VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Komponenta výukového serveru TI - P-úplné problémy

Component of Teaching Server for Theoretical Computer Science - Pcomplete problems

Tomáš Kirnig

Bakalářská práce

Vedoucí práce: Ing. Martin Kot, Ph.D.

Ostrava, 2025

VŠB TECHNICKÁ UNIVERZITA OSTRAVA | FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Č.j.VSB/25/116933
vsboee25037FA8

Zadání bakalářské práce

Student: **Tomáš Kirnig**

Studijní program: B0613A140014 Informatika

Téma: Komponenta výukového serveru TI - P-úplné problémy

Component of Teaching Server for Theoretical Computer Science - P-complete problems

Jazyk vypracování: čeština

Zásady pro vypracování:

V rámci diplomových a bakalářských prací vzniká výukový server pro předměty teoretické informatiky. Jedná se o sadu dynamických webových stránek umožňujících studentům pochopení různých typů úloh a problémů. Na rozdíl od běžných výukových textů s pevně daným počtem ukázkových příkladů umí tyto stránky generovat libovolně mnoho ukázek na základě vstupů od uživatele. Cílem této konkrétní bakalářské práce je vytvořit komponentu pro pomoc s výukou tzv. P-úplných problémů.

Vytvořte dynamické webové stránky umožňující uživateli následující:
1. Simulovat výpočet řešení problému Monotone Circuit Value Problem (MCVP) a alespoň 2 dalších P-úplných problémů.
2. Vstupy těchto algoritmů bude moci uživatel zadávat třemi způsoby:
a) Vhodným, uživatelsky přívětivým, způsobem ručně.
b) Nechat si vstup vygenerovat zcela náhodně podle nastavených parametrů.
c) Vybrat z předpřipravené sady vhodně zvolených vstupů.
3. Bude možné si zobrazit převod instance problému MCVP na ty dva zvolené P-úplné problémy. Přitom:
a) Instanci MCVP pro převod bude možné zadat kterýmkoliv z výše uvedených způsobů.
b) Převod si bude moci uživatel krokovat se zobrazením slovního vysvětlení jednotlivých kroků.
c) Na převodem vytvořenou instanci bude opět možné použít výše požadovanou simulaci výpočtu řešení.

Seznam doporučené odborné literatury:

[1] Miyano, S., Shiraishi, S., Shoudai, T.: "A List of P-Complete Problems", Kyushu University, RIFIS-TR-CS-17, December 1990, dostupné z URL: https://catalog.lib.kyushu-u.ac.jp/opac_download_md/3123/rifis-tr-17.pdf
[2] Sawa, Z.: "Teoretická informatika", podklady pro přednášky, VŠB - Technická univerzita Ostrava, dostupné z URL: https://www.cs.vsb.cz/sawa/ti/slides/ti-slides-03.pdf
[3] Papadimitriou, C.: Computational Complexity, Addison Wesley, 1993
[4] Arora, S., Barak, B.: Computational Complexity: A Modern Approach, Cambridge University Press, 2009

17. listopadu 2172/15
708 00 Ostrava-Poruba
Česká republika

spojovatelka: +420 597 321 111
epodatelna: epodatelna@vsb.cz
ID datové schránky: d3kj88v

IČ: 61989100
DIČ: CZ61989100

email: studijni.fei@vsb.cz
www.fei.vsb.cz

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Martin Kot, Ph.D.**

Datum zadání: 01.09.2024

Datum odevzdání: 30.04.2026

Garant studijního programu: Ing. Tomáš Fabián, Ph.D.

V IS EDISON zadáno: 18.08.2025 09:42:23

Abstrakt

Tato bakalářská práce se zabývá vývojem výukové webové aplikace pro demonstraci P-úplných problémů. Cílem je usnadnit studentům pochopení a procvičování těchto problémů. Aplikace se zaměřuje na tři P-úplné problémy: *Monotone Circuit Value Problem* (MCVP), *problém neprázdnosti bezkontextové gramatiky* a *kombinatorické hry na grafech*. Práce zahrnuje implementaci modulů pro interaktivní zadávání vstupů (ručně, náhodnou generací nebo výběrem z připravené sady) a simulaci jejich řešení. Uživatel může také sledovat krokový převod instance MCVP na další zmíněné problémy a následně jejich řešení.

Klíčová slova

Teoretická informatika, P-úplné problémy, Monotone Circuit Value Problem, webová aplikace, simulace, převod instancí

Abstract

This bachelor's thesis focuses on the development of a teaching-oriented web application for illustrating P-complete problems. The main goal is to facilitate students' understanding and practice of such tasks. The application focuses on the *Monotone Circuit Value Problem* (MCVP), *Empty Grammar*, and the *Combinatorial Game*. The project implements modules for interactive input of problem instances (manually, via random generation, or by selecting from a pre-defined set) and provides a simulation of their solutions. Users can also observe a step-by-step reduction from an MCVP instance to the other mentioned problems and subsequently explore how those are solved. The resulting application demonstrates key concepts of theoretical computer science, including the notion of P-completeness, and provides a flexible basis for educational use.

Keywords

Theoretical computer science, P-complete problems, Monotone Circuit Value Problem, web application, simulation, instance reduction

Obsah

Seznam použitých zkratek a symbolů

MCVP	–	Monotone Circuit Value Problem
AND	–	Logický součin – hradlo typu AND
OR	–	Logický součet – hradlo typu OR
P	–	Třída problémů řešitelných v polynomiálním čase
NC	–	Nick's Class – třída efektivně paralelizovatelných problémů
BFS	–	Breadth First Search – Algoritmus průchodu do šířky
CFG	–	Context-Free Grammar – Bezkontextová gramatika
DAG	–	Directed Acyclic Graph – Orientovaný acyklický graf
DFS	–	Depth First Search – Algoritmus průchodu do hloubky
DOM	–	Document Object Model – Objektový model dokumentu
HTML	–	Hypertext Markup Language – Hypertextový značkovací jazyk
ID	–	Identifier – Identifikátor
JSON	–	JavaScript Object Notation – Objektový zápis JavaScriptu
LS	–	Levá strana
PS	–	Pravá strana
SPA	–	Single Page Application – Jednostránková webová aplikace
TI	–	Teoretická informatika
UI	–	User Interface – Uživatelské rozhraní

Seznam obrázků

Kapitola 1

Úvod

Teoretická informatika poskytuje způsob, jak zkoumat, porozumět a optimalizovat možnosti a limity výpočetních systémů. Hlavním bodem tohoto zkoumání je teorie složitosti, která klasifikuje problémy na základě zdrojů potřebných pro jejich vyřešení, jako je čas a paměť. Jednou z nejvýznamnějších složitostních tříd je třída P, zahrnující problémy řešitelné v polynomiálním čase na deterministickém Turingově stroji [9] – tedy takové, jejichž časovou složitost lze vyjádřit jako $O(n^k)$ [10] pro nějakou konstantu k, kde n je velikost vstupu. Přestože jsou problémy v této třídě obvykle pokládány za „efektivně řešitelné“, projevují se mezi nimi významné odlišnosti, zejména pokud začneme řešit jejich paralelizovatelnost.

V tomto kontextu hraje klíčovou roli podmnožina *P-úplných* problémů (*P-complete* problems). P-úplné problémy představují výpočetně nejnáročnější úlohy v rámci třídy P. Jsou to problémy, na které lze s logaritmickou paměťovou složitostí převést jakýkoliv jiný problém z třídy P [6, 1]. Hlavní problém u této podmnožiny spočívá v předpokladu, že P-úplné problémy pravděpodobně nelze efektivně paralelizovat. To znamená, že tyto úlohy nespadají do třídy NC, která obsahuje problémy řešitelné v polylogaritmickém čase $O(\log^k n)$ pomocí paralelního výpočtu s polynomiálně mnoha procesory [7]. Z toho vyplývá, že na rozdíl od NC problémů řešení P-úplných problémů vyžaduje sekvenční zpracování. Studium P-úplnosti nám tak pomáhá vymezit hranici mezi paralelizovatelným a čistě sekvenčním výpočtem.

Tento projekt představuje interaktivní ukázku konceptu P-úplnosti prostřednictvím webové aplikace. Cílem je vytvořit nástroj, který umožní uživatelům vizualizovat a pochopit výpočet a převod mezi zvolenými P-úplnými problémy.

Jako hlavní problém byl zvolen *Monotone Circuit Value Problem* (MCVP), ve kterém se vyhodnocuje logický obvod tvořený pouze hradly typu AND a OR [7]. Pro demonstraci univerzálnosti konceptu P-úplnosti aplikace implementuje také simulace dvou dalších problémů:

- **Kombinatorické hry na grafu:** Úloha analyzující existenci vítězné strategie v deterministické hře dvou hráčů [7].

- **Problém neprázdnosti jazyka bezkontextové gramatiky:** Rozhoduje, zda daná gramatika generuje neprázdný jazyk [7].

Hlavním přínosem vytvořené aplikace je možnost vizualizace nejen samotného řešení těchto úloh, ale především *převodů* (redukcí) mezi nimi. Uživatel může sledovat krokovou transformaci instance MCVP na instanci hry nebo gramatiky, což názorně ilustruje princip polynomiálních redukcí a vzájemnou převoditelnost P-úplných problémů [7].

Výsledkem práce je webová aplikace navržená jako flexibilní nástroj pro výuku. Umožňuje uživatelům pracovat s vlastními vstupy, generovat náhodné instance pro testování a využívat předpřipravené sady úloh.

Text práce je členěn do několika částí. Po úvodu následuje kapitola věnovaná teoretickému základu, definicím klíčových pojmů a principům redukcí. Další část se zabývá analýzou a návrhem aplikace, volbou technologií a popisem datových struktur. Stěžejní částí práce je kapitola věnovaná detailnímu popisu implementace všech tří problémů a jejich vzájemných převodů. Závěr práce shrnuje dosažené výsledky a navrhuje možnosti dalšího rozšíření.

Kapitola 2

Teoretický základ

Tato kapitola zavádí základní pojmy z teorie složitosti a popisuje tři P-úplné problémy: Monotone Circuit Value Problem, kombinatorické hry na grafu a problém neprázdnosti bezkontextových gramatik.

2.1 Základní pojmy a definice

Nejprve jsou zavedeny klíčové pojmy z teorie složitosti, o které se tato práce opírá.

Definice 1 (P-úplnost [7, 6]) *Problém A se nazývá **P-úplný**, jestliže platí dvě podmínky:*

1. *$A \in P$ (problém patří do třídy P).*
2. *Pro každý problém $B \in P$ platí $B \leq_L A$ (každý problém z třídy P lze na problém A převést pomocí redukce v logaritmickém prostoru).*

Definice 2 (Třída P [9]) *Třída **P** (Polynomial time) obsahuje všechny rozhodovací problémy, které jsou řešitelné na deterministickém Turingově stroji v čase $O(n^k)$, kde n je velikost vstupu a k je nezáporná konstanta.*

Definice 3 (Logaritmická redukce [7]) *Nechť A a B jsou jazyky (problémy). Řekneme, že A je **převeditelný v logaritmickém prostoru** na B (značíme $A \leq_L B$), jestliže existuje funkce f vyčíslitelná Turingovým strojem s logaritmickou paměťovou složitostí taková, že pro každé slovo w platí:*

$$w \in A \iff f(w) \in B$$

2.2 Monotone Circuit Value Problem

Monotone Circuit Value Problem je základní problém v teorii složitosti, který patří mezi P-úplné problémy [7]. Jeho definice je následující:

- **Vstup:** Booleovský obvod bez negací, tvořený pouze hradly AND ($\wedge$) a OR ($\vee$), společně se vstupními hodnotami (pouze hodnoty 0 nebo 1 - false nebo true).

- **Výstup:** Hodnota výstupního uzlu obvodu (výstupního hradla).

Obvod lze reprezentovat jako orientovaný acyklický graf (DAG), kde uzly představují buď vstupní proměnné (listy) nebo logická hradla (vnitřní uzly). Hrany reprezentují tok logických hodnot – přenášejí výsledky vyhodnocení z jednoho uzlu jako vstupy do uzlů následujících. V monotónním obvodu chybí hradla NOT, což zajišťuje, že funkce reprezentovaná obvodem je monotónní – zvýšení hodnoty libovolného vstupu nikdy nesníží hodnotu výstupu [7].

2.2.1 P-úplnost MCVP

MCVP je P-úplný problém [7, 5]. Vyhodnocení monotónního obvodu lze provést v polynomiálním čase postupným vyhodnocováním uzlů od vstupů směrem k výstupu. MCVP je sice řešitelný v polynomiálním čase, ale jeho P-úplnost znamená, že efektivní paralelizace je pravděpodobně nemožná – vyhodnocení obvodu musí probíhat sekvenčně od vstupů k výstupu.

2.3 Kombinatorické hry na grafu

Kombinatorická hra dvou hráčů na orientovaném grafu je další příklad P-úplného problému [7]:

- **Vstup:** Orientovaný graf, kde každý uzel představuje herní pozici přiřazenou Hráči I nebo Hráči II, hrany representují možné tahy a jeden uzel je označen jako startovní pozice.

- **Výstup:** Rozhodnutí, zda Hráč I má výherní strategii ze startovní pozice.

Hráči se střídají v tazích podle toho, komu je přiřazen aktuální uzel. Hra končí, když hráč na tahu nemá žádný možný tah – tento hráč pak prohrává. Výherní strategie pro Hráče I znamená, že existuje způsob volit tahy tak, aby Hráč I vyhrál bez ohledu na to, jak hraje Hráč II [7].

2.3.1 P-úplnost kombinatorických her

Tento problém je P-úplný [7, 5]. Určení výherní strategie lze provést v polynomiálním čase pomocí tzv. retrográdní analýzy [7], která zpětně vyhodnocuje pozice od koncových uzlů. P-úplnost problému implikuje obtížnost paralelizace: výherní strategie nelze určit efektivně paralelně, protože analýza pozic na sebe vzájemně závisí.

2.4 Neprázdnost bezkontextové gramatiky

Problém neprázdnosti jazyka bezkontextové gramatiky (CFG Non-emptiness Problem) je dalším příkladem P-úplného problému [7]. Bezkontextová gramatika generuje jazyk pomocí přepisovacích pravidel.

Formálně je gramatika definována jako čtveřice $G = (N, \Sigma, P, S)$ [8], kde:

- N je konečná množina neterminálních symbolů (neterminálů).
- Σ je konečná množina terminálních symbolů (terminálů), disjunktní s N.
- P je konečná množina přepisovacích pravidel tvaru $A \to \alpha$, kde $A \in N$ a $\alpha \in (N \cup \Sigma)^*$.
- $S \in N$ je počáteční symbol (start symbol).

Problém spočívá v tom, zda daná gramatika generuje alespoň jedno slovo složené pouze z terminálních symbolů.

2.4.1 P-úplnost problému neprázdnosti

Problém neprázdnosti jazyka bezkontextové gramatiky je P-úplný [7, 5]. Tento problém lze vyřešit v polynomiálním čase pomocí algoritmu iterativního označování produktivních neterminálů. Podobně jako u předchozích problémů, i zde P-úplnost vyplývá ze sekvenčních závislostí: produktivita neterminálu závisí na produktivitě jiných neterminálů, což brání paralelnímu zpracování.

2.5 Principy převodů mezi problémy

P-úplné problémy jsou vzájemně převoditelné, což dokazuje jejich výpočetní rovnocennost. Následující převody ilustrují, jak lze MCVP převést na kombinatorickou hru a na problém neprázdnosti gramatiky.

2.5.1 Převod z MCVP na kombinatorickou hru

Uzly obvodu MCVP odpovídají herním pozicím [7]:

- **Hradlo OR** vytváří pozici pro Hráče 1. Ten si může vybrat libovolnou následující pozici odpovídající potomkům hradla. Stačí, aby jedna z možností vedla k jeho výhře.
- **Hradlo AND** vytváří pozici pro Hráče 2. Ten vybírá následující pozici a snaží se zabránit výhře Hráče 1. Hráč 1 vyhraje pouze pokud vyhrává ve všech možných pokračováních.
- **Proměnná s hodnotou 1** odpovídá konečné pozici, ve které Hráč 2 nemá žádné tahy – Hráč 1 tedy vyhrává.

- **Proměnná s hodnotou 0** odpovídá konečné pozici, ve které Hráč 1 nemá žádné tahy a prohrává.

2.5.2 Převod z MCVP na bezkontextovou gramatiku

Každý uzel obvodu se stane symbolem gramatiky [7]:

- **Kořen obvodu** se mapuje na počáteční symbol gramatiky S.
- **Hradlo AND s potomky** A **a** B vytvoří pravidlo $X \to AB$, kde X reprezentuje hradlo. Řetězec lze z X odvodit právě tehdy, když lze odvodit řetězce z obou potomků A i B.
- **Hradlo OR s potomky** A **a** B vytvoří dvě pravidla: $X \to A$ a $X \to B$.
- **Proměnná s hodnotou 1** generuje epsilon pravidlo $X \to \varepsilon$, což umožňuje odvození prázdného řetězce.
- **Proměnná s hodnotou 0** vytvoří pravidlo $X \to t$, kde t je terminál bez dalších odvozovacích pravidel.

Kapitola 3

Analýza a návrh

Tato kapitola popisuje zvolené technologie a architekturu aplikace. Dále představuje datové struktury pro reprezentaci jednotlivých problémů a principy návrhu uživatelského rozhraní.

3.1 Volba technologií

Aplikace byla implementována jako webová, protože nevyžaduje instalaci a běží v libovolném moderním prohlížeči. Webové řešení také umožňuje snadnou údržbu a okamžitou distribuci aktualizací.

3.1.1 React a Vite

Pro vývoj byla použita knihovna *React* [4]. React umožňuje rozdělit uživatelské rozhraní na znovupoužitelné komponenty, což usnadnilo vývoj modulů pro jednotlivé problémy. Díky reaktivitě se rozhraní automaticky synchronizuje se stavem dat bez nutnosti ruční manipulace s DOM.

Pro sestavení aplikace byl použit nástroj *Vite* [11]. Vite nabízí rychlé vývojové prostředí s okamžitou odezvou při úpravách kódu a optimalizuje soubory pro produkční verzi.

3.2 Architektura a struktura kódu

Aplikace je navržena jako *Single Page Application* (SPA) – veškerá logika a navigace probíhá v rámci jedné stránky, podobně jako u desktopové aplikace. Kořenová komponenta `App.jsx` spravuje globální stav a přepínání mezi moduly.

Kód je strukturován tak, aby logika byla oddělena od vizualizace. Každý modul (MCVP, hry, gramatiky) disponuje vlastní adresářovou strukturou:

- **Hlavní komponenta:** Funguje jako kontejner, který spravuje lokální stav a propojuje vstupní data s algoritmy a vizualizací.

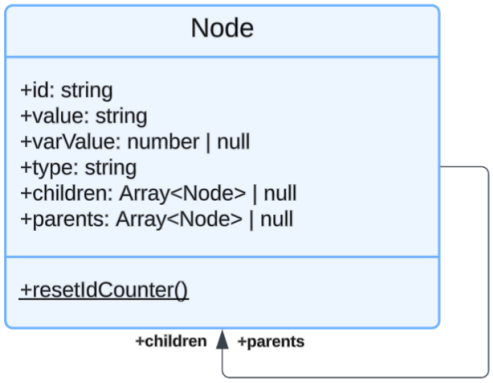
- **Adresář Utils:** Zde jsou soustředěny čisté algoritmy – parsery pro zpracování vstupu, generátory náhodných instancí a evaluátory pro výpočet řešení.
- **Vizualizační komponenty:** Starají se o vykreslení grafů a stromů pomocí knihovny *react-force-graph*.

3.3 Návrh datových struktur

Pro práci s P-úplnými problémy bylo nutné navrhnout jejich vnitřní reprezentaci v paměti.

3.3.1 Reprezentace obvodu MCVP

Pro modelování logického obvodu byl využit orientovaný acyklický graf. Každý uzel obvodu je reprezentován instancí třídy `Node`. Tato třída, jak ukazuje třídní diagram na obrázku 3.1, uchovává informaci o typu operace (AND, OR) nebo hodnotě proměnné. Seznam odkazů na potomky umožňuje rekurzivní průchod stromem při vyhodnocování.



Obrázek 3.1: Třídní diagram pro hlavní třídy MCVP

3.3.2 Reprezentace herního grafu

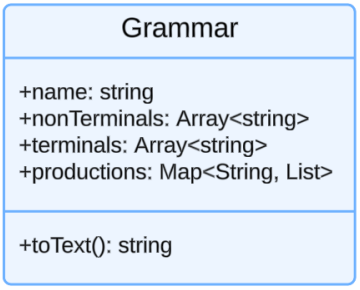
Na rozdíl od MCVP může herní graf obsahovat cykly, proto byly navrženy třídy `GamePosition` a `GameGraph` (viz obrázek 3.2). Třída `GamePosition` reprezentuje jednotlivou pozici a uchovává informaci o hráči na tahu spolu se seznamy předchůdců a následníků, což je nezbytné pro retrográdní analýzu.

3.3.3 Reprezentace bezkontextové gramatiky

U bezkontextových gramatik je zaměřeno na reprezentaci přepisovacích pravidel. Třída `Grammar` (obrázek 3.3) spravuje množiny symbolů a seznamy produkcí. Každá produkce propojuje levou

Obrázek 3.2: Třídní diagram pro hlavní třídy kombinatorické hry

stranu (neterminál) s polem symbolů na pravé straně. Tato struktura umožňuje snadno identifikovat produktivní neterminály.



Obrázek 3.3: Třídní diagram pro hlavní třídu bezkontextové gramatiky

3.4 Návrh uživatelského rozhraní

Rozhraní je navrženo tak, aby student nebyl zahlcen informacemi a mohl se soustředit na podstatu problému. Rozložení stránky bylo sjednoceno napříč všemi moduly:

- **Ovládací panel:** Horní část obrazovky, kde uživatel volí způsob zadání vstupu a spouští simulace.
- **Interaktivní plátno:** Hlavní plocha pro vizualizaci grafů, kde uživatel může s prvky přímo manipulovat.
- **Informační panely:** Modální okna nebo postranní panely zobrazující detaily o probíhajícím výpočtu nebo krokovém převodu.

Pro vizualizaci byla použita knihovna *react-force-graph* [2], která pomocí fyzikální simulace automaticky rozmísťuje uzly a hrany. Stylování je realizováno frameworkem *Bootstrap 5* [3], který zajišťuje jednotný vzhled a responzivitu na různých velikostech obrazovek.

Kapitola 4

Implementace

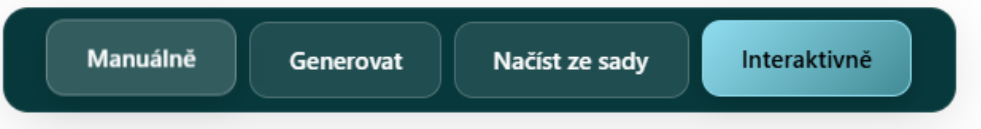
Tato kapitola detailně popisuje implementaci jednotlivých modulů aplikace. Zaměřuje se na technické řešení algoritmů pro parsování, vyhodnocování a generování instancí P-úplných problémů.

4.1 Monotone Circuit Value Problem

Modul pro MCVP tvoří jádro aplikace a slouží jako výchozí bod pro převody na ostatní problémy.

4.1.1 Formát vstupu a parsování výrazů

Aplikace umožňuje zadat monotónní obvod několika způsoby (viz obrázek 4.1). Jako první je textový zápis ve formě logického výrazu, který je následně převeden na strukturu obvodu (DAG).



Obrázek 4.1: Výběr způsobu zadání vstupu pro MCVP

4.1.1.1 Gramatika vstupního jazyka

Vstupní výrazy odpovídají jednoduché gramatice popsané v kapitole 2.2. Pro připomenutí:

- Operátory `O` (OR) a `A` (AND) reprezentují logický součet a součin.
- Proměnné jsou zapsány ve tvaru `x1[0]` nebo `x2[1]`, kde číslo v hranatých závorkách udává hodnotu proměnné (0 nebo 1).
- Výrazy lze libovolně uzávorkovat pomocí kulatých závorek pro určení priority vyhodnocení.

Příklad platného výrazu: `((x1[1] A x2[0]) O (x3[1] A x4[1]))`

4.1.1.2 Lexikální a syntaktická analýza

Převod textového výrazu na stromovou strukturu probíhá ve dvou fázích, které jsou implementovány v modulu `Parser.js`:

1. **Tokenizace (lexikální analýza):** Funkce `tokenize()` rozdělí vstupní řetězec na posloupnost tokenů. Každý token je dvojice (typ, hodnota), například `['LPAREN', '(']` nebo `['VARIABLE', 'x1[1]']`. Tokenizér rozpoznává závorky, operátory (A, O) a proměnné pomocí regulárních výrazů.

2. **Parsování (syntaktická analýza):** Třída `Parser` implementuje rekurzivní sestupný parser respektující prioritu operátorů. Výrazy s operátorem OR mají nejnižší prioritu, následují výrazy s operátorem AND a nejvyšší prioritu mají atomické výrazy v závorkách nebo proměnné. Parser vytváří uzly typu `Node` (viz třídní diagram v sekci 3.3.1, obrázek 3.1), které tvoří strukturu grafu reprezentující zadaný obvod.

Výsledkem parsování je orientovaný graf, kde vstupní uzly jsou proměnné s přiřazenými hodnotami a vnitřní uzly reprezentují logické operace.

4.1.2 Vyhodnocení obvodu

Vyhodnocení monotónního obvodu probíhá rekurzivním průchodem grafu, který je implementován v modulu `EvaluateTree.js`.

4.1.2.1 Algoritmus vyhodnocení

Funkce `evaluateTree()` používá algoritmus průchodu do hloubky (DFS) s memoizací:

1. **Listy (proměnné):** Pro uzly typu `variable` funkce vrací přímo přiřazenou hodnotu (0 nebo 1).

2. **Vnitřní uzly (operace):** Pro uzly reprezentující logické operace funkce nejdříve vyhodnotí všechny potomky. Pokud je uzel typu AND, výsledek je 1 právě tehdy, když všichni potomci mají hodnotu 1. Pokud je uzel typu OR, výsledek je 1 pouze tehdy, když alespoň jeden potomek má hodnotu 1.

3. **Memoizace:** Vyhodnocené hodnoty uzlů jsou uloženy do cache (slovníku), aby nedocházelo k opakovanému výpočtu stejných uzlů.

4. **Detekce cyklů:** Algoritmus používá množinu `visiting` pro detekci případných cyklů v grafu, které by způsobily nekonečnou rekurzi.

Časová složitost algoritmu je $O(n)$, kde n je počet uzlů v obvodu, protože každý uzel je vyhodnocen právě jednou díky memoizaci.

4.1.2.2 Krokové vyhodnocení

Aplikace také nabízí krokovatelné vyhodnocení pomocí funkce `evaluateTreeWithSteps()`. Tato funkce provádí stejný výpočet jako `evaluateTree()`, ale navíc zaznamenává při každém výpočtu i stav zbytku obvodu. Uživatel tak může sledovat, jak se hodnoty šíří od vstupů k výstupu obvodu a lépe pochopit princip vyhodnocování.

Komponenta `StepByStepTree` zobrazuje každý krok graficky – aktuálně vyhodnocovaný uzel je zvýrazněn a vedle grafu je zobrazena informace o hodnotách vstupů a výsledku operace.

4.1.3 Interaktivní editace obvodu

Kromě textového zadání umožňuje aplikace vytvářet a upravovat obvody pomocí grafického editoru implementovaného v komponentě `InteractiveMCVPGraph`. Uživatel může:

- **Přidávat uzly:** Pomocí tlačítek pod grafem lze vložit nové uzly typu AND, OR nebo vstupní proměnnou s hodnotou 0 nebo 1. Nový uzel se umístí do grafu, ale není propojen s ostatními uzly.
- **Vytvářet hrany:** Označit uzel, ten následně může:
  - Nastavit na AND nebo OR operaci.
  - Nastavit na proměnnou s hodnotou 0 nebo 1.
  - Odstranit uzel z grafu.
  - Vytvořit nebo odstranit hranu s dalším uzlem.
- **Reorganizovat graf:** Uzly lze přesouvat myší pro lepší vizuální uspořádání obvodu.

Grafický editor využívá knihovnu *react-force-graph* pro vykreslování a interakci (viz sekce 3.4). Aplikace průběžně aktualizuje interní strukturu grafu odpovídající aktuálnímu stavu grafu.

4.1.4 Generování náhodných obvodů

Aplikace obsahuje generátor náhodných obvodů v modulu `Generator.js`. Uživatel může upravit dva parametry:

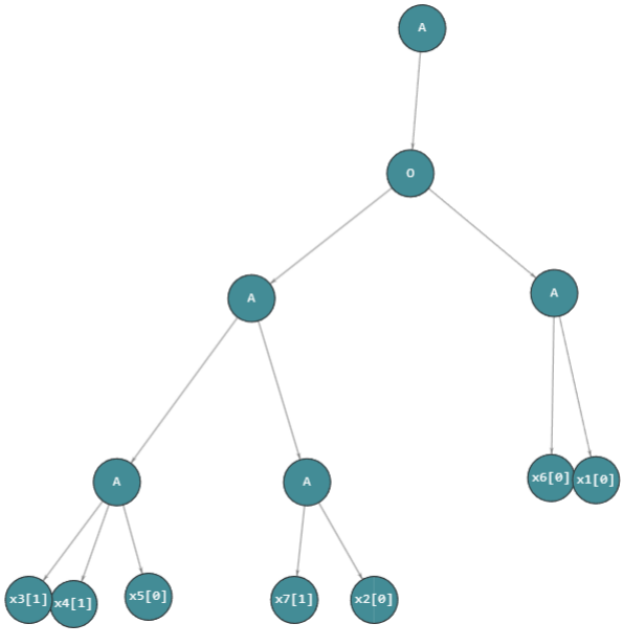
- **Počet hradel:** Kolik uzlů s operacemi (hradel AND a OR) bude obvod obsahovat.
- **Počet proměnných:** Kolik uzlů s proměnnými (listů) bude obvod obsahovat.

Generování probíhá v těchto fázích: Nejprve algoritmus vytvoří zadaný počet proměnných, s náhodnou hodnotou 0 nebo 1. Pak postupně přidává hradla. Pro každé hradlo se vypočítá cílový počet potomků podle vzorce $\lceil n/k \rceil$, kde n je počet dosud dostupných uzlů a k počet zbývajících

hradel. K této hodnotě se přidá náhodná odchylka ±20% – tak je dosaženo vyváženosti mezi úplnou náhodností a vyrovnanou distribucí. Hradlo pak náhodně vybere určený počet uzlů jako potomky a náhodně zvolí operaci (AND nebo OR) a přidá se do množiny použitelných uzlů pro potomky. Poslední hradlo spojí všechny zbývající uzly a stane se kořenem. Výsledný graf je platný DAG.

4.1.5 Vizualizace obvodu

Vizualizace monotónního obvodu je implementována v komponentě `TreeRenderCanvas`. Obvod je zobrazen jako orientovaný graf s výstupním hradlem nahoře a vstupy dole, hodnoty se tedy šíří směrem zdola nahoru (viz obrázek 4.2).



Obrázek 4.2: Vizualizace MCVP obvodu pomocí TreeRenderCanvas

4.1.5.1 Hierarchické rozložení

Pro vizualizaci grafu používá aplikace knihovnu *react-force-graph-2d*. Uzly jsou automaticky umístěny do vrstev podle jejich vzdálenosti od kořene díky režimu `dagMode="td"` (top-down neboli shora dolů). Rozložení kombinuje hierarchickou strukturu s fyzikálními silami pro optimální vizuální uspořádání grafu.

4.2 Kombinatorická hra

4.2.1 Formát vstupu

Aplikace nabízí tři způsoby zadávání herního grafu:

- **Interaktivní editace:** Uživatel vytváří a upravuje graf pomocí grafického editoru (viz sekce 4.2.2).
- **Generování náhodných her:** Automatické vytvoření náhodného grafu podle zadaných parametrů (viz sekce 4.2.3).
- **Předpřipravené sady:** Načtení ukázkových připravených příkladů (viz sekce 4.2.4).

Všechny metody využívají komponentu `DisplayGraph` pro vizualizaci herního grafu (viz sekce 4.2.5).

4.2.2 Interaktivní editace grafu

Komponenta `ManualInput` umožňuje vytvářet a upravovat herní grafy pomocí interaktivního grafického editoru. Uživatel může:

- **Přidávat uzly:** Vytvořit novou pozici.
- **Upravovat uzly:** Kliknutím na uzel ho označíme. S označeným uzlem lze:
  - Změnit hráče který je na tahu (Hráč I nebo Hráč II)
  - Odstranit pozici z grafu
  - Nastavit jako počáteční pozici
  - Použít jako zdroj nebo cíl pro vytvoření hrany
  - Vytvořit hranu z nebo do tohoto uzlu
  - Odstranit hrany k nebo od tohoto uzlu
- **Reorganizovat graf:** Uzly lze přesouvat myší pro lepší vizuální uspořádání.

Aplikace průběžně validuje graf. Uživatel je upozorněn, pokud není nastavena počáteční pozice, což je nutné pro spuštění analýzy.

4.2.3 Generování náhodných her

Modul `Generator.js` obsahuje funkci `generateGraph()` pro vytváření náhodných herních grafů. Uživatel nastavuje dva parametry:

- **Počet pozic:** Kolik uzlů (herních pozic) bude graf obsahovat.
- **Pravděpodobnost hrany:** Hodnota 0% – 100% určující, jak pravděpodobné je vytvoření hrany mezi dvěma uzly.

Algoritmus generování probíhá ve dvou fázích:

1. **Vytvoření kostry:** Vytvoří se uzly očíslované 0 až $n-1$ a každému se náhodně přiřadí hráč. Pro každý uzel $i > 0$ pak vznikne hrana z náhodného předchozího uzlu (s indexem menším než i) do uzlu i. Tím je zajištěno, že uzel 0 (počáteční pozice) může dosáhnout všechny ostatní uzly, a současně vznikne acyklická kostra grafu.
2. **Přidání dalších hran:** Pro každou dvojici uzlů i, j (kde $i \neq j$), podle dané pravděpodobnosti se přidá hrana $i \to j$, pokud ještě neexistuje. Hrany mohou být přidány v libovolném směru, což umožňuje vznik cyklů v grafu.

Vnitřní reprezentace grafu pomocí tříd `GamePosition` a `GameGraph` je popsána v sekci 3.3.2. Diagram těchto tříd je uveden na obrázku 4.3.

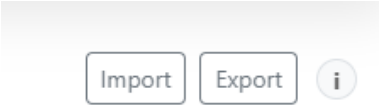
Obrázek 4.3: Třídní diagram pro hlavní třídu kombinatorické hry

4.2.4 Předpřipravené sady

Ve složce `Sady/CombinatorialGame` najdeme předpřipravené herní grafy různé velikosti a složitosti.

4.2.4.1 Ukládání a načítání her

Herní grafy lze exportovat a importovat ve formátu JSON pomocí komponenty `FileTransferControls` (viz obrázek 4.4). Formát obsahuje pole `nodes` s uzly, pole `edges` s hranami a identifikátor `startingPosition`.



Obrázek 4.4: Ovládací prvky pro import a export herních grafů

4.2.5 Vizualizace grafu

Vizualizace herního grafu využívá komponentu `DisplayGraph`, která využívá knihovnu *react-force-graph-2d*. Graf zobrazuje:

- **Uzly:** Každý uzel reprezentuje herní pozici s označením hráče na tahu (I nebo II).
- **Počáteční pozice:** Označena oranžovou barvou.
- **Hrany:** Možné tahy jsou zobrazeny jako směrované hrany. Hrany patřící do výherní strategie jsou zvýrazněny tlustší čarou a žlutou barvou.

4.2.6 Algoritmus analýzy

Řešení problému kombinatorických her je implementováno v modulu `ComputeWinner.js`. Algoritmus využívá retrográdní analýzu založenou na principech uvedených v sekci 2.3.

4.2.6.1 Vyhodnocení výherních pozic

Algoritmus určuje, zda je daná pozice výherní pro hráče, který je v ní na tahu, nebo zda skončí remízou. Vyhodnocení probíhá následovně:

1. **Koncové pozice:** Pozice bez dalších tahů jsou prohrávající pro hráče, který je v nich na tahu.
2. **Zpětné šíření:** Od koncových pozic se postupně propagují výsledky:
   - Pokud existuje tah do prohrávající pozice soupeře, aktuální pozice je vyhrávající.
   - Pokud tentýž hráč pokračuje v tahu (bez změny hráče), tah do vyhrávající pozice znamená, že i původní pozice je vyhrávající.
   - Pokud všechny tahy vedou do vyhrávajících pozic soupeře, aktuální pozice je prohrávající.
3. **Neurčené pozice:** Pozice, jejichž status nelze jednoznačně určit (například kvůli cyklům), zůstávají označeny jako remíza.

Časová složitost je $O(V + E)$, kde V je počet uzlů a E počet hran.

4.2.6.2 Optimální tahy

Funkce `getOptimalMoves()` identifikuje hrany, které jsou součástí výherní strategie. Hrana z pozice u do pozice v je optimální, pokud obě pozice jsou výherní pro Hráče I. Tyto hrany jsou zvýrazněny ve vizualizaci.

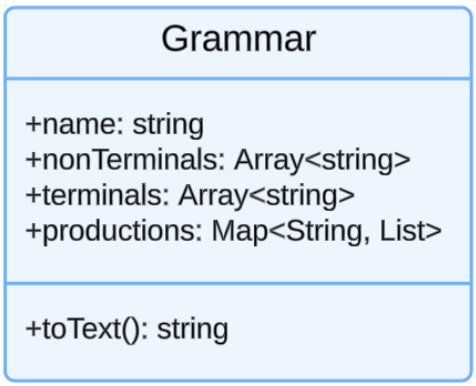
4.2.7 Krokové vyhodnocení

Pro detailnější průchod grafem implementuje komponenta `StepByStepGame` krokovatelnou analýzu. Algoritmus zaznamenává každý krok aktualizace stavu grafu a uživatel jimi může procházet pomocí navigačních tlačítek. Aktuálně aktualizovaný uzel je v grafu zvýrazněn. Tato funkce pomáhá pochopit, jak algoritmus postupně řeší hru a jak se vypořádává s cykly.

4.3 Bezkontextové gramatiky

4.3.1 Formát vstupu

Aplikace nabízí tři způsoby zadávání bezkontextové gramatiky: manuální zadání, náhodné generování a výběr z připravených sad. Všechny formy vstupu využívají stejnou komponentu pro zobrazení derivačního stromu gramatiky implementovanou ve třídě `Grammar` (viz sekce 3.3.3). Třídní diagram této struktury je na obrázku 4.5.



Obrázek 4.5: Třídní diagram pro hlavní třídu bezkontextové gramatiky

4.3.2 Manuální zadání gramatiky

Komponenta `ManualInput` poskytuje rozhraní ve formě textového okna pro přímé zadávání pravidel gramatiky ve formátu `LS -> PS`. Algoritmus v modulu `GrammarParser.js` automaticky rozpoznává neterminály (velká písmena) a terminály.

4.3.3 Generování náhodných gramatik

Modul `GrammarGenerator.js` obsahuje funkci `generateGrammar()` pro vytváření náhodných bezkontextových gramatik na základě vstupních parametrů.

4.3.3.1 Základní parametry

Základní parametry určují počet neterminálů (1–10), počet terminálů (1–10) a maximální délku pravé strany (1–5).

4.3.3.2 Pokročilé parametry

Pokročilé parametry umožňují kontrolu nad počtem pravidel na neterminál, povolením levé/pravé rekurze a režimem generování epsilon pravidel.

4.3.3.3 Algoritmus generování

Algoritmus nejprve vytvoří symboly a následně pro každý neterminál generuje náhodný počet pravidel. Při tvorbě pravých stran zohledňuje nastavené pravděpodobnosti pro terminály, neterminály a rekurzivní volání.

4.3.4 Předpřipravené sady

Ve složce `Sady/Grammar` jsou uloženy předpřipravené příklady gramatik ve formátu JSON, jehož struktura je popsána v sekci 3.3.3.

4.3.5 Algoritmus vyhodnocení

Modul `GrammarEvaluator.js` obsahuje implementaci algoritmu pro vyhodnocení neprázdnosti gramatiky založenou na teorii popsané v sekci 2.4.

4.3.5.1 Identifikace produktivních neterminálů

Algoritmus pracuje iterativně a postupně označuje produktivní neterminály. Implementace používá optimalizovanou verzi s frontou (work-list algorithm), což odpovídá principu BFS. Časová složitost je $O(|P| \cdot l)$.

4.3.5.2 Rekonstrukce derivace

Pokud je počáteční symbol S produktivní, funkce `buildNode()` rekurzivně sestaví derivačního strom. Pro každý neterminál náhodně vybírá z jeho produktivních pravidel, přičemž preferuje nerekurzivní varianty pro zamezení příliš hlubokých stromů.

4.3.6 Vizualizace derivačního stromu

Pokud gramatika generuje neprázdný jazyk, aplikace vizuálně ukáže derivační strom pomocí komponenty `DerivationTreeVisual`, využívající knihovnu *react-force-graph-2d* v režimu top-down.

4.3.7 Krokové vyhodnocení

Komponenta `StepByStepGrammar` využívá modul `GrammarStepEvaluator.js` k interaktivní simulaci algoritmu. Zobrazuje seznam pravidel s barevným zvýrazněním aktuálního stavu a množinu dosud nalezených produktivních symbolů.

4.4 Implementace převodů mezi problémy

Pro ilustraci vzájemné převoditelnosti P-úplných problémů popsané v sekci 2.5 aplikace implementuje dva klíčové převody z instance MCVP.

4.4.1 Převod na kombinatorickou hru

Třída `MCVPToGameStepGenerator` prochází MCVP graf rekurzivně a pro každý uzel vytváří odpovídající pozici ve hře dle pravidel definovaných v sekci 2.5.1. Pro efektivitu používáme memoizaci. Vizualizace probíhá pomocí komponenty `MCVPtoCombinatorialGameConverter`, která zobrazuje převod krokovatelně.

4.4.2 Převod na bezkontextovou gramatiku

Převod implementuje třída `MCVPToGrammarConverter` dle principů ze sekce 2.5.2. Nejprve přiřadí uzlům unikátní symboly a poté rekurzivně generuje pravidla odpovídající typům hradel. Stejně jako u prvního převodu, komponenta `MCVPtoGrammarConverter` nabízí krokovou vizualizaci celého procesu.

Kapitola 5

Závěr

Cílem této práce bylo vytvořit interaktivní webovou aplikaci pro vizualizaci a demonstraci převoditelnosti a řešení P-úplných problémů na příkladu tří vybraných P-úplných problémů: Monotone Circuit Value Problem, kombinatorických her na grafu a problému neprázdnosti jazyka bezkontextové gramatiky.

5.1 Dosažené výsledky

Výsledná aplikace splňuje všechny stanovené cíle a poskytuje jednotné prostředí pro práci se třemi P-úplnými problémy. Pro každý problém je implementováno kompletní řešení zahrnující:

- **Flexibilní vstupní systém:** Uživatelé mohou zadávat instance problémů třemi způsoby – manuálním zadáním, generováním náhodných instancí nebo pomocí předpřipravených příkladů. Tyto možnosti umožňují jak experimentování s vlastními příklady, tak rychlé testování algoritmu na náhodných datech.
- **Vizualizaci řešení:** Každý problém je doprovázen grafickou reprezentací, která využívá knihovnu *react-force-graph-2d* pro interaktivní zobrazení grafových struktur. Uživatel může manipulovat se zobrazenými grafy, přibližovat je a přesouvat uzly pro lepší orientaci.
- **Krokové vyhodnocení:** Implementace krokovatelného průchodu algoritmů umožňuje sledovat každý jednotlivý krok výpočtu s textovým vysvětlením.
- **Export a import dat:** Možnost ukládání a načítání instancí problémů ve formátu JSON podporuje sdílení příkladů a vytváření knihoven testovacích případů.

Aplikace taktéž demonstruje dva konkrétní převody z MCVP na kombinatorickou hru a na bezkontextovou gramatiku. Oba převody jsou implementovány krokovatelně, takže uživatel může sledovat, jak se jednotlivé uzly obvodu transformují na odpovídající struktury v cílovém problému.

Tato vizualizace názorně ilustruje techniku polynomiálních redukcí a vzájemnou převoditelnost P-úplných problémů.

5.2 Vzdělávací přínos

Aplikace představuje vzdělávací nástroj pro výuku teorie složitosti s interaktivním přístupem. Krokové vyhodnocení s textovými vysvětleními umožňuje pochopit fungování algoritmů na konkrétních příkladech, zatímco vizualizace převodů demonstruje vzájemnou převoditelnost P-úplných problémů. Generování náhodných instancí pak podporuje experimentální učení a pozorování chování algoritmů na různých strukturách vstupů.

5.3 Možnosti dalšího rozvoje

Přestože aplikace poskytuje funkční implementaci všech plánovaných funkcí, existuje prostor pro další rozšíření:

- **Další P-úplné problémy:** Aplikace by mohla být rozšířena o další P-úplné problémy, jako je například vyhodnocování booleovských formulí v konjunktivní normální formě nebo problém dosažitelnosti v grafech s omezenou šířkou.
- **Více převodů:** Implementace dalších směrů převodů – například z kombinatorických her na gramatiky nebo opačným směrem z gramatik na MCVP – by poskytla kompletnější obraz vzájemné převoditelnosti těchto problémů.
- **Výkonnostní optimalizace:** Pro velmi velké instance (stovky uzlů) by mohly být implementovány optimalizace vykreslování a výpočtu, například pomocí virtualizace zobrazení nebo progresivního vykreslování.

Výsledná aplikace může sloužit jako doplněk k tradičním výukovým materiálům v kurzech teorie složitosti a teoretické informatiky, kde pomáhá studentům lépe pochopit abstraktní koncepty prostřednictvím konkrétních interaktivních příkladů.

Literatura

1. ARORA, Sanjeev; BARAK, Boaz. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. ISBN 978-0521424264.
2. ASTURIANO, Vasco. *react-force-graph: React component for 2D, 3D, VR and AR force directed graphs [online]* [GitHub]. 2024. Dostupné také z: `https://github.com/vasturiano/react-force-graph`. [cit. 2026-02-03].
3. BOOTSTRAP TEAM. *Bootstrap: Build fast, responsive sites [online]* [OpenJS Foundation]. 2024. Dostupné také z: `https://getbootstrap.com/`. Verze 5.3.8. [cit. 2026-02-03].
4. META PLATFORMS, INC. *React: The library for web and native user interfaces [online]* [Meta Open Source]. 2024. Dostupné také z: `https://react.dev/`. Verze 19.2. [cit. 2026-02-03].
5. MIYANO, Satoru; SHIRAISHI, Shunsuke; SHOUDAI, Takayoshi. *A List of P-Complete Problems [online]*. 1990-12. Tech. zpr., RIFIS-TR-CS-17. Fukuoka: Kyushu University, Research Institute of Fundamental Information Science. Dostupné také z: `https://catalog.lib.kyushu-u.ac.jp/opac_download_md/3123/rifis-tr-17.pdf`. [cit. 2026-02-03].
6. PAPADIMITRIOU, Christos H. *Computational Complexity*. Addison Wesley, 1994. ISBN 978-0201530827.
7. SAWA, Zbyněk. *Teoretická informatika – Podklady pro přednášky [online]* [Ostrava: VŠB-TUO, Katedra informatiky]. 2025. Dostupné také z: `https://www.cs.vsb.cz/sawa/ti/slides/ti-slides-08.pdf`. [cit. 2026-02-03].
8. SAWA, Zbyněk. *Úvod do teoretické informatiky – Bezkontextové gramatiky [online]* [Ostrava: VŠB-TUO, Katedra informatiky]. 2014. Dostupné také z: `https://www.cs.vsb.cz/kot/download/uti2014/uti-10-cz.pdf`. [cit. 2026-02-03].
9. SAWA, Zbyněk. *Úvod do teoretické informatiky – Turingovy stroje [online]* [Ostrava: VŠB-TUO, Katedra informatiky]. 2014. Dostupné také z: `https://www.cs.vsb.cz/kot/download/uti2014/uti-08-cz.pdf`. [cit. 2026-02-03].

10. SAWA, Zbyněk. *Úvod do teoretické informatiky – Výpočetní složitost algoritmů [online]* [Ostrava: VŠB-TUO, Katedra informatiky]. 2014. Dostupné také z: `https://www.cs.vsb.cz/kot/download/uti2014/uti-12-cz.pdf`. [cit. 2026-02-03].

11. VITE TEAM. *Vite: The Build Tool for the Web [online]* [VoidZero Inc.]. 2024. Dostupné také z: `https://vite.dev/`. Verze 6. [cit. 2026-02-03].