Logic and Computation Project 1 Report

Developing an Educational Web Application for Boolean Satisfiability Solvers

[redacted]

[redacted]

Due Date: 11/9/2020

Introduction

This project focuses on educating students about simple boolean satisfiability (SAT) solvers, which are programs that test whether a propositional logic formula is ever true by designating the correct true or false value to each unassigned variable. In order to understand our motivation for this project, all that one must do is simply use a search engine to query a phrase to the tune of "SAT solver explained." The top results are a slew of reading material including course lecture notes, research papers, online forum answers, and of course, the Wikipedia page on the topic. Though a determined student would undoubtedly be able to use the aforementioned resources to gain a solid understanding of the material, the process would be inefficient and would require a relatively high level of motivation. As a result, many of those who would benefit a great deal from gaining exposure to a topic like SAT solving, which lies outside the scope of their previous course material, would not do so because of the hassle associated with the task.

In the case of Northeastern University, undergraduates who have taken the Fundamentals of Computer Science I course and have exposure to discrete mathematics, either through the institution's Discrete Structures course or elsewhere, are prepared to learn about SAT solving. However, only those who take courses that include this content in their curricula, such as Logic and Computation, will be guaranteed to learn about boolean satisfiability solvers.

After investigating possible improvements to the quality of free educational resources for SAT solving, it was determined that adding a degree of interactivity to our tool would greatly benefit students. This decision was justified by research comparing interactive learning to other common approaches in education. In a paper published in the peer-reviewed journal *Advances in Medical Education and Practice*, first-year computer science students and dentistry students were studied by comparing feedback from classes with and without the PollEverywhere Audience Response System (ARS). Of those studied, 95% reported that it made their thought-processes clearer, among other benefits, and 81.7% stated that it "improved their motivation to learn."[1] Though this research did specifically study first-year computer science students, one could argue that it focused on in-person learning and is therefore not applicable to online educational tools.

However, another study from Carnegie Mellon University concluded that interactive activities in online courses helped make students six times more likely to learn the given material.[2] More specifically, the study compared "Massive Open Online Courses" (MOOCs), which solely rely on passive learning through the likes of instructional videos, to a similar curriculum that also included interactive activities. The study used eleven weekly quizzes, a final exam, and the number of students who dropped the course as metrics. As a result, it was hypothesized that a simple interactive web-app had potential to improve the current online landscape of free educational tools that teach students about SAT solving.

[1] Collins and Meguid, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5364003/
[2] Koedinger, et. al., https://dl.acm.org/doi/pdf/10.1145/2724660.2724681

Approach

As previously touched upon, current educational tools to understand boolean satisfiability solvers are limited to passive learning approaches, where all students must do is read and watch content. After examining the popular resources on the web that were available to students, it was determined that a web page[3] published by the University of Buffalo was among the most helpful of the resources that had been available. The page provides a video to help explain what a satisfiability problem is and then walks through an example problem using Lingeling,[4] an efficient SAT solver that won the award for most impressive performance in the 2014 SAT Competition.[5] More specifically, given a set of courses and their respective prerequisites, the page utilized the Lingeling algorithm to determine if taking all courses would be possible.

Unlike the website made by the University of Buffalo, the traditional approach to SAT-solving has been to utilize the DPLL[6] algorithm. Rather than just walking through the act of plugging in a boolean formula to a more efficient procedure, this project aims to explain this process. As a result, the simpler DPLL approach was used, in order to maximize digestibility of the material for a beginning student. In other words, this project did not seek to improve the commonly-used algorithm; instead, the goal was to simplify and explain it through clear tutorials, hints, and an achievable degree of interactivity, given the time constraints. Furthermore, there is a standard for teaching functional programming commonly taught to undergraduate students (HtDP)[7] that was used throughout all example code, in order to promote good programming practices and be consistent.

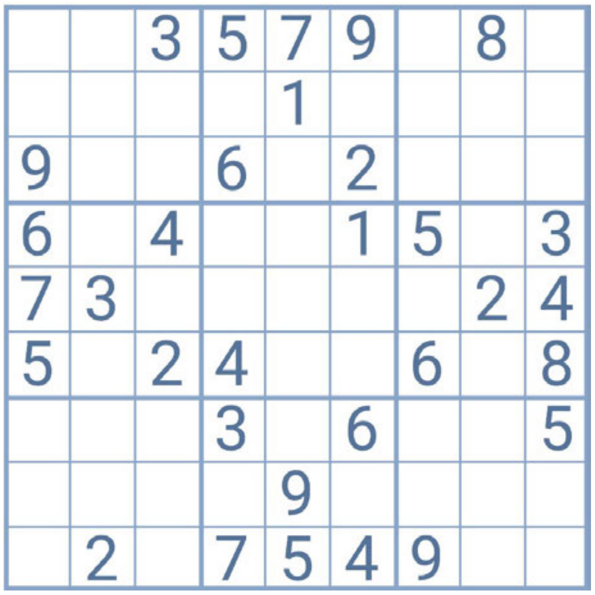
The project's choice of language assumed that the reader had already gone through approximately two-thirds of the course, up until the introduction of lambdas. The reader was expected to be proficient in Intermediate Student Language syntax and familiar with recursion, along with the design recipe. As a result, ISL with lambdas was used, due to its ability to use 'local' and 'lambda'. Since 'let' and its related bindings are typically not used in Fundies I, the two aforementioned local bindings were solely used, which still granted all the needed functionality that 'let' or 'letrec' provided.[8] By defining data structures for each type of boolean expression, 'cond' statements were used instead of 'case-match' statements with identifiers for each data type. The SAT solver written in ISL took a boolean formula, found the expression's free variables, and mimicked the DPLL algorithm by testing the value of the formula and substituting each boolean value for every free variable (Figure 5 in Appendix). Additionally, the boolean formula was not constrained to having to be in CNF; this allowed for students to understand the DPLL through a variant of the algorithm that did not necessitate them to recall what CNF was, adding unnecessary difficulty to the problem.

This web application[9] contained intuitive notes and hints to guide the reader to the correct implementation of the DPLL algorithm. More specifically, walkthroughs of the algorithm and implementation for

[3] https://cse.buffalo.edu/~erdem/cse331/support/sat-solver/index.html
[4] Biere, https://github.com/arminbiere/lingeling
[5] http://satcompetition.org/2014/SAT-COMP.pdf
[6] Tinelli, http://homepage.cs.uiowa.edu/~tinelli/classes/196/Fall09/notes/dpll.pdf
[7] Felleisen, et al., https://htdp.org/2019-02-24/
[8] https://docs.racket-lang.org/htdp-langs/intermediate.html#%28part._intermediate-syntax%29
[9] Jung and Simon, https://github.com/rymaju/fundies-dpll

different example problems were provided. React was used as the frontend JSX framework for the web application and Prism.js[10] was used to highlight Racket code. Though the web application was initially intended to contain a basic integrated development environment (IDE), it was determined that the marginal benefit of adding such a feature was not worth the time it would consume. Students were expected to have DrRacket installed anyway, and all of the provided code was copyable, so adding an IDE was deemed to be unnecessary.

In terms of the specific structure of the tutorial, the user was first asked to solve a difficult Sudoku puzzle (Figure 1) and from there was introduced to the concept of boolean satisfiability in the context of the game. After walking through what the rules of Sudoku look like in propositional logic, the immensity of the resulting boolean formula for the puzzle was demonstrated. This was done without spending an unreasonable amount of time combining and simplifying the constraints using boolean algebra.



		3	5	7	9		8	
				1				
9			6		2			
6		4			1	5		3
7	3						2	4
5		2	4			6		8
			3		6			5
				9				
	2		7	5	4	9		

Fig. 1: Example Sudoku Problem[11]

Upon establishing why a computer program would be needed to determine if such a problem was satisfiable, the user was introduced to the DPLL algorithm. The procedure's pseudocode was first walked through, followed by the aforementioned ISL implementation, which was broken down into digestible explanations of each function. After describing in full sentences what a given function should do, students were also given the opportunity to copy the function template and attempt to write it themselves in DrRacket. Furthermore, each function came with multiple choice quizzes pertaining to its check-expects, in order to engage the user and promote interactive learning.

[10] https://prismjs.com/

[11] "Impossible Sudoku Puzzles - Play Sudoku Online." *Sudoku Online*, www.sudokuonline.io/impossible.

For the conclusion for the tutorial, the user is asked to implement the code to determine whether a configuration of light switches and logic gates would ever allow for the given bulb to turn on (Figure 2). Though the student is expected to find the solution, the boolean formula describing the problem was provided, and a diagram of a solution is provided (Figure 4 of Appendix).

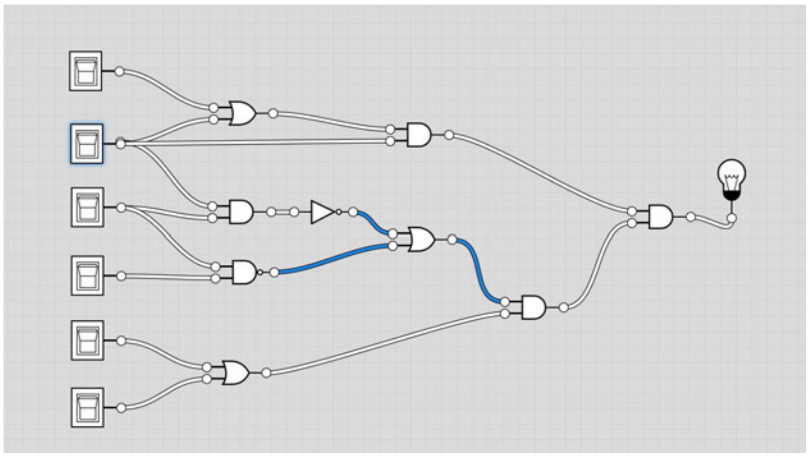


Fig. 2: Switch Problem Diagram

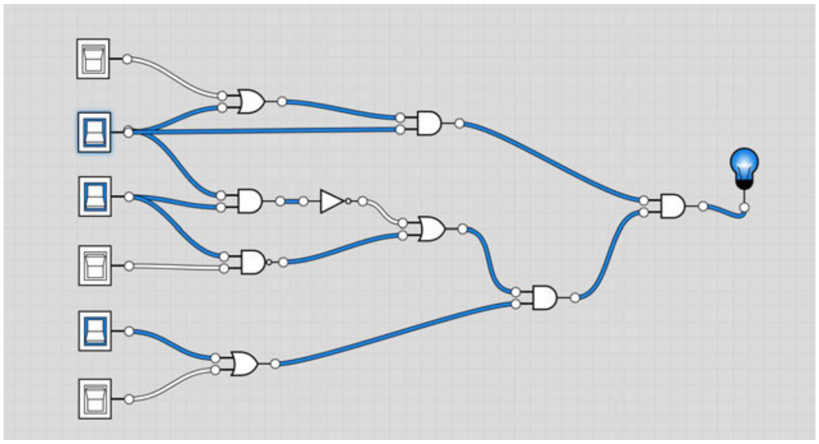
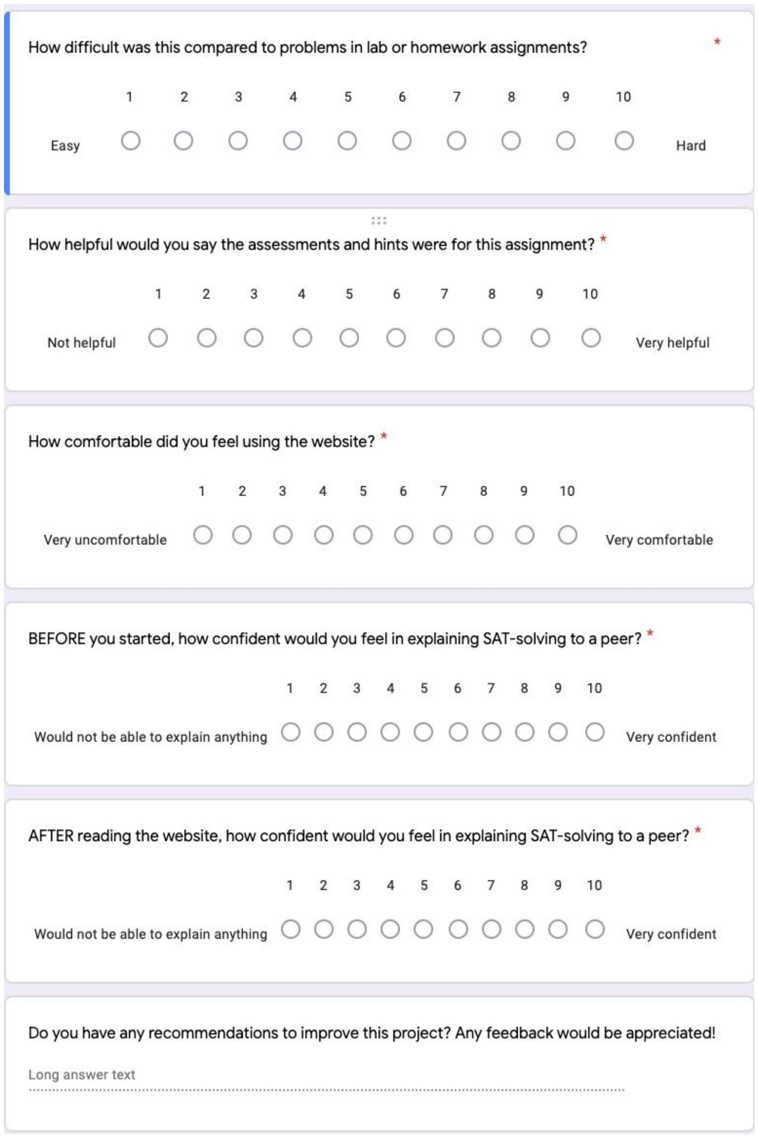


Fig. 3: Diagram of Example Solution for Switch Problem

As an extra challenge, students were provided two challenge questions to work on, in order to help solidify the newly learned material. The first problem asked the user to redesign the substitute function so that it did not naively investigate clauses which would not affect the boolean value of the expression that could already be determined. The second problem then provided a new data definition for a boolean formula that necessitated it to be in CNF and asked the student to rewrite the SAT solver to accommodate the redefined data.

Once the web application was developed, it was hosted on Netlify.[12] In terms of understanding the tool's efficacy, ten Fundies 1 students tested the tool and filled out a survey with a mix of qualitative and numerical feedback, where users assigned a score from 1 to 10 for five questions (Figure 4).



How difficult was this compared to problems in lab or homework assignments? *

Easy 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10 ○ Hard

How helpful would you say the assessments and hints were for this assignment? *

Not helpful 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10 ○ Very helpful

How comfortable did you feel using the website? *

Very uncomfortable 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10 ○ Very comfortable

BEFORE you started, how confident would you feel in explaining SAT-solving to a peer? *

Would not be able to explain anything 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10 ○ Very confident

AFTER reading the website, how confident would you feel in explaining SAT-solving to a peer? *

Would not be able to explain anything 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10 ○ Very confident

Do you have any recommendations to improve this project? Any feedback would be appreciated!

Long answer text

Fig. 4: Survey Questions

When designing the questions for the small study, the metrics of this project were broken down into the following categories:

[12] Jung and Simon, https://dpll.netlify.app/

1. Was the site usability and tutorial readability acceptable for users?
    a. This was gauged by observing the responses to the second and third questions on the survey.
2. Was the level of difficulty associated with completing the tutorial and its corresponding assignments appropriate?
    a. This was determined by the first, and contrasting fourth with fifth questions on the survey.

3. What areas of the tool need improvement?
    a. This was assessed by the qualitative feedback from the final survey question.

The ordering of the aforementioned metrics is also indicative of how they were prioritized. The measure of readability and usability for our project preceded any other metric for the tool. Additionally, the level of difficulty associated with completing the tutorial was deemed to be the second most important metric. This prioritization was made due to the fact that the material needed to introduce enough material that it challenged students intellectually, while not reducing the tool's accessibility by overwhelming them. As a result of the relative objectivity that a rating survey offered, the qualitative feedback pertaining to how the tool could be improved. Nevertheless, written feedback still proved to be quite valuable, due to the increased content depth that it allowed reviewers to provide, such as a potential new feature or a specific wording suggestion for one of the hints.

Methodology

Our process was divided into the following steps:

1. Code DPLL in ISL with Lambdas à la Fundies 1
2. Document and split code by function
3. For each function, write the corresponding prompts, hints, and an explanation that follows the design recipe
4. Create a basic web app, using React.js and Prism.js to highlight the Racket code,
    a. Design functions for displaying code and quizzes.
5. Insert the DPLL code, all written sections, and the quizzes into the code.
6. Test the efficacy of the tutorial on willing students
7. Write project report
8. Release and distribute the site for students, using free static site hosting offered by Netlify

As anticipated, the most challenging part of this methodology was ensuring that this tool provided educational value to students through its structure and content. Additionally, composing the introduction relating to Sudoku and the function walkthroughs were surprisingly time-consuming, due to the fact that the language needed to be refined multiple times. Despite the surprising challenge this posed, practicing the writing style required to clearly convey the educational content to the target audience proved to be an interesting undertaking.

Results

Though the variance in the survey's numerical responses was relatively high, and the sample was 8 students, the results still served as preliminary indication that the web application served as a successful first step in developing an interactive tool for learning about SAT solvers. In regards to the metric of site usability and readability, the student responses indicated that the assessment and hints were helpful, and they were generally comfortable using the tool (Figures 5 and 6).

For the survey's second question, the average response was approximately 8.25, where 10 was "very helpful" and 1 was "not helpful." Nevertheless, one user gave a 3 for the former question, which was considered to be an outlier in the data, considering that the next lowest rating was 7. When excluding this outlier, the rating increased to 9.

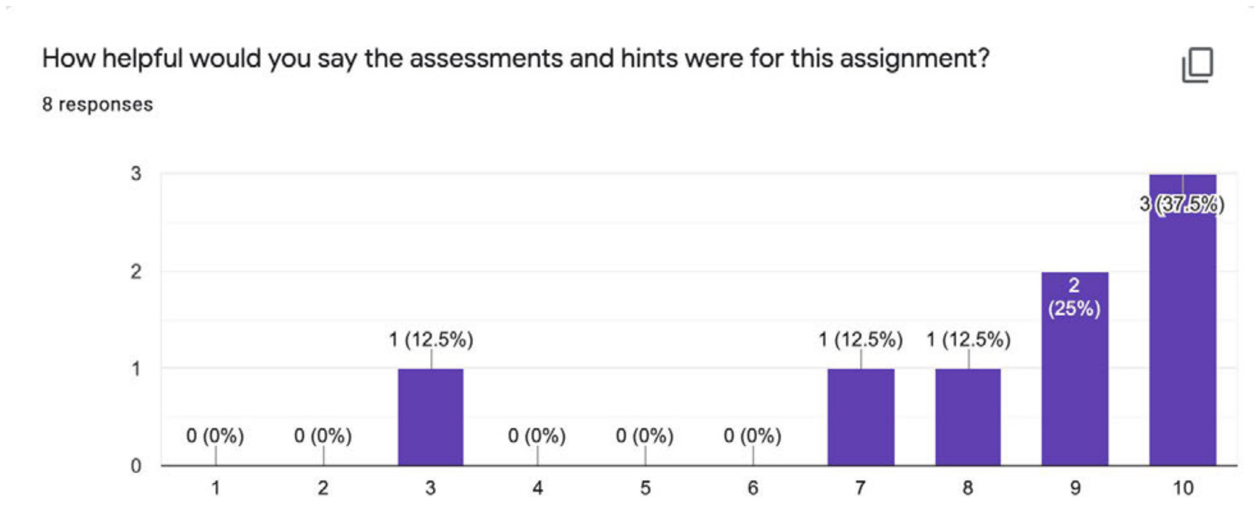


Fig. 5: Question 2 Survey Results

The third question yielded strong evidence that the web application was adequately easy to use. In fact, the average rating for this question was 9, where 10 represented "very comfortable" and 1 was "very uncomfortable."

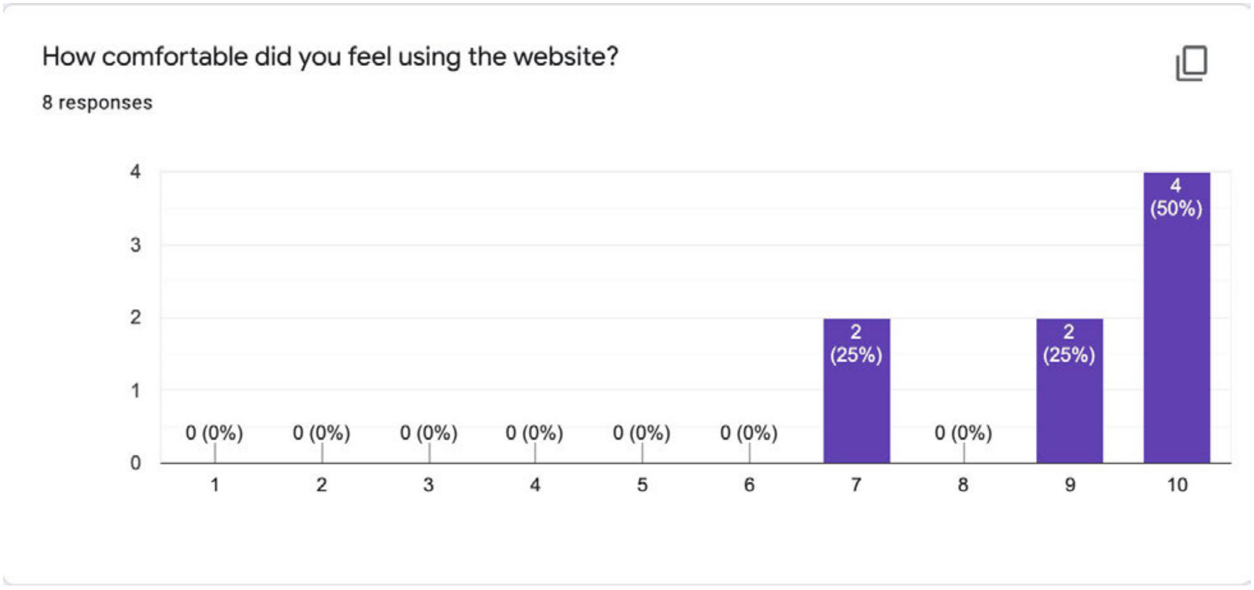


Fig. 6: Question 3 Survey Results

Regarding the second most important metric, which was the level of difficulty associated with the tutorial and its assignments, the questionnaire's results indicated that it was appropriately challenging, though there was room for improvement (Figures 7 - 9). The first question of the survey, which asked for the difficulty relative to homeworks and labs in Fundies I, generated varied responses; where 1 was "Easy" and 10 was "Hard," the users' ratings ranged from 2 to 9. Though the mean was 6, reaching an adequate level of challenge without limiting the tool's accessibility to the target audience, the variance in numerical responses was unideal. Instead, one would aim for such a tool to be less polarizing to different students in future iterations, however this variability could also have been a byproduct of a small sample size.

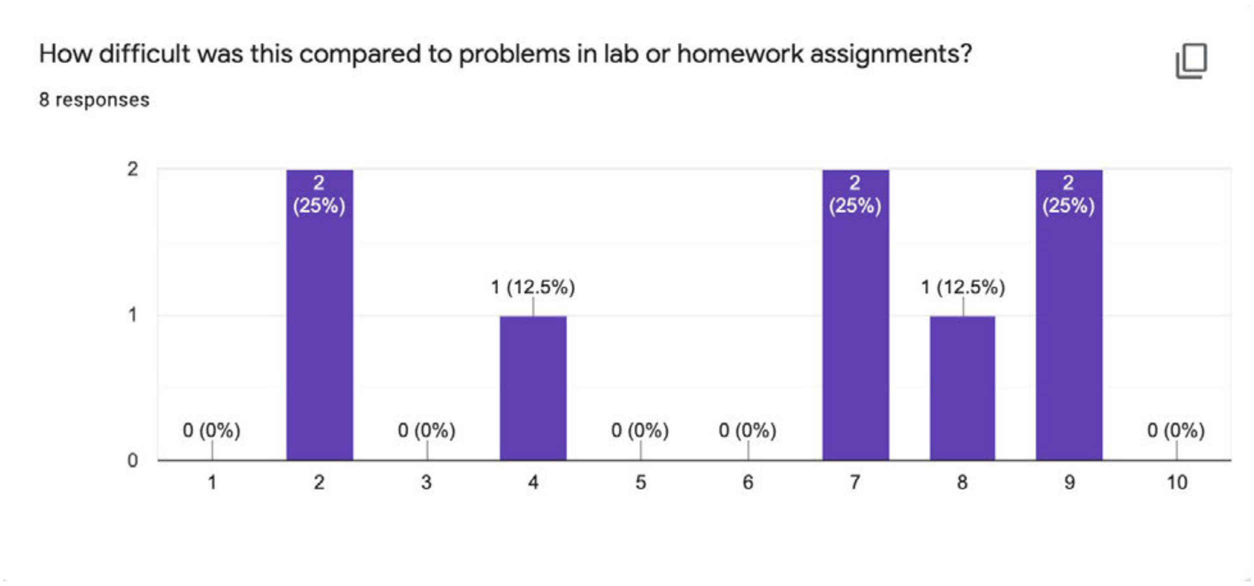


Fig. 7: Question 1 Survey Results

By analyzing the fourth and fifth questions together, additional insight was provided, relating to how much users learned on average after using the page. Overall, participants were more confident with SAT solvers than anticipated, though this increase from the expected level of knowledge on the topic was still only marginal and can once again be explained by the small sample size. More specifically, the participants on average responded with a 3, where a 1 signified that they "would not be able to explain anything," while a 10 meant that they were "very confident" in their knowledge of the material.

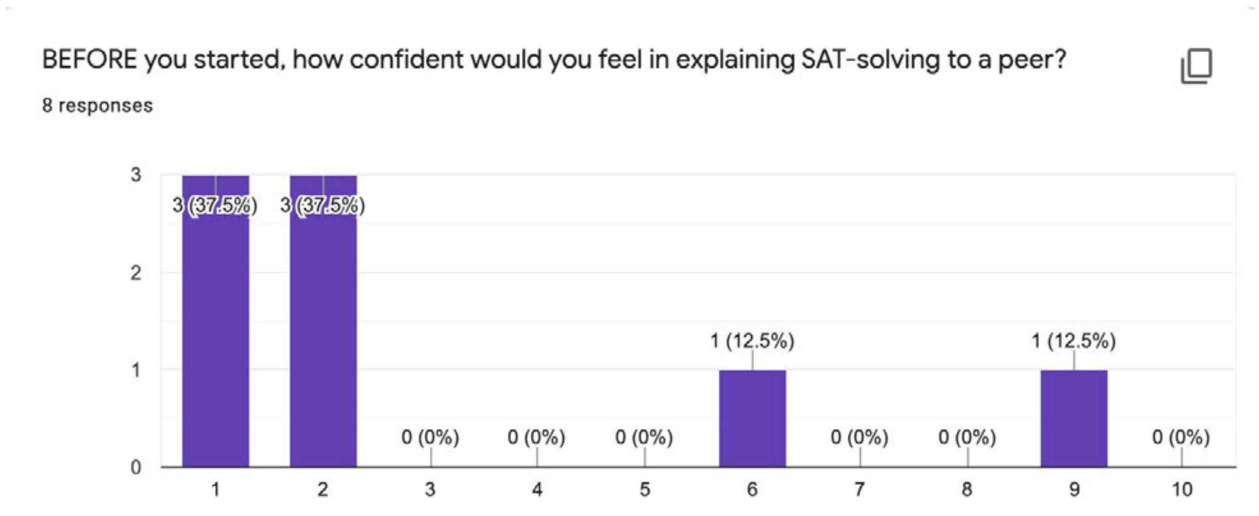


Fig. 8: Question 4 Survey Results

The results of the fifth question were acceptable though there was certainly room for improvement, with an average numerical response of 7.5 , where the scale holds the same meaning as in the prior question. The difference in means between the last two numerical responses of the survey was 4.5, indicating a satisfactory gain in knowledge of SAT solving, given the fact that this was the initial release of the tool.

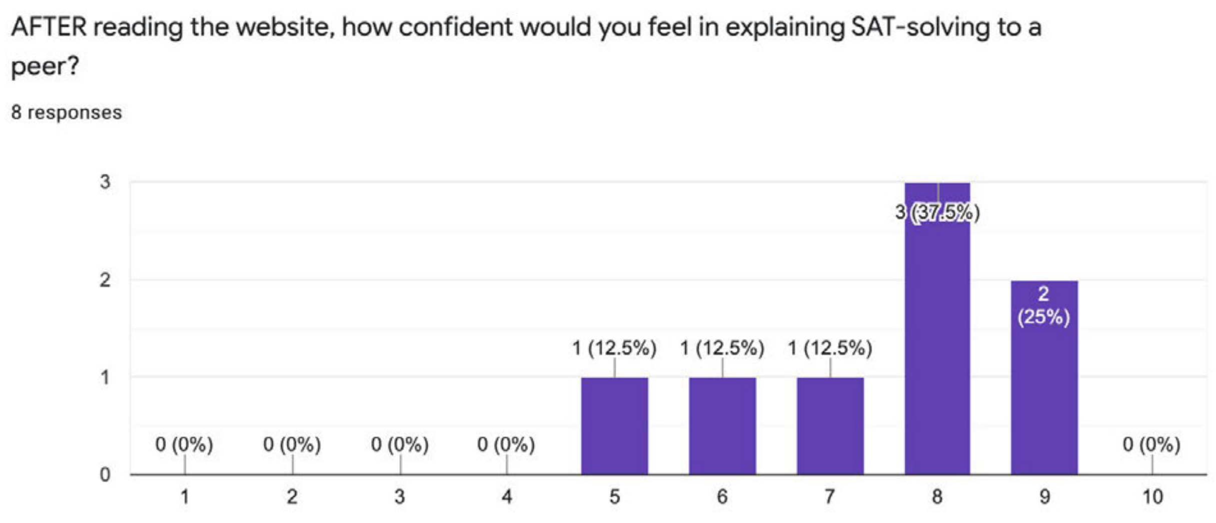


Fig. 9: Question 5 Survey Results

The third and final metric, which was qualitative user feedback, yielded two insights: adding an IDE would be more convenient for users in the future, and providing a table of contents would allow students to better understand the application's structure before fully immersing themselves in it.

Though this project did result in a relatively effective interactive tool to help introduce students to boolean satisfiability solvers, it could be further generalized. More specifically, one could increase the tool's target audience to anyone with an interest in computer science or apply a similar interactive approach to other fields. In regard to the latter concept, one could even go so far as to generate an entire virtual collection of accessible online tools that could be used to teach a plethora of subjects, with a focus on topics that remain underrepresented in the curricula of introductory computer science courses.

Furthermore, though the project was deemed to have met its expectations, and the study served its purpose of validating the tool's efficacy to a satisfactory degree, the sample size and the number of questions could be expanded in the future. In other words, given the scope and timeframe of this project, the existing data was found to be adequate, but this miniature study should ideally be further developed if one were to seek additional feedback on an improved version of the web application.

Summary

Unlike past approaches to teaching boolean satisfiability, this endeavor has offered an interactive approach, which studies have shown to be a viable replacement to tools that solely rely on passive learning. Furthermore, though the results have demonstrated that this web application has been successful in meeting the predetermined standards, there is a lot of progress to be made. The application's use of language could always be refined, an IDE could be implemented, and a table of contents could also be added, among other possible improvements. Along with further advancing the quality of the end product, larger scale studies could be conducted to gain further insight into how students can optimally learn on a web application, which could be further generalized to other topics in computer science and logic.

References

3 Intermediate Student, docs.racket-lang.org/htdp-langs/intermediate.html.

DPLL, www.cs.miami.edu/home/geoff/Courses/CSC648-12S/Content/DPLL.shtml.

Arminbiere. "Arminbiere/Lingeling." *GitHub*, github.com/arminbiere/lingeling.

Felleisen, Matthias, et al. *How to Design Programs, Second Edition*, 2019, htdp.org/2019-02-24/.

[redacted]

[redacted]

[redacted]

Koedinger, Kenneth R., et al. "Learning Is Not a Spectator Sport." Proceedings of the Second (2015) ACM Conference on Learning @ Scale - L@S '15, 2015, doi:10.1145/2724660.2724681.

Meguid, Eiman Abdel, and Matthew Collins. "Students' Perceptions of Lecturing Approaches: Traditional versus Interactive Teaching." *Advances in Medical Education and Practice*, Volume 8, 2017, pp. 229–241., doi:10.2147/amep.s131851.

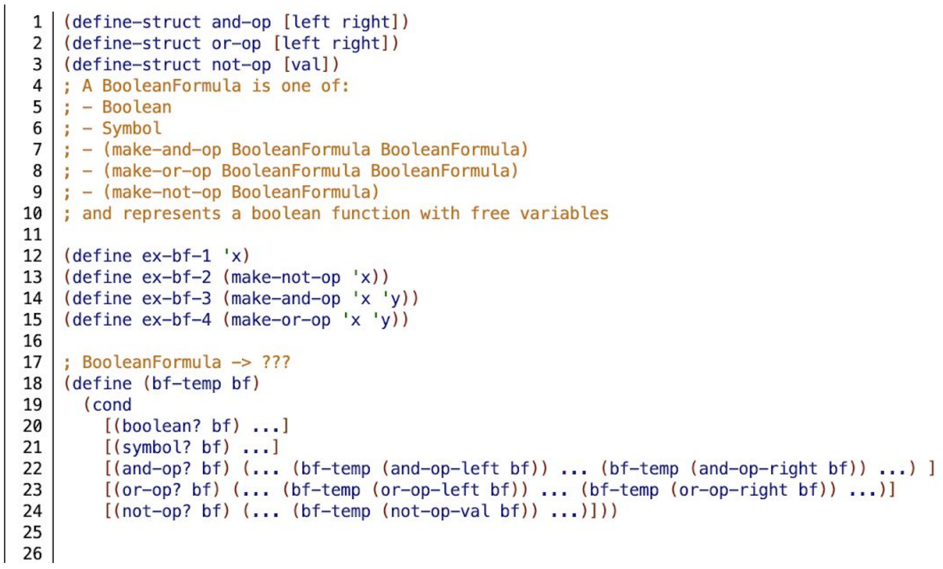
Prism, prismjs.com/.

"Satisfiability and SAT Solvers." *SAT and SAT Solvers*, cse.buffalo.edu/~erdem/cse331/support/sat-solver/index.html.

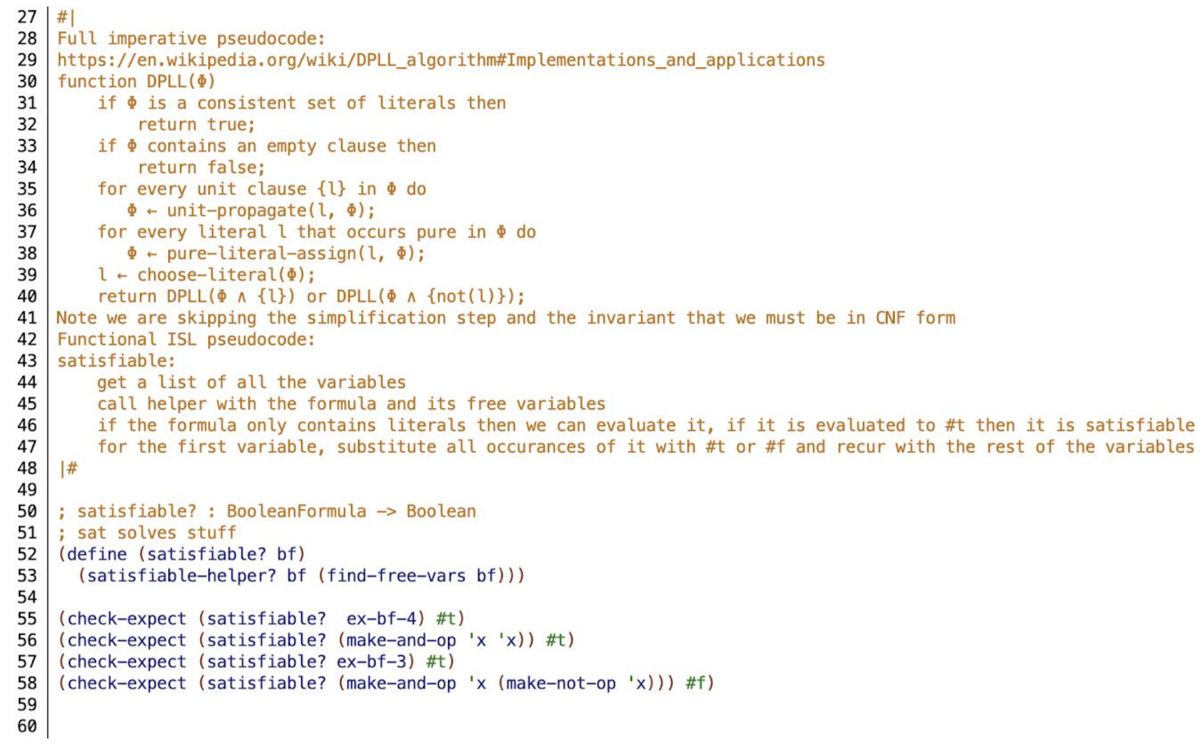
Tinelli, Cesare. "The DPLL Procedure." *University of Iowa College of Liberal Arts and Sciences*, 2009, homepage.cs.uiowa.edu/~tinelli/classes/196/Fall09/notes/dpll.pdf.

Appendix

Figure 10: Code for Satisfiability Checker in DrRacket ISL [13]

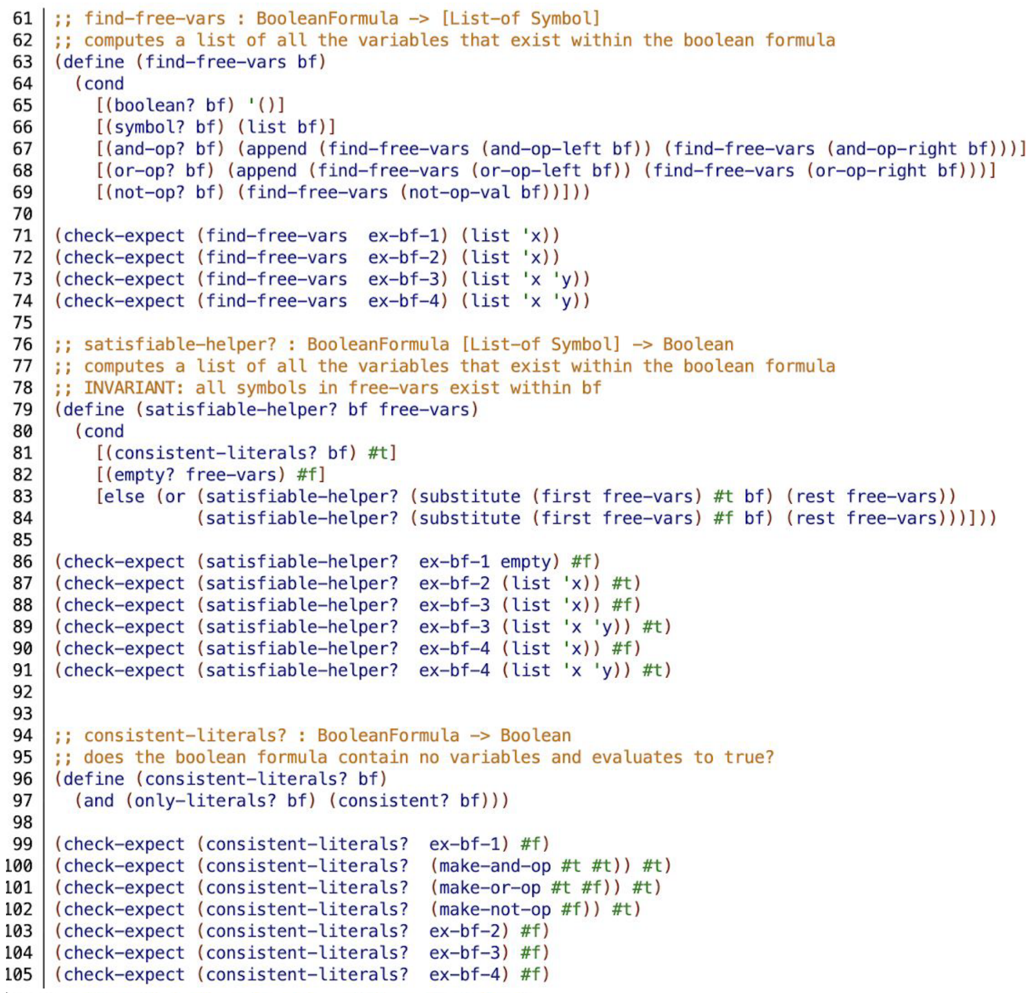


```
(define-struct and-op [left right])
(define-struct or-op [left right])
(define-struct not-op [val])
; A BooleanFormula is one of:
; - Boolean
; - Symbol
; - (make-and-op BooleanFormula BooleanFormula)
; - (make-or-op BooleanFormula BooleanFormula)
; - (make-not-op BooleanFormula)
; and represents a boolean function with free variables

(define ex-bf-1 'x)
(define ex-bf-2 (make-not-op 'x))
(define ex-bf-3 (make-and-op 'x 'y))
(define ex-bf-4 (make-or-op 'x 'y))

; BooleanFormula -> ???
(define (bf-temp bf)
  (cond
    [(boolean? bf) ...]
    [(symbol? bf) ...]
    [(and-op? bf) (... (bf-temp (and-op-left bf)) ... (bf-temp (and-op-right bf)) ...) ]
    [(or-op? bf) (... (bf-temp (or-op-left bf)) ... (bf-temp (or-op-right bf)) ...)]
    [(not-op? bf) (... (bf-temp (not-op-val bf)) ...)]))


```

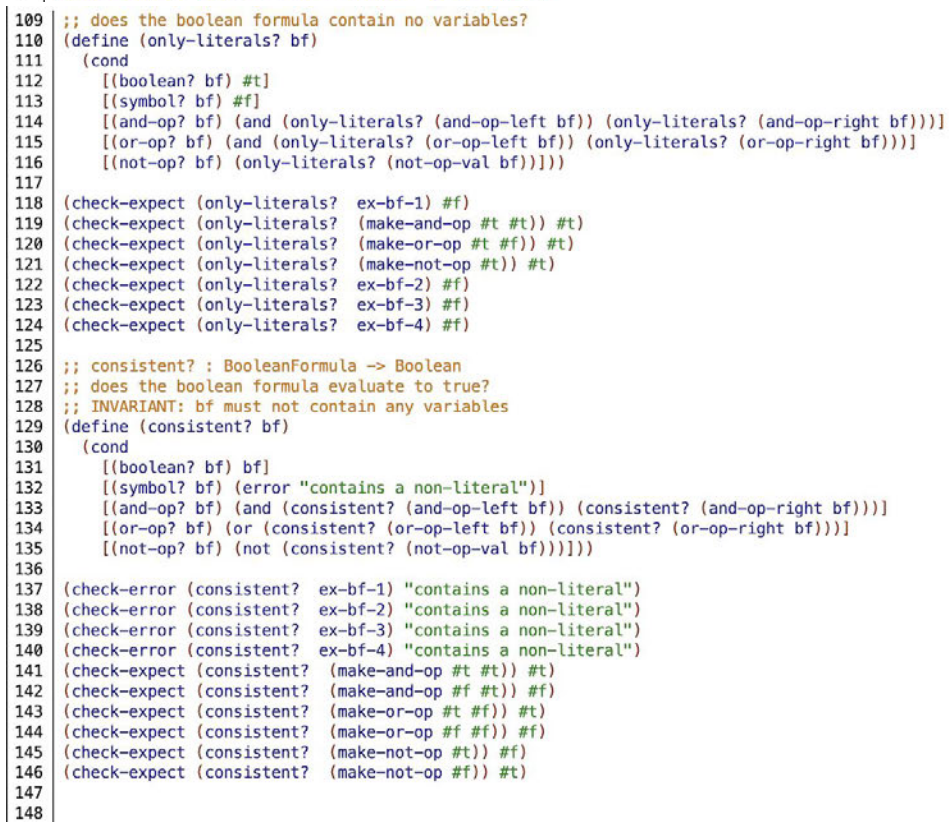


```
#|
Full imperative pseudocode:
https://en.wikipedia.org/wiki/DPLL_algorithm#Implementations_and_applications
function DPLL(Φ)
    if Φ is a consistent set of literals then
        return true;
    if Φ contains an empty clause then
        return false;
    for every unit clause {l} in Φ do
       Φ ← unit-propagate(l, Φ);
    for every literal l that occurs pure in Φ do
       Φ ← pure-literal-assign(l, Φ);
    l ← choose-literal(Φ);
    return DPLL(Φ ∧ {l}) or DPLL(Φ ∧ {not(l)});
Note we are skipping the simplification step and the invariant that we must be in CNF form
Functional ISL pseudocode:
satisfiable:
    get a list of all the variables
    call helper with the formula and its free variables
    if the formula only contains literals then we can evaluate it, if it is evaluated to #t then it is satisfiable
    for the first variable, substitute all occurances of it with #t or #f and recur with the rest of the variables
|#

; satisfiable? : BooleanFormula -> Boolean
; sat solves stuff
(define (satisfiable? bf)
  (satisfiable-helper? bf (find-free-vars bf)))

(check-expect (satisfiable?  ex-bf-4) #t)
(check-expect (satisfiable? (make-and-op 'x 'x)) #t)
(check-expect (satisfiable? ex-bf-3) #t)
(check-expect (satisfiable? (make-and-op 'x (make-not-op 'x))) #f)


```

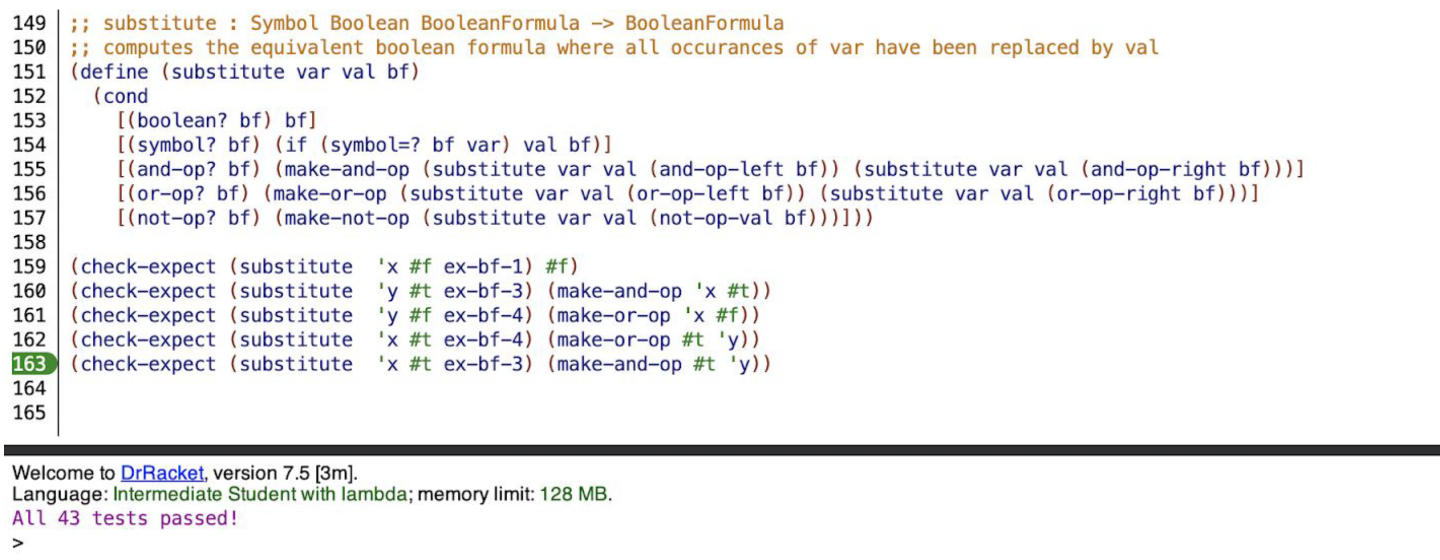
[redacted]



```
;; find-free-vars : BooleanFormula -> [List-of Symbol]
;; computes a list of all the variables that exist within the boolean formula
(define (find-free-vars bf)
  (cond
    [(boolean? bf) '()]
    [(symbol? bf) (list bf)]
    [(and-op? bf) (append (find-free-vars (and-op-left bf)) (find-free-vars (and-op-right bf)))]
    [(or-op? bf) (append (find-free-vars (or-op-left bf)) (find-free-vars (or-op-right bf)))]
    [(not-op? bf) (find-free-vars (not-op-val bf))]))

(check-expect (find-free-vars  ex-bf-1) (list 'x))
(check-expect (find-free-vars  ex-bf-2) (list 'x))
(check-expect (find-free-vars  ex-bf-3) (list 'x 'y))
(check-expect (find-free-vars  ex-bf-4) (list 'x 'y))

;; satisfiable-helper? : BooleanFormula [List-of Symbol] -> Boolean
;; computes a list of all the variables that exist within the boolean formula
;; INVARIANT: all symbols in free-vars exist within bf
(define (satisfiable-helper? bf free-vars)
  (cond
    [(consistent-literals? bf) #t]
    [(empty? free-vars) #f]
    [else (or (satisfiable-helper? (substitute (first free-vars) #t bf) (rest free-vars))
              (satisfiable-helper? (substitute (first free-vars) #f bf) (rest free-vars)))]))

(check-expect (satisfiable-helper?  ex-bf-1 empty) #f)
(check-expect (satisfiable-helper?  ex-bf-2 (list 'x)) #t)
(check-expect (satisfiable-helper?  ex-bf-3 (list 'x)) #f)
(check-expect (satisfiable-helper?  ex-bf-3 (list 'x 'y)) #t)
(check-expect (satisfiable-helper?  ex-bf-4 (list 'x)) #f)
(check-expect (satisfiable-helper?  ex-bf-4 (list 'x 'y)) #t)


;; consistent-literals? : BooleanFormula -> Boolean
;; does the boolean formula contain no variables and evaluates to true?
(define (consistent-literals? bf)
  (and (only-literals? bf) (consistent? bf)))

(check-expect (consistent-literals?  ex-bf-1) #f)
(check-expect (consistent-literals?  (make-and-op #t #t)) #t)
(check-expect (consistent-literals?  (make-or-op #t #f)) #t)
(check-expect (consistent-literals?  (make-not-op #f)) #t)
(check-expect (consistent-literals?  ex-bf-2) #f)
(check-expect (consistent-literals?  ex-bf-3) #f)
(check-expect (consistent-literals?  ex-bf-4) #f)
```



```
;; does the boolean formula contain no variables?
(define (only-literals? bf)
  (cond
    [(boolean? bf) #t]
    [(symbol? bf) #f]
    [(and-op? bf) (and (only-literals? (and-op-left bf)) (only-literals? (and-op-right bf)))]
    [(or-op? bf) (and (only-literals? (or-op-left bf)) (only-literals? (or-op-right bf)))]
    [(not-op? bf) (only-literals? (not-op-val bf))]))

(check-expect (only-literals?  ex-bf-1) #f)
(check-expect (only-literals?  (make-and-op #t #t)) #t)
(check-expect (only-literals?  (make-or-op #t #f)) #t)
(check-expect (only-literals?  (make-not-op #t)) #t)
(check-expect (only-literals?  ex-bf-2) #f)
(check-expect (only-literals?  ex-bf-3) #f)
(check-expect (only-literals?  ex-bf-4) #f)

;; consistent? : BooleanFormula -> Boolean
;; does the boolean formula evaluate to true?
;; INVARIANT: bf must not contain any variables
(define (consistent? bf)
  (cond
    [(boolean? bf) bf]
    [(symbol? bf) (error "contains a non-literal")]
    [(and-op? bf) (and (consistent? (and-op-left bf)) (consistent? (and-op-right bf)))]
    [(or-op? bf) (or (consistent? (or-op-left bf)) (consistent? (or-op-right bf)))]
    [(not-op? bf) (not (consistent? (not-op-val bf)))]))

(check-error (consistent?  ex-bf-1) "contains a non-literal")
(check-error (consistent?  ex-bf-2) "contains a non-literal")
(check-error (consistent?  ex-bf-3) "contains a non-literal")
(check-error (consistent?  ex-bf-4) "contains a non-literal")
(check-expect (consistent?  (make-and-op #t #t)) #t)
(check-expect (consistent?  (make-and-op #f #t)) #f)
(check-expect (consistent?  (make-or-op #t #f)) #t)
(check-expect (consistent?  (make-or-op #f #f)) #f)
(check-expect (consistent?  (make-not-op #t)) #f)
(check-expect (consistent?  (make-not-op #f)) #t)


```



```
;; substitute : Symbol Boolean BooleanFormula -> BooleanFormula
;; computes the equivalent boolean formula where all occurances of var have been replaced by val
(define (substitute var val bf)
  (cond
    [(boolean? bf) bf]
    [(symbol? bf) (if (symbol=? bf var) val bf)]
    [(and-op? bf) (make-and-op (substitute var val (and-op-left bf)) (substitute var val (and-op-right bf)))]
    [(or-op? bf) (make-or-op (substitute var val (or-op-left bf)) (substitute var val (or-op-right bf)))]
    [(not-op? bf) (make-not-op (substitute var val (not-op-val bf)))]))

(check-expect (substitute  'x #f ex-bf-1) #f)
(check-expect (substitute  'y #t ex-bf-3) (make-and-op 'x #t))
(check-expect (substitute  'y #f ex-bf-4) (make-or-op 'x #f))
(check-expect (substitute  'x #t ex-bf-4) (make-or-op #t 'y))
(check-expect (substitute  'x #t ex-bf-3) (make-and-op #t 'y))
```

```
Welcome to DrRacket, version 7.5 [3m].
Language: Intermediate Student with lambda; memory limit: 128 MB.
All 43 tests passed!
>
```