Thomas Stüber (3750920)
Benjamin Coban ()

Tübingen, den 21. Oktober 2017

Übungen zur Vorlesung "SAT-Solving und Anwendungen"

(Abgabe 4)

Aufgabe 4.1

1. Die Idee ist hier, dass während dem berechnen der gelernten Klausel jede Klausel aus der original Formel in der Menge OC gespeichert wird, welche dann als Attribut für die gelernte Klausel gesetzt wird. Hierbei muss beachtet werden, dass die Reason-Clauses der einzelnen Variablen selbst gelernte Klauseln sein könnten. Daher wird unterschieden ob die Reason-Clause der jeweiligen Variablen gelernt wurde, in dem Fall werden ihre origClauses hinzugefügt, oder ob es sich um eine Klausel aus der Eingabe handelt, in dem Fall wird die Klausel einfach hinzugefügt. Dies ist nötig, da nur Klauseln aus der Eingabe getrackt werden sollen. Pseudocode:

Input: an empty clause ec, the set of clauses C
Output: backtracking level modified
if *current decision level* == 0 **then**
| return −1
end
lv = the last assigned variable before the conflict cause;
reason = the reason of lv;
newClause = resolve(ec,reason);
if *ec is learned clause* **then**
| $OC = OC \cup ec.originClauses$
else
| $OC = OC \cup \{ec\}$
end
if *reason is learned clause* **then**
| $OC = OC \cup reason.originClauses$
else
| $OC = OC \cup \{reason\}$
end
while *stop criterion not yet met* **do**
| cv = chooseLit(newClause);
| reason = reason of cv;
| newClause = resolve(newClause,reason);
| **if** *reason = learned clause* **then**
| | $OC = OC \cup reason.origClauses$
| **else**
| | $OC = OC \cup \{reason\}$
| **end**
end
newClause.originClauses = OC;
$C = C \cup \{newClause\}$;
level = computeBacktrackingLevel(newClause);
return level;

Algorithm 1: modified analyzeConf(ec,C)

2. Hier ist die Idee, dass eine Formel nur dann unerfüllbar ist, wenn es einen Konflikt auf Level 0 gibt. Die letzte gelernte Klausel enthält in origClause die beiden Klauseln die sich wiedersprochen haben (bzw. deren origClauses falls diese gelernt waren) und alle Reason-Clauses die zu der gelernten Klausel geführt haben. Damit ist in origClause der gelernten Klausel schon der Grund enthalten, warum eine UP-Variable umbelegt werden müsste (was den Widerspruch darstellt). Nimmt man nun noch alle Reason-Clauses (bzw deren origClauses falls es gelernte Klauseln sind) der Variablen auf Level 0 hinzu erhält man den Grund, warum alle Variablen auf Level 0 durch UP so belegt sein müssen wie sie belegt sind. Zusammen ergibt sich also eine Menge von Klauseln, die unerfüllbar ist, da sie die Klauseln enthält, die festlegen warum eine Variable durch UP so belegt werden muss wie der Algorithmus sie belegt hat, und gleichzeitig die Klauseln enthält, die zu dem Widerspruch geführt haben dass diese Variable umbelegt werden müsste. Pseudocode:

Input: Clauseset C
Output: SAT or (UNSAT,UC)
level = 0;
$\alpha = \emptyset$;
while *true* **do**
  UP(C, α);
  if C *contains an empty clause* **then**
    ec = empty clause;
    level = analyzeConf(ec,C);
    if *level* == -1 **then**
      UC = lastLearnedClause.origClauses;
      forall $(v \mapsto b) \in \alpha$ **do**
        if *reason(v) is learned clause* **then**
          $UC = UC \cup reason(v).origClause$;
        else
          $UC = UC \cup \{reason(v)\}$
        end
      end
      return (UNSAT,UC);
    end
    backtrack(level);
  end
  else
    if $\alpha \vDash C$ **then**
      return SAT
    end
    level++;
    choose $x \notin \alpha$;
    $\alpha = \alpha \cup [x \mapsto 0]$;
  end
end

Algorithm 2: sat(C) modified

Aufgabe 4.2
Siehe Quelltext in diesem Archiv, alles relevante wurde kommentiert.

Aufgabe 4.3
Im folgenden die Ausgabe für die beiden Testformeln, jeweils vor und nach UP:

```
{ 1 2 -3 }, sat = false, unassigned = 3
{ 3 4 }, sat = false, unassigned = 2
{ -1 }, sat = false, unassigned = 1
{ -2 1 }, sat = false, unassigned = 2
{ 2 3 -4 }, sat = false, unassigned = 3
```

```
Variable 1: [OPEN
Adjacence List: [{ 1 2 -3 }, sat = false, unassigned = 3,
{ -1 }, sat = false, unassigned = 1,
{ -2 1 }, sat = false, unassigned = 2]
]
```

```
Variable 2: [OPEN
Adjacence List: [{ 1 2 -3 }, sat = false, unassigned = 3,
{ -2 1 }, sat = false, unassigned = 2,
{ 2 3 -4 }, sat = false, unassigned = 3]
]

Variable 3: [OPEN
Adjacence List: [{ 1 2 -3 }, sat = false, unassigned = 3,
{ 3 4 }, sat = false, unassigned = 2,
{ 2 3 -4 }, sat = false, unassigned = 3]
]

Variable 4: [OPEN
Adjacence List: [{ 3 4 }, sat = false, unassigned = 2,
{ 2 3 -4 }, sat = false, unassigned = 3]
]


{ 1 2 -3 }, sat = true, unassigned = 0
{ 3 4 }, sat = true, unassigned = 0
{ -1 }, sat = true, unassigned = 0
{ -2 1 }, sat = true, unassigned = 0
{ 2 3 -4 }, sat = false, unassigned = 0


Variable 1: [FALSE
Adjacence List: [{ 1 2 -3 }, sat = true, unassigned = 0,
{ -1 }, sat = true, unassigned = 0,
{ -2 1 }, sat = true, unassigned = 0]
]

Variable 2: [FALSE
Adjacence List: [{ 1 2 -3 }, sat = true, unassigned = 0,
{ -2 1 }, sat = true, unassigned = 0,
{ 2 3 -4 }, sat = false, unassigned = 0]
]

Variable 3: [FALSE
Adjacence List: [{ 1 2 -3 }, sat = true, unassigned = 0,
{ 3 4 }, sat = true, unassigned = 0,
{ 2 3 -4 }, sat = false, unassigned = 0]
]

Variable 4: [TRUE
Adjacence List: [{ 3 4 }, sat = true, unassigned = 0,
{ 2 3 -4 }, sat = false, unassigned = 0]
]
```

```
{ 1 2 -3 }, sat = false, unassigned = 3
{ 3 4 }, sat = false, unassigned = 2
{ -1 }, sat = false, unassigned = 1
{ -2 1 }, sat = false, unassigned = 2
{ 2 3 -4 }, sat = false, unassigned = 3


Variable 1: [OPEN
Adjacence List: [{ 1 2 -3 }, sat = false, unassigned = 3,
{ -1 }, sat = false, unassigned = 1,
{ -2 1 }, sat = false, unassigned = 2]
]

Variable 2: [OPEN
Adjacence List: [{ 1 2 -3 }, sat = false, unassigned = 3,
{ -2 1 }, sat = false, unassigned = 2,
{ 2 3 -4 }, sat = false, unassigned = 3]
]

Variable 3: [OPEN
Adjacence List: [{ 1 2 -3 }, sat = false, unassigned = 3,
{ 3 4 }, sat = false, unassigned = 2,
{ 2 3 -4 }, sat = false, unassigned = 3]
]

Variable 4: [OPEN
Adjacence List: [{ 3 4 }, sat = false, unassigned = 2,
{ 2 3 -4 }, sat = false, unassigned = 3]
]


{ 1 2 -3 }, sat = true, unassigned = 0
{ 3 4 }, sat = true, unassigned = 0
{ -1 }, sat = true, unassigned = 0
{ -2 1 }, sat = true, unassigned = 0
{ 2 3 -4 }, sat = false, unassigned = 0


Variable 1: [FALSE
Adjacence List: [{ 1 2 -3 }, sat = true, unassigned = 0,
{ -1 }, sat = true, unassigned = 0,
{ -2 1 }, sat = true, unassigned = 0]
]

Variable 2: [FALSE
Adjacence List: [{ 1 2 -3 }, sat = true, unassigned = 0,
{ -2 1 }, sat = true, unassigned = 0,
{ 2 3 -4 }, sat = false, unassigned = 0]
]
```

```
Variable 3: [FALSE
Adjacence List: [{ 1 2 -3 }, sat = true, unassigned = 0,
{ 3 4 }, sat = true, unassigned = 0,
{ 2 3 -4 }, sat = false, unassigned = 0]
]

Variable 4: [TRUE
Adjacence List: [{ 3 4 }, sat = true, unassigned = 0,
{ 2 3 -4 }, sat = false, unassigned = 0]
]
```

Wie leicht von Hand nachzuvollziehen ist stimmt die Belegung der Variablen durch UP, die in diesen beiden speziellen Fällen eindeutig durch festgelegt ist. Die erste Formel ist tatsächlich nicht erfüllbar, da alle Variablen durch UP belegt wurden und es einen Konflikt gibt, die zweite Formel ist erfüllbar wie die Ausgabe zeigt.