CNF Converter

Michael Humphrey

Colorado School of Mines

Fall 2021

Overview + Design

- Convert (mostly) any first-order logical expression into Conjunctive Normal Form (CNF)
  - Handle a number of different types of inputs (operator representations)
  - Handle different types of outputs (Infix, Prefix)
- Fast conversion to CNF
  - Multi-threaded
  - Memory efficient + leakless
  - Ability to track runtime and input/output size
- Easy to use + portable
- Fully parenthesized infix notation expected for input proposition

Grammar

$$
\begin{array}{lcl}
\langle exp \rangle & \rightarrow & \langle term \rangle \\
 & | & \langle uop \rangle \langle exp \rangle \\
 & | & \langle exp \rangle \langle bop \rangle \langle exp \rangle \\
\langle uop \rangle & \rightarrow & \neg \\
\langle bop \rangle & \rightarrow & \wedge | \vee | \rightarrow | \Leftrightarrow
\end{array}
$$

- Expressions are Terminals, UOPExpressions, BOPExpressions.
- Only 5 operators are considered, although adding more is trivial

Sample Inputs

- ((A) AND ((B) OR (C)))
- ((A) IFF ((B) IFF (C)))
- ((NOT (A)) OR ((B) AND (C)))
- ((A) $\rightarrow$ (B))
- $((A) \wedge ((B) \vee (\neg(C))))$

Sample Outputs

- $((A \vee (B \wedge \neg(C \vee (D \wedge E)))) \rightarrow \neg F)$
  - $(\neg B \vee C \vee E \vee \neg F) \wedge (\neg B \vee C \vee D \vee \neg F) \wedge (\neg A \vee \neg F)$
- $(C \wedge \neg(D \wedge ((E \rightarrow F) \vee (A \vee \neg B))))$
  - $(\neg D \vee B) \wedge (\neg D \vee \neg A) \wedge (\neg D \vee \neg F) \wedge (\neg D \vee E) \wedge (C)$
- $(A \Leftrightarrow (B \Leftrightarrow C))$
  - $(\neg C \vee \neg B \vee A) \wedge (\neg C \vee C \vee A) \wedge (B \vee \neg B \vee A) \wedge (B \vee C \vee A) \wedge (\neg A \vee \neg C \vee B) \wedge (\neg A \vee \neg B \vee C)$

Implementation

- Implemented in C
- All major functions are recursive down left and right expressions (for BOps) or down right expression (for UOps).
- Multi-threaded using pthreads

Runtime

- Most expressions take $< \frac{1}{200} th$ of a second to convert to CNF.
  - Some very large expressions (up to 50,000 clauses once in CNF) take up to 0.3 seconds.
- No checking is currently done to check if proposition already in CNF (could considerably speed program up on *some* inputs)

Challenges

- Memory
  - Managing memory through recursive functions, creating and manipulating expressions
  - Ensuring no memory leaks are possible, and edge cases are handled
- Multi-threading
  - New thread for each recursive call (left and right)
    - Much slower than sequential, system could potentially not create new thread if maxed out
  - New thread per recursive call up to certain recursion depth
    - Solves issue of limited threads
    - Still slower than sequential (less so for large inputs).

Questions

QUESTIONS?