m volnými proměnnými.*

Na místě je jistě otázka, jak se liší *monadická* logika druhého řádu od plné logiky druhého řádu (SO). MSO je restrikce SO v tom smyslu, že MSO umožňuje kvantifikovat pouze přes podmnožiny domény, zatímco SO obsahuje kvantifikátory přes podmnožiny relací libovolných arit.

V literatuře se občas vyskytuje značení MSO_1 a MSO_2, kde první zmíněná zkratka je MSO, jak jsme ji definovali, zatímco druhá je MSO doplněná kvantifikacemi přes relace arity 2 (což jsou například hrany). Jak ukážeme později, Courcellovu větu lze rozšířit na libovolné MSO_k pro pevné k.

Příklad 36. Příkladem formule z MSO_2 je například formule popisující hamiltonicitu grafu. Protože se nyní může ve formuli objevovat kvantifikace několika druhů, explicitně je vyznačíme – „$\in V$“ (a „$\subseteq V$“) značí kvantifikace přes vrcholy (a jejich množiny), „$\in E^{\mathcal{G}}$“ (a „$\subseteq E^{\mathcal{G}}$“) značí kvantifikaci přes hrany. Definice by to samozřejmě příslušným způsobem odrážela. Dále se ve formuli objevují některé spojky a relace ($\rightarrow$, $\subseteq$), které jsme nedefinovali, ale lze na ně pohlížet jako na zkratky pro často používané vztahy, které jsou vyjádřitelné i v původním jazyce.

$$\begin{aligned}\varphi_{Hamiltonian} \equiv &\exists H \subseteq E^{\mathcal{G}} \\ &[\forall u\ \in V\ \exists v, w \in V\ : (v\ \neq w) \wedge ((u,v) \in H) \wedge ((u,w) \in H) \wedge \\ &\wedge [\forall x \in V : (u,x) \in H \rightarrow ((x = v) \vee (x = w))]] \wedge \\ &\qquad\qquad H \textit{ je 2-faktor...} \\ &\wedge [\forall W \subseteq V : ((W \neq V) \wedge (\exists u\ \in V : u\ \in W)) \rightarrow \\ &\rightarrow (\exists v, w \in V : (v,w) \in H \wedge (v\ \in W) \wedge (w \notin W))] \\ &\qquad\qquad \textit{... a je souvislá.}\end{aligned}$$

2.4 Logické hry

V předchozí podkapitole jsme se již setkali s charakterizací stromové šířky pomocí hry na „lupiče a policisty“. Přestože se takový pohled může zdát zprvu jen

k pousmání, kombinatorické hry se používají v mnoha dalších oblastech a, jak ukážeme v tomto oddíle, jsou jádrem našeho důkazu Courcellovy věty. Většina následujících výsledků pochází z oboru zvaného *teorie konečných modelů*. Pro bližší informace nebo další kontext se lze obrátit na knihy Libkina [Lib04], Grädela et al. [GKL$^+$07] nebo přímo Grädelovu kapitolu z předešlé knihy, kde se zabývá vztahem teorie konečných modelů a deskriptivní složitosti [Gr 07]. Zajímavý je též Grädelův článek o vztahu různých logických jazyků a her [Gr 11].

Níže popisované dvě hry patří mezi *poziční hry*, které důkladně (ač z jiného úhlu pohledu) zkoumal Beck v knize [Bec08]. Naše definice spíš vychází ze článku [KLR11], který tyto hry řadí blíže mezi „hry s oblázky“ (*pebble games*), ačkoliv ani tento termín situaci příliš nevyjasňuje, neboť se v literatuře nepoužívá konzistentně.

Definice 37. *Poziční hra* mezi dvěma hráči, hráčem 0 a hráčem 1, se skládá z množiny *pozic* P, acyklické binární relace *tahů* $M \subseteq P \times P$, dvou disjunktních množin P_0 a P_1 určujících, ze kterých pozic táhne který hráč, a počáteční pozice p_0. Tuto hru značíme (P, M, P_0, P_1, p_0). Požadujeme, aby tahy vedly pouze z pozic, které jsou přiřazené jednomu z hráčů (neboli aby M splňovala $\forall(p, p') \in M : p \in P_0 \cup P_1$). Naproti tomu povolujeme, aby tahy vedly do pozic, ze kterých žádný tah nevede. Též povolujeme existenci pozic, které nejsou přiřazené žádnému z hráčů (neboli $p \in P \setminus (P_0 \cup P_1)$).

Na začátku je na tahu hráč i, pro nějž platí $p_0 \in P_i$. Pro p_c aktuální pozici:

- Hráč na tahu vybere nějaký tah vedoucí z p_c (nějaké $p' \in P$, pro které $(p_c, p') \in M$). Potom:
- Pokud $p' \in P_0$, táhne hráč 0,
- Pokud $p' \in P_1$, táhne hráč 1.

Hra skončí v okamžiku, kdy z aktuální pozice p_k nevede žádný tah (pro p_k neexistuje žádná dvojice $(p_k, p_l) \in M$). V takovém případě rozlišujeme tři situace:

- Hráč 0 *vyhrál*, pokud $p_k \in P_1$,
- Hráč 1 *vyhrál*, pokud $p_k \in P_0$,
- Nastala *remíza*, pokud $p_k \notin P_0 \cup P_1$.

Hráč má *vyhrávající strategii*, pokud může vyhrát pokaždé, bez ohledu na tahy druhého hráče.

2.4.1 Hintikka hry

Začneme definicí *Hintikka her*. Jejich myšlenka je velice jednoduchá a hry lze považovat takřka za folklór, protože jsou ve všeobecném povědomí, ačkoliv ne každý zná jejich původ [Hin73]. Pro náš důkaz nejsou zcela nutné, ale dají se použít jako jeden ze způsobů ověření platnosti formule (neboli relace $\mathcal{A} \models \varphi(\bar{a})$). Je ještě jeden důvod, proč těmto hrám věnovat pozornost. Jistá jejich modifikace, o které se zmíníme v Kapitole 4, je základem jiného přístupu ke Courcellově větě, který se v současné době zdá být asi nejnadějnějším, jde-li o přímou konstrukci praktického algoritmu ze vstupní formule [KLR11].

Definice 38. *Hintikka hra* pro strukturu $\mathcal{A}$ a formuli $\Phi(\bar{x})$ v negační normální formě a dosazení $\bar{a}$ do volných proměnných Φ je poziční hra pro dva hráče, kterým můžeme říkat třeba v duchu kombinatorické teorie her *stavitel* a *ničitel*. Stavitel se snaží dokázat, že $\mathcal{A} \models \Phi(\bar{a})$, ničitel opak. *Pozice* hry jsou dvojice $(\varphi, \bar{b})$, kde φ je podformule Φ a $\bar{b}$ je dosazení do volných proměnných φ. Hru značíme $\mathrm{MCG}(\mathcal{A}, \Phi(\bar{a}))$ a $(\Phi, \bar{a})$ je její počáteční pozice.

Každá pozice určuje, kdo je na tahu a jaké jsou jeho možnosti:

- Z pozice $(\exists u\varphi, \bar{b})$ (resp. $(\exists U\varphi, \bar{b})$) táhne stavitel výběrem nějakého $u \in A$ (resp. $U \subseteq A$) do pozice (φ, b'), kde b' se získá z $\bar{b}$ doplněním podle volby u (resp. U)
- Z pozice $(\psi_1 \wedge \psi_2, \bar{b})$ táhne stavitel výběrem $\psi \in \{\psi_1, \psi_2\}$ do pozice $(\psi, \bar{b})$.

Ničitel táhne obdobným způsobem z pozic s formulemi $\forall u\varphi$, $\forall U\varphi$ a $\psi_1 \vee \psi_2$.

Hra se zastaví v pozici $(\varphi, \bar{b})$, když je φ atom nebo negovaný atom. Stavitel *vyhrál*, pokud $\mathcal{A} \models \varphi(\bar{b})$.

Jak lze vidět, formule v aktuální pozici určuje, kdo táhne v dalším kole hry. Hráči se tedy nemusejí střídat. Například obsahuje-li formule posloupnost existenčních kvantifikátorů následovanou všeobecným kvantifikátorem, nejprve vybere stavitel ohodnocení všech volných proměnných formule, která následuje za existenčními kvantifikátory, a pak je teprve na tahu ničitel.

Platí, že $\mathcal{A} \models \Phi(\bar{a})$ právě tehdy, když má stavitel vyhrávající strategii ve hře $\mathrm{MCG}(\mathcal{A}, \Phi(\bar{a}))$. Pro konečné $\mathcal{A}$ je pak také zřejmé, že $\mathcal{A} \models \Phi(\bar{a})$ lze rozhodnout vyčerpávající rekurzivní simulací $\mathrm{MCG}(\mathcal{A}, \Phi(\bar{a}))$.

2.4.2 k-typy a elementární ekvivalence

Než přistoupíme k dalšímu typu her, vratíme se na chvíli k FO a FO[k] (a potom analogicky MSO a MSO[k]). Připomeňme, že dle Tvrzení 30 a 35 obsahuje FO[k] a MSO[k] jen konečně mnoho neekvivalentních sentencí. Označme si je $\varphi_1, \ldots, \varphi_M$. Pro každou strukturu $\mathcal{A}$ se můžeme podívat postupně na platnost všech těchto sentencí na $\mathcal{A}$ a sestrojit M-tici nul a jedniček, která tuto informaci vyjadřuje. Takových M-tic může být také jen konečně mnoho. Každá struktura $\mathcal{A}$ je tedy nějakého *typu*, podle toho, jaká M-tice jí přísluší, a všechny sentence z FO[k] mají na strukturách stejného typu stejnou platnost. Formálně řečeno:

Definice 39. Dvě struktury $\mathcal{A}, \mathcal{B}$ nazveme FO[k]-*ekvivalentními* (a píšeme $\mathcal{A} \equiv_k^{FO} \mathcal{B}$), pokud na nich platí právě stejné sentence z FO[k], neboli pro každou $\varphi \in \mathrm{FO}[k]$ platí $\mathcal{A} \models \varphi$ právě tehdy, když $\mathcal{B} \models \varphi$.

Zápis $(\mathcal{A}, \bar{a}) \equiv_k^{FO} (\mathcal{B}, \bar{b})$ značí ekvivalenci struktur s konstantami, kterou zavedeme obdobně jako výše, ovšem jako sentence použijeme Φ a Ψ vzniklé nahrazením volných proměnných formulí $\varphi(\bar{x})$ a $\psi(\bar{y})$ dle Tvrzení 24.

Tato relace je zjevně ekvivalencí a její rozkladové třídy nazýváme FO[k]-*typy*.

Tvrzení 40. *Relace $\equiv_k^{FO}$ je konečného indexu, neboli* FO[k]*-typů je konečně mnoho.*

Důkaz. Viz úvahu výše. □

Definice 41. Pro $\varphi_1, \ldots, \varphi_M$ neekvivalentní sentence FO[k], strukturu $\mathcal{A}$ a pro k-typ t nechť je α_t sentence následujícího tvaru: $\alpha_t \equiv (\bigwedge_{i \in T} \varphi_i) \wedge (\bigwedge_{j \notin T} \neg\varphi_j)$, kde T je množina indexů formulí φ_i, pro které platí $\mathcal{A} \models \varphi_i$. Tuto formuli nazýváme *charakteristická formule typu t a struktury $\mathcal{A}$.*

Poznámka. V úvaze, která je i důkazem Tvrzení 40, si ještě můžeme povšimnout jednoho detailu. Mohlo by se zdát, že k-typů může být až 2^M pro M rovné počtu neekvivalentních FO[k] sentencí. Ve skutečnosti tomu tak není – dokonce k-typů nemůže být více než M.

Povšimněme si, že pro každé dva k-typy t_1 a t_2 a strukturu $\mathcal{A}$ se charakteristické formule k-typů α_{t_1} a α_{t_2} vzájemně vylučují – když $\mathcal{A} \models \alpha_{t_1}$, pak $\mathcal{A} \not\models \alpha_{t_2}$. Proto lze každý k-typ úplně charakterizovat jeho charakteristickou formulí α_t.

Samotná formule α_t ale také pochází z FO[k] (je jen booleovskou kombinací FO[k] formulí a neobsahuje nové kvantifikátory) a jak víme, takových neekvivalentních je M. Proto je i k-typů nejvýše M a tedy ne každá M-tice nul a jedniček odpovídá nějakému k-typu, jak by se mohlo zdát.

Pro pořádek ještě explicitně uveďme rozšíření definice $\equiv_k^{FO}$ na MSO:

Definice 42. Dvě struktury $\mathcal{A}$ a $\mathcal{B}$ nazveme MSO[k]-*ekvivalentními* (a píšeme $\mathcal{A} \equiv_k^{MSO} \mathcal{B}$), pokud na nich platí právě ty stejné sentence MSO[k], neboli pro každou $\varphi \in$ MSO[k] platí ekvivalence $\mathcal{A} \models \varphi \Leftrightarrow \mathcal{B} \models \varphi$.

Rozkladové třídy $\equiv_k^{MSO}$ nazýváme MSO[k]-*typy*. Protože se už dále nikde nezabýváme ve vztahu ke strukturám jinými *typy*, než MSO[k]-typy, budeme zkráceně používat výraz *k-typ*.

2.4.3 Ehrenfeucht-Fraïssé hry

Definice 43. Nechť jsou $(\mathcal{A}, \bar{c}, \bar{C})$, $(\mathcal{B}, \bar{d}, \bar{D})$ dvě τ-struktury s k konstantami a K unárními relacemi, dále $\bar{a} = (a_1, \ldots, a_n)$, resp. $\bar{b} = (b_1, \ldots, b_n)$ jsou n-tice prvků z $dom(\mathcal{A})$, resp. $dom(\mathcal{B})$. Buďte $\bar{x} = (\bar{a}, \bar{c})$ a $\bar{y} = (\bar{b}, \bar{d})$ pomocné l-tice pro $l = k + n$. Nechť je navíc $\bar{A} = (A_1, \ldots, A_N)$, resp. $\bar{B} = (B_1, \ldots, B_N)$ N-tice podmnožin A, resp. B a $\bar{X} = (\bar{A}, \bar{C})$ a $\bar{Y} = (\bar{B}, \bar{D})$ jsou pomocné L-tice pro $L = K + N$. Pak $(\bar{a}, \bar{A}, \bar{b}, \bar{B})$ je *částečný isomorfismus mezi $\mathcal{A}$ a $\mathcal{B}$*, pokud:

1) pro všechny $i, j \leq l$ platí, že $x_i = x_j$ právě tehdy, když $y_i = y_j$,

2) pro každý k-ární relační symbol $R_i \in \tau$ a každou k-tici $(i_1, \ldots, i_k) \in \{1, \ldots, l\}^k$ platí, že $(x_{i_1}, \ldots, x_{i_k}) \in R_i^{\mathcal{A}}$ právě tehdy, když $(y_{i_1}, \ldots, y_{i_k}) \in R_i^{\mathcal{B}}$,

3) pro všechna $i \leq n$ a $j \leq N$ platí, že $x_i \in X_j$ právě tehdy, když $y_i \in Y_j$.

Počet množin m je libovolný nezáporný a pro případ $m = 0$ zkracujeme zápis částečného isomorfismu na $(\bar{a}, \bar{b})$.

Příklad 44. Pro dva grafy G a H a n-tice vrcholů $(v_1, \ldots, v_n)$ z V_G a $(w_1, \ldots, w_n)$ z V_H je $(\bar{v}, \bar{w})$ částečný isomorfismus mezi G a H, pokud je zobrazení $h(v_i) = w_i$ isomorfismem podgrafů indukovaných vrcholy $\bar{v}$, resp. $\bar{w}$.

Nechť jsou dále $V_1, \ldots, V_m$ podmnožiny V_G a $W_1, \ldots, W_m$ podmnožiny V_H. Indexovou množinu I_{v_i} příslušností vrcholu v_i do množin $V_1, \ldots, V_m$ si můžeme

představit jako barvu vrcholu (obdobně pro w_i, množiny $W_1, \ldots, W_m$ a indexovou množinu I_{w_i}). Pak je $(\bar{v}, \bar{V}, \bar{w}, \bar{W})$ částečným isomorfismem mezi G a H, pokud zobrazení h navíc zachovává barvy.

Nyní můžeme konečně zavést druhý typ logických her, který alternativně popisuje relaci $\equiv_k^{MSO}$. Touto relací se zabýval Fraïssé [Fra54] a popis hrami podal Ehrenfeucht [Ehr61] (původně pro FO a relaci $\equiv_k^{FO}$, rozšíření na $\equiv_k^{MSO}$ přišlo až později). Tento popis nese mnoho výhod. Některé přínosy pro teorii konečných modelů zmíníme v následujícím oddíle. Dále nám tento alternativní popis umožňuje dokázat kompoziční lemma (Lemma 51), které je základem našeho důkazu Courcellovy věty.

Definice 45. *Ehrenfeucht-Fraïssé hra* o k tazích je hra pro dva hráče, které budeme opět nazývat *ničitel* a *stavitel*. Hraje se na dvou strukturách $\mathcal{A}$ a $\mathcal{B}$ a značíme ji $\mathrm{EFG}(\mathcal{A}, \mathcal{B}, k)$. Stavitel se snaží dokázat $\mathcal{A} \equiv_k^{MSO} \mathcal{B}$ a ničitel opak. V tahu i si nejprve ničitel vybere jednu ze struktur ($\mathcal{A}$ nebo $\mathcal{B}$) a v ní pak:

- buď vybere prvek ($a_i \in A$ nebo $b_i \in B$), tah nazýváme *prvkový*,
- nebo vybere podmnožinu ($A_i \subseteq A$ nebo $B_i \subseteq B$), tah nazýváme *množinový*.

Pro prvkový tah odpoví stavitel výběrem prvku ve druhé struktuře, pro množinový tah výběrem podmnožiny ve druhé struktuře. Po i tazích tedy hráči vybrali nějakou posloupnost i prvků a množin, pro strukturu $\mathcal{A}$ prvky $\bar{a}$ a množiny $\bar{A}$ a pro strukturu $\mathcal{B}$ prvky $\bar{b}$ a množiny $\bar{B}$.

Pokud se v kterémkoliv tahu stane, že $(\bar{a}, \bar{A}, \bar{b}, \bar{B})$ není částečný isomorfismus mezi $\mathcal{A}$ a $\mathcal{B}$, řekneme, že stavitel *prohrál*. Naopak, pokud je $(\bar{a}, \bar{A}, \bar{b}, \bar{B})$ po k tazích částečný isomorfismus, stavitel *vyhrál*.

Staviteli se v literatuře obvykle říká *duplikátor*, protože se snaží na druhé struktuře co nejlépe „duplikovat“ tahy ničitele. Můžeme ale také říct, že se snaží úspěšně *postavit* částečný isomorfismus.

Ehrenfeucht-Fraïssé hry jsou klíčovým nástrojem teorie konečných modelů díky větě, kterou brzy vyslovíme a dokážeme. Slouží k dokazování výsledků o popisovací schopnosti různých jazyků, např. pomocí nich lze snadno dokázat, že nelze sestrojit FO formuli, která by rozhodovala souvislost grafu. Podstatou důkazu těchto výsledků je obvykle hledání vyhrávající strategie stavitele a výzkum se snaží najít postačující podmínky pro existenci takové strategie. I nám se bude jedno tvrzení hodit.

Tvrzení 46. *Nechť má stavitel vyhrávající strategii pro hru* $\mathrm{EFG}(\mathcal{A}, \mathcal{B}, k)$. *Dále nechť je* c *prvek* $\mathrm{dom}(\mathcal{A})$ (*resp.* C *podmnožina* $\mathrm{dom}(\mathcal{A})$) *a* d *prvek z* $\mathrm{dom}(\mathcal{B})$ (*resp.* D *podmnožina* $\mathrm{dom}(\mathcal{B})$). *Pak pokud byl* c (*resp.* C) *ničitelův první tah a* d (*resp.* D) *odpověď na něj, stavitel má vyhrávající strategii i pro* $\mathrm{EFG}((\mathcal{A}, c), (\mathcal{B}, d), k-1)$ (*resp.* $\mathrm{EFG}((\mathcal{A}, C), (\mathcal{B}, D), k-1)$).

Důkaz. Protože je d (resp. D) vybráno podle stavitelovy vyhrávající strategie, dokáže stavitel zachovat částečný isomorfismus i ve zbývajících $k-1$ tazích. Proto je jedno, zda byl tento tah vybrán v průběhu té samé hry, nebo jej přidáme dodatečně jako konstantu (resp. relaci) struktury. □

2.4.4 Ehrenfeuchtova-Fraïssého věta

Pro potřeby důkazu ještě rozšíříme definici charakteristické formule typu pro formule s jednou volnou proměnnou.

Definice 47. Nechť jsou $\varphi_1(x), \ldots, \varphi_N(x)$ všechny neekvivalentní formule s jednou volnou proměnnou z MSO[k], $\mathcal{A}$ struktura k-typu t a a prvek $dom(\mathcal{A})$. Pak $\alpha_t(x)$ definujeme jako $\alpha_t(x) \equiv (\bigwedge_{i \in T} \varphi_i(x)) \wedge (\bigwedge_{j \notin T} \neg\varphi_j(x))$, kde T je množina indexů formulí $\varphi_i(x)$, pro které platí $\mathcal{A} \models \varphi_i(a)$. Analogicky definujeme $\alpha_t(X)$ pro podmnožinu A.

Věta 48. *Struktury $\mathcal{A}$ a $\mathcal{B}$ jsou stejného k-typu (tj. $\mathcal{A} \equiv_k^{MSO} \mathcal{B}$) právě tehdy, když má stavitel vyhrávající strategii v* EFG$(\mathcal{A}, \mathcal{B}, k)$.

Důkaz. ($\Leftarrow$) Předpokládáme, že stavitel má vyhrávající strategii v EFG$(\mathcal{A}, \mathcal{B}, k)$ a chceme ukázat, že $\mathcal{A} \equiv_k^{MSO} \mathcal{B}$, tedy že pro každou sentenci $\varphi \in$ MSO[k] platí $\mathcal{A} \models \varphi$ právě tehdy, když $\mathcal{B} \models \varphi$.

Postupujeme indukcí podle k. Základní případ, tedy vítězství stavitele ve hře o 0 tazích, vlastně říká, že podstruktury indukované konstantami jsou isomorfní. Jsou-li struktury isomorfní, jistě na nich platí právě ty stejné atomické sentence (booleovské kombinace atomů).

Předpokládejme, že implikace platí pro $i = k$. Mějme libovolnou sentenci $\varphi \in$ MSO[$i+1$]. Jak víme, ta je ekvivalentní nějaké booleovské kombinaci sentencí tvaru $\exists x\psi(x)$ pro $\psi \in$ MSO[i]; ψ může být i tvaru $\exists X\psi(X)$, ale celý postup je analogický a tak jej provedeme jen pro kvantifikaci prvního řádu. Pak stačí implikaci dokázat pro každou takovou sentenci. Bude-li totiž jejich platnost stejná na obou strukturách, bude jistě stejná i platnost jejich booleovské kombinace.

Podle Definice 23 (o platnosti formule) pokud $\mathcal{A} \models \exists x\psi(x)$, pak pro nějaké $a \in dom(\mathcal{A})$ platí $\mathcal{A} \models \psi(a)$. Buď b stavitelova odpověď v $\mathcal{B}$ na ničitelovu volbu a v prvním kole EFG$(\mathcal{A}, \mathcal{B}, i+1)$. Dle předpokladu, že v této hře má stavitel vyhrávající strategii, a Tvrzení 46, má stavitel vyhrávající strategii i v EFG$((\mathcal{A}, a), (\mathcal{B}, b), i)$. Proto z indukčního předpokladu platí $(\mathcal{A}, a) \equiv_i^{MSO} (\mathcal{B}, b)$, a tedy i $(\mathcal{B}, b) \models \Psi$, z čehož plyne díky Tvrzení 34, že $\mathcal{B} \models \psi(b)$. To jsme chtěli dokázat.

Pokud $\mathcal{A} \not\models \exists x\psi(x)$, pak i $\mathcal{B} \not\models \exists x\psi(x)$. Situace, že by formule na jedné struktuře platila a na druhé ne, nemůže nastat. Pro spor předpokládejme, že platí např. na $\mathcal{A}$. Pak podle úvahy v předešlém odstavci musí platit i na $\mathcal{B}$.

Nyní víme, že všechny existenční sentence, ze kterých se φ skládá, mají na $\mathcal{A}$ i $\mathcal{B}$ stejnou platnost, a tedy ji musí mít i jejich booleovská kombinace – samotná sentence φ.

($\Rightarrow$) Nyní jsou $\mathcal{A}$ a $\mathcal{B}$ dvě struktury stejného k-typu t a chceme ukázat, že stavitel má vyhrávající strategii na EFG$(\mathcal{A}, \mathcal{B}, k)$. Postupujeme opět indukcí.

Základní krok je snadný a odpovídá pohledu na základní krok předchozí implikace z druhé strany: když na dvou strukturách platí ty stejné formule hloubky 0, jsou to právě atomické sentence, což znamená, že jsou struktury isomorfní. Na isomorfních strukturách má stavitel vždy vyhrávající strategii.

Předpokládáme, že na $\mathcal{A}$ a $\mathcal{B}$ platí tytéž MSO[$i+1$] sentence. Nechť ničitel táhne bez újmy na obecnosti do $\mathcal{A}$ (tahy do $\mathcal{B}$ se dokáží symetricky) výběrem prvku $a \in dom(\mathcal{A})$ nebo množiny $A \subseteq dom(\mathcal{A})$; postup pro množinové tahy

je opět analogický, a proto jej vynecháváme. Vezměme $\alpha_t(x)$ charakteristickou formuli i-typu t struktury $\mathcal{A}$ a prvku a (Definice 47).

Ta je kvantifikátorové hloubky i. Z předpokladu $\mathcal{A} \equiv_{i+1}^{MSO} \mathcal{B}$ platí na $\mathcal{A}$ a $\mathcal{B}$ tytéž sentence hloubky $i+1$, tedy i formule $\exists x \alpha_t(x)$. Protože jsme vzali $\alpha_t(x)$ závislou na volbě a, víme, že $\mathcal{A} \models \alpha_t(a)$.

Protože $\exists x \alpha_t(x)$ platí i na $\mathcal{B}$, existuje $b \in dom(\mathcal{B})$ svědek této formule na $\mathcal{B}$. Protože $\alpha_t(x)$ plně charakterizuje i-typ struktury a výběr prvku a platí jak na $\mathcal{A}$, tak na $\mathcal{B}$, pro každou formuli ψ hloubky i platí $\mathcal{A} \models \psi(a) \Leftrightarrow \mathcal{B} \models \psi(b)$. To lze opět Tvrzením 34 ekvivalentně převést na $(\mathcal{A}, a) \models \Psi \Leftrightarrow (\mathcal{B}, b) \models \Psi$, z čehož máme $(\mathcal{A}, a) \equiv_i^{MSO} (\mathcal{B}, b)$.

To nám už díky indukčnímu předpokladu dává, že stavitel má vyhrávající strategii na $\mathrm{EFG}((\mathcal{A}, a), (\mathcal{B}, b), i)$. Protože jsme vybrali a libovolně, můžeme takto najít stavitelovu odpověď na každé a. Pokud má ale stavitel vyhrávající strategii na $\mathrm{EFG}((\mathcal{A}, a), (\mathcal{B}, b), i)$ pro každé a, pak ji má i na $\mathrm{EFG}(\mathcal{A}, \mathcal{B}, i+1)$. To jsme chtěli dokázat.

Stavitelovu odpověď na množinové tahy najdeme obdobně. □

2.4.5 Aplikace Ehrenfeuchtovy-Fraïssého věty

Jak již bylo zmíněno, Ehrenfeucht-Fraïssé hry jsou nástrojem teorie konečných modelů pro dokazování výsledků o popisné schopnosti logických jazyků. Typický postup je následující. Mějme vlastnost $\mathcal{P}$ a logický jazyk $\mathcal{L}$ a nějaké rozdělení $\mathcal{L}$ do tříd $\mathcal{L}[0], \mathcal{L}[1], \ldots, \mathcal{L}[k], \ldots$. Abychom ukázali, že $\mathcal{P} \notin \mathcal{L}$, najdeme pro každé $k \geq 0$ dvě konečné struktury $\mathcal{A}_k$ a $\mathcal{B}_k$ takové, že:

- $\mathcal{A}_k \equiv_k^{\mathcal{L}} \mathcal{B}_k$
- $\mathcal{A}$ má vlastnost $\mathcal{P}$, ale $\mathcal{B}$ nikoliv

Protože jsou obě struktury stejného k-typu a mají na nich všechny formule $\mathcal{L}[k]$ stejnou platnost, a přesto má vlastnost $\mathcal{P}$ jen jedna z nich, můžeme prohlásit, že žádná $\mathcal{L}$ formule nemůže $\mathcal{P}$ rozhodnout.

Příklad 49. Například uveďme tvrzení, že v FO nelze popsat acyklicita grafu. Nebudeme zacházet do detailů (ty lze nalézt opět v Libkinově knize [Lib04]), ale uveďme, že pro každé k je potřeba sestrojit dva grafy G_k^1 a G_k^2 následujícím způsobem. G_k^1 je cesta délky $2m$, G_k^2 jsou disjunktní cyklus a cesta, oboje délky m. Pak se ověří, že pro $m > 2^{k+1}$ platí $G_k^1 \equiv_k^{FO} G_k^2$, ale zjevně G_k^1 je acyklický a G_k^2 ne.

V důkazu Courcellovy věty tuto metodologii potřebovat nebudeme, ale zcela od věci není. Například se pomocí ní dokáže, že hamiltonicitu nelze popsat formulí z MSO_1. Dostaneme-li rozhodovací problém nad strukturami s omezenou stromovou šířkou a napadne nás použít Courcellovu větu, můžeme se v případě neúspěšného snažení pokusit dokázat, že vlastnost nelze pomocí MSO popsat. To ještě nemusí znamenat, že by problém nešel řešit jiným způsobem, ale je to dobrý první krok.

Výzkum her se za tímto účelem zabývá například hledáním postačujících podmínek pro existenci vyhrávající strategie stavitele. Jeden z přístupů zavádí jistou modifikaci hry. V ní požadujeme, aby si nejprve ničitel vybral jednu ze struktur a zahrál všechny množinové tahy, potom na to odpoví stavitel ve zbývající

struktuře a následně oba hrají klasickou Ehrenfeucht-Fraïssé hru, ovšem pouze s prvkovými tahy. Takové úpravě se říká *Ajtaiova-Faginova hra* [AF90]. Druhá rodina přístupů zavádí jistou normu, porovnávající podobnost struktur v nějakém „okolí“ prvku, zvanou *Hanfova* nebo *Gaiffmanova lokálnost* [Lib04].

Jistě není bez zajímavosti, že poslední zajímavější (tedy takový, kterému se dostalo dlouhodobé pozornosti odborníků) pokus o důkaz P≠NP od Deolalikara využíval právě metod deskriptivní složitosti. Důkaz za nadějný považovali například Cook nebo Vardi.

V zásadě důkaz používá zmíněných výsledků od Fagina a Immermana (totiž že NP = ∃SO a P = FO(LFP)) a následně se snaží použít výše popsaných (a pokročilejších) technik k důkazu, že FO(LFP) není dostatečná pro popis problému 3SAT. Aktuální stav důkazu je ten, že Immerman poukázal na dvě zásadní chyby, které se Deolalikarovi zatím nepodařilo opravit a Immerman je považuje za neodstranitelné. Přesto je tento pokus hodnocen poměrně kladně, zejména proto, že představuje čerstvý pohled na řešení zmíněného slavného problému.

Více se vztahem deskriptivní složitosti a hierarchie parametrizované složitosti (W hierarchie) zabývá například článek od Downeyho et al. [DFR97]. Zajímavou otázku o možné existenci širší třídy problémů řešitelných na grafech omezené stromové šířky, ale nejspíš nedefinovatelných pomocí MSO, otevírá Kolman et al. ve článku [KLS09].

3 Courcellova věta

Konečně se dostáváme k hlavnímu předmětu práce, Courcellově větě. Pro její důkaz budeme potřebovat lemma, které mluví o vztahu k-typů struktur indukovaných uzlem stromového rozkladu a struktur indukovaných rodičem tohoto uzlu. Umožní nám pak použít znalost k-typů menších struktur k rychlému určení k-typu větších struktur, až nakonec zjistíme k-typ celé vstupní struktury. Definice a lemma pochází z práce Gottloba et al. [GPW10].

Následuje důkaz Courcellovy věty, který je přímou aplikací tohoto lemmatu. Na závěr kapitoly zmíníme několik způsobů, jak algoritmus rozšířit i na kvantifikace přes relace vyšších arit, neboli jak jej rozšířit i pro MSO_k formule, a dále jak algoritmus učinit efektivnějším.

3.1 Kompoziční lemma

Připomeňme, že zkratka $(\mathcal{A}[\mathcal{S}_s], \bar{a})$ značí podstrukturu $\mathcal{A}$ indukovanou prvky kapes podstromu stromového rozkladu zakořeněného v s.

Definice 50. Nechť je $w \geq 1$ a $\mathcal{A}$ a $\mathcal{B}$ jsou τ-struktury. Dále nechť $(a_0, \ldots, a_w)$ a $(b_0, \ldots, b_w)$ jsou dvě $(w+1)$-tice prvků z $\mathcal{A}$ a $\mathcal{B}$.

Dvě $(w+1)$-tice $(a_0, \ldots, a_w)$ a $(b_0, \ldots, b_w)$ nazveme *ekvivalentními* a píšeme $(a_0, \ldots, a_w) \equiv (b_0, \ldots, b_w)$, pokud pro každý relační symbol $R \in \tau$ a všechny α-tice $(i_1, \ldots, i_\alpha) \in \{0, \ldots w\}^\alpha$, kde α je arita R, platí ekvivalence $R^{\mathcal{A}}(a_{i_1}, \ldots, a_{i_w}) \Leftrightarrow R^{\mathcal{B}}(b_{i_1}, \ldots, b_{i_w})$.

Ekvivalentně lze říct, že $((a_0, \ldots, a_w), (b_0, \ldots, b_w))$ je částečným isomorfismem mezi $\mathcal{A}$ a $\mathcal{B}$.

Lemma 51. *Nechť $\mathcal{A}$ a $\mathcal{B}$ jsou τ-struktury, $\mathcal{S}$ a $\mathcal{T}$ jejich normalizované stromové rozklady šířky w a nechť je s vnitřní uzel v $\mathcal{S}$ a t vnitřní uzel v $\mathcal{T}$. Buď k pevné přirozené číslo.*

(1) Permutační uzly. *Nechť je s' jediný syn s v $\mathcal{S}$ a t' jediný syn t v $\mathcal{T}$. Dále označme $\bar{a}$, $\bar{a}'$, $\bar{b}$ a $\bar{b}'$ kapsy uzlů s, s', t a t', v tomto pořadí.*

Pokud $(\mathcal{A}[\mathcal{S}_{s'}], \bar{a}') \equiv_k^{MSO} (\mathcal{B}[\mathcal{T}_{t'}], \bar{b}')$ a existuje permutace π taková, že $\pi(\bar{a}) = \bar{a}'$ a $\pi(\bar{b}) = \bar{b}'$, pak $(\mathcal{A}[\mathcal{S}_s], \bar{a}) \equiv_k^{MSO} (\mathcal{B}[\mathcal{T}_t], \bar{b})$.

(2) Nahrazovací uzly. *Nechť je s' jediný syn s v $\mathcal{S}$ a t' jediný syn t v $\mathcal{T}$. Dále označme $\bar{a} = (a_0, a_1, \ldots, a_w)$, $\bar{a}' = (a'_0, a_1, \ldots, a_w)$, $\bar{b} = (b_0, b_1, \ldots, b_w)$ a $\bar{b}' = (b'_0, b_1, \ldots, b_w)$ kapsy uzlů s, s', t a t', v tomto pořadí.*

Pokud $(\mathcal{A}[\mathcal{S}_{s'}], \bar{a}') \equiv_k^{MSO} (\mathcal{B}[\mathcal{T}_{t'}], \bar{b}')$ a $\bar{a} \equiv \bar{b}$, pak $(\mathcal{A}[\mathcal{S}_s], \bar{a}) \equiv_k^{MSO} (\mathcal{B}[\mathcal{T}_t], \bar{b})$.

(3) Větvící uzly. *Nechť s_1 a s_2 jsou dva synové s v $\mathcal{S}$ a t_1 a t_2 dva synové t v $\mathcal{T}$.*

Pokud $(\mathcal{A}[\mathcal{S}_{s_1}], \bar{a}) \equiv_k^{MSO} (\mathcal{B}[\mathcal{T}_{t_1}], \bar{b})$ a $(\mathcal{A}[\mathcal{S}_{s_2}], \bar{a}) \equiv_k^{MSO} (\mathcal{B}[\mathcal{T}_{t_2}], \bar{b})$, pak $(\mathcal{A}[\mathcal{S}_s], \bar{a}) \equiv_k^{MSO} (\mathcal{B}[\mathcal{T}_t], \bar{b})$.

Důkaz. Důkaz lemmatu provedeme podáním vyhrávající strategie stavitele na větší struktuře z předpokladu existence vyhrávající strategie na menších strukturách. Poznamenejme, že se důkaz nese v duchu *kompozičních vět*, jako je například Fefermanova-Vaughtova věta [Mak04]. Ve článku [GPW10], ze kterého lemmata vycházejí, se jejich důkaz nenachází. Postupujeme ale obdobně jako například Frick ve své disertaci [Fri01].

Upozorněme také, že všechny indukované struktury uzlu s (a dalších), se kterými pracujeme, obsahují konstanty určující pořadí prvků v kapse A_s.

V případech prvních dvou typů uzlů stromového rozkladu (tedy uzlů permutačních a nahrazovacích) postupujeme obecně následovně. Podle předpokladu a podle Ehrenfeuchtovy-Fraïssého věty existuje vyhrávající strategie stavitele ve hře $H' = \mathrm{EFG}((\mathcal{A}[\mathcal{S}_{s'}], \bar{a}'), (\mathcal{B}[\mathcal{T}_{t'}], \bar{b}'), k)$. Popíšeme vyhrávající strategii pro stavitele ve hře $H = \mathrm{EFG}((\mathcal{A}[\mathcal{S}_s], \bar{a}), (\mathcal{B}[\mathcal{T}_t], \bar{b}), k)$, což opět podle Ehrenfeuchtovy-Fraïssého věty stačí k důkazu této části lemmatu. Vyhrávající strategie stavitele pro H bude krok po kroku napodobovat vyhrávající strategii stavitele v H'.

V každém kroku rozlišíme dva základní případy, totiž zda ničitel hraje prvkový nebo množinový tah. Zabýváme se pouze tahy do indukované podstruktury struktury $\mathcal{A}$; pro tahy do indukované podstruktury v $\mathcal{B}$ najdeme symetricky.

(1) *permutační uzly.* Tento případ je nejjednodušší. $(\mathcal{A}[\mathcal{S}_{s'}], \bar{a}')$ a $(\mathcal{A}[\mathcal{S}_s], \bar{a})$ (resp. $(\mathcal{B}[\mathcal{T}_{t'}], \bar{b}')$ a $(\mathcal{B}[\mathcal{T}_t], \bar{b})$) mají stejné domény, tedy posunem od s' k s (resp. t' k t) se nepotkáme s novými prvky.

Uvažme nejprve ničitelův prvkový tah $a \in A[\mathcal{S}_s]$. Pokud $a \notin A_s$, vybereme b podle stavitelovy strategie v $(\mathcal{B}[\mathcal{T}_{t'}], \bar{b}')$. Naopak je-li $a \in A_s$, je to nějaké $a_i \in \{a_0, \ldots, a_w\}$. Pak odpovíme b_i.

Dále uvažme ničitelův množinový tah $A \subseteq A[\mathcal{S}_s]$. A rozdělíme na dvě disjunktní části $A^+ \uplus \bar{A}_s = A$, kde $A^+ = A \cap \{a_0, \ldots, a_w\}$ a $\bar{A}_s = A \setminus A^+$. Pak odpovíme množinou $B = B^+ \uplus \bar{B}_t$, kde $\bar{B}_t$ je stavitelova odpověď na $\bar{A}_s$ v $(\mathcal{B}[\mathcal{T}_{t'}], \bar{b}')$, a $B^+ = \{b_i | a_i \in A^+\}$.

V $(\mathcal{A}[\mathcal{S}_s], \bar{a})$ se nenacházejí žádné nové prvky nebo relace oproti $(\mathcal{A}[\mathcal{S}_{s'}], \bar{a}')$. Tento výběr tedy musí být částečným isomorfismem, byl-li jím výběr v původní $(\mathcal{A}[\mathcal{S}_{s'}], \bar{a}')$. Tím jsme ukázali stavitelovu odpověď v $(\mathcal{B}[\mathcal{T}_t], \bar{b})$ na ničitelovy tahy do $(\mathcal{A}[\mathcal{S}_s], \bar{a})$.

(2) *nahrazovací uzly.* V $(\mathcal{A}[\mathcal{S}_s], \bar{a})$ se vyskytuje nový prvek a_0. Pro ničitelův prvkový tah $a \in A[\mathcal{S}_s]$ vybereme buď b dle stavitelovy strategie v $(\mathcal{B}[\mathcal{T}_{t'}], \bar{b}')$, pokud $a \neq a_0$, nebo b_0, pokud $a = a_0$.

Pro množinový tah $A \subseteq A[\mathcal{S}_s]$ vybereme buď B dle stavitelovy strategie v $(\mathcal{B}[\mathcal{T}_{t'}], \bar{b}')$, pokud $a_0 \notin A$, nebo vybereme $B' \cup b_0$ pro B' stavitelovu odpověď na $A' = A \setminus a_0$ v $(\mathcal{B}[\mathcal{T}_{t'}], \bar{b}')$.

Protože je $A_{s'}$ separátor, víme z vlastností stromového rozkladu, že a_0 je v $(\mathcal{A}[\mathcal{S}_s], \bar{a})$ v relaci jen s nějakými prvky z A_s. Navíc předpoklad $\bar{a} \equiv \bar{b}$, říká, že na prvcích A_s se relace z τ chovají stejně, jako na prvcích B_t. Z toho lze vidět, že výše zvolené volby a a A zachovají částečný isomorfismus a máme dobrou odpověď.

(3) *větvící uzly.* Tento případ se od předchozích poněkud liší. Předpokladem jsou opět podle Ehrenfeuchtovy-Fraïssého věty vyhrávající strategie stavitele ve hrách $H_1 = \mathrm{EFG}((\mathcal{A}[\mathcal{S}_{s_1}], \bar{a}), (\mathcal{B}[\mathcal{T}_{t_1}], \bar{b}), k)$ a $H_2 = \mathrm{EFG}((\mathcal{A}[\mathcal{S}_{s_2}], \bar{a}), (\mathcal{B}[\mathcal{T}_{t_2}], \bar{b}), k)$. Opět popíšeme vyhrávající strategii stavitele pro hru $H = \mathrm{EFG}((\mathcal{A}[\mathcal{S}_s], \bar{a}), (\mathcal{B}[\mathcal{T}_t], \bar{b}), k)$ a tím podle Ehrenfeuchtovy-Fraïssého věty dokážeme tuto část lemmatu.

Tento případ se liší od předchozích tak, že se musíme zabývat dvěmi podstrukturami místo jedné. Nalézt odpovídající stavitelův tah by proto mohlo být obtížné, ale díky vlastnostem stromového rozkladu a přítomnosti prvků kapes jako konstant v podstrukturách synů to je možné.

Pro ničitelův prvkový tah $a \in A[\mathcal{S}_s]$ rozlišujeme tři případy. Pokud $a \in A[\mathcal{S}_{s_1}]$ a zároveň $a \notin A[\mathcal{S}_{s_2}]$, vybereme jako b stavitelovu odpověď v $(\mathcal{B}[\mathcal{T}_{t_1}], \bar{b})$. Pokud naopak $a \in A[\mathcal{S}_{s_2}]$ a $a \notin A[\mathcal{S}_{s_1}]$, vybereme odpověď v $(\mathcal{B}[\mathcal{T}_{t_2}], \bar{b})$. Třetí možnost je, že $a \in (A[\mathcal{S}_{s_1}] \cap A[\mathcal{S}_{s_2}])$, neboli, protože A_s je separátor, $a = a_i \in \{a_0, \ldots, a_w\}$. Pak zvolíme odpovídající b_i.

Pro $A \subseteq A[\mathcal{S}_s]$ množinový tah nejprve rozdělíme A na tři disjunktní množiny $A_1 \uplus A_2 \uplus A^+ = A$, kde $A_1 = \{a \in A | a \in A[\mathcal{S}_{s_1}] \wedge a \notin A[\mathcal{S}_{s_2}]\}$, A_2 definujeme obdobně a $A^+ = A \cap \{a_0, \ldots, a_w\}$ (rovnost opět platí díky tomu, že A_s je separátor). Odpověď B zvolíme jako $B = B_1 \uplus B_2 \uplus B^+$ pro B_1 stavitelovu odpověď na A_1 v $(\mathcal{A}[\mathcal{S}_{s_1}], \bar{a})$, B_2 stavitelovu odpověď na A_2 v $(\mathcal{A}[\mathcal{S}_{s_2}], \bar{a})$ a $B^+ = \{b_i | a_i \in A^+\}$.

U prvkových tahů je korektnost volby b pro $a \notin A^+$ zřejmá z předpokladu. Aby fungovala i volba b pro $a \in A^+$, musíme ukázat, že obě strategie (pro $(\mathcal{B}[\mathcal{T}_{t_1}], \bar{b})$ i $(\mathcal{B}[\mathcal{T}_{t_2}], \bar{b})$) se na ní shodnou. To ale plyne z toho, že obě tyto struktury obsahují prvky $\bar{b}$ jako konstanty, a proto se na nich musí shodovat i všechny relace.

Skutečnost, že B_1 a B_2 dobře zrcadlí volbu A_1 a A_2, plyne opět z předpokladu. Stavitelovy strategie pro $(\mathcal{B}[\mathcal{T}_{t_1}], \bar{b})$ a $(\mathcal{B}[\mathcal{T}_{t_2}], \bar{b})$ se shodnou i na volbě B^+ proto, že jsou $\bar{b}$ konstantami v obou strukturách. □

3.2 Courcellova věta a algoritmus

Courcellova věta je přímou aplikací předchozího lemmatu. Vyslovíme a dokážeme ji v základní podobě (pro rozhodovací problémy, tedy problémy popsané sentencemi). V následující kapitole budeme diskutovat možná rozšíření. Nejprve v návaznosti na Definici 50 (o elementární ekvivalenci $(w+1)$-tic) zavedeme *normalizaci kapsy*, kterou budeme potřebovat v algoritmu.

Definice 52. Pro strukturu $\mathcal{A}[\bar{a}]$ indukovanou kapsou $\bar{a} = (a_0, \ldots, a_w)$ myslíme *normalizací kapsy* $\bar{a}$ strukturu získanou z $\mathcal{A}[\bar{a}]$ nahrazením všech výskytů prvků v relacích čísly od 0 do w dle pozice prvku v kapse. Značíme $norm(\mathcal{A}[\bar{a}])$.

Ihned je vidět, že pro dvě kapsy $\bar{a}$ a $\bar{b}$ platí $norm(\mathcal{A}[\bar{a}]) = norm(\mathcal{A}[\bar{b}])$ právě tehdy, když $\bar{a} \equiv \bar{b}$ dle Definice 50.

Věta 53. *Rozhodovací problém definovaný MSO sentencí φ nad τ-strukturami stromové šířky nejvýše w lze vyřešit pro strukturu $\mathcal{A}$ v čase $\mathcal{O}(f(|\varphi|, w) \cdot |\mathcal{A}|)$ pro nějakou funkci $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$.*

Důkaz. Dokážeme předvedením algoritmu. Na vstupu předpokládáme MSO formuli φ nad abecedou τ kvantifikátorové hloubky k, τ-strukturu $\mathcal{A}$ a její normalizovaný stromový rozklad $\mathcal{T}$. Ten lze najít použitím postupu ze článku [Bod96] a následnou aplikací tvrzení o normalizovaném rozkladu (Tvrzení 19).

Algoritmus prochází rozklad od listů ke kořeni a vyhodnocuje k-typy všech podstruktur indukovaných jednotlivými uzly. Zkráceně budeme tedy někdy hovořit o k-typu uzlu. Pro každý tento k-typ si algoritmus zapamatuje první nalezenou strukturu tohoto typu jakožto jeho *svědka*. Navíc si všímá druhů přechodů od syna (synů) k otci (analogicky k předchozímu lemmatu) a nalezené přechody ukládá do tabulek pro pozdější použití. Používá následující datové struktury:

- *types* je kladné celé číslo. Je to počítadlo nalezených k-typů. Umožní každému k-typu přiřadit přirozené číslo, dosud nepřiřazené žádnému z k-typů.
- Každý uzel t má následující atributy, z nichž některé mohou být prázdné; pak řekneme, že se rovnají hodnotě `null`.
  - $t.bag$ je kapsa uzlu t
  - $t.type$ je číslo označující k-typ struktury $(\mathcal{A}[\mathcal{T}_t], t.bag)$
  - $t.child$ je syn uzlu t, má-li t jediného syna (tedy je-li t permutační nebo nahrazovací uzel)
  - $t.lchild$ a $t.rchild$ jsou pravý a levý syn uzlu t, má-li t dva syny (tedy je-li t větvící uzel)
- W je tabulka svědků nalezených k-typů. Pro $i \leq types$ je $W[i]$ nějaká struktura $(\mathcal{A}[\mathcal{T}_t], \bar{a})$, o níž víme, že je k-typu i.
- P je tabulka nalezených přechodů od syna, jehož k-typ je i, k otci, když je kapsa syna permutací π kapsy otce. Pak je $P(i, \pi)$ číslo označující k-typ otce.
- R je tabulka nalezených přechodů od syna, jehož k-typ je i, k otci, když se kapsa syna liší od kapsy otce pouze na první pozici a $\mathcal{A}[\bar{a}]$ je struktura indukovaná kapsou otce. Pak je $R(i, norm(\mathcal{A}[\bar{a}]))$ číslo označující k-typ otce.
- B je tabulka nalezených přechodů od synů, jejichž k-typy jsou čísla i a j, k otci s kapsou identickou kapsám synů. Pak je $B(i, j)$ číslo označující k-typ otce.

Datové struktury W, P, R a B budeme implementovat například jako hashovací tabulky, způsob implementace ovlivní jen konstantu schovanou v $\mathcal{O}(f(|\varphi|, w))$.

Připomeňme, že relaci $\equiv_k^{MSO}$ lze testovat simulací Ehrenfeucht-Fraïssé hry pro dané dvě struktury. Relaci $\mathcal{A} \models \varphi$ lze testovat simulací příslušné Hintikka hry $\mathrm{MCG}(\mathcal{A}, \varphi)$. Pro účely algoritmu nechť představuje výraz $\mathrm{MCG}(\mathcal{A}, \varphi)$ funkci, která vrací *true* v případě, že stavitel má vyhrávající strategii v Hintikka hře $\mathrm{MCG}(\mathcal{A}, \varphi)$, a která vrací *false* v opačném případě.

Algoritmus decide($\mathcal{A}$, $\mathcal{T}$, φ)

Vstup: $\mathcal{A}$ relační τ-struktura, $\mathcal{T}$ její stromový rozklad šířky w a t jeho kořen, φ MSO formule kvantifikátorové hloubky k

Výstup: **true**, pokud $\mathcal{A} \models \varphi$, **false**, pokud $\mathcal{A} \not\models \varphi$.

recursive_label(t)

return $\mathrm{MCG}(W[t.type], \varphi)$

Procedura recursive_label(t)

Vstup: t uzel stromového rozkladu $\mathcal{T}$
Výstup (nepřímý): Nastaví u uzlu t a všech uzlů jeho podstromu atribut *.type* odpovídající k-typu struktur jim odpovídajících.

```
if t je list then
  bruteforce(t)
else if t je permutační uzel then
  recursive_label(t.child)
  if P(t.child.type, π) ≠ null then
    t.type ← P(t.child.type, π)
  else
    bruteforce(t)
    P(t.child.type, π) ← t.type
  end if
else if t je nahrazovací uzel then
  recursive_label(t.child)
  if R(t.child.type, A[t.bag]) ≠ null then
    t.type ← R(t.child.type, norm(A[t.bag]))
  else
    bruteforce(t)
    R(t.child.type, norm(A[t.bag])) ← t.type
  end if
else if t je větvící uzel then
  recursive_label(t.lchild)
  recursive_label(t.rchild)
  if B(t.lchild.type, t.rchild.type) ≠ null then
    t.type ← B(t.lchild.type, t.rchild.type)
  else
    bruteforce(t)
    B(t.lchild.type, t.rchild.type) ← t.type
  end if
end if
```

Procedura bruteforce(t)

Vstup: t uzel stromového rozkladu $\mathcal{T}$
Výstup (nepřímý): Nastaví do atributu $t.type$ k-typ struktury $(\mathcal{A}[\mathcal{T}_t], \bar{a})$.

```
for i = 1 → types do
  if (A[T_t], ā) ≡_k^MSO W[i] then
    t.type ← i
    end
  end if
end for{S} k-typem uzlu t jsme se ještě nesetkali; zavedeme jej.
types ← types + 1
t.type ← types
```

Procedura decide($\mathcal{A}, \mathcal{T}, \varphi$) rozhodne formuli φ nad $\mathcal{A}$ v lineárním čase s konstantou závislou pouze na stromové šířce $tw(\mathcal{A})$ a kvantifikátorové hloubce formule $k = qr(\varphi)$. To nahlédneme takto. Nechť $T = |\mathrm{MSO}[k]|$, neboli T je index relace $\equiv_k^{MSO}$. Jistě T závisí pouze na k. Jak jsou omezené velikosti tabulek W, P, R a B?

- W obsahuje nejvýše T položek.
- P obsahuje nejvýše $T \cdot (w+1)!$ položek, pro $(w+1)!$ představující počet různých permutací π u permutačního uzlu.
- R obsahuje nejvýše $T \cdot |\mathcal{R}|$ položek, pro $|\mathcal{R}| = \prod_{R_i \in \tau} 2^{(w+1)^{\alpha_i}}$ počet všech možných realizací relací z τ na $w+1$ prvcích představujících kapsu nahrazovacího uzlu.
- B obsahuje nejvýše T^2 položek, jednu za každou dvojici k-typů setkávajících se u větvícího uzlu.

Zřejmě tedy jejich velikosti nezávisí žádným způsobem na velikosti vstupní struktury $|\mathcal{A}|$. Dále čas simulace Ehrenfeucht-Fraïssé hry při testování ekvivalence dvou struktur $\equiv_k^{MSO}$ zřejmě závisí jen na k a velikostech testovaných struktur.

Představme si, že jsou již tabulky naplněné informacemi o všech k-typech a o všech možných přechodech mezi uzly. To lze zajistit například induktivní konstrukcí všech možných k-typů a přechodů, jak to dělá Gottlob et al. [GPW10]. Protože ještě $\mathcal{A}$ neznáme, můžeme považovat takto strávený čas za „přípravu“, která trvá jen konstantně dlouho.

Jádrem algoritmu je procedura *recursive_label*. Ta vykonává následující:

- Vyhodnotí k-typy všech listů. Jak listy, tak všechny struktury $W[i]$, vůči kterým jsou listy testovány, jsou svou velikostí nezávislé na $|\mathcal{A}|$. Listů je nejvýše lineárně vzhledem k $|\mathcal{A}|$ a toto vyhodnocení tedy celkově zabere čas $\mathcal{O}((f(|\varphi|, w)) \cdot |\mathcal{A}|)$.
- Vyhodnotí k-typy všech vnitřních uzlů z předpočítaných tabulek. Uzlů je též nejvýše lineárně a každý vyhodnotíme pouhým nahlédnutím do tabulky, takže dohromady to zabere čas $\mathcal{O}((f(|\varphi|, w)) \cdot |\mathcal{A}|)$.

Dohromady $2 \cdot \mathcal{O}((f(|\varphi|, w)) * |\mathcal{A}|) = \mathcal{O}((f(|\varphi|, w)) * |\mathcal{A}|)$. My sice tabulky předpočítané nemáme, ale všechen čas strávený na jejich naplnění v době běhu algoritmu můžeme započítat do zmíněné "přípravy", a tak jej schovat do konstanty. Proto běží v čase $\mathcal{O}((f(|\varphi|, w)) \cdot |\mathcal{A}|)$ i popisovaná procedura *recursive_label*.

V praxi žádnou „přípravu“ neprovádíme; takový postup by byl zbytečným plýtváním a popsaný algoritmus bude mít ve svých tabulkách jen ty informace, které skutečně potřebuje. Pro účely nahlédnutí horního odhadu složitosti algoritmu se ale představa předpočítaných tabulek hodí.

Druhým krokem algoritmu *decide* je test relace $W[t.type] \models \varphi$ (pro t kořen stromového rozkladu struktury $\mathcal{A}$) pomocí Hintikka hry $\mathrm{MCG}(W[t.type], \varphi)$. Jak už jsme nahlédli výše, velikost všech struktur $W[i]$ nezávisí na velikosti $\mathcal{A}$, a proto vyhodnocení dané Hintikka hry trvá pouze čas $\mathcal{O}(f(|\varphi|, w))$. □

Tímto jsme podali samostatný důkaz Courcellovy věty.

3.3 Rozšíření a dodatky

Popis relace $\equiv_k^{MSO}$ pomocí Ehrenfeucht-Fraïssé her se ukázal být velmi praktickým v důkazu Lemmatu 51 a jak ukážeme dále, umožňuje i snadno nahlédnout některé další vlastnosti našeho algoritmu, potenciální optimalizace a rozšíření.

3.3.1 Optimalizační varianty a kvantifikace v MSO_k

Jak už jsme zmínili, neexistuje MSO_1 formule rozhodující hamiltonicitu grafu [EF95]. Přesto je známým faktem, že tento problém lze nad grafy omezené stromové šířky efektivně rozhodnout. Proto nás nemusí překvapit, že již rok po publikaci původní Courcellovy věty, která umožňovala pouze monadické kvantifikace a mluvila o rozhodovacích problémech, vyšel článek od Arnborga et al. [ALS91] zobecňující ji na kvantifikátory vyšších arit, vyhodnocovací a dokonce i optimalizační problémy. Vyhodnocovací problém je zadán formulí s volnými proměnnými a jako výstup požadujeme nějaké její splňující ohodnocení, pokud existuje; optimalizační problém minimalizuje či maximalizuje funkce velikostí množin, které jsou volnými proměnnými formule. Pomocí tohoto výsledku lze díky formuli popisující hamiltonicitu grafu, kterou jsme uvedli jako příklad MSO_2 formule (Příklad 36), rozhodnout i tento problém přes Courcellovu větu.

Přístup přes Ehrenfeucht-Fraïssé hry popsaný v této práci umožňuje snadno nahlédnout rozšíření na MSO_k pro pevné k. Definice Ehrenfeucht-Fraïssé her svými prvkovými a množinovými tahy reflektovala prvkové a množinové kvantifikátory formule. Zabýváme-li se tedy formulemi, které kvantifikují i přes relace a jejich podmnožiny, musíme pro ně definici her rozšířit, což znamená také rozšířit definici částečného isomorfismu. Nebudeme zacházet do technických detailů, ale uvedeme příklad.

Chceme-li pro grafy umožnit kvantifikaci přes hrany (jak činí např. formule rozhodující hamiltonicitu), přibudou do Ehrenfeucht-Fraïssé hry „hranové“ a „množinově hranové“ tahy. Vybrané vrcholy, hrany a podmnožiny vrcholů a hran v obou grafech pak určují částečný isomorfismus, pokud mimo standardních požadavků:

- hrany prvního grafu se zobrazují na hrany ve druhém grafu tak, že isomorfismus souhlasí i na příslušných vrcholech (jak jejich volbě, tak jejich „barvě“),
- částečný isomorfismus zachovává „barvy“ hran.

Barvami myslíme příslušnosti do vybraných množin vrcholů nebo hran.

3.3.2 Speciální Ehrenfeucht-Fraïssé hry

Tento a následující oddíl mluví o zlepšení faktoru $f(|\varphi|, w)$ ve složitosti algoritmu implementujícího Courcellovu větu. Je na místě podotknout, že existují dolní odhady [FG04], z nichž plyne, že bez ohledu na použitý přístup nelze nikdy získat f elementární v k a w. To v podstatě znamená, že pro libovolný algoritmus implementující Courcellovu větu existuje „zákeřná“ formule φ, pro níž bude f nabývat hodnot věžové funkce.

Mluvíme-li tedy dále o „zlepšování faktoru $f(|\varphi|, w)$“, myslíme tím postupy, které mohou usnadnit sestrojení formule, pro níž bude mít algoritmus složitost blízkou složitosti ručně sestrojených algoritmů pro daný problém. Například Kneis et al. [KLR11] představuje algoritmus, který nabývá pro formuli popisující problém minimálního vrcholového pokrytí stejné složitosti, jako specializovaný algoritmus pro tento problém. Pro problém 3-obarvitelnosti má algoritmus faktor složitosti závislý na w a k roven 9^w, zatímco pro specializovaný algoritmus je to 3^w. I to je velice dobrý výsledek. Tímto přístupem se dále zabýváme v Oddíle 4.2.

Vztah jednotlivých druhů tahů (prvkových a množinových) Ehrenfeucht-Fraïssé hry a kvantifikátorů formule reflektuje následující úvaha. Hry byly původně zavedeny jako nástroj popisující relaci $\equiv_k^{FO}$ a obsahovaly jen prvkové tahy, odpovídající prvkovým kvantifikátorům formule. Naše aktuální definice povoluje druhy tahů v libovolném pořadí střídat, stejně jako se mohou kvantifikátory (prvkové a množinové) libovolně střídat v MSO formuli.

Uvažme existenční monadické formule, to jest formule MSO, které začínají posloupností k množinových existenčních kvantifikátorů, za nimiž následuje FO formule hloubky r. Pro tuto třídu formulí lze analogicky k ekvivalenci $\equiv_k^{MSO}$ a MSO[k]-typu definovat ekvivalenci $\equiv_{k,r}^{\exists\mathrm{MSO}}$ a $\exists$MSO[k]-FO[r]-typ. Všimněme si, že všechny takové formule jsou jistě i v MSO[$k + r$]. Hra charakterizující tuto ekvivalenci se nazývá *Ajtai-Fagin hra* [AF90] a hraje se následovně:

- Ničitel si vybere $\mathcal{A}$ nebo $\mathcal{B}$ libovolně a jeho prvky obarví $c = 2^k$ barvami.
- Stavitel ve druhé struktuře obarví prvky c barvami.
- Oba hráči mezi sebou hrají EFG($\mathcal{A}, \mathcal{B}, r$), v níž ale mohou používat pouze prvkové tahy.

První tahy zřejmě odpovídají výběrům všech k množin ze začátku formule – barva při vrcholu vyjadřuje jeho příslušnost do těchto množin. Navazující hra EFG($\mathcal{A}, \mathcal{B}, r$) omezená na prvkové tahy odpovídá FO podformuli v druhé části formule.

Kdybychom měli jistotu, že na vstupu našeho algoritmu bude právě $\exists$MSO[k]-FO[r] formule, mohli bychom místo EFG($\mathcal{A}, \mathcal{B}, k+r$) v algoritmu použít pro zjištění k-typu výše popsanou Ajtai-Fagin hru. Je zřejmé, že vyhodnocení takové hry je podstatně efektivnější, než vyhodnocení EFG($\mathcal{A}, \mathcal{B}, k + r$). Navíc MSO[$k + r$]-typů je jistě mnohem více, než $\exists$MSO[k]-FO[r]-typů. Právě počet typů ale určuje zásadním způsobem faktor $f(|\varphi|, w)$ časové složitosti algoritmu a proto je žádoucí se jej snažit co nejvíce snížit.

Nemohli bychom tedy i my brát v potaz pořadí a druh kvantifikátorů vstupní formule a tím zlepšit i faktor $f(|\varphi|, w)$ složitosti algoritmu? Záměrem této práce není popsat všechny technické detaily, které by takový postup zahrnoval, ale podáme alespoň přibližný nástin toho, co by obnášel.

Buď φ MSO formule nad abecedou τ popisující nějakou vlastnost τ-struktur. Převeďme φ na ekvivalentní formuli ψ, která je v prenexním tvaru. Pak $\psi \equiv \bar{Q}_1\bar{q}_1\bar{Q}_2\bar{q}_2 \ldots \bar{Q}_n\bar{q}_n\psi_{atom}(\bar{x}, \bar{X})$, kde $\bar{Q}_i$ jsou nějaké posloupnosti množinových kvantifikátorů (nerozlišujeme obecné a existenční kvantifikátory), $\bar{q}_i$ jsou posloupnosti prvkových kvantifikátorů a $\psi_{atom}(\bar{x}, \bar{X})$ je atomická podformule s volnými prvkovými proměnnými $\bar{x}$, které byly vybrány kvantifikátory $(\bar{q}_i)_{i=1}^n$, a volnými množinovými proměnnými $\bar{X}$, které byly vybrány kvantifikátory $(\bar{Q}_i)_{i=1}^n$. Délka $\bar{Q}_1$ a $\bar{q}_n$

může být nulová podle toho, jakým kvantifikátorem začíná a končí kvantifikátorová část formule.

Informaci o pořadí a druzích kvantifikátorů můžeme zakódovat do $2n$-tice čísel $(P_1, p_1, P_2, p_2, \ldots, P_n, p_n)$ pro P_i délku $\bar{Q}_i$ a p_i délku $\bar{q}_i$, zkráceně tuto $2n$-tici označujme $\bar{\mathcal{P}}$. Můžeme říci, že ψ pochází z množiny formulí MSO$[\bar{\mathcal{P}}]$ (označující právě formule s tímto typem střídání kvantifikátorů) a příslušným způsobem také zavést ekvivalenci $\equiv_{\bar{\mathcal{P}}}^{\mathrm{MSO}}$. Tuto ekvivalenci charakterizuje jistá modifikace Ehrenfeucht-Fraïssé hry – hru upravíme tak, že prvních P_1 tahů je množinových, dalších p_1 tahů je prvkových a tak dále, až posledních p_n tahů je prvkových. Tuto hru označujme EFG$(\mathcal{A}, \mathcal{B}, \bar{\mathcal{P}})$.

Pak lze v našem algoritmu v proceduře *bruteforce* nahradit test $\equiv_k^{MSO}$ testem relace $\equiv_{\bar{\mathcal{P}}}^{\mathrm{MSO}}$. Simulace jí příslušné hry je efektivnější a k-typů této relace je jistě méně, než původní relace $\equiv_k^{MSO}$, už jen proto, že MSO$[\bar{\mathcal{P}}] \subsetneq$ MSO$[k]$ pro $k = \sum_{i=1}^{n}(P_i + p_i)$.

3.3.3 Zobecněné kvantifikátory

Vraťme se ještě k formuli pro 3-obarvitelnost grafu (Příklad 32). Formule začíná trojicí existenčních množinových kvantifikátorů a o množinách jimi vybraných pak pomocí dvou FO formulí řekne, že tvoří rozklad množiny vrcholů (formule Partition(R, G, B)), a že jsou nezávislé. Když si představíme vyhodnocování této formule hrubou silou, lze říct, že postupně zkouší všechny možné trojice podmnožin vrcholů a testuje, zda tvoří rozklad a jsou nezávislé. Místo toho by mohla rovnou zkoušet jen všechny trojice podmnožin vrcholů, které jsou rozklady – těch je totiž o trochu méně (3^n oproti 2^{3n}) – a až pro ty testovat, zda jsou nezávislé. To by vyžadovalo do jazyka zavést jakýsi „zobecněný kvantifikátor“.

Jak se ukazuje, *zobecněné kvantifikátory* (*generalized quantifiers*) [Wes11] nepředstavují nijak novou myšlenku. Zjednodušeně (a z pohledu informatiky) si můžeme *zobecněný kvantifikátor pro formuli* $\varphi(\bar{x}, \bar{X})$ představovat jako „iterátor“ přes splňující ohodnocení této formule. V příkladu formule 3-obarvitelnosti grafu by to znamenalo „iterovat“ přes splňující ohodnocení formule Partition(R, G, B).

Rozhodně je mimo rozsah této práce se zobecněnými kvantifikátory zabývat blíže. Za pozornost ale stojí, že jejich použitím v metaalgoritmických větách (jako je Courcellova věta) se nikdo, zdá se, nezabývá. Navíc rozšířit současný přístup přes Ehrenfeucht-Fraïssé hry na zobecněné kvantifikátory definovatelné MSO formulemi by nemělo být příliš komplikované, a přineslo by to jisté zlepšení faktoru $f(|\varphi|, w)$ ve složitosti algoritmu, stejně jako lepší popisnou schopnost jazyka, v němž vlastnosti struktur definujeme.

Ani toto rozšíření nám samozřejmě neumožní „obejít“ známé dolní odhady [FG04].

4 Kam dál

V této kapitole nastíníme několik zajímavých současných výsledků souvisejících s Courcellovou větou a jejím praktickým využitím.

4.1 Datalog a logické programování

Původním hlavním záměrem práce byla implementace algoritmu, který dává důkaz Courcellovy věty, dle postupu Gottloba et al. v článku [GPW10]. Při bližším pohledu vychází najevo, že záměr článku je poněkud jiný, než nalezení efektivního obecného algoritmu. Autoři v něm ukazují, že jistá podmnožina databázového dotazovacího jazyka Datalog [CGT90, AHV95] (který je sám syntakticky podmnožinou jazyka Prolog, ale jejich sémantika se liší) umožňuje sestavení efektivních programů rozhodujících určité těžké problémy nad strukturami s omezenou stromovou šířkou. Toto pozorování pak zobecňují ve stylu Courcellovy věty: uvedou konstrukci, která pro každou zadanou MSO sentenci φ a dané w sestaví datalogový program, jenž φ rozhodne nad strukturou stromové šířky w v čase lineárním ve velikosti $\mathcal{A}$ a délce programu $\mathcal{P}$ pro pevné $k = qr(\varphi)$.

Článek pokračuje *ruční* konstrukcí datalogového programu, který rozhoduje 3-obarvitelnost grafu. Tento program sice opravdu řeší daný problém efektivně, je ale *ručně* zkonstruovaný a nese velké množství optimalizací, které všeobecný postup, popsaný dříve ve článku, nedokáže udělat, a bez nichž by výsledný program nevykazoval zdaleka tak dobré výsledky.

Autoři v úvodu článku zmiňují některé výhody řešení těžkých problémů pomocí Datalogu. Přestože není jejich obecný postup v praxi použitelný, v mnoha ohledech poskytují užitený vhled do problematiky a cesta převodu formule na logický program je jistě lákavá. Mezi výhody tohoto přístupu patří:

- *Popisnost logických programů.* Logika MSO dovoluje zapisovat složité vztahy velice stručným způsobem, ale její sémantika se nehodí pro praktické použití. Logické programy zachovávají popisnost původní formule, ale zároveň umožňují efektivní vyhodnocení.

- *Existující optimalizace.* Software používaný pro vyhodnocení logických programů (ať už mluvíme o Datalogu, nebo jiném obecnějším jazyku) má za sebou dlouhý vývoj, spoustu výzkumu a obecné optimalizace, ze kterých každý zkonstruovaný program těží.

- *Stručnost.* Obvyklý postup důkazu Courcellovy věty, totiž konstrukce tzv. stromového automatu, má za výsledek program exponenciální velikosti vzhledem k velikosti formule. V logickém programovacím jazyce lze často zkon-

struovat malý program, který složité větvení přenechává vyhodnocujícímu software.

Výzkum datalogu navíc v posledních letech prožívá jisté oživení v souvislosti s uvedením rozšíření Datalog$^{\pm}$ [CGL^{+}10]. Proto se domníváme, že další výzkum vztahu mezi logickým programováním a logickými jazyky by mohl přinést ovoce i pro praktické implementace Courcellovy věty.

4.2 Rozšířené Hintikka hry

V roce 2009 vyšel zajímavý článek Langera a Kneise s názvem „A Practical Approach to Courcelle's Theorem" [KL09]. V tomto článku autoři dokazují optimalizační variantu Courcellovy věty, řešící problémy typu $\mathrm{opt}|U| : U \subseteq V\varphi(U)$ pro $\mathrm{opt} \in \{\min, \max\}$, $\varphi \in \mathrm{FO}$ a V množinu vrcholů grafu. I takto omezený jazyk stačí pro popis mnoha těžkých optimalizačních problémů, jako je nalezení minimálního vrcholového pokrytí, minimální dominující množiny nebo maximální nezávislé množiny grafu.

Přístup omezení jazyka, kterým se problém popisuje, je zajímavý sám o sobě. MSO je mocný jazyk, ale právě kvůli tomu má dodnes Courcellova věta spíše jen teoretický význam. Již v předchozí kapitole jsme viděli, jak může přidání dodatečných informací do formule (například pomocí zobecněného kvantifikátoru) pomoci při jejím vyhodnocování.

Článek dále obsahuje některá zajímavá pozorování o Ehrenfeucht-Fraïssé hrách, která umožňují dále optimalizovat jejich simulaci, a v závěru článku se dozvídáme, že experimentální výsledky potvrzují jistý potenciál tohoto postupu.

Autoři ve své snaze nalézt efektivní přístup k plné Courcellově větě nepolevili a v nedávné době spolu s Rossmanithem uvedli článek „Courcelle's Theorem – A Game-Theoretic Approach" [KLR11]. V tomto článku skutečně podávají alternativní důkaz Courcellovy věty pro LinMSO problémy, neboli problémy popsatelné jako optimalizace lineární funkce velikostí množin MSO formule.

Zásadní změnou proti předchozímu článku ale je přesun pozornosti od Ehrenfeucht-Fraïssé her k Hintikka hrám. Autoři zavádí *rozšířenou Hintikka hru*, která se hraje nad indukovanými podstrukturami a proto nemá kompletní informaci o celé struktuře. Setkáváme se tedy poprvé s hrou, která obsahuje remízové pozice. Autoři ukazují, jak z částečných řešení těchto her pro menší struktury lze kompozicí získat částečná řešení pro větší struktury, až získáme řešení pro celou strukturu.

Výhodou použití Hintikka her je větší blízkost původní formuli a tím pádem i vlastnosti, kterou popisuje. Zatímco náš důkaz z celé formule použil pouze informaci o její kvantifikátorové hloubce (a případně o střídání prvkových a množinových kvantifikátorů), přístup používající Hintikka hry blízce kopíruje standardní vyhodnocování formule, ale úspěšně přitom využívá i znalost jejího stromového rozkladu malé šířky.

Navíc v našem přístupu přes Ehrenfeucht-Fraïssé hry není zřejmé, jak rozšířit kompoziční lemma způsobem, který by umožňoval nejen formuli rozhodnout, ale i nalézt její splňující ohodnocení. I tento problém se autorům podařilo překonat. Uvážíme-li, že odhady složitostí běhu obecného algoritmu pro formule popisující konkrétní problémy se často neliší (nebo ne příliš) od složitostí specializovaných algoritmů a experimentální výsledky jsou přesvědčivé, je asi zřejmé, proč se tento přístup zdá být velmi nadějným.

4.3 Jiné typy rozkladů

Jak jsme již zmínili, stromový rozklad a stromová šířka byly zavedeny při práci na *Graph Minor Project* Robertsona se Seymourem [RS84, RS86]. V rámci stejného projektu se objevilo několik dalších „šířek“ a rozkladů, jako je *cestová šířka* (*pathwidth*) a *větvící šířka* (*branchwidth*). Hledání vhodných rozkladů se tím samozřejmě nezastavilo. Hlavním problémem stromového rozkladu a rozkladů výše zmíněných je především jejich nepoužitelnost pro husté grafy.

V roce 2000 přišel Courcell a Olariu [CO00] s novým typem rozkladu a šířky, *klikovou šířkou* (*cliquewidth*), které jsou vhodné právě pro použití s hustými grafy. Navíc téhož roku dokázal Courcelle et al. [CMR00] metaalgoritmickou větu pro grafy omezené klikové šířky obdobnou té, která je předmětem naší práce. Klikovou šířku zmiňují i autoři článku diskutovaného v předchozím oddíle [KLR11]. Tvrdí, že popisovaný algoritmus by byl podstatně jednodušší pro klikové rozklady a hlavní praktickou překážou pro jejich širší použití je obtížnost hledání klikových rozkladů potřebné (omezené) šířky.

Ještě jiným druhem šířky je *ranková šířka* (*rankwidth*), která je téměř ekvivalentní klikové šířce (struktura má omezenou rankovou šířku, právě když má omezenou klikovou šířku) a poprvé ji zavedl Oum a Seymour [iOS06]. Protože je klikové šířce téměř ekvivalentní, vztahuje se na ni i výše zmíněný metaalgoritmický výsledek pro klikovou šířku [CMR00]. Přímo pro rankový rozklad tentýž výsledek metodou rozšířených Hintikka her dokázal Langer et al. [LRS11].

Poslední typ šířky, který zmíníme, pochází opět od Courcella [Cou10]. Je jím *speciální stromová šířka* (*special treewidth*), jejíž definice se v prvních dvou bodech shoduje s definicí standardní stromové šířky, ovšem poslední bod požaduje, aby uzly rozkladu obsahující konkrétní vrchol indukovaly v rozkladu *cestu*, nikoliv podstrom. Z toho lze vidět, že se jedná o silnější požadavek a autor článku zařazuje tuto šířku mezi cestovou šířku a stromovou šířku. To autorovi umožňuje dokázat metaalgoritmickou větu pro grafy s omezenou speciální stromovou šířkou a ukázat přitom, že konstanta složitosti nevzroste tolik, jako v případě obdobné věty pro standardní stromovou šířku.

5 Implementace

Na úvod komentáře implementace algoritmu popsaného v Kapitole 3 uveďme pohled na „větší obrázek“ vztahu optimalizací a rychlosti algoritmů. Článek [Bix02] se zabývá zrychlením řešení obecných úloh lineárního programování v průběhu 20 let. Autor píše, že celkové zrychlení bylo asi 30,000,000×. Z toho podíl zrychlení hardware za tuto dobu byl asi „jen“ tisícinásobný. Za zbývající zrychlení v řádu desítek tisíc byly zodpovědné teoretické optimalizace algoritmů.

Původním záměrem práce bylo vytvořit v praxi upotřebitelnou implementaci Courcellovy věty. Bohužel se až později ukázalo, že článek [GPW10], na němž je náš důkaz Courcellovy věty a související algoritmus založen, se pro praktickou implementaci nehodí. Naopak velmi nadějně vypadající článek [KLR11] vyšel teprve přibližně dva měsíce před termínem dokončení práce. Je zřejmé, že algoritmy vycházející z Courcellovy věty se stále ještě nachází v té fázi výzkumu, kdy má větší přínos hledání správného teoretického přístupu spíše než optimalizace zvoleného algoritmu například použitím nízkoúrovňového programovacího jazyka.

Ze zmíněných důvodů byl pro konečnou implementaci zvolen programovací jazyk Python [vR07]. Jeho hlavní předností je snadná čitelnost (jedním z cílů autora jazyka byla co největší blízkost syntaxe pseudokódu), velké množství dostupných knihoven se svobodnou licencí a také vzrůstající popularita jeho použití ve vědeckých výpočtech [JOP$^+$, S$^+$].

5.1 Struktura knihovny `pycourcelle`

Program je implementován jako knihovna `pycourcelle`, která obsahuje čtyři hlavní moduly.

Prvním je modul `graphs`, který se stará o získání stromového rozkladu a následnou normalizaci (dle Tvrzení 19). Obsahuje také další funkce související s manipulací se vstupním grafem a jeho stromovým rozkladem. Zásadní měrou se opírá o knihovnu pro práci s grafy NetworkX [HSS08], pro nalezení stromového rozkladu používá (Javovou) knihovnu LibTW [vDvdHS06], která vznikla jako práce Bodlaenderových studentů.

Druhý modul se jmenuje `formulae` a slouží pro práci s MSO formulemi. Obsahuje parser postavený na knihovně LEPL [Coo] a třídy pro příslušné typy formulí. Hierarchie tříd formulí sleduje paradigma objektově orientovaného programování, což značně zjednodušilo implementaci tohoto modulu. Modul dále zajišťuje převod formule do prenexního normálního tvaru, jehož bezkvantifikátorová část odpovídá požadavkům negační normální formy, a tudíž je takto získaná formule vhodná pro použití v obou typech popisovaných her.

Třetí modul `games` implementuje popisované logické hry – Hintikka hru MCG

a Ehrenfeucht-Fraïssé hru EFG. Tato implementace sleduje obecnou definici poziční hry, kterou jsme zavedli v úvodu Kapitoly 2.4. To umožňuje opět použít objektově orientované paradigma a například sdílet metodu pro vyhodnocení hry.

Poslední modul `algorithms` obsahuje hlavní třídu `CourcelleAlgorithm`, která používá všechny předchozí moduly. Jedná se o samotnou implementaci algoritmu vyplývajícího z důkazu Courcellovy věty, která přijímá na vstupu graf a formuli a aplikací metody `decide()` danou formuli nad grafem rozhodne. Struktura třídy odpovídá popisu algoritmu v Oddíle 3.2. Mimo tuto třídu se v tomto modulu nacházejí ještě třídy pro konkrétní algoritmy, jako je 3-obarvitelnost a hledání indukovaného podgrafu.

Pro účely ladění a analýzy běhu algoritmu je součástí knihovny též modul `utils`, který obsahuje funkci `report`, jíž lze využít jako *dekorátor* (dekorátor je syntaktický konstrukt jazyka Python umožňující snadno modifikovat existující funkce a metody nějakou funkcí vyššího řádu). Umístění řádku `@report` nad definici funkce nebo metody způsobí, že program bude při jejím volání vypisovat vstupní argumenty, a při vrácení hodnoty bude vypisovat tuto hodnotu. Příkladem použití je právě předchozí modul `algorithms` a metoda `decide`, jejíž vyhodnocení je programem vypisováno v poměrně přehledné podobě právě díky tomuto dekorátoru.

Jako demonstraci použití modulů výše popsaných nakonec obsahuje knihovna modul `examples` s funkcemi `test_graphs`, `test_formulae` atd. Jeho zdrojový kód lze chápat jako rychlý úvod do použití knihovny `pycourcelle`. Též obsahuje funkce pro testování rychlosti běhu některých algoritmů. Výsledky tohoto testu popisuje Kapitola 5.3.

5.2 Instalace a použití

Knihovna `pycourcelle` pro svůj běh vyžaduje nainstalovaný programovací jazyk Python ve verzi 2.7 nebo vyšší a Java Runtime Environment (kvůli knihovně LibTW). Dále je silně doporučeno použít nástroj `virtualenv` pro vytvoření izolovaného „kontejneru“ za účelem snadné instalace Python knihoven. Většina používaných distribucí OS Linux jím disponuje ve svých standardních repozitářích.

Kontejner vytvořený nástrojem `virtualenv` umožňuje doinstalovat libovolné Python balíčky bez potřeby zvláštních práv a zároveň bez ovlivnění zbytku systému. Navíc je do kontejneru automaticky nainstalován nástroj `pip`, který spolupracuje s online databází Python balíčků a automaticky si stáhne všechny potřebné závislosti (v našem případě knihovny NetworkX a LEPL).

V příloze práce se nachází soubor `PyCourcelle-0.1.tar.gz`; toto je Python balíček obsahující všechny potřebné informace o závislostech a podobně. Není jej potřeba rozbalovat ani pro něj vytvářet zvláštní adresář – o to vše se stará `virtualenv` a `pip`. Pomocí následujících příkazů vytvoříme `virtualenv` kontejner, aktivujeme jej a nainstalujeme knihovnu `pycourcelle`. Výstup pro stručnost vynecháváme:

```
$ virtualenv pycourcelle_testing
$ source pycourcelle_testing/bin/activate
$ pip install PyCourcelle-0.1.tar.gz
```

Mezi závislostmi balíčku `PyCourcelle` je i nástroj IPython [PG07], který

slouží k usnadnění práce s interaktivním interpretem jazyka Python a je poměrně populární ve vědecké komunitě. Nabízí například dokončování názvů pomocí stisknutí tabulátoru (*tab completion*) a vylepšený ladící nástroj `ipdb`. Následující příkazy ukazují spuštění interpretu `ipython` a krátkou ukázku práce s knihovnou `networkx`:

```
$ ipython
In [1]: import networkx as nx
In [2]: k4 = nx.complete_graph(4)
In [3]: c5 = nx.cycle_graph(5)
```

V prvním příkazu jsme načetli `networkx` a přiřadili jí alias `nx`. Ve druhém příkazu jsme vytvořili úplný graf K_4, ve třetím příkazu cyklus na 5 vrcholech C_5. NetworkX obsahuje funkce pro generování mnoha dalších typů grafů, jako třeba pravidelných mřížek, náhodných grafů atd. Též je možné načíst graf ze souboru; NetworkX podporuje většinu v praxi používaných formátů. Dokumentace NetworkX je volně dostupná a na vysoké úrovni, proto není potřeba se jí dále zabývat. Užitečná může být znalost funkce `ipython`u pro zobrazení dokumentace k libovolnému objektu – stačí za objekt umístit symbol `?`, například takto (nezajímavé položky vynecháváme):

```
In [4]: nx.cycle_graph?
Type:           function
Base Class:     <type 'function'>
Definition:     nx.cycle_graph(n, create_using=None)
Docstring:
        Return the cycle graph C_n over n nodes.

        C_n is the n-path with two end-nodes connected.

        Node labels are the integers 0 to n-1
        If create_using is a DiGraph, the direction is
        in increasing order.
```

Následujícími příkazy získáme stromový rozklad dříve definovaného C_5, zakořeníme jej a normalizujeme (dle Tvrzení 19):

```
In [5]: from pycourcelle import graphs
In [6]: td = graphs.TD(c5)
In [7]: rooted_td = td.get_rooted()
In [8]: rooted_td.normalize()
```

Nyní bychom chtěli ukázat příklad práce s modulem `formulae`. Vezměme například jednoduchou formuli, která říká, že ve grafu je množina, jež obsahuje dva různé vrcholy, mezi nimiž je hrana. Matematický zápis by byl $\varphi \equiv \exists W \exists x \exists y (\neg(x = y)) \wedge ((x, y) \in E^{\mathcal{G}}) \wedge (x \in W) \wedge (y \in W)$. Za pomoci modulu `formulae` to provedeme následovně (`"""` v Pythonu ohraničuje víceřádkový řetězec):

```
In [9]: from pycourcelle import formulae
In [10]: phi_text = """EW( Ex( Ey( (-(x=y))
   ....: & (e(x,y)) & (W(x)) & (W(y)) )))"""
In [11]: phi = formulae.Formula.parse(phi_text)
```

Zvolená formule téměř úplně demonstruje námi zvolenou syntaxi pro reprezentaci MSO formulí. Existenční kvantifikátor značíme písmenem `E`, všeobecný kvantifikátor `A`; prvkové proměnné jsou malá písmena abecedy, množinové proměnné všechna velká písmena abecedy vyjma prve zmíněných dvou. Dále máme symboly pro logické spojky následující: `&` pro $\wedge$, `|` pro $\vee$, `->` pro $\rightarrow$ a `-` pro $\neg$. Konečně atomické formule máme třech podob, totiž pro dvě prvkové proměnné `x` a `y` syntaxe `x=y` odpovídá $x = y$, `e(x,y)` odpovídá $(x, y) \in E^{\mathcal{G}}$ a pro W množinovou proměnnou značí `W(x)` vztah $x \in W$.

Dále ukážeme, jak rozhodnout `phi` nad vytvořeným grafem C_5, jak rozhodnout 3-obavitelnost nad K_4 a také jak rozhodnout, zda C_5 obsahuje K_4 jako indukovaný podgraf. První problém řeší obecný algoritmus implementovaný třídou `CourcelleAlgorithm`, který na vstupu přijímá formuli a graf. Druhý problém řeší algoritmus `SubgraphAlgorithm`, který na vstupu přijímá testovaný graf a podgraf, který v něm hledáme. Poslední problém rozhoduje algoritmus `ThreecolAlgorithm`, jenž na vstupu přijímá jediný graf.

Poslední dva algoritmy jsou implementovány jako třídy, které dědí od obecné třídy `CourcelleAlgorithm`. Vytvoření nové instance algoritmu najde stromový rozklad grafu, normalizuje jej a připraví si datové struktury pro běh algoritmu. Zavolání metody `decide` nejprve rekurzivně zjistí k-typ celého grafu (voláním metody `label`) a následně otestuje platnost formule na svědku zjištěného k-typu grafu pomocí Hintikka hry. Právě tento poslední krok můžeme nahradit vlastním algoritmem, který spoustíme nad nalezeným svědkem; vyhodnocení pomocí Hintikka hry trvá často velmi dlouho. Proto někdy testujeme pouze čas provedení metody `label`.

Tedy zpět k interpretu:

```
In [12]: alg1 = algorithms.CourcelleAlgorithm(c5, phi)
In [13]: alg2 = algorithms.ThreecolAlgorithm(k4)
In [14]: alg3 = algorithms.SubgraphAlgorithm(c5, k4)
```

A algoritmy vyhodnotíme…

```
In [15]: alg1.decide()
Out[15]: True
In [16]: alg2.decide()
Out[16]: False
In [17]: alg3.decide()
Out[17]: False
```

Více se lze o struktuře knihovny a způsobu použití dozvědět čtením zdrojového souboru `examples.py`, případně samotných implementací. Zdrojové soubory obsahují komentáře, jejich struktura je ve většině případů poměrně přehledná a význam datových struktur a metod zřejmý.

5.3 Výkon a omezení

Pro zajímavost jsme porovnali výkon knihovny v jednom případě při použití oficiálního interpreta jazyka Python a ve druhém případě při použití optimalizujícího *just-in-time* kompilátoru PyPy [BCFR09]. Tento projekt je stále ještě ve vývoji,

ale v poslední době nabývá čím dál větší popularity a například pro čistě numerické programy dosahuje výkonnostních výsledků téměř srovnatelných s programy napsanými v jazyce C.

Následující tabulka ukazuje dobu běhu algoritmu pro dva typy úloh. První je pouze `label` fáze algoritmu určení 3-obarvitelnosti čtvercové mřížky. Například pro mřížku 3×3 je v tabulce řádek `3COL 3x3?`. Druhý typ úloh je plný běh algoritmu rozhodujícího, zda je zadaný graf podgrafem vstupního grafu. Pro problém rozhodující, zda je graf K_4 podgrafem C_5, píšeme například `C5 in K4?`. Byly provedeny tři běhy programu, uvádíme průměr časů v sekundách.

Problém	**Python 2.7**	**PyPy**
`3COL 3x3?`	0,72	0,47
`3COL 4x4?`	1,58	1,62
`3COL 5x5?`	6,29	4,00
`3COL 6x6?`	22,8	21,1
`3COL 7x7?`	120	8,84
`3COL 8x8?`	1358	885
`K3 in 2x10?`	181	103
`2x2 in 2x10?`	264	130
`2x2 in 2x12?`	278	141
`2x2 in 2x14?`	241	124

Z těchto výsledků zřejmě plyne, že náš program není nijak vhodný pro použití na vstupech, s nimiž se setkáváme v praxi. Pro jakékoliv další použití programu je tedy potřeba na tato omezení pamatovat.

5.4 Možné optimalizace

Knihovna v aktuální podobě neobsahuje téměř žádné optimalizace výkonu. Zajímavá myšlenka, kterou knihovna obsahuje, pochází od Kneise et al. [KL09] a týká se vyhodnocování Ehrenfeucht-Fraïssé her. V podstatě jde o to, že v množinových tazích nezkoušíme všechny možné volby podmnožiny, ale „odložíme“ je až do prvkových tahů. Při prvkovém tahu pak prozkoumáme všechny možnosti rozšíření dosud vybraných množin tímto prvkem. Tak se vyhneme zbytečným volbám množin, jejichž prvky bychom například v dalších tazích vůbec nevybrali.

Dále také implementace obsahuje zlepšení popsané v Oddíle **??**, které při vyhodnocení relace $\equiv_k^{MSO}$ reflektuje pořadí jednotlivých typů kvantifikátorů ve formuli.

Asi největší slabinou stávající implementace je, že při vyhodnocování Ehrenfeucht-Fraïssé her se částečný isomorfismus neustále kopíruje. Možnosti řešení tohoto problému jsou dvě. První je použití persistentních datových struktur obvyklých ve funkcionálním programování – takové struktury jsou neměnné a tedy nemůžou přenést žádnou změnu stavu mezi sousedními větvemi vyhodnocení hry, ale zároveň interně většinu dat sdílejí a tedy nezaberou zdaleka tolik paměti, jako stávající přístup. Výhodou je též snadná paralelizace tohoto přístupu. Druhou možností je pečlivá implementace vyhodnocování hry tak, aby bylo možné datovou strukturu

pro částečný isomorfismus sdílet; tento přístup paralelizaci vylučuje.

Obecně je typickým prvním krokem při optimalizaci Python programu zaměřeného na výkon přepis nejvíce zatěžovaných smyček výkonnějším způsobem. To se provádí obvykle jedním ze dvou způsobů. Tradiční postup je kód přímo přepsat do jazyka C tak, aby pak bylo možné z něj zkompilovat Python modul a ten načíst do původního programu. Novější způsob je založený na software Cython [BBE08], který kompiluje zdrojové kódy v jazyce podobném Pythonu (v podstatě je tento jazyk syntaktická nadmnožina Pythonu, která oproti němu obsahuje deklarace typů a další primitiva užitečná pro optimalizaci) do optimalizovaného C/C++ kódu. Ten se opět zkompiluje do strojového kódu a lze jej načíst v původním programu.

Jak jsme ale již zmínili v úvodu této kapitoly, v současné době se zdá, že větší možnosti optimalizace algoritmu se týkají jeho teoretických základů, spíše než implementačních detailů.

6 Závěr

Ackoliv dnes patří Courcellova věta mezi klasické výsledky spadající do oblasti algoritmů a teorie složitosti, do učebnic těchto disciplín se zatím tato věta s kompletním důkazem často nedostala*. Důkaz Courcellovy věty je ovšem možno najít v řadě učebnic či knih z oblasti logiky [FG06]. Typickým prostředkem důkazu Courcellovy věty jsou tzv. stromové automaty. Za hlavní přínos této práce považujeme poskytnutí samostatného důkazu Courcellovy věty založeného na Ehrenfeucht-Fraïssé hrách.

Přestože popsaný přístup přes Ehrenfeucht-Fraïssé hry není zcela obvyklý, má některé výhody. Například poskytuje elementární nástroje teorie konečných modelů pro dokazování výsledků o popisné schopnosti jazyka. V tomto smyslu je zajímavá otázka ohledně popsatelnosti třídy „spravedlivých“ problémů [KLS09]. O některých se ví, že je lze řešit v polynomiálním čase, je-li stromová šířka grafu omezená. Nezdá se ale, že je lze popsat pomocí monadické logiky druhého řádu. Právě na tuto otázku by bylo zajímavé použít například v této práci popsané techniky teorie konečných modelů.

Použitý přístup přes Ehrenfeucht-Fraïssé hry také přivedl naši pozornost ke zobecněným kvantifikátorům. Možnostmi, které poskytují, se nejspíš zatím nikdo v souvislosti s Courcellovou větou nezabývá. My uvádíme zatím pouze vcelku elementární výsledek o zlepšení vyhodnocování Ehrenfeucht-Fraïssé hry lepším využitím znalosti formule a o Ehrenfeucht-Fraïssé hrách pro formule se zobecněnými kvantifikátory (viz Oddíly 3.3.2 a 3.3.3), ale domníváme se, že má smysl se jimi zabývat více.

Třetím přínosem práce je skutečná implementace algoritmu založeného na důkazu Courcellovy věty. Ačkoliv není svým výkonem vhodná pro použití v praxi, má svou hodnotu jako demonstrace popsaných výsledků, včetně popsaného zlepšení vyhodnocování Ehrenfeucht-Fraïssé her reflektujícího pořadí prvkových a množinových kvantifikátorů ve formuli. Navíc se v současnosti zdá vhodnější zaměřit svou pozornost směrem teoretického výzkumu.

Ve zbývajících odstavcích stručně shrneme některé možné další směry bádání souvisejícího s Courcellovou větou.

Jak nastiňuje Kapitola 4, jedním ze směrů výzkumu Courcellovy věty je hledání vhodného jazyka pro popis problémů rychle řešitelných nad strukturami s omezenou stromovou šířkou. O Courcellově větě se často mluví jako o „rámci“ pro řešení těchto problémů. Jak se ukazuje, má monadická logika druhého řádu jakožto takový rámec pro praktické použití jisté nevýhody a má smysl zkoumat i jiné cesty.

*Ovšem možná se situace brzy zlepší – doc. Fiala v současnosti pracuje na skriptech k přednášce o stromové šířce, která důkaz Courcellovy věty obsahují.

Užitečným rozšířením jazyka jsou například již zmíněné zobecněné kvantifikátory. Další možností je cesta logického programování. Otázky, které si například můžeme klást, se týkají podoby logických nebo funkcionálních programů, jež lze efektivně vyhodnotit nad strukturami omezené stromové šířky.

Také se můžeme ptát na vhodnost použití stromové šířky jako pevného parametru algoritmu. Jak zmiňuje Kneis et al. [KLR11], hlavní překážkou širšího použití klikové šířky je absence efektivních algoritmů pro hledání klikových rozkladů (srovnejme s lineárním Bodlaenderovým algoritmem pro stromovou šířku [Bod96] a výsledky o existenci dobrých heuristik [BK10]). Kliková šířka umožňuje například práci s hustými grafy.

Opačným směrem se vydává Courcelle [Cou10] s pojmem speciální stromové šířky. Ta je více omezující než standardní stromová šířka, což ale umožňuje dát spojení „efektivní algoritmus“ jeho očekávaný význam, neboli vyhnout se neelementárním dolním odhadům Fricka a Groheho [FG04] pro metaalgoritmické věty týkající se stromové šířky a MSO.

Výzkum v této oblasti je atraktivní zejména díky tomu, do kolika různých oblastí zasahuje – dotýká se například teorie grafů a algoritmů, teorie konečných modelů a logických her, teorie databází a logického programování a jiných. Propojení hlubších výsledků z oblastí, které spolu jinak příliš nesouvisejí a vzájemně nespolupracují, může přinést nový vhled do problematiky a významný pokrok.

Literatura

[AF90] Miklós Ajtai and Ronald Fagin. Reachability is harder for directed than for undirected finite graphs. *J. Symb. Log.*, 55(1):113–150, 1990.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[ALS91] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.

[BB73] Umberto Bertelè and Francesco Brioschi. On non-serial dynamic programming. *J. Comb. Theory, Ser. A*, 14(2):137–148, 1973.

[BBE08] Stefan Behnel, Robert Bradshaw, and Greg Ewing. *Cython: C-Extensions for Python*, 2008. `http://www.cython.org`.

[BCFR09] Carl Friedrich Bolz, Antonio Cuni, Maciej Fijałkowski, and Armin Rigo. Tracing the meta-level: PyPy's tracing JIT compiler. In *Proceedings of the 4th workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages and Programming Systems*, pages 18–25, Genova, Italy, 2009. ACM. `http://pypy.org`.

[Bec08] József Beck. *Combinatorial games: tic-tac-toe theory*. Encyclopedia of mathematics and its applications. Cambridge University Press, 2008.

[Bix02] Robert E. Bixby. Solving real-world linear programs: A decade and more of progress. *Operations Research*, 50(1):3–15, 2002.

[BK10] Hans L. Bodlaender and Arie M. C. A. Koster. Treewidth computations i. upper bounds. *Inf. Comput.*, 208(3):259–275, 2010.

[Bod96] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25(6):1305–1317, 1996.

[CGL+10] Andrea Calì, Georg Gottlob, Thomas Lukasiewicz, Bruno Marnette, and Andreas Pieris. Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *LICS*, pages 228–242. IEEE Computer Society, 2010.

[CGT90] Stefano Ceri, Georg Gottlob, and Letizia Tanca. *Logic Programming and Databases*. Springer, 1990.

[CMR00] Bruno Courcelle, Johann A. Makowsky, and Udi Rotics. Linear time solvable optimization problems on graphs of bounded clique-width. *Theory Comput. Syst.*, 33(2):125–150, 2000.

[CO00] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Applied Mathematics*, 101(1-3):77–114, 2000.

[Coo] Andrew Cooke. LEPL: A recursive descent parser for Python, 2009–. `http://www.acooke.org/lepl/`.

[Cou90] Bruno Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inf. Comput.*, 85(1):12–75, 1990.

[Cou10] Bruno Courcelle. Special tree-width and the verification of monadic second-order graph properties. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS*, volume 8 of *LIPIcs*, page 13–29. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2010.

[DF99] R. Downey and M. Fellows. *Parameterized complexity*, volume 19. Springer New York, 1999.

[DFR97] R. Downey, M. Fellows, and K. Regan. Descriptive complexity and the W hierarchy. In P. Beame and S. Buss, editors, *Proof Complexity and Feasible Arithmetics*, AMS-DIMACS Series in Discrete Mathematics and Theoretical Computer Science, page 119–134. American Mathematical Society, 1997.

[EF95] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer, 1995.

[Ehr61] A. Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fund. Math*, 49(129-141):13, 1961.

[Fag74] Ronald Fagin. Generalized First-Order Spectra and Polynomial-Time Recognizable Sets. In Richard Karp, editor, *Complexity of Computation*, page 43–73. American Mathematical Society, June 1974.

[FG04] Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic*, 130(1-3):3–31, 2004.

[FG06] J. Flum and M. Grohe. *Parameterized Complexity Theory (Texts in Theoretical Computer Science. An EATCS Series)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[Fra54] Ronald Fraïssé. Sur quelques classifications des systèmes de relations. *Publications Scientifiques*, Série A(1):35–184, 1954.

[Fri01] Markus Frick. Easy instances for model checking, 2001.

[GKL+07] Erich Grädel, Ph. G. Kolaitis, L. Libkin, M. Marx, J. Spencer, M. Y. Vardi, Y. Venema, and S. Weinstein. *Finite Model Theory and Its Applications*. Texts in Theoretical Computer Science. Springer, 2007.

[GPW10] Georg Gottlob, Reinhard Pichler, and Fang Wei. Monadic datalog over finite structures of bounded treewidth. *ACM Trans. Comput. Logic*, 12:3:1–3:48, November 2010.

[Gr 07] Erich Grädel. Finite model theory and descriptive complexity. In *Finite Model Theory and Its Applications*, page 125–230. Springer, 2007.

[Gr 11] Erich Grädel. Back and forth between logics and games. In *Lectures in Game Theory for Computer Scientists*, page 99–145. Springer, 2011.

[Har09] John Harrison. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press, 2009.

[Hin73] J. Hintikka. *Logic, language-games and information: Kantian themes in the philosophy of logic*. Oxford University Press, USA, 1973.

[HSS08] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*, pages 11–15, Pasadena, CA USA, August 2008. `http://networkx.lanl.gov`.

[Imm82] Neil Immerman. Relational queries computable in polynomial time (extended abstract). In *STOC*, page 147–152. ACM, 1982.

[Imm99] Neil Immerman. *Descriptive complexity*. Graduate texts in computer science. Springer, 1999.

[iOS06] Sang il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96(4):514–528, 2006.

[JOP+] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python, 2001–. `http://www.scipy.org`.

[KL09] Joachim Kneis and Alexander Langer. A practical approach to courcelle's theorem. *Electr. Notes Theor. Comput. Sci.*, 251:65–81, 2009.

[Klo94] Ton Kloks. *Treewidth, Computations and Approximations*, volume 842 of *Lecture Notes in Computer Science*. Springer, 1994.

[KLR11] Joachim Kneis, Alexander Langer, and Peter Rossmanith. Courcelle's theorem - a game-theoretic approach. *CoRR*, abs/1104.3905, 2011.

[KLS09] Petr Kolman, Bernard Lidický, and Jean-Sebastien Sereni. On fair edge deletion problems, 2009.

[LaP93] Andrea S. LaPaugh. Recontamination does not help to search a graph. *J. ACM*, 40(2):224–245, 1993.

[Lib04] Leonard Libkin. *Elements of finite model theory*. Springer Verlag, 2004.

[LRS11] Alexander Langer, Peter Rossmanith, and Somnath Sikdar. Linear-time algorithms for graphs of bounded rankwidth: A fresh look using game theory. *CoRR*, abs/1102.0908, 2011.

[Mak04] Johann A. Makowsky. Algorithmic uses of the feferman-vaught theorem. *Ann. Pure Appl. Logic*, 126(1-3):159–213, 2004.

[PG07] Fernando Pérez and Brian E. Granger. IPython: a System for Interactive Scientific Computing. *Comput. Sci. Eng.*, 9(3):21–29, May 2007. `http://ipython.scipy.org`.

[RS84] Neil Robertson and Paul D. Seymour. Graph minors. iii. planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984.

[RS86] Neil Robertson and Paul D. Seymour. Graph minors. ii. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

[S$^+$] W. A. Stein et al. *Sage Mathematics Software*. The Sage Development Team. `http://www.sagemath.org`.

[ST93] Paul D. Seymour and Robin Thomas. Graph searching and a min-max theorem for tree-width. *J. Comb. Theory, Ser. B*, 58(1):22–33, 1993.

[vDvdHS06] Thomas van Dijk, Jan-Pieter van den Heuvel, and Wouter Slob. LibTW: Computing treewidth, 2006. `http://www.treewidth.com`.

[vR07] Guido van Rossum. Python programming language. In *USENIX Annual Technical Conference*. USENIX, 2007.

[Wes11] Dag Westerståhl. Generalized quantifiers. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Summer 2011 edition, 2011. `http://plato.stanford.edu.`