Formally Verified Constraints Solvers
A Guided Tour

Catherine Dubois
ENSIIE & Samovar
catherine.dubois@ensiie.fr
https://www.ensiie.fr/~dubois

Introduction

Constraint solvers are complex tools implementing tricky algorithms and heuristics manipulating intricate data structures. It is well-known that they have bugs. Certifying the output of such tools is extremely important in particular when they are used for critical systems or in verification tools. There are mainly two ways for having confidence in the computed results: making the solver produce not only the output but also proof logs that can be easily verified by an external checker or proving the correctness of the solver itself. The former approach is widespread in the Boolean satisfiability community through formats such as DRAT which can be considered as a standard. The latter approach has been followed for example for developing Compcert and sel4 respectively a C compiler developed and formally verified with the help of the Coq proof assistant Coq and a micro-kernel developed and formally verified with the proof assistant Isabelle/HOL.

Main Objectives

- Develop a family of formally verified constraints solver (for finite domains)
- Formalize and understand deeply algorithms and results used in CP

Constraint Satisfaction Problem (CSP)

Definition of a CSP

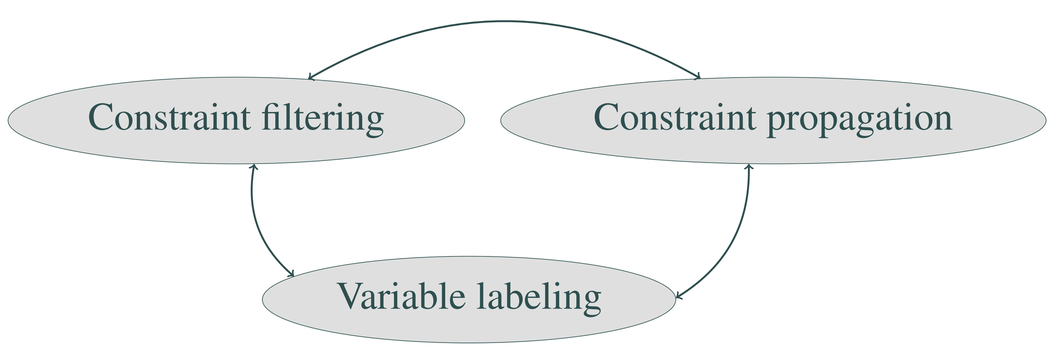
A CSP is a triple (X, D, C) where
- X: a set of variables,
- D: a function that maps each variable of X to its domain **here finite** set of possible values),
- C: a set of constraints (relations btw variables) over variables of X,
  arity of a constraint = number of its variables.

A solution of (X, C, D) is a valid (compatible with D) assignment defined for all the variables in X that satisfies all the constraints in C. A CSP is unsatisfiable (UNSAT) when it has no solution.

Solving can mean finding one solution or all solutions, showing UNSAT, completing a solution, finding the optimal solution, etc.

CSP Solving

Main idea of CP(FD) solving algorithms = repeatedly removing inconsistent values from the domains through 3 interleaved processes



Maintain/enforce a local consistency property during search

Many local consistency properties: arc-consistency (AC), generalized arc-consistency (GAC), path consistency, bound-consistency, etc.

CoqbinFD: a formally verified constraints CP(FD) solver

CoqbinFD has been developed with the Coq interactive proof assistant, proved sound and complete $\longrightarrow$ it yields a correct solution when there is at least one, answers UNSAT when no solution;

CoqbinFD is generic, parametrized by the language of constraints itself;

CoqbinFD only deals with binary constraint;

CoqbinFD is implementing a classical algorithm AC3 (Mackworth 77) and AC2001 (Bessières, Regin, Zhang 05) (at the heart of main existing solvers), focusing on arc-consistency; smallskip

CoqbinFD is written in OCaml, extracted from Coq;

CoqbinFD has been easily adapted to bound-consistency.

For more details see [1] and http://www.ensiie.fr/~dubois/CoqsolverFD/

From non-binary CSPs to binary CSPs

A non-binary CSP can be solved by translating/encoding it into an equivalent binary one and using well-established binary CSP techniques or using extended versions of binary techniques directly on the non-binary problem (usually the recent case, however still research interest for encodings).

One of the most popular encodings is the hidden variable encoding (HVE) (1990).

original CSP		HVE Encoding
(X, D, C)	$\rightsquigarrow$	(X', D', C')
$x \in X$	$\rightsquigarrow$	$x \in X'$ Ordinary variable
	$\rightsquigarrow$	$D'(x) = D(x)$
binary constraint c	$\rightsquigarrow$	c
n-ary constraint c' $(n > 2)$	$\rightsquigarrow$	$v_{c'} \in X'$ Hidden variable
$vars(v'_c) = \{x_1, \ldots, x_n\}$	$\rightsquigarrow$	$D'(v_{c'}) = D(x_1) \times D(x_2) \ldots D(x_n)$
	$\rightsquigarrow$	$proj_i(v_{c'}, x_i) \in C', i \in [1..n]$

Our work:
- Formalisation in Coq of the hidden variable encoding, proof of its correctness;
- Extension of CoqbinFD as a formally verified CP(FD) solver for both binary and non-binary constraints.

See [2] and www.ensiie.fr/~dubois/HVE_nary for more details.

Domains as interval lists

In existing solvers, the main domain representations are range sequences (Gecode, ECLiPSe Prolog, SICStus Prolog, FaCiLe) implemented as lists or binary trees, gap intervals trees (Naxos Solver), bit vectors (ILOG Solver, Choco), successor vectors, sparse sets (OscaR, Abscon, Castor) .

Focus is put on domains represented as minimal sorted sequences of ranges.
Example : the domain $\{10, 2, 4, 5, 7, 1, 11, 3\}$ is represented by the list $[(1, 5); (7, 7); (10, 11)]$.

Our contribution:
- Formalisation in Coq of range sequences;
- Partial verification of FaCiLe (an OCaml constraints library);
- Extension to provide a new implementation of finite sets (Coq FSet interface).

From this formal development, we are deriving a Coq formalisation of gap interval lists (work in progress with A. Ledein).

See [4] and https://gitlab.com/finite_set_Coq/real_intervals_list for more details.

AllDifferent global constraint

AllDifferent$(x_1, \ldots x_n)$ means that all variables $x_1 \ldots x_n$ must be pairwise different.
It can be solved by decomposing it into a set of binary inequalities : $\bigcup_{1 \leq i < j \leq n}\{x_i \neq x_j\}$ (loss in filtering level) or using a dedicated filtering/propagation algorithm (e.g. Regin 94).

Following Regin's algorithm, a solution of AllDifferent$(x_1, \ldots x_n)$ is a matching (of the value graph) that covers $V = \{x_1, \ldots x_n\}$.
Enforcing GAC means removing all the edges that cannot be in any matching covering V.

Work in progress: formalisation in Coq of the first step of the Regin's filtering algorithm, ie computation of a maximal matching in the graph of values

Approach 1: verification a posteriori $\longrightarrow$ development of a verified checker [3]

The maximal matching algorithm $\mathcal{M}$ is written in OCaml, it returns a matching and a vertex cover (witness).
$\longrightarrow$ The algorithm $\mathcal{M}$ is not verified in Coq, considered as untrusted.
The checker verifies that the matching and the vertex cover are 2 sets with the same cardinality, and thus it is a maximal matching.
$\longrightarrow$ The checker is developed in Coq, proved correct and extracted as an OCaml executable code.

Approach 2: Verification of a functional algorithm in Coq

The algorithm computes alternating paths, augmenting paths, transfer functions etc.
The proof of its correctness relies on the Berge's theorem (1957) that we are formalizing in Coq: *a matching m in a graph is maximum iff there is no augmenting path for m in g.*

Conclusion and future work

We have developed and verified in Coq a binary constraints solver for finite domains and some variants of it. Some challenges remain.

Efficiency. Efficiency of the extracted solver CoqbinFD is reasonable but we cannot compete with the existing solvers!
We expect it to serve as a reference implementation but some efforts are still needed.
$\longrightarrow$ More efficient data structures for variables, domains, queues, etc.
$\longrightarrow$ Efficient dedicated filters/propagators (but loss of genericity!)
$\longrightarrow$ Imperative data structures (refinement, primitive arrays)

Modularity. CoqbinFD has a too rigid structure: it is difficult to plug-in new domain representations, new local consistencies,…

$\longrightarrow$ Introduce general interfaces, generic parameters, ... yes but there are optional/mandatory features, implementation constraints, etc.
$\longrightarrow$ Define a Software Product Line of **verified** CP solvers

Reuse. For AllDifferent, and many others global contraints, a large background about graph theory (matchings, network flows, relaxation methods, …) is needed but …there is no large Coq library about graphs and operational research.

$\longrightarrow$ Big effort is needed
$\longrightarrow$ Large libraries are needed and must be shared between provers
$\longrightarrow$ Interoperability of provers, encyclopedia of proofs

References

[1] Matthieu Carlier, Catherine Dubois, and Arnaud Gotlieb. A Certified Constraint Solver over Finite Domains. In *Formal Methods (FM)*, volume 7436 of *LNCS*, pages 116–131, Paris, 2012.

[2] Catherine Dubois. Formally verified decomposition of non-binary constraints into binary constraints. In *Workshop on Functional and Logic Programming*, 2020.

[3] Catherine Dubois, Sourour Elloumi, Benoit Robillard, and Clément Vincent. Graphes et couplages en coq. In *Journées Francophones des Langages Applicatifs*, January 2015.

[4] Amélie Ledein and Catherine Dubois. Facile en coq : vérification formelle des listes d'intervalles. In *Journées Francophones des Langages Applicatifs*, January 2020.

Acknowledgements

This is joint work with A. Butant (ENSIIE 2016), M. Carlier, V. Clément (ENSIIE 2014), S. Elloumi (ENSTA), A. Gotlieb (Simula, Norway), A. Ledein (ENSIIE 2020) and H. Mlodecki (ENSIIE 2017). Many thanks to them!