A linear algorithm for renaming a set of clauses as a Horn set*

Jean-Jacques Hébrard

Laboratoire d'Informatique, Université de Caen, F-14032 Caen Cedex, France

Abstract


A new algorithm for renaming a set of clauses as a Horn set is presented. Its time and space complexity is linear (w.r.t. the number of occurrences of literals) on a random access machine, and it can be considered as a generalization of the algorithm of Even, Itai and Shamir which decides whether a set of 2-clauses is satisfiable. Breadth-first search plays here a crucial part for achieving linear complexity.

0. Introduction

We say that we rename a variable p in a propositional formula if we replace in it every occurrence of p (resp. ¬p) by ¬p (resp. p). The aim of this paper is to present an algorithm which decides whether it is possible to transform a given set of clauses into a set of Horn clauses by renaming some variables. For instance, with

\[ S = \{ \neg p \lor \neg q \lor \neg s, r \lor s, q \lor p \lor \neg s, p \lor \neg r \} \]

the answer would be affirmative (just rename p and r).

* This paper was presented at the “Journées sur le Problème de la Satisfiabilité”, Barbizon, France, 1991. The Editor-in-Chief of Theoretical Computer Science would like to acknowledge the help of Olivier Dubois (Université Pierre et Marie Curie, Paris) in the publication of this paper.
Renaming was first considered by Meltzer [9] in the context of automatic theorem proving. Renaming a set of clauses as a Horn set can be very useful to solve the satisfiability problem, since renaming does not change satisfiability, and satisfiability can be decided in linear time for a Horn set [3, 5].

Various algorithms for renaming a set of clauses as a Horn set have been proposed. Lewis [6] achieved a quadratic algorithm by reducing the problem to the satisfiability of a set of 2-clauses (i.e. clauses with only two literals). Aspvall [1] improved this method and obtained a linear algorithm. The improvement consists in decreasing the number of 2-clauses associated with the problem by extending the set of variables. Mannila and Mehlhorn [8], Lindhorst and Shahrokhi [7] reduced the problem to find the strongly connected components of a directed graph, and gave an $O(mn)$ algorithm [7], where $m$ and $n$ are the number of clauses and the number of variables, respectively. Chandru et al. [2] associated with the set of clauses a directed bipartite graph in which every vertex represents either a clause or a variable, and obtained a linear algorithm by introducing specific concepts on bipartite graphs.

We show that a simple analysis of the problem gives a new linear algorithm. We consider the binary relation over the literals which naturally express the constraints of Horn renaming. Our algorithm essentially performs, with an appropriate control, a breadth-first search of the graph of this relation. It can be considered as a generalization of the algorithm of Even, Itai and Shamir which decides whether a set of 2-clauses is satisfiable [4].

1. Preliminaries

We recall that a positive literal is a propositional variable, and a negative literal is of the form $\neg p$ where $p$ is a propositional variable and $\neg$ the negation symbol. A clause is a finite set of literals, and it is said to be a Horn clause if the number of positive literals it contains is zero or one. Throughout the paper, $S$ denotes a fixed finite set of clauses.

Notation. Let $l$ be a literal. If $l = p$ (resp. $l = \neg p$) then $\bar{l}$ denotes $\neg p$ (resp. $p$).

Let $L$ be a set of literals. $L$ is said to be coherent if it does not contain $p$ and $\neg p$ for every propositional variable $p$. $L$ is complete if $p \in L$ or $\neg p \in L$ for every propositional variable $p$ occurring in a clause of $S$. A renaming is a coherent and complete set of literals. Let $R$ be a renaming, a variable $p$ is renamed $\iff \neg p \in R$. If $p \in R$, we write $R(p) = p$ and $R(\neg p) = \neg p$ (if $p$ is not renamed); if $\neg p \not\in R$, we write $R(p) = \neg p$ and $R(\neg p) = p$ (if $p$ is renamed). $R$ is said to be a Horn renaming if $\{R(l_1), \ldots, R(l_n)\}$ is a Horn clause for every clause $\{l_1, \ldots, l_n\} \in S$.

Remark that given a renaming $R$, $R(l)$ is a positive literal $\iff l \in R$. Therefore, two literals of a Horn renaming never belong to the same clause. More precisely, let $l$ and
t be two literals belonging to the same clause and $H$ a Horn renaming, if $l \in H$ then $\overline{t} \notin H$. This leads us to define the relation $\Rightarrow$ over the literals:

$$l \Rightarrow t$$

iff there exists $C \in \mathcal{S}$ such that $l \in C$, $\overline{t} \in C$ and $l \neq \overline{t}$.

The transitive reflexive closure of $\Rightarrow$ is denoted by $\Rightarrow^*$, and $\text{Clos}(l)$ denotes the set $\{t \mid l \Rightarrow^* t\}$.

**Remark (Duality).** $l \Rightarrow t$ iff $t \Rightarrow l$, and $l \Rightarrow^* t$ iff $t \Rightarrow^* l$.

A set of literals $L$ is said to be *closed* if $\text{Clos}(l) \subseteq L$ for every $l \in L$.

**Proposition 1.1.** A renaming is a Horn renaming if and only if it is closed.

This proposition follows directly from the definitions, its proof is omitted.

2. Sketch of the algorithm

The existence of a Horn renaming implies that $\text{Clos}(p)$ or $\text{Clos}(\neg p)$ is coherent for every propositional variable $p$. Conversely, if $\text{Clos}(p)$ or $\text{Clos}(\neg p)$ is coherent for every $p$ then it is possible to construct a Horn renaming $H$. The construction is given by the algorithm in Fig. 1. Initially, $H$ is empty. Then literals are incorporated into $H$ so that it always remains closed and coherent. More precisely, we consider a variable $p$ such that $p \notin H$ and $\neg p \notin H$. Either $\text{Clos}(p) \setminus H$ or $\text{Clos}(\neg p) \setminus H$ is chosen and its elements are inserted into $H$. The only condition is that the chosen set must be coherent. Note that the choice is irreversible. The algorithm terminates as soon as $H$ is complete.

**Algorithm**

$$H \leftarrow \emptyset;$$

for each propositional variable $p$ do

if $p \notin H$ and $\neg p \notin H$ then

if $\text{Clos}(p) \setminus H$ and $\text{Clos}(\neg p) \setminus H$ are coherent then

$[T \leftarrow \text{Choice}(\text{Clos}(p) \setminus H, \text{Clos}(\neg p) \setminus H);$

$H \leftarrow H \cup T]$ 

else if $\text{Clos}(p) \setminus H$ is coherent then $H \leftarrow H \cup (\text{Clos}(p) \setminus H)$

else if $\text{Clos}(\neg p) \setminus H$ is coherent then $H \leftarrow H \cup (\text{Clos}(\neg p) \setminus H)$

else return (failure)

Fig. 1.
Proposition 2.1. If the algorithm (Fig. 1) returns “failure” then there is no Horn renaming. Otherwise the final value of $H$ is a Horn renaming.

The proof of Proposition 2.1 is based on the following lemma.

Lemma 2.2. (i) Let $H$ be a closed set of literals and $l$ a literal such that $l \notin H$. If $t \in \text{Clos}(l)$ then $\bar{t} \notin H$.

(ii) Let $H$ be a closed and coherent set of literals, and $l$ a literal such that $l \notin H$. If $\text{Clos}(l) \setminus H$ is coherent then $H \cup \text{Clos}(l)$ is coherent and closed.

Proof. (i) We have $l \Rightarrow t$ and $\bar{t} \Rightarrow \bar{t}$ (duality). If $\bar{t} \in H$ then $\bar{t} \in H$ ($H$ is closed). Contradiction.

(ii) If $H \cup \text{Clos}(l)$ is not coherent, there exists a literal $t$ such that $t \in \text{Clos}(l) \setminus H$ and $\bar{t} \in H$; according to (i) this is impossible. $H \cup \text{Clos}(l)$ is closed since $H$ is closed. \[ \square \]

Proof of Proposition 2.1. $H$ is always coherent and closed (Lemma 2.2). If the algorithm returns “failure”, there exists a propositional variable $p$ such that $\text{Clos}(p)$ and $\text{Clos}(\neg p)$ are not coherent, thus, there is no Horn renaming. Otherwise, when the algorithm stops, $H$ is a Horn renaming since it is complete coherent and closed. \[ \square \]

3. Breadth-first search

Consider the directed graph, with multiple arcs, associated with $\Rightarrow$ (the number of arcs from $l$ to $t$ is equal to the number of clauses containing $l$ and $t$). $\text{Clos}(p) \setminus H$ and $\text{Clos}(\neg p) \setminus H$ are calculated by performing a breadth-first search of this graph. We shall see that the use of breadth-first search is essential for obtaining linear complexity.

$\text{Clos}(p) \setminus H$ and $\text{Clos}(\neg p) \setminus H$ are determined simultaneously by alternating one step of computation for $\text{Clos}(p) \setminus H$ with one step of computation for $\text{Clos}(\neg p) \setminus H$. If $\text{Clos}(p) \setminus H$ (resp. $\text{Clos}(\neg p) \setminus H$) is coherent and its calculation completed first then its elements are placed in $H$. The calculation of $\text{Clos}(p) \setminus H$ (resp. $\text{Clos}(\neg p) \setminus H$) is stopped as soon as it becomes incoherent; in this case the algorithm continues with $\text{Clos}(\neg p) \setminus H$ (resp. $\text{Clos}(p) \setminus H$) only.

We give below a detailed description of the algorithm in Figs. 2 and 3. Breadth-first search is implemented by using two queues $Q$ and $\bar{Q}$ with operations DEQUEUE and ENQUEUE. DEQUEUE($Q$) deletes the first element of $Q$; ENQUEUE($Q$, $l$) inserts $l$ at the end of $Q$. HEAD($Q$) denotes the first element of $Q$. The variables $T$ and $\bar{T}$ denote $\text{Clos}(p) \setminus H$ and $\text{Clos}(\neg p) \setminus H$, respectively. The procedures Onestep-$T$ and Onestep-$\bar{T}$ allow a step-by-step calculation of $\text{Clos}(p) \setminus H$ and $\text{Clos}(\neg p) \setminus H$. The definition of Onestep-$T$ is given by Fig. 3 (Onestep-$\bar{T}$ is obtained by substituting $\bar{T}$ for $T$ and $\bar{Q}$ for $Q$).

Notations. The clauses are supposed to be numbered $C_1, \ldots, C_n$. If $l$ and $\bar{t}$ belong to $C_i$ then $(l, i, t)$ denotes the corresponding arc, and $t$ is said to be a successor of $l$. 

Algorithm

\[ H \leftarrow \emptyset; \]
\[ \text{for each propositional variable } p \text{ do} \]
\[ \quad \text{if } p \notin H \text{ and } \neg p \notin H \text{ then} \]
\[ \quad \text{begin} \]
\[ \quad \quad T \leftarrow \{p\}; \quad \bar{T} \leftarrow \{-p\}; \quad \text{coherent}(T) \leftarrow \text{true}; \quad \text{coherent}(\bar{T}) \leftarrow \text{true}; \]
\[ \quad \quad Q \leftarrow \emptyset; \quad \bar{Q} \leftarrow \emptyset; \]
\[ \quad \quad \text{if } p \text{ has at least one successor then ENQUEUE}(Q, p); \]
\[ \quad \quad \text{if } \neg p \text{ has at least one successor then ENQUEUE}(\bar{Q}, \neg p); \]
\[ \quad \text{while } Q \neq \emptyset \text{ and } \bar{Q} \neq \emptyset \text{ and coherent}(T) \text{ and coherent}(\bar{T}) \text{ do} \]
\[ \quad \quad \text{Onestep-T; Onestep-}\bar{T}; \]
\[ \quad \quad \text{if } Q = \emptyset \text{ and coherent}(T) \text{ then } H \leftarrow H \cup T \]
\[ \quad \quad \text{else if } \bar{Q} = \emptyset \text{ and coherent}(\bar{T}) \text{ then } H \leftarrow H \cup \bar{T} \]
\[ \quad \quad \text{else if } Q \neq \emptyset \text{ and not coherent}(T) \]
\[ \quad \quad \quad \text{then [while } Q \neq \emptyset \text{ and coherent}(T) \text{ do Onestep-T;} \]
\[ \quad \quad \quad \quad \text{if coherent}(T) \text{ then } H \leftarrow H \cup T \]
\[ \quad \quad \quad \quad \text{else return(failure)]} \]
\[ \quad \quad \quad \text{else if } \bar{Q} \neq \emptyset \text{ and not coherent}(T) \]
\[ \quad \quad \quad \quad \text{then [while } \bar{Q} \neq \emptyset \text{ and coherent}(\bar{T}) \text{ do Onestep-}\bar{T}; \]
\[ \quad \quad \quad \quad \text{if coherent}(\bar{T}) \text{ then } H \leftarrow H \cup \bar{T} \]
\[ \quad \quad \quad \quad \text{else return(failure)]} \]
\[ \quad \quad \text{else return(failure)} \]
\[ \quad \text{end.} \]

Fig. 2.

We still have to show how the arcs originating from a given literal can be examined one by one. Each clause is now considered as a sequence of distinct literals.

Notations

- For every literal \( l \), \( \text{clauses}(l) \) denotes the increasing sequence of integers \( i \) such that \( l \) belongs to \( C_i \).
- By definition, \( t \) is a successor of \( l \) (i.e. \( l \Rightarrow t \)) iff there exists an ordered pair \((j, k)\) such that \( \text{clauses}(l)(j) = i \), \( C_i(k) \neq \bar{i} \) and \( l \neq \bar{i} \); in general, \((j, k)\) is not unique. We write \( t = \text{successor}[(j, k), l] \).
- Consider the list of ordered pairs \((j, k)\) associated with the successors of \( l \), in lexicographic order \(( (j, k) \leq ((j', k') \text{ iff } j < j' \text{ or } (j = j' \text{ and } k \leq k') \)). The element following \((j, k)\) in the list is denoted by \( \text{follow}[(j, k), l] \); if \((j, k)\) is the last element, \( \text{follow}[(j, k), l] = \text{nil} \). The first element of the list is denoted by \( \text{follow}[(0, 0), l] \); if the list is empty, \( \text{follow}[(0, 0), l] = \text{nil} \).
procedure Onestep-T;
begin
  l ← HEAD(Q);
  let (l, i, t) be an arc not yet examined;
  if t ¢ H and t ¢ T then
    if t ∈ T then coherent(T) ← false
    else [T ← T ∪ {t};
      if t has at least one successor then ENQUEUE(Q, t)];
  if every arc (l, i, t) has been examined then DEQUEUE(Q)
end;

Fig. 3.

procedure Onestep-T′;
begin
  l ← HEAD(Q);
  address ← follow[address, l];
  t ← successor[address, l];
  if t ¢ H and t ¢ T then
    if t ∈ T then coherent(T) ← false
    else [T ← T ∪ {t};
      if follow[(0, 0), t] ≠ nil then ENQUEUE(Q, t)];
  if follow[address, l] = nil then [DEQUEUE(Q); address ← (0, 0)]
end;

Fig. 4.

We can now give a more precise description of Onestep-T (Fig. 4). The variable “address” is used for determining the current successor of HEAD(Q), it takes the value (0, 0) each time the first element of Q changes.

4. Linear complexity

Let N = |C₁| + ⋯ + |Cₙ|, where |Cᵢ| denotes the number of literals of Cᵢ. In general, the graph associated with ⇒ has O(N²) arcs, but we show that the algorithm always explores less than 2N arcs. Each processed arc corresponds to a call Onestep-T or Onestep-\overline{T}.

Consider the arcs processed during an iteration of the for-loop (Fig. 2). If the last executed instruction of the iteration is “H ← H ∪ T” (resp. “H ← H ∪ \overline{T}”) then the arcs processed by Onestep-T (resp. Onestep-\overline{T}) are said to be accepted. In the case where
the last executed instruction is "return(failure)", the accepted arcs are those which are processed by Onestep-\( T \) (resp. Onestep-\( \overline{T} \)) if the number of calls Onestep-\( T \) (resp. Onestep-\( \overline{T} \)) performed during the iteration is greater than or equal to (resp. greater than) the number of calls Onestep-\( \overline{T} \) (resp. Onestep-\( T \)). Every iteration of the for-loop determines a set of accepted arcs. Remark that the total number of arcs processed by the algorithm is less than or equal to twice the total number of accepted arcs.

Let \((l, i, t)\) and \((r, j, s)\) be two accepted arcs. We say that \((l, i, t)\) is accepted before \((r, j, s)\) if the call Onestep-\( T \) or Onestep-\( \overline{T} \) which accepts \((l, i, t)\) is executed before the call accepting \((r, j, s)\).

The following lemmas allow one to show that the total number of accepted arcs is less than \( N \).

**Lemma 4.1.** Let \((l, i, t)\) and \((r, j, s)\) be two accepted arcs. If \((l, i, t)\) is accepted before \((r, j, s)\) then \( r \neq \overline{t} \).

**Proof.** If \( r = \overline{t} \), the algorithm terminates before arc \((r, j, s)\) can be processed. \( \Box \)

**Lemma 4.2.** Let \((l, i, t)\) and \((r, j, s)\) be two accepted arcs. If \( l \neq r \) then \( i \neq j \).

**Proof.** Assume \((l, i, t)\) is accepted before \((r, j, s)\). If \( l \neq r \) then the use of a queue for implementing breadth-first search entails that all arcs originating from \( l \) are accepted before \((r, j, s)\). Now suppose \( i = j \), then \( C_i \) contains \( l \) and \( r \), and \((l, i, \overline{t})\) is accepted before \((r, j, s)\). Impossible (Lemma 4.1). \( \Box \)

**Proposition 4.3.** The total number of arcs processed by the algorithm is less than \( 2N \).

**Proof.** We just have to show that the number of accepted arcs is less than \( N \). This follows from the fact that the number of accepted arcs associated with \( C_i \) is less than or equal to \(|C_i| - 1\) (Lemma 4.2). \( \Box \)

**Remark.** If depth-first search is used instead of breadth-first search, the number of processed arcs can be \( \Theta(N^2) \).

**Proposition 4.4.** The algorithm can be implemented with time and space complexity \( O(N) \).

**Proof.** The construction of the sequences \( \text{clauses}(l) \) requires a simple reading of the clauses and can be achieved in time \( O(N) \). The operations \( \text{successor}[(j, k), l] \) and \( \text{follow}[(j, k), l] \) can be performed in constant time. If the sets \( T \), \( \overline{T} \) and \( H \) are represented by boolean arrays \((l \in T \text{ iff } T(l) = 1)\), the operations \( T \cup \{t\} \), \( t \in H \) and \( t \in T \) take constant time. Finally the total time required by Onestep-\( T \) and Onestep-\( \overline{T} \) is \( O(N) \), since the cost of each call is constant and the total number of calls is less than
2N (Proposition 4.3). We still have to show that the operations $H \cup T$ and $H \cup \bar{T}$ take time $O(N)$. Each operation $H \cup T$ (resp. $H \cup \bar{T}$) can be performed in time proportional to the number of elements of $T$ (resp. $\bar{T}$), by using an auxiliary list containing literals $l$ such that $T(l) = 1$ (resp. $\bar{T}(l) = 1$). Then the total cost of $H \cup T$ and $H \cup \bar{T}$ is $O(N)$ since it is proportional to the length of $H$. □

References