Modular Constraint Solver Cooperation via Abstract Interpretation

ICLP 2020

Pierre Talbot, Éric Monfroy, Charlotte Truchet
{pierre.talbot}@uni.lu

University of Luxembourg

22nd September 2020

UNIVERSITÉ DU LUXEMBOURG

UNIVERSITÉ DE NANTES

université angers

Introduction

Many research communities centered around solving techniques and constraint languages:

- SAT solving: propositional formulas,
- Linear programming: linear relations either on real numbers, integers, or both (mixed),
- Constraint programming: Boolean and arithmetic constraints with specialized predicates (global constraints),
- Answer set programming: Horn clauses w/o functions (initially),
- ...

Even larger if we consider heuristics approaches such as genetic algorithms, evolutionary algorithms or local search.

The challenge

- Each field has developed its own theory and terminology.
- Pro: Very specialized and efficient on their constraint languages.
- Drawback: Hard to transfer knowledge from one field to another.

The challenge

- Each field has developed its own theory and terminology.
- Pro: Very specialized and efficient on their constraint languages.
- Drawback: Hard to transfer knowledge from one field to another.

The overarching project

Take a step back, and try to find a unified theory.

Candidate theory: abstract interpretation

Abstract interpretation is a framework to statically analyse programs and catch bugs (*Cousot and Cousot, 1977*).

Interesting features for constraint reasoning

- Mathematical background on lattice theory.
- Abstract domains are lattices with operators encapsulating a constraint language.
- Product of domains to combine several abstract domains, thus constraint solving techniques.
- Under-approximation and over-approximation to characterize the solutions of an abstract element (soundness and completeness).

Context

AbSolute: A constraint solver written in OCaml to experiment our ideas.

- 2013: Constraint solver with linear programming, constraint programming, temporal reasoning (*Pelleau and al., 2013*).
  - Mostly over continuous domains, using over-approximations.
  - Cartesian product among abstract domains.
- 2019: Under-approximation and discrete constraint solving, with logical combination of abstract domains (*Talbot and al., 2019*).

This work

Focus on domain transformers: abstract domains parametrized by other abstract domains.

Contributions

- Two domain transformers to combine abstract domains *sharing variables*.
    1. Interval propagators completion: Arithmetic constraints over product of domains.
    2. Delayed product: Exchange of over-approximations among abstract domains.
- *Shared product* to combine domain transformers.

Plan

- Introduction
- Abstract interpretation for constraint reasoning
- Domain transformers to combine domains
- Conclusion

Abstract interpretation in a nutshell

$$
\begin{tikzcd}
 & \Phi \arrow[dl, "{[\![\,]\!]^\sharp}"'] \arrow[dr, "{[\![\,]\!]^\flat}"] & \\
A^\sharp \arrow[rr, "\gamma"] & & C^\flat
\end{tikzcd}
$$

An example

$$\begin{tikzcd}[column sep=small]
& x > 2.25 \wedge x < 2.75 \in \Phi \arrow[dl, "\llbracket\rrbracket^{\sharp}"'] \arrow[dr, "\llbracket\rrbracket^{\flat}"] & \\
\mathbb{F} \times \mathbb{F} \arrow[rr, "\gamma"] & & \mathcal{P}(\mathbb{R})
\end{tikzcd}$$

Over-approximation

$$
\begin{tikzcd}
 & x > 2.25 \land x < 2.75 \in \Phi \arrow[dl, "\llbracket\rrbracket^\sharp"'] \arrow[dr, "\llbracket\rrbracket^\flat"] & \\
{[2.25..2.75]} \arrow[rr, "\gamma"] & & \{x \in \mathbb{R} \mid x > 2.25 \land x < 2.75\}
\end{tikzcd}
$$

Under-approximation

$$
\begin{tikzcd}
& x > 2.25 \wedge x < 2.75 \in \Phi \arrow[dl, "\llbracket\rrbracket^\sharp"'] \arrow[dr, "\llbracket\rrbracket^\flat"] & \\
{[2.375..2.625]} \arrow[rr, "\gamma"] & & \{x \in \mathbb{R} \mid x > 2.25 \wedge x < 2.75\}
\end{tikzcd}
$$

Abstract domain for constraint reasoning

Lattice $\langle A, \leq \rangle$ representable in a machine where:

- $\leq$ is the order, where $a \leq b$ if b "contains more information than" a,
- $\bot$ is the smallest element, $\sqcup$ the join, ...
- $[\![.]\!]^\sharp : \Phi \to A$ and $\gamma : A \to C^\flat$,
- $closure : A \to A$ to refine an abstract element,
- $split : A \to \mathcal{P}(A)$ to divide an element into sub-elements,
- $state : A \to \{true, false, unknown\}$ to retreive the "solving state" of an element.

Solver by abstract interpretation

A solver by abstract interpretation, with A an abstract domain:

$\texttt{solve}(a \in A)$
$a \leftarrow \texttt{closure(a)}$
if $\texttt{state(a)} = \texttt{true}$ **then**
    return $\{a\}$
else if $\texttt{state(a)} = \texttt{false}$ **then**
    return $\{\}$
else
    $\langle a_1, \ldots, a_n \rangle \leftarrow \texttt{split(a)}$
    return $\bigcup_{i=0}^{n} \texttt{solve}(a_i)$
end if

We call $\texttt{solve}(\llbracket\varphi\rrbracket^\sharp)$ to obtain the solutions of the formula φ.

Plan

- Introduction
- Abstract interpretation for constraint reasoning
- Domain transformers to combine domains
- Conclusion

Direct product: combination of abstract domains

Consider the formula $\varphi \triangleq x > 4 \wedge x < 7 \wedge y + z \leq 4$.

- $x > 4 \wedge x < 7$ can be treated in the box abstract domain *Box*,
- $y + z \leq 4$ can be treated in the octagon abstract domain *Octagon*.

Solution: Rely on the direct product $\mathit{Box} \times \mathit{Octagon}$.

Direct product: combination of abstract domains

Consider the formula $\varphi \triangleq x > 4 \land x < 7 \land y + z \leq 4$.

- $x > 4 \land x < 7$ can be treated in the box abstract domain *Box*,
- $y + z \leq 4$ can be treated in the octagon abstract domain *Octagon*.

Solution: Rely on the direct product $\mathit{Box} \times \mathit{Octagon}$.

Direct product

$\langle A_1 \times \ldots \times A_n, \leq \rangle$ is an abstract domain where each operator is defined coordinatewise:

- $(a_1, \ldots, a_n) \leq (b_1, \ldots, b_n) \Leftrightarrow \bigwedge_{1 \leq i \leq n} a_i \leq_i b_i$
- $\gamma((a_1, \ldots, a_n)) \triangleq \bigcup_{1 \leq i \leq n} \gamma_i(a_i)$
- $\mathit{closure}((a_1, \ldots, a_n)) \triangleq (\mathit{closure}_1(a_1), \ldots, \mathit{closure}_n(a_n))$
- ...

Issue: domains do not exchange information.

Interval propagators completion

Consider the constraint $\varphi \triangleq x > 1 \land x + y + z \leq 5 \land y - z \leq 3$.

- $x > 1$ can be interpreted in boxes,
- $y - z \leq 3$ in octagons,
- but $x + y + z \leq 5$ is too general for any of these two...
- ...and it shares its variables with the other two.

Solution: Use the notion of *propagator functions* to connect variables between abstract domains.

Interval propagators completion

Example: Propagator $x \geq y$

We assume a projection function $project : A \times Vars \rightarrow I$, $project(a, x) = [x_\ell..x_u]$ and $project(a, y) = [y_\ell..y_u]$:

$$[\![x \geq y]\!] = \lambda a.a \sqcup_A [\![x \geq y_\ell]\!]_A \sqcup_A [\![y \leq x_u]\!]_A$$

- $IPC(A) = A \times \mathcal{P}(Prop)$ is a domain transformer equipping A with propagators,
- We can rely on $IPC(Box \times Octagon)$ with a propagator for $x + y + z \leq 5$,
- The bound constraints will automatically be exchanged between both domains thanks to the propagator.

Delayed product

IPC exchanges bound constraints, can we do better?

- $\varphi \triangleq x > 1 \land x + y + z \leq 5 \land y - z \leq 3$.
- **Observation**: When x is instantiated in $x + y + z \leq 5$, we can transfer the constraint in octagons.
- We have the *delayed product* $DP(A_1, A_2)$ to transfer instantiated constraints from A_1 into a more specialized abstract domain A_2.
- For instance, consider the abstract domain $DP(IPC(Box \times Octagon), Octagon)$, whenever $x = 3$, we can transfer $3 + y + z \leq 5$ into the octagon.

Delayed product (improved closure)

Even better?

- $\varphi \triangleq x > 1 \land x + y + z \leq 5 \land y - z \leq 3$.
- **Observation**: We can transfer *over-approximations* of $x + y + z \leq 5$ in octagons.
- For instance, if $x = [1..3]$, we can transfer $1 + y + z \leq 5 \Leftrightarrow y + z \leq 4$ into the octagon.
- A solution of $y + z \leq 4$ will also be a solution of $x + y + z \leq 5$, since x must be at least equal to 1.
- Formally: $\gamma(a \sqcup [\![x + y + z \leq 5]\!]^\sharp) \subseteq \gamma(a \sqcup [\![y + z \leq 4]\!]^\sharp)$.

Shared product

- Domain transformers combine abstract domain.
- How to combine domain transformers? Especially when they share sub-domains.

Solution: Shared product

- A "top-level" product combining domain transformers and abstract domains.
- Merge the shared sub-domains in domain transformers using the join $\sqcup$.

Application

We experimented on the flexible job-shop scheduling problem.

- Temporal constraints of the form $x + y \leq d$ (with 3 variables).
- We can treat most of the constraints in $IPC(Box \times Octagon)$.
- Over-approximations can be sent in octagons for better efficiency.

Results

- Competitive w.r.t. state of the art (Chuffed) on set of instances with few machines.
- Our goal is not (yet) to beat benchmarks, but to prove the feasibility of our approach.

Plan

- Introduction
- Abstract interpretation for constraint reasoning
- Domain transformers to combine domains
- Conclusion

Related work

- Satisfiability modulo theories (SMT)
  - Focus on logical properties, abstract domains focus more on semantics and modularity.
  - Nelson-Oppen is a fixed cooperation scheme, we can run several cooperation schemes concurrently.
- Abstract Conflict Driven Learning (*D'Silva et al., 2013*).
  - Very nice theoretical framework to integrate solving and abstract interpretation.
  - Still a big gap between theory and practice.
- $\mathcal{TOY}$ (*Estévez-Martín et al., 2009*): notion of bridges among variables, subsumed by *IPC* in our framework.

We aim to reduce the gap between practice and theory.

Conclusion

- Constraint solver = abstract domain.
- Cooperation scheme = domain transformer.
- We show two cooperation schemes (*IPC* and *DP*).
- The shared product allows us to use several cooperation schemes concurrently and in a modular way.

`github.com/ptal/AbSolute/tree/iclp2020`