CLProver++: An Ordered Resolution Prover for Coalition Logic

Ullrich Hustadt¹  Paul Gainer¹  Clare Dixon¹  Cláudia Nalon²  Lan Zhang³

1 Department of Computer Science, University of Liverpool
   {uhustadt,pgainer,cl Dixon}@liverpool.ac.uk
2 Department of Computer Science, University of Brasilia
   nalon@unb.br
3 Information School, Capital University of Economics and Business, China
   lan@cueb.edu.cn

Abstract: We present CLProver++, a theorem prover for Coalition Logic a non-normal modal logic for
reasoning about cooperative agency. CLProver++ is based on recent work on an ordered resolution calculus
for Coalition Logic. We provide an overview of this calculus, give some details of the implementation of
CLProver++ and present an evaluation of the performance of CLProver++.

1 Introduction

Coalition Logic CL was introduced by Pauly [5] as a logic for reasoning about what groups of agents can bring about
by collective action. CL is a propositional modal logic over a countably infinite set Π of propositional symbols and a
non-empty, finite set Σ ⊆ N of agents with modal operators of the form [A], where A ⊆ Σ. The formula [A]φ, where
A is a set of agents and φ is a formula, can be read as the coalition of agents A can bring about φ. Formally,
the semantics of CL formulae is given by Concurrent Game Models (CGMs), see [3]. The satisfiability problem of CL
is PSPACE-complete [5]. Various decision procedures for Coalition Logic exist including tableaux and unrefined res-
olution calculi.

In the following we present a new decision procedure for CL based on ordered resolution, briefly describe its imple-
mentation and present its evaluation.

2 Ordered Resolution for CL

Our ordered resolution calculus does not operate on CL formulae, but on formulae of Vector Coalition Logic (VCL) in a
clausal normal form.

Let Σ = k. A coalition vector is a k-tuple such that for every a, 1 ≤ a ≤ k, c[a] is either an integer number not equal to zero or the symbol ∗ and for every a, a', 1 ≤ a < a' ≤ k, if c[a] < 0 and c[a'] < 0 then c[a] = c[a'].

The set WFFVCL of Vector Coalition Logic (VCL) formulae is inductively defined as follows: (i) if p is a proposi-
tional symbol in Π, then p and ¬p are VCL formulae; (ii) if φ is a propositional formula and ψ is a VCL formula,
then (φ → ψ) is a VCL formula; (iii) if ψi, 1 ≤ i ≤ n, n ∈ N, are VCL formulae, then so are (ψ1 ∧ . . . ∧ ψn), also
written n i=1 ψi, and (ψ1 ∨ . . . ∨ ψn), also written n i=1 ψi; and (iv) if c is a coalition vector and φ is a VCL formula,
then so is cφ. The semantics of WFFVCL formulae is given by Concurrent Game Models extended with choice
functions (CGM∗) that give meaning to coalition vectors.

Intuitively, a coalition vector represents the choices made by each agent. Each number represents a choice function
that selects an agent's move (action) depending on the current world and, possibly, the moves of other agents.

A coalition problem in DSNF VCL is a tuple (I, U, N) such that I is a set of initial clauses, and U is a set of global
clauses, are finite sets of propositional clauses Vm1=1 l, and N, the set of coalition clauses, consists of VCL formulae
of the form n i=1 l′ i → c with m, n ≥ 0 and l′ i, l, for all 1 ≤ i ≤ m, 1 ≤ j ≤ n, are literals such that within
every conjunction and every disjunction literals are pairwise different, and a vector is a coalition vector.

Intuitively, initial clauses are true at one distinguished world in a CGM∗ while global and coalition clauses are
true at every world in a CGM∗, the later imposing a constraint Vm1=1 l on the worlds that a coalition can 'reach' from
a world w by its actions, provided the condition m i=1 l′ i is satisfied at the world w.

There are two more ingredients to our calculus that we need to introduce, the notion of the merge of two coalition
vectors and the notion of atom orderings.

Let  and  be two coalition vectors of length k. The coalition vector  is an instance of  and  is more
general than , written , if  = [i] for every i, 1 ≤ i ≤ k, with [i] ≠ ∗. We say that a coalition vector  is a common instance of  and  if  is an instance of both  and . A coalition vector  is a merge of  and, denoted  or , if  is a common instance of  and , and for any common instance  of  and we have ⊑ . If there exists a merge for two coalition vectors  and  then we say that  and  are mergeable.

An atom ordering is a well-founded and total ordering ⊢ on the set Π. The ordering ⊢ is extended to literals such
that for each p ∈ Π, ¬p ⊢ p, and for each q ∈ Π such that q ⊢ p then q ⊢ ¬p and ¬q ⊢ ¬p. A literal l is maximal
with respect to a propositional disjunction C iff for every literal l' in C, l' ≠ l.

The ordered resolution calculus RESCL is then given by the rules shown in Figure 1.

Theorem 1 Let ϕ be a CL formula. Then there is a coalition problem C in DSNF VCL that is satisfiable if and only
IRES1
\[
\begin{array}{c}
C \lor l \in \mathcal{I} \\
D \lor \neg l \in \mathcal{I} \\
C \lor D \in \mathcal{I}
\end{array}
\]
GRES1
\[
\begin{array}{c}
C \lor l \in \mathcal{U} \\
D \lor \neg l \in \mathcal{U} \\
C \lor D \in \mathcal{U}
\end{array}
\]
VRES1
\[
\begin{array}{c}
P \rightarrow \tau_1(C \lor l) \in \mathcal{N} \\
Q \rightarrow \tau_2(D \lor \neg l) \in \mathcal{N} \\
P \land Q \rightarrow \tau_1 \land \tau_2(C \lor D) \in \mathcal{N}
\end{array}
\]
VRES2
\[
\begin{array}{c}
Q \rightarrow \tau(C \lor D) \in \mathcal{N} \\
\bigwedge_{i=1}^n l_i \rightarrow \tau_{\text{false}} \in \mathcal{N}
\end{array}
\]
RW
\[
\begin{array}{c}
\forall l \in \mathcal{U} \\
\exists \exists \tau \in \mathcal{N}
\end{array}
\]
where \((\mathcal{I}, \mathcal{U}, \mathcal{N})\) is a coalition problem in DSNF\(_{\text{CL}}\); \(P\), \(Q\) are conjunctions of literals; \(C\), \(D\) are disjunctions of literals; \(l\), \(l_i\) are literals; \(\tau\), \(\tau_1\), \(\tau_2\) are coalition vectors; in VRES1, \(\tau_1\) and \(\tau_2\) are mergeable; and in IRES1, GRES1, VRES1 and VRES2, \(l\) is maximal with respect to \(C\) and \(\neg l\) is maximal with respect to \(D\).

Figure 1: Resolution Calculus RES\(_{\text{CL}}\)

if \(\varphi\) is satisfiable. Furthermore, any derivation by RES\(_{\text{CL}}\) from \(C\) terminates and \(\varphi\) is unsatisfiable if and only if there is a refutation of \(C\) by RES\(_{\text{CL}}\).

3 CLProver++

CLProver++ [2] is a C++ implementation of the resolution based calculus RES\(_{\text{CL}}\) described in Section 2. CLProver++ also implements unit propagation, pure literal elimination, forward subsumption and backward subsumption. Clauses in a coalition problem are split into a set \(W_0\) of worked-off clauses and set \(U_0\) of usable clauses. The main loop of the prover heuristically selects a clause \(G\) from \(U_0\), moves it to \(W_0\) and performs all inferences between \(G\) and clauses in \(W_0\). The set \(N_0\) of newly derived clauses is subject to forward subsumption and the remaining clauses in \(N_0\) may optionally be used to backward subsume clauses in \(U_0\) and \(W_0\). Feature vector indexing [6], a non-perfect indexing method, is used to store \(U_0\) and \(W_0\), and to retrieve a superset of candidates for subsumption or resolution efficiently.

To evaluate the performance of CLProver++ we have compared it with CLProver and TATL (September 2014 version). CLProver [4] is a prototype implementation in SWI-Prolog of the calculus RES\(_{\text{CL}}\). It also implements forward subsumption but uses no heuristics to guide the search for a refutation. TATL [1] is an implementation in OCaml of the two-phase tableau calculus by Goranko and Shkatov for ATL [3], that can also be used to decide the satisfiability of CL formulae.

We have used two classes \(B_1\) and \(B_2\) of randomly generated CL formulae for the evaluation that are available from the CLProver++ website [2]. \(B_1\) consists of twelve sets \(S_i\), \(1 \leq i \leq 12\), of 100 formulae each, with each formula in \(S_i\) having length 100 \(\times\) \(i\). \(B_2\) consists of 12 sets \(S_i\), \(1 \leq i \leq 12\) of 100 formulae in conjunctive normal form with \(i\) conjuncts of the form \((-)[A_1](l_1 \lor l_2) \land ((-) [A_2](l_3 \lor l_4) \lor (-))[A_3](l_5 \lor l_6))\) with elements of each conjunct generated randomly.

Figures 2 and 3 show the total runtime of each of the provers on each of the sets in \(B_1\) and \(B_2\), respectively. Execution of a prover on a formula was stopped after 1000 CPU seconds. The time to transform a formula into a coalition problem is not included, but is negligible. Overall, CLProver++ outperforms all other systems by a large margin.

References


