Abstract Interpretation of Constraint Programming

Seminar GDR AI

Pierre Talbot
16 June 2021

University of Luxembourg
Parallel Computing & Optimisation Group (PCOG)
This seminar in a nutshell!

We present the “fusion” of...

Constraint reasoning + Abstract interpretation
(and lattice theory)

that gives us abstract constraint reasoning.
This seminar in a nutshell!

We present the “fusion” of...

Constraint reasoning + Abstract interpretation

WHY?
- A framework for combining constraint solvers.
- Constraint solving on GPUs.

that gives us abstract constraint reasoning.
Abstract constraint reasoning

- Data structures = lattices
Abstract constraint reasoning

- Data structures = lattices
- Algorithms = extensive functions
- Example: \( f(x) = x \sqcup [2..\infty] \) models the constraint \( x \geq 2 \).
- Lattice + Extensive function = Abstract domains
I. A framework for combining constraint solvers

- SAT [DHK13]
- SMT [CCM13]
- Logic programming [Cou20]
- Constraint programming [Pel+13]
- Linear programming
- Multi-objective optimization
- Multilevel programming
- Abstract domains
I. A framework for combining constraint solvers

- SAT [DHK13]
- SMT [CCM13]
- Logic programming [Cou20]
- Constraint programming [Pel+13]
- Linear programming
- Multi-objective optimization
- Multilevel programming
- Abstract domains
II. Towards a theory for constraint solving on GPUs

• $f(x) = x \sqcup [2..\infty]$ models the constraint $x \geq 2$.
• $g(x) = x \sqcup [-\infty..2]$ models the constraint $x \leq 2$.
• Concurrent execution: $f \parallel g = [2..2]$

A new twist on an old idea: *asynchronous iterations* of abstract interpretation [Cou77].
I. A framework for combining constraint solvers
   1. (Traditional) Constraint Programming
   2. Abstract Constraint Programming
   3. Products of Abstract Domains
   4. Soundness and Completeness

II. Towards a theory for constraint solving on GPUs

III. Conclusion
(Traditional) Constraint Programming
An example of constraint problem

**Constraint problem**: Tasks have a duration, use resources (%CPU/%GPU), and have precedence relations.

**Goal**: Find a minimal schedule of the tasks on the HPC.
An example of constraint problem

- **Constraint programming**: we only specify what should be the solution using relations on variables (*declarative programming*).
- But we do not program how to compute the solution.
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: $i \prec j \in P$ 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:**

  \[
  \forall (i \ll j) \in P, \ s_i + d_i \leq s_j
  \]  
  \[
  \forall j \in [1..n], \ \forall i \in [1..n] \setminus \{j\}, \ b_{i,j} \iff (s_i \leq s_j \land s_j < s_i + d_i)
  \]  
  \[
  \forall j \in [1..n], \ r_{k,j} + \left( \sum_{i \in [1..n] \setminus \{j\}} r_{k,i} \times b_{i,j} \right) \leq c_k
  \]

1. Temporal constraints (eq. 1)

2. Resources constraints (eq. 2 and 3): *tasks decomposition* of cumulative.
A CSP is a pair $\langle d, C \rangle$, example:

$\langle \{ T_1 \mapsto \{1, 2, 3, 4\}, T_2 \mapsto \{2, 3, 4\}\}, \{ T_1 \geq T_2, T_1 \neq 4\} \rangle$

A solution is $\{ T_1 \mapsto 2, T_2 \mapsto 2\}$. 
How does a constraint solver work?

A constraint solving algorithm: propagate and search

- **Propagate**: Remove inconsistent values from the variables’ domain.

  \[
  T_1 \geq T_2 \quad \{ T_1 \mapsto \{1, 2, 3, 4\}, T_2 \mapsto \{2, 3, 4\}\}
  \]
  \[
  T_1 \neq 4 \quad \{ T_1 \mapsto \{2, 3, 4\}, T_2 \mapsto \{2, 3, 4\}\}
  \]
  \[
  T_1 \geq T_2 \quad \{ T_1 \mapsto \{2, 3\}, T_2 \mapsto \{2, 3, 4\}\}
  \]
  \[
  T_1 \neq 2 \quad \{ T_1 \mapsto \{2, 3\}, T_2 \mapsto \{2, 3\}\}
  \]
  \[
  T_1 \geq T_2 \quad \{ T_1 \mapsto \{2, 3\}, T_2 \mapsto \{2, 3\}\}
  \]

A constraint \( c \) is implemented by a *propagator* function \( p_c : D \to D \).

- **Search**: Divide the problem into (complementary) subproblems explored using *backtracking*.
  - Subproblem 1: \( \langle \{ T_1 \mapsto \{2\}, T_2 \mapsto \{2, 3\}\}, \{ T_1 \geq T_2, T_1 \neq 4\} \rangle \)
  - Subproblem 2: \( \langle \{ T_1 \mapsto \{3\}, T_2 \mapsto \{2, 3\}\}, \{ T_1 \geq T_2, T_1 \neq 4\} \rangle \)
A classic solver in constraint programming:

1. solve(⟨d, C⟩)
2. ⟨d′, C⟩ ← propagate(⟨d, C⟩)
3. if d′ is an assignment then
   4. return {d′}
4. else if d′ has an empty domain then
   5. return {} 
5. else
   6. ⟨d₁, ..., dₙ⟩ ← branch(d′)
   7. return ∪ₙᵢ₌₀ solve(⟨dᵢ, C⟩)
8. end if
Abstract Constraint Programming
Abstract domain for constraint reasoning [Pel+13; Tal+21]

An abstract domain $\langle \text{Abs}, \leq, \sqcup, \perp, \gamma, [], \text{refine}, \text{split} \rangle$ is a lattice such that:

- $\text{Abs}$ is a set of elements representable in a machine.
- $\leq$ is a partial order.
- $\sqcup$ performs the join of two elements ("union of information").
- $\perp$ is the smallest element ("initial state").
- $\gamma : A \rightarrow D^b$ is a monotone concretization function.
- $[] : \Phi \rightarrow \text{Abs}$ is a partial interpretation function turning a constraint into an element of the abstract domain.
- $\text{refine} : \text{Abs} \rightarrow \text{Abs}$ is an extensive function, e.g., $a \leq \text{refine}(a)$, refining an abstract element ("gain information").
- $\text{split} : \text{Abs} \rightarrow \mathcal{P}(\text{Abs})$ is an extensive function dividing an abstract element into a set of sub-elements.
- $\models : \text{Abs} \times \Phi : a \models \varphi$ holds whenever $\gamma(a) \subseteq \llbracket \varphi \rrbracket^b$.
- ...
Box abstract domain \[ [\text{Box}] \]

- Let \( I \) be the lattice of integer intervals, and \( X \) a set of variables.
- Then \( \text{Box} = [X \rightarrow I] \) is the abstract domain of box.

It treats constraints of the form

\[
x \leq d \quad x \geq d
\]

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

Example of abstract domain operations:

- \( [x \leq d] \triangleq \{x \mapsto [-\infty..d]\} \),
- \( \sigma \leq \tau \triangleq \forall x \in \text{dom}(\sigma), \ x \in \text{dom}(\tau) \land \sigma(x) \leq \tau(x) \) where \( \text{dom}(\sigma) \) denotes the domain of \( \sigma \),
- \( \sigma \sqcup \tau \triangleq \lambda x. \begin{cases} \sigma(x) \sqcup \tau(x) & \text{if } x \in \text{dom}(\sigma) \cap \text{dom}(\tau) \\ \sigma(x) & \text{if } x \in \text{dom}(\sigma) \setminus \text{dom}(\tau) \\ \tau(x) & \text{if } x \in \text{dom}(\tau) \setminus \text{dom}(\sigma) \end{cases} \)
An integer octagon is defined over a set of variables \((x_0, \ldots, x_{n-1})\) and constraints:

\[
\pm x_i - \pm x_j \leq d
\]

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

Complexity of the main operations:

- \(\text{join}\) is \(O(n^2)\).
- \(\text{refine}\): Floyd-Warshall algorithm in \(O(n^3)\), incremental version in \(O(n^2)\) to add a single constraint [CRK18].
- \(o \models \varphi\) is in constant time when \(\varphi\) is a single octagonal constraint.
Example of integer octagon

Take the following constraints:

\[
\begin{align*}
    x_0 & \geq 1 \land x_0 \leq 3 \\
    x_0 - x_1 & \leq 1 \\
    x_1 & \geq 1 \land x_1 \leq 4 \\
    -x_0 + x_1 & \leq 1
\end{align*}
\]

Bound constraints on \(x_0\) and \(x_1\) are represented by the yellow box, and octagonal constraints by the green box.
Abstract constraint solver

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

1: $\text{solve}(a \in Abs)$
2: $a \leftarrow \text{refine}(a)$
3: if $\text{split}(a) = \{a\}$ then
4: $\text{return } \{a\}$
5: else if $\text{split}(a) = \{\} \text{ then}$
6: $\text{return } \{\}$
7: else
8: $\langle a_1, \ldots, a_n \rangle \leftarrow \text{split}(a)$
9: $\text{return } \bigcup_{i=0}^n \text{solve}(a_i)$
10: end if

**Conservative extension:** We encapsulate propagators in an abstract domain $PP$.

**Many abstract domains:** Octagon, Polyhedron, products, …
Products of Abstract Domains
Three kinds of constraints in RCPSP

- 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 PP abstract domain.

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

\[ \forall j \in [1..n], \ \forall i \in [1..n] \setminus \{j\}, \ b_{i,j} \iff (s_i \leq s_j \land s_j < s_i + d_i) \]

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

Equivalence constraints connect the PP and octagon abstract domains.
We can define a direct product over $PP \times Oct$ as follows:

$$(p, o) \sqcup (p', o') = (p \sqcup_{PP} p', o \sqcup_{Oct} o')$$

$$\left\{ \begin{array}{l}
[[\varphi]_{PP}, [[\varphi]_{Oct}] \\
([[\varphi]_{PP}, \bot_{Oct}) & \text{if } [[\varphi]_{Oct} \text{ is not defined} \\
(\bot_{PP}, [[\varphi]_{Oct}) & \text{if } [[\varphi]_{PP} \text{ is not defined}
\end{array} \right.$$  

$\text{refine}((p, o)) = (\text{refine}(p), \text{refine}(o))$

**Issue:** domains do not exchange information.
Reduced product via equivalence constraints [Tal+19]

We can improve the refinement operator of the direct product by connecting constraints from both domains via equivalence constraints.

- Let \( \varphi_1 \Leftrightarrow \varphi_2 \) be an equivalence constraint where \( [\varphi_1]_{PP} \) and \( [\varphi_2]_{Oct} \) are defined, then we have:

\[
\text{prop} \Leftrightarrow (p, o, \varphi_1 \Leftrightarrow \varphi_2) \triangleq
\begin{cases}
  p \models_{PP} \varphi_1 & \Rightarrow (p, o \sqcup [\varphi_2]_{Oct}) \\
  p \models_{PP} \neg \varphi_1 & \Rightarrow (p, o \sqcup [\neg \varphi_2]_{Oct}) \\
  o \models_{Oct} \varphi_2 & \Rightarrow (p \sqcup [\varphi_1]_{PP}, o) \\
  o \models_{Oct} \neg \varphi_2 & \Rightarrow (p \sqcup [\neg \varphi_1]_{PP}, o) \\
  (p, o) \text{ otherwise}
\end{cases}
\]

- **Result:** A generic reduced product to combine abstract domains with disjoint set of variables.
Consider the constraint $\varphi \triangleq D_1 > 1 \land T_1 + T_2 \leq D_1 \land T_1 - T_2 \leq 3$.

- $D_1 > 1$ can be interpreted in boxes,
- $T_1 - T_2 \leq 3$ in octagons,
- but $T_1 + T_2 \leq D_1$ is too general for any of these two because it has 3 variables...
- ...and it shares its variables with the other two.

**Solution**: Use the notion of propagator functions to connect variables between abstract domains.
Abstract domain: Interval propagators completion (IPC)

- Lattice structure: $IPC(A) = A \times \mathcal{P}([A \to A])$.
- We equip $A$ with a pair of projective functions $\lfloor t \rfloor_a$ and $\lceil t \rceil_a$ projecting resp. the lower and upper bound of the term $t$ in $a \in A$.

The goal is to use $IPC(\text{Box} \times \text{Octagon})$ with a propagator for $T_1 + T_2 \leq D_1$:

$$\lceil T_1 + T_2 \leq D_1 \rceil = \lambda a. a$$

$$\sqcup_A \lfloor T_1 + T_2 \leq \lfloor t \rfloor_a \rfloor \quad \text{Send an over-approximation to octagon.}$$

$$\sqcup_A \lfloor \lceil T_1 + T_2 \rceil_a \leq D_1 \rfloor \quad \text{Send an over-approximation to box.}$$
Interval propagators completion

\[
\begin{align*}
\llbracket T_1 + T_2 \leq D_1 \rrbracket &= \lambda a. a \\
\bigtriangledown_A \llbracket T_1 + T_2 \leq \lfloor t \rfloor_a \rrbracket_A &\quad \text{Send an over-approximation to octagon.} \\
\bigtriangledown_A \llbracket \lceil T_1 + T_2 \rceil_a \leq D_1 \rrbracket_A &\quad \text{Send an over-approximation to box.}
\end{align*}
\]

Example

- Let \( D_1 \in [1..3] \), then \( T_1 + T_2 \leq 3 \) is sent to the octagon.
- Let \( T_1 + T_2 \in [2..4] \), then \( 2 \leq D_1 \) is sent to the box.
- New over-approximations are sent whenever a bound is updated.

Exchange of over-approximations among abstract domains.
Soundness and Completeness
Abstract constraint reasoning

• $\Phi$ is the set of all first-order logical formulas.
• $C^b$ is the concrete domain.
• $A^\#$ is the abstract domain.
• $x > 2.25 \land x < 2.75 \in \Phi$ is a logical formula.
Abstract constraint reasoning

\[ x > 2.25 \land x < 2.75 \]

\[ \{ x \in \mathbb{R} \mid x > 2.25 \land x < 2.75 \} \]

- \( x > 2.25 \land x < 2.75 \in \Phi \) is a logical formula.
- \( \{ x \in \mathbb{R} \mid x > 2.25 \land x < 2.75 \} \) is the \textit{concrete solutions set} of this formula.
Abstract constraint reasoning

\[ x > 2.25 \land x < 2.75 \]

\[ F \times F \xrightarrow{\alpha} \mathcal{P}(\mathbb{R}) \]

\[ \mathbb{R} \]

- It is not possible to represent all real numbers in a machine.
- We rely on the abstract domain of floating point intervals \( F \times F \).
Abstract constraint reasoning

\[ x > 2.25 \land x < 2.75 \]

\[ \mathbb{F} \times \mathbb{F} \leftrightarrow \mathcal{P}(\mathbb{R}) \]

\[ e.g. \{x \in \mathbb{R} \mid x > 2.25 \land x < 2.75\} \]

- Tradeoff between **completeness** and **soundness**: either all solutions with extra, or a subset without extra.

- **Over-approximation**: \( \llbracket x > 2.25 \land x < 2.75 \rrbracket \uparrow = [2.25..2.75] \in \mathbb{F}^2 \)
  (2.25 and 2.75 are not solutions).

- **Under-approximation**: \( \llbracket x > 2.25 \land x < 2.75 \rrbracket \downarrow = [2.375..2.625] \in \mathbb{F}^2 \)
  (2.26 and 2.74 are missing solutions).
Concrete domain for constraint reasoning

- Let $V$ be a set of values (universe of discourse) and $X$ a set of variables.
- We have $Asn = [X \rightarrow V]$, the set of all assignments of the variables to values.
- The **concrete domain** is the following lattice $D^b = \langle \mathcal{P}(Asn), \supseteq \rangle$.

Using the usual Tarski model-theoretic semantics of first-order logic, we can interpret a logical formula $\varphi$ in the concrete domain ($A$ is a structure):

$$
\llbracket . \rrbracket^b : \Phi \rightarrow D^b \\
\llbracket \varphi \rrbracket^b = \{ a \in Asn \mid A \models_a \varphi \}
$$

Example:

$$
\llbracket x \in \{1, 2\}, y \in \{1, 3\}, x \geq y \rrbracket^b = \{ \{ x \mapsto 1, y \mapsto 1 \}, \{ x \mapsto 2, y \mapsto 1 \} \}
$$
Using this formal framework, we establish two important properties of abstract domains:

\[
\exists i \in \mathbb{N}, \ (\gamma \circ \text{refine}^i \circ [.])(\varphi) \subseteq [\varphi]^b \quad \text{(under-approximation)}
\]

\[
\forall i \in \mathbb{N}, \ (\gamma \circ \text{refine}^i \circ [.])(\varphi) \supseteq [\varphi]^b \quad \text{(over-approximation)}
\]
Further theoretical investigations [Tal+21] (draft)

When reasoning in this framework, fundamental questions arise:

- **Compositionality**: given two under-/over-approximating refinement functions $f$ and $g$, under what conditions $f \circ g$ preserves under-/over-approximations?

- How to define propagation which is an over-approximating refinement operator which becomes under-approximating on unsplittable elements.
  
  $\Rightarrow$ **Search tree abstract domain**.

- ...

It is possible to establish general theorems valid for any/many abstract domains.
How to create an appropriate combination of abstract domains for a particular formula?

“Type inference”: In which abstract domain goes each subformula $\varphi_i \in \varphi$?
Towards a theory for constraint solving on GPUs
Constraint solving on GPUs  (Ongoing research project with Frédéric Pinel)

- $f(x) = x \sqcup [2..\infty]$ models the constraint $x \geq 2$.
- $g(x) = x \sqcup [-\infty..2]$ models the constraint $x \leq 2$.
- Concurrent execution: $f \parallel g = [2..2]$

In parallel on shared memory? No problem, because they do not modify the same memory cell... but what if?
Parallel execution of refinement functions

Here, both $f$ and $g$ modify the same memory cell: race condition?

```c
void update_lb(int new_lb) {
    if (new_lb > lb) {
        lb = new_lb;
    }
}
```

Indeed, it is possible that after $f \parallel g$, we have [1..3] instead of [2..3].
Key idea: With lattice data structure and fixpoint of refinement, our model is tolerant to race conditions.

- Key idea: we execute $f \parallel g$ until we reach a fixpoint.
- Assume a race condition, then $f \parallel g = [1..3]$.
- But $f \parallel g$ is not at a fixed point, so it is reexecuted.
- The second time, $f \parallel g = [2..3]$, because $g$ is at a local fixpoint and cannot write in $lb$ anymore.
We have experimented this idea with Turbo\textsuperscript{1}, a constraint solver with both propagation and search on the GPU.

- Almost no synchronization (2 \texttt{__syncthreads}, mostly due to the opaque scheduling strategy of NVIDIA GPU).
- No atomic statement (actually, just one for the optimisation bound but avoidable!).

Still many optimisations to make, currently around one order of magnitude faster than GeCode on simple scheduling problem.

\textsuperscript{1}https://github.com/ptal/turbo/
An architecture for constraint solving on GPU

- OR-parallelism across SM.
- AND-parallelism inside each SM.
- Enable the usage of cache L1 for fast memory access.
Conclusion
• Abstract interpretation a “grand unification theory” among the fields of constraint reasoning?
• Not there yet, but interesting theory and promising results!
References


