SATソルバを作ってみる

計算機と記号論理学

ゆかたゆ

注意点

- 理論の解説ではありません
- 「作ってみた」動画です

- 説明が不十分な点があります

分かる人向けの説明

- 前半では CNF-SAT を作ります
- 後半では再帰下降構文解析をします
- 古典論理の完全性と健全性を暗に仮定します

SATソルバとは

- SAT … satisfiability problem
- 論理パズルを解くプログラム
  - 数独
  - 有界モデル検査

- NP完全

- SATソルバの性能が上がる
  → NP問題全ての計算が高速に

準備

言葉の意味①

- 真　　…「正しい」こと
- 偽　　…「間違っている」こと

- 命題　　…真か偽で答えられる文

- 命題変数…真か偽を代入できる変数

言葉の意味 ②

- 充足可能 … 場合によって，または常に成立する
- 恒偽　　… 絶対に成立しないこと

- 矛盾　　… 「A かつ Aではない」状態

記号の意味①

- ¬A　　… A ではない

- A ∧ B … A かつ B
  - 共通部分の記号 ∩ と似ている
  - 「連言」ともいう

- A ∨ B … A または B
  - 和集合の記号 ∪ と似ている
  - 「選言」ともいう

記号の意味 ②

- A → B … A ならば B
  - 「Aの時は必ずB」という意味
  - ¬A ∨ B と同じ
- A ↔ B … A と B は等しい
  - (A → B) ∧ (B → A) の略

括弧の付け方

- 必要に応じてつける
- 括弧の中身を先に考える

- 優先度
  - ¬
  - ∧，∨
  - →，↔

注意点

- 二重線の矢印は特別な意味を持つので区別する
  - ⇒ や ⇔ は今回は使わない

論理式の例

- A
- $A \lor B \lor C$
- $(A \land B) \lor A \rightarrow (D \lor (E \leftrightarrow F))$

連言標準形

連言標準形①

- 全ての論理式は「連言標準形」にできる
- 変換方法は後半で解説
- 計算機で扱いやすい
  - 詳しく知りたい人は「エルブランの定理」を参照

連言標準形 ②

- 命題変数 … 真か偽を代入できる変数
  - A，B
- リテラル … 命題変数 か，その否定
  - ¬A ，B

連言標準形 ③

- 節 … リテラルを「または」でつないだもの
  - A，¬B∨C，A∨B∨¬C
- 連言標準形 … 節を「かつ」でつないだもの
  - A，A∧(B∨C)，(¬B∨C)∧(A∨B)

実装（CNF-SAT）

DPLL ① / 概要

- DPLL … アルゴリズムの名前
- 最新のアルゴリズムもこれがベース

- 「充足可能」か「恒偽」が導かれるまで
  条件を突き合わせていく

DPLL ② / 手順

- 手順 (DPLL)
  - リテラル1つだけの節がある場合 … それを L とする
    - L を含む節は常に真なので除去
    - L の否定は常に偽なので，全ての節から除去
    - 繰り返す
  - 適当にリテラルを選ぶ … それをLとする
    - L が真の仮定でDPLLをする
    - L が偽の仮定でDPLLをする

DPLL ③ / 実行例

- A ∧ (A ∨ ¬B) ∧ (¬A ∨ C)
  - リテラルとして A を選ぶ
  - A は常に真なので削除する
  - (A ∨ ¬B) は常に真なので削除する
- (¬A ∨ C)
  - ¬A は常に偽なので削除する
- C
  - リテラルとしてCを選ぶ
  - C は常に真なので削除する

DPLL ④ / 結果

- DPLL は最終的に下記のどちらかになる
  - $A \wedge \neg A$
  - $\square$ (空)

- それぞれ 恒偽，充足可能 に対応

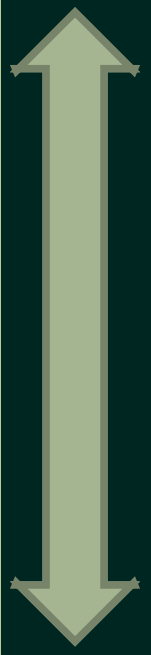
DPLL ⑤ / 停止性

- DPLL は必ず停止する
  - 1つ目の操作 → 全ての節の長さが1以上減る
  - 2つ目の操作 → 再帰後に必ず1つ目の操作が可能

プログラミング

構文解析

言語階層



- 正規言語 …有限オートマトンで受理
  - 正規表現で表せる言語
- 文脈自由言語 … プッシュダウン・オートマトンで受理
  - 今回扱う
- 文脈依存言語 … 線形拘束オートマトンで受理
- 帰納的可算言語 … チューリングマシンで受理

- 詳しくは各自調べましょう

文脈自由言語

- 文脈自由言語にもいろいろある
  - LL(1)，LR(0)，LALR，……
- 命題論理の構文は LL(1)
  - 再帰下降構文解析 という方法が使える
  - 説明すると長くなるので今回は説明を省く

命題論理の文法

記号	意味
::=	「とは」
¦	「または」

- 〈リテラル〉::=〈原子論理式〉¦ ¬〈リテラル〉¦ (〈式〉)
- 〈連言〉::=〈リテラル〉¦〈連言〉∧〈リテラル〉
- 〈選言〉::=〈連言〉¦〈選言〉∨〈連言〉
- 〈式〉::=〈選言〉¦〈選言〉→〈式〉

連言標準形への変換（例）

- 「「「Aでない」かつB」またはC」
  - 「Aではない」を D と置く
- 「「DかつB」またはC」
  - 「DかつB」を E と置く
- 「EまたはC」

- 変換後
  - 「Aの否定」＝ D
  - 「DかつB」 ＝ E
  - 「EまたはC」 ＝ 真

命題論理の変換規則 ①

- 〈リテラル: L_1〉 ::= 〈原子論理式: A〉 ¦
  ¬〈リテラル: L_2〉 ¦
  (〈式: E〉)

- 変換規則
  - 〈原子論理式: A〉 → A
  - ¬〈リテラル: L〉 → $(L_1 \vee L_2) \wedge (\neg L_1 \vee \neg L_2)$
  - (〈式: E〉) → E

命題論理の変換規則 ②

- 〈連言: C1〉 ::=〈連言: C2〉 $\underbrace{\wedge 〈\text{リテラル: L}〉}_{\text{任意回}}$
- $(C1 \vee \neg C2 \vee \neg L) \wedge (\neg C1 \vee C2) \wedge (\neg C1 \vee L)$

命題論理の変換規則 ③

- 〈選言: D1〉 ::= 〈選言: D2〉 ∨ 〈連言: C〉

任意回

- (¬D1 ∨ D2 ∨ C) ∧ (D1 ∨ ¬D2) ∧ (D1 ∨ ¬C)

命題論理の変換規則 ④

- 〈式〉::=〈選言〉¦〈選言〉→〈式〉
- A → B を ¬A ∨ B に変換した後は同様

プログラミング

動作確認

- 今回作ったソルバはあまり性能が高くない
- 簡単な論理パズルを解いてみる
  - Aさんは「BさんとCさんは嘘つきです」と言っています。
  - Bさんは「Aさんは嘘つきです」と言っています。
  - Cさんは「Bさんは嘘つきです」と言っています
- $(A \leftrightarrow \neg B \land \neg C) \land (B \leftrightarrow \neg A) \land (C \leftrightarrow \neg B)$

最後に

発展

- もっと高速なSATがありふれている
  - MiniSAT： コードが読みやすい， C++
- DPLLに「学習」という要素を追加すると速くなる
  - CDCL (Conflict-driven clause learning)
- 他にも色々…
  - 監視リテラル [Moskewicz+ 01]
  - リスタート戦略 [Gomes+ 98, Luby+ 93, Audemard+ 12]
  - 部分解キャッシング[Pipatsrisawat+ 07]

学習 ① / 動機

- DPLLの手順 (復習)
  - リテラル1つだけの節がある場合〈多項式オーダー〉
    - L を含む節は常に真なので除去
    - L の否定は常に偽なので，全ての節から除去
  - 適当にリテラルを選ぶ〈指数オーダー〉
    - L が真の仮定でDPLLをする (再帰)
    - L が偽の仮定でDPLLをする (再帰)
- なるべく多項式オーダーに近づけたい

学習 ② / 例

- (A ∨ ¬B ∨ Z) ∧ (¬A ∨ Z) ∧ ……
- 仮に A=真，B=真，……，Y=真，Z=偽 → 矛盾
  - これはAの割り当てと無関係に起こる
- 学習節として (¬B ∨ Z) を追加
  - 今後は B=真 の時点で即座に Z=真 と分かる
- 一般に，(A ∨ p ∨ q) ∧ (¬A ∨ r ∨ s) で p=q=r=s=偽
  - (p ∨ q ∨ r ∨ s) を学習

学習 ③ / バックジャンプ

- 概念のみ説明
  - A=偽，B=偽 を仮定
  - 再帰の深い場所で A∨B を学習
  - 探索を打ち切り， B=偽 を仮定した場所まで戻る

- アルゴリズムの直感が無いと理解しづらいと思います

学習 ④ / 追記

- 学習節は短いほど価値がある
- (実装する時間が無かった)
  - MiniSATを見ましょう

終わり

サンプルコード

前半： https://github.com/yukatayu/Rustic_SAT_Solver/

後半： https://github.com/yukatayu/Rustic_PL_Solver/