AI Project 3: Simple Automated Reasoning System

Roy Howie

April 7, 2017

1 Task

Create an automated reasoning system which resolves queries using either modus ponens or the resolution principle.

2 Definitions

2.1 Horn Database

A horn database is a set of sentences of the form $p_1 \wedge \cdots \wedge p_k \rightarrow q$. That is, for each sentence, the antecedent consists of a set of conjuncts and the consequent consists of a single symbol.

For example, $old(p) \wedge wise(p) \rightarrow sage(p)$ works but $child(p, q) \rightarrow father(q, p) \vee mother(q, p)$ does not.

2.2 Modus Ponens

Modus ponens is a complete operation when working with Horn databases which involves the combination of logical sentences.

More formally, if $p_1 \wedge \cdots \wedge p_k \rightarrow q$ and $r_1 \wedge \cdots \wedge r_l \rightarrow p_i$, then $p_1 \wedge \cdots \wedge p_{i-1} \wedge (r_1 \wedge \cdots \wedge r_l) \wedge p_{i+1} \wedge \cdots \wedge p_k \rightarrow q$.

For example, if $a \wedge b \rightarrow e$ and $c \wedge d \rightarrow a$, then $c \wedge d \wedge b \rightarrow e$.

2.3 Backward Chaining via Resolution

This is a search method used which combines sentences in the database in search of a given conclusion.

For example, suppose we wish to prove e. Suppose we know the following facts:

- $\star$ $a \wedge b \rightarrow e$
- $\star$ $c \wedge d \rightarrow a$
- $\star$ $T \rightarrow b$
- $\star$ $T \rightarrow c \wedge d$

We then aim for a contradiction, beginning with $e \rightarrow F$.

$$\begin{aligned} e &\rightarrow F && \text{(assumption)} \\ a \wedge b &\rightarrow F && (a \wedge b \rightarrow e) \\ c \wedge d \wedge b &\rightarrow F && (c \wedge d \rightarrow a) \\ T \wedge b &\rightarrow F && (T \rightarrow c \wedge d) \\ T \wedge T &\rightarrow F && (T \rightarrow b) \\ T &\rightarrow F \end{aligned}$$

This proves e.

2.4 Unification

Unification is involved in the combination of sentences when performing backward chaining via resolution. It is best explained via example.

Suppose we have the following database:

- $\star$ $father(p, q) \rightarrow child(q, p)$
- $\star$ $T \rightarrow father(John, Alice)$

We wish to prove the query $child(Alice, John)$ with backward chaining. We create a binding list σ which binds Alice to q and John to p. We then resolve $child(q, p) \rightarrow F$ with the first rule in the above database, yielding $father(p, q) \rightarrow F$. We would reach the desired contradiction if we could then use $T \rightarrow father(John, Alice)$. This would require that John was bound to p and Alice to q. Checking σ, this is indeed the case. Hence, $child(Alice, John)$ is true.

3 Algorithm

Find below the algorithm used to search a database written in pseudocode.

```
1: procedure ResolveQuery(q)
2:     L ← q                                              ▷ L is a list
3:     while L is not empty do
4:         p ← L[0]
5:         if p = T → F then return success
6:         p_1, ··· , p_n ← resolve p with other queries in L or in the database
7:         Append p_i to L for every i
8:     return failure                                     ▷ L is empty, so the query could not be proven
```

4 Implementation

4.1 Setup

The automated reasoning system was written in `JavaScript`. The demo was written using the `Meteor` framework on the `node.js` runtime. The source code can be found at `http://github.com/royhowie/csc545/` under the `project3/app` directory.

4.2 Rules & Facts

Note that the system is capable of reasoning about any set of coherent rules and facts. Indeed, the demo app provides an interface for changing the rules and facts in the database.

By default, it is loaded with the following rules and facts:

4.2.1 Rules

- $\star$ $father(p, q) \rightarrow child(q, p)$
- $\star$ $father(r, q) \wedge married(r, p) \rightarrow child(q, p)$

- $lawyer(p) \rightarrow rich(p)$
- $old(p) \wedge wise(p) \rightarrow sage(p)$
- $rich(p) \rightarrow old(p)$
- $sage(p) \rightarrow clever(p)$

4.2.2 Facts

- $T \rightarrow father(John, Alice)$
- $T \rightarrow lawyer(John)$
- $T \rightarrow married(John, Mary)$
- $T \rightarrow wise(John)$

5 Demo

A running version of the demo can currently be seen at `162.243.111.226:4044`.

6 Conclusion

After many hours of toiling away, a simple automated reasoning system (SARS) was born. Currently capable of "understanding" an arbitrary set of rules and facts presented in Horn form, SARS is able to respond to queries in a quasi-intelligent manner.

Originally, SARS was very dumb and did not understand how to bind variables. For example, if told $lawyer(p) \rightarrow rich(p)$ and $T \rightarrow laywer(John)$, it would consider both $rich(John)$ and $rich(Alice)$ to be true statements, despite the lack of evidence needed to prove the latter. However, in a moment of profundity, the author was able to teach SARS to understand single variables. Suddenly, the world—or at least that section which consists only of antecedents and consequents with single-argument atoms—was available to SARS.

However, intelligence is truly relative. SARS was still confused by multiple-argument atoms. To the casual passerby, the fact that SARS was confused by $father(p, q) \rightarrow child(q, p)$ was indeed damning. No one would consider such a machine intelligent! Again, after much struggle, the author was able to teach SARS to understand such statements. But SARS still had much to learn.

Indeed, SARS remained puzzled by statements such as $father(r, q) \wedge married(r, p) \rightarrow child(q, p)$ and was unable to understand that $father(r, q)$ is not much different from $father(p, q)$. SARS was unable to reconcile $father(p, q) \rightarrow child(q, p)$ with the aforementioned rule. Once more, the author managed to teach SARS to understand that variables live within the context of a statement. Just because $T \rightarrow father(John, Alice)$ does not necessarily mean the binding list must be $\sigma = \{\, q : John,\ p : Alice \,\}$. In truth, it is much more important that John is the first argument of $father$ and Alice the second. SARS learned variable positioning.

And such is the current circumstance of SARS. And although SARS has come far in its wordly understanding, it has a lot to learn before it's able to to convince anyone of its intelligence.

On a more serious note, SARS was a much larger endeavor than I expected. I quickly learned that data structures deemed useful at the outset either did not carry enough information or did not hold information in an ideal format to solve theorems. Indeed, upon further thought or inspection, I often realized subroutines intended to aid the algorithm provided in Section 3 missed important edge cases or approach the problem inefficiently. In hindsight, the algorithm used to search the database when resolving queries could be made much more efficient. SARS could also use a much better database representation of atoms and their respective arguments, which would have made solving the problem mentioned in paragraph 4 of this section much easier.