Resolution in Solving Graph Problems

Kailiang Ji

Resolution in Solving Graph Problems

Kailiang Ji*

INRIA & Paris Diderot
23 avenue d'Italie, CS 81321, 75214 Paris Cedex 13, France.
kailiang.ji@inria.fr

Abstract. Resolution is a proof-search method for proving unsatisfiability problems. Various refinements have been proposed to improve the efficiency of this method. However, when we try to prove some graph properties, it seems that none of the refinements have an efficiency comparable with traditional graph traversal algorithms. In this paper we propose a way of encoding some graph problems as resolution. We define a selection function and a new subsumption rule to avoid redundancies while solving such problems.

Keywords: Resolution, Graph Problem, Selection Function, Subsumption

1 Introduction

Since the introduction of Resolution [13], many refinements have been proposed to increase the efficiency of this method, by avoiding redundancies. A first refinement, *Hyper-resolution*, has been introduced by Robinson himself the same year as Resolution [12]. More recently *Ordered resolution* [11, 14], *(Polarized) resolution modulo* (PRM) [5, 4], and finally *Ordered polarized resolution modulo* (OPRM) [1] introduced more restrictions. However, as we shall see, these kind of refinements are still redundant.

In order to address the question of the redundancy of proof search methods, we encode graph problems, e.g. accessibility or cycle detection, as Resolution problem, and we compare two ways to solve these problems: by using a proof-search method and by a direct graph traversal algorithm. If the proof-search method simulates graph traversal step by step, and in particular never visits twice the same part of the graph, we can say that it avoids redundancies. Otherwise, this helps us analyze and eliminate the redundancies of the method, by analysing why the method visits twice the same part of the graph.

The two graph problems can be expressed by predicate formulae with class variables (monadic second-order logic)[3, 6]. For example, the cycle detection problem can be expressed as:

$$\exists Y(s_1 \in Y \wedge \forall x(x \in Y \Rightarrow \exists x'(\mathbf{edge}(x, x') \wedge x' \in Y))).$$

* This work is supported by the ANR-NSFC project LOCALI (NSFC 61161130530 and ANR 11 IS02 002 01)

One can use the method of reducing formulae in finite domain predicate logic to the effectively propositional (EPR)[10] case, for proving the problems automatically. For example, replace the sub-formula $\forall xA$ by $A(s_1/x) \wedge A(s_2/x) \wedge \cdots \wedge A(s_n/x)$, and $\exists xA$ by $A(s_1/x) \vee A(s_2/x) \vee \cdots \vee A(s_n/x)$, in which $s_1, \ldots, s_n$ are the constants for all the vertices of a graph. By adding the theory of the graph as a set of rewrite rules[6], these problems can be proved by standard theorem provers, such as IPROVER[9]. If the graph is a Kripke structure, we can use model checking tools to solve the problems above, by expressing them in temporal logic [2]. Besides, Büchi automata [8] can be used to solve the cycle detection problem. In this paper we give a propositional encoding of the two problems above. To reduce the search space and make the resolution work faster, we add a selection function and a new subsumption rule to eliminate the redundancies generated in the resolution procedure. This method works for encodings of several graph problems. Its generality still remains to be investigated.

The paper is organized as follows. In Section 2, we briefly present Polarized resolution modulo. In Section 3, we explain how graph problems are encoded. In Section 4, we show that Polarized resolution modulo is still redundant. To reduce the redundant, we introduce our new method by adding a selection function and a new subsumption rule, without losing completeness of the system. Section 5 discusses some related work. Finally in section 6, we give an implementation and conclude our work.

2 Polarized Resolution Modulo

In Polarized Resolution Modulo (see Figure 1), clauses are divided into two sets: one-way clauses (or theory clauses) and ordinary clauses. Each one-way clause has a selected literal and resolution is only permitted between two ordinary clauses, or a one-way clause and an ordinary clause, provided the resolved literal is the selected one (the one underlined later) in the one-way clause.

Resolution $\dfrac{P \vee C \qquad \neg Q \vee D}{\sigma(C \vee D)}\ \sigma = mgu(P, Q)$

Factoring $\dfrac{L \vee K \vee C}{\sigma(L, C)}\ \sigma = mgu(L, K)$

Ext.Narr. $\dfrac{P \vee C}{\sigma(D \vee C)}$ if $\underline{\neg Q} \vee D$ is a one-way clause of $\mathcal{R}$, $\sigma = mgu(P, Q)$

Ext.Narr. $\dfrac{\neg P \vee C}{\sigma(D \vee C)}$ if $\underline{Q} \vee D$ is a one-way clause of $\mathcal{R}$, $\sigma = mgu(P, Q)$

Fig. 1. Polarized Resolution Modulo

Proving the completeness of the rules in Figure 1 requires to prove a cut elimination lemma [5, 4] for Polarized Deduction Modulo, the deduction system

with a set of rewrite rules, containing for each one-way clause $\underline{\neg P} \vee C$ the rule $P \rightarrow^{-} C$ and for each one-way clause $\underline{P} \vee C$ the rule $P \rightarrow^{+} \neg C$.

Like in OPRM, in this paper we define a selection function to select literals in an ordinary clause which have the priority to be resolved and add the selection function to PRM.

Note that when applying a Resolution rule between an ordinary clause and a one-way clause, we are in fact using an Extended Narrowing rule on this ordinary clause. We write $\Gamma \mapsto_{\mathcal{R}} C$ if C can be derived from the set of clauses Γ by applying finitely many inference rules of PRM.

3 Basic Definitions

To express the problems, we consider a propositional language which contains two atomic propositions P_i and Q_i for each natural number. We denote a graph as $G = \langle V, E \rangle$, in which V is a set of vertices enumerated by natural numbers, E is a set of directed edges of the graph. The sequence of vertices $l = s_0, s_1, \ldots, s_k$ is a *walk* if and only if $\forall 0 \leqslant i < k$, $(s_i, s_{i+1}) \in E$. The walk l is *closed* if and only if $\exists 0 \leqslant j \leqslant k$ such that $s_k = s_j$. The walk l is *blocked* if and only if s_k has no successors. The method we proposed is inspired by graph traversal algorithms. In the following sections, we introduce some terminology inspired by graph traversal algorithms.

Definition 1 (Black and White Literals). *Let G be a graph and $\{s_1, \ldots, s_n\}$ be the set of all the vertices in G. For any $1 \leqslant i \leqslant n$, the literal P_i is called a* black *literal and the literal Q_i is called a* white *literal.*

Intuitively, the black literals denote the vertices that have already been visited, while the white literals denote the unvisited ones.

Definition 2 (Original Clause). *Let G be a graph and $\{s_1, \ldots, s_n\}$ be the set of all the vertices in G. For each graph traversal problem starting from s_i $(1 \leqslant i \leqslant n)$, we define the* original *clause* $\mathsf{ori}(s_i, G)$ *as*

$$P_i \vee Q_1 \vee \cdots \vee Q_n.$$

Definition 3 (Traversal Clause). *A clause with only white and black literals is called a* traversal clause.

Definition 4 (Success Clause). *Let C be a traversal clause, if there is no i, such that both P_i and Q_i are in C, then C is called a* success clause.

Among the three kinds of clauses, the original clause is related to the starting point of the graph traversal algorithm, the traversal clause is the current process of the travelling, and the success clause means that the solution is found and the traversal procedure can be finished. Obviously, the original clauses and success clauses are also traversal clauses.

4 Closed-Walk Detection

In this section, we present an algorithm to find out whether there exists a closed walk starting from a given vertex.

4.1 Encoding of closed-walk detection problem

For this encoding, we view the graph as a set of rewrite rules, and the initial situation is denoted by the original clause.

E-coloring rule Let G be a graph and $V = \{s_1, \ldots, s_n\}$ be the set of all the vertices in G. For each pair of vertices $\langle s_i, s_j \rangle$ in V, if there exists an edge from s_i to s_j, then we formalize this edge as a *E-coloring rewrite rule*

$$Q_i \hookrightarrow P_j.$$

Correspondingly, the one-way clause for the rewrite rule is $\underline{Q_i^{\perp}} \vee P_j$ (called *E-coloring clause*). The set of all the E-coloring clauses for graph G is denoted as $EC(G)$.

Resolution for closed walk detection Let G be a graph and s be a vertex of G, then the the problem of checking whether, starting from s, there exists a closed walk can be encoded as the set of clauses $\{\mathsf{ori}(s, G)\} \cup EC(G)$. By applying resolution rules among these clauses, a success clause can be derived, if and only if there exists a closed walk starting from s.

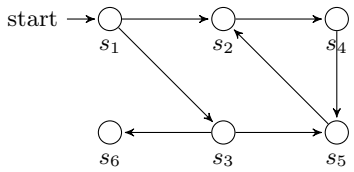


Fig. 2.

Example 1. Consider the graph in Figure 2, check whether there exists a closed walk starting from s_1. For this problem, the original clause is

$$P_1 \vee Q_1 \vee Q_2 \vee Q_3 \vee Q_4 \vee Q_5 \vee Q_6$$

and the set of E-coloring clauses for this graph are

$$\underline{Q_1^{\perp}} \vee P_2, \quad \underline{Q_1^{\perp}} \vee P_3, \quad \underline{Q_2^{\perp}} \vee P_4, \quad \underline{Q_3^{\perp}} \vee P_5, \quad \underline{Q_3^{\perp}} \vee P_6, \quad \underline{Q_4^{\perp}} \vee P_5, \quad \underline{Q_5^{\perp}} \vee P_2.$$

Resolution steps For the original clause, apply Resolution rule with E-coloring clause $\underline{Q_1^{\perp}} \vee P_2$, which yields

$$P_1 \vee P_2 \vee Q_2 \vee Q_3 \vee Q_4 \vee Q_5 \vee Q_6,$$

then apply Resolution rule with E-coloring clause $\underline{Q_2^{\perp}} \vee P_4$, which yield

$$P_1 \vee P_2 \vee P_4 \vee Q_3 \vee Q_4 \vee Q_5 \vee Q_6,$$

then apply Resolution rule with E-coloring clause $\underline{Q_4^{\perp}} \vee P_5$, which yields

$$P_1 \vee P_2 \vee P_4 \vee Q_3 \vee P_5 \vee Q_5 \vee Q_6,$$

then apply Resolution rule with E-coloring clause $\underline{Q_5^{\perp}} \vee P_2$, the generated clause

$$P_1 \vee P_2 \vee P_4 \vee Q_3 \vee P_5 \vee Q_6,$$

is a success clause, meaning that in Figure 2, there exists a closed walk starting from s_1.

Theorem 1. *Let G be a graph and s be a vertex in G. Starting from s, there exists a closed walk if and only if starting from $\{\mathsf{ori}(s, G)\} \cup EC(G)$, a success clause can be derived.*

5 Blocked-walk Detection

In this section, we present a method to check whether, starting from a vertex, there exists a blocked walk or not.

5.1 Encoding of blocked-walk detection problem

For this encoding stategy, the graph is viewed as a set of rewrite rules, which is different from the rules in the former section. The intial situation is denoted by the original clause.

A-coloring rule Let G be a graph and $V = \{s_1, \ldots, s_n\}$ be the set of vertices of G. For each vertex s_i in V, assume that starting from s_i, there are edges to $s_{i_1}, \ldots, s_{i_j}$, then we formalize the set of edges starting from s_i as an A-coloring rule

$$Q_i \hookrightarrow P_{i_1} \vee \cdots \vee P_{i_j}.$$

Correspondingly, the one-way clause for the rewrite rule is $\underline{Q_i^{\perp}} \vee P_{i_1} \vee \cdots \vee P_{i_j}$ (called A-coloring clause). The set of all the A-coloring clauses for graph G is denoted as $AC(G)$.

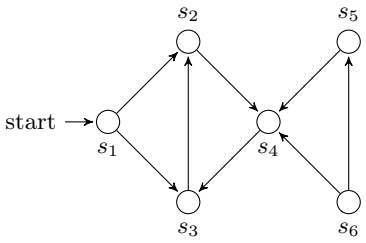


Fig. 3.

Resolution for blocked walk detection Let G be a graph and s be a vertex of G, then the the problem of checking that starting from s, whether there exists a blocked walk can be encoded as the set of clauses $\{\mathsf{ori}(s, G)\} \cup AC(G)$. By applying resolution rules among these clauses, a success clause can be derived, if and only if *there is no blocked walk* starting from s.

Example 2. Consider the graph in Figure 3 and the problem of whether there exists a blocked walk starting from s_1. For this problem, the original clause is

$$P_1 \vee Q_1 \vee Q_2 \vee Q_3 \vee Q_4 \vee Q_5 \vee Q_6$$

and the set of A-coloring clauses for this graph are

$$\underline{Q_1^{\perp}} \vee P_2 \vee P_3, \quad \underline{Q_2^{\perp}} \vee P_4, \quad \underline{Q_3^{\perp}} \vee P_2, \quad \underline{Q_4^{\perp}} \vee P_3, \quad \underline{Q_5^{\perp}} \vee P_4, \quad \underline{Q_6^{\perp}} \vee P_4.$$

Resolution steps For the original clause, apply Resolution rule with A-coloring clause $\underline{Q_1^{\perp}} \vee P_2 \vee P_3$, which yields

$$P_1 \vee P_2 \vee P_3 \vee Q_2 \vee Q_3 \vee Q_4 \vee Q_5 \vee Q_6,$$

then apply resolution rule with A-coloring clause $\underline{Q_2^{\perp}} \vee P_4$, which yields

$$P_1 \vee P_2 \vee P_3 \vee P_4 \vee Q_3 \vee Q_4 \vee Q_5 \vee Q_6,$$

then apply resolution rule with A-coloring clause $\underline{Q_3^{\perp}} \vee P_2$, which yields

$$P_1 \vee P_2 \vee P_3 \vee P_4 \vee Q_4 \vee Q_5 \vee Q_6,$$

then apply resolution rule with A-coloring clause $\underline{Q_4^{\perp}} \vee P_3$, and the generated clause

$$P_1 \vee P_2 \vee P_3 \vee P_4 \vee Q_5 \vee Q_6,$$

is a success clause, meaning that there is no blocked walk starting from s_1.

Theorem 2. *Let G be a graph and s_1 be a vertex of G. Starting from s_1, there is no blocked walk if and only if, starting from $\{\mathsf{ori}(s_1, G)\} \cup AC(G)$, a success clause can be derived.*

6 Simplification Rules

A drawback of the traditional automatic theorem proving methods is that they are only practical for graphs of relatively small size. In this section, the reason why the method is not as efficient as tranditional traversal methods is analyzed. To address the problems in our method, we designed some new strategies. Finally the completeness of the system with newly added rules is proved.

6.1 Selection Function

We define a selection function, which applies on a traversal clause and returns a set of literals that have priority when applying resolution rules. We show that the number of resolution steps strongly depend on the literals that are selected. More precisely, the number of literals that are selected will also affect the number of resolution steps.

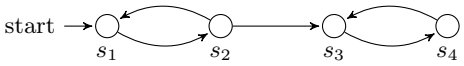


Fig. 4.

Example 3. For the graph in Figure 4, we prove the property:

starting from s_1, there exists a closed walk.

The original clause is:

$$P_1 \vee Q_1 \vee Q_2 \vee Q_3 \vee Q_4,$$

and the E-coloring clauses of the graph are

$$\underline{Q_1^{\perp}} \vee P_2, \quad \underline{Q_2^{\perp}} \vee P_1, \quad \underline{Q_2^{\perp}} \vee P_3, \quad \underline{Q_3^{\perp}} \vee P_4, \quad \underline{Q_4^{\perp}} \vee P_3.$$

Starting from the original clause, we can apply resolution as follows: First, apply resolution with E-coloring clause $\underline{Q_1^{\perp}} \vee P_2$, which yields

$$P_1 \vee P_2 \vee Q_2 \vee Q_3 \vee Q_4. \tag{1}$$

Then for (1), apply resolution with E-coloring clause $\underline{Q_2^{\perp}} \vee P_1$, which yields

$$P_1 \vee P_2 \vee Q_3 \vee Q_4. \tag{2}$$

The clause (2) is a success clause. However, from (1), if we apply resolution with another E-coloring clause instead, we will need more resolution steps to get a success clause.

As can be seen from Example 3, each time if there exists a pair of P_i and Q_i, and select Q_i to be resolved, we may have less resolution steps to get a success clause.

Definition 5 (Grey literals). *Let C be a traversal clause. For the pair of white literals and black literals $\langle Q_i, P_i\rangle$, if both Q_i and P_i are the members of C, then Q_i is called a* grey literal *of C. The set of grey literals of C is defined as follows:*

$$\mathsf{grey}(C) = \{Q_i \mid \textit{both } P_i \textit{ and } Q_i \textit{ are in } C\}$$

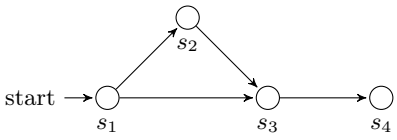


Fig. 5.

Example 4. For the graph in Figure 5, we prove the property:

starting from s_1, there is no blocked walk.

The original clause is :

$$P_1 \vee Q_1 \vee Q_2 \vee Q_3 \vee Q_4,$$

and the A-coloring clauses of the graph are:

$$\underline{Q_1^{\perp}} \vee P_2 \vee P_3, \quad \underline{Q_2^{\perp}} \vee P_3, \quad \underline{Q_3^{\perp}} \vee P_4$$

Resolution steps For the original clause, apply resolution with A-coloring clause $\underline{Q_1^{\perp}} \vee P_2 \vee P_3$, which yields

$$P_1 \vee P_2 \vee P_3 \vee Q_2 \vee Q_3 \vee Q_4. \tag{3}$$

Then for (3), we can apply resolution rules with A-coloring clauses $\underline{Q_2^{\perp}} \vee P_3$ and $\underline{Q_3^{\perp}} \vee P_4$, and two new traversal clauses are generated:

$$P_1 \vee P_2 \vee P_3 \vee Q_3 \vee Q_4, \tag{4}$$

$$P_1 \vee P_2 \vee P_3 \vee P_4 \vee Q_2 \vee Q_4. \tag{5}$$

Then for (4), apply resolution rule with A-coloring clause $\underline{Q_3^{\perp}} \vee P_4$, which yields

$$P_1 \vee P_2 \vee P_3 \vee P_4 \vee Q_4, \tag{6}$$

and for this clause, we cannot apply resolution rules any more. For (5), we can apply resolution rule with A-coloring clause $\underline{Q_2^{\perp}} \vee P_3$, and the clause generated is the same as (6). Obviously, the resolution steps for generating (5) and the steps started from (5) are redudant.

To avoid the redundant steps in Example 4, each time we select only one grey literal. So the selection function can be defined as follows.

Definition 6 (Selection function). *For any traversal clause C, the selection function δ is defined as:*

$$\delta(C) = \begin{cases} \mathsf{single}(\mathsf{grey}(C)), & \mathsf{grey}(C) \neq \emptyset \\ C, & \textit{Otherwise} \end{cases}$$

in which single *is a process to select only one literal from a set of literals.*

Notations The *Polarized resolution modulo with selection function* is written as PRM_δ. We write $\Gamma \rightarrow^{\delta}_{\mathcal{R}} C$ if the clause C can be derived from the set of clauses Γ in the system PRM_δ.

6.2 Elimination Rule

But, as we shall see, selecting literals, which is at the base of PRM, OR, OPRM, and this method are not sufficient as we also have to restrict the method at the level of clauses. In spite of several clause elimination procedures had been applied to the procedure of resolution method[7], none of them works efficient to our problem.

Example 5. For the graph in Figure 5, we prove the property:

starting from s_1, there exists a closed walk.

The original clause is :

$$P_1 \vee Q_1 \vee Q_2 \vee Q_3 \vee Q_4,$$

and the E-coloring clauses of the graph are:

$$\underline{Q_1^{\perp}} \vee P_2, \quad \underline{Q_1^{\perp}} \vee P_3, \quad \underline{Q_2^{\perp}} \vee P_3, \quad \underline{Q_3^{\perp}} \vee P_4$$

Resolution steps For the original clause, apply resolution rules with $\underline{Q_1^{\perp}} \vee P_2$ and $\underline{Q_1^{\perp}} \vee P_3$, two new traversal clauses are generated:

$$P_1 \vee P_2 \vee Q_2 \vee Q_3 \vee Q_4, \tag{7}$$

$$P_1 \vee P_3 \vee Q_2 \vee Q_3 \vee Q_4, \tag{8}$$

for (7), apply resolution rule with $\underline{Q_2^{\perp}} \vee P_3$, which yields

$$P_1 \vee P_2 \vee P_3 \vee Q_3 \vee Q_4, \tag{9}$$

then for (9), apply resolution rule with $\underline{Q_3^{\perp}} \vee P_4$, which yields

$$P_1 \vee P_2 \vee P_3 \vee P_4 \vee Q_4, \tag{10}$$

for (10), we cannot apply resolution rules any more. And in all the generated clauses, only the traversal clause (8) remains to be resolved. For (8), apply resolution rule with $\underline{Q_3} \vee \neg P_4$, the same E-coloring clause as for (9), thus in the implementation of proof search algorithm for PRM [1], the derivation will continue, while in fact the rest steps of the derivation are useless.

To avoid the redundant steps showed in Example 5, a new elimination rule is defined as follows.

Definition 7 (Path subsumption elimination rule(PSER)). *Let M be a set of A(E)-coloring clauses and C be a traversal clause. If we have $C, M \rightarrow^{\delta}_{\mathcal{R}} C_1$ and $C, M \rightarrow^{\delta}_{\mathcal{R}} C_2$, in which $\mathsf{grey}(C_1) = \mathsf{grey}(C_2)$, then the literals C_1 and C_2 can be replaced by each other.*

After each step of resolution, we try to apply PSER on the set of traversal clauses before applying other resolution rules. By PSER, the clause (8) in Example 5 is useless, thus can be deleted during the derivation.

Theorem 3 (Completeness). *PRM_{δ} with PSER is complete.*

7 Implementation

In this section, we talk about the issues during the implementation, and then present the evaluation data for some graphs.

7.1 How to deal with success clause

In normal proof search algorithms, the derivation will not stop until (i) an empty clause is derived, in this case the input set of clauses is unsatisfiable or (ii) no new clauses can be generated by applying resolution rules, in this case the input set of clauses is satisfiable. However, in the specific problems of this chapter, when a success clause is derived, the derivation should stop and report that a success clause can be derived, which is different from "Satisfiable" or "Unsatisfiable". To implement our method in automatic theorem provers, there may have two ways to deal with the success clauses:

- give a set of rewrite rules, when a success clause is derived, make sure that this clause can be rewritten into empty clause.
- take success clause as the same role of empty clause, in this case when a success clause is derived, the derivation stop and report the input set of clauses is unsatisfiable.

For the first case, one way is to introduce class variables and take the atomic propositions P_i and Q_i as binary predicates. Thus P_i is replaced by $P(s_i, Y)$ and Q_i is replaced by $Q(s_i, Y)$. Thus the success clause

$$P_1 \vee P_2 \vee \cdots \vee P_i \vee Q_{i+1} \vee \cdots \vee Q_k$$

is replaced by

$$P(s_1, Y) \vee P(s_2, Y) \vee \cdots \vee P(s_i, Y) \vee Q(s_{i+1}, Y) \vee \cdots \vee Q(s_k, Y).$$

The rewrite rules added are

1. $P(x, add(y, Z)) \hookrightarrow x = y^{\perp} \wedge P(x, Z)$
2. $Q(x, nil) \hookrightarrow \perp$
3. $Q(x, add(y, Z)) \hookrightarrow x = y \vee Q(x, Z)$
4. $x = x \hookrightarrow T$
5. for each two vertices s_i and s_j, if they are not the same vertex, then $s_i = s_j \hookrightarrow \perp$

This method is a variation of the theory defined in [6]. The main problem of this method is that, for any two different vertices in a graph, a rewrite rule should be added to the system to express teh non-equal properties.

For the second case, a procedure to check whether a clause is a success clause should be added to the loop-body of the program. For the position where to embed this procedure, a simple proof search algorithm is given as follows:

```
program main_loop
  initial
    original clause in U, A(E)-coloring clauses in P
  while U != empty
   c := select(U)
   U := U \ {c} (* remove c from U *)
   if c is an empty or a success clause, then return "Unsat"
   P := P + {c} (* add c to P *)
   U := U + generate(c,P)
  done
  return "Sat"
end.
```

where `select(U)` selects a clause from `U`, `grey(c)` is the set of grey literals in `c` and `generate(c,P)` produces all the clauses by applying an inference rule between `c` and a clause in `P`.

7.2 Embedding path subumption elimination rule into the proof-search algorithm

Normally, to run the path subsumption elimination rule, each time before applying resolution rules between the selected traversal clause in the passive set `U` and the coloring cluases in the active set `P`, we need to give a comparation between the selected clause and each traversal clause in `P`. To make it simple, before the loop part for the resolution steps, a new empty set `G` is given, and for the selected traversal clause in `U`, if the grey literal of the traversal clause are in `G`, then just add the clause to the active set, otherwise, add the grey literal to `G` and apply resolution between this clause and the coloring clauses.

Algorithm By adding path subsumption elimination rule into the algorithm above, the new algorithm is as follows:

```
program main_loop
  initial
    original clause in U, blackening clauses in P
    G is empty (* G is a set of sets of grey literals *)
  while U != empty
   c := select(U)
   U := U \ {c} (* remove c from U *)
   if c is an empty or a success clause, then return "Unsat"
   P := P + {c} (* add c to P *)
   g := grey(c)
   if g is not a member of S then
      G := G + {g}
      U := U + generate(c,P)
  done
  return "Sat"
end.
```

7.3 Experimental Evaluation

To implement the strategy in this chapter, the procedure of checking success clauses, the selection function, and the path subsumption elimination rule are embeded into iProver modulo [1], a resolution based automatic theorem prover. Then the two kinds of problems for some randomly generated graphs are solved using this prover.

Table 1. Closed Walk and Blocked Walk Detection

Graph				Result and Time			
Prop	**N(v)**	**N(e)**	**Num**	**Sat**	**Succ**	**PRM**	**PRM+P**
Closed Walk	1.0×10^3	1.0×10^3	100	95	5	25m40s	25m0s
	1.0×10^3	1.5×10^3	100	50	50	1h06m40s	1h02m46s
	1.0×10^3	2.0×10^3	100	23	77	1h09m44s	1h09m46s
Blocked Walk	1.0×10^3	1.0×10^3	100	100	0	7m04s	
	1.0×10^3	1.5×10^3	100	100	0	10m29s	
	1.0×10^3	2.0×10^3	100	100	0	17m48s	
	1.0×10^3	2.5×10^3	100	100	0	35m16s	
	1.0×10^3	3.0×10^3	100	100	0	1h06m28s	
	1.0×10^3	1.0×10^4	100	0	100	24h50m43s	

Table 1 is the data of running the graph examples. The experiments are implemented on Intel® Core™ i5-2400 CPU @ 3.10GHz × 4 with Linux. For the closed walk detection problem, from the running time of all the 100 examples of PRM+P is almost equal to PRM, some of which is occupied by running the

path sumption elimination rule. However, in some of the examples, when the path subsumption elimination rule is applied, it do saves a lot of time. For the blocked walk detection problem, as can be seen from the table, the running time increases while we have more edges in the graphs.

8 Conclusion and Future Work

In this paper, two graph problems, closed walk and blocked walk detection, are considered. To make it simple, we encoded the problems with propositional formulae, and the edge relationship are encoded as rewrite rules. To improve the efficiency of the implementation, a selection function and a new subsumption elimination rule are defined. At last, an implementation about solving these two problems is presented.

As the number of literals in the original clause is equal to the number of vertices in the graph, if the graph is large enough, the space resources during the implementation will be ran out. In spite of [15] had given the idea of introducing new atoms as abbreviations or 'definitions' for sub-formulae, this cannot be used directly to our case. In the future works, we will encode the vertices with Boolean vectors.

Safety and liveness are two important problems in model checking [2]. The safety property says that something "bad" will never happen and the liveness property says that something "good" will happen. To prove the safety of a system, we need to find a finite path to the "bad" thing or prove that all the accessible states are not "bad". This problem can be treated as a blocked-walk detection problem. For the liveness of a system, we need to find an infinite path such that all the states on the path are not "good" or on each infinite path, there exists a "good" state. This problem can be treated as a closed-walk detection problem. An infinite path (closed walk) is found out when a success clause is derived. Thus, the work in this paper can be used in automatically verfying temporal properties of a transition system.

Acknowledgements. I am grateful to Gilles Dowek, for his careful reading and comments.

References

1. Burel, G.: Embedding Deduction Modulo into a Prover. In: Dawar, A., Veith, H. (eds.) CSL 2010. LNCS, vol. 6247, pp. 155–169. Springer Berlin Heidelberg (2010)
2. Clarke, Jr., E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press, Cambridge, MA, USA (1999)
3. Courcelle, B.: The monadic second-order logic of graphs. i. recognizable sets of finite graphs. Information and computation 85(1), 12–75 (1990)
4. Dowek, G.: Polarized Resolution Modulo. In: Calude, C.S., Sassone, V. (eds.) TCS 2010. IFIP AICT, vol. 323, pp. 182–196. Springer Berlin Heidelberg (2010)

5. Dowek, G., Hardin, T., Kirchner, C.: Theorem Proving Modulo. Journal of Automated Reasoning 31, 33–72 (2003)
6. Dowek, G., Jiang, Y.: Axiomatizing Truth in a Finite Model (2013), `https://who.rocq.inria.fr/Gilles.Dowek/Publi/classes.pdf`, manuscript
7. Heule, M., Järvisalo, M., Biere, A.: Clause elimination procedures for cnf formulas. In: Logic for Programming, Artificial Intelligence, and Reasoning. pp. 357–371. Springer (2010)
8. Khoussainov, B., Nerode, A.: Automata theory and its applications, vol. 21. Springer Science & Business Media (2001)
9. Korovin, K.: iProver– An Instantiation-Based Theorem Prover for First-Order Logic (System Description). In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS, vol. 5195, pp. 292–298. Springer Berlin Heidelberg (2008)
10. Navarro-Pérez, J.A.: Encoding and Solving Problems in Effectively Propositional Logic. Ph.D. thesis, The University of Manchester (2007)
11. Reiter, R.: Two results on ordering for resolution with merging and linear format. Journal of the ACM (JACM) 18(4), 630–646 (1971)
12. Robinson, J.A.: Automatic deduction with hyper-resolution. Journal of Symbolic Logic 39(1), 189–190 (1974)
13. Robinson, J.A.: A machine-oriented logic based on the resolution principle. Journal of the ACM (JACM) 12(1), 23–41 (1965)
14. Slagle, J.R., Norton, L.M.: Experiment with an automatic theorem-prover having partial ordering inference rules. Communications of the ACM 16(11), 682–688 (1973)
15. Tseitin, G.S.: On the complexity of derivation in propositional calculus. In: Automation of reasoning, pp. 466–483. Springer (1983)

.1 Correctness of the Encoding

To prove that this kind of encoding suit for all closed walk detection problems, a proof of the theorem below is given.

Theorem 4. *Let G be a graph and s be a vertex in G. Starting from s, there exists a closed walk if and only if starting from $\{\mathsf{ori}(s, G)\} \cup EC(G)$, a success clause can be derived.*

Before proving this theorem, several notations and lemmas are needed, which will also be used in the following sections.

Notations Let C_1, C_2, C_3 be clauses, Γ be a set of clauses:

- if C_3 is generated by applying resolution between C_1 and C_2, then write the resolution step as $C_1 \xrightarrow{C_2} C_3$; if the resolution is based on a selection function δ, then the resolution step is written as $C_1 \xrightarrow{C_2}_{\delta} C_3$.
- if C_2 is generated by applying resolution between C_1 and a clause in Γ, then write the resolution step as $C_1 \xrightarrow{\Gamma} C_2$; if the resolution is based on a selection function δ, then the resolution step is written as $C_1 \xrightarrow{\Gamma}_{\delta} C_2$.
- if C_1 is generated by one step of resolution on some clauses in Γ, then write the resolution step as $\Gamma \longrightarrow \Gamma, C_1$; if the resolution is based on a selection function δ, then the resolution step is written as $\Gamma \longrightarrow_{\delta} \Gamma, C_1$.

Lemma 1. *For any two traversal clauses, we cannot apply resolution rules between them.*

Proof. All the literals in traversal clauses are positive. □

Lemma 2. *If resolution rules can be applied between a traversal clause and a coloring clause, then one and only one traversal clause can be derived.*

Proof. As all the literals in the traversal clause are positive and there is only one negative literal in the coloring clause, straightforwardly, only one traversal clause can be derived. □

Proposition 1. *Let M be a set of coloring clauses, $C_1, \ldots, C_n$ be traversal clauses and S be a success clause. If $M, C_1, \ldots, C_n \rightarrow S$, then there exists $1 \leqslant i \leqslant n$, such that $M, C_i \rightarrow S$, and the length of the later derivation is at most equal to the former one.*

Proof. By induction on the size of the derivation $M, C_1, \ldots, C_n \rightarrow S$.

- If S is a member of $C_1, \ldots, C_n$, then there exists the derivation $M, S \rightarrow S$ without applying any resolution rules.
- If S is not a member of $C_1, \ldots, C_n$, then in each step of the derivation, by Lemma 1, the resolution rules can only be applied between a traversal clause and a coloring clause. Assume the derivation is $M, C_1, \ldots, C_n \longrightarrow M, C_1, \ldots, C_n, C' \rightarrow S$, in which, by Lemma 2, C' is a traversal clause. Then for the derivation $M, C_1, \ldots, C_n, C' \rightarrow S$, by induction hypothesis, $M, C' \rightarrow S$ or there exists $1 \leqslant i \leqslant n$ such that $M, C_i \rightarrow S$, with the steps of the derivation at most equal to $M, C_1, \ldots, C_n, C' \rightarrow S$. If $M, C_i \rightarrow S$, then the steps of the derivation are less than $M, C_1, \ldots, C_n \rightarrow S$, thus this derivation is as needed. If $M, C' \rightarrow S$, then by Lemma 1, there exists C_j in $C_1, \ldots, C_n$, such that $C_j \xrightarrow{M} C'$, thus the derivation $M, C_j \rightarrow S$, with the derivation steps at most equal to $M, C_1, \ldots, C_n \rightarrow S$, is as needed.

□

Proposition 2. *Let M be a set of coloring clauses, C be a traversal clause, and S be a success clause. If $M, C \rightarrow S(\pi_1)$[1], then there exists a derivation path $C(C_0) \xrightarrow{M} C_1 \xrightarrow{M} C_2 \cdots \xrightarrow{M} C_n(S)$.*

Proof. By induction on the size of the derivation π_1.

- If C is a success clause, then the derivation path can be built directly.
- Otherwise, by Lemma 1, in each step of the derivation, the resolution rules can only be applied between a traversal clause and a coloring clause. Assume the derivation is $M, C \longrightarrow M, C, C' \rightarrow S$, then for the derivation $M, C, C' \rightarrow S$, by Proposition 1, there exists a derivation $M, C \rightarrow S(\pi_2)$[2] or $M, C' \rightarrow S$,

[1] we denote the derivation as π_1.
[2] we denote the derivation as π_2.

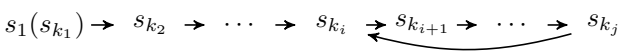
with the length less than π_1. For π_2, by induction hypothesis, there exists a derivation path $C(C_0) \xrightarrow{M} C_1 \cdots \xrightarrow{M} C_n(S)$, and this is just the derivation as needed. For $M, C' \to S$, by induction hypothesis, there exists a derivation path $C' \xrightarrow{M} C'_1 \cdots \xrightarrow{M} C'_m(S)$. As $C \xrightarrow{M} C'$, the derivation path $C \xrightarrow{M} C' \xrightarrow{M} C'_1 \cdots \xrightarrow{M} C'_m(S)$ is as needed.

□

Now it is ready to prove Theorem 4. The proof is as follows.

Proof of Theorem 4

Proof. – For the right direction, we assume that the path is



By the method of generating E-coloring clauses of a graph, there exist E-coloring clauses:

$$\underline{Q^{\perp}_{k_1}} \vee P_{k_2}, \underline{Q^{\perp}_{k_2}} \vee P_{k_3}, \ldots, \underline{Q^{\perp}_{k_{i-1}}} \vee P_{k_i}, \underline{Q^{\perp}_{k_i}} \vee P_{k_{i+1}}, \ldots, \underline{Q^{\perp}_{k_j}} \vee P_{k_i}.$$

Then starting from the original clause $C_1 = P_1 \vee Q_1 \vee \cdots \vee Q_n$, the derivation

$$C_1 \xrightarrow{D_1} C_2 \xrightarrow{D_2} \cdots C_{i-1} \xrightarrow{D_{i-1}} C_i \xrightarrow{D_i} \cdots C_j \xrightarrow{D_j} C_{j+1}$$

can be built, in which C_{j+1} is a success clause and for each $1 \leqslant m \leqslant j$, D_m is the E-coloring clause $\underline{Q^{\perp}_{k_m}} \vee P_{k_{m+1}}$.

– For the left direction, by Proposition 2, starting from the original clause $C_1 = P_1 \vee Q_1 \vee \cdots \vee Q_n$, there exists a derivation path

$$C_1 \xrightarrow{D_1} C_2 \xrightarrow{D_2} \cdots C_{i-1} \xrightarrow{D_{i-1}} C_i \xrightarrow{D_i} \cdots C_j \xrightarrow{D_j} C_{j+1},$$

in which C_{j+1} is a success clause and for each $1 \leqslant m \leqslant j$, D_m is an E-coloring clause. As C_{j+1} is a success clause, for each black literal P_i in the clause C_{j+1}, there exists an E-coloring clause $\underline{Q^{\perp}_i} \vee P_{k_i}$ in $D_1, \ldots, D_j$. Thus for each black literal P_i in the clause C_{j+1}, there exists a vertex s_{k_i} such that there is an edge from s_i to s_{k_i}. As the number of black literals in C_{j+1} is finite, for each vertex s_i, if P_i is a member of C_{j+1}, then starting from s_i, there exists a path which contains a cycle. As the literal P_1 is in C_{j+1}, starting from s_1, there exists a path to a cycle.

□

.2 Correctness of the Encoding

Theorem 5. *Let G be a graph and s_1 be a vertex of G. Starting from s_1, there is no blocked walk if and only if, starting from $\{\mathsf{ori}(s_1, G)\} \cup AC(G)$, a success clause can be derived.*

Before proving this theorem, a lemma is needed.

Lemma 3. *Let G be a graph and s_1 be a vertex of G. Starting from s_1, if all the reachable vertices are traversed in the order $s_1, s_2, \ldots, s_k$ and each reachable vertex has at least one successor, then starting from $\{\mathsf{ori}(s_1, G)\} \cup AC(G)$, there exists a derivation path $C_1(\mathsf{ori}(s_1, G)) \xrightarrow{D_1} C_2 \xrightarrow{D_2} \cdots C_k \xrightarrow{D_k} C_{k+1}$, in which C_{k+1} is a success clause and $\forall 1 \leqslant i \leqslant k$, D_i is an A-coloring clause of the form $\underline{Q_i^{\perp}} \vee P_{i_1} \vee \cdots \vee P_{i_j}$.*

Proof. As $s_1, s_2 \ldots, s_k$ are all the reachable vertices starting from s_1, for a vertex s, if there exists an edge from one of the vertices in $s_1, s_2, \ldots, s_k$ to s, then s is a member of $s_1, s_2, \ldots, s_k$. Thus, after the derivation $C_1 \xrightarrow{D_1} C_2 \xrightarrow{D_2} \cdots C_j \xrightarrow{D_j} C_{j+1}$, for each black literal P_i, the white literal Q_i is not in C_{j+1}, thus C_{j+1} is a success clause. □

Now it is ready to prove Theorem 5. The proof is as follows.

Proof of Theorem 5

Proof.
- For the right direction, assume that all the reachable vertices starting from s_1 are traversed in the order $s_1, s_2, \ldots, s_k$. For the resolution part, by Lemma 3, starting from the original clause, a success clause can be derived.
- For the left direction, by Proposition 2, starting from the original clause $C_1 = \mathsf{ori}(s_1, G)$, there exists a derivation path

$$C_1 \xrightarrow{D_1} C_2 \xrightarrow{D_2} \cdots C_j \xrightarrow{D_j} C_{j+1},$$

in which C_{j+1} is a success clause and $\forall 1 \leqslant i \leqslant j$, D_i is an A-coloring clause with $Q_{k_i}^{\perp}$ underlined. As there is no i such that both P_i and Q_i are in C_{j+1}, for the vertices in $s_{k_1}, s_{k_2}, \ldots, s_{k_j}$, the successors of each vertex is a subset of $s_{k_1}, s_{k_2}, \ldots, s_{k_j}$. As the black literal P_1 is in the clause C_{j+1}, by the definition of success clause, the white literal Q_1 is not in C_{j+1}, thus s_1 is a member of $s_{k_1}, s_{k_2}, \ldots, s_{k_j}$. Then recursively, for each vertex s, if s is reachable from s_1, then s is in $s_{k_1}, s_{k_2}, \ldots, s_{k_j}$. Thus starting from s_1, all the vertices reachable have successors.

□

.3 Completeness

For the completeness of our method, we first prove that PRM_δ is complete, then we prove that PRM_δ remains complete when we apply PSER eagerly.

Proposition 3 (Completeness of PRM_δ). *Let M be a set of coloring clauses and $C_1, \ldots, C_n$ be traversal clauses. If $M, C_1, \ldots, C_n \to S$, in which the clause S is a success clause, then starting from $M, C_1, \ldots, C_n$, we can build a derivation by selecting the resolved literals with selection function δ in Definition 6 and get a success clause.*

Proof. By Proposition 1 and Proposition 2, there exists $1 \leqslant i \leqslant n$, such that $C_i(C_{i_0}) \xrightarrow{D_1} C_{i_1} \cdots \xrightarrow{D_n} C_{i_n}(S)$. As there are no white literals in any clauses of $D_1, \ldots, D_n$ and in each step of the resolution, the resolved literal in the traversal clause is a white literal, the order of white literals to be resolved in the derivation by applying Resolution rule with coloring clauses in $D_1, \ldots, D_n$ will not affect the result. Thus use selection function δ to select white literals to be resolved, until we get a traversal clause S' such that there are no grey literals in it. By the definition of success clause, S' is a success clause. □

Lemma 4. *Let M be a set of coloring clauses and C be a traversal clause. Assume $C(H_0) \xrightarrow{D_1}_\delta H_1 \xrightarrow{D_2}_\delta \cdots H(H_i) \xrightarrow{D_i}_\delta \cdots \xrightarrow{D_n}_\delta H_n$ in which H_n is a success clause and for each $1 \leqslant j \leqslant n$, the coloring clause D_j is in M, and $M, C \to^\delta K$ such that $\mathsf{grey}(H) = \mathsf{grey}(K)$. If $K, D_1, \ldots, D_n \to^\delta K'$, and K' is not a success clause, then there exists a coloring clause D_k in $D_1, \ldots, D_n$, such that $K' \xrightarrow{D_k}_\delta K''$.*

Proof. As K' is not a success clause, assume that the literals P_i and Q_i are in K'. As Q_i cannot be introduced in each step of resolution between a traversal clause and a coloring clause, Q_i is in C and K. As the literal P_i is in clause K', during the derivation of K', there must be some clauses which contains P_i:

- if the literal P_i is in K, as Q_i is also in K, Q_i is a grey literal of K. As $\mathsf{grey}(H) = \mathsf{grey}(K)$, the literal P_i is also in H, and as P_i cannot be selected during the derivation, it remains in the traversal clauses $H_{i+1}, \ldots, H_n$.
- if the literal P_i is introduced by applying Resolution rule with coloring clause D_j in $D_1, \ldots, D_n$, which is used in the derivation of H_n as well, so the literal P_i is also a member of H_n.

In both cases, the literal P_i is in H_n. As H_n is a success clause, the literal Q_i is not a member of H_n. As Q_i is in C, there exists a coloring clause D_k in $D_1, \ldots, D_n$ with the literal $Q_i^\perp$ selected. Thus, $K' \xrightarrow{D_k}_\delta K''$. □

Lemma 5. *Let M be a set of A(E)-coloring clauses and C be a traversal clause. If we have $M, C \to^\delta H$ and $M, C \to^\delta K$, such that $\mathsf{grey}(H) = \mathsf{grey}(K)$, then starting from M, H a success clause can be derived if and only if starting from M, K a success clause can be derived.*

Proof. Without loss of generality, prove that if starting from M, H we can get to a success clause, then starting from M, K, we can also get to a success clause. By Proposition 2, starting from C, there exists $H_0(C) \xrightarrow{M}_\delta H_1 \xrightarrow{M}_\delta \cdots H_i(H) \xrightarrow{M}_\delta \cdots \xrightarrow{M}_\delta H_n$, in which H_n is a success clause. More precisely, $H_0(C) \xrightarrow{D_1}_\delta H_1 \xrightarrow{D_2}_\delta \cdots H_i(H) \xrightarrow{D_{i+1}}_\delta \cdots \xrightarrow{D_n}_\delta H_n$, where for each $1 \leqslant j \leqslant n$, the coloring clause D_j is in M. Then by Lemma 4, starting from M, K, we can always find a coloring clause in $D_1, \ldots, D_n$ to apply resolution with the new generated traversal clause, until we get a success clause. As the white literals in the generated traversal clauses decrease by each step of resolution, we will get a success clause at last. □

Theorem 6 (Completeness). *PRM_δ with $PSER$ is complete.*

Proof. By Lemma 5, each time after we apply PSER, the satisfiability is preserved. □