Lucid Expressions: Boolean Expressions That Reduce to Canonical Form in Polynomial Time

This document describes Lucid Expressions (LEs).

Lucid Expressions are a form of textual, ordered, existential graph,
with symmetric rules of inference,
that can be reduced to their canonical form in polynomial time.

The basic structure of this document is as follows...

- Introduction

  This section gives an overview of existential graphs (EG) and abstract reduction systems, the shortcomings of the EG system, and how LE solves them.

  - Logic and Reduction Systems

    - Logic systems can be viewed as a type of abstract reduction system.

      > ...where logical formulas are transformed using rewrite rules based on logical equivalences.

    - A good reduction system is sound, complete, terminates, and is confluent.
    - A good logic system is sound, complete, and... *terminates, and is confluent?*
    - Reduction systems with rules that are reversible are locally confluent.
    - Rules that are guaranteed to simplify ordered expressions are also globally confluent and guaranteed to terminate.

    > Therefore, an ordered rewrite system with reduction rules that are sound, complete, and reversible is also confluent and guaranteed to terminate.
    > And such a system could also be a system of logic.

  - Existential Graphs

    - EG is a system of logic for automated reasoning *and* also a reduction system.
    - EG is sound and complete, but not confluent. Not many logic systems are.
    - EG has 5 inference rules; insertion/erasure, double cut elimination, and iteration/de-iteration.

      > Insertion/erasure, and iteration/de-iteration are *reversible, symmetric pairs of operations*.
      > Double-cut elimination is not.

    - The fact that the rules are not all reversible destroys confluence in EGs.

- Lucid Expressions
  This section explains LEs by way of describing the differences between them and EGs.

LE is a system of logic, and a reduction system, based on existential graphs.
Unlike the EG, which prioritize the readability of graphs, LE prioritizes making expressions as simple as possible for computers to understand.

- Notation/Syntax:
  LEs uses a linear, textual notation, as opposed to EGs
  - LEs use the constants T and F to represent an empty 'sheet of assertion' and an empty cut.
  - Variable are lower-case hexadecimal numbers
  - Expressions may be wrapped in parentheses and separated by a space: like this... (a b)

    > The parentheses are supposed to look like the edges of a cut in an EG.

    > EG interprets (a b) as NOT(AND(a,b)) because this makes things easier for humans.
    > LE interprets (a b) as NAND(a,b) because this makes things easier for computers while preserving functional comnpleteness.

- Inference Rules
  - Insertion/Erasure: Insert and erase double cuts anywhere in an expression.

    > Put another way, these rewrite rules are valid for any subterm S in an expression E...
    > E[S] => E[S->(T (T S))],
    > E[(T (T S))] => E[(T (T S))->S

  - Iteration/Deiteration: Replicate/remove copies of a subterm to/from a sibling.

    > E[(L X(T))] => E[(L X(T->L))],
    > E[(L X(L))] => E[(L X(L->T))],
    > E[(X(T) R)] => E[(X(T->R) R)],
    > E[(X(R) R)] => E[(X(R->T) R)]

  > These six rewrite rules are sound, complete, and reversible(symmetric).

- LEs are ordered

  > For example, (T (a (b c)) is a simpler expression than (((c b) a) T).

  > Defining a rewrite order) makes it possible to guarantee that rules only simplify expressions, and guarantees that reduction terminates.

- Reduction
  - LEs are minimized (or proved) by repeatedly applying the rules of inference to an expression, but only one-way, when the result would *reduce* the expression.
  - Expressions that minimize to T/F are tautologies/contradictions.
  - Expressions are reduced by looking for opportunities to rewind/reverse an application of one of the inference rules.

- Completely reversing all the inferences embedded in an expression produces an expression that is canonical.
- The reduction method computes, and remembers, all *grounding cofactors* of all subterms in *mostly-canonical* expressions.

  > An expression E = (x y) is a *mostly-canonical* expression if x and y are canonical but E is not.
  > There's a big difference between mostly-canonical and all-canonical.
  > Mostly-canoncial is slightly apocryphal.
  > With all-canonical, well, with all-canonical there's usually only one thing you can do 😉.

  > A *term cofactor* of a boolean function F is derived by replacing all instances of a given term S in F with a constant value (T of F).

  > A grounding cofactor of a boolean function is a cofactor that is equivalent to T of F.

- The reduction method uses grounding cofactors to discover opportunities to apply the rules in a way that reduces an expression.
- Reduction is Abductive.

  > Applying a rule does not always result in a shorter expression.
  > Those applications are invalid and ignored.
  > This is abductive reasoning (educated guesses).
  > But there are guaranteed to be a linear # of educated guesses, which makes this method tractable.

- If there are no grounding cofactors then an expression is canonical.
- LEs can be reduced in polynomial time.

  > The satisfiability of LEs can also be determined in polynomial time.

- Reduction Algorithm

  This section provides pseudo code for the reduction algorithm and some examples.

- Conclusion

  It's been shown that LEs provide a tractable form of automated reasoning.

- Appendix: Completeness

  This section proves that the ME rules of inference are complete.

- Appendix: Computational Complexity

  This section proves that the computational complexity of reduction is O(n4) in the very worst case.

References

Dau

Mathematical Logic with Diagrams, Based on the Existential Graphs of Peirce;Dau.

Includes proofs of soundness and completeness for EGs

Cofactors

Linear Cofactor Relationships in Boolean Functions; Zhang, Chrzanowska-Jeske, Mishchenko, Burch

Rewriting Logic

Rewriting Logic as a Logical and Semantic Framework; Mart-Oliet, LEseguer

Cut Elimination

Confluence as a cut elimination property; Dowek

I'm not the first person to notice that cut elimination is a problem.

Orderings

Orderings for term-rewriting systems; Dershowitz

Existential Graphs of Peirce

Mathematical Logic with Diagrams, Based on the Existential Graphs of Peirce;Dau.