Logic and Computability

BLG 345E

Project #4

Res. Assist. Ali Esad Uğur
ugura20@itu.edu.tr

Instructor:
Asst. Prof. Mehmet Tahir Sandıkkaya

Faculty of Computer and Informatics Engineering
Department of Computer Engineering

1. Search Engine (DPLL Core)

1.1. Detailed Description

1.1.1. Goal and Execution Flow

The primary goal is to implement the core DPLL search function that systematically explores the space of possible variable assignments. Your function will recursively select an unassigned variable (a "decision") and branch on its truth value, relying on the Inference Engine (Project #3) to prune the search space after each step. For each decision level, your Search Engine must call the Inference Engine with a decision (assignment). You will be provided with a mock Inference Engine to be able to complete your task.

The DPLL recursive process should operate as follows:

1. **Run Inference:** Call the Inference Engine to perform Boolean Constraint Propagation (BCP) on the current partial assignment.
2. **Check Status:** Analyze the result returned by the Inference Engine.
    - If SAT is returned, a satisfying assignment is found. Halt and return success.
    - If CONFLICT is returned, the current branch is contradictory. Backtrack (return failure) to the previous decision point.
3. **Make Decision (Branching):** If CONTINUE is returned, select an unassigned variable and assign it a truth value.
4. **Recurse:** Call the DPLL function recursively with the new assignment. If the recursive call fails, undo the assignment and try the opposite value (the other branch).

1.1.2. Decision Making and Heuristics

The choice of which unassigned variable to pick next (Decision Variable Selection) is critical for performance. Your module must implement a non-trivial **variable ordering heuristic**. Simple chronological ordering is insufficient.

- **Required Heuristic:** Implement a strategy such as **Maximum Occurrences in Minimum-Sized Clauses (MOM)**, or **Jeroslow-Wang (JW)**. These heuristics prioritize variables that appear frequently in short, critical clauses, aiming to satisfy them early and trigger strong BCP.

1.1.3. Trace Integration and Decision Level Management

To support the backtracking mechanism in Project #5, your module must maintain and contribute to the **Complete Ordered Trace**.

1. **Decision Levels:** Every time the Search Engine makes a decision, a new Decision Level is created. The engine must track the current decision level and include it in the trace record for every decision.
2. **Trace Recording:** Every assignment made by the Search Engine must be recorded as a "Decision Premise" in the same trace structure used by Project #3. This allows the final analysis to distinguish between assignments made by search (decisions) and assignments made by deduction (propagations).
3. **Backtracking:** When a conflict forces a return, your engine must cleanly unwind all assignments made at the current decision level and all subsequent propagated assignments, restoring the variable states and the Watch Lists (Project #2) to the state just before the last decision.

1.1.4. Output Structure

The DPLL function must ultimately return a complete solution or a proof of unsatisfiability, encapsulated in a comprehensive data structure.

- **If SAT:** Return the final, complete assignment model.
- **If UNSAT:** Return the CONFLICT status from the initial call, along with the complete trace that led to the final conclusion that no satisfying assignment exists.

1.2. Output and Integration Specification

This engine has multiple input/output levels.

1. **BCP Trigger Input:** A txt file with two lines that is **outputted** by the Search Engine. The line are namely, TRIGGER_LITERAL and CURRENT_DL. This will be the input for Project #3. Generated before every call to the Inference Engine, specifying the new assignment (decision).

```
# --- BCP TRIGGER INPUT ---
TRIGGER_LITERAL: 1
DL: 1
```

2. **BCP Output:** A txt file that is an **input** for the Search Engine. It shows the result of the Inference Engine (Project #3) for a decision level.

```
--- STATUS ---
STATUS: CONTINUE
DL: 0
CONFLICT_ID: None

--- BCP EXECUTION LOG  ---
[DL 0] UNIT       L=-2   | C3
[DL 0] ASSIGN     L=-2   |
[DL 0] SHIFT      L=2    | C1 2->3
[DL 0] UNIT       L=-3   | C2
[DL 0] ASSIGN     L=-3   |
[DL 0] SATISFIED  L=-3   |

--- CURRENT VARIABLE STATE ---
1     | UNASSIGNED
2     | FALSE
3     | FALSE
```

3. **Final Output - Master Trace:** A txt file that is an **output** of the Search Engine. This file will be used by Project #5. This trace is the chronological record of every event — Decision (P#4), Propagation (P#3) and Conflict (P#3) — appended throughout the entire search. This is the official execution log used by Project #5 to generate the final proof. This can be created with combining the BCP Output files for each decision level.

2. Sample Run

2.1. Inputs

In the flow of the whole SAT solver, the output of Project #2 — the parsed CNF — is passed to the Inference Engine (Project #3) for assessing decision level 0. This means whether there is a unit clause to be propagated or not. If not, the Search Engine (your module) will receive the BCP Output file with the Current Variable States set as UNASSIGNED. If some propagation are done, the Search Engine receives an input accordingly.

2.2. Outputs

As mentioned above, there are mainly two outputs. The intermediate output that is the input for Project #3, and the final output that is the input for Project #5.

2.3. Execution Trace

We will trace the formula: $(\neg A \vee B) \wedge (\neg B \vee \neg C) \wedge (C \vee A) \wedge (\neg B \vee C)$, which is SATISFIABLE under assignments $A \leftarrow 1, B \leftarrow 1, C \leftarrow 0$. To showcase backtracking, we will map the literals numerically in alphabetical order. Note that the decisions made in this execution trace may not be correct. They are set just to showcase the different situations.

1. **Initial State:**

   The search begins with all variables unassigned, since there are no unit clauses (DL 0).

```
--- STATUS ---
STATUS: CONTINUE
DL: 0
CONFLICT_ID: None

--- BCP EXECUTION LOG  ---

--- CURRENT VARIABLE STATE ---
1    | UNASSIGNED
2    | UNASSIGNED
3    | UNASSIGNED
```

2. **Execution Phase 1: Search and Inference (DL 1):**

(a) Action: Search Engine selects variable B(2) using its heuristic.

(b) Intermediate Output:

```
# --- BCP TRIGGER INPUT ---
TRIGGER_LITERAL: 2
DL: 1
```

(c) BCP Result: Inference Engine runs BCP and returns the result.

```
--- STATUS ---
STATUS: UNSAT
DL: 1
CONFLICT_ID: 4

--- BCP EXECUTION LOG (Clause Status Changes & Assignments) ---
[DL1] DECIDE     L=2    |
[DL1] UNIT       L=-3   | C2
[DL1] ASSIGN     L=-3   |
[DL1] CONFLICT          | Violation: C4
[DL1] BACKTRACK         |


--- CURRENT VARIABLE STATE AFTER BCP ---
1    | UNASSIGNED
2    | TRUE
3    | FALSE
```

3. **Execution Phase 2: Backtrack the Conflict and New Decision (still DL 1):**

(a) Action: Since the decision ended in a conflict, the Search Engine needs to backtrack to the previous position, which is all literals UNASSIGNED, and make a new decision. Since the solver moved backwards, the decision level does not change in this case.

(b) Intermediate Output:

```
# --- BCP TRIGGER INPUT ---
TRIGGER_LITERAL: -2
DL: 1
```

(c) BCP Result: Inference Engine runs BCP and returns the result.

```
--- FINAL STATUS ---
STATUS: SAT
DL: 1
CONFLICT_ID: None

--- BCP EXECUTION LOG (Clause Status Changes & Assignments) ---
[DL1] DECIDE     L=-2   |
[DL1] UNIT       L=-1   | C1
[DL1] ASSIGN     L=-1   |
[DL1] UNIT       L=3    | C3
[DL1] ASSIGN     L=3    |
[DL1] SATISFIED         |

--- CURRENT VARIABLE STATE AFTER BCP ---
1    | FALSE
2    | FALSE
3    | TRUE
```

4. **Execution Phase 3: Finalize and Create the Execution Trace:**

   (a) Action: Reads the status value from the input and finalizes the execution. For the execution trace, the engine combines the execution logs that it recieved from Inference Engine throughout the process.

   (b) Search Engine Result:

```
STATUS: SAT

--- FINAL VARIABLE STATE ---
1    | FALSE
2    | FALSE
3    | TRUE
```

3. Deliverables for Successful Completion

3.1. Code Module

The complete implementation for the Inference Engine module must be provided. The code should be written neatly and heavily commented for clarifications.

3.2. Technical Report (at least 5 pages)

A technical document detailing your implementation choices:

1. **DPLL Algorithm Flow:** Provide pseudocode or a detailed flowchart for the classes and functions you implemented, specifically highlighting the steps for checking the other watched literal and searching for a replacement.

2. **Decision Heuristics:** Explain the heuristic you used for making decisions in detail and with reasons.

3. **Conflict Reporting:** Detail how the final conflicting clause ID is correctly identified and performed backtracking to the previous decision level.

3.3. Test Suite

You also need to generate new test cases in addition to the one provided to you. These test cases must be covering edge cases as well as regular runs such as CONFLICT and UNSAT situations. The results of your tests should be included and discussed in your report.