CSC2512
Advanced Propositional Reasoning

CSC2512: Other Work: Proof extraction

- For any clause c, if we unit propagate –c, in the formula F and obtain an empty clause (a conflict) then it must be the case that F ⊨ c by the soundness of UP.

- However, we can have F ⊨ c but UP(-c) does not generate a conflict. UP is not a complete rule of inference.

- Nevertheless, it is “complete” along a sequence of resolution steps.

CSC2512: Other Work: Proof extraction

- Given a resolution proof as a sequence of clauses where c_n is not an input clause.

$$c_1, c_2, \ldots, c_n$$

- we can observe that if we negate c_n and unit propagate the literals in the formula $c_1, c_2, \ldots, c_{n-1}$ we will obtain a conflict (one of these clauses will be falsified)

If c_n = (A, B) as a result of resolving (A,x) and (B,-x) UP falsifies one of these clauses (depending on if it propagates x or –x first).

CSC2512: Other Work: Proof extraction

- This gives rise to the RUP (reverse unit propagation) technique for extracting proofs from a clause learning SAT solver.

- Output to a log all learnt clauses in the sequence they are learnt.

- Verify each learnt clause c_i in the order they it was learnt by negating c_i and unit propagating through the set of clauses U $\{c_1, \ldots, c_{i-1}\}$

- If we obtain a conflict we know that c_i is a logical consequence of the input formula and the previously verified learnt clauses.

CSC2512: Other Work: Proof extraction

- Eventually we can verify a unit clause (x) whose partner (-x) has previously been verified thus showing that the proof is sound.
- This procedure verifies the UNSAT result (just like the satisfying assignment can be used to verify the SAT result).
- Furthermore, we can instrument the UP checking process so that we only keep the learnt clauses and input clauses that are eventually needed to verify the final empty clause.
- Note that this works even when we have lost track of the resolution steps involved in computing a learnt clause.
- This set of clauses responsible for UNSAT can be output.

CSC2512: Other Work: Proof extraction

- Note also that the final sequence of learnt clauses are not a traditional resolution proof. This sequence is called a “clausal” proof, and it can be much shorter than a resolution proof.

- Unit Prop is needed to verify a clausal proof, whereas a much simple algorithm can verify a resolution proof.

- A clausal proof can be expanded into a resolution proof by tracking the clauses the unit prop steps need to derive a contradiction.

Paper: Trimming while Checking Clausal Proofs Marijn J.H. Heule, Warren A. Hunt, Jr., and Nathan Wetzler

CSC2512: Other work: Assumptions

Assumptions. A useful technique is solving subject some set of literals called assumptions:

- $A = \{l_1, l_2, \ldots, l_k\}$
- We start the SAT solver and force it to pick a next unassigned literal in A as a **decision** until there are no more unassigned literals in A.
- If a literal of A is forced to TRUE we skip over it for the next decision.
- If a literal of A is forced to FALSE we stop: Say l_1, l_2 …, l_i are the decisions already made, and l_j is forced to FALSE: then we have the following clause
  $(\neg l_1, \neg l_2, \ldots, \neg l_i, \neg l_j)$

CSC2512: Other work: Assumptions

- If we assign all literals in A we then continue the normal SAT solving process with freedom to pick the decision variables as we want.
- If this results in UNSAT, some clause (perhaps empty) falsified by the A decisions will be learnt.

- In any event, if the formula becomes UNSAT under A, we obtain a clause falsified by A (the clause specifies that some subset of A is impossible).

CSC2512: Other work: Assumptions

- Assumptions are very useful for **incremental** SAT solving where we want to SAT solver a sequence of related formulas F_1, F_2, …, F_n

- If $F_1 \subseteq F_2 \subseteq \cdots \subseteq F_n$ then we can use one instance of the SAT solver. Solve F_1 then add the additional clauses of F_2 and solve again, add the additional clauses of F_2 …
  - The advantage if this is that the SAT solver gets to reuse all of its learnt clauses.
- But if we must remove clauses between SAT solver invocations we have a problem: some of the learnt clauses might no longer be valid.

CSC2512: Other work: Assumptions

- For clause removals we can use assumptions: if we will later want to remove the clause C = $(x_1, x_2, \ldots, x_n)$ we can add a brand new variable (often called a selection variable) to the clause: $(x_1, x_2, \ldots, x_n, s)$
- Then if we want to include C we assume $\neg$**s.** Any learnt clauses were derived from resolving against C will now also contain **s** (**s** appears nowhere else in the formula so it can't be resolved away)
- When we want to exclude C from the formula we can assume **s.** C and all clauses learnt using C contain **s** so all of these clauses will be satisfied by assuming **s** and the solver will then solve the remaining clause.

CSC2512: Papers

- An Empirical Study of Branching Heuristics through the Lens of Global Learning Rate Jia Hui Liang, Hari Govind, Pascal Poupart, Krzysztof Czarnecki, and Vijay Ganesh. IJCAI 2018

- DRAT-trim: Efficient Checking and Trimming Using Expressive Clausal Proofs Nathan Wetzler, Marijn J. H. Heule, Warren A. Hunt Jr. Sat 2014.

- Speeding Up Assumption-Based SAT, Randy Hickey, Fahiem Bacchus. Sat 2019

CSC2512: Clause Deletion

- A new clause is learned from every conflict.
- In practice the solver starts to slow down after it accumulates too many clauses.
- So deleting some of these learned clauses has proved to be effective.
- Earlier clauses were deleted whenever memory was about to be exhausted. Clauses were deleted by size (delete the largest ones first) or by activity (delete those clauses that had not recently been used in learning new clauses.

CSC2512: Clause Deletion

- In 2009 Audemard and Simon developed a new idea for selecting which clauses to delete called the LBD score.
- Once we learn a 1-UIP clause, minimize it, and use it to backtrack asserting a new literal, we can count the number of different decision levels in the clause. This is called the clause's LBD score.
- Audemard and Simon found that very aggressive clause deletion, where frequently ½ of the learnt clauses highest LBD score are deleted, gives a significant boost in performance.
- Now however theoretical completeness is sacrificed (although it can be regained by slowing increasing the clause deletion trigger from 10,000 clauses to 20,000, 30,000, etc. (any increasing sequence will suffice).

CSC2512: Papers

1. Predicting Learnt Clauses Quality in Modern SAT Solvers, Gilles Audemard, Laurent Simon, IJCAI 2009.
2. Coverage-Based Clause Reduction Heuristics for CDCL Solvers, Hidetomo Nabeshima and Katsumi Inoue, SAT 2017

CSC2512: Preprocessing

- An additional essential part of modern SAT solvers is preprocessing and inprocessing.
- Preprocessing is the technique of converting the input CNF F to a new CNF F' such that if F' is UNSAT then so is F, and furthermore if π is a satisfying model of F' then π can in poly-time be converted into π' a satisfying model for F.

CSC2512: Preprocessing

- A number of useful techniques for preprocessing have been developed. The most important of these is Bounded Variable Elimination, Clause subsumption, and self-subsuming resolutions.

CSC2512: Preprocessing

- Bounded Variable Elimination. This uses a single step of the DP algorithm. We eliminate the variable x from the formula by taking all clauses A containing x and all clauses B containing ¬x and generate all resolvant pairs:
  $R = \{R[c_{1,}c_2] \mid c_1 \in A \;\; c_2 \in B\}$
- All tautologies are removed from R. Furthermore, clauses in R might be subsumed by other clauses. So we reduce R by removing these clauses.
- **Bounded**: we preform this step if
  $|R| < |A| + |B|$
  (i.e., we obtain fewer clauses)

CSC2512: Preprocessing

- Implementing this efficiently requires clever scheduling and data structure techniques.

CSC2512: Preprocessing

- Subsumption. Checking for subsumption can be speeded up using **Bloom Filters**. We map each literal in F to a number in the range [0,63]. Then for each clause **c** we construct a 64 bit map, by setting all bits mapped to by literals in **c.**
- Now if **c**' ⊆ **c** then the bit map of **c**' must be a subset of **c** bit map. That is, the and of these two bit maps must equal the bit map of **c'**.

  – **E.g., Say we do the following mapping**
  x = 0, -x = 1, y = 2, -y = 3, z = 4, -z = 5.

  then the bit map of the two clauses
  c = (x,y,z) = [1, 0, 1, 0, 1, 0]
  c'= (x,z) = [1, 0, 0, 0, 1, 0]

  [1, 0, 1, 0, 1, 0] AND [1, 0, 0, 0, 1, 0] = [1, 0, 0, 0, 1, 0] = c' bit map

CSC2512: Preprocessing

- This test is fast, and if it fails then we know that **c** is not subsumed by **c'.** If it succeeds then we don't actually know that **c** is subsumed by **c'**. This is a **one way test.** So if the test succeeds (i.e., the AND of the two bit maps is equal to the bit map of **c'** we have to follow this with actually testing to see **c'** ⊆ **c**
  - **E.g. if in the variable mapping r = 4 then c" = (x,r) = [1, 0, 0, 0, 1, 0] – same bit map as c'**

CSC2512: Preprocessing

- (A,x) (B,¬x) where B ⊆A. Clearly(B,¬x) is not a subset of (A,x), so there is no clause subsumption. However, consider the resolvant: (A,B) == (A) (since B ⊆A). The resolvant subsumes (A,x). So in this case we can remove x from (A,x). This is called a **self-subsuming** resolution.
  - E.g.
    (a, b, ¬c, **x**) and (b, ¬**x**)
        ➔ (a,b,¬c) and (b, ¬**x**)

CSC2512: Papers

1. Effective Preproessing in SAT through Variable and Clause Elimination, Niklas Een and Armin Biere, SAT 2005.

CSC2512: MUSes and MCSes

- MUS computation.

- In many applications we want to know why something is unsatisfiable. We can extract a minimal unsatisfiable subset of the formula: a **MUS**.
- Note not a **minimum** unsatisfiable subset (which is a much harder problem)
- Since the **MUS** typically much smaller than the input formula **F.** It can provide much more specific information about a cause of unsatisfiablity in **F.**

CSC2512: MUSes and MCSes

- A MUS (Minimal Unsatisfiable Set) **M** is an UNSAT set of clauses **M** such that for any clause c in **M**:

  M \ {c} is SAT   //M is set inclusion minimal

  - If **F** is SAT then it contains no MUSes. If it is UNSAT it contains at least one MUS and usually contains many different MUSes.

- A correction set **C** of a CNF **F** is a subset of **F** such that:

  F \ C is SAT

  A correction set **C** is a **minimal correction set (MCS)** if no proper subset of **C** is a correction set of **F**

  - If **F** is SAT only the empty set is a MCS. But if **F** is UNSAT, then any MCS cannot be empty and generally, there are many MCSes.

CSC2512: MUSes and MCSes

- MUS/MCS hitting set duality (Reiter AIJ 2087).

- Consider an UNSAT formula **F**, let **AllMuses(F)** be the collection of all MUSes in **F.** Each M ∈ **AllMuses(F)** is a set of clauses, a subset of **F**. That is **AllMuses(F)** is a collection of sets.
- Similarly, let **AllMCSes(F)** be the collection of all MCSes of **F**

CSC2512: MUSes and MCSes

- Given a collection of sets **K**, **HS** is a **hitting set** of **K** iff for every set S ∈ **K** we have that **HS ∩ S ≠ ∅**
  - **HS** has a non-empty intersection with every set in the collection.
- A set **HS** is a **minimal hitting set** of **K** if it is a hitting set and no proper subset of **HS** is a hitting set of **K.**

CSC2512: MUSes and MCSes

- Reiter's result:

A set **C** ⊆ **F** is an MCS of **F iff** it is a minimal hitting set of **AllMuses(F)**. And a set **M** ⊆ **F** is an MUS of **F iff** it is a minimal hitting set of **AllMCSes(F).**

Also, a set **C** ⊆ **F** is an correction set (not necessarily minimal) of **F iff** it is a hitting set of all unsatisfiable subsets of **F** (not necessarily minimal). And a set **M** ⊆ **F** is unsatisfiable (not necessarily minimal) **iff** it is a hitting set of all correction sets (not necessarily minimal) of **F.**

CSC2512: MUS extraction

- Given an UNSAT formula **F**, we want to compute one of its **MUSes** (and we don't care which one).
- We can do this with a sequence of calls to a SAT solver.

CSC2512: MUS extraction

- A **critical** clause of an UNSAT formula **U**, is a clause whose removal makes **U** SAT.
    - A MUS **M** is an UNSAT formula all of whose clauses are critical.

- Divide **F** into two sets
    - **crits:** a set of clauses that we know must be in the **MUS** we are extracting (they are critical).
    - **unkn** a set of clauses that might be in the **MUS** but we don't know yet.

    - **crits U unkn** is the working formula—it is an unsat formula that is a subset of **F** and thus it contains one of **F's** MUSes

CSC2512: MUS extraction

```
1. crits ← ∅  unkn ← F
2. while unkn ≠ ∅
    1. c ← choose c ∈ unkn
    2. sat? = SatSolve(crits U unkn \ {c})
    3. if sat?
        1. crits = crits U {c}
    4. unkn = unkn - \{c}
```

This simple algorithm iteratively tests the clauses of an initial UNSAT formula (**F**) removing clauses not needed to retain UNSAT, and keeping those clauses whose removal makes the formula SAT.

CSC2512: MUS extraction

This simple algorithm can be significantly improved.

1. When we find that removing **c** from unkn makes crits U unkn (so **c** is critical for the **MUS** contained in crits U unkn) we can use the satisfying truth assignment π to find other critical clauses.
2. When removing **c** from unkn keeps crits U unkn UNSAT (**c** need not be in the **MUS** we are extracting), then we can extract from the SAT solver a subset of the clauses in crits U unkn sufficient to cause UNSAT using **assumptions**

CSC2512: MUS extraction

Model Rotation. Given that **c** is found to be critical, find other critical clauses.

When is a clause **c** critical for the working formula crits U unkn.

There exists a truth assignment satisfying (crits U unkn) \ {**c**}—removing c makes the working formula SAT

We have found a truth assignment π satisfying (crits U unkn) \ {**c**}

So we try to change one of the truth assignments in π so that we satisfy **c** and every other clause in (crits U unkn), except for one other clause **c'**. So now we have a new truth assignment π' that satisfies (crits U unkn) \ {**c'**}

This shows that **c'** is also critical for the current working formula and we can move it into crits.

CSC2512: MUS extraction

Assumptions. Instead of giving the SAT solver the CNF (crits U unkn) we add a **selector** variable to each clause of **F.**

The selector variables are brand new variables (one new variable per clause). So every clause $c_i \in$ **F** is replaced with the clause $(c_i \vee b_i)$ where b_i is the new selector variable for clause c_i.

Then we call the SAT solver with all clauses in **F** and the assumptions $\{\neg b_1, \neg b_2, \ldots, \neg b_m\}$. These assumptions force the SAT solver to try to satisfy all of the clauses c_i

CSC2512: MUS extraction

Assumptions.

When we want to remove the clause ci from the working formula, we stop assuming ¬bi. Now the SAT solver is free to satisfy (c_i v b_i) by simply making b_i true.

If the working formula is UNSAT then the SAT solver will return a subset of the assumptions $\{\neg b_1, \neg b_2, \ldots, \neg b_m\}$ in a conflict clause ($b_{j1}, b_{j2}, \ldots, b_{jk}$). This clause says that the subset of clauses $\{c_{j1}, c_{j2}, \ldots, c_{jk}\}$ is UNSAT (at least one of them must be falsified by any truth assignment.

Now we can use this subset to further reduce the working formula.

CSC2512: MUS extraction

```
1. crits_A ⟵ ∅
2. unkn_A = {¬b_i | c_i ∈ F}    //Two sets of assumptions
3. while unkn_A ≠ ∅
   1. choose ¬b_i ∈ unkn_A
   2. (sat?, π, conflict) =
              SatSolve(F, crits_A U unkn_A \ {¬b_i})
   3. if sat?
      1. new_crits = Model_Rotate(c_i, π)
      2. crits_A = crits_A U {¬b_i | c_i ∈ new_crits}
      3. unkn_A = unkn_A \ {¬b_i | c_i ∈ new_crits}
   4. else
      1. unkn_A = unkn_A ∩ {¬b_i | b_i ∈ conflict}
```

CSC2512: MUS extraction

However we can get even better improvements by moving beyond the simple algorithm.

Paper:

Using Minimal Correction Sets to more Efficiently Compute Minimal Unsatisfiable Sets, Fahiem Bacchus and George Katsirelos (CAV 2015).