A4a - Logical Agent using Resolution Theorem Proving

CS 4300
Fall 2016
Leland Stenquist - 2,3,6
Ryan Keepers - 1,5,5

1. Introduction

This is a quick overview on the validation and data from the Resolution Theorem Prover. The intent is not to produce significant analysis of the algorithm. Rather, it is to verify correct operation of the function, and to lightly investigate its overall behavior and algorithmic complexity.

2. Method

The goal of A4a is to create a program, called CS3400_RTP, that executes the PL-Resolution algorithm. The PL-Resolution algorithm takes as input a knowledge base and a theory. Ours also adds a list of the variables. Using PL-Resolution the theory is tested against the knowledge base and hopefully proved by contradiction. This is done by adding the opposite of the theory to the Knowledge base. The algorithm then resolves the knowledge base. This means it looks for two clauses that have opposite literals. If it finds one it deletes the literal and then ORs the clauses together adding results, if it is unique, to the knowledge base. The algorithm ends when one of the clauses returns the empty set or there are no new, unique clauses being created. If the algorithm finds a clause that is the empty set, the theory is considered proved, else it is considered false.

Our function uses multiple helper methods:

1. **CS4300_RESOLVE**: This is described above. It performs the resolve functionality of the algorithm. We also have three helper functions
2. **CS4300_remove_totalogy**: Removes any tautologies from a cnf structure

3. **CS3400_cnf_union**: Unions two of the structures used to represent a cnf
4. **CS3400_cnf_contains**: Checks to see if a cnf structure contains a clause
5. **CS3400_cnf_print**: Prints out the contents of a cnf structure
6. **CS3400_cnf_unique**: Get rid of duplicates in a clause
7. **CS4300_resolve_tester**: A test for resolve
8. **CS4300_rtp_tester**: A test for RTP
9. **CS4300_NEG_THM**: negate a disjunctive clause

Once the PL-Resolution algorithm has proved a theory it returns an empty array. If it fails we return the knowledge base.

3. Verification of Program

Two methods of verification were used for this version of PL-Resolution. The first was an algorithm written out by hand. The second is by running programmatic testing.

Hand Written Verification

<table>
<tr><th colspan="2">PL-Resolution Verification</th></tr>
<tr><td colspan="2">KB(1).clauses = [-1 2 3 4]
KB(2).clauses = [-2]
KB(3).clauses = [-3]
KB(4).clauses = [1]

thm: 4</td></tr>
<tr><th>Human</th><th>Algorithm</th></tr>
<tr><td>Clauses:
1. [-1 2 3 4]
2. [-2]
3. [-3]
4. [1]
5 .[-4]

resolve(1, 2) = [[-1 3 4]]
resolve(1, 3) = [[-1 2 4]]
resolve(1, 4) = [[2 3 4]]
resolve(1, 5) = [[-1 2 3]]
resolve(2, 3) = []</td><td>
Clauses(1).clauses = [-1 2 3 4]
Clauses(2).clauses = [-2]
Clauses(3).clauses = [-3]
Clauses(4).clauses = [1]
Clauses(5).clauses = [-4]

resolve(1, 2) = [[-1 3 4]]
resolve(1, 3) = [[-1 2 4]]
resolve(1, 4) = [[2 3 4]]
resolve(1, 5) = [[-1 2 3]]
resolve(2, 3) = []</td></tr>
</table>

resolve(2, 4) = [] resolve(2, 5) = [] resolve(3, 4) = [] resolve(3, 5) = [] resolve(4, 5) = [] **New:** 1. [-1 3 4] 2. [-1 2 4] 3. [2 3 4] 4. [-1 2 3]	resolve(2, 4) = [] resolve(2, 5) = [] resolve(3, 4) = [] resolve(3, 5) = [] resolve(4, 5) = [] **New**(1).clauses = [-1 3 4] **New**(2).clauses = [-1 2 4] **New**(3).clauses = [2 3 4] **New**(4).clauses = [-1 2 3]
Clauses: 5. ... 6. [-1 3 4] 7. [-1 2 4] 8. [2 3 4] 9. [-1 2 3] ... resolve(1, 6-9) = [] resolve(2, 6-9) = [],[[-1 4]],[[3 4]],[[-1 3]] resolve(3, 6-9) = [[-1 4]],[],[[2 4]],[[-1 2]] resolve(4, 6-9) = [[3 4]],[[2 4]],[],[[2 3]] resolve(5, 6-9) = [[-1 3]],[[-1 2]],[[2 3]], [] resolve(6-9,6-9) = [] **New:** 5. … 6. [-1 4] 7. [3 4] 8. [-1 3] 9. [2 4] 10. [-1 2] 11. [2 3]	 **Clauses**(6).clauses = [-1 3 4] **Clauses**(7).clauses = [-1 2 4] **Clauses**(8).clauses = [2 3 4] **Clauses**(9).clauses = [-1 2 3] ... resolve(1, 6-9) = [] resolve(2, 6-9) = [],[[-1 4]],[[3 4]],[[-1 3]] resolve(3, 6-9) = [[-1 4]],[],[[2 4]],[[-1 2]] resolve(4, 6-9) = [[3 4]],[[2 4]],[],[[2 3]] resolve(5, 6-9) = [[-1 3]],[[-1 2]],[[2 3]], [] resolve(6-9,6-9) = [] **New**(5).clauses =[-1 4] **New**(6).clauses =[3 4] **New**(7).clauses = [-1 3] **New**(8).clauses =[2 4] **New**(9).clauses =[-1 2] **New**(10).clauses = [2 3]
Clauses: ... 10. [-1 4] 11. [3 4] 12. [-1 3] 13. [2 4] 14. [-1 2] 15. [2 3] ... resolve(1, 10-15) = []	 **Clauses**(10).clauses = [-1 4] **Clauses**(11).clauses = [3 4] **Clauses**(12).clauses = [-1 3] **Clauses**(13).clauses = [2 4] **Clauses**(14).clauses = [-1 2] **Clauses**(15).clauses = [2 3] ... resolve(1, 10-15) = []

resolve(2, 10-15) = [],[],[],[[4]],[[-1]],[[3]] resolve(3, 10-15) = [],[[4]],[[-1]],[],[],[[2]] resolve(4, 10-15) = [[4]],[],[[3]],[],[[2]],[] resolve(5, 10-15) = [[-1]],[[3]],[],[[2]],[],[] resolve(10-15, 10-15) = [] **New:** 11. … 12. [-1] 13. [2] 14. [3] 15. [4]	resolve(2, 10-15) = [],[],[],[[4]],[[-1]],[[3]] resolve(3, 10-15) = [],[[4]],[[-1]],[],[],[[2]] resolve(4, 10-15) = [[4]],[],[[3]],[],[[2]],[] resolve(5, 10-15) = [[-1]],[[3]],[],[[2]],[],[] resolve(10-15, 10-15) = [] **New**(12).clauses = [-1] **New**(13).clauses = [2] **New**(14).clauses = [3] **New**(15).clauses = [4]
Clauses: ... 16. [-1] 17. [2] 18. [3] 19. [4] ... resolve(1, 16-19) = [] resolve(2, 16-19) = [],[[]],[],[] resolve(3, 16-19) = [],[],[[]],[] resolve(4, 16-19) = [[]],[],[],[] resolve(5, 16-19) = [],[],[],[[]] resolve(16-19, 16-19) = [] The empty set was found in resolvents so we return []. We proved our theorum.	 **Clauses**(16).clauses = [-1] **Clauses**(17).clauses = [2] **Clauses**(18).clauses = [3] **Clauses**(19).clauses = [4] ... resolve(1, 16-19) = [] resolve(2, 16-19) = [],[[]],[],[] resolve(3, 16-19) = [],[],[[]],[] resolve(4, 16-19) = [[]],[],[],[] resolve(5, 16-19) = [],[],[],[[]] resolve(16-19, 16-19) = [] The empty set was found in resolvents so we return []. We proved our theorum.

Programmatic Verification

Solutions were determined by visual analysis. These solutions were recorded in testing functions. The algorithm was then run against these solutions printing out a report of correctness.

The following tests are provided via handin:

1. CS4300_resolve_tester()
2. CS4300_rtp_tester()

4. Data and Analysis

Table 1.

KB = [-1 2 3 4],[1],[-2],[-3]	Thm = [4]
Sentences in KB	4
Clauses produced	15
Loop count in RTP	4
Count of RESOLVE calls	338

Table 2.

KB = [-1 2 3 4],[1],[-2],[-3]	Thm = [-4]
Sentences in KB	4
Clauses produced	7
Loop count in RTP	3
Count of RESOLVE calls	186

Table 3.

KB = [-1 2 3 4],[1],[-2],[-3]	Thm = [2 -4]
Sentences in KB	4
Clauses produced	7
Loop count in RTP	3
Count of RESOLVE calls	186

Table 4.

KB = [-1 2 3 4],[1],[-2],[-3]	Thm = [-2 4]
Sentences in KB	4
Clauses produced	2

Loop count in RTP	1
Count of RESOLVE calls	9

Table 5.

KB = [1 2 3 4 5 6]	Thm = [1 2 3 4 5 6]
Sentences in KB	1
Clauses produced	68
Loop count in RTP	6
Count of RESOLVE calls	7252

5. Interpretation

The most important consideration for the cost of the algorithm seems to be the number of loops which must occur in order to resolve the theorem. As seen in tables 1 and 2, the difference of a loop matters significantly in the count of calls to the Resolve function. Through various tests we noted that the earliest loops showed the greatest proportional increase in clause counts and therefore resolve checks, often increasing the resolve count from the last loop by a factor of 4 or 5 within each of the first three loops. Therefore, the worst case scenario occurs when a single clause of n literals must be whittled down by 1 literal in each loop before it returns the null set.

6. Critique

This algorithm is not fast. While using a knowledge base for your agent may create a more “intelligent AI,” it will not create an computationally efficient AI.

7. Log

Ryan : 7 hrs.
Sections 1, 4, 5
Leland : 9 hrs
Sections 2, 3, 6