Using Forward Chaining to Go Backward

Nil Geisweiller

SingularityNET Foundation,
Zug, Switzerland
nil@singularitynet.io

1 Introduction

If all you have is a forward chainer, is it possible to emulate a process that is functionally equivalent to backward chaining? This question stems from the observation that some rewriting systems, such as MORK [9], perform better on forward chaining than on backward chaining [2]. This is to be contrasted with some other rewriting systems such as PeTTa [4], a Prolog-based rewriting system, which performs equally well on forward and backward chaining [2]. Since forward chaining in MORK outperforms forward chaining in PeTTa [2], it gives us an incentive to emulate backward chaining using forward chaining to obtain a MORK-based backward chainer that outperforms PeTTa.

2 Forward vs Backward Chaining

Forward chaining is a reasoning process that begins with axioms and apply inference rules to produce theorems. For instance, using the axioms and inference rules of a fragment of propositional calculus (borrowed from Metamath [5]), given with a type theoretic representation:

- $\text{ax}_1 : \phi \to (\psi \to \phi)$ $\qquad\qquad$ $\text{ax}_2 : (\phi \to (\psi \to \chi)) \to ((\phi \to \psi) \to (\phi \to \chi))$
- $\text{mp} : (\phi \to \psi)$ -> ϕ -> ψ

where -> is the arrow type and $\to$ is the implication logic connector, one can build a proof of the identity theorem in a forward way:

1. apply mp to ax_2 and ax_1 to obtain $(\text{mp } \text{ax}_2\ \text{ax}_1) : (\phi \to \psi) \to (\phi \to \phi)$,
2. apply mp to step 1 and ax_1 to obtain $(\text{mp } (\text{mp } \text{ax}_2\ \text{ax}_1)\ \text{ax}_1) : \phi \to \phi$.

On the contrary, a backward chainer starts with the proposition to be proven, and expands the proof backward till it reaches the axioms. For instance, the process may start with $x? : \phi \to \phi$ where $x?$ is a hole, then expand $x?$ backward using mp to obtain $(\text{mp } y?\ z?) : \phi \to \phi$, and keep expanding until obtaining the full proof $(\text{mp } (\text{mp } \text{ax}_2\ \text{ax}_1)\ \text{ax}_1) : \phi \to \phi$. The advantage of backward chaining is that the search space is constrained by proofs that can only prove the target proposition, resulting in a lot of pruning. The drawback is that it is more sophisticated, involving full syntactic unification and maintaining a stack of hypothetical paths.

3 Emulating Backward Chaining with Forward Chaining

Given a theory described by axioms and inference rules, the idea is to invert the inference rules using the *blackbird* combinator [8] to obtained an *inverted theory*. Applying forward chaining on this inverted theory will have the effect of producing hypothetical truths that preserve their

conclusions, therefore emulating backward expansion. For instance, given the theory described earlier, we apply the blackbird combinator, recalled below:

$$B_1 : (\chi \text{ -> } \theta) \text{ -> } (\phi \text{ -> } \psi \text{ -> } \chi) \text{ -> } \phi \text{ -> } \psi \text{ -> } \theta$$

to the Modus Ponens inference rule to obtain[1]:

$$\lambda x.((B_1\ x)\ \text{mp}) : (\psi \text{ -> } \chi) \text{ -> } (\phi \to \psi) \text{ -> } \phi \text{ -> } \chi$$

which can be read as "given a hypothetical proof of χ assuming ψ, produce a hypothetical proof of χ assuming $\phi \to \psi$ and ϕ", precisely corresponding to Modus Ponens backward expansion. For the sake of simplicity let us rename this inverted inference rule:

$$\text{mp}^i \stackrel{\text{def}}{=} \lambda x.((B_1\ x)\ \text{mp})$$

We can now use it in a forward way to prove[2] the identity theorem shown earlier. We start from a trivial hypothetical truth using the *idiot* combinator [8]:

1. $I : (\phi \to \phi) \text{ -> } (\phi \to \phi)$

then apply mp^i twice (backward expansion)

2. $(\text{mp}^i\ I) : (\psi \to (\phi \to \phi)) \text{ -> } \psi \text{ -> } (\phi \to \phi)$
3. $(\text{mp}^i\ (\text{mp}^i\ I)) : (\chi \to (\psi \to (\phi \to \phi))) \text{ -> } \chi \text{ -> } \psi \text{ -> } (\phi \to \phi)$

then apply the result of step 3 to axiom ax_2 (reach the second axiom)

4. $((\text{mp}^i\ (\text{mp}^i\ I))\ \text{ax}_2) : (\phi \to (\psi \to \phi)) \text{ -> } (\phi \to \psi) \text{ -> } (\phi \to \phi)$

then apply the result of step 4 to axiom ax_1 twice (reach the first axiom)

5. $(((\text{mp}^i\ (\text{mp}^i\ I))\ \text{ax}_2)\ \text{ax}_1) : (\phi \to \psi) \text{ -> } (\phi \to \phi)$
6. $((((\text{mp}^i\ (\text{mp}^i\ I))\ \text{ax}_2)\ \text{ax}_1)\ \text{ax}_1) : \phi \to \phi$

giving us a prove of the identity theorem. The proof is equivalent to the one provided earlier $(\text{mp}\ (\text{mp}\ \text{ax}_2\ \text{ax}_1)\ \text{ax}_1)$ but is constructed inside-out. The reader may notice that this process of construction emulates a backward chaining depth-first search, and that the proof encodes a search path along the search tree. It is also possible to emulate breadth-first search by using the *bluebird* combinator [8] and may be the subject of future work.

4 Experimental Results

Experiments have been conducted to compare such backward chaining emulation with regular backward chaining based on a corpus ported from Metamath [5] to MeTTa [1, 7]. To establish a fair comparison, care has been taken to ensure that both chainers search isomorphic spaces and run on the same back-end, PeTTa. It has been found that the emulation is twice slower in the worse cases to almost as fast in the best cases compared to regular backward chaining [3]. This indicates that it could be a competitive approach on rewriting systems that excel at forward chaining such as MORK, though more work is required to determine exactly how much. Although we understand that brute force search, however fast, is never going to replace inference control [6], we believe it still serves as a base that may underlie the performance of the whole system, and thus is worth our attention.

[1] The second argument of B_1 is the inference rule to invert, here $\text{mp} : (\phi \to \psi) \text{ -> } \phi \text{ -> } \psi$, thus $(\phi \to \psi) \text{ -> } \phi \text{ -> } \psi$ must unify up to alpha conversion with $\phi \text{ -> } \psi \text{ -> } \chi$, which it does.

[2] Consider that variables are scoped within each line, not across lines, and thus unification is performed using alpha conversion when necessary.

References

[1] MeTTa Language Official Website. https://metta-lang.dev.

[2] Nil Geisweiller. Chaining Benchmarks Comparing PeTTa and MORK on Metamath. https://github.com/trueagi-io/chaining/tree/main/experimental/metamath/benchmarks.

[3] Nil Geisweiller. Emulating Backward Chaining via Forward Chaining. https://github.com/trueagi-io/chaining/tree/main/experimental/backward-via-forward.

[4] Patrick Hammer. PeTTa (Prolog Meta Type Talk) Github Repository. https://github.com/trueagi-io/PeTTa.

[5] Norman D. Megill and David A. Wheeler. *Metamath: A Computer Language for Pure Mathematics*. Lulu Press, Morrisville, North Carolina, 2019.

[6] Nil Geisweiller. Estimating the Probability of a Conjecture to be a Theorem with PLN for Inference Control. *Artificial Intelligence and Theorem Proving*, 2025.

[7] Alexey Potapov. MeTTa Specification. https://wiki.opencog.org/w/File:MeTTa_Specification.pdf, August 2021.

[8] Raymond M. Smullyan. *To Mock a Mockingbird and Other Logic Puzzles: Including an Amazing Adventure in Combinatory Logic*. Knopf Doubleday, New York, 1st edition, 1985.

[9] Adam Vandervorst. MORK (MeTTa Optimal Reduction Kernel) Github Repository. https://github.com/trueagi-io/MORK.