Дополнительные главы матлогики

Структуры данных в алгоритмах DPLL и CDCL

Егор Орачев

Санкт-Петербургский Государственный университет

13 декабря, 2021

Терминология

- **Объекты**
  - Переменная x
  - Литерал l это либо переменная x, либо ее отрицание $\overline{x}$
  - Кляуза (англ. clause) это конечное множество литералов $\omega = \{l_1, ..., l_n\}$
  - Формула это конечное множество кляуз $\phi = \{\omega_1, ..., \omega_m\}$
- **Интерпретация**
  - Присвоенное переменной значение $\upsilon(x) \in \{0,\ 1\}$, $\upsilon(\overline{x}) = 1 - \upsilon(x)$
  - Кляуза $\omega = \{l_1, ..., l_n\}$ выполнена, если хотя бы для одного литерала $\upsilon(l_i) = 1$
  - Формула $\phi = \{\omega_1, ..., \omega_m\}$ выполнена, если все ω_i выполнены для υ

Алгоритм

1. Поиск поиск переменной для присваивания (**decision process**).
2. Получение логических следсвий из присваивания (**implication process**)
3. Отмена присвиваний и откат к раннему состоянию (**backtracking process**).

```
1:  function DPLL(F)
2:      while F includes a clause C such that |C| ⩽ 1 do
3:          if C = ∅ then return UNSATISFIABLE
4:          else if C = {v} then F = F|v
5:      end while
6:      if F = ∅ then return SATISFIABLE
7:      Choose a literal u using a branching rule
8:      if DPLL(F|u) = SATISFIABLE then return SATISFIABLE
9:      if DPLL(F|ū) = SATISFIABLE then return SATISFIABLE
10:     return UNSATISFIABLE
11: end function
```

Figure: Общая структура dpll алгоритма (изображение из работы [6])

Проблемы

- Как эффективно удалять кляузы?
- Как эффективно искать *unit*-кляузы?
- Как эффективно удалять литералы из кляуз?
- Как эффективно откатывать изменения?
- Как эффективно хранить граф импликаций?
- ...

Структура презентации

1. Adjacency lists
    1. Assigned literal hiding
    2. The counter-based approach
    3. Counter-based with satisfied
    4. Satisfied clause and assigned literal hiding
2. Lazy data structures
    1. Sato's head/tail lists
    2. Chaff 's watched literals
    3. Head/tail lists with literal sifting
    4. Watched literals with literal sifting
3. Tries
4. Детали реализации

1. Adjacency lists

- Каждая кляуза хранит список литералов, которые в нее входят
- Каждый литерал хранит список кляуз, в которых он встречается
- *Adjacency* т.е. литералы и кляузы смежны (что достигается засчет списков)
- В общем случает *adjacency list* — когда литерал имеет **полный** список кляуз, в которые он входит

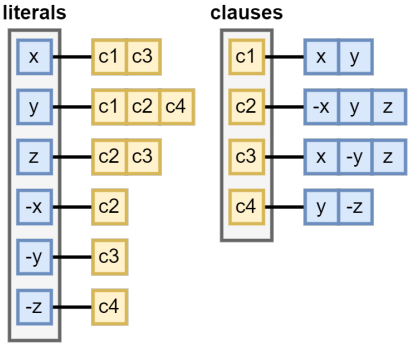

Figure: Списки смежности для формулы $\{x, y\}, \{\overline{x}, y, z\}, \{x, \overline{y}, z\}, \{y, \overline{z}\}$

1.1 Assigned literal hiding

- **Идея**: разделить литералы внутри кляузы на отлельные списки
- При присваивании значения литералу, перемещать его в соответсвующий списов
- **Backtracking**: необходимо применить операции в обратном порядке
- *На практике всегда уступает отсальным структурам, поэтому в явном виде не используется*

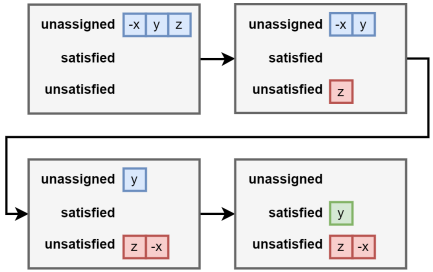

Figure: Пример использования для $\{\overline{x}, y, z\}$, последовательность присваиваний $z \to 0$, $\overline{x} \to 0$, $y \to 1$

1.2 The counter-based approach

- **Идея**: вести только подсчет литералов каждого вида внутри кляузы
- Завести три счетчика: **total**, **satisfied**, **unsatisfied**
- Обновлять счетчики, когда литералу присвоено значение
- Необходимо итерироваться по всему списку литералов, чтобы найти *unit-литерал*
- **Backtracking**: необходимо вернуть счетчики в изначальное состояние
- **Применение**: GRASP

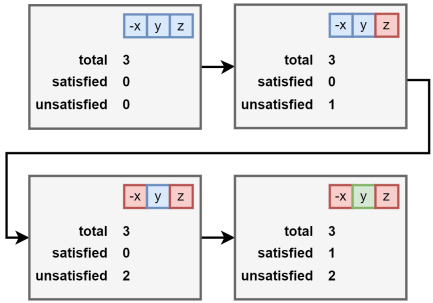

Figure: Пример использования для $\{\overline{x}, y, z\}$, последовательность присваиваний $z \to 0$, $\overline{x} \to 0$, $y \to 1$

1.3 Counter-based with satisfied clause hiding

- **Проблема**: список кляуз большой (и может расти)
- **Мотивация**: при очерендном присваивании литералу значения интересуют только невыполненные еще кляузы
- **Идея:** удалять (прятать) выпленную кляузу из спиков всех литералов
- **Backtracking**: необходимо восстановить списки
- **Применение**: Scherzo covering problem
- *На практике недостаточно эффективно в сравнении с простым counter-based подходом*

1.4 Satisfied clause and assigned literal hiding

- **Проблема**: поиск unit-литерала
- **Мотивация**: при очерендном присваивании литералу значения хотим найти новый unit-литерал в кляузах
- **Идея:** удалять (прятать) невыполненные литералы внутри кляузы
- **Backtracking**: необходимо восстановить литералы
- *На практике недостаточно эффективно в сравнении с простым counter-based подходом*

2. Lazy data structures

- **Проблема**: список кляуз большой (и может расти)
- **Идея**: когда литералу присвоено значение, не во всех кляузах происходит что-то *интересное*
- Каждая кляуза хранит список литералов, которые в нее входят
- Каждый литерал хранит **сокращенный** список кляуз, в которых он встречается
- В общем случает *lazy data structures* — когда литерал имеет **сокращенный** список кляуз

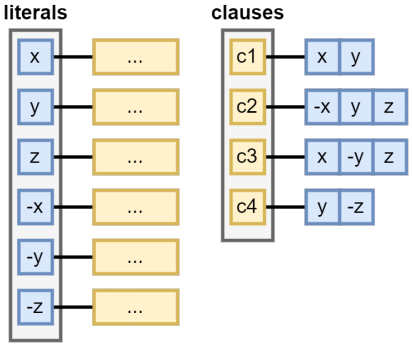

Figure: Представление формулы $\{x, y\}, \{\overline{x}, y, z\}, \{x, \overline{y}, z\}, \{y, \overline{z}\}$

2.1 Sato's head/tail lists

- Два указателя внутри каждой кляузы: *head (H)* и *tail (T)*, изначально указывают на начало и конец кляузы соответсвенно
- У каждого литера есть два списка кляуз: где он *head* и *tail* соответсвенно
- Каждый раз, когда литералу присвоено значение, новые *head* или *tail* необходимо найти
- **Backtracking**: необходимо восстановить все указатели
- **Применение**: SATO SAT solver

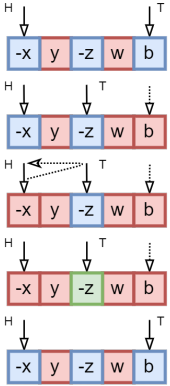

Figure: Пример использования для $\{\overline{x}, y, \overline{z}, w, b\}$, последовательность присваиваний $y \to 0$, $w \to 0$, $b \to 0$, $\overline{x} \to 0$, $\overline{z} \to 1$

2.2 Chaff 's watched literals

- **Идея**: важные состояния - 0 свободных литералов, 1 или много
- Два указателя в каждой кляузе на литерал
- У каждого литера есть список кляуз, где он *watched*
- Каждый раз, когда литералу присвоено значение, новый *watched*-литерал необходимо найти
- **Backtracking**: ничего не надо делать
- **Применение**: Chaff & MiniSat

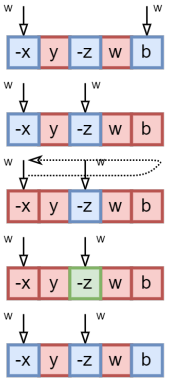

Figure: Пример использования для $\{\overline{x}, y, \overline{z}, w, b\}$, последовательность присваиваний $y \to 0$, $w \to 0$, $b \to 0$, $\overline{x} \to 0$, $\overline{z} \to 1$

2.3 Head/tail lists with literal sifting

- **Проблема**: проблематично откатывать изменения
- **Идея**: переупорядочивать литералы в соответсвии с уровнем дерева решения
- Требуются всего 4 указателя на литералы внутри кляузы
- *sifting* - просеивание
- **Backtracking**: отодвинуть *head/tail* указатели влево/вправо соответсвенно

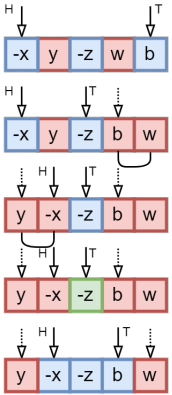

Figure: Пример использования для $\{\overline{x}, y, \overline{z}, w, b\}$, последовательность присваиваний $y \to 0$, $w \to 0$, $b \to 0$, $\overline{x} \to 0$, $\overline{z} \to 1$

2.4 Watched literals with literal sifting

- **Проблема**: необходимо итерироваться по всем литералам
- **Идея**: переупорядочивать литералы в соответсвии с уровнем дерева решения
- Требуются всего 4 указателя на литералы внутри кляузы
- Когда литералу присвоено значение, надо двигать указатель только в пределах неотсортированных регионов
- **Backtracking**: ничего не надо делать

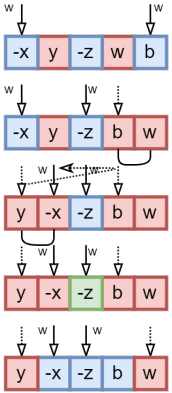

Figure: Пример использования для $\{\overline{x}, y, \overline{z}, w, b\}$, последовательность присваиваний $y \to 0$, $w \to 0$, $b \to 0$, $\overline{x} \to 0$, $\overline{z} \to 1$

3. Tries

- **Идея**: использовать **trie** или префиксное дерево для представления формулы
- **Структура**:
  - Узел дерева nil
  - Улез дерева $\square$
  - Узел дерева $\langle v, pos, neg, rest \rangle$
  - Путь от вершины до $\square$ задает кляузу
- **Интерпретация**:
  - Формула $S = P \cup N \cup R$
  - $P = \{v \cup P_1, ..., v \cup P_n\}$
  - $N = \{\overline{v} \cup N_1, ..., \overline{v} \cup N_m\}$
  - $R = \{R_1, ..., R_l\}$

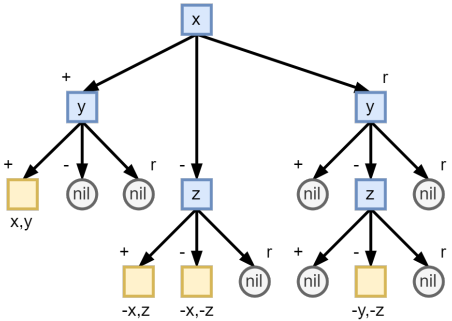

Figure: Пример дерева для формулы $\{x, y\}, \{\overline{x}, z\}, \{\overline{x}, \overline{z}\}, \{\overline{y}, \overline{z}\}$

4. Детали реализации (1)

- **Гибридный подход**: adjacency lists, counters, assigned literal и satisfied clause* hiding
- Каждый литерал дополнительно хранит список позиций, на которых он встречается в каждой кляузе
- Использование бит-вектора для literal hiding
- Максимальный размер кляузы 32 литерала $\{a, b\} \to \{a, u\}, \{\overline{u}, b\}$

```
typedef struct literal_info{
  int is_assigned;
  int n_occur;
  int * lit_in_clauses;
  int * lit_in_clause_locs;
  int is_unit;
  int antecedent_clause;
}literal_info;

literal_info linfo[MAX_VARS][2];
```

```
typedef struct clause_info{
  int * literals;
  int current_length;
  int original_length;
  int is_satisfied;
  int binary_code;
  int current_ucl;
}clause_info;

clause_info * clauses;
int n_clause, r_clauses;
```

Figure: Структуры для описания информации о литералах и кляузах (изображения из работы [6])

4. Детали реализации (2)

- Информация о выполненых кляузах и о тех, в которых вычеркнули литерал
- Для каждого уровня присваивания литералу значения — количесвто выполненых кляуз и тех, из которых вычеркнули литерал
- Используется для **backtracking**
- Для каждой переменной информация о присваивании значения
- Используется для **non-chronological backtracking** и обратного bfs-обхода для **clause learning**

```
typedef struct changes_info{
  int clause_index;
  int literal_index;
}changes_info;

changes_info changes[MAX_CLAUSES];
unsigned int changes_index;
unsigned int n_changes[MAX_VARS][2];
```

```
typedef struct assign_info{
  int type;
  int depth;
  int decision;
}assign_info;

assign_info assign[MAX_VARS];
```

Figure: Структуры для описания стека изменений и информации о присваиваниях значений (изображения из работы [6])

Материалы презентации

1. Zhang, Hansong and Mark E. Stickel. (1994). “Implementing the Davis-Putnam Algorithm by Tries.” Journal of Automated Reasoning
2. João P. Marques Silva and Karem A. Sakallah. (1997). GRASP — a new search algorithm for satisfiability. In Proceedings of the 1996 IEEE/ACM international conference on Computer-aided design (ICCAD '96). IEEE Computer Society, USA, 220–227.
3. Lintao Zhang and Sharad Malik. 2002. The Quest for Efficient Boolean Satisfiability Solvers. In "Proceedings of the 14th International Conference on Computer Aided Verification". Springer-Verlag, Berlin, Heidelberg, 17–36.
4. Lynce, I. and Marques-silva, J.. (2002). Efficient Data Structures for Fast SAT Solvers.
5. Eén, N. and Sörensson, N. (2003). An Extensible SAT-solver. SAT.
6. Ahmed, T. (2009). An Implementation of the DPLL Algorithm.