Chapter 5

More on Clause Learning and Non-Chronological Backtracking and Algorithm 7

The basic idea behind clause learning is the addition of a clause to a theory. The learned clause is formed taking into account the assignments due to which the theory is falsified with respect to the current partial interpretation, representing a constraint preventing the same assignments to occur in the future by generating unit literals for propagation before the theory is falsified (in that way) again. To understand the clause learning and non-chronological backtracking used by our implementation of Algorithm 4, additional relevant information is discussed in this chapter.

We use resolution to obtain the learned clause. The basic idea of resolution is to take two clauses and merge them to form a new clause while removing a literal and its complement. Consider two clauses, $C_1 = C \cup \{v = x\}$ and $C_2 = C' \cup \{v \neq x\}$. Resolution performed on C_1 and C_2 results in $C \cup C'$. Now, consider an interpretation I that satisfies both C_1 and C_2. Of course I must falsify one of $v = x$, $v \neq x$. If I falsifies $v = x$, it must satisfy C, since

I satisfies C_1. On the other hand, if I falsifies $v \neq x$, it must satisfy C', since I satisfies C_2. Either way, I satisfies $C \cup C'$. The above discussion shows that the original clauses C_1 and C_2 "entail" the *resolvent* (the clause obtained by resolving C_1 and C_2). That is, any interpretation that satisfies the original clauses also satisfies the resolvent. Therefore adding the resolvent to a theory containing C_1 and C_2 does not affect the models of the theory. Thus resolution is a sound inference method, and we can safely use it to help guide our search for a model.

The version of resolution that we use in our algorithm is slightly stronger and slightly more general. Notice that our prior discussion of soundness of resolution involved considering cases based on whether a certain literal is satisfied by a given interpretation. In our function, an additional parameter L plays the role of that literal. The clause that is returned by our function consists of (1) the literals from clause C that are satisfied by at least one interpretation that *does not satisfy* L, and (2) the literals from C' that are satisfied by at least one interpretation that *also satisfies* L. That is, **resolve**(C, L, C') = $\{L' \in C \mid \{L\} \not\models L'\} \cup \{L' \in C' \mid \{L\} \not\models L'\}$.

Let's look at two examples of how **resolve**(C, L, C') works:

Example 1

Let us consider two clauses made up of variables $V = \{1,2,3\}$ such that for each $v \in V$, $dom(v) = \{0,1,2\}$.

C: $\{1 = 0, 2 = 2\}$

L: $3 = 2$

C': $\{1 = 1, 2 = 0, 3 \neq 2\}$

R: $\{1 = 0, 2 = 2, 1 = 1, 2 = 0\}$ is the clause returned by the function **resolve**(C, L, C')**.**

Example 2

Let us consider two clauses made up of variables $V = \{1,2,3\}$ such that for each $v \in V$, $dom(v) = \{0,1,2\}$.

C: $\{1=0, 2=2, 3=2\}$

L: $3=2$

C': $\{1=1, 2=0, 3=0, 3=1\}$

R: $\{1=0, 2=2, 1=1, 2=0\}$ is the clause returned by the function **resolve**(C, L, C')**.**

To show that the definition of **resolve**(C, L, C') is sound, we need to show that every model of $\{C, C'\}$ satisfies **resolve**(C, L, C').

Assume interpretation I satisfies $\{C, C'\}$. We need to show I satisfies **resolve**(C, L, C').

Consider two cases.

Case 1: $I \models L$. Since $I \models C'$, I satisfies one of the literals in C' that can be satisfied by some interpretation that also satisfies L. Consequently, I satisfies **resolve**(C, L, C').

Case 2: Otherwise. Since $I \models C$, I satisfies one of the literals in C that can be satisfied by some interpretation that does not satisfy L. Consequently, I satisfies **resolve**(C, L, C').

So, when some clause $C \in T$ is falsified w.r.t. P, we use resolution to form the learned clause. For this purpose, we define a "reason" for each literal L that is falsified by the current partial interpretation P. This "reason" is a specific clause, $L.reason$, obtained in one of the following three ways. If L became false because of a unit literal L' in P s.t. $\{L'\} \models\!\!\!| L$, then $L.reason$ is the unit clause that produced unit literal L'. If L became false because of a decision literal L' in P s.t. $\{L'\} \models\!\!\!| L$, then $L.reason$ is the clause $\{L', \bar{L'}\}$. (Notice that the clause $\{L', \bar{L'}\}$ is saisfied by every interpretation, and so by every model of T.) Otherwise $L.reason$ is the clause $\{v = x \mid v$ is the variable that occurs in L and $x \in dom(v)\}$. (Again, such a clause is satisfied by every interpretation.)

We should observe that of these three cases for $L.reason$, only the first is needed for clause learning in the Boolean setting. The need for the third case in the Finite Domain setting was recognized in Lal's thesis. Unfortunately, as we will discuss shortly, Lal's thesis did not recognize the need for the second case, leading to a subtle error in his clause learning algorithm.

So, in the clause learning algorithm, we take the reason clause, *L.reason*, of the literal *L* falsified latest in conflict clause *C*. We resolve clause *C* and *L.reason* w.r.t. to literal *L*. We check to see if the resultant clause has only one literal that was assigned a value at the current level (that way, we can backtrack so as to make the resolvent a unit clause, and continue the search from there!). If so, then we are done. If not, the process is repeated till we get a resolvent with only one literal that was falsified at the current level. The resolvent is then added to the theory and we backtrack to the appropriate level based on the learned clause. That is, we backtrack to the earliest level at which the learned clause is unit. After backtracking, the learned clause becomes unit and the search continues from there.

In Algorithm 7, there are global variables T and $level$ for storing the theory and the current decision level. In addition, there is another global variable *index* for keeping track of the order in which literals are falsified.

Each literal *L* has four associated fields: $value$, $level$, *reason*, and $index$. Fields $L.value$ and $L.level$ work as before. *L.reason* is as described in previous discussion. $L.index$ is used to keep track of the order in which literals were falsified in the current partial interpretation.

Algorithm 7

Solve()
$level \leftarrow 0$
$index \leftarrow 0$
for all literals L **do**
$L.value \leftarrow u$
$L.level \leftarrow -1$
for all clauses C **do**
$C.sat \leftarrow f$
$C.count \leftarrow$ number of literals in C
$C.level \leftarrow -1$
DPLL()

```
T is a theory
DPLL()
 while(true)
  if (checkSat() = t) then
   return {L | L.value = t}
  if (checkSat() = f) then
    if (level = 0) then
     return false
    else
     C ← conflictClause()
     learnedClause ← analyzeConflict(C)
     level ← findBacktrackingLevel(learnedClause)
     T ← T ∪ {learnedClause}
     backtrack(level)
  C ←findUnitClause()
  if (C ≠ NULL) then
   L ←unitLiteral(C)
   reason ← C
  else
   L ←pickLiteral()
   reason ← {L, L̄}
   level ← level + 1
  propagate(L, reason)
```

```
conflictClause()
 for all clauses C ∈ T do
  if (C.count = 0) then
   return C
 return NULL
```

```
findBacktrackingLevel(C)
 max = −1
 for all literals L ∈ C do
  if (L.level > max and L.level < level) then
   max ← L.level
 if (max ≠ −1) then
  return max
 else
  return level − 1
```

findUnitClause()
 for all clauses $C \in T$ **do**
  if ($C.sat = f$ **and** $C.count = 1$) **then**
   return C
 return $NULL$

unitLiteral(C)
 return a literal L in C s.t. $L.value = u$

propagate($L, reason$)
 $P \leftarrow \{L \mid L.value = t\}$
 $index \leftarrow index + 1$
 for all literals L' s.t. $P \cup \{L\} \models L'$ **do**
  if ($L'.value = u$) **then**
   $L'.value \leftarrow t$
   $L'.level \leftarrow level$
   for all clauses C s.t. $L' \in C$ **do**
    if ($C.sat = f$) **then**
     $C.sat \leftarrow t$
     $C.level \leftarrow level$
 for all literals L' s.t. $P \cup \{L\} \models\!\!\!| L'$ **do**
   if ($L'.value = u$) **then**
    $L'.value \leftarrow f$
    $L'.level \leftarrow level$
    $L'.index \leftarrow index$
    if ($\{L\} \models\!\!\!| L'$) **then**
     $L'.reason \leftarrow reason$
    else
     $L'.reason \leftarrow \{v = x \mid v$ is the variable that occurs in L' **and** $x \in dom(v)\}$
    for all clauses C s.t. $L' \in C$ **do**
     $C.count \leftarrow C.count - 1$

```
backtrack(level)
 for all literals L do
  if (L.level > level) then
   if (L.value = t) then
    for all clauses C s.t. L ∈ C
     if (C.level > level) then
      C.sat ← f
      C.level ← −1
   if (L.value = f) then
    for all clauses C s.t. L ∈ C do
     C.count ← C.count + 1
   L.value ← u
   L.level ← −1
```

```
analyzeConflict(C)
 if (potent(C)) then
  return C
 L ← maxLit(C)
 resolvent ← resolve(C, L, L.reason)
 return analyzeConflict(resolvent)
```

```
potent(C, level)
 if there is exactly one L ∈ C s.t. L.level = level then
  return t
 else
  return f
```

```
maxLit(C)
 return a literal L ∈ C with maximal L.index
```

```
resolve(C, L, C′)
 return {L′ ∈ C | {L} ⊭ L′} ∪ {L′ ∈ C′ | {L} ⊭ L′}
```

The functions **checkSat**(), and **pickLiteral**() work the same as before. Let's see the other functions:

conflictClause(): This function returns a clause falsified w.r.t. to the current partial interpretation, if one exists; otherwise *NULL*.

findUnitClause(): This function returns a clause unit w.r.t. to the current partial interpretation, if one exists; otherwise *NULL*.

unitLiteral(C): This function returns a literal unassigned w.r.t. to the curent partial interpretation in the clause C.

propagate(L, *reason*)**:** This function satisfies all literals L' s.t. $P \cup \{L\} \models L'$ and $P \not\models L'$. The function also satisfies all not yet satisfied clauses C containing such L'. This function falsifies all unassigned literals L' s.t. $P \cup \{L\} \not\models L'$ and decrements $C.count$ of all clauses C s.t. $L' \in C$. Now this function also sets the *reason* and *index* fields for the falsified literals.

backtrack($level$)**:** This function restores the state just before the decision that began level $level + 1$.

analyzeConflict($C, level$)**:** This function returns a clause to be added to the theory i.e. the learned clause. The clause returned should contain exactly one literal at the current $level$.

potent(C, $level$): This function returns $true$ if the clause C contains exactly on only one literal L having $L.level = level$, else returns *false*.

maxLit(C): This function returns a literal $L \in C$ that was falsified with respect to the current partial interpretation after all other literals in C. That is, it returns literal $L \in C$ with maximal $L.index$.

findBacktrackingLevel(C): This function returns the second highest level associated with a literal $L \in C$. If the clause C consists of only one literal, then it returns $level - 1$. There are other options available, such as we can always return $level - 1$.

resolve(C, L, C'): This function returns the clause $\{L' \in C \mid \{L\} \not\models L'\} \cup \{L' \in C' \mid \{L\} \not\models L'\}$. We use the same form of resolution as used by Lal [15].

Now let's look at few examples to see how Algorithm 7 works:

Example 1

Lets reconsider Example 2 from Section 2.4. Theory T is made up of variables $V = \{1,2\}$ such that for each $v \in V$, $dom(v) = \{0,1\}$. The theory T consists of the four clauses given below. The current partial interpretation $P = \emptyset$. **Solve()** initializes the data structures and makes the first call **DPLL()**.

$P = \emptyset$

$Clause1$ $\{1 = 0,\ 2 = 0\}$ $U(2)$

$Clause2$ $\{1 \neq 0,\ 2 = 0\}$ $U(2)$

$Clause3$ $\{1 = 0,\ 2 \neq 0\}$ $U(2)$

$Clause4$ $\{1 \neq 0,\ 2 \neq 0\}$ $U(2)$

T is neither satisfied nor falsified and there are no unit literals, so we pick a literal. Assume the "decision literal" is $1 = 0$ from $Clause1$. The *level* parameter is incremented to 1, since $1 = 0$ will be our first decision literal, and literal $1 = 0$ is propagated. As $1 = 0$ is a decision literal, the newly falsified literals $1 \neq 0$ and $1 = 1$ have their *reason* set to be the clause $\{1 = 0,\ 1 \neq 0\}$ and their *index* set to 1.

$P = \{1 = 0_1\}$

$Clause1$ $\{1 = 0,\ 2 = 0\}$ $S(2)$

$Clause2$ $\{\cancel{1 \neq 0},\ 2 = 0\}$ $U(1)$

$Clause3$ $\{1 = 0,\ 2 \neq 0\}$ $S(2)$

$Clause4$ $\{\cancel{1 \neq 0},\ 2 \neq 0\}$ $U(1)$

Now $Clause1.sat = t$, $Clause1.level = 1$ and $Clause3.sat = t$, $Clause3.level = 1$. Notice that $Clause2$ is unit, as $Clause2.sat = f$ and $Clause2.count = 1$. So we must satisfy unit literal $2 = 0$. So literal $2 = 0$ is propagated at level 1. As $2 = 0$ is a unit literal, the newly falsified literals $2 \neq 0$ and $2 = 1$ have their *reason* set to be $Clause2$ and their *index* set to be 2.

$P = \{1 = 0_1,\ 2 = 0_1\}$

$Clause1\ \{1 = 0,\ 2 = 0\}\ \ S(2)$

$Clause2\ \{\cancel{1 \neq 0},\ 2 = 0\}\ \ S(1)$

$Clause3\ \{1 = 0,\ \cancel{2 \neq 0}\}\ \ S(1)$

$Clause4\ \{\cancel{1 \neq 0},\ \cancel{2 \neq 0}\}\ \ F(0)$

Now $Clause2.sat = t$ and $Clause2.level = 1$. But T is falsified, as $Clause4.count = 0$. As current level is not equal to 0, $Clause5$ is learned and added to the theory. $Clause5$ is formed using the $Clause4$ and the reason clause of the literal falsified last in $Clause2$. As literal $2 \neq 0$ has index greater than index of $1 \neq 0$ in $Clause4$, we pick the reason clause of $2 \neq 0$ and $Clause4$ to form the learned $Clause5$. The function **resolve**($Clause2$, $2 \neq 0$, $Clause4$) will create $Clause5$ to be $\{1 \neq 0\}$ having only one literal $1 \neq 0$ at the current *level*. After learning the clause, we backtrack to level 0 based on learned $Clause5$.

$P = \emptyset$

$Clause1\ \{1 = 0,\ 2 = 0\}\ \ U(2)$

$Clause2\ \{1 \neq 0,\ 2 = 0\}\ \ U(2)$

$Clause3\ \{1 = 0,\ 2 \neq 0\}\ \ U(2)$

$Clause4\ \{1 \neq 0,\ 2 \neq 0\}\ \ U(2)$

$Clause5\ \{1 \neq 0\}\qquad U(1)$

After backtracking to level 0, $Clause5$ is unit, as $Clause5.sat = f$ and $Clause5.count = 1$. So we must satisfy unit literal $1 \neq 0$ in $Clause5$. Thus, literal $1 \neq 0$ is propagated at level 0. As $1 \neq 0$ is a unit literal, the newly falsified literal $1 = 0$ has its *reason* set to be $Clause5$ and *index* set to be 3.

$P = \{1 \neq 0_0\}$

$Clause1\ \{\cancel{1 = 0},\ 2 = 0\}\ \ U(1)$

$Clause2\ \{1 \neq 0,\ 2 = 0\}\ \ S(2)$

$Clause3\ \{\cancel{1 = 0},\ 2 \neq 0\}\ \ U(1)$

$Clause4\ \{1 \neq 0,\ 2 \neq 0\}\ \ S(2)$

$Clause5\ \{1 \neq 0\}$ $S(1)$

Now $Clause2.sat = t$, $Clause2.level = 0$ and $Clause4.sat = t$, $Clause4.level = 0$. Notice that $Clause1$ is unit, as $Clause1.sat = f$ and $Clause1.count = 1$. So we must satisfy unit literal $2 = 0$. So literal $2 = 0$ is propagated at level 0. As $2 = 0$ is a unit literal, the newly falsified literals $2 \neq 0$ and $2 = 1$ have their *reason* set to be $Clause1$ and *index* set to 4.

$P = \{1 \neq 0_0,\ 2 = 0_0\}$

$Clause1\ \{\sout{1 = 0},\ 2 = 0\}$ $S(1)$

$Clause2\ \{1 \neq 0,\ 2 = 0\}$ $S(2)$

$Clause3\ \{\sout{1 = 0},\ \sout{2 \neq 0}\}$ $F(0)$

$Clause4\ \{1 \neq 0,\ \sout{2 \neq 0}\}$ $S(1)$

$Clause5\ \{1 \neq 0\}$ $S(1)$

Now $Clause1.sat = t$ and $Clause1.level = 0$. But T is falsified, as $Clause3.count = 0$. As the theory is falsified with respect to the current partial interpretation when level equals 0, we return *false*, implying that the theory is unsatisfiable.

Example 2

Given below is a theory T with variables $V = \{1,2\}$ such that for each $v \in V$, $dom(v) = \{0,1\}$. The theory T consists of the two clauses given below. **Solve()** initializes the data structures and makes the first call **DPLL()**.

$P = \emptyset$

$Clause1\ \{1 = 0,\ 1 \neq 1\}\ U(2)$

$Clause2\ \{1 = 1,\ 1 \neq 0\}\ U(2)$

T is neither satisfied nor falsified and there are no unit literals, so a decision literal is picked. Assume the "decision literal" is $1 = 0$ from $Clause1$. The *level* parameter is incremented to 1, since $1 = 0$ will be our first decision literal, and literal $1 = 0$ is propagated. As $1 = 0$ is a decision literal, the literals newly falsified $1 \neq 0$ and $1 = 1$ have their *reason*

set to be the clause $\{1 = 0,\ 1 \neq 0\}$ and *index* set to be 1.

$P = \{1 = 0_1\}$

$Clause1$ $\{1{=}0,\ 1 \neq 1\}$ $S(2)$

$Clause2$ $\{\cancel{1 = 1}, \cancel{1 \neq 0}\}$ $F(0)$

Now $Clause1.sat = t$ and $Clause1.level = 1$. But T is falsified, as $Clause2.count = 0$. As current level is not equal to 0, $Clause3$ is learned and added to the theory. $Clause3$ is formed using the $Clause2$ and reason clause of the literal falsified last in $Clause2$. As literals $1 = 1$ and $1 \neq 0$ were falsified w.r.t. P together and so have the same index, we pick the reason clause of one of them to form the learned $Clause5$. Assume we pick the reason clause of $1 = 1$. The function **resolve**($Clause2$, $1 = 1$, $\{1 = 0,\ 1 \neq 0\}$) will create $Clause3$ to be $\{1 \neq 0\}$ having only one literal $1 \neq 0$ at the current *level*. After learning the clause we backtrack to level 0 based on learned $Clause3$.

$P = \emptyset$

$Clause1$ $\{1 = 0,\ 1 \neq 1\}$ $U(2)$

$Clause2$ $\{1 = 1,\ 1 \neq 0\}$ $U(2)$

$Clause3$ $\{1 \neq 0\}$ $U(1)$

After backtracking to level 0, $Clause3$ is unit, as $Clause3.sat = f$ and $Clause3.count = 1$. So we must satisfy unit literal $1 \neq 0$ in $Clause3$. Thus, literal $1 \neq 0$ is propagated at level 0. As $1 \neq 0$ is a unit literal, the newly falsified literal $1 = 0$ has its *reason* set to be $Clause3$ and *index* set to be 2.

$P = \{1 \neq 0_1\}$

$Clause1$ $\{\cancel{1 = 0}, \cancel{1 \neq 1}\}$ $F(0)$

$Clause2$ $\{1 = 1,\ 1 \neq 0\}$ $S(2)$

$Clause3$ $\{1 \neq 0\}$ $S(1)$

Now $Clause2.sat = t$ and $Clause2.level = 1$. But T is falsified, as $Clause1.count = 0$. As the theory is falsified with respect to the current partial interpretation when level equals

0, we return *false*, implying that the theory is unsatisfiable.

Note: The algorithm implemented by Lal [15] is essentially the same as Algorithm 7 but the $L'.reason$ when L' is falsified because of a decision literal L was not set properly to be the clause $\{L, \bar{L}\}$. Consequenstly, Lal's version of the **analyzeConflict**() function fails to return a learned clause on problems such as that in Example 2 above. Due to this reason, Lal's solver crashes on certain inputs.