Some Suggestions for Final Projects Computability and Logic

Logic-Based Projects

Equivalence Rules

- Program an interface that allows the user to rewrite statements using a given set of equivalence rules
- Program an algorithm that given any two equivalent statements, transforms the one statement into the other using a given set of equivalence rules
- Try to reduce the set of equivalence rules as provided on HW 1 even further, while still remaining complete (prove this!). If some set can not be reduced any further, prove that that is so (i.e. prove the independence of each of the equivalence rules).

Short Truth-Tables

- Program an interface that allows the user to fill out a short truth-table. In this, you may want to take a look how Boole lets the user fill out a full truth-table.

Truth Trees

- Program an interface that allows the user to construct a truth tree. Possible additional features:
    - Different levels of computer assistance (ranging from no feedback at all to the computer automatically generating a tree)
    - Added rules to speed things up. E.g:
        - Equivalence rules
        - Quick inference rules (Modus Ponens, Modus Tollens, Disj. Syll, etc. Careful: result of rule must be what normal decomposition process obtains. E.g. Using $\wedge$ Elim to just get P from P $\wedge$ Q is not good, since you also need Q)
        - Splitting rule (i.e. at any time you can branch on φ and $\neg\varphi$)
        - Close branch with any φ and $\neg\varphi$ (i.e. not just atomic)

Natural Deduction

- Program an interface for doing Natural Deduction Proofs, but better than Fitch:
    - Added rules. E.g:
        - Quick inference rules (Modus Ponens, Modus Tollens, Disj. Syll, Excluded Middle (i.e. get $\varphi \vee \neg\varphi$), etc.
        - Equivalence rules
    - Directly accessible Tablet of Rules rather than annoying pull-down menu
    - Re-use of earlier established results (e.g. once you have proven DeMorgan's from more basic rules, you can then use DeMorgan's as 1 step in future proofs)
    - More visual representation of proofs (e.g. more like Slate)

Sequent Systems

- Program interface for Sequent Systems (? Slate is hard to beat!)

Existential Graphs

- Program interface for Existential Graphs. Possible features:
    - Load/save proofs
    - Video-like editing
    - Use of subproofs/sequents
    - Working backwards
    - Hook up with Kinect

Axiom Systems

- Program interface for Axiom System
- Compare and contrast (and prove equivalency) of different axiom systems
- Develop your own axiom system
- Program automated axiom independence checker
- Program axiom independence checker interface

Proof Systems: All

- Program a proof converter (takes in proof of one system and simulates it in a different system)
    - Truth-Tree-to-Natural-Deduction
    - Natural-Deduction-to-Sequent-System
    - Sequent-System-to-Existential-Graphs
    - Or any other conversion
- Compare efficiency different proof systems
- Develop your own proof system
    - Prove soundness and completeness
    - Hybrid of different styles
        - Derived Sequent System proposal: (how) can we get the 'best of different systems'?
        - User-friendly axiom system with multiple rules (seen differently: natural deduction system without subproofs)
- Program an Automated Theorem Prover or Automated Proof Generator
    - If it displays its work/process/result in a user-friendly manner, even better!

Puzzle Logics (for logic puzzles)

- Solve puzzles using formal logic
- Develop formal systems ('puzzle logics') that are more efficient in solving logic puzzles
    - Extra: Prove completeness of your puzzle logic

Axiomatization of location in Tarski's World

- Axiomatize location in Tarski's World (and prove that your axioms are complete), or prove that it cannot be done. Do this for 8x8 finite worlds as well as infinite worlds.

Non-Standard Interpretations of Number Theory

- Explore non-standard interpretations of axioms of number theory. Keep adding elementary number-theoretic truths to the axioms and see if you can still find non-standard models. If you add the induction scheme, can you still find a non-standard model?

Formal Proofs of Number Theory

- Find formal proofs of theorems (together with helpful lemma's) in number theory (I was able to formally prove Euclid's Theorem: that there is no greatest prime number; ask me for Fitch files)

Formal Proofs of Set Theory

- Find formal proofs of theorems (together with helpful lemma's) in set theory

Formalize parts of Metalogic

- Either come up with a new logic in which we can write things like $\Gamma \vDash \psi$ as part of that formal language and come up with inference rules for that, or define predicates in FOL (e.g. Implies(x,y)) and define axioms involving those predicates, so that metalogical proofs (or at least parts thereof) can be formalized.

Computation-Based Projects

Improve Turing Machines software. For example:

- Zooming in/out and moving around the transition graph (like AM software)
- Defining and running multiple inputs at once (see JFLAP)
- More practical file format (XML?)
- Move head to different location / make changes directly to tape

Improve Abacus Machines software. For example:

- All windows in one
- All menu's in 1 row (maybe some more helpful buttons of common tasks (e.g. simulate))
- Registers start at 1, not 0
- Delete icon
- Simpler/more intuitive 'Select' icon (cursor arrow; see TM software)
- When expanding registers window, it should show more registers, not stretch registers
- Define multiple inputs and all run at once (like new TM)

- More user-friendly speed (step, fast, like TM)

Universal Turing Machine

Design a Universal Turing machine that uses a binary alphabet.

Other Universal Machines

Research other kinds of Universal machines. There is an interesting 'competition' for finding the smallest Universal machine that may be interesting to look at.

Abacus-Machines-to-Turing-Machine Converter

Write a program that takes in a description of an Abacus-machine, and outputs a description of a Turing-machines that emulates that Abacus Machine using the convention as described in class. Work with the file formats of our Abacus-machine and/or Turing-machine software programs.

Recursive-Function-to-Abacus-Machine Converter

Write a program that takes in a description of a recursive function, and outputs a description of a Abacus-machines that computes that function using our convention as described in class. If your code can work with the file format of our Abacus-machine, even better!

Recursive Function Software

Program an interface for creating recursive functions. Define processes of Composition, Recursion, and Minimization, but also define recursive relations, and implement several additional techniques techniques to create recursive functions and relations from other recursive functions and relations, such as Definition by Cases, Bounded Mini-/Maximization, Graph, Logical Operators, and Substitution

Other definitions of Computation

Research some form of computation we haven't looked at. Of course, a proof of computational equivalence would be great!

Automated Code Generator

Write a program that, given some specification of some function, generates code that computes that function (and that's actually potentially useable, i.e. no Turing machine code).

Automated Code Certifier

Write a program that takes some other program and function, and certifies (i.e. proves, or at least checks) whether that program computes that function.