Modelling puzzles with Logic4Fun

Overview, Core Components, Other Features

Ranald Clouston

March 5, 2025

Australian National University

Modelling into Logic with Logic4Fun

One of the central themes of this course is modelling informal language into formal logic.

We will practice this in tutorials and lectures via a website called Logic4Fun.

Your mission: help a program solve logic puzzles by stating them in formal language the solver can understand.

We will get a feel for the system by taking on the Four Horsemen problem.

Overview of Logic4Fun

Model Finding

Logic4Fun finds models: a semantic situation which makes the statements you enter true.

- If a model is found, our statements are satisfiable.
- If we want to check validity, enter the negation of what you want to check, and see if any models are found.
- With logic puzzles, the expectation is that exactly one model can be found - unique satisfiability. With zero models, or more than one, you have probably made a mistake.
- By default Logic4Fun only gives the first 3 models it finds.
- Logic4Fun is *not* complete: it can time out without giving you an answer.

Model Finding for Propositional Logic

With propositional logic statements, the models Logic4Fun finds are truth table rows.

Let's try entering some propositional logic into the 'Constraints' box:

- $\neg(p \to p)$. or in text: `NOT (p IMP p).`
- $p \vee q \to p \to r$. or `p OR q IMP p IMP r.`
- Add a line to the one above: $(p \to q) \to q \wedge \neg r$. or `(p IMP q) IMP q AND NOT r.`

Reflections on our Examples

A few notes:

- Like all formal machine-checked languages, Logic4Fun is fussy about syntax: parentheses and capitalisations matter, each constraint must end with a full stop, etc.
- Multiple constraints are implicitly conjoined, so we can refactor by breaking top level `AND`s into multiple lines, or put multiple lines together using `AND`.
- Our work produced a warning, 'No user defined vocabulary was used'. This is because the names p, q, r are not explicitly defined. In our case no harm is done because they are correctly 'guessed' by Logic4Fun to be unknown truth values.
- The choice between logical symbols and the recognised keywords like `IMP` is arbitrary: it is the abstraction that matters, not the notation, provided that Logic4Fun, or the human reading your work, understands your notation.

Model Finding More Generally

Propositional logic seems an awkward fit for the Four Horsemen puzzle.

- But possible! Introduce a proposition for all possible situations: ‘Mountback rides the bay’, ‘Mountback rides the chestnut’, ’the bay is jumping’, etc, then painstakingly write down propositions to encode which of these facts are incompatible.

What would a more tractable notion of model look like?

- We want an assignment of riders to horses, and horses to activites....
- i.e. we want (injective and surjective) functions from the set of riders to the set of horses, and of horses to activities.

Core Components: Sorts, Vocabulary, Constraints

Sorts

The notion of set is captured in Logic4Fun by sorts.

- All sorts are finite, and all that we use will be non-empty.
- All sorts come with an ordering.

There are three ways to define a sort:

`Foo.`	could have any size. Its elements will be called $0, 1, \ldots$
`Foo cardinality = 4.`	has size 4. Its elements are called $0, 1, 2, 3$.
`Foo enum : bar, baz, qux.`	has 3 elements with the given names, in that order.

Which approach is right for the Four Horsemen puzzle?

Built-In Sorts

Logic4Fun has two built-in sorts: `bool` and `natnum`.

`bool` has two elements, `FALSE` and `TRUE`.

`natnum` is the natural numbers from 0 to, rather arbitrarily, a maximum of 29.

- Sufficient for logic puzzles but certainly not acceptable for a general prover!
- In some places the name `int` is used instead; they are synonyms.

These sorts comes with built-in operations, which we will meet soon.

Functions

We declare functions between our sorts in the Vocabulary section.

e.g define a unary function `bar` from sort `foo` to `natnum` by

- `function bar (foo) : natnum.`

Comma separate for multiple arguments: `function bar (foo,bool,foo) : natnum.`

- When we use binary functions we usually write them infix.

If we want more than one function we can write each in a new line inside curly braces:

```
function {
   ⋮
}
```

Predicates and Names

We can do all our work with the function notation, but there are two cases common enough to have specific notation:

A predicate is a function with result sort `bool`.

- We often want to assert certain statements are `FALSE` or `TRUE`.
- Instead of `function bar (foo) : bool` we may write `predicate bar (foo)`.

A name is a nullary function, i.e. an element of a sort.

- Instead of `function bar () : foo` we may write `name bar : foo`.

Built-in Functions

Logic4Fun comes with some built-in functions, some particular to the built-in sorts and some that work for any sort `S`. Link to full list, but here are some important examples:

- The propositional connectives, which are all predicates e.g. `predicate OR (bool,bool)`. Includes `XOR` and `IFF`.
- Equality predicate, `= (S,S)`. Instead of `NOT (x = y)` we can write `x <> y`.
- Inequality predicates : `< (S,S)` and similarly <=, >, >=.
- `SUCC (S) : S` which gives the next element of `S` but is undefined if there is none.
- `+ (S,natnum) : S` which gives the *n*th next element of `S` if possible.
- Predecessor `PRED` and subtraction `-`, both similarly partial.
- `name MIN : S`, also written `MIN_S`, the first element of `S`. Similarly `MAX`.
- `DIF (S,S) : natnum` gives the absolute difference betwen two `S` elements.

Properties of Functions

After a function's declaration we can write one (or, comma separated, more) properties inside curly braces. These are some properties of functions (including predicates and names) commonly useful enough to be built in. Link to full list, and some useful examples:

- `all_different` makes the function injective. Similarly, `surjective`.
- `partial` means the function can fail to return an answer, as with e.g. `SUCC`.
- For binary functions only, `commutative`.
- If the answers we are getting are too detailed, `hidden` stops a function printing.

We now have enough notation to fill in the Vocabulary box for the Four Horsemen puzzle.

Constraints

Logic4Fun will try to guess what our functions, predicates, and names are, but we have probably not yet provided enough information to get a unique answer.

The Constraints box is where we write all the things of sort `bool` we would like to hold, extracted from the puzzle clues.

- Let's do that with our Four Horseman
- As we can see, we sometimes need to work to find all the constraints hiding in a jumble of natural language.
- Important constraints will sometimes be entirely implicit (not in the text), but rely e.g. on our common sense understanding of how the concepts that are in the text behave.

Other Features

Comments

Logic4Fun has single line comments, by writing % to the left.

Useful both for annotating complex contraints, and for blocking out constraints to see what models are generated without them.

EST

`EST` is a built-in predicate, defined on any sort, meaning 'Exists Such a Thing'.

Useful because some Logic4Fun functions are partial (although total by default)

- e.g. given a name `foo` in some sort, the constraint `EST (SUCC foo)` means that there exists a successor of `foo`. We could also write this `foo <> MAX`.

We will mostly try to use total functions, but it sometimes saves effort to use the ordering on a sort, in which case `EST` can be useful to state that something is ‘in bounds’.

Quirks

Although Logic4Fun is in some ways very impressive, it is under development and has some rough edges.

- The live parse warnings are a bit out of synch with the 'Check Syntax' button; it is the latter that matters.
- The 'Diagnose' button currently does nothing.
- The 'Settings' page displays weirdly in some browsers.
- No support for password change or recovery!
    - Write your password down some place(s) safe.
    - If you do lose your password contact course staff on Ed. We can either delete your account or give you a new password (which will therefore not be completely private).

ALL and SOME

Let's look at the Four Trees puzzle. A function `position` is provided from defined sort `tree enum :  oak, ash, elm, fir` to `natnum`.

- We want the result of `position` to be from 1 to 4. How do we specify this?
- With propositional logic: `position oak = 1 OR position oak = 2 OR position oak = 3 OR position oak = 4`, and similarly for the other trees.

Tedious! Instead use the built-in predicate `ALL`, or $\forall$, of first order logic.

- `ALL` takes two arguments: a variable of any sort, and an expression of sort `bool` which might mention that variable.
- e.g. `ALL x (position x > 0).` The parentheses are necessary here.

`SOME`, or $\exists$, defined similarly.

Towards First Order Logic

Logic4Fun is specialised for logical problems where sorts are finite, and indeed very small.

In this setting `ALL` and `SOME` are not truly necessary, but can save work.

- sometimes, a lot of work!

In the wild we often quantify over sets that are infinite, e.g. mathematicians with "for any natural number x, ...".

So the safe little world of Logic4Fun is not sufficient to explore first order logic as used in all places; we will now step away from our tool and explore first order logic in general.