Combining Constraint Languages via Abstract Interpretation

ICTAI 2019

Pierre Talbot, David Cachera, Éric Monfroy, Charlotte Truchet
{pierre.talbot}@univ-nantes.fr

University of Nantes

4th November 2019

UNIVERSITÉ DE NANTES

Introduction

Context: Project AbSolute (Pelleau and al., 2013)

- Mixed constraint solver on integer and real numbers in OCaml.
- Based on abstract interpretation (especially *abstract domains*) and constraint programming (CP).
- Abstract domains = partially ordered sets with operators.

- CP $\cup$ abstract domains $\cup$ reduced products.
- Abstract interpretation $\setminus$ widenings $\cup$ backtracking.

Constraint programming

Constraint satisfaction problem (CSP)

A CSP is a pair $\langle d, C\rangle$, example :

$$\langle\{x \mapsto \{1,2,3\}, y \mapsto \{1,2,3\}\}, \{x > y, x \neq 2\}\rangle$$

A solution is $\{x \mapsto 3, y \mapsto 1\}$.

Classic solver VS solver by abstract interpretation

A classic solver in CP:

$\texttt{solve}(\langle d, C\rangle)$
$\langle d', C\rangle \leftarrow \texttt{propagate}(\langle d, C\rangle)$
if $d' = \{a\}$ **then**
    return $\{a\}$
else if $d' = \{\}$ **then**
    return $\{\}$
else
    $\langle d_1, \ldots, d_n\rangle \leftarrow \texttt{branch}(d')$
    return $\bigcup_{i=0}^{n} \texttt{solve}(\langle d_i, C\rangle)$
end if

Classic solver VS solver by abstract interpretation

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

$\texttt{solve}(a \in Abs)$
$a \leftarrow \texttt{closure(a)}$
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

Conservative extension: We encapsulate a CSP in an abstract domain.

Research question

Global constraints are crucial to efficiently solve CSP but:

- There are a lot (> 400).
- Most of these are very specialized.
- Only a small subset is implemented in mainstream solvers.

We should think about more general methods.
We propose to rely on **abstract domains**.

Global constraints VS abstract domains

- Global constraints: capture a sub-structure + efficient solving algorithm for this structure.
- Abstract domain for CP:
  - Exact representation, or by over-approximation of a constraint language.
  - Partially ordered set equipped with several operations (consistency, entailment, join, ...).
- Various **discrete and continuous** abstract domains: interval, octagon, polyhedra, etc.
- **Combination and transformators** over these domains: reduced product, reduced product by reification, partitioning, etc.

Contributions

- Adapt integer octagon abstract domain to CP.
- Design of a **generic reduced product** where domains communicate through equivalence constraints.
- We are guided by a scheduling application: *Resource-constrained project scheduling problem* (RCPSP, RCPSP/max).

Decompose global constraints into abstract domains.

Plan

- Introduction
- Scheduling problem RCPSP
- Abstract domains for RCPSP
- Conclusion

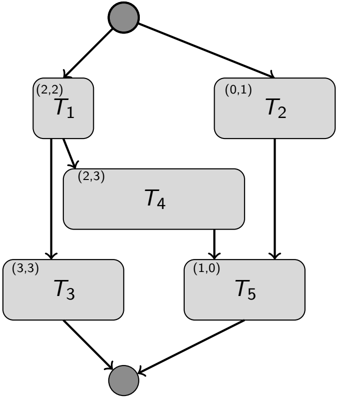
Scheduling problem RCPSP

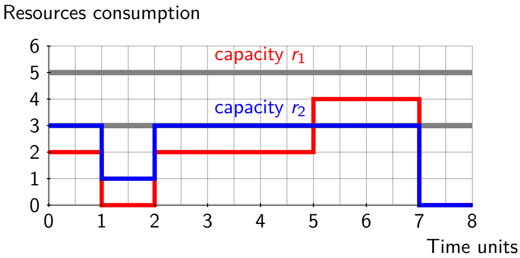
NP-complete optimisation problem:

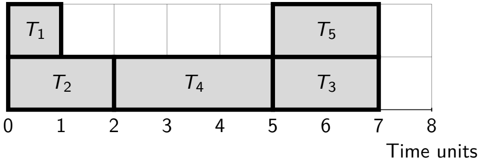
- T is a set of tasks, $d_i \in \mathbb{N}$ the duration of task i.
- P are the precedences among tasks: $i \ll j \in P$ if i must terminate before j starts.
- R is a set of resources where $k \in R$ has a capacity $c_k \in \mathbb{N}$.
- Each task i uses a quantity $r_{k,i}$ of resources k.

Goal: find a (minimal) planning of tasks T that satisfies precedences in P without exceeding the capacity of available resources.

Example with 5 tasks and 2 resources







Constraints model

- **Variables** : $s_i \in \{0..h-1\}$ is the starting time of task i.
- **Constraints** :

$$\forall (i \ll j) \in P,\ s_i + d_i \leq s_j \tag{1}$$

$$\begin{aligned} &\forall j \in [1..n],\ \forall i \in [1..n] \setminus \{j\}, \\ &\qquad b_{i,j} \Leftrightarrow (s_i \leq s_j \wedge s_j < s_i + d_i) \end{aligned} \tag{2}$$

$$\forall j \in [1..n],\ r_{k,j} + (\sum_{i \in [1..n] \setminus \{j\}} r_{k,i} * b_{i,j}) \leq c_k \tag{3}$$

1. Temporal constraints (eq. 1)
2. Resources constraints (eq. 2 and 3): *tasks decomposition* of `cumulative`.

Three kinds of constraints

- In green: octagonal constraints treated by octagon abstract domain.
- In red: equivalence constraints treated in a specialized reduced product.
- In blue: interval constraints treated by the CSP abstract domain.

$$\forall (i \ll j) \in P,\ s_i + d_i \leq s_j$$

$$\begin{aligned} &\forall j \in [1..n],\ \forall i \in [1..n] \setminus \{j\}, \\ &\qquad b_{i,j} \Leftrightarrow (s_i \leq s_j \wedge s_j < s_i + d_i) \end{aligned}$$

$$\forall j \in [1..n],\ r_{k,j} + (\sum_{i \in [1..n] \setminus \{j\}} r_{k,i} * b_{i,j}) \leq c_k$$

Equivalence constraints **connect** the CSP and octagon abstract domains.

Plan

- Introduction
- Scheduling problem RCPSP
- Abstract domains for RCPSP
- Conclusion

CP Abstract Domain

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

- $\bot$ is the smallest element.
- $\sqcup$ performs the union (*join*) of two elements.
- $[\![.]\!] : \Phi \to Abs$ is a partial function turning a constraint into an element of the abstract domain.
- $closure : Abs \to Abs$ propagates the constraints in the abstract domain.
- $\models$: $Abs \times \Phi$ where $a \models \varphi$ holds if $\gamma(a) \leq [\![\varphi]\!]^\natural$.
- $\ldots$

Integer octagon (Miné, 2004)

An integer octagon is defined over a set of variables $(x_0, \dots, x_{n-1})$ and constraints:

$$\pm x_i - \pm x_j \leq d$$

where $d \in \mathbb{Z}$ is a constant.

Complexity of the main operations:

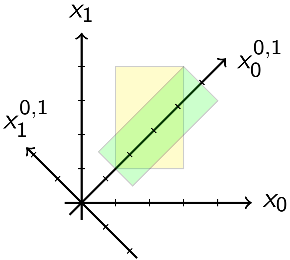
- *join* is $\mathcal{O}(n^2)$.
- *closure*: Floyd-Warshall algorithm in $\mathcal{O}(n^3)$, incremental version in $\mathcal{O}(n^2)$ to add a single constraint (Chawdhary and al., 2018). Normal form equivalent to **path consistency** (Dechter and al., 1991).
- $o \models \varphi$ is in constant time when φ is a single octagonal constraint.

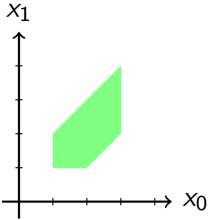
Example of integer octagon

Take the following constraints:

$$
\begin{array}{ll}
x_0 \geq 1 \wedge x_0 \leq 3 & x_1 \geq 1 \wedge x_1 \leq 4 \\
x_0 - x_1 \leq 1 & -x_0 + x_1 \leq 1
\end{array}
$$

Bound constraints on x_0 and x_1 are represented by the yellow box, and octagonal constraints by the green box.





Direct product: combination of abstract domains

We can define a direct product over $CSP \times Oct$ as follows:

$$(csp, o) \sqcup (csp', o') = (csp \sqcup_{CSP} csp', o \sqcup_{Oct} o')$$

$$[\![c]\!] = \begin{cases} ([\![c]\!]_{CSP}, [\![c]\!]_{Oct}) & \\ ([\![c]\!]_{CSP}, \bot_{Oct}) & \text{if } [\![c]\!]_{Oct} \text{ is not defined} \\ (\bot_{CSP}, [\![c]\!]_{Oct}) & \text{if } [\![c]\!]_{CSP} \text{ is not defined} \end{cases}$$

$$closure((csp, o)) = (closure(csp), closure(o))$$

Issue: domains do not exchange information.

Reduced product via equivalence constraints

We consider a reduced product to connect constraints from both domains via equivalence constraints.

- Let $c_1 \Leftrightarrow c_2$ be an equivalence constraint where $[\![c_1]\!]_{CSP}$ and $[\![c_2]\!]_{Oct}$ are defined, then we have:

$$
\mathit{prop}_{\Leftrightarrow}(\mathit{csp}, o, c_1 \Leftrightarrow c_2) \triangleq
\begin{cases}
\mathit{csp} \models_{CSP} c_1 \implies (\mathit{csp}, o \sqcup [\![c_2]\!]_{Oct}) \\
\mathit{csp} \models_{CSP} \neg c_1 \implies (\mathit{csp}, o \sqcup [\![\neg c_2]\!]_{Oct}) \\
o \models_{Oct} c_2 \implies (\mathit{csp} \sqcup [\![c_1]\!]_{CSP}, o) \\
o \models_{Oct} \neg c_2 \implies (\mathit{csp} \sqcup [\![\neg c_1]\!]_{CSP}, o) \\
(\mathit{csp}, o) \text{ otherwise}
\end{cases}
$$

- This propagator is sound: it does not remove solutions.

Reduced product via equivalence constraints

We improve the closure operator by propagating the set of equivalence constraints R.
Closure operator of the reduced product $CSP \times Oct$:

$$closure_R(csp, oct, R) = (\bigsqcup_{r \in R} prop_{\Leftrightarrow}(csp, oct, r), R)$$

$$closure((csp, oct, R)) = \\ \quad (closure_R(closure(csp'), closure(oct'), R))$$

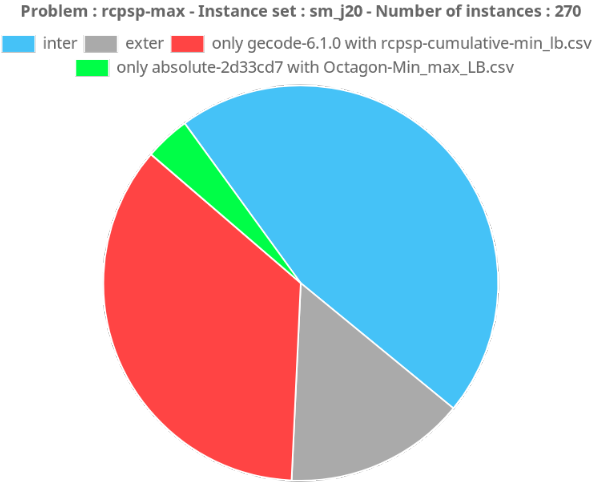
Let e be an element of the reduced product, then the closure operator can be applied to a fixed point $closure(e) = e$.

Result: A generic reduced product to combine abstract domains with disjoint set of variables.

Plan

Benchmarks

- In brief: We are bettered by state of the art methods (e.g. CHUFFED with *lazy clause*).
- In comparison to GeCode/cumulative:



Related work

- Satisfiability modulo theories (SMT)
  - SMT theories and abstract domains very close but on different underlying theories (logic VS posets).
  - Implementation of Nelson-Oppen usually not formalized and mysterious.
- 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.

We aim to reduce the gap between practice and theory.

Conclusion

1. New structures:
   - Investigate integer octagon abstract domains for CP.
   - A new combination: reduced product by equivalence constraints.
     $\Rightarrow$ It allows us to use reified constraints in abstract domains.
2. A case study: RCPSP.
   Bonus: We can handle **continuous temporal constraint** with discrete resources.
3. Automated benchmarking framework (including Chuffed, GeCode, AbSolute).

`github.com/ptal/AbSolute/tree/ictai2019`

Thank you!