18760: Project 1

A SATISFIABILITY SOLVER

March 17, 2015

Andrew Mort (amort@andrew.cmu.edu)
Chad Cole (cjcole@cmu.edu)

Introduction

For this project, we were required to create an SAT solver that will read benchmarks in the DIMACS CNF format. Our solvers will be tested based on metrics of speed and range of tests that it can accommodate. As such, these were the factors that weighed most heavily on our minds during the concept phase of the project. Efficiency was the top goal in our project therefore this lead to the use of simple data structures with O(1) amortized runtime operations. To address the coolness factor, we decided to implement three of the metrics discussed in class: (1) Two-literal Watching, (2) Clause Learning, and (3) Non-Chronological Backtracking.

Idea

a: **Two-literal Watching** Two-literal Watching is a metric to reduce the amount of work performed by the solver. The usefulness of the technique becomes apparent when looking at large clauses. The set of literals is effectively reduced to two, and the solver needs to perform no work at all on the other literals. This is because the SAT solver now only cares about conflicts or unit clauses. If it finds a clause is satisfied it doesn't care - a satisfied clause doesn't need any more work. In other words, for Two-literal Watching you have a clause with two or more variables. In the case where there are two or more unassigned variables, the watched variables must be unassigned. Therefore, if you assign one of these to 0, you must visit the clause and either move to another unassigned variable or find that the clause goes unit. Only then is it possible that some other unit clause could imply the final watched variable to 0, making the clause a conflict.

b: **Clause Learning** Clause learning takes advantage of work that was already done by the solver for conflict clauses found under previous implications. The learned clause is added to the end of the clause list and it causes a conflict more quickly so that you don't make the same decision and run into the same conflict again.

c: **Non-Chronological Backtracking** Non-Chronological Backtracking reduces the search space for the solver. Take for instance normal backtracking, where when a conflict is reached, the most recently made decision is reversed. The problem with this is that the most recent decision might not have caused the conflict. Non-Chronological Backtracking skips to the clause that caused the conflict saving work for the solver.

Implementation

a: **Two-literal Watching** The clauses are represented as rows in a column vector. Each of these rows (clauses) is itself a vector of integers that represents the polarity of the variables. Two-literal watching is implemented using a vector sized to the total number of variables. Each entry in this vector is a tuple of sets that together contain the positive and the negative lists of clauses that are currently watching the positive or negative version of this variable respectively. Not every instance of each variable is

Implementation continued on next page...

contained in these lists, only the watched variables from each clause. The watched list vector is indexed by the absolute value of each possible variable: i.e.

```
watched_list[i].pos
```

will access the set of clauses that contain the watched variable i in its positive polarity. Similarly for the negative polarity.
At the beginning of the program, the watch list is empty and the vector of assigned values is initialized to `UNASSIGNED`. The watch list is populated by calling the function `assign` with the variable 0, since 0 is not a valid variable. The clauses are then propagated and the first two literals in each clause are chosen as the watched variables. Any unit clauses are added to the assignment queue so that they can be assigned values from the beginning.
Now that the watch list is initialized and populated, we should move on to discussing how it is used. The solver uses a queue (`assignment_queue`) to keep track of which variables to process next. These variables come either from a clause going unit or from a variable decision being made. For each variable in the assignment queue, the assignment is made and then the watch list for the opposite polarity of that variable is inspected. For each clause in the watch list, the program will choose a new watched variable that is currently unassigned, find a unit clause and add it to the assignment queue, or determine if the new assignment creates a conflict.
When a conflict occurs, the index of the conflicting clause is returned and operated on by the backtrack function which implements clause learning and the non-chronological backtracking. When a backtrack does occur, we don't have to change the watched literals which saves precious computation time. If there is no conflict and no more unit clauses remain, the assign function will return -1 to indicate that no faults have occurred and the program will make the next assignment until all variables are assigned.

b: Clause Learning When a variable is assigned and it causes a conflict, the index to the clause that had the conflict is returned. The program then knows that a conflict has occurred and calls the backtrack function with the conflicting clause. The firstUIP algorithm is used to find the conflict clause which is then added to the overall clause list. The firstUIP algorithm starts by setting C equal to the conflicting clause and then finding the resolvent of C and the antecedent of the most recently assigned variable in C. The antecedent of a variable is the clause that caused that variable to become unit. The resolvent is found by combing C and the antecedent of C and removing both polarities of the most recently assigned variable in C. The resolvent is then assigned to C and the loop continues until there are no longer more than 2 variables in C that were assigned in the current level. The checking for the loop is performed by finding the two most recent variables in C and ensuring that the second most recently assigned variable was assigned in the current level. The resolvent is found by looping through C and removing the most recent variable by moving the back element in C to take its place. Once C has been created without the most recent variable, all the variables from the antecedent are copied into C except for duplicates already in C and the most recently assigned variable from C. In this way we can easily find the resolvent and the antecedent. The final value of C is the

Implementation continued on next page...

learned clause that gets added to main clause list. If there are any unassigned variables in this clause after removing the assignments from backtracking, we labels these as watched variables.

c: Non-Chronological Backtracking Non-Chronological backtracking is implemented using the learned clause C from the clause learning step. The most recently assigned variable in C should now be the only variable in C that was assigned in the current level. Therefore, to do backtracking, we can look at the next most recently assigned variable, which will be in the next highest level after the current level. We will backtrack to this level and then make the opposite assignment of the assignment performed at that level. The assigned variables and levels after this level are removed since we don't care about them anymore. Another thing that is done is to keep track of how many times we've backtracked to the same variable by keeping a count. When this count is greater than 1, meaning that we've set both positive and negative versions of this variable, we will continue to backtrack up to the next level. This prevents the simulator from trying to set a variable continuously from positive to negative and back to positive again. Although the clause learning would prevent this from occurring forever, we can further reduce this using the technique described.

d: Variable Selection The variable selection is done by randomly selecting a variable from the remaining unassigned variables. The variable assignments are kept in a vector and this vector is indexed randomly. From this index location the vector is searched until the first unassigned variable is found. If no unassigned variables are found, the function returns 0 which indicates that the problem has been satisfied.

e: Semi Random Restarts As described in the non-chronological backtracking section, the number of times we've backtracked to a certain level is recorded. When we've made more than 1 backtrack for each level, we go back to level 0 and essentially restart from the beginning. Since the variable selection is random, the next assignment is randomly chosen and the SAT solver starts over.

Test Results

The sat solver is able to correctly solve each test we provided with as you can see in the appendix.

Conclusion

The goal for the project was to implement a SAT solver with Two-literal Watching, Clause Learning, and Non-chronological Backtracking. These were successfully implemented as discussed above. However, during the process, we implemented a branching heuristic that chooses the next variable on which to make a decision randomly. When all of the tests are run together, they take about 30 minutes to complete. If we were to do this again, we would add more intelligent branching heuristics and try to reduce the number of times that we traverse the vectors. In the end, we have completed a solver that correctly resolves all test in a reasonable amount of time.

Appendix

Solver Output:

```
bin/test.sh bin/760sat tests/

tests/largeBenchmarks/SAT/ais/ais8.cnf:

s SATISFIABLE
v −1 2 −3 −4 −5 −6 −7 −8 −9 −10 −11 −12 −13 −14 −15 16 17 −18 −19 −20 −21 −22 −23 −24 −25
     −26 −27 −28 −29 30 −31 −32 −33 −34 −35 −36 37 −38 −39 −40 −41 −42 43 −44 −45 −46 −47
     −48 −49 −50 −51 −52 −53 −54 55 −56 −57 −58 −59 60 −61 −62 −63 −64 −65 −66 −67 −68 −69
     70 −71 −72 −73 −74 −75 −76 −77 78 −79 −80 −81 −82 83 −84 −85 86 −87 −88 −89 −90 −91 −92
      −93 94 −95 −96 −97 −98 −99 −100 −101 −102 103 −104 −105 −106 −107 −108 109 −110 −111
     −112 −113 0

real    0m0.005s
user    0m0.004s
sys     0m0.001s


tests/largeBenchmarks/SAT/hanoi/hanoi4.cnf:

s SATISFIABLE
v −1 −2 −3 −4 5 −6 7 8 −9 −10 −11 −12 13 −14 −15 16 17 18 −19 −20 −21 22 23 24 25 26 −27 −28
      −29 −30 31 32 −33 −34 −35 −36 −37 −38 −39 40 −41 −42 43 −44 45 46 −47 48 49 −50 −51 52
      53 54 55 −56 −57 −58 59 60 61 62 63 −64 −65 66 67 −68 −69 −70 71 −72 73 74 −75 −76 −77
      −78 79 −80 81 −82 −83 84 −85 −86 87 −88 −89 90 91 92 −93 −94 −95 96 97 98 99 −100 101
     −102 −103 −104 105 106 −107 −108 −109 −110 −111 −112 113 −114 −115 −116 117 −118 119
     120 −121 −122 −123 −124 125 −126 127 128 −129 −130 −131 −132 133 −134 −135 136 137 138
     −139 −140 −141 −142 143 144 145 −146 147 −148 149 −150 −151 152 153 −154 −155 156 −157
     −158 −159 −160 −161 −162 163 164 165 166 −167 −168 −169 −170 171 −172 173 −174 −175
     −176 −177 −178 179 −180 181 182 183 184 −185 −186 −187 −188 189 190 191 −192 193 −194
     195 196 −197 198 −199 −200 −201 −202 −203 −204 205 −206 −207 −208 −209 210 211 212 −213
      −214 −215 216 217 −218 219 −220 −221 −222 −223 −224 225 226 −227 −228 229 −230 −231
     −232 −233 −234 235 236 237 −238 239 −240 −241 242 −243 244 −245 −246 −247 −248 −249
     −250 −251 −252 −253 254 255 256 −257 258 −259 260 −261 −262 263 −264 265 −266 −267 −268
      −269 −270 271 −272 −273 −274 275 −276 −277 −278 −279 −280 281 282 283 −284 285 −286
     287 288 289 −290 −291 −292 −293 −294 295 −296 −297 −298 −299 −300 301 −302 303 304 305
     −306 −307 −308 309 −310 −311 312 313 −314 −315 −316 317 −318 −319 −320 321 −322 −323
     −324 −325 −326 327 328 329 −330 331 −332 −333 334 335 −336 −337 −338 −339 −340 −341 342
      −343 −344 −345 −346 347 −348 349 350 −351 −352 −353 −354 355
−356 −357 −358 359 −360 −361 −362 363 −364 365 −366 367 −368 −369 −370 −371 372 373 −374 375
      −376 377 −378 −379 380 381 −382 −383 −384 −385 386 −387 −388 −389 −390 −391 −392 −393
     394 395 396 −397 398 −399 −400 401 −402 −403 −404 405 −406 −407 −408 409 −410 −411 −412
      413 −414 −415 −416 −417 418 −419 420 421 −422 423 −424 −425 426 427 −428 −429 −430
     −431 −432 −433 −434 −435 −436 −437 438 439 −440 441 442 −443 −444 −445 −446 447 −448
     −449 −450 451 −452 −453 −454 455 −456 457 −458 459 −460 −461 −462 −463 464 465 −466 467
      −468 −469 −470 −471 472 473 −474 −475 −476 −477 478 −479 −480 −481 −482 −483 −484 −485
      486 487 488 −489 −490 −491 492 493 −494 −495 −496 497 −498 −499 −500 501 −502 −503
     −504 505 −506 −507 −508 −509 −510 511 −512 513 −514 −515 −516 517 −518 −519 520 −521
     −522 −523 −524 −525 −526 −527 −528 −529 530 531 532 −533 534 535 −536 −537 −538 539 540
      541 −542 −543 −544 −545 −546 547 −548 −549 −550 551 −552 −553 −554 −555 −556 557 −558
     559 −560 −561 −562 −563 −564 −565 566 −567 −568 −569 −570 −571 −572 573 −574 −575 −576
     577 578 −579 −580 581 582 583 −584 585 −586 587 −588 −589 −590 −591 −592 −593 −594 −595
      −596 597 −598 −599 −600 −601 −602 603 −604 605 −606 −607 −608 −609 −610 −611 612 −613
     −614 −615 −616 −617 −618 −619 −620 621 −622 623 −624 −625 −626 627 −628 629 −630 −631
     −632 −633 −634 −635 −636 −637 638 639 −640 −641 −642 643 −644 −645 −646 −647 −648 649
     −650 651 −652 −653 −654 −655 −656 −657 658 −659 −660 −661 −662 663 −664 −665 −666 −667
     −668 669 −670 671 −672 −673 −674 675 −676 −677 −678 −679 680 −681 −682 −683 −684 685
     −686 −687 −688 689 −690 −691 −692 −693 694 695 −696 −697 698 −699 −700 −701 −702 −703
     704 −705 −706 −707 −708 −709 710 −711 −712 −713 −714 715 −716 717 718 0

real    0m0.154s
user    0m0.154s
sys     0m0.000s


tests/largeBenchmarks/SAT/parity/par16−3.cnf:

s SATISFIABLE
v 1 2 3 4 5 −6 −7 −8 −9 10 11 12 13 14 15 16 17 −18 −19 20 21 22 23 24 25 26 −27 −28 −29 −30
      31 32 33 34 35 36 37 38 39 40 41 42 −43 44 45 46 47 −48 −49 −50 −51 −52 −53 54 55 56
     −57 −58 −59 60 61 62 63 64 −65 −66 −67 −68 −69 −70 −71 −72 −73 −74 −75 −76 77 −78 −79
     −80 −81 −82 −83 −84 −85 −86 −87 88 89 90 91 92 93 −94 −95 −96 −97 −98 99 100 101 102
     −103 −104 105 106 107 −108 −109 −110 −111 −112 −113 −114 −115 116 117 118 119 −120 −121
      −122 −123 −124 −125 −126 −127 −128 129 130 131 132 −133 −134 −135 −136 −137 −138 139
     140 141 142 143 144 145 −146 −147 −148 −149 −150 −151 −152 −153 154 155 −156 −157 −158
     159 160 161 −162 −163 −164 −165 −166 −167 −168 −169 −170 171 172 −173 −174 −175 176 177
      178 −179 −180 −181 −182 −183 −184 −185 −186 −187 188 189 −190 −191 −192 193 194 195
     −196 −197 −198 −199 −200 −201 −202 −203 −204 205 206 207 208 209 210 211 212 −213 −214
     −215 −216 −217 218 219 220 221 222 223 224 225 226 −227 −228 −229 −230 −231 −232 −233
     −234 −235 −236 −237 −238 −239 −240 −241 −242 −243 244 245 246 247 −248 −249 −250 −251
     252 253 254 255 −256 −257 −258 −259 −260 −261 −262 −263 264 265 266 267 268 269 270 271
      272 −273 −274 275 276 277 −278 −279 −280 281 −282 −283 −284 −285 −286 −287 −288 −289
     290 291 −292 −293 −294 295 296 297 −298 −299 −300 −301 −302 −303 −304 −305 −306 −307
     −308 309 310 311 312 313 314 315 −316 −317 −318 −319 −320 −321 −322 −323 −324 −325 −326
      −327 −328 −329 −330 −331 −332 −333 −334 −335 −336 337 338 339 340 −341 −342 −343 −344
     −345 −346 −347 −348 349 −350 −351 −352 −353 354 355 356 357 358 359 −360 −361 −362 −363
      −364 −365 366 367 368 369 370 371 372 373 374 −375 −376 377 378 379 −380 −381 −382 383
      384 385 386 387 388 389 390 391 −392 −393 394 395 396 −397 −398 −399 400 401 402 403
     404 −405 −406 −407 −408 409 410 −411 −412 −413 414 415 416 417 −418 −419 −420 −421 −422
      −423 −424 −425 426 427 −428 −429 −430 431 432 433 434 435 436 437 438 −439 −440 −441
     −442 443 444 445 446 447 −448 −449 −450 −451 452 453 454 455 −456 −457 −458 −459 −460
     −461 462 463 464 −465 −466 −467
−468 −469 −470 −471 −472 473 474 475 476 477 478 479 480 481 −482 −483 −484 −485 486 487 488
      489 −490 −491 −492 −493 −494 −495 496 497 498 499 500 501 502 −503 −504 −505 −506 −507
      −508 −509 −510 511 512 −513 −514 −515 −516 −517 −518 −519 −520 −521 −522 −523 524 525
     526 527 −528 −529 530 531 532 533 534 535 −536 −537 −538 −539 −540 −541 −542 −543 −544
     −545 546 −547 −548 549 −550 −551 552 553 −554 −555 −556 557 −558 −559 −560 561 −562 563
      −564 565 −566 567 −568 569 570 −571 572 −573 574 575 576 577 −578 579 −580 581 582 583
      584 −585 586 −587 588 589 590 591 −592 593 −594 595 596 597 598 −599 600 −601 602 −603
      −604 −605 606 607 −608 609 610 −611 −612 613 614 −615 616 617 −618 −619 620 621 −622
     623 624 −625 −626 627 628 −629 630 631 −632 −633 634 635 −636 637 638 −639 −640 641 642
      −643 644 645 −646 −647 648 649 −650 651 −652 653 −654 655 656 −657 658 −659 660 −661
     662 663 −664 665 666 667 −668 669 670 −671 672 −673 −674 675 676 677 −678 679 −680 −681
      682 683 684 −685 686 −687 −688 689 690 691 −692 693 −694 −695 696 697 698 −699 700 701
      −702 703 704 705 −706 707 −708 709 710 711 712 −713 714 715 716 717 718 719 −720 721
     −722 −723 −724 −725 −726 727 728 −729 −730 −731 −732 −733 734 735 −736 −737 −738 −739
     −740 741 742 −743 −744 −745 −746 −747 748 749 −750 −751 −752 −753 −754 755 756 757 −758
      −759 −760 −761 762 763 764 −765 −766 −767 −768 769 770 771 −772 −773 −774 −775 776 777
      −778 779 −780 −781 −782 783 784 −785 786 −787 −788 −789 790 791 792 793 −794 −795 −796
      −797 −798 799 −800 −801 −802 −803 −804 −805 −806 −807 −808 −809 −810 −811 −812 −813
     −814 −815 −816 −817 −818 −819 −820 −821 −822 −823 −824 −825 −826 827 828 829 830 −831
     −832 −833 834 −835 −836 −837 −838 −839 −840 −841 −842 −843 −844 −845 −846 −847 −848
     −849 −850 −851 −852 −853 −854 −855 −856 −857 −858 −859 −860 −861 −862 −863 −864 −865
     −866 −867 −868 −869 −870 −871 −872 −873 −874 −875 876 877 −878 −879 −880 −881 −882 −883
      −884 −885 −886 −887 −888 −889 890 −891 −892 −893 −894 −895 −896 897 898 899 −900 −901
     −902 −903 −904 −905 −906 −907 −908 −909 −910 −911 −912 −913 −914 −915 −916 −917 −918
     −919 −920 −921 −922 −923 −924 925 −926 −927 −928 −929 −930 −931 932 933 −934 −935 −936
     −937 −938 939 −940 −941 −942 −943 −944 −945 946 947 948 949 950 951 −952 −953 −954 −955
      −956 −957 −958 −959 −960 −961 −962 −963 −964 −965 −966 −967 −968 −969 −970 −971 −972
     −973 −974 −975 −976 −977 −978 −979 −980 981 −982 −983 −984 −985 −986 −987 −988 −989
     −990 −991 −992 −993 −994 −995 −996 −997 −998 −999 −1000 −1001 1002 1003 −1004 −1005
     −1006 −1007 −1008 −1009 −1010 −1011 −1012 −1013 −1014 −1015 0

real    0m5.288s
user    0m5.282s
sys     0m0.002s


tests/largeBenchmarks/UNSAT/bejing/2bitadd_10.cnf:

s UNSATISFIABLE

real    27m42.467s
user    27m40.563s
sys     0m0.718s


tests/largeBenchmarks/UNSAT/dubois/dubois25.cnf:

s UNSATISFIABLE

real    0m0.640s
user    0m0.635s
```

```
sys     0m0.001s


tests/largeBenchmarks/UNSAT/phole/hole6.cnf:

s UNSATISFIABLE

real    1m58.203s
user    1m58.130s
sys     0m0.000s


tests/smallBenchmarks/SAT/sanity/sanity2.cnf:

s SATISFIABLE
v -1 2 0

real    0m0.004s
user    0m0.000s
sys     0m0.002s


tests/smallBenchmarks/SAT/sanity/sanity3.cnf:

s SATISFIABLE
v 1 2 -3 4 -5 0

real    0m0.001s
user    0m0.000s
sys     0m0.000s


tests/smallBenchmarks/SAT/tiny/rand10_10.cnf:

s SATISFIABLE
v 1 2 -3 4 -5 -6 7 -8 9 -10 0

real    0m0.003s
user    0m0.001s
sys     0m0.001s


tests/smallBenchmarks/SAT/tiny/rand10_20.cnf:

s SATISFIABLE
v -1 -2 -3 4 -5 6 7 -8 9 -10 0

real    0m0.001s
user    0m0.000s
sys     0m0.001s


tests/smallBenchmarks/SAT/tiny/rand20_30.cnf:

s SATISFIABLE
v 1 2 -3 4 5 -6 7 -8 -9 -10 -11 12 13 -14 15 16 17 -18 -19 -20 0

real    0m0.003s
user    0m0.002s
sys     0m0.000s


tests/smallBenchmarks/SAT/tiny/rand25_40.cnf:

s SATISFIABLE
v 1 -2 -3 -4 -5 6 -7 8 9 10 -11 12 -13 14 -15 -16 -17 18 -19 20 -21 22 23 -24 -25 0

real    0m0.003s
user    0m0.002s
sys     0m0.000s


tests/smallBenchmarks/SAT/tiny/rand30_50.cnf:
```

Appendix continued on next page. . .

```
s SATISFIABLE
v −1 2 −3 4 5 −6 7 8 −9 −10 −11 −12 13 14 15 −16 17 18 19 −20 −21 −22 23 −24 25 26 −27 −28
     29 30 0

real    0m0.003s
user    0m0.002s
sys     0m0.000s


tests/smallBenchmarks/SAT/tiny/rand3_3.cnf:

s SATISFIABLE
v −1 2 3 0

real    0m0.002s
user    0m0.000s
sys     0m0.001s


tests/smallBenchmarks/SAT/tiny/rand3_6.cnf:

s SATISFIABLE
v −1 2 −3 0

real    0m0.002s
user    0m0.000s
sys     0m0.002s


tests/smallBenchmarks/SAT/tiny/rand4_6.cnf:

s SATISFIABLE
v −1 2 3 4 0

real    0m0.003s
user    0m0.000s
sys     0m0.001s


tests/smallBenchmarks/SAT/tiny/rand4_8.cnf:

s SATISFIABLE
v −1 −2 3 −4 0

real    0m0.003s
user    0m0.000s
sys     0m0.002s


tests/smallBenchmarks/SAT/tiny/rand5_10.cnf:

s SATISFIABLE
v −1 2 −3 4 −5 0

real    0m0.003s
user    0m0.001s
sys     0m0.000s


tests/smallBenchmarks/SAT/tiny/rand5_20.cnf:

s SATISFIABLE
v −1 2 −3 4 −5 0

real    0m0.003s
user    0m0.001s
sys     0m0.000s


tests/smallBenchmarks/SAT/tiny/rand5_5.cnf:

s SATISFIABLE
```

Appendix continued on next page. . .

```
v −1 −2 −3 4 −5 0

real    0m0.003s
user    0m0.001s
sys     0m0.001s


tests/smallBenchmarks/UNSAT/sanity/sanity1.cnf:

s UNSATISFIABLE

real    0m0.002s
user    0m0.000s
sys     0m0.001s


tests/smallBenchmarks/UNSAT/sanity/sanity4.cnf:

s UNSATISFIABLE

real    0m0.002s
user    0m0.001s
sys     0m0.000s


tests/smallBenchmarks/UNSAT/sanity/sanity5.cnf:

s UNSATISFIABLE

real    0m0.002s
user    0m0.002s
sys     0m0.000s


tests/smallBenchmarks/UNSAT/tiny/rand10_100.cnf:

s UNSATISFIABLE

real    0m0.002s
user    0m0.002s
sys     0m0.000s


tests/smallBenchmarks/UNSAT/tiny/rand10_50.cnf:

s UNSATISFIABLE

real    0m0.002s
user    0m0.001s
sys     0m0.000s


tests/smallBenchmarks/UNSAT/tiny/rand5_30.cnf:

s UNSATISFIABLE

real    0m0.002s
user    0m0.001s
sys     0m0.001s
```