Knowledge Representation & Reasoning: LK Calculus Reasoner

Luke Slater (luke.slater.1@kaust.edu.sa)

September 30, 2015

1 Implementation

The basic LK system has been implemented with all logical and structural rules, discluding the cut rule and those involving quantifiers. It is implemented in Javascript.

In retrospect, I should not have implemented an undecidable problem in Javascript. Javascript is not an appropriate language for logical argument deduction.

2 Hilbert Calculus Proofs

Axiom 1 Proof

```
Step 0
Formula 0
  Operation: IN
  Value:  ⊢ A ⇒ A
Step 1
Formula 0
  Operation: IR
  Value: A ⊢ A
```

Axiom 2 Proof

```
Step 0
Formula 0
  Operation: IN
  Value:  ⊢ A ⇒ (B ⇒ A)
Step 1
Formula 0
  Operation: IR
  Value: A ⊢ B ⇒ A
Step 2
Formula 0
  Operation: IR
  Value: A, B ⊢ A
Step 3
Formula 0
  Operation: WL
  Value: A ⊢ A
```

Axiom 4 Proof

```
Step 0
Formula 0
  Operation: IN
  Value:  ⊢ ((¬A) ⇒ (¬B)) ⇒ (B ⇒ A)
Step 1
Formula 0
  Operation: IR
  Value: (¬A) ⇒ (¬B) ⊢ B ⇒ A
Step 2
```

```
Formula 0
  Operation: IR
  Value: (¬A) ⇒ (¬B), B ⊢ A
Step 3
Formula 0
  Operation: PL
  Value: B, (¬A) ⇒ (¬B) ⊢ A
Step 4
Formula 0
  Operation: IL
  Value:  ⊢ ¬A, A
Formula 1
  Operation: IL
  Value: B, ¬B ⊢
Step 5
Formula 0
  Operation: NR
  Value: A ⊢ A
Formula 1
  Operation: NL
  Value: B ⊢ B
```

Axiom 4m Proof

```
Step 0
Formula 0
  Operation: IN
  Value:  ⊢ (A ⇒ B) ⇒ ((A ⇒ (¬B)) ⇒ (¬A))
Step 1
Formula 0
  Operation: IR
  Value: A ⇒ B ⊢ (A ⇒ (¬B)) ⇒ (¬A)
Step 2
Formula 0
  Operation: CR
  Value: A ⇒ B ⊢ (A ⇒ (¬B)) ⇒ (¬A), (A ⇒ (¬B)) ⇒ (¬A)
Step 3
Formula 0
  Operation: IL
  Value:  ⊢ A, (A ⇒ (¬B)) ⇒ (¬A)
Formula 1
  Operation: IL
  Value: B ⊢ (A ⇒ (¬B)) ⇒ (¬A)
Step 4
Formula 0
  Operation: PR
  Value:  ⊢ (A ⇒ (¬B)) ⇒ (¬A), A
Formula 1
  Operation: IR
  Value: B, A ⇒ (¬B) ⊢ ¬A
Step 5
Formula 0
  Operation: IR
  Value: A ⇒ (¬B) ⊢ ¬A, A
Formula 1
  Operation: IL
  Value:  ⊢ A, ¬A
```

```
Formula 2
  Operation: IL
  Value: B, ¬B ⊢
Step 6
Formula 0
  Operation: WL
  Value:  ⊢ ¬A, A
Formula 1
  Operation: PR
  Value:  ⊢ ¬A, A
Formula 2
  Operation: CR
  Value: B, ¬B ⊢
Step 7
Formula 0
  Operation: NR
  Value: A ⊢ A
Formula 1
  Operation: NR
  Value: A ⊢ A
Formula 2
  Operation: NL
  Value: B ⊢ B
```

3 Semantic Entailment

4 Modus Ponens

Modus Ponens Proof

```
Step 0
Formula 0
  Operation: IN
  Value: A ⇒ B, A ⊢ B
Step 1
Formula 0
  Operation: PL
  Value: A, A ⇒ B ⊢ B
Step 2
Formula 0
  Operation: IL
  Value: A ⊢ A
Formula 1
  Operation: IL
  Value: B ⊢ B
```

5 Time Complexity

6 Performance Optimisation

It seems that there are many potential performance enhancements which could be applied to attempt to improve the performance of the calculus prover. In the methods I have attempted, the focus has been on reducing the number of track growth between steps of the search:

- Remove dead tracks: these are tracks with subformulas without any possible solutions, such as:
  - $\vdash$ A
  - Empty set both sides of the sequent
  - Total of one symbol on both sides of the sequent
- Limitation of cut rules; many unhelpful tracks were being generated by the constant application of structural rules, so these were limited. The following cases were disallowed:
  - The last rule was CR and the current rule is WR
  - The last rule was CR and the current rule is CR
  - The last rule was PR and the current rule is CR
  - The last rule was PR and the current rule is WR
  - The last rule was CL and the current rule is WL
  - The last rule was CL and the current rule is CL
  - The last rule was PL and the current rule is CL
  - The last rule was PL and the current rule is WL

I think that there is potential that some of these rules could prevent the discovery of proofs in some cases - particularly the limitation of repeated structural rules in the case of more complicated formulas. However, I did not come accross any such problems during testing.

Given more time to rewrite the system to be a bit more flexible, I think that a major improvement could be to disallow the use of structural rules entirely, until it has been found that there is no possible solution to be found by the application of logical rules only. It should be possible to traverse all possibilities, since the LK system without structural rules should terminate.

The main difficulty here is to implement effective backtracking, figuring out where to start applying structural rules in the case of a dead path. It might seem like the better option to start applying structural rules from the last non-dead step, but this will cause problems. Take the example of the Law of Excluded Middle ($\vdash$ A $\lor$ $\neg$A), the proof for which requires an application of the CR rule as the first step - it is possible to apply a maximum of two steps before exhausting logical rule possibilities (in the case we use OR2 and NR).

Thus, a solution would have to attempt to find a solution using only logical rules, and upon failure apply structural rules on the first step, potentially leading to many more sub-problems than is actually necessary.

I think the superior solution would be to implement a modified version of the LK system which operates on sets rather than lists. This works because we know that the contraction and permutation rules mean the order and number of symbols in the sequence don't matter, and cannot prevent a valid proof being found.

Furthermore, many implementations of LK calculus also use 'weakening,' which allows the addition of arbitrary elements to the set of formulae. This too allows us to assume a proof (in which a formula is considered equivalent to A $\vdash$ A) without the use of structural rules. This also leads to shorter proofs.

In combination, we