1 title

Good morning everybody, today, it is an honour for me to present my dedicated proof assistant named “Phometa” which is a tool that can be used to construct a formal system and prove its theorems.

2 Derivation systems can reason many formal systems.

Derivation system is capable to reason about many formal systems. For example, propositional logic, we can prove that [point] $p \land q$ holds under assumption that [point] p and $p \to q$ hold, using grammars and derivation rules of simple arithmetic.

[next page]

The similar things also happen in simple arithmetic and typed lambda calculus as well.

3 Problem of drawing a derivation tree.

However, drawing a derivation tree manually in a paper is painful and tedious. This is because its width grows exponentially to its height so it is hard to fit in a paper, if a mistake or unification occur, the entire tree might need to be redrawn again, and most importantly, it is doesn’t have a precise way to verify that a tree is error free.

So, I decided to build “Phometa” to encode any formal system that support derivation tree in order to solve the problem above.

4 Screenshot of “Phometa” in action

[point to screen] Here is a screenshot that shows Phometa in action, you can see that the derivation tree from the first slide can be encoded into Phometa systematically and the tree doesn’t have problem of its width grows exponentially to its height anymore.

We also exploit the power of visualisation such as using underlines to group sub-terms instead of brackets.

5 Formal system ingredient - Grammar (Backus-Nour Form)

To encode any formal system, we need to defines its grammars and its derivation rules. For propositional logic, we can define grammars starting with “prop” [point to bnf of prop] which is, in fact Backus-Nour Form. Each alternative can be mapped into Phometa grammar node [point to grammar node of "prop"] of property "choice". Then a term [point to the top of parse tree] can be instantiated to one of these alternatives. In addition, Phometa also allow a term to be Phometa also allow Grammar node also has a property "metavar_regex" that allow "prop" to have meta variables under a condition that it must follows this [point to regex] regex pattern.

Now let see node grammar of "atom", we use literals instead meta variables here and the different between them is that a meta variable can match any term of the same grammar but a literal only match to itself.

6 Grammar (Backus-Nour Form)

We also need a grammar for "context" which is a set of assumptions and "judgement" which is a pair of "context" and "prop" now we can show the parse tree of the judgement in the screenshot on the previous slide.

7 Formal system ingredient - Rule (Derivation Rule)

Once we define grammars, we can define derivation rules. Nevertheless propositional logic has so many rules so I will just explain one of them which is "imply-elim". This rule state [point to derivation tree] that we can prove that B holds if $A \to B$ and A hold using the same assumptions. Premises and conclusion of this derivation tree can be mapped directly into this rule node using properties "premise" and "conclusion" [point to those keywords respectively].

A rule will be used mainly to build a tree, for example [point to the sub-theorem], we have an unfinished tree with a goal state that q holds under assumptions that p and $p \to q$ hold, we can apply rule "imply-elim", the rule will *pattern match* its conclusion against the goal, this results in γ as list of p and $p \to q$, and B as q. This pattern matching result will substitute each of premises to get further sub-goals that can be fulfilled similar method to this goal.

Some meta variable such as A appears in premises but not in the conclusion, so it cannot be substituted by pattern matching result, this kind of meta variable is called "parameter" where we need to specify explicitly every time when we apply a rule. This is become clearer when we go though during demonstration.

8 Formal system ingredient - Theorem (Derivation Tree)

The last ingredient theorem node which are, in fact, derivation trees.

Each tree consists of goal (on r.h.s.), a rule (on l.h.s.), and possibly further sub-trees for sub-goals (above).

9 Demonstration

Now, I will show you a video that demonstrate Phometa, this should take around 10 minutes before we resume back to the slides.

10 Cascade Premise — Hypothesis Rule

According to the demonstration, hypothesis rule is clearly not a ordinary inference rule because it can penetrate throughout of assumptions. In fact hypothesis rule use spacial premise called "Cascade Premise", which will try to call its first sub-rule, if success then include sub-goals of sub-rules and its sub-goals, otherwise cascade down to lower sub-rule and repeat again. So for hypothesis it call hypothesis-base that can prove the judgement that has the conclusion as the the last assumption, if it fails, remove its last assumption and call itself again, until the right assumption is found.

11 Meta-Reduction — Context Manipulation

Another feature that is missing out from the demonstration is meta-reduction, well, context is supposed to be a set but we encode it as a list, hence we should be able to freely permute and duplicate assumptions in context around implicitly. To archive this ability, we can define a rule that has exactly one premise and set the flag "allow_reduction", so we can manipulate term in active sub-goals of a theorem by clicking one of its sub-term, then we can select the corresponded rule in the keymap plan, then the conclusion will pattern match against the target term and replace it with substituted version of premise. For example this judgement as this context as sub-term so if we use "context-commutative" the context will swap the last two assumption.

12 Implementation — Elm language and its Signals

The main language that is used to implement Phometa is Elm which is haskell-like programming language that compile to JavaScript. One of the most attractive feature of Elm is Signal which is an object that change its value over the time. For example `Mouse.isDown` is a Signal of type `Bool` and its value will be changed whenever the mouse is clicked or released.

We can manipulate signals around [point] as described in dependency graph, eventually, we will get a signal of type HTML which can be displayed in the web browser, for example, the code on the screen will be complied to Html that say “foo”, if we click mouse on the screen, it will change to “bar”. It will go to “To click again” again once we release the mouse.

13 Implementation — Model-Controller-View (MCV)

However, programming in practical require internal state of the program, so we can use foldp or “fold from the past” to accumulated an incoming event with the previous state of the program. This results in MCV pattern and the entire implementation is split into 3 questions.

first, how to model the global state of the program

second, how to write a function that merge an incoming event with the past model

and third, how to and write a function to convert the global state of the program into html.

Therefore, the logical and user interface of the programming is completely separated, this combine with the fact that it is functional programming can greatly reduce the possibility of bugs and errors.

Please note that, this is a simplified version of phometa main entry, the real thing need to deal with outside world as well such as communicating with the server to load and save the proof repository.

14 Strengths, Limitation, and Future Work

Phometa has additional features over the derivation system such as prove by lemma, cascade premise, and meta-reduction.

It is also easier to learn compared to popular proof assistants such as Coq or Agda.

And ultimately, it is impossible to construct an invalid proof.

But on the down side, phometa need to load everything when it starts, this will become problematic when the repository is large.

There are also several other limitation that can be considered as future works for example, some proving step should be more automatic, nodes should be exportable to latex source code, and importation among modules.

15 Achievement & Conclusion

As the result of this project, I have successfully designed and implemented a proof assistant to solve the problem regarding to derivation tree at the beginning of this presentations..

I also showed that this program work well for most of formal systems by writing several famous formal systems in standard library.

I also showed that the program is easy to use by anybody by writing a tutorial that is short and doesn't require prior knowledge.

In conclusion, I believe that Phometa is powerful and easy enough to be used as a replacement of traditional derivation tree. This is at the very least, can help student to do their coursework that need to draw derivation trees.