Logical Operations on *ri*-Tries

Andrew Matusiewicz¹, Neil Murray¹, Erik Rosenthal²

¹ ILS Inst., Dept. of Computer Science, State University of New York, Albany, NY 12222, USA
² Dept. of Mathematics, University of New Haven, West Haven, CT 06516, USA

Summary

The reduced implicite trie (*ri*-trie), introduced in (Murray and Rosenthal, 2005), is a target language for knowledge compilation. It has the property that any query can be processed in time linear in the size of the query. In this paper, conjunction and negation are developed as update operations for *ri*-tries that do not require recompilation. Conjunction has been implemented, and experimental results, though preliminary, are promising.

Knowledge Compilation

Many queries to propositional knowledge bases are in \( \mathcal{NP} \), even those as simple as clause implication. The idea of knowledge compilation introduced in (Kautz and Selman, 1991) to try to manage the expense of such queries. The idea is to pay the exponential penalty once by compiling the knowledge base into a target language that would guarantee fast response to queries. They specified that the size of the target language be polynomial in the size of the original theory, and that query response time be polynomial in the size of the compiled theory. The result would then be polynomial response time to all queries.

**ri**-tries

The reduced implicite trie (*ri*-trie) takes a slightly different approach. Admit large compiled theories on which queries can be answered quickly. The *ri*-trie may have size exponential in the size of the original knowledge base, but it has been shown that *ri*-tries guarantee response time linear in the size of the query.

A reduced implicite trie (*ri*-trie) is a trie whose branches represent the relatively prime implicates (Murray and Rosenthal, 2007). If \( \mathcal{F} \) is a logical formula, then a relatively prime implicate is one for which no proper prefix (with respect to the variable ordering) is also an implicate. If the leaf node of a branch in an *ri*-trie is labeled \( \mathcal{p}_i \), then (extension with variables of index greater than \( i \) is a branch in the complete implicite trie of \( \mathcal{F} \). These extensions correspond to implicates of \( \mathcal{F} \) that are not relatively prime and that are represented implicitly by that branch in the *ri*-trie. The *ri*-trie of a logical formula \( \mathcal{F} \) can be obtained with the recursively defined \( \mathcal{R} \)-operator, introduced in (Murray and Rosenthal, 2005).

**Updates vs Recomputation**

It is typical in the knowledge compilation paradigm to assume that the intractable part of the processing is done only once (or at least not very often). In the absence of an efficient updating technology, this favors knowledge bases that are static, i.e., a single compilation is expected to provide a repository that remains useful over a large number of queries. The original knowledge base can always be modified and then recompiled, but in general this is expensive. As a result, updates that can be installed into the compiled knowledge base with recompiling have the potential to widen applicability considerably.

**Disjunction of *ri*-tries**

The INT operator on two *ri*-tries produces an *ri*-trie that is a branch-wise intersection of the given tries. This trie is logically equivalent to the disjunction of the two given tries by virtue of the fact that a clause is implied by \( \mathcal{F} \lor \mathcal{G} \) iff it is implied by both \( \mathcal{F} \) and \( \mathcal{G} \).

**Negation of *ri*-tries**

The \( \mathcal{R} \)-operator by itself produces a trie for a formula \( \mathcal{F} \) in which every leaf is labeled \( 0 \) or \( 1 \). Truth functional simplifications then yield the desired *ri*-trie. Without the simplifications, the trie is called a constant leaf trie (*cl*-trie). Merely swapping the 0’s and 1’s in a *cl*-trie will produce a representation of \( \neg \mathcal{F} \), but this is not the *cl*-trie of \( \neg \mathcal{F} \). The difficulty is the third conjunct. The *NEG*-operator recursively simplifies the first two conjuncts and then applies the \( \mathcal{R} \)-operator to produce the third.

**Conjunction of *ri*-tries**

In order to conjoin the *ri*-tries for formulas \( \mathcal{F} \) and \( \mathcal{G} \), first suppose that their first two subtrees can be conjoined pairwise. By definition, their intersection must represent the third subtree of the *ri*-trie for \( \mathcal{F} \lor \mathcal{G} \). Simply recurse like this on the first and second subtrees until we reach a leaf or a missing child. These are our base cases, as they represent constants in corresponding the *cl*-trie. Applying conjunction to these base cases will complete a recursive definition of a conjunction operator.

**Structure sharing**

Structure sharing allows us to reclaim some space by collapsing redundant subtrees to form a DAG-based trie as opposed to a tree. This can make the difference between trie sizes that are linear or exponential in the number of variables.

Formally, the nodes in an *ri*-trie are labeled with literals or the conjunction of literals. These extensions correspond to implicates of \( \mathcal{F} \) and \( \mathcal{G} \) that are not relatively prime implicates.

**Experiments**

In Table 1, each row represents the average time in milliseconds of five runs involving two random 3-CNF formulas over 25 variables. The first two columns are the number of clauses in the two formulas. The next three columns are time in milliseconds for compiling the first formula, compiling the second formula and conjointing it to the *ri*-trie for the first, and conjointing the two formulas and compiling the conjoint. The last column is the size of the compiled *ri*-trie in nodes.

**Conclusions**

In all but one case, it is less costly to update the trie than to recompose with the new formula. The advantage improves as the number of clauses in the first formula increases. The conjecture is that smaller clause sets are, in a sense, more satisfiable; their *ri*-tries are larger, but the cost of computing them is closer to linear than exponential. Larger clause sets lead to smaller *ri*-tries that require much more computation, compile times increase and update times decrease.

These results are based on a prototype only and are very preliminary. But they indicate (not surprisingly) that updating operations are a potentially useful alternative to recomposing.

**REFERENCES**


**Table 1: Update Experiments**

| # Clauses, \( \mathcal{F} \) | # Clauses, \( \mathcal{G} \) | \( \mathcal{F} \lor \mathcal{G} \) | \( \mathcal{F} \land \mathcal{G} \) | \( \mathcal{F} \lor \mathcal{G} \) & \( \mathcal{F} \land \mathcal{G} \) | \( \mathcal{F} \lor \mathcal{G} \) & \( \mathcal{F} \land \mathcal{G} \) & \( \mathcal{F} \lor \mathcal{G} \) & \( \mathcal{F} \land \mathcal{G} \) | \( \mathcal{F} \lor \mathcal{G} \) & \( \mathcal{F} \land \mathcal{G} \) |
|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|-----------------|
| 40               | 45               | 1498            | 4316            | 5520            | 1929            | 100             | 20              | 50              | 20              | 50              | 20              |
| 50               | 55               | 4754            | 5722            | 5702            | 12732           | 100             | 50              | 40              | 20              | 100             | 40              |
| 60               | 65               | 5552            | 4364            | 3756            | 8234            | 100             | 50              | 40              | 20              | 100             | 40              |
| 70               | 75               | 5538            | 4364            | 3756            | 8234            | 100             | 50              | 40              | 20              | 100             | 40              |
| 80               | 85               | 5538            | 4364            | 3756            | 8234            | 100             | 50              | 40              | 20              | 100             | 40              |
| 90               | 95               | 5538            | 4364            | 3756            | 8234            | 100             | 50              | 40              | 20              | 100             | 40              |
| 100              | 100              | 5538            | 4364            | 3756            | 8234            | 100             | 50              | 40              | 20              | 100             | 40              |