Labb 2

19 november 2020

Philip Salqvist

phisal@kth.se
CINTE
1992-04-17

Innehåll

1 Algorithm

The proof validating algorithm begins by reading from the input file, and assigns premises, the goal and proof to the lists *Prems*, *Goal* and *Proof*. Thereafter it validates the proof starting with the first line and continues to do so line by line. The algorithm will terminate when all of the lines in the Proof have been validated and if the last line of the proof is equal to the goal. To ensure this a base case is stated. In the base case the list with proofs is empty and *Goal* will unify with *Goal* if they are equal:

```
proof_check(Prems, Goal, [], [[_, Goal, _]|Valid_list]).
```

The algorithm will continue by trying to match a row in the proof with an existing rule stated in the script. If a corresponding rule is found and validated, we will recursively traverse in the list of proofs while simultaneously adding the validated rule to the list containing validated rows, named *Valid_list*:

```
proof_check(Prems, Goal, [Proof_h|Proof_t], Valid_list) :-
        rule(Prems, Proof_h, Valid_list),
        proof_check(Prems, Goal, Proof_t, [Proof_h|Valid_list]).
```

If the algorithm fails to find a matching rule however, we can draw the conclusion that the current row is an assumtion, hence a box has been opened. In this case, *box_check* will run to verify if the box is valid. When the algorithm is done verifying the box, we will run *proof_check* recursively on the remaining list of proofs, while simultaneously adding the valid box to the list containing verified elements:

```
proof_check(Prems, Goal, [[[Row_nr, A, assumption]|Box_t]|Proof_t], Valid_list) :-
        check_box(Prems, Goal, Box_t, [[Row_nr, A, assumption]|Valid_list]),
        proof_check(Prems, Goal, Proof_t, [[[Row_nr, A, assumption]|Box_t]|Valid_list])
```

Inside the box, two scenarios can occur. The algorithm can either match rows to corresponding rules and verify the rules as previously explained above, or we can encounter another box inside the box currently being verified. For the latter, the inner box will be verified first until all rows are checked, and we will then continue to verify the outer box until that reaches it's base case and will thus terminate:

```
check_box(Prems, Goal, [], Valid_list).

check_box(Prems, Goal, [[[Row_nr, A, assumption]|Box_t]|Proof_t], Valid_list) :-
        check_box(Prems, Goal, Box_t, [[Row_nr, A, assumption]|Valid_list]),
        check_box(Prems, Goal, Proof_t, [[[Row_nr, A, assumption]|Box_t]|Valid_list]).

check_box(Prems, Goal, [Proof_h|Proof_t], Valid_list) :-
        rule(Prems, Proof_h, Valid_list),
        check_box(Prems, Goal, Proof_t, [Proof_h|Valid_list]).
```

When the predicate rules run, the algorithm will match the current row to a corresponding stated rule. In this case three different type of rules have been stated. If the row is a premise, the algorithm will check if the element in the list with index 1 is a member of our list of premises. If the row is derived form a rule that references previous rows, the algorithm will see if these rows are members of *Valid_list*. In the third case, the row

is derived from a rule referencing a closed box. If this is the case the algorithm will run the predicate *get_box* which returns the requested box. It will then call the predicate *get_row* to get the first row of the box, and then to get the last row of the box. The algorithm will then check if the first row of the box is the assumption that we are looking for, and if the last row is the conclusion that we are looking for.

2 Result

All of the included tests that are valid are proven to be valid by the algorithm. Regarding the invalid tests, all of the tests are proven to be invalid except two. The tests invalid20 and invalid28 are accepted by the program even though they are invalid. This seems to be due to the fact that the predicate *get_rows* is searching for the requested rule by traversing through the list of verified rows and boxes, but doesn't really consider which box it is searching through. Thus the predicate will return the requested row, even if the row is found in a box that isn't relevant to the current rule being examined.

3 Table of predicates

Predicate	True	False
proof_check(Prems, Goal, Proof).	The entire proof is valid.	If not, false.
proof_check(Prems, Goal, [], [[_, Goal, _]|Valid_list]).	When the list of proofs is empty. And, when the second element of the row added latest to Valid_list matches with Goal.	If not, false.
proof_check(Prems, Goal, [Proof_h|Proof_t], Valid_list)	When the head of the proof list has been verified and the remaining part of the proof has been verified recursively.	If not, false.
proof_check(Prems, Goal, [[[Row_nr, A, assumption]|Box_t]|Proof_t], Valid_list)	The current row is an assumption, hence a box has been opened. When this box is verified and the remaining part of the proof is verified, this predicate is true.	If not, false.
check_box(Prems, Goal, [], Valid_list).	When the list of rows in the box is empty.	If not, false.
check_box(Prems, Goal, [[[Row_nr, A, assumption]|Box_t]|Proof_t], Valid_list)	The current row inside the box is an assumption, a box inside the box has been opened. When all of the rows in the inner box is verified, and the outer box is verified, this predicate is true.	If not, false.
check_box(Prems, Goal, [Proof_h|Proof_t], Valid_list)	When the current row is verified and the remaining part of the box is verified.	If not, false.
get_row(Row_nr, [Box_h|Remaining_valid], [Row_nr, A, Rule])	When the requested row is a member of the list with verified rows and boxes head. If not we recursively see if it is a member of the head in the remaining part of the list. If it is not found, Prolog will backtrack, although backtracking is stopped by a cut.	If the row is not found in Valid_list.

Predicate	True	False
get_box(Row_nr, [Box_h|Remaining_valid], Box_h)	If our requested row is a member of Box_h. If not, we recursively check if it is a member of Remaining_validś head.	If not, false.
rule(Prems, [Row_nr, A, premise], Valid_list)	If the second element in the row is a member of the list containing premises.	If not, false.
rule	For the remaining rules, if the row is derived form a rule that references previous rows, the algorithm will see if these rows are members of Valid_list. The row could also be derived from a rule referencing a box. In this case, the algorithm will try to get the first and last row of the box. It will go on to check if the first row is equal to the assumption and if the last row is equal to the conclusion specific to the rule that is applied in the current row that is being examined.	If not, false.

4 Appendix A

```
verify(InputFileName) :-
        see(InputFileName),
        read(Prems), read(Goal), read(Proof),
        seen,
        proof_check(Prems, Goal, Proof).

proof_check(Prems, Goal, Proof) :-
  proof_check(Prems, Goal, Proof, []).

%basecase of recursion where Proof_t = [] and Goal can unify with Goal.
%If Goal and Goal have different values, they will not unify and algorithm will answer no.

proof_check(Prems, Goal, [], [[_, Goal, _]|Valid_list]).


proof_check(Prems, Goal, [Proof_h|Proof_t], Valid_list) :-
        rule(Prems, Proof_h, Valid_list),
        proof_check(Prems, Goal, Proof_t, [Proof_h|Valid_list]).

%boxes
%If the row does not match any of our rules, the other option is that
%the row is an assumption and we have opened a box.

proof_check(Prems, Goal, [[[Row_nr, A, assumption]|Box_t]|Proof_t], Valid_list) :-

%run check_box while adding the assumption to the valid_list

        check_box(Prems, Goal, Box_t, [[Row_nr, A, assumption]|Valid_list]),
        proof_check(Prems, Goal, Proof_t, [[[Row_nr, A, assumption]|Box_t]|Valid_list]).

%base case for a box

check_box(Prems, Goal, [], Valid_list).

check_box(Prems, Goal, [[[Row_nr, A, assumption]|Box_t]|Proof_t], Valid_list) :-
        check_box(Prems, Goal, Box_t, [[Row_nr, A, assumption]|Valid_list]),
        check_box(Prems, Goal, Proof_t, [[[Row_nr, A, assumption]|Box_t]|Valid_list]).

%checking rules inside a box

check_box(Prems, Goal, [Proof_h|Proof_t], Valid_list) :-
        rule(Prems, Proof_h, Valid_list),
        check_box(Prems, Goal, Proof_t, [Proof_h|Valid_list]).

%checking if a row belongs to the previously added element in Valid_list.
%If it does, we will return that row.
%cut in end of statement to prevent backtracking loop.

get_row(Row_nr, [Box_h|Remaining_valid], [Row_nr, A, Rule]) :-
        member([Row_nr, A, Rule], Box_h),
  !.

get_row(Row_nr, [Box_h|Remaining_valid], [Row_nr, A, Rule]) :-
  get_row(Row_nr, Ramaining_valid, [Row_nr, A, Rule]).

%Find box in the head of
%Valid_list based on Row_nr and return that box

get_box(Row_nr, [Box_h|Remaining_valid], Box_h) :-
  member([Row_nr, _, _], Box_h).

%If we can't find it in the head of Valid_list
%search in the tail of Valid_list.

get_box(Row_nr, [Box_h|Remaining_valid], Box) :-
  get_box(Row_nr, Remaining_valid, Box).



%rules
```

```
rule(Prems, [Row_nr, A, premise], Valid_list) :-
        member(A, Prems).

rule(Prems, [Row_nr, A, impel(X,Y)], Valid_list) :-
        member([X, Premise, _], Valid_list),
        member([Y, imp(Premise, A), _], Valid_list).

rule(Prems, [Row_nr, A, copy(X)], Valid_list) :-
  member([X, A, _], Valid_list).

rule(Prems, [Row_nr, A, andel1(X)], Valid_list) :-
  member([X, and(A,_), _], Valid_list).

rule(Prems, [Row_nr, B, andel2(X)], Valid_list) :-
  member([X, and(_,B), _], Valid_list).

rule(Prems, [Row_nr, or(A,_), orint1(X)], Valid_list) :-
  member([X, A, _], Valid_list).

rule(Prems, [Row_nr, or(_,B), orint2(X)], Valid_list) :-
  member([X, B, _], Valid_list).

rule(Prems, [Row_nr, B, impel(X, Y)], Valid_list) :-
  member([X, A, _], Valid_list),
  member([Y, imp(A, B), _], Valid_list).

rule(Prems, [Row_nr, cont, negel(X, Y)], Valid_list) :-
  member([X, A, _], Valid_list),
  member([Y, neg(A), _], Valid_list).

rule(Prems, [Row_nr, A, contel(X)], Valid_list) :-
  member([X, cont, _], Valid_list).

rule(Prems, [Row_nr, neg(neg(A)), negnegint(X)], Valid_list) :-
  member([X, A, _], Valid_list).

rule(Prems, [Row_nr, A, negnegel(X)], Valid_list) :-
  member([X, neg(neg(A)), _], Valid_list).

rule(Prems, [Row_nr, neg(A), mt(X, Y)], Valid_list) :-
  member([X, imp(A, B), _], Valid_list),
  member([Y, neg(B), _], Valid_list).


rule(Prems, [Row_nr, imp(A, B), impint(X, Y)], Valid_list) :-
  get_row(X, Valid_list, First_row),
  get_row(Y, Valid_list, Last_row),
  [X, A, assumption] = First_row,
  [Y, B, _] = Last_row.


rule(Prems, [Row_nr, or(A, neg(A)), lem], Valid_list).

rule(Prems, [Row_nr, and(A, B), andint(X, Y)], Valid_list) :-
  member([X, A, _], Valid_list),
  member([Y, B, _], Valid_list).

rule(Prems, [Row_nr, neg(A), negint(X,Y)], Valid_list) :-
        get_row(X, Valid_list, First_row),
  get_row(Y, Valid_list, Last_row),
        [X, A, assumption] = First_row,
        [Y, cont, _] = Last_row.

rule(Prems, [Row_nr, A, orel(X,Y,Z,U,V)], Valid_list) :-
  get_box(Y, Valid_list, Box_1),
  get_row(Y, Box_1, First_row_1),
  get_row(Z, Box_1, Last_row_1),
  get_box(U, Valid_list, Box_2),
  get_row(U, Box_2, First_row_2),
  get_row(V, Box_2, Last_row_2),
  member([X, or(_Y, _U), _], Valid_list),
  [Y, _Y, assumption] = First_row_1,
```

```
    [U, _U, assumption] = First_row_2,
    [Z, A, _] = Last_row_1,
    [V, A, _] = Last_row_2.

rule(Prems, [Row_nr, A, pbc(X,Y)], Valid_list) :-
        get_box(X, Valid_list, Box),
        member([X, neg(A), assumption], Box),
        member([Y, cont, _], Box).
```

5 Appendix B

```
%valid

[imp(p,imp(q,r))].

imp(and(p,q),r).

[
[1, imp(p,imp(q,r)), premise],
[
[2, and(p,q), assumption],
[3, p, andel1(2)],
[4, imp(q,r), impel(3,1)],
[5, q, andel2(2)],
[6, r, impel(5,4)]
],
[7, imp(and(p,q),r), impint(2,6)]
].

%invalid

[imp(p,imp(q,r))].

imp(and(p,q),r).

[
[1, imp(p,imp(q,r)), premise],
[
[2, and(p,q), assumption],
[3, p, andel1(2)],
[4, imp(q,r), impel(3,1)],
[5, q, andel2(2)]
],
[6, imp(and(p,q),r), impint(2,6)]
].
```