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/24/132422
vsboee2403DA70

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: 30.04.2025

Garant studijního programu: doc. Mgr. Miloš Kudělka, Ph.D.

V IS EDISON zadáno: 30.10.2024 09:50:55

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), *prázdnou gramatiku* a *kombinatorickou hru*. 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
P	–	Třída problémů řešitelných v polynomiálním čase
NC	–	Nick's Class – třída efektivně paralelizovatelných problémů

Seznam obrázků

Seznam tabulek

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 – tedy takové, jejichž časová složitost lze vyjádřit jako $O(n^k)$ pro nějakou konstantu k (například $k = 2$ pro kvadratickou složitost, $k = 3$ pro kubickou), kde n je velikost vstupu – na deterministickém Turingově stroji [1]. Přestože jsou problémy v této třídě obvykle pokládány za „efektivně řešitelné“, projevují se mezi nimi významné odlišnosti, obvzvláště když 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. Hlavní problém u této podmnožiny problémů 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 [2]. 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 složený z pouze dvou logických hradel [3] (tedy logický součin a součet). 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 [3].

- **Prázdnost bezkontextových gramatik:** Problém rozhodující, zda daná gramatika generuje neprázdný jazyk [3].

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 [2].

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 použitým technologiím a architektuře aplikace. Jádro práce tvoří tři kapitoly, z nichž každá se detailněji věnuje jednomu z implementovaných problémů: nejprve Monotone Circuit Value Problem (MCVP), následně kombinatorické hry na grafu a nakonec problém prázdnosti bezkontextových gramatik. Závěr práce shrnuje dosažené výsledky a navrhuje možnosti dalšího rozšíření.

Kapitola 2

Použité technologie a architektura

Tato kapitola popisuje technologický základ vytvořené aplikace a její architekturu.

2.1 Webové technologie

Pro nejjednodušší distribuci výukové aplikace byly zvoleny webové technologie, které nabízejí řadu výhod oproti desktopovým nebo mobilním aplikacím. Webová aplikace nevyžaduje instalaci a běží v libovolném moderním webovém prohlížeči, což zajišťuje maximální dostupnost pro uživatele napříč různými platformami a operačními systémy. Dalším přínosem je snadná údržba – aktualizace aplikace se projeví u všech uživatelů bez nutnosti instalace nových verzí koncovým uživatelem.

2.2 React a Vite

Jako hlavní framework (aplikační rámec) pro vývoj uživatelského rozhraní byl zvolen *React* [4]. React je moderní JavaScriptová knihovna vyvinutá společností Meta (dříve Facebook), která umožňuje vytvářet interaktivní uživatelská rozhraní na bázi komponent. Hlavní výhodou Reactu je koncept *reaktivity* – uživatelské rozhraní se automaticky aktualizuje při změně dat bez nutnosti manuální manipulace s DOM (Document Object Model).

React využívá deklarativní přístup k tvorbě UI. Na rozdíl od tradičního imperativního programování, kde vývojář musí krok za krokem instruovat prohlížeč, jak upravit rozhraní, v Reactu stačí popsat, jak má výsledné rozhraní vypadat, a React se postará o potřebné změny v DOM. Tento přístup výrazně zjednodušuje vývoj komplexnějších aplikací a minimalizuje chyby spojené s nekonzistentním stavem.

Pro sestavení aplikace byl použit nástroj (bundler) *Vite* [5]. Vite zajišťuje přípravu všech souborů aplikace (kódu, stylů, obrázků) pro běh v prohlížeči. Během vývoje Vite umožňuje vidět změny v kódu okamžitě po uložení souboru bez nutnosti obnovovat stránku. Pro finální verzi aplikace Vite veškeré soubory optimalizuje a zmenší pro rychlejší načítání.

2.3 Architektura a struktura kódu

Aplikace je strukturována jako *Single Page Application* (SPA), kde celá aplikace běží v rámci jedné HTML stránky a navigace mezi jednotlivými moduly probíhá bez opětovného načítání stránky. Hlavní komponenta `App.jsx` funguje jako kořen aplikační struktury a spravuje globální stav aplikace pomocí React Hooks, především `useState` pro udržení aktuální stránky a předávaných dat.

Zdrojový kód je rozdělen do tří hlavních modulů podle řešených problémů – MCVP, kombinatorické hry a bezkontextové gramatiky (viz kapitoly 3, 4 a 5). Každý modul obsahuje vlastní logiku a vizualizační komponenty a je organizován do separátních složek podle funckionality:

- **Hlavní komponentu** – kontejnerovou komponentu řídící celý modul
- **Utils** – pomocné funkce obsahující algoritmy (parsery, generátory, evaluátory)
- **InputSelectionComponents** – komponenty pro různé způsoby zadání vstupu
- **Vizualizační komponenty** – komponenty pro grafické zobrazení problémů a jejich řešení

Společné prvky jako tlačítka nebo modální okna jsou sdíleny mezi všemi moduly. Pro jendotné nastavení grafů byly vytvořeny pomocné moduly v adresáři `src/Hooks` – modul `useGraphColors` spravuje barvy použité v grafických vizualizacích a modul `useGraphSettings` obshahuje parametry pro vzhled grafů (velikost uzlů, vzdálenosti apod.).

Tato struktura umožňuje snadné úpravy na jednom místě.

2.4 Vizualizace grafů

Pro vizualizaci grafů používá aplikace knihovnu *D3.js* [6]. Grafy jsou automaticky rozmístěny – uzly se vzájemně odpuzují a hrany je přitahují k sobě. Pro stromové struktury v MCVP modulu (viz kapitola 3) jsou uzly uspořádány do úrovní shora dolů, s kořenem nahoře a listy dole.

Během vyhodnocování se uzly a hrany v grafu vybarvují podle aktuálního stavu, což uživateli umožňuje sledovat průběh výpočtu.

2.5 Stylování a responzivita

Pro vzhled uživatelského rozhraní aplikace využívá framework *Bootstrap 5* [7]. Bootstrap poskytuje předpřipravené styly pro tlačítka, formuláře a další prvky rozhraní. Použití Bootstrapu urychlilo vývoj a zajistilo jednotný vzhled.

2.6 Správa vstupů a výstupů

Každý modul aplikace nabízí tři způsoby zadání vstupu:

1. **Manuální zadání** – uživatel zadává vstup pomocí textového pole nebo grafického editoru
2. **Náhodné generování** – aplikace vygeneruje náhodný příklad podle zadaných parametrů
3. **Připravené příklady** – výběr z předpřipravených ukázkových příkladů

Aplikace umožňuje ukládání a načítání dat pomocí souborů ve formátu JSON. Soubory lze načíst přetažením do aplikace nebo výběrem ze složky. Formát nahraného souboru je validován, aby nedošlo k chybám při zpracování neplatných dat.

2.7 Notifikace a zpětná vazba

Pro zobrazení upozornění a chybových hlášek aplikace používá knihovnu *react-toastify* [8]. Notifikace se objevují v rohu obrazovky a automaticky mizí po chvíli. Aplikace zobrazuje notifikace zejména při chybách ve vstupu, úspěšném uložení dat nebo varováních při neplatných operacích.

Notifikace poskytují okamžitou zpětnou vazbu pro uživatele.

2.8 Vývojové nástroje

Pro zajištění kvality kódu aplikace používá nástroj *ESLint* [9], který kontroluje dodržování standardů a detekuje potenciální chyby v kódu.

2.9 Nasazení aplikace

Pro zveřejnění aplikace nástroj Vite připraví a optimalizuje všechny soubory. Protože veškerá logika aplikace běží přímo v prohlížeči uživatele, není potřeba žádný speciální server.

Aplikace je nasazena pomocí služby *Vercel*, která je přímo propojena s GitHub repozitářem. Při každé změně v hlavní větvi repozitáře dojde automaticky k novému nasazení aplikace. Toto řešení výrazně zjednodušuje proces aktualizace a zajišťuje, že živá verze aplikace je vždy aktuální.

Kapitola 3

Monotone Circuit Value Problem

3.1 Teoretický základ

Monotone Circuit Value Problem je základní problém v teorii složitosti, který patří mezi P-úplné problémy [3]. Jeho definice je následující:

- **Vstup:** Booleovský obvod bez negací, tvořený pouze hradly AND ($\wedge$) a OR ($\vee$), společně se vstupními proměnnými (pouze hodnoty 0 nebo 1 - false nebo true).
- **Výstup:** Hodnota výstupního uzlu obvodu (kořene stromu).

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 [1].

3.1.1 P-úplnost MCVP

MCVP je P-úplný problém [2]. Vyhodnocení monotónního obvodu lze provést v polynomiálním čase postupným vyhodnocováním uzlů od vstupů směrem k výstupu. Přestože je problém v třídě P, jeho P-úplnost naznačuje, že pravděpodobně neexistuje efektivní paralelní algoritmus pro jeho řešení – problém vyžaduje sekvenční zpracování.

3.2 Formát vstupu a parsování výrazů

Aplikace umožňuje zadat monotónní obvod několika způsoby. Jako první je textový zápis ve formě logického výrazu, který je následně převeden na stromovou strukturu.

3.2.1 Gramatika vstupního jazyka

Vstupní výrazy odpovídají jednoduché gramatice:

- 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]))`

3.2.2 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. Metody odpovídají pravidlům gramatiky: `parseOrExpr()` zpracovává OR výrazy, `parseAndExpr()` zpracovává AND výrazy a `parseFactor()` zpracovává závorky nebo proměnné. Parser vytváří uzly typu `Node`, které tvoří strom reprezentující strukturu obvodu.

Výsledkem parsování je binární strom, kde listy jsou proměnné s přiřazenými hodnotami a vnitřní uzly reprezentují logické operace.

3.3 Vyhodnocení obvodu

Vyhodnocení monotónního obvodu probíhá rekurzivním průchodem stromu implementovaným v modulu `EvaluateTree.js`.

3.3.1 Algoritmus vyhodnocení

Funkce `evaluateTree()` používá algoritmus 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 operační uzly funkce rekurzivně 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 právě tehdy, když alespoň jeden potomek má hodnotu 1.

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

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.

3.3.2 Krokové vyhodnocení

Pro vzdělávací účely aplikace nabízí krokové vyhodnocení pomocí funkce `evaluateTreeWithSteps()`. Tato funkce provádí stejný výpočet jako `evaluateTree()`, ale navíc zaznamenává každý vyhodnocený uzel spolu s hodnotami jeho potomků do pole kroků. Uživatel tak může sledovat, jak se hodnoty šíří od listů k kořeni stromu, 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.

3.4 Interaktivní editace obvodu

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

- **Přidávat uzly:** Kliknutím do prázdné oblasti grafu lze vytvořit nový uzel. Uživatel vybere typ uzlu (proměnná s hodnotou 0, proměnná s hodnotou 1, hradlo AND nebo hradlo OR).

- **Vytvářet hrany:** Táhnutím myši z jednoho uzlu do druhého vytvoří hranu reprezentující tok dat v obvodu.

- **Odstraňovat prvky:** Pravým kliknutím na uzel nebo hranu lze prvek odstranit z obvodu.

- **Reorganizovat graf:** Uzly lze přesouvat myší pro lepší vizuální uspořádání obvodu.

Grafický editor využívá knihovnu D3.js pro vykreslování a interakci. Force-directed layout automaticky rozmísťuje uzly tak, aby byl graf přehledný.

3.5 Generování náhodných obvodů

Aplikace obsahuje generátor náhodných monotónních obvodů v modulu `Generator.js`. Uživatel může specifikovat:

- **Počet proměnných:** Určuje, kolik vstupních uzlů bude obvod obsahovat.
- **Počet operací:** Určuje, kolik vnitřních uzlů (hradel) bude obvod obsahovat.
- **Strukturu:** Algoritmus zajišťuje, že výsledný obvod je platný orientovaný acyklický graf s jedním kořenovým uzlem.

Generátor pracuje následovně: nejprve vytvoří zadaný počet proměnných s náhodnými hodnotami, poté postupně přidává operační uzly. Každý operační uzel náhodně vybere dva existující uzly jako potomky a náhodně zvolí operaci (AND nebo OR). Výsledkem je náhodný, ale vždy vyhodnotitelný obvod.

3.6 Vizualizace obvodu

Vizualizace monotónního obvodu je implementována v komponentě `TreeCanvas`. Obvod je zobrazen jako stromový graf s kořenem nahoře a listy dole.

3.6.1 Hierarchické rozložení

Pro stromové struktury používá aplikace hierarchické rozložení z knihovny D3.js. Uzly jsou automaticky umístěny do vrstev podle jejich vzdálenosti od kořene. Algoritmus zajišťuje, že se hrany nekříží a strom je vizuálně vyvážený.

3.6.2 Barevné kódování

Uzly jsou rozlišeny barvami podle typu a hodnoty:

- **Proměnné s hodnotou 1:** Zelená barva indikuje pravdivou hodnotu.
- **Proměnné s hodnotou 0:** Červená barva indikuje nepravdivou hodnotu.
- **Hradlo AND:** Modrá barva reprezentuje logický součin.
- **Hradlo OR:** Oranžová barva reprezentuje logický součet.

Během krokového vyhodnocování se aktuálně zpracovávaný uzel zvýrazní silnějším ohraničením, aby uživatel mohl sledovat postup výpočtu.

3.7 Převod na kombinatorickou hru

Aplikace implementuje polynomiální redukci z MCVP na problém kombinatorických her, což demonstruje P-úplnost obou problémů. Převod je realizován v modulech `ConversionCombinatorialGame.js` a `MCVPtoCombinatorialGameConverter.jsx`.

3.7.1 Pravidla převodu

Každý uzel monotónního obvodu je namapován na pozici ve hře dvou hráčů následujícím způsobem [3]:

- **Hradlo OR:** Převedeno na pozici pro Hráče 1. Hráč 1 volí, kterou z následujících pozic (odpovídajících potomkům v obvodu) zvolí. Pokud některý potomek vede k výhře pro Hráče 1, může tuto možnost zvolit.
- **Hradlo AND:** Převedeno na pozici pro Hráče 2. Hráč 2 volí následující pozici. Aby Hráč 1 vyhrál, musí vyhrát ve všech možných pokračováních, protože Hráč 2 se snaží Hráči 1 zabránit ve výhře.
- **Proměnná s hodnotou 1:** Převedena na terminální pozici, kde Hráč 2 nemá žádné tahy – to znamená, že Hráč 2 prohrává, tedy Hráč 1 vyhrává.
- **Proměnná s hodnotou 0:** Převedena na terminální pozici, kde Hráč 1 nemá žádné tahy – to znamená, že Hráč 1 prohrává.

Tento převod zachovává výsledek: obvod vyhodnotí na 1 právě tehdy, když Hráč 1 má vítěznou strategii ve výsledné hře.

3.7.2 Implementace převodu

Třída `MCVPToGameStepGenerator` provádí rekurzivní průchod stromem obvodu. Pro každý navštívený uzel vytvoří odpovídající pozici ve hře a hrany vedoucí k pozicím odpovídajícím potomkům. Algoritmus používá memoizaci pomocí slovníku, aby zajistil, že každý uzel je zpracován pouze jednou, i když se v obvodu vyskytuje vícekrát (v případě sdílených podstromů).

Komponenta `MCVPtoCombinatorialGameConverter` vizualizuje převod krok po kroku. Na levé straně obrazovky je zobrazen původní obvod se zvýrazněným aktuálně zpracovávaným uzlem, na pravé straně je zobrazen dosud vytvořený graf hry. Uživatel může procházet kroky převodu vpřed a vzad pomocí navigačních tlačítek.

3.8 Převod na bezkontextovou gramatiku

Druhý implementovaný převod transformuje monotónní obvod na bezkontextovou gramatiku, kde problém vyhodnocení obvodu odpovídá problému prázdnosti jazyka generovaného gramatikou.

3.8.1 Princip převodu

Každý uzel obvodu je mapován na symbol gramatiky [3]:

- **Kořen obvodu:** Odpovídá počátečnímu symbolu gramatiky S.
- **Hradlo AND s potomky A a B:** Vytvoří se pravidlo tvaru $X \to AB$, kde X je neterminál odpovídající hradlu. Řetězec lze odvodit z X právě tehdy, když lze odvodit řetězce z obou A i B.
- **Hradlo OR s potomky A a B:** Vytvoří se dvě pravidla: $X \to A$ a $X \to B$. Řetězec lze odvodit z X, pokud jej lze odvodit alespoň z jednoho z A nebo B.
- **Proměnná s hodnotou 1:** Mapována na pravidlo generující terminální symbol (například $X \to \varepsilon$ nebo $X \to a$), což znamená, že z tohoto symbolu lze odvodit neprázdný řetězec.
- **Proměnná s hodnotou 0:** Mapována na neterminál bez pravidel, což znamená, že z tohoto symbolu nelze nic odvodit.

Výsledná gramatika generuje neprázdný jazyk právě tehdy, když obvod vyhodnotí na 1.

3.8.2 Implementace převodu

Třída `MCVPToGrammarConverter` (definovaná v souboru `MCVPtoGrammarConverter.jsx`) implementuje převod ve dvou fázích:

1. **Mapování uzlů na symboly:** Třída `NonTerminalGenerator` přiřazuje každému uzlu obvodu unikátní symbol gramatiky. Kořen je mapován na počáteční symbol S, ostatní uzly dostanou symboly A, B, C, atd.
2. **Generování pravidel:** Funkce `createProductionsRecursively()` rekurzivně prochází strom a pro každý uzel vytváří odpovídající gramatická pravidla podle typu uzlu (AND, OR nebo proměnná).

Pomocná třída `ConversionGrammar` (v souboru `ConversionGrammar.js`) rozšiřuje standardní třídu `Grammar` o metody pro inkrementální přidávání neterminálů, terminálů a pravidel, což usnadňuje krokovou konstrukci gramatiky během převodu.

Komponenta `MCVPtoGrammarConverter` opět umožňuje krokové sledování převodu. Uživatel vidí, jak je obvod postupně transformován na gramatiku, a může sledovat, která pravidla gramatiky vznikají z jednotlivých uzlů obvodu.

3.9 Ukládání a načítání obvodů

Aplikace podporuje export a import obvodů ve formátu JSON. Modul `Serialization.js` obsahuje funkce:

- **`treeToFlatGraph()`:** Převede stromovou strukturu obvodu na JSON objekt obsahující pole uzlů a pole hran. Každý uzel má unikátní ID, typ, hodnotu a případně přiřazenou binární hodnotu.
- **`flatGraphToTree()`:** Provádí opačný převod – ze serializovaného JSON objektu rekonstruuje stromovou strukturu s referenčními vazbami mezi uzly.

Tato funkcionalita umožňuje uživatelům ukládat zajímavé obvody pro pozdější použití nebo sdílet příklady s ostatními. Aplikace obsahuje sadu předpřipravených příkladů ve složce `Sady/MCVP`, které demonstrují různé scénáře a konfigurace obvodů.

Kapitola 4

Kombinatorické hry na grafu

Kapitola 5

Prázdnost bezkontextových gramatik

Kapitola 6

Závěr

Literatura

1. PAPADIMITRIOU, Christos H. *Computational Complexity*. Addison-Wesley, 1993. ISBN 978-0201530827.
2. ARORA, Sanjeev; BARAK, Boaz. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009. ISBN 978-0521424264.
3. SAWA, Zbyněk. *Teoretická informatika – Podklady pro přednášky* [Online]. 2024. Dostupné také z: `https://www.cs.vsb.cz/sawa/ti/slides/ti-slides-08.pdf`.
4. META PLATFORMS, INC. *React – A JavaScript library for building user interfaces* [Online]. 2024. Dostupné také z: `https://react.dev/`.
5. VITE TEAM. *Vite – Next Generation Frontend Tooling* [Online]. 2024. Dostupné také z: `https://vitejs.dev/`.
6. BOSTOCK, Mike. *D3.js – Data-Driven Documents* [Online]. 2024. Dostupné také z: `https://d3js.org/`.
7. BOOTSTRAP TEAM. *Bootstrap – The most popular HTML, CSS, and JS library in the world* [Online]. 2024. Dostupné také z: `https://getbootstrap.com/`.
8. KHADRA, Fadi. *react-toastify – React notification made easy* [Online]. 2024. Dostupné také z: `https://fkhadra.github.io/react-toastify/`.
9. ESLINT TEAM. *ESLint – Pluggable JavaScript linter* [Online]. 2024. Dostupné také z: `https://eslint.org/`.

Příloha A

Dlouhý zdrojový kód