Overview of the System

History of System

CalcCheck is based on the 1993 book, A Logical Approach to Discrete Math (“LADM”) written by Gries and Schneider. They book covers the fundamental base logic behind discrete mathematics and its syntax. In 1995 revisions were added to include precise logic foundations for propositional logic and again in 1997 for predicate logic. In LADM we are given proper discrete mathematical syntax that we find directly implemented into Calccheck.

CalcCheck itself was written, maintained and updated by Dr.Wolfram Kahl a professor at McMaster University in 2011. It was originally written with hard coded algorithms however when it was remade in 2016 it gained many features including the ability to reference axioms. He found there was a need for an effective universal proof checker to help students who were learning - and working with - proofs.

Reason for Selection

The reason we chose CalcCheck for our project is due to its prevalence in our course as well as having access to the creator of the application here at McMaster. In addition to that we found Calccheck to be a challenging tool and wanted to learn more about it and help others in our class learn more as well.

Usage and Relevance to Class

Discrete Mathematic (2DM3) is the prerequisite of software specification and correctness course. It provides fundamental knowledge to help student who learns logic and proof based problem. Calcheck tool was firstly introduced in 2DM3 by professor Wolfram Kahl, and it was used most of the time in the class. In this section, the usage and relevance of Calcheck to the 3EA3 course will be firstly explained, then some examples will be provided.

Calcheck is highly relevance to the class and its being used extensively. Problems that the student tries to proof in class are taking form of Calcheck’s syntax. For example, in 3EA3, students are often asked to write proofs for proposition and induction problems. If students want to write a proof in a formal manner, they can follow Calcheck’s syntax to achieve it, such that, students must provide a theorem first and then based on the theorem, they can manipulate the equation, and eventually to satisfy the opposite side of the equation.In the following example, student can input these steps into Calcheck and justify their calculation of proof.

Proposition example:

This proof demonstrates the use of the Golden Rule to prove a new theorem involving conjunction. First, we apply the Golden Rule to the expression that we started with (X ∧ X). We then achieve our next step, resulting in: *X* ≡ *X*≡ *X* ∨ *X. Using the Idempotency of* ∨ and the Identity of Equivalence, we arrive at our final answer 'X'.

Given theorem. *P* ∧ *Q* ≡ *P* ≡ *Q* ≡ *P* ∨ *Q* (Golden Rule)
Proof: For any X we observe
The conjunction is idempotent: *X* ∧ *X* ≡ *X*

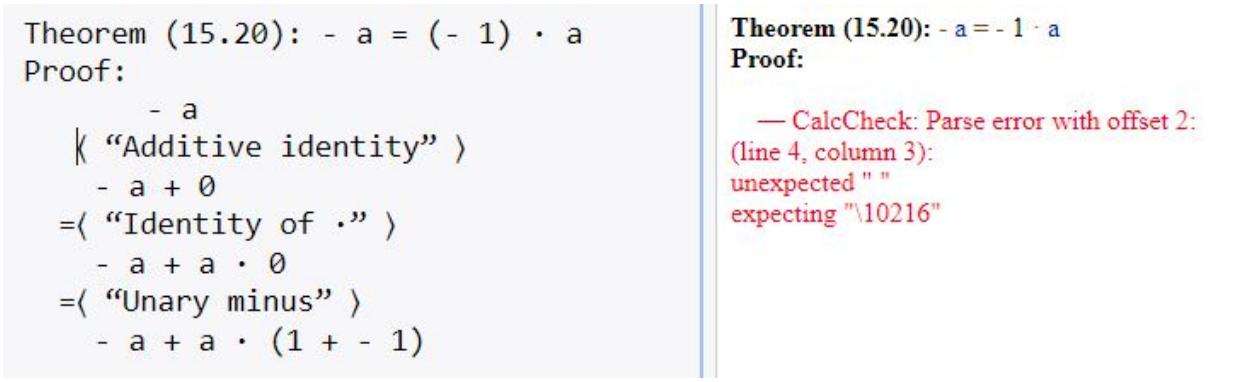
X∧*X*
≡ <Golden Rule, parsed *(P* ∧ *Q* ≡ *P* ≡ *Q* ≡ *P* ∨ *Q)*>
X ≡ *X*≡ *X* ∨ *X*
≡ <∨ is idempotent>
X≡*X*≡*X*
≡ <*P* ≡ *P*>
X

Induction example:

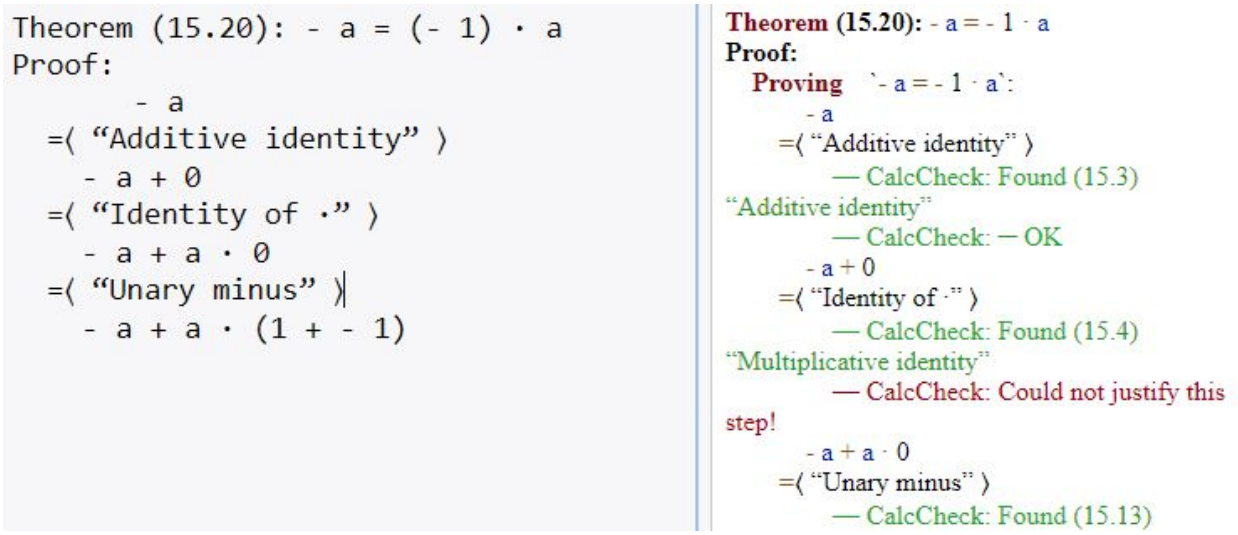
The below proof outlines how we can do induction on natural numbers in CalcCheck. First, we choose with variable to complete our induction proof on - here we only have the m variable to choose from, so we start with that. First we outline our base case, which is trivial since we only need to apply one theorem to complete it. Next, we start our induction step. We show that we are proving this theorem for all m + 1 variables by using S m. Using our induction hypothesis and the definition of +, we arrive at our answer, proving this theorem to be true.

Theorem "Right-identity of +": m + 0 = m
Proof:
By induction on `m : ℕ`:
Base case:
0 + 0
≡⟨ "Definition of + for 0" ⟩
0
Induction step:
S m + 0
≡⟨ "Definition of + for `S`" ⟩
S (m + 0)
≡⟨ Induction hypothesis ⟩
S m

Calcheck does not aid proof findings, rather than proof checking. The tool offers immediate feedback for students to either confirm that they are on track or inform them that they have failed in certain steps.The following examples contain some error feedback from Calcheck tool when the students try to verify their proof.



Error: Syntax error. The location with the cursor is missing an equals sign



Error: The theorem here is used incorrectly and should not be used in this context

Obtaining, Installing, and Learning to use the System

Accessibility and Learnability

Manaar:

CalcCheck is accessible through its IP address. There is currently no real hostname for a URL. The IP address is http://130.113.68.214:93xx/, where the xx is a substitute for whichever page of the application is being accessed. For example, 9301 is used for the most basic proofs, equational theory of integers, while 9323 is used for something a little more complicated – disjunctions. Due to the fact that is a website and not a downloadable application, it is easily and quickly accessed online without taking up significant space on the machine it is being used. However, the downside to this is that it does not run without Wifi, so users are limited to using the application only while they can be online. In a world where free Wi-Fi is in abundance, this should not be an issue, especially not for students who have free Wi-Fi on campus.

For McMaster students enrolled in CS 3EA3, there are links to CalcCheck on the course resource page, specifying the type of exercise at the end of each link. However, it is otherwise not that easy to figure out how to get to CalcCheck. Googling it leads to documentation on how to get comfortable with the syntax and layout. While that is also very important information, there is nothing jumping out at the user on how to get to the website. However, since CalcCheck was developed as “A Proof-Checker for Gries and Schneider’s “Logical Approach to Discrete Math””, and LADM is used as a textbook for courses, it is very likely that the link will be discoverable to the students in those courses, as instructors should use it as a practical supplement to plausible textbook theories. The hands-on experience is more likely to help students retain the information in their long-term memory

The software is very simple to learn how to use. Not only is there a lot of helpful documentation online to guide new users on how the system works, the site itself also gives some tips. Googling “CalcCheck McMaster” leads to this page: http://calccheck.mcmaster.ca/CalcCheckDoc/ which has many helpful links for the new user to take advantage of. The first one explains how to use the CalcCheck language itself. The second one shows how to use that language on CalcCheck Web. There are many special symbols used in the proofs and the link helps to clarify the keyboard shortcuts for them, as well as how to generally use the site (ex., control + enter = syntax check but control + option + enter = answer check). The third link talks about substitution, how to use it, and provides examples. The fourth link has a list of theorems that are to be used in the proof. These four links together have everything a new user needs to get started with CalcCheck. As with everything else, more practice will lead to more efficiency, the user will be more familiar with axioms and using the right syntax, and generally being better with using the CalcCheck language in CalcCheck Web.

Advantages + Disadvantages

Manaar:

Advantages	Disadvantages
Quick, because web based. Doesn't require any downloads, which could have caused permissions issues	No memory, because web based. Refreshing the pages loses info
Supports writing proofs, Validates and verifies proof	Doesn't support free types or definitions, but similar language "Zed" does
Proof checker	Not proof assist currently, but consider future extension for added functionality
Stand alone as language. Self-defined to match textbook,	Possibly difficult to grasp for those not familiar with textbook
Avoid syntax mistakes by following rules from guide. Or copy, pasting and modifying correct lines.	Specific syntax, such as space after negation. Syntax checker doesn't clarify error. Just says that one exists somewhere.

Algorithm Implementation with Specification and Correctness Algorithm (blue ones to be demoed during presentation)

This example is a breakdown of the Self Inverse of Unary Minus Theorem. In cases such as this one, the goal is to select one side of the equation and apply theorems to it until it matches the other side of the equation. Each step is broken down with further details in the chart below:

Theorem (15.17) “Self-inverse of unary minus”: - (- a) = a Proof: - (- a) =⟨ “Identity of +” ⟩ 0 + - (- a) =⟨ “Unary minus” ⟩ a + (- a) + - (- a) =⟨ “Unary minus” ⟩ a + 0 =⟨ “Identity of +” ⟩ a	Choose the left side as the start state The identity of + theorem says that 0 + a = a Apply the theorem, treating -(-a) as the a in the theorem. So essentially add 0 to it, since that’s equal to just -(-a). The unary minus theorem says that the sum of a and -a is 0 So substitute a + (-a) for 0 Now use the same theorem in reverse Substitute 0 in for (-a) + -(-a), the positive and negative of (-a) With just a+0 left, recall the theorem that says adding 0 to something keeps the result at that something, ergo adding 0 makes no difference So remove the 0

The example above started with the left side of the equation and used variations of 2 theorems to to transform it into the right side of the equation. This is essentially how proofs work in CalcCheck.

The next example is a little more complicated, but uses the same fundamental concepts of choosing applicable theorems to transform the selected side of the equation into the other side. Note that in this example, the theorem is selected by number instead of by name

Theorem (15.20): - a = (- 1) · a Proof: (- a) =⟨ (15.3) ⟩ - a + 0 =⟨ (15.9) ⟩ - a + 0 · a =⟨ (15.13) ⟩ - a + (1 + - 1) · a =⟨ (15.5) ⟩ (- a + 1 · a) + (- 1) · a =⟨ (15.4) ⟩ (- a + a) + (- 1) · a =⟨ (15.13) ⟩ 0 + (- 1) · a	Choose the left side of the equation Select the theorem to be applied in the next step (-a) = -a + 0, so write the right side instead of left Select the theorem to be applied in the next step 0 = 0 * a, so substitute the right side instead of the left Select the theorem to be applied in the next step 0 = 1 + -1, so substitute the right side instead of the left Select the theorem to be applied in the next step Multiply the a with the 1 and the -1, and rewrite w/o grouping Select the theorem to be applied in the next step Rewrite the 1*a as just a, since 1*a = a in the theorem Select the theorem to be applied in the next step -a + a = 0, so substitute the right side instead of the left side

=⟨ (15.3) ⟩ (- 1) · a	Select the theorem to be applied in the next step 0 + (- 1) · a = (- 1) · a, since adding the 0 makes no difference, so write the right side instead of the left

Once again, the proof began by picking the left side of the equality and transforming it to match the right side. The example below is more complicated than the last two and introduces some theorems the last two examples didn't need.

Theorem (15.27): (a - b) · (c - d) = (a · c + b · d) - (a · d + b · c) Proof:	
(a - b) · (c - d)	Start with the left side of the equality
=⟨ (15.14) ⟩	Select a theorem to be applied
(a + - b) · (c + - d)	Rewrite a-a in the format a+-a because the sum is easier to split than the subtraction.
=⟨ (15.5) ⟩	Select a theorem to be applied
(a + - b) · c + (a + - b) · - d	Treat a+-b as one term, multiply with c and add to it being multiplied by -d
=⟨ (15.5) ⟩	Select a theorem to be applied
a · c + - b · c + - d · a + - d · - b	Expand it out, write all multiplications explicitly
=⟨ (15.23) ⟩	Select a theorem to be applied
a · c + - b · c + - d · a + d · b	When two negatives are being multiplied, the negatives get cancelled out and they're both left positive
=⟨ (15.2) ⟩	Select a theorem to be applied
(a · c + d · b) + - b · c + - d · a	Regroup, the first part of the goal now exists in the brackets
=⟨ (15.20) ⟩	Select a theorem to be applied
(a · c + d · b) + - 1 · (b · c) + - 1 · (d · a)	Rewrite the second part of the equation by separating the negative from the value as -1 and then grouping the new values together
=⟨ (15.20) ⟩	Select a theorem to be applied
(a · c + d · b) + - (b · c) + - (d · a)	Rewrite the second part of the equation by making each term negative instead of writing it as multiplied by -1
=⟨ (15.19) ⟩	Select a theorem to be applied
(a · c + d · b) + - ((b · c) + (d · a))	Group the second half together and take the negative outside the brackets

=⟨ (15.14) ⟩ (a · c + d · b) - ((b · c) + (d · a))	Select a theorem to be applied Rewrite the +-(second half) as just -(second half)

The examples up until this point have been for proofs, where the main takeaway is to pick one side of the equality and work over to the other side. The next couple of examples, however are for calculations, not proofs, so they work a little differently. The goal is no longer to PROVE the equality, but to FIND the equality, a simplification of a given expression.

Calculation: (2 · x + 3 · y)[y ≔ x + 6][x ≔ y + 7] =⟨ Substitution ⟩ 2 · (y + 7) + 3 · ((y + 7) + 6) =⟨ (15.1a) ⟩ 2 · (y + 7) + 3 · (y + (7 + 6)) =⟨ Evaluation ⟩ 2 · (y + 7) + 3 · (y + 13) =⟨ (15.5) ⟩ 2 · y + 2 · 7 + 3 · y + 3 · 13 =⟨ (15.5) ⟩ (2 + 3) · y + 2 · 7 + 3 · 13 =⟨ Evaluation ⟩ 5 · y + 53	Substitute every y with x+6, and then substitute every x with y+7 Change the associativity of y+7+6 so that 7+6 is grouped together and they can be added Perform any evaluations, so now y+7+6 can be simplified to y+13 Expand the expression so that each multiplication is written explicitly, no groups Group the coefficients together and rewrite the expression Carry out all evaluations and get a very simplified statement at the end

The above example started out with an expression and two substitutions. The final result a simplified result of that expression after the substitutions. The below example does the same. The difference is the order in which the substitutions take effect.

Calculation: (2 · x + 3 · y)[x ≔ y + 7][y ≔ x + 6] =⟨ Substitution ⟩ 2 · ((x + 6) + 7) + 3 · (x + 6) =⟨ (15.1a) ⟩ 2 · (x + (6 + 7)) + 3 · (x + 6) =⟨ Evaluation ⟩ 2 · (x + 13) + 3 · (x + 6)	Substitute every x with y+7, and then substitute every y with x+6 Change the associativity of x+6+7 so that 6+7 is grouped together and they can be added Perform any evaluations, so now x+6+7 can be simplified to x+13

=⟨ (15.5) ⟩ 2 · x + 2 · 13 + 3 · x + 3 · 6 =⟨ (15.5) ⟩ (2 + 3) · x + 2 · 13 + 3 · 6 =⟨ Evaluation ⟩ 5 · x + 44	Expand the expression so that each multiplication is written explicitly, no groups Group the coefficients together and rewrite the expression Carry out all evaluations and get a very simplified statement at the end

For more challenging examples refer to the **Usage and Relevance to Class** section of this document.

Use for Program Specifications and Correctness

Comparison to Z

CalcCheck was created by Dr.Kahl for the sole purpose of checking mathematical proofs based off of the format used in the textbook: A Logical Approach to Discrete Math or LADM. When we met with Dr.Kahl, he referenced another language that is similar to CalcCheck, but can be used for other purposes that the scope of CalcCheck does not yet cover, called Z. Z is also used for mathematical specifications, but it differs from CalcCheck in a few important ways. First, Z allows definitions, so you can easily define free-types for your own mathematical problems, which is a feature that has not yet been implemented in CalcCheck. Also, there is no tool for Zed that can be used to check the correctness of your proofs, and the language only exists for the purpose of modelling proofs so far. However, there is an extension of Z that contains a proof checker called Proof Power which can be used for a similar purpose as CalcCheck. Proof Power is based on the logic of Z, similarly as CalcCheck is based on the logic E defined in the LADM textbook. Overall, Z is pretty similar to CalcCheck in a general perspective, as it is simply a mathematical language, used richly for specifications and communication of proofs.

Underlying/Soundness of Logic

The overall purpose of CalcCheck was to be a proof checker for mathematical proofs based on the LADM style of proofs found in the textbook that us students used for Compsci 2DM3. The logic of the proofs are based around a large set of axioms found in the textbook, and CalcCheck guides us to apply these axioms to prove a set of theorems to further student's skills in writing mathematical proofs. When examining the soundness of CalcCheck, we first have to examine

the underlying logic of this tool. CalcCheck was based off of a logic called E, and we use this logic in CalcCheck to write and verify our proofs. In the LADM textbook, they refer to this equational logic as a form of propositional calculus. This logic E contains two parts, one part is a set of axioms, and another part consists of three inference rules that we know: Leibniz, Transitivity, and Substitution. The logic E also contains theorems, which can be defined as: an axiom, or a theorem that has been proved by using the inference rules as their premises, or a boolean expression that can be proved equal to a previously proved theorem or an axiom. We know that all the axioms presented in the LADM textbook are true, as they can be proved to be correct via truth tables or arguing that for each inference rule, if their premises are true then so are they. When examining a logic, we know it is sound when anything that can be proved using pre-defined axioms in the logic is valid. An easy way to think of it is: Doing true things always leads to true results. Since in CalcCheck we are proving theorems, and theorems in E are simply things that we prove using inference rules, axioms or previously proved theorems we know that the premises of the theorems are valid, and this is the first step to verifying soundness of a logic. We know that the theorems that we can prove using logic E are true because we only use true rules to verify them. Therefore, we know that the underlying logic E used by LADM and by CalcCheck is sound.

Due to the fact CalcCheck is based off LADM it is important to understand the underlying logic to Discrete Math itself. Discrete Math is the study of mathematical structures that are fundamentally discrete rather than continuous. In our case we focus on first order logic, also know as predicate logic, which uses quantified variables over non-logical objects as well as the use of sentence which contain variables. This differs from propositional logic which does not use quantifiers or relations.

Future Improvements

CalcCheck is a relatively new tool developed by Dr.Kahl that is constantly under development, as Dr.Kahl and the rest of his team find ways that they can expand and improve the scope and abilities of this tool. When we met with Dr.Kahl, we discussed the different future expansions that he would consider making for this tool. The first thing we asked him about was if he would consider expanding the error messages in checking proofs to be more specific. For example, if you tried to apply many axioms in one step, it would send an error message telling you specifically which axiom could not be applied and why. Dr.Kahl told us that this tool was created to be a proof **checker**, not a proof **assistant**, so implementing a feature like this is not a priority for him at this time. However, he did say he would consider implementing this as an **extension** of CalcCheck in the future, to help students learn in a more interactive way about writing mathematical proofs. He also added that he would like to expand CalcCheck to accept free-type definitions, since this would be a substantial addition to the already helpful features CalcCheck uses, and would expand the type of scenarios that we can model using this language. We also

inquired about how he came up with the syntax of the proofs, and the language that he developed. This language was based off of the proof models shown in the LADM textbook, which students use to learn about developing proofs in the second year Computer Science course, 2DM3. He also added that the language is completely independent of the tool - and this syntax only exists to make it easier for students to follow how the proofs are built in the textbook.

Quiz questions

Questions:

1. THEOREM | LEMMA | COROLLARY | PROPOSITION | FACT.
   What's the difference?

2. What axiom would this fun fact about calc check prove?
   (“Absorption”) == (`3.44`)
   (“Absorption”) == (`3.44`) == (`3.44a`)
   (“Absorption”) == (`3.44`) == (`3.44b`)
   (`3.44a`) != (`3.44b`)

Answers:

1. Theorem, Lemma, Corollary, Proposition and Fact all mean THE SAME THING, in fact these words can be used interchangeably. Axiom is almost the same with the exception that it cannot be proven.
2. This is an adaptation of non-transitivity!