Vývojová dokumentace – Tableaux solver

1. Formule

formula.hpp

Nejprve se načtou ze vstupu formule, každá v paměti uložená ve formě vytvořujícího stromu. Kořen stromu (načtená formule) je vlastníkem podstromu (unique_ptr na podstrom). Tato reprezentace byla zvolena proto, jelikož je jednoduchá na parsování a ve vytvořeném tablu se odkazujeme na podstromy formule (podformule).

Každá formule pak obsahuje také její stringovou reprezentaci, která se spočítá v rekurzivní funkci `set_string_representation()` Zamezí se tak opětovnému procházení stromu a skládání reprezentace formule pro její tisknutí.

Na každém typu základních formulí (kterých je šest) existuje pro tablo metodu takzvané atomické tablo. Na instanci formule se pak volá funkce `atomic_fork`. Vysvětleno v sekci algoritmu tablo metody.

formula.hpp
- formula_binary.hpp:
  - formula_conjunction.hpp
  - formula_disjunction.hpp
  - formula_equivalence.hpp
  - formula_implication.hpp
- formula_unary.hpp
  - formula_negation.hpp
- formula_variable.hpp

1a. Parser formulí

formula_parser.hpp
formula_parser.cpp

Parser získává ze vstupního stringu tokeny
`tokens formula_parser::get_formula_tokens(const std::string& unparsed_formula)`
reprezentující části vstupu.

formula_parse_token.cpp
formula_parse_token.hpp

Vstupní string rekurzivně rozkládá na menší části, které jdou rozparsovat do podformulí podle vytvořujícího stromu.

`parse_token parse_token::try_get_token_and_iterate_over(string_const_iter& it, const string_const_iter& end)`
získává tokeny sekvenčním čtením řetězce.

Pro příklad, string

```
((r IMP p) IMP s)
```

se rozloží na tři tokeny:

```
(r IMP p)
IMP
s
```

Každý token si zároveň uchovává typ tokenu:
podformule, spojka, výroková proměnná

Podle typu tokenů `formula_ptr formula_parser::assemble_formula(const tokens& tokens)` určí, zda jdou tokeny složit do formule.

První token, `(r IMP p)` je podformulí, se v tomto případě stejným způsobem rozparsuje na tokeny a vrátí se jeho rozparsovaný vytvořující strom. To, že jde opravdu o formuli, se zjistí až při jejím rozkladu na tokeny, předtím se jednalo pouze o předpoklad na základě existence závorek a jejich hloubky.

V případě, že tokeny nejdou složit do formule nebo je načten špatný token, tak `throw std::invalid_argument("Input format error");`

2. Algoritmus tablo metody a reprezentace tabla

tableaux_branch.hpp
tableaux_branch.cpp
tableaux_node.hpp

Algoritmus musí podporovat práci s více formulemi, je možné i pracovat s teoriemi.

Tablo algoritmus pracuje s větvemi tabla, každá větev má nějakou formuli, podle které se dále bude rozvíjet. Navíc si musí pro nějakou větev pamatovat, jaké výrokové proměnné se už na ní vyskytly (pro odvození sporu) a jaké axiomy teorie už byly použité. Větve tabla se musí rozvíjet postupně.

Tablo se reprezentuje stromem, pro každý list existuje instance `class tableaux_branch`, která si pamatuje výše uvedené informace. Tato třída je jakýsi observer, pouze nad stromem operuje a přes ní se do stromu přidávají další vrcholy. Vrcholy jsou reprezentovány instancemi `class tableaux_tree_node`

Použití i detailnější popis těchto struktur vysvětlen na průběhu algoritmu.

Průběh algoritmu:

Nejdříve se získá kořen tabla, který bude jeho vlastníkem (podobně jako u stromu formule, rodiče mají unique_ptr na child nody). Kořen tabla bude ukazovat na formuli (formula_ptr_) s nějakou pravdivostní hodnotou (truth_value_) z prvního řádku vstupu.

Na začátku se vytvoří počáteční větev, `initial_branch_` , která obsahuje všechny axiomy z teorie (jako instance tableaux_tree_node) a položkou, podle které se bude rozvíjet (head_), bude formula_ptr_ kořene tabla, tedy formule z prvního řádku vstupu.

Pak začně algoritmus, jehož hlavní část je BFS. Do fronty se přidá initial_branch_, poté se pokaždé provede rozvinutí větve podle položky (head_) - větev se může rozdělit (fork) nebo zůstane zase jedna, případně žádná - spor nebo žádná položka k rozvinutí na větvi už není. Nové větve se přidají zpět do fronty. Pokračuje se, dokud fronta není prázdná.

Jak probíhá fork větví:

Na formuli, podle které se bude rozvíjet, se volá polymorfický

```
virtual std::vector<tableaux_branch> atomic_fork(tableaux_branch& previous, const
tableaux_tree_node& developed) const = 0;
```

který se chová jako atomické tablo - jehož listy jsou podformule rozvíjící se formule s nějakou pravdivostní hodnotou.

Podle atomického tabla vznikají nové větve, nahrazující tu původní. Na každou novou větev přibydou do struktury formula_queue_ další nody, jejichž formule se budou dále rozvíjet. Pokud se stane, v případě proměnné, že formula_queue_ je prázdná, tak se vezme axiom z axiom_queue_. Na konci forku se pro každou novou větev z formula_queue_ vyberou nové nody, podle kterých se bude rozvíjet. Větve se tak prodlouží.

Rozhodnutí, zda formula_queue_ bude fronta nebo zásobník záleží pouze na tom, v jakém pořadí chceme vybírat další nody pro rozvinutí - toto pořadí je důležité pouze pro čitelnost - v tomto případě lepší stack.

Reprezentace tabla na vstupu

tableaux_printer.cpp
tableaux_printer.hpp

Tablo je reprezentováno klasicky jako strom, pro správné vypsání na konzoli nebo do souboru se vychází ze znalosti horizontálních délek podstromů při tisknutí nějaké položky.

Každý node tabla má proměnné:

```
size_t subtree_horizontal_length_;
size_t print_start_index_;
size_t print_mark_index_;
```

'Mark' je myšlena čára '|' spojující dva vrcholy při vypsání.

Nejdříve se určí délky podstromů,

```
static size_t calculate_subtree_block_length(tableaux_tree_node* root);
```

pak počáteční indexy formulí pro výpis na řádce (tiskne se do délky formule s režií navíc pro tablo položku)

```
static void
calculate_formula_line_offsets(tableaux_tree_node*
root, size_t start, size_t end);
```

a pak se určí marky (při tisknutí položek s jedním synem chceme mít zarovnanou marku na stejné pozici, takže syn pozici marky dědí, až na případy, kdy jde o list tabla).

```
static void calculate_mark_line_offsets(tableaux_tree_node* root);
```

Vypsání probíhá BFS průchodem stromu, vypisováním vždy na tři řádky:
1. formule
2. marky
3. propojení dvou synů (vezmou se pozice marek a mezi nimi se vypíše "--------------"

V prvním průchodu se vytiskne kořen, v druhém jeho děti, atd.p