This draft was last updated Wed Oct 12 21:46:17 CDT 2016.

DESIGN AND IMPLEMENTATION OF SOME COMPUTATINAL TOOLS FOR MATHEMATICAL LOGIC

ABSTRACT. We design and implement tools to parse formulas, enumerate models of axioms, refute some conjectures, and assist in enumerating a theory.

1. INTRODUCTION

Objects of interest in mathematical logic include axioms and models. Some examples of axioms and some of their models are represented in listings 1 and 2. Operations can be performed on these objects to derive other relevant strings of symbols and models. The goals of this project are to identify some of these operations which can be carried out by a computer, to implement these operations and combine them as tools, and to use these tools to perform experiments.

Definitions will be taken from [1] (henceforth referred to as "Ebbinghaus").

2. PARSING FORMULAS FROM MATHEMATICAL LOGIC

2.1. **Definitions.** Let A be the set of logic symbols $\{\neg, \wedge, \vee, \rightarrow, \forall, \exists,), (, v_1, v_2, v_3, ...\}$ and let S be a set of relation, function, and constant symbols. Our working examples are $S_{\text{eq}} := \{\text{R}\}$ for equivalence relations and $S_{\text{gr}} := \{\circ, \text{e}\}$ for groups[1]. Fix symbol set S and consider the alphabet $A_S := A \cup S$. Define A_S^* as the set of all strings of symbols from this alphabet. Our interest is in a specific subset of strings called formulas, denoted $L^S \subset A_S^*$. Towards defining L^S, first define the set $T^S \subset A_S^*$ of terms using the calculus[2] given schematically[3]:

$$\begin{aligned} t := \quad & \texttt{c} && (T1) \\ | \quad & \texttt{v} && (T2) \\ | \quad & \texttt{f}\ t_1\ t_2\ ...\ t_n && (T3) \end{aligned}$$

This schematic is read: "a term t is either a constant symbol $\texttt{c}$, a variable symbol $\texttt{v}$, or a n-ary function symbol $\texttt{f}$ followed by n terms t_1, t_2, ..., t_n. The subscripts after terms are only for indexing and for indicating that the terms need not be equal. An example term from group theory is $\circ\ \texttt{e} \circ \texttt{v}_1 \texttt{e}$, and is derived by first applying rule (T1), (T2), and (T3) to obtain $\circ \texttt{v}_1 e$, then applying (T1) and (T3) to obtain the full string.

[1] Alteratively, groups can have symbol set $S_{\text{grp}} = \{\circ, \text{e}, ^{-1}\}$, allowing universal axioms whose advantages will be mentioned later.

[2] Also known as a "syntax" or "grammer", a calculus is a set of rules for deriving new strings from existing strings.

[3] This schematic, resembling Backus-Naur form, will inform our later design of a parser. This schematic is equivalent to Ebbinghaus' schematic: the set of rules $\{\frac{}{c}, \frac{}{v}, \frac{t_1, t_2, ..., t_n}{f t_1 t_2 ... t_n}\}$.

We can now define the set L^S of formulas ϕ analogously:

$$\begin{aligned}
\phi := \quad & t_1 = t_2 && (F1)\\
| \quad & \mathtt{R}\ t_1\ t_2\ ...\ t_n && (F2)\\
| \quad & \neg\ \phi && (F3)\\
| \quad & \phi_1\ \wedge\ \phi_2 && (F4)\\
| \quad & \phi_1\ \vee\ \phi_2 && (F5)\\
| \quad & \phi_1\ \rightarrow\ \phi_2 && (F6)\\
| \quad & \phi_1\ \leftrightarrow\ \phi_2 && (F7)\\
| \quad & \forall\ \mathrm{x}\ \phi && (F8)\\
| \quad & \exists\ \mathrm{x}\ \phi && (F9)
\end{aligned}$$

For example, the equivalence relation formula $\forall$`xRxx` employs rules $(F2)$, $(T2)$, and $(T2)$ to obtain `Rxx`, followed by rule $(F8)$.

2.2. **Parsing.** We design a tool to parse any given formulas. The input is a text file with the symbol set S followed by formulas, for example see code listings 1 and 2. With each function and relation symbol, an arity is declared. The following notation is allowed:

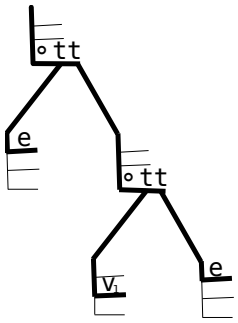
- Unicode symbols are allowed.
- Symbols for variables can also be declared.
- Symbols may not be initial substrings of each other, otherwise there may be ambiguity when parsing.
- Binary function and relation symbols can be written between terms by declaring them as `infix`.
- Parentheses can sometimes be omitted. Precedence is highest for $\forall, \exists, \neg$, then $\wedge, \vee, \rightarrow, \leftrightarrow$, which are left-associative. Function symbols in infix notation are left-associative and of equal precedence.

Each formula is parsed into a syntax tree which corresponds to the derivation of the formula, each interior node corresponds to the application of a calculus rule, and children correspond to earlier derived strings used in the calculus rule corresponding to the parent.

The parsing algorithm reads through the string from left to right. The following algorithm is described for parsing a term[4]. Read the first symbol and choose the calculus rule which starts with that symbol. If it is a constant or a variable, the term is parsed. If a function symbol is encountered, read arity number of terms the same way. An example of the tree from the earlier example $\circ e \circ v_1 e$ is illustrated in figure 1.

[4]The calculus for formulas do not allow parsing in this way, since multiple rules start with a formula. This ambiguity can be overcome by using prefix connectives $\wedge, \vee, \rightarrow, \leftrightarrow$. But to keep infix connectives, we transformed the rules to equivalent ones which eliminate ambiguity.

FIGURE 1. The derivation tree of term $\circ\, \mathtt{e} \circ \mathtt{v}_1\mathtt{e}$. Start at the root. Each node has three choices corresponding to above schematic for calculus rules for terms. Bold lines indicate rules used.



LISTING 1. groups.txt

```
relation =(2,infix) {a0==a1}
function +(2,infix)
constant 0
variable x,y,z

axiom    ∀x∀y∀z (x+y)+z = x+(y+z)
axiom    ∀x x+0=x
axiom    ∀x ∃y x+y=0

model 1_1
 +
  0
model 2_1
 +
  0 1
  1 0
model 3_1
 +
  0 1 2
  1 2 0
  2 0 1
model 4_1
 +
  0 1 2 3
  1 0 3 2
  2 3 0 1
  3 2 1 0
model 4_2
 +
  0 1 2 3
  1 3 0 2
  2 0 3 1
  3 2 1 0
model 4_3
 +
  0 1 2 3
  1 0 3 2
  2 3 1 0
  3 2 0 1
model 4_4
 +
  0 1 2 3
```

LISTING 2. equivRelations.txt

```
relation   R(2,infix)
variable   x,y,z

axiom      ∀x xRx
axiom      ∀x∀y xRy → yRx
axiom      ∀x∀y∀z ((xRy ∧ yRz) → xRz)

model 1_1
 R
  1
model 2_1
 R
  1 0
  0 1
model 2_2
 R
  1 1
  1 1
model 3_1
 R
  1 0 0
  0 1 0
  0 0 1
model 3_2
 R
  1 1 0
  1 1 0
  0 0 1
model 3_3
 R
  1 0 1
  0 1 0
  1 0 1
model 3_4
 R
  1 0 0
  0 1 1
  0 1 1
model 3_5
 R
  1 1 1
```

3. The Satisfaction Relation and Enumerating Models

3.1. **Definitions.** Fix a symbol set S. Define a size n structure $\mathfrak{A} := \{\{0, 1, 2, ..., n-1\}; \mathfrak{a}\}$ where $\mathfrak{a}$ is a set of, for each function symbol f, relation symbol R, and constant symbol c in S, a corresponding function $f^{\mathfrak{A}}$, relation $R^{\mathfrak{A}}$, and constant $c^{\mathfrak{A}}$ over the underlying set $\{0, 1, ..., n-1\}$. Define a sentence as a formula with no occurances of free[5] variables.

Consider a term t in a structure $\mathfrak{A}$. The interpretation $t^{\mathfrak{A}}$ of t in $\mathfrak{A}$ is, for each case of t:

$$\begin{aligned} t = c: &\qquad c^{\mathfrak{A}} \in \mathfrak{a} \\ t = ft_1...t_n: &\qquad f^{\mathfrak{A}}(t_1^{\mathfrak{A}}, ..., t_n^{\mathfrak{A}}) \end{aligned}$$

We will not consider the interpretation of a variable symbol since we only consider sentences which have quantified variables, which will be handled next.

Consider a sentence ϕ. Define satisfaction relation $\mathfrak{A} \models \phi$ for each case of ϕ as

$\phi = Rt_1...t_n$: $(t_1^{\mathfrak{A}}, ..., t_n^{\mathfrak{A}}) \in R^{\mathfrak{A}}$

$\phi = t_1 \equiv t_2$: $t_1^{\mathfrak{A}} = t_2^{\mathfrak{A}}$

$\phi = \neg\psi$: not $\mathfrak{A} \models \psi$

$\phi = \psi \wedge \xi$: $\mathfrak{A} \models \psi$ and $\mathfrak{A} \models \xi$

$\phi = \psi \vee \xi$: $\mathfrak{A} \models \psi$ or $\mathfrak{A} \models \xi$

$\phi = \psi \rightarrow \xi$: $\mathfrak{A} \models \psi$ implies $\mathfrak{A} \models \xi$

$\phi = \psi \leftrightarrow \xi$: $\mathfrak{A} \models \psi$ iff $\mathfrak{A} \models \xi$

$\phi = \forall x\psi$: for all $a \in \{0, 1, ..., n-1\}$, $\mathfrak{A} \models \psi$ where free occurances of x are replaced by a

$\phi = \exists x\psi$: there exists $a \in \{0, 1, ..., n-1\}$ st $\mathfrak{A} \models \psi$ where free occurances of x are replaced by a

$\mathfrak{A} \models \phi$ is read "$\mathfrak{A}$ satisfies ϕ" or "$\mathfrak{A}$ is a model of ϕ". Generalizing to sets of sentences $\Phi := \{\phi_1, \phi_2, ..., \phi_m\}$, $\mathfrak{A} \models \Phi$ means $\mathfrak{A}$ models each sentence in Φ.

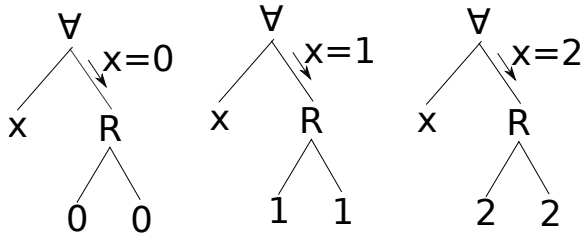
For example, listings 1 and 2 shows models of group axioms ("groups") as function tables and models of equivalence relation axioms ("equivalence relations") as relation tables.

3.2. **Evaluating Satisfaction.** Given axioms Φ and a corresponding structure $\mathfrak{A}$, to test whether $\mathfrak{A} \models \Phi$, first parse the sentences into syntax trees as described earlier. For each syntax tree, start at the root and evaluate satisfaction of each node by first evaluating satisfaction of each child, then evaluating that node by checking against the definition of satisfaction above. The tricky part is quantifiers, whose subtree is traversed once for each each possible variable value as illustrated in figure 2. Each relation node is evaluated by looking up values in its relation table. Similarly, functions are evaluated from function tables.

The above algorithm relies on operations to traverse between parents and children. Since the trees are fixed, the traversal is fixed, so it is wise to hard-code this traversal as computer code. We have a code generator which does this, handling quantifiers with loops. We call this code generator our "model enumerator generator".

[5]Free variables occurances are unquantified. For example $\forall x(Rxy \wedge \exists yRyy)$ is not a sentence since y is not bound by a quantifier in Rxy. The subformula $\exists yRyy$ is a sentence.

FIGURE 2. When checking a structure against a quantified sentence such as $\forall x R x x$, the quantifier is re-traversed for each variable value.



3.3. **Enumerating Models.** Given axioms, we enumerate all possible candidate structures, which are just tables of values, which correspond to binary strings or n-ary strings of given length. We start with candidate structures of size 1, then size 2, and so on. On each candidate structure, we evaluate satisfaction. We print the structures that satisfy the axioms.

This described procedure is generic, but it employs a naive brute-force search. The ultimate goal is to avoid the brute-force checking of candidate structures, that is, to generate models directly. Some axioms allow generatingtheir models directly without a brute-force search. For example, the axiom of reflexivity for a relation corresponds to the diagonal being all 1's. A more complicated example was done [2] for models of equivalence relations by inspecting models generated by brute-force and noticing the pattern.

If brute force is necessary, it is desireable to reduce the search space. For example, each row and column of a group function table is a permutation of the underlying set, so generate only candidate models whose rows and columns are permutations. If models are available for some of the axioms, only those models should be checked against the remaining axioms. For example, find models of commutative groups by checking whether precomputed groups satisfy the additional axiom of commutativity. Similarly, reuse already computed substructures. For example, groups can be reused when enumerating rings of fields.

In the case of axioms being universal sentences[6], a trick is to build candidate models from models of one size smaller by adding a row and column to relation or function tables. If one just wants models up to isomorphism, one can throw away isomorphic models to limit the number of smaller models to build upon, but this involves permutating rows and columns of the tables for function or relation.

4. Consequence Relation: Refuting Conjectures and Enumerating a Theory

4.1. **Definitions.** Fix symbol set S. Consider set Φ of sentences representing axioms, and another sentence ϕ representing a conjecture. $\Phi \models \phi$, read "ϕ is a consequence of Φ", means that all models of Φ also model ϕ. Note that if we find one counterexample model $\mathfrak{A}$ such that $\mathfrak{A} \models \Phi$ but $\mathfrak{A} \not\models \phi$, then $\Phi \not\models \phi$.

For example, for group theory, $\Phi_{gr} \not\models \forall x \forall y (x \circ y \to y \circ x)$ since there is a model of a noncommutative group, the smallest one has size six. Likewise, $\Phi_{gr} \not\models \neg\forall x \forall y (x \circ y \to y \circ x)$ since there is a commutative group of size 1. This example shows that $\Phi \not\models \phi$ does not necessarily imply $\Phi \models \neg\phi$.

The theory of axioms Φ is the set $\Phi^{\models} := \{\phi : \Phi \models \phi\}$.

[6]A universal sentence has no existential quantifiers, which logically prohibits sentences of the form $\neg\forall\neg\phi$.

4.2. **Searching for a Counterexample to a Conjecture.** An algorithm follows from the above definitions. Given a set of axioms Φ and a conjecture ϕ, we attempt to show that $\Phi \not\models \phi$ by checking models of Φ until finding a counterexample of ϕ. If a counterexample is not found, then we remain unsure whether $\Phi \not\models \phi$.

4.3. **Assistance in Enumerating a Theory.** We first generate all conjectures [we are still looking for a clever method]. A naive method is to enumerate all strings from our alphabet, try to parse each, and only keep the sentences.

Given axioms Φ, generate models up to size m using the above method. Next, generate all conjectures of growing size, and try to refute each one against the models of Φ. Many conjectures may be refuted, leaving fewer conjectures to be checked manually. Although this does not enumerate a theory, it does lessen the burden of hand proving all possible conjectures.

A question arises: Is it possible to enumerate a theory without using such brute force methods? Similar to how some axioms allow their models to be enumerated without a brute force search, as we have shown for equivalence relations.

Another question arises: Can a theory up to sentence length n be enumerated with only models up to size m? A bound is the length of the sentence $\phi_{\leq m}$ which says that any model has at most size m. Perhaps another bound can be found through the "Kolmogorov complexity" of a sentence of length at most n.

5. Provability and Consistency

5.1. **Definitions and Theorems.** $\Phi \vdash \phi$, read "Φ proves ϕ" or "Φ derives ϕ", means sentence ϕ can be proved[7] from set Φ of axioms. Φ is consistent means that it cannot prove both ϕ and $\neg\phi$.

Godel's completeness theorem says that $\Phi \vdash \phi$ iff $\Phi \models \phi$. Another form of Godel's completeness theorem says that Φ is consistent iff Φ is satisfiable.

5.2. **Consisistency of Axioms and Provability of a Conjecture.** Godel's completeness theorem allows us show consistency of axioms by showing their satisfiability. Likewise, show $\Phi \nvdash \phi$ by showing $\Phi \not\models \phi$.

6. Other Topics of Interest

6.1. **Searching for and Comparing Axioms.** One can use the above tools to search for alternative axioms, perhaps with some desired or optimal properties. The tools can also check if axioms are logically equivalent[8] up to some model size. The tools can also be used to check if models of some axioms are a subset of models of other axioms, up to model size.

6.2. **As a Tool For Other Areas of Math.** Perhaps proving lemmas (eg satisfiable, consistent, consequence, derivable) up to model size m could be useful to other areas of Math.

6.3. **Looking for Connections.** Count the numbers of models of each size, and input to the Online Encyclopedia of Integer Sequences to make connections with other areas of math. For example, one can discover that equivalence relations correspond to partitions by noticing that models of equivalence relations, indexed by size, come in quantities 1, 2, 5, 15, 52, ...[9], which correspond to counting partitions.

---

[7]Proves means derivable from a set of derivation rules, see Ebbinghaus.

[8]Logically equivalent sentences have the same models.

[9]These are the Bell numbers.

7. Future Work

The conjecture refuter is under construction.

After the conjecture refuter is complete, we will look for an elegant way to list all sentences, starting with the smallest ones. These sentences will be conjectures for our conjecture refutor.

References

[1] H.-D. Ebbinghaus, J. Flum, W. Thomas, *Mathematical Logic*, second edition, Springer, 1994.

[2] Dworzanski, P., *Enumerating Equivalence Relations*, implemented and to be written-up, 2016.