Distributed Reasoning in a Peer-to-Peer Setting

P. Adjiman and **P. Chatalic** and **F. Goasdoué** and **M.-C. Rousset** and **L. Simon** [1]

Abstract. In a peer-to-peer inference system, each peer can reason locally but can also solicit some of its acquaintances, which are peers sharing part of its vocabulary. In this paper, we consider peer-to-peer inference systems in which the local theory of each peer is a set of propositional clauses defined upon a local vocabulary. An important characteristic of peer-to-peer inference systems is that the global theory (the union of all peer theories) is not known (as opposed to partition-based reasoning systems). The contribution of this paper is twofold. We provide the first consequence finding algorithm in a peer-to-peer setting: it is anytime and computes consequences gradually from the solicited peer to peers that are more and more distant. We exhibit a sufficient condition on the acquaintance graph of the peer-to-peer inference system for guaranteeing the completeness of this algorithm. We also present first experimental results that are promising.

1 Introduction

Recently peer-to-peer systems have received considerable attention because their underlying infrastructure is appropriate to scalable and flexible distributed applications over Internet. In a peer-to-peer system, there is no centralized control or hierarchical organization: each peer is equivalent in functionality and cooperates with other peers in order to solve a collective task. Peer-to-peer systems have evolved from simple keyword-based peer-to-peer file sharing systems like Napster [13] and Gnutella [5] to schema-based peer data management systems like Edutella [14] or Piazza [8], which handle semantic data description and support complex queries for data retrieval. In those systems, the complexity of answering queries is directly related to the expressivity of the formalism used to state the semantic mappings between peers schemas [7].

In this paper, we are interested in peer-to-peer inference systems in which each peer can answer queries by reasoning from its local (propositional) theory but can also ask queries to some other peers with which it is semantically related by sharing part of its vocabulary. This framework encompasses several applications like peer-to-peer information integration systems or intelligent agents, in which each peer has its own knowledge (about its data or its expertise domain) and some partial knowledge about some other peers. In this setting, when solicited to perform a reasoning task, a peer, if it cannot solve completely that task locally, must be able to distribute appropriate reasoning subtasks among its acquainted peers. This leads to a step by step splitting of the initial task among the peers that are relevant to solve parts of it. The outputs of the different splitted tasks must then be recomposed to construct the outputs of the initial task.

We consider peer-to-peer inference systems in which the local theory of each peer is composed of a set of propositional clauses defined upon a set of propositional variables (called its local vocabulary). Each peer may share part of its vocabulary with some other peers. We investigate the reasoning task of finding consequences of a certain form (e.g., clauses involving only certain variables) for a given input formula expressed using the local vocabulary of a peer. Note that other reasoning tasks like finding implicants of a certain form for a given input formula can be equivalently reduced to the consequence finding task.

The contribution of this paper is twofold. We provide the first consequence finding algorithm in a peer-to-peer setting: it is anytime and computes consequences gradually from the solicited peer to peers that are more and more distant. We exhibit a sufficient condition on the acquaintance graph of the peer-to-peer inference system for guaranteeing the completeness of this algorithm.

It is important to emphasize that the problem of distributed reasoning that we consider in this paper is quite different from the problem of reasoning over partitions obtained by decomposition of a theory ([3, 1]). In that problem, a centralized large theory is given and its structure is exploited to compute its best partitioning in order to optimize a partition-based reasoning algorithm. In our problem, the whole theory (i.e., the union of all the local theories) is not known and the partition is imposed by the peer-to-peer architecture. As we will illustrate it on an example (Section 2), the algorithms based on transmitting clauses between partitions in the spirit of ([1, 3, 4]) are not appropriate for our consequence finding problem. Our algorithm splits clauses if they involve vocabularies of several peers. Each piece of a splitted clause is then transmitted to the corresponding theory to find its consequences. The consequences that are found for each piece of splitted clause must then be recomposed to get the consequences of the clause that had been splitted.

The paper is organized as follows. Section 2 defines formally the peer-to-peer inference problem that we address in this paper. In Section 3, we describe a distributed consequence finding algorithm and we state its properties. Section 4 reports some experimental results. Related work is summarized in Section 5. We conclude with a short discussion in Section 6.

2 Peer-to-peer inference: problem definition

A peer-to-peer inference system (P2PIS) is a network of peer theories. Each peer P is a finite set of propositional formulas of a language $\mathcal{L}_P$. We consider the case where $\mathcal{L}_P$ is the language of clauses without duplicated literals that can be built from a finite set of propositional variables $\mathcal{V}_P$, called the *vocabulary* of P. Peers can be semantically related by having common variables in their respective vocabularies, called *shared variables*. In a P2PIS, no peer has the knowledge of the global P2PIS theory. Each peer only knows its own local theory and that it shares some variables with some other peers of the P2PIS (its *acquaintances*). It does not necessarily know *all* the peers with which it shares variables. When a new peer joins a P2PIS

[1] LRI, CNRS & Université Paris-Sud – INRIA Futurs, Bâtiment 490, 91405, Orsay Cedex, France

it simply declares its acquaintances in the P2PIS, i.e., the peers it knows to be sharing variables with. A P2PIS can be formalized as an *acquaintance graph*.

Definition 1 (Acquaintance graph) *Let $\mathcal{P} = (P_i)_{i=1..n}$ be a family of clausal theories on their respective vocabularies $\mathcal{V}_{P_i}$, let $\mathcal{V} = \cup_{i=1..n}\mathcal{V}_{P_i}$. An acquaintance graph is a graph $\Gamma = (\mathcal{P}, \text{ACQ})$ where $\mathcal{P}$ is the set of vertices and $\text{ACQ} \subseteq \mathcal{V} \times \mathcal{P} \times \mathcal{P}$ is a set of labelled edges such that for every $(v, P_i, P_j) \in \text{ACQ}$, $i \neq j$ and $v \in \mathcal{V}_{P_i} \cap \mathcal{V}_{P_j}$.*

A labelled edge (v, P_i, P_j) expresses that peers P_i and P_j know each other to be sharing the variable v. For a peer P and a literal l, $\text{ACQ}(l, P)$ denotes the set of peers sharing with P the variable of l.

For each theory P, we consider a subset of *target variables* $\mathcal{TV}_P \subseteq \mathcal{V}_P$, supposed to represent the variables of interest for the application, (e.g., observable facts in a model-based diagnosis application, or stored classes in an information integration application). The goal is, given a clause (called the *query*) provided as an input to a given peer, to find all the consequences (called *answers*) that belong to some *target language*.

The point is that the query only uses the vocabulary of the queried peer, but that the expected answers may involve target variables of different peers. The target languages handled by our algorithm are defined in terms of target variables and require that a shared variable has the same target status in all the peers sharing it.

Definition 2 (Target Language) *Let $\Gamma = (\mathcal{P}, \text{ACQ})$ be a P2PIS, and for very peer P, let $\mathcal{TV}_P$ be the set of its target variables such that if $(v, P_i, P_j) \in \text{ACQ}$ then $v \in \mathcal{TV}_{P_i}$ iff $v \in \mathcal{TV}_{P_j}$. For a subset SP of peers of $\mathcal{P}$, we define its* target language *$\mathcal{T}arget(SP)$ as the language of clauses (including the empty clause) involving only variables of $\bigcup_{P \in SP} \mathcal{TV}_P$.*

Among the possible answers we distinguish *local answers*, involving only target variables of the solicited peer, *navigational answers*, which involve target variables of a single peer, and *integrating answers* which involve target variables of several peers.

Definition 3 (Proper prime implicate w.r.t a theory) *Let P be a clausal theory and q be a clause. A clause m is said to be:*

- *a* prime implicate *of q w.r.t P iff $P \cup \{q\} \models m$ and for any other clause m', if $P \cup \{q\} \models m'$ and $m' \models m$ then $m' \equiv m$.*
- *a* proper prime implicate *of q w.r.t P iff it is a prime implicate of q w.r.t P but $P \not\models m$.*

Definition 4 (Consequence finding problem) *Let $\mathcal{P} = (P_i)_{i=1..n}$ be a family of clausal theories with respective target variables $(\mathcal{TV}_{P_i})_{i=1..n}$ and let $\Gamma = (\mathcal{P}, \text{ACQ})$ be a $P2PIS$. The* consequence finding problem *is, given a peer P and a clause $q \in \mathcal{L}_P$ to find the set of proper prime implicates of q w.r.t $\bigcup_{i=1..n} P_i$ which belong to $\mathcal{T}arget(\mathcal{P})$.*

The following example illustrates the main characteristics of the distributed algorithm presented in Section 3.

Example 1 Let us consider 4 peers. P_1 describes a tour operator. Its theory expresses that its current far destinations (F) are either Kenya (K) or Chile (C). These far destinations are international destinations (I) and expensive (E). The peer P_2 is only concerned with police regulations and expresses that a passport is required (P) for international destinations. P_3 focuses on sanitary conditions for travelers. It expresses that, in Kenya, yellow fever vaccination (Y) is strongly recommended and that a strong protection against paludism should be taken (PL) when accomodation occurs in Lodges (L). P_4 describes accommodation conditions for the trips : Lodges for Kenya and Hotels (H) for Chile. It also expresses that when anti-paludism protection is required, accommodations are equipped with appropriate anti-mosquito protections (AM). Shared variables are indicated on the edges of the acquaintance graph (Figure 1) and target variables are defined by : $\mathcal{TV}_{P_1} = \{E\}, \mathcal{TV}_{P_2} = \{P\}, \mathcal{TV}_{P_3} = \{L, Y, PL\}$ and $\mathcal{TV}_{P_4} = \{L, H, PL, AM\}$.

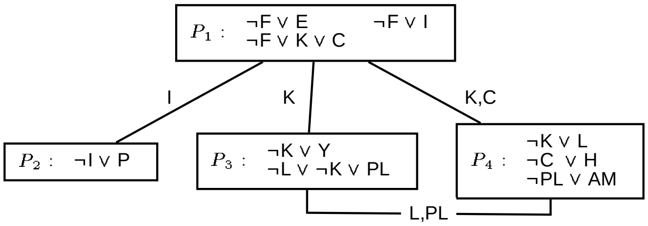

Figure 1. Acquaintance graph for the tour operator example

Suppose that the query F is asked to peer P_1. The local consequences that can be derived by local reasoning on P_1 are E,I and $\mathsf{K} \vee \mathsf{C}$. E is returned immediately as a local answer since it is in $\mathcal{T}arget(P_1)$. Since I is shared with P_2, it is transmitted to P_2, which produces the clause P. Since P is in $\mathcal{T}arget(P_2)$, it is a navigational answer for the initial query. The clause $\mathsf{K} \vee \mathsf{C}$ is made of shared variables. Our algorithm splits such clauses and transmits each component (here K and C) separately to the acquaintances of P_1 sharing respectively K and C with P_1. C is therefore transmitted to P_4, which returns the clause H as unique answer. Similarly, the clause K is transmitted (independently) to peers P_4 and P_3 (both sharing the variables K with P_1). On P_4, this produces locally the clause L. Since $\mathsf{L} \in \mathcal{T}arget(P_4)$ it is returned as an answer for K. But L is also shared, and therefore it is transmitted to the peer P_3, which in turn produces the clause $\neg\mathsf{K} \vee \mathsf{PL}$. This clause is splitted in two pieces $\neg\mathsf{K}$ and PL. P_4 is asked for PL and returns AM as its only answer for PL. P_1 is asked for computing the implicates of $\neg\mathsf{K}$, while the complementary query K is still under process. We will see in Section 3 that this is handled in our algorithm by an history which keeps track of the current reasoning branch: when a same reasoning branch contains two complementary literals, it is stopped and returns $\Box$ as answer. In our example, the answer produced by P_1 for $\neg\mathsf{K}$ is thus $\Box$, which is sent back to P_3. P_3 now combines the answers obtained from the two reasoning branches resulting from the splitting of $\neg\mathsf{K} \vee \mathsf{PL}$, namely AM returned by P_4 for PL, and $\Box$ returned by P_1 for $\neg\mathsf{K}$. This produces AM as answer produced by P_3 for $\neg\mathsf{K} \vee \mathsf{PL}$. P_3 sends this answer back to P_4 as an answer for L. P_4 transmits to P_1 each of the answers L, PL and AM it has computed for L, as answers for K. We do not detail the reasoning branch corresponding to the propagation of K in P_3, which adds a new answer, Y, to the set of answers that are obtained by $P1$ for K. As they are produced, those answers are combined with the only answer for C (i.e., H). Therefore, the set of answers for the initial query that will have been produced at the end is: $\{\mathsf{H} \vee \mathsf{L}, \mathsf{H} \vee \mathsf{PL}, \mathsf{H} \vee \mathsf{AM}, \mathsf{H} \vee \mathsf{Y}\}$. Among those answers, it is important to note that some of them (e.g., $\mathsf{H} \vee \mathsf{Y}$) involve target variables from different peers. Such implicates cannot be obtained by partition-based algorithms like in [1]. This is made possible thanks to the splitting/recombining strategy of our algorithm.

3 Distributed consequence finding algorithm

The message passing distributed algorithm that we provide is described in Section 3.2. We show that it terminates and that it computes the same results as the recursive algorithm described in Section

3.1. We exhibit a property of the acquaintance graph that guarantees the completeness of this recursive algorithm, and therefore of the message passing distributed algorithm (since both algorithms compute the same results).

For both algorithms, we will use the following notations :

- for a literal q, $Resolvent(q, P)$ denotes the set of clauses obtained by resolution between q and a clause of P,

- for a literal q, $\bar{q}$ denotes its complementary literal,

- for a clause c of a peer P, $S(c)$ (resp. $L(c)$) denotes the disjonction of literals of c whose variables are shared (resp. not shared) with any acquaintance of P. The condition $S(c) = \square$ thus expresses that c does not contain any variable shared with an acquaintance of P,

- an history $hist$ is a sequence of triples (l, P, c) (where l is a literal, P a peer, and c a clause). An history $[(l_n, P_n, c_n), \ldots, (l_1, P_1, c_1), (l_0, P_0, c_0)]$ represents a branch of reasoning initiated by the propagation of the literal l_0 within the peer P_0, and the splitting of the clause c_0: for every $i \in [0..n-1]$, c_i is a consequence of l_i and P_i, and l_{i+1} is a literal of c_i, which is propagated in P_{i+1},

- $\circledvee$ is the distribution operator on sets of clauses: $S_1 \circledvee \cdots \circledvee S_n$ = $\{c_1 \vee \cdots \vee c_n \,|c_1 \in S_1, \ldots, c_n \in S_n\}$. If $L = \{l_1, \ldots, l_p\}$, we will use the notation $\circledvee_{l \in L} S_l$ to denote $S_{l_1} \circledvee \cdots \circledvee S_{l_p}$.

3.1 Recursive consequence finding algorithm

Let $\Gamma = (\mathcal{P}, \text{ACQ})$ be a P2PIS. $RCF(q, P, \Gamma)$ computes implicates of the literal q w.r.t $\mathcal{P}$, starting with the computation of consequences of q w.r.t P, and then being guided by the topology of the acquaintance graph of the P2PIS. To ensure termination, it is necessary to keep track of the literals already processed by peers. This is done by the recursive algorithm $RCFH(q, SP, \Gamma, hist)$, where $hist$ is the history of the reasoning branch ending up to the propagation of the literal q in SP (a set of acquaintances of the last peer added to the history).

Algorithm 1: Recursive consequence finding algorithm

```
$RCF(q, P, \Gamma)$
(1) return $RCFH(q, \{P\}, \Gamma, \emptyset)$

$RCFH(q, SP, \Gamma, hist)$
(1) if $(\bar{q}, \_, \_) \in hist$ return $\{\square\}$
(2) else if there exists $P \in SP$ s.t $q \in P$ or for every $P \in SP$,
     $(q, P, \_) \in hist$ return $\emptyset$
(3) else LOCAL $\leftarrow \{q\} \cup (\bigcup_{P \in SP} Resolvent(q, P))$
(4) if $\square \in$ LOCAL return $\{\square\}$
(5) else LOCAL $\leftarrow \{c \in \text{LOCAL} | L(c) \in \mathcal{T}arget(SP)\}$
(6) if for every $c \in$ LOCAL, $S(c) = \square$, return LOCAL
(7) else RESULT $\leftarrow$ LOCAL
(8)     foreach $c \in$ LOCAL s.t $S(c) \neq \square$
(9)         let $P$ be the peer of $SP$ s.t. $c \in Resolvent(q, P)$
(10)        $\mathcal{P}' \leftarrow \mathcal{P} \backslash \{\neg q \vee c\}$,     $\Gamma' \leftarrow (\mathcal{P}', \text{ACQ})$
(11)        foreach literal $l \in S(c)$
(12)            ANSWER$(l) \leftarrow RCFH(l, \text{ACQ}(l, P), \Gamma', [(q, P, c)|hist])$
(13)        DISJCOMB $\leftarrow (\circledvee_{l \in S(c)} \text{ANSWER}(l)) \circledvee \{L(c)\}$
(14)        RESULT $\leftarrow$ RESULT $\cup$ DISJCOMB
(15)    return RESULT
```

Theorem 1 *$RCF(q, P, \Gamma)$ is sound and terminates.*

Sketch of proof: For the soundness, it is easy to show (by induction on the number of recursive calls) that every result returned by $RCFH(q, \{P\}, \Gamma, \emptyset)$ is an implicate of $\mathcal{P} \cup \{q\}$. For the termination, we notice that at each recursive call, a *new* triple (sl, P, c) is added to the history. If the algorithm did not terminate, the history would be infinite, which is not possible since the number of peers, literals and clauses within a P2PIS is finite. **Soundness:** if the history $hist$ is not empty, it is of the form $[(l_n, P_n, c_n), \ldots, (l_0, P_0, c_0)]$. We prove by induction on the number rc of recursive calls of $RCFH(q, SP, \Gamma, hist)$ that every result returned by $RCFH(q, SP, \Gamma, hist)$ is an implicate of $\mathcal{P} \cup \{q, l_n, \ldots, l_0\}$ which belongs to the target language.

- rc = 0: either one of the conditions of Line (1), Line (2), Line (4) or Line (6) is satisfied.

  - If either there exists a peer P such that $(\bar{q}, \_, \_) \in hist$ or $\square \in$ LOCAL: in both cases, $\square$ is returned by the algorithm (in respectively Line (1) and Line (4)) and it is indeed an implicate of $\mathcal{P} \cup \{q, l_n, \ldots, l_0\}$ belonging to the target language.

  - If there exists $P \in SP$ such that $q \in P$ or for every $P \in SP$, $(q, P, \_) \in hist$: in this case, an empty result is returned (Line(2)) because q is already in $\mathcal{P} \cup \{q, l_n, \ldots, l_0\}$.

  - In the last case (Line (6)), $r \in$ LOCAL, and it is obvioulsy an implicate of $\mathcal{P} \cup \{q, l_n, \ldots, l_0\}$ (as a resolvent of q and of a clause of $\bigcup_{P \in SP} P$), and it belongs to the target language.

- Suppose the induction hypothesis true for $rc \leq p$, and let $\Gamma = (\mathcal{P}, \text{ACQ})$ a P2PIS such that $RCFH(q, SP, \Gamma, hist)$ requires $p + 1$ recursive calls to terminate. Let r a result returned by $RCFH(q, SP, \Gamma, hist)$.

-If $r \in$ LOCAL, it is obvioulsy an implicate of $\mathcal{P} \cup \{q, l_n, \ldots, l_0\}$ belonging to the target language.

-If $r \notin$ LOCAL, it is obtained at Line (13): there exist a clause c= $S(c) \vee L(c)$ such that $S(c) = ll_1 \vee \cdots \vee ll_k$ and $r = r_1 \vee \cdots \vee r_k \vee L(c)$, where every r_i is a result returned by $RCFH(ll_i, \text{ACQ}(ll_i, P), \Gamma', [(q, P, c)|hist])$ (Line (12)). According to the induction hypothesis (the number of recursive calls of $RCFH(ll_i, \text{ACQ}(ll_i, P), \Gamma', [(q, P, c)|hist])$ for every ll_i is less than p), every r_i is an implicate of $\mathcal{P} \backslash \{\neg q \vee c\} \cup \{ll_i, q, l_n, \ldots, l_0\}$ belonging to the target language. Therefore, $r_1 \vee \cdots \vee r_k$ is an implicate of $\mathcal{P} \backslash \{\neg q \vee c\} \cup \{S(c), q, l_n, \ldots, l_0\}$, which belongs to the target language. Since $L(c)$ belongs to the target language and $c = S(c) \vee L(c)$, r (i.e, $r_1 \vee \cdots \vee r_k \vee L(c)$) belongs to the target language, and is an implicate of $\mathcal{P} \backslash \{\neg q \vee c\} \cup \{c, q, l_n, \ldots, l_0\}$. Since $c \in$ LOCAL, c is an implicate of $\mathcal{P} \cup \{q\}$, and therefore r is an implicate of $\mathcal{P} \cup \{q, l_n, \ldots, l_0\}$ belonging to the target language.

Termination: at each recursive call, a *new* triple (sl, P, c) is added to the history. If the algorithm did not terminate, the history would be infinite, which is not possible since the number of peers, literals and clauses within a P2PIS is finite. □

The following theorem exhibits a sufficient condition for the algorithm to be complete.

Theorem 2 *Let $\Gamma = (\mathcal{P}, \text{ACQ})$ be a P2PIS all local theories of which are saturated by resolution. If for every P, P' and $v \in \mathcal{V}_P \cap \mathcal{V}_{P'}$ there exists a path between P and P' in Γ, all edges of which are labelled with v, then for every literal $q \in \mathcal{L}_P$, $RCF(q, P, \Gamma)$ computes all the proper prime implicates of q w.r.t $\mathcal{P}$ which belong to $\mathcal{T}arget(\mathcal{P})$.*

Sketch of proof: If the history $hist$ is not empty, it is of the form $[(l_n, P_n, c_n), \ldots, (l_0, P_0, c_0)]$. We prove by induction on the number rc of recursive calls that $RCFH(q, SP, \Gamma, hist)$ computes all the proper prime implicates of q w.r.t $\mathcal{P} \cup \{l_n, \ldots, l_0\}$ which belong to the target language.

- rc = 0: either one of the conditions of Line (1), Line (2), Line (4) or Line (6) is satisfied.

  - If either there exists a peer P such that $(\bar{q}, \_, \_) \in hist$ or $\square \in$ LOCAL: in both cases, $\square$ is the only prime implicate of q w.r.t $\mathcal{P} \cup \{l_n, \ldots, l_0\}$ and is returned by the algorithm (respectively Line

(1) and Line (4)).

- If there exists $P \in SP$ such that $q \in P$ or for every $P \in SP$, $(q, P, \_) \in hist$: in this case, all the prime implicates of q w.r.t $\mathcal{P} \cup \{l_n, \ldots, l_0\}$ are prime implicates of $\mathcal{P} \cup \{l_n, \ldots, l_0\}$. Therefore, the set of proper prime implicates of q w.r.t $\mathcal{P} \cup \{l_n, \ldots, l_0\}$ is empty: it is returned by the algorithm (Line(2)).

- If for every $P \in SP$, every resolvent of q w.r.t P has no shared variable with any acquaintance of P: if $\mathcal{P}$ satisfies the property stated in the theorem, this means that every prime implicate of q w.r.t P shares no variable with any other theory of $\mathcal{P}$.
According to Lemma 1, the set of proper prime implicates of q w.r.t $\mathcal{P} \cup \{l_n, \ldots, l_0\}$ is included in LOCAL, and thus every proper prime implicates of q w.r.t $\mathcal{P} \cup \{l_n, \ldots, l_0\}$, which is in the target language, is returned by the algorithm (Line(6)).

• Suppose the induction hypothesis true for $rc \leq p$, and let $\Gamma = (\mathcal{P}, \text{ACQ})$ a P2PIS satisfying the property stated in the theorem and such that $RCFH(q, SP, \Gamma, hist)$ requires $p + 1$ recursive calls to terminate. Let m be a proper prime implicate of q w.r.t $\mathcal{P} \cup \{l_n, \ldots, l_0\}$, which is in the target language. Let us show that it belongs to the result returned by $RCFH(q, SP, \Gamma, hist)$.

- If m is a resolvent of q w.r.t a given P of SP, then $m \in$ LOCAL and is returned by the algorithm if it is in the target language.

-If m is not a resolvent of q w.r.t a given P of SP, then, according to Lemma 1, either q shares its variable with clauses in $\mathcal{P} \cup \{l_n, \ldots, l_0\} \backslash \bigcup_{P \in SP} P$, or there exists a clause $\neg q \vee c$ in $\bigcup_{P \in SP} P$ such that $\neg q \vee c$ has shared variables with $\mathcal{P} \cup \{l_n, \ldots, l_0\} \backslash \bigcup_{P \in SP} P$ and m is a proper prime implicate of c w.r.t $\mathcal{P} \backslash \{\neg q \vee c\} \cup \{q, l_n, \ldots, l_0\}$. Therefore, if the non shared variables of c are target variables, c is involved in an iteration of the loop of Line (8). If Γ verifies the property stated in the theorem, Γ' satisfies it too. According to the induction hypothesis (the number of recursive calls to obtain ANSWER(l) in Line (12) is less than p), for every $l \in S(l)$, ANSWER(l) includes the set of proper prime implicates of l w.r.t $\mathcal{P} \backslash \{\neg q \vee c\} \cup \{q, l_n, \ldots, l_0\}$, which are in the target language. We now apply Lemma 2 to infer that $DisjComp$, which is computed in Line (13), includes the set of proper prime implicates of c w.r.t $\mathcal{P} \backslash \{\neg q \vee c\} \cup \{q, l_n, \ldots, l_0\}$, which are in the target language, and in particular m. □

Lemma 1 *Let P be a set of clauses and m be a proper prime implicate of q w.r.t P. Let $P' \subseteq P$ saturated by resolution such that it contains clauses sharing the variable of q. If m is a proper prime implicate of q w.r.t P, then :*

- either m is a proper prime implicate of q w.r.t P',

- or the variable of q is shared with clauses of $P \backslash P'$,

- or there exists a clause $\neg q \vee c$ of P' such that c has shared variables with clauses of $P \backslash P'$ and m is a proper prime implicate of c w.r.t $P \backslash \{\neg q \vee c\} \cup \{q\}$.

Sketch of proof: Let m be a proper prime implicate of q w.r.t P. If m is different from q, there exists a clause $\neg q \vee c$ in P such that m is a proper prime implicate of c w.r.t $P \backslash \{\neg q \vee c\} \cup \{q\}$.

- If such a clause does not exist in P', it exists in $P \backslash P'$ and therefore q shares its variable with clauses of $P \backslash P'$.

- If there exists a clause $\neg q \vee c$ in P' such that m is a proper prime implicate of c w.r.t $P \backslash \{\neg q \vee c\} \cup \{q\}$, and m is not a proper prime q w.r.t P', then for every proof of m there must exist a clause c' in $P \backslash P'$ with which either q or $\neg q \vee c$ must be resolved. Therefore, either q or c has shared variables with clauses of $P \backslash P'$. □

Lemma 2 *Let P be a set of clauses, and let $c = l_1 \vee \cdots \vee l_n$ be a clause. For every proper prime implicate m of c w.r.t P, there exists $m_1, \ldots, m_n$ such that $m \equiv m_1 \vee \cdots \vee m_n$, and for every $i \in [1..n]$, m_i is a proper prime implicate of l_i w.r.t P.*

Sketch of proof: Let m be a proper prime implicate of c w.r.t P. For every literal l_i, let $Mod(l_i)$ be the set of models of P which make l_i true. If $Mod(l_i) = \emptyset$, that means that □ is the only proper prime implicate of l_i w.r.t P. For every i such that $Mod(l_i) \neq \emptyset$, every model in $Mod(l_i)$ is a model of $P \cup \{c\}$, and then a model of m ; therefore, m is a proper implicate of $P \cup \{l_i\}$, and, by definition of proper *prime* implicates, there exists a proper prime implicate m_i of l_i w.r.t P such that $m_i \models m$. Consequently, there exists $m_1, \ldots, m_n$ such that $m_1 \vee \cdots \vee m_n \models m$, and for every $i \in [1..n]$, m_i is a proper prime implicate of l_i w.r.t P (m_i may be □). Since $P \cup \{l_1 \vee \cdots \vee l_n\} \models m_1 \vee \cdots \vee m_n$, and m is a proper implicate of $l_1 \vee \cdots \vee l_n$ w.r.t P, we necessarily get that $m \equiv m_1 \vee \cdots \vee m_n$. □

3.2 Message-based consequence finding algorithm

In this section, we exhibit the result of the transformation of the previous recursive algorithm into a message-based distributed consequence finding algorithm. Each peer has the algorithm implemented locally. It is composed of three procedures, each one being triggered by the reception of a message. The procedure RECEIVEQUERYMESSAGE is triggered by the reception of a $query$ message $m(Sender, Receiver, query, hist, l)$ sent by the peer $Sender$ to the peer $Receiver$ which executes the procedure: on the demand of $Sender$, with which it shares the variable of l, it processes the literal l. The procedure RECEIVEANSWERMESSAGE is triggered by the reception of an $answer$ message $m(Sender, Receiver, answer, hist, r)$ sent by the peer $Sender$ to the peer $Receiver$ which executes the procedure: it processes the answer r (which is a clause the variables of which are target variables) sent back by $Sender$ for the literal l (last added in the history) ; it may have to combine it with other answers for literals being in the same clause as l. The procedure RECEIVEFINALMESSAGE is triggered by the reception of a $final$ message $m(Sender, Receiver, final, hist, true)$: the peer $Sender$ notifies the peer $Receiver$ that answer computation for the literal l (last added in the history) is completed. Those procedures handle two data structures stored at each peer: ANSWER$(l, hist)$ caches the answers resulting from the propagation of l within the reasoning branch corresponding to $hist$; FINAL$(q, hist)$ is set to true when the propagation of q within the reasoning branch of the history $hist$ is completed. The reasoning is initiated by the user (denoted by a particular peer $User$) sending to a given peer P a message $m(User, P, query, \emptyset, q)$, which triggers the procedure RECEIVEQUERYMESSAGE$(m(User, P, query, \emptyset, q))$ that is locally executed by P. In the description of the procedures, since they are locally executed by the peer which receives the message, we will denote by $Self$ the receiver peer.

The following theorem states two important results: first, the message-based distributed algorithm computes the same results as the algorithm of Section 3.1, and thus, is complete under the same conditions as in Theorem 2 ; second the user is notified of the termination when it occurs, which is crucial for an anytime algorithm.

Theorem 3 *Let r be a result returned by $RCF(q, P, \Gamma)$. If P receives from the user the message $m(User, P, query, \emptyset, q)$, then a message $m(P, User, answer, [(q, P, \_)], r)$ will be produced. If r is the last result returned by $RCF(q, P, \Gamma)$, then the user will be notified of the termination by a message $m(P, User, final, [(q, P, true)], true)$.*

Algorithm 2: Message passing procedure for processing queries

```
RECEIVEQUERYMESSAGE($m(Sender, Self, query, hist, q)$)
(1) if $(\bar{q}, \_, \_) \in hist$
(2)   send $m(Self, Sender, answer, [(q, Self, \Box)|hist], \Box)$
(3)   send $m(Self, Sender, final, [(q, Self, true)|hist], true)$
(4) else if $q \in Self$ or $(q, Self, \_) \in hist$
(5)   send $m(Self, Sender, final, [(q, Self, true)|hist], true)$
(6) else
(7)   LOCAL$(Self) \leftarrow \{q\} \cup Resolvent(q, Self)$
(8)   if $\Box \in$ LOCAL$(Self)$
(9)     send $m(Self, Sender, answer, [(q, Self, \Box)|hist], \Box)$
(10)    send $m(Self, Sender, final, [(q, Self, true)|hist], true)$
(11)  else
(12)    LOCAL$(Self) \leftarrow \{c \in$ LOCAL$(Self)|\ L(c) \in \mathcal{T}arget(Self)\}$
(13)    if for every $c \in$ LOCAL$(Self)$, $S(c) = \Box$
(14)      foreach $c \in$ LOCAL$(Self)$
(15)        send $m(Self, Sender, answer, [(q, Self, c)|hist], c)$
(16)      send $m(Self, Sender, final, [(q, Self, true)|hist], true)$
(17)    else
(18)      foreach $c \in$ LOCAL$(Self)$
(19)        if $S(c) = \Box$
(20)          send $m(Self, Sender, answer, [(q, Self, c)|hist], c)$
(21)        else
(22)          foreach literal $l \in S(c)$
(23)            if $l \in \mathcal{T}arget(Self)$
(24)              ANSWER$(l, [(q, Self, c)|hist]) \leftarrow \{l\}$
(25)            else
(26)              ANSWER$(l, [(q, Self, c)|hist]) \leftarrow \emptyset$
(27)            FINAL$(l, [(q, Self, c)|hist]) \leftarrow false$
(28)            foreach $RP \in$ ACQ$(l, Self)$
(29)              send $m(Self, RP, query, [(q, Self, c)|hist], l)$
```

Algorithm 3: Message passing procedure for processing answers

```
RECEIVEANSWERMESSAGE($m(Sender, Self, answer, hist, r)$)
(1) $hist$ is of the form $[(l', Sender, c'), (q, Self, c)|hist']$
(2) ANSWER$(l', hist) \leftarrow$ ANSWER $(l', hist) \cup \{r\}$
(3) RESULT$\leftarrow \circledvee_{l \in S(c)\setminus\{l'\}}$ANSWER$(l, hist) \circledvee \{L(c) \vee r\}$
(4) if $hist' = \emptyset, U \leftarrow User$ else $U \leftarrow$ the first peer $P'$ of $hist'$
(5) foreach $cs \in$ RESULT
(6)   send $m(Self, U, answer, [(q, Self, c)|hist'], cs)$
```

Algorithm 4: Message passing procedure for notifying termination

```
RECEIVEFINALMESSAGE($m(Sender, Self, final, hist, true)$)
(1) $hist$ is of the form $[(l', Sender, true), (q, Self, c)|hist']$
(2) FINAL$(l', hist) \leftarrow true$
(3) if for every $l \in S(c)$, FINAL$(l, hist) = true$
(4)   if $hist' = \emptyset\ U \leftarrow User$ else $U \leftarrow$ the first peer $P'$ of $hist'$
(5)   send $m(Self, U, final, [(q, Self, true)|hist'], true)$
(6)   foreach $l \in S(c)$
(7)     ANSWER$(l, [(l, Sender, \_), (q, Self, c)|hist']) \leftarrow \emptyset$
```

Sketch of proof: We prove by induction on the number of recursive calls of $RCFH(q, SP, \Gamma, hist)$ that:

(1) for any result r returned by $RCFH(q, SP, \Gamma, hist)$, there exists $P \in SP$ such that P is bound to send a message $m(P, S, answer, [(q, P, \_)|hist], r)$ after receiving the message $m(S, P, query, hist, q)$,

(2) if r is the last result returned by $RCFH(q, SP, \Gamma, hist)$, all the peers $P \in SP$ are bound to send the message $m(P, S, final, [(q, P, true)|hist], true)$, where S is the first peer in the history.

- $rc = 0$: either one of the conditions of Lines (1), (2), (4) or (6) of the algorithm $RCFH(q, SP, \Gamma, hist)$ is satisfied. We have shown in the proof of Theorem 2 that if the conditions of Lines (1) and (4) are satisfied, $\Box$ is the only result returned by the algorithm. The condition of Line (1) of the algorithm $RCFH(q, SP, \Gamma, hist)$ corresponds to the condition of Line (1) of the algorithm ReceiveQueryMessage($m(S, P,$ $query,$ $hist, q)$) for any P of SP, which triggers the sending of a message $m(P, S, answer, [(q, P, \Box)|hist], \Box)$ (Line (2)) and of a message $m(P, S, final, [(q, P, true)|hist], true)$ (Line(3)). If the condition of Line (4) of the algorithm $RCFH(q, SP, \Gamma, hist)$ is satisfied, since LOCAL is $\bigcup_{P \in SP}$ LOCAL(P), there exists $P \in SP$ such that $\Box \in P$. That condition corresponds to the condition of Line (8) of the algorithm for ReceiveQueryMessage($m(S, P, query, hist, q)$), which triggers the sending of a message $m(P, S, answer, [(q, P, \Box)|hist], \Box)$ (Line (9) and of a message $m(P, S, final, [(q, P, true)|hist], true)$ (Line (10)). The condition (2) of the algorithm $RCFH(q, SP, \Gamma, hist)$, in which no result is returned (see proof of Theorem 2), corresponds to the condition of Line (4) of the algorithm ReceiveQueryMessage($m(S, P, query, hist, q)$), for every $P \in SP$, which only triggers the sending of a final message (Line (5)). Finally, if the condition of Line (6) of the algorithm $RCFH(q, SP, \Gamma, hist)$ is satisfied, r is necessarily an element of LOCAL, i.e., there exists $P \in SP$ such that $r \in$ LOCAL(P). The condition of Line (6) of the algorithm $RCFH(q, SP, \Gamma, hist)$ corresponds to the condition of Line (13) in ReceiveQueryMessage($m(S, P, query, hist, q)$), which triggers the sending of all the messages $m(P, S, answer, [(q, P, c)|hist], c)$, where c is a clause of LOCAL(P) (Line (15)), and in particular the message $m(P, S, answer, [(q, P, r)|hist], r)$. It triggers too the sending of a final message (Line (16)) for P. If r is the last result returned by $RCFH(q, SP, \Gamma, hist)$, such final messages has been sent by every $P \in SP$.

- Suppose the induction hypothesis true for $rc \leq p$, and let $\Gamma = (\mathcal{P}$, ACQ) a P2PIS such that $RCFH(q, SP, \Gamma, hist)$ requires $p + 1$ recursive calls to terminate.

- If $r \in$ LOCAL (r is not the last result returned by the algorithm) there exists $P \in SP$ such that $r \in$ LOCAL(P), and r is one of the clauses c involved in the iteration of the loop of Line (18) of the algorithm ReceiveQueryMessage($m(S, P, query, hist, q)$), and verifying the condition of Line (19), which triggers the sending of the message $m(P, S, answer, [(q, P, r)|hist], r)$ (Line (20)).

-If $r \notin$ LOCAL, there exists $P \in SP$ and a clause $c : l_1 \vee \cdots \vee l_k \vee L(c)$ of LOCAL(P) such that c is involved in the iteration of the loop of Line (8) of the algorithm $RCFH(q, P, \Gamma, hist)$, and r is an element $r_1 \vee \cdots \vee r_k \vee L(c)$ of $(\circledvee_{l \in S(c)}$ANSWER$(l)) \circledvee \{L(c)\}$ computed at Line (12), where each ANSWER(l) is obtained as the result of $RCFH(l,$ ACQ$(l, P), \Gamma', [(q, P, c)|hist])$ (Line (13)), which requires less than p recursive calls. By induction, for each literal $l_i \in S(c)$, there exists $RP_i \in$ ACQ(l_i, P) such that RP_i sends a message $m(RP_i, P, answer, [(l_i, RP_i, \_), (q, P, c)|hist], r_i)$ if it has received the message $m(P, RP_i, query, [(q, P, c)|hist], l_i)$. The loop of Line (11) of the algorithm $RCFH(q, SP,$ $\Gamma,$ $hist)$ corresponds to the loop of Line (22) of the algorithm ReceiveQueryMessage($m(S, P, query, hist, q)$), which triggers the sending of the messages $m(P,$ RP_i $,$ $query,$ $[(q, P, c)|hist],$ $l_i)$ for each literal $l_i \in S(c)$ (Line (29)). Therefore, according to the induction hypothesis, for every $l_i \in S(c)$, RP_i sends a message $m(RP_i, P, answer, [(l_i, RP_i, \_), (q, P, c)|hist], r_i)$. When the last of those messages (let us say $m(RP_j, P, answer, [(l_j, RP_j, \_), (q, P, c)|hist], r_j))$ is processed, r is produced by Line (3) of ReceiveAnswerMessage($m(RP_j, P, answer, [(l_j, RP_j, \_), (q, P, c)|hist], r_j)$), and there exists a peer U such that P is bound to send to it the message $m(P, U, answer, [(q, P, c)|hist], r)$ (Line (6)).

- If r is the last result returned by the algorithm $RCFH(q, SP, \Gamma, hist)$, for every $P \in SP$, for every $c \in$ LOCAL(P), for every $l \in S(c)$, $RCFH(l,$ ACQ$(l, P), \Gamma', [(q, P, c)|hist])$ has finished, and, by induction, every peer RP of ACQ(l, P) has sent a message $m(RP, P, final, [(l, RP, true), (q, P, c)|hist], true)$. Therefore, the condition of Line (3) of the algorithm ReceiveFinalMessage($m(RP, P, final, [(l, RP, \_), (q, P, c)|hist], true)$) is satisfied, which triggers the sending of a message $m(P, U, final, [(q, P, true)|hist], true)$ (Line (5)). □

For sake of simplicity, both recursive and distributed algorithms have been presented as applying to literals. It does not mean that the queries that we consider are limited to literals. Clausal queries can be handled by splitting them into literals and then using the $\circledvee$ operator to recompose the results obtained for each literal.

4 Experiments

A P2PIS architecture has been developped in Java and deployed on a cluster of 28 Athlon 1800+ Linux machines with 1GB memory. Local implicates computation is ensured by a local version of the distributed algorithm we presented; we optimized its whole behavior by filtering as many subsumed clauses as possible (because of the distributed and anytime properties of our algorithm, not all subsumed clauses can be detected).

Benchmark generation: Evaluating distributed systems performances is not an easy task. We chose to focus on random theories because they have been extensively studied in previous (centralized) work [17] and represent real challenges for compilation: it is well known that very small theories can easily produce very large compiled theories. Our benchmark generator takes the characteristics of the acquaintance graph as an input (d nodes and e edges), and fill its nodes with uniform-random 3CNF theories, all having the same (fixed) number m of clauses and variables (n). All edges of the connected graph are labelled with a given number q of variables that both peers connected by the edge do share. In order to simply encode shared variables while ensuring that theorem 2 applies, we add two clauses in both peers to enforce the equivalence between a local and a remote variable. Thus, we have a global theory of $d.m$ random clauses of length 3, $d.n$ variables, and $4.q.e$ clauses of length 2 that encodes equivalences of shared variables. Another parameter, p, is the number of target variables of each peer.

In our experiments, we fixed $q = 1$ and $e = 1.3 * d$ (to obtain not too constrained graphs), with $d \in \{5, 10, 28\}$. We also limited our tests to small peers (less than 30 clauses). Such theories may already contains an important number of implicates [17]. State of the art algorithms [19] can solve only up to 150 clauses and 50 variables at random, wich already corresponds to 5 peers of 30 clauses.

Experimental analysis: The first observations we made while running our experiments was that short clauses usually came first and that produced clauses did contain target variables from a number of distant peers. We also observed large differences in the algorithm behavior for different queries on the same P2PIS.

We report on each line of Table 1 average values over ten P2PIS benchmarks. Each benchmark consists itself of a synthesis over a number $\#Q$ of different queries on the same P2PIS. Column $\#Q$ gives the number of asked queries, the number of queries that finished before the timeout and their computation time. The column #Imp gives the number of found implicates and its median. The last column gives the time to produce respectively the first 2, 10, 100 and 1000 answers (when applicable). Those values give clues on production speed of the algorithm. We considered a timeout of 30s for each queries, which represents a reasonable waiting time for a human asking queries to the P2PIS.

d	m	n	p	#Q (#Ended)	#Imp	Time
5	22	11	2	55 (19, 5s)	14 (7)	2,8,–,–
5	22	11	5	55 (3, 1s)	7799 (1431)	1,3,5,8
10	22	11	5	40 (4, 1s)	23132 (1651)	1,2,5,9
10	30	15	5	40 (4, 2s)	7099 (648)	3,7,13,17
28	22	11	5	112 (20, 2s)	8120 (513)	2,3,6,10
28	30	15	5	112 (7, 4s)	1132 (54)	8,13,20,24

Table 1. Results for multiple queries over different P2PIS

The first lines ($d = 5$) shows that reducing the size of the target language does not necessarily lead to performance enhancements. On one hand, each peer may ask less queries and may quickly find all results (19 queries finished in 5s) but on the other hand, the larger the target language is, the faster the first answers will come.

For the experiments with 10 peers, large differences in mean and median values for #Imp may be explained by a very disparate distribution, which emphasizes that the same P2PIS behaves in very different ways depending on the query. We can see that even if only 10% of the queries end, the production speed of the algorithm is good (1000 production clauses in less that 10s for $m = 22$). When the complexity of local theories grows ($m = 30$), the effect is a slow-down of the production speed.

At last, we tested the scalability of our approach by deploying the architecture over the 28 nodes of the cluster, with different theory sizes. Even if only few queries end within the timeout, our algorithm scales up well, and quickly produces a large number of target clauses. We see that when local theories are harder, first results still come in a reasonable amount of time. Note that `zres` [19] could not process within one hour the global theories (988 clauses, 420 variables) corresponding to the union of the local theories of these benchmarks.

5 Related work

The message passing distributed algorithm that we have described in Section 3 may be viewed as a distributed version of an Ordered Linear deduction [2] to produce new target clauses, which was extended by [18] in order to produce all implicates of a given clause belonging to some target language, and further extended to the first order case by Inoue in [9]. New derived clauses (the "proper implicates" in section 2) computation has already been extensively studied (see [11] for a survey). In particular, this problem, also known as Φ-implicates computation has been addressed in [18, 9] and in [10].

We have already pointed out the differences between our work and [1]. In a peer-to-peer setting, using tree-decomposition of the acquaintance graph is not possible, but we can apply our algorithm to partitioned theories in place of the one of [1]. It may benefit from the theory tree-decomposition to speed-up results. As we have shown in the introducing example, the algorithm of [1] requires to be complete that the global target language is the union of the local target languages. [6] relies on that observation in order to encode a P2PIS with peer/super-peers topology into a partitioned propositional theory and to use the consequence finding algorithm of [1]. The global knowledge on the target variables of the whole P2PIS must be known and is distributed among the super-peers. The model-based diagnosis framework for distributed embedded systems [16] is based on [1]. We think it can benefit from our approach to apply to a real peer-to-peer setting in which no global knowledge has to be shared.

In distributed ATMS [12], agents exchange nogood sets in order to converge to a globally consistent set of justifications. In contrast with a peer-to-peer vision, such a distributed vision of ATMS relies on a global knowledge shared by all the agents and aims at getting a unique global solution.

6 Conclusion

The contributions of this paper are both theoretical and practical. We have provided the first distributed consequence finding algorithm in a peer-to-peer setting, and we have exhibited a sufficient condition for its completeness. We have developped a P2PIS architecture that implements this algorithm and for which the first experimental results look promising. This architecture is used in a joint project with France Télécom, which aims at enriching peer-to-peer web applications with reasoning services (e.g., Someone [15]).

So far, we have restricted our algorithm to deal with a vocabulary-based target language. However, it can be adapted to more sophisticated target languages (e.g., implicates of a given, maximal, length, language based on literals and not only variables). This can be done by adding a simple tag over all messages to encode the desired target language. Another possible extension of our algorithm is to allow more compact representation of implicates, as it is done in [19]. That work relies on an efficient clause-distribution operator. It can be adapted by extending messages in our algorithm in order to send *compressed* sets of clauses instead of one clause as it is the case right now, without changing the deep architecture of our algorithm.

REFERENCES

[1] E. Amir and S. McIlraith, 'Partition-based logical reasoning', in *KR'00*.

[2] C. L. Chang and R. C. Lee, *Symbolic Logic and Mechanical Theorem Proving*, Computer Science Classics, Academic Press, 1973.

[3] R. Dechter and I. Rish, 'Directed resolution: the davis-putnam procedure revisited', in *KR'94*.

[4] A. del Val, 'A new method for consequence finding and compilation in restricted languages', in *AAAI'99*.

[5] Gnutella. http://gnutella.wego.com.

[6] F. Goasdoue and M.-C. Rousset, 'Querying distributed data through distributed ontologies: a simple but scalable approach', *IEEE Intelligent Systems*, (18), (2003).

[7] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov, 'Schema mediation in peer data management systems', in *ICDE'03*.

[8] A. Halevy, Z. Ives, I. Tatarinov, and Peter Mork, 'Piazza: data management infrastructure for semantic web applications', in *WWW'03*.

[9] K. Inoue, 'Linear resolution for consequence finding', *Artificial Intelligence*, (56), (1992).

[10] A. Kean and G. Tsiknis, 'An incremental method for generating prime implicants/implicates', *Journal of Symbolic Computation*, **9**, (1990).

[11] P. Marquis, *Handbook of Defeasible Reasoning and Uncertainty Management Systems*, volume 5, chapter Consequence Finding Algorithms, Kluwer Academic Publishers, 1999.

[12] C.L. Mason and R.R. Johnson, *Distributed Artificial Intelligence II*, chapter DATMS: a framework for distributed assumption based reasoning, Pitman, 1989.

[13] Napster. http://www.napster.com.

[14] W. Nedjl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch, 'Edutella: a p2p networking infrastructure based on rdf', in *WWW'02*.

[15] M. Plu, P. Bellec, L. Agosto, and W. van de Velde, 'The web of people: A dual view on the WWW', in *Int. World Wide Web Conf.*, (2003).

[16] G. Provan, 'A model-based diagnosis framework for distributed embedded systems', in *KR'02*.

[17] R. Schrag and J. Crawford, 'Implicates and prime implicates in random 3-sat', *Artificial Intelligence*, **81**, (1996).

[18] P. Siegel, *Représentation et utilisation de la connaissance en calcul propositionnel*, Ph.D. dissertation, Université d'Aix-Marseille II, 1987.

[19] L. Simon and A. del Val, 'Efficient consequence finding', in *IJCAI'01*.