Abstract Interpretation of Constraint Programming

Seminar GDR AI

Pierre Talbot

16 June 2021

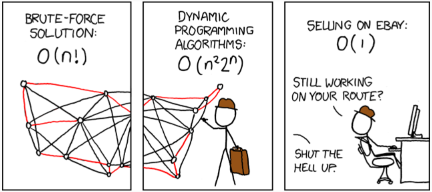
University of Luxembourg
Parallel Computing & Optimisation Group (PCOG)

This seminar in a nutshell!

We present the "fusion" of...

Constraint reasoning + Abstract interpretation

(and lattice theory)



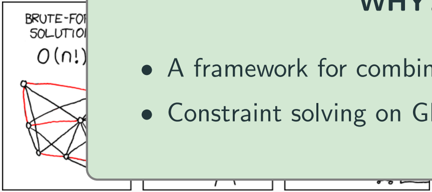
that gives us abstract constraint reasoning.

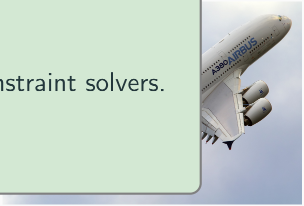
This seminar in a nutshell!

We present the "fusion" of...

Constraint reasoning + Abstract interpretation

ory)



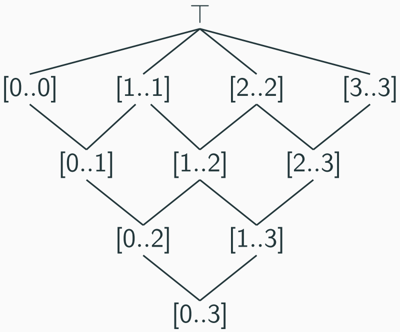


WHY?

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

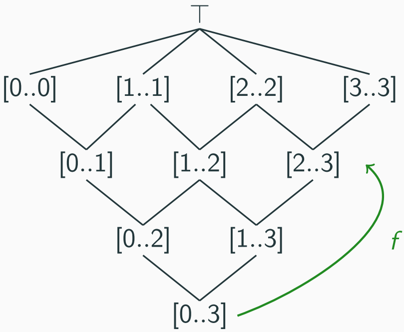
that gives us abstract constraint reasoning.

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

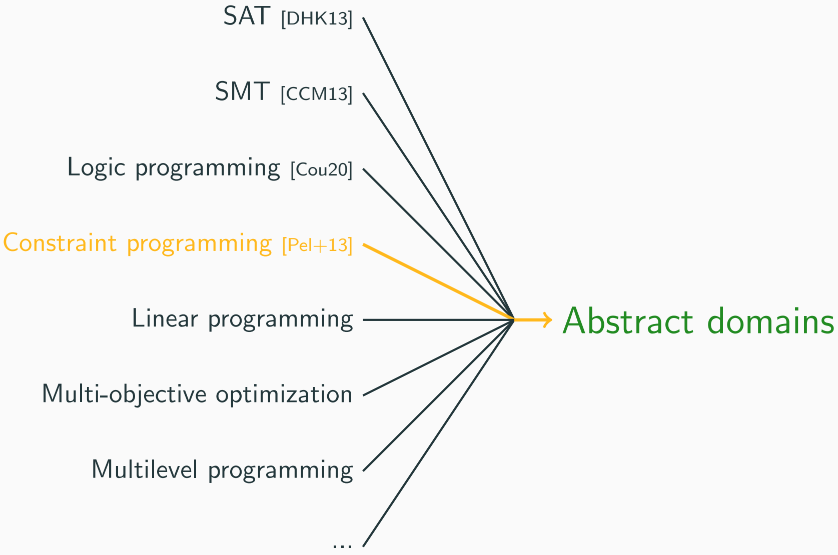
Multi-objective optimization

Multilevel programming

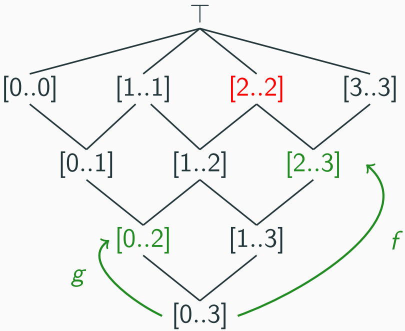
...

$\longrightarrow$ Abstract domains

I. A framework for combining constraint solvers



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 || g = [2..2]`

A new twist on an old idea: *asynchronous iterations* of abstract interpretation [Cou77].

Plan

I. A framework for combining constraint solvers

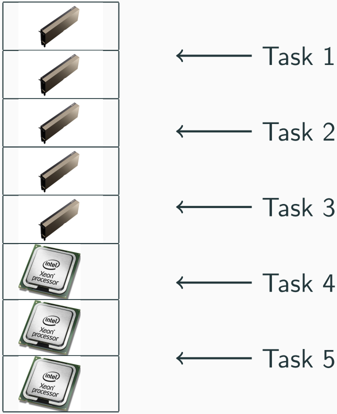
1. (Traditional) Constraint Programming
2. Abstract Constraint Programming
3. Products of Abstract Domains
4. Soundness and Completeness

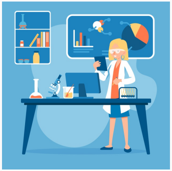
II. Towards a theory for constraint solving on GPUs

III. Conclusion

(Traditional) Constraint Programming

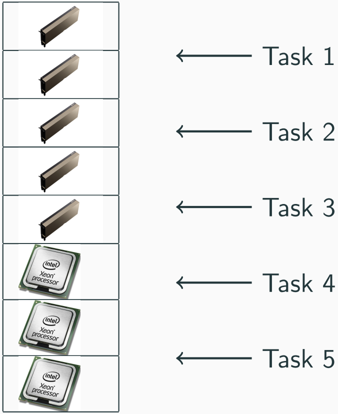
An example of constraint problem

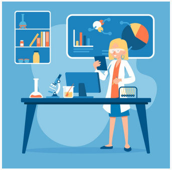




Constraint problem: Tasks have a duration, use resources (#CPU/#GPU), and have precedence relations.
Goal: Find a minimal schedule of the tasks on the HPC.

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.

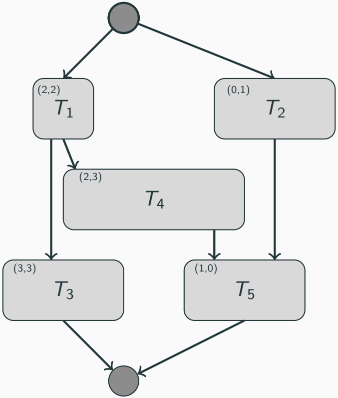
Scheduling problem RCPSP

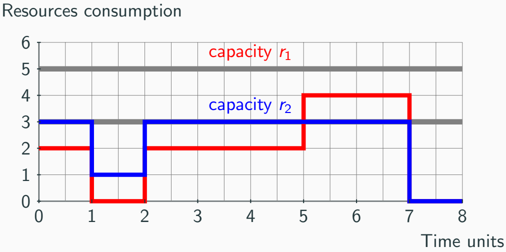
NP-complete optimisation problem:

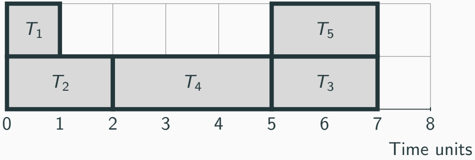
- T is a set of tasks, $d_i \in \mathbb{N}$ the duration of task i.
- P are the precedences among tasks: $i \ll j \in P$ if i must terminate before j starts.
- R is a set of resources where $k \in R$ has a capacity $c_k \in \mathbb{N}$.
- Each task i uses a quantity $r_{k,i}$ of resources k.

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** :

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

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

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

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

How does a constraint solver work?

Constraint satisfaction problem (CSP)

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.

$$
\begin{array}{ll}
T_1 \geq T_2 & \{T_1 \mapsto \{1,2,3,4\}, T_2 \mapsto \{2,3,4\}\} \\
T_1 \neq 4 & \{T_1 \mapsto \{2,3,4\}, T_2 \mapsto \{2,3,4\}\} \\
T_1 \geq T_2 & \{T_1 \mapsto \{2,3\}, T_2 \mapsto \{2,3,4\}\} \\
T_1 \neq 2 & \{T_1 \mapsto \{2,3\}, T_2 \mapsto \{2,3\}\} \\
T_1 \geq T_2 & \{T_1 \mapsto \{2,3\}, T_2 \mapsto \{2,3\}\}
\end{array}
$$

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

Constraint solver: propagate and search

A classic solver in constraint programming:

```
solve(⟨d, C⟩)
⟨d′, C⟩ ← propagate(⟨d, C⟩)
if d′ is an assignment then
    return {d′}
else if d′ has an empty domain then
    return {}
else
    ⟨d_1, ..., d_n⟩ ← branch(d′)
    return ⋃_{i=0}^{n} solve(⟨d_i, C⟩)
end if
```

Abstract Constraint Programming

Abstract domain for constraint reasoning [Pel+13; Tal+21]

An abstract domain $\langle Abs, \leq, \sqcup, \bot, \gamma, [\![.]\!], refine, split \rangle$ is a lattice such that:

- *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").
- $\bot$ is the smallest element ("initial state").
- $\gamma : A \rightarrow D^\flat$ is a monotone concretization function.
- $[\![.]\!] : \Phi \rightarrow Abs$ is a partial interpretation function turning a constraint into an element of the abstract domain.
- $refine : Abs \rightarrow Abs$ is an extensive function, *e.g.*, $a \leq refine(a)$, refining an abstract element ("gain information").
- $split : Abs \rightarrow \mathcal{P}(Abs)$ is an extensive function dividing an abstract element into a set of sub-elements.
- $\models$: $Abs \times \Phi$: $a \models \varphi$ holds whenever $\gamma(a) \subseteq [\![\varphi]\!]^\flat$.
- ...

Box abstract domain [CC77]

- Let I be the lattice of integer intervals, and X a set of variables.
- Then $Box = [X \nrightarrow I]$ is the abstract domain of box.

It treats constraints of the form

$$x \leq d \qquad x \geq d$$

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

Example of abstract domain operations:

- $[\![x \leq d]\!] \triangleq \{x \mapsto [-\infty..d]\}$,
- $\sigma \leq \tau \triangleq \forall x \in dom(\sigma),\ x \in dom(\tau) \wedge \sigma(x) \leq \tau(x)$ where $dom(\sigma)$ denotes the domain of σ,
- $\sigma \sqcup \tau \triangleq \lambda x. \begin{cases} \sigma(x) \sqcup \tau(x) & \text{if } x \in dom(\sigma) \cap dom(\tau) \\ \sigma(x) & \text{if } x \in dom(\sigma) \setminus dom(\tau) \\ \tau(x) & \text{if } x \in dom(\tau) \setminus dom(\sigma) \end{cases}$

Integer octagon [Min06]

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:

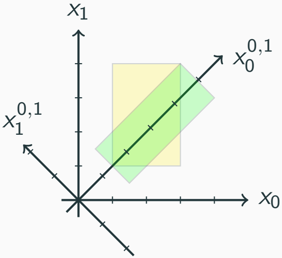
- *join* is $\mathcal{O}(n^2)$.
- *refine*: Floyd-Warshall algorithm in $\mathcal{O}(n^3)$, incremental version in $\mathcal{O}(n^2)$ to add a single constraint [CRK18].
- $o \vDash \varphi$ is in constant time when φ is a single octagonal constraint.

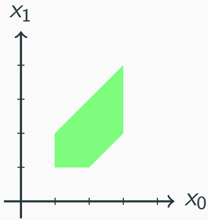
Example of integer octagon

Take the following constraints:

$$x_0 \geq 1 \wedge x_0 \leq 3 \qquad x_1 \geq 1 \wedge x_1 \leq 4$$
$$x_0 - x_1 \leq 1 \qquad -x_0 + x_1 \leq 1$$

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:

$\texttt{solve}(a \in Abs)$
$a \leftarrow \texttt{refine(a)}$
if $\texttt{split(a)} = \{\texttt{a}\}$ **then**
  return $\{a\}$
else if $\texttt{split(a)} = \{\}$ **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 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.

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

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

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

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

Direct product: combination of abstract domains

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

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

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

$$refine((p, o)) = (refine(p), refine(o))$$

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:

$$
\begin{aligned}
&\mathit{prop}_{\Leftrightarrow}(p, o, \varphi_1 \Leftrightarrow \varphi_2) \triangleq \\
&\qquad \begin{cases}
p \vDash_{PP} \varphi_1 \implies (p, o \sqcup [\![\varphi_2]\!]_{Oct}) \\
p \vDash_{PP} \neg\varphi_1 \implies (p, o \sqcup [\![\neg\varphi_2]\!]_{Oct}) \\
o \vDash_{Oct} \varphi_2 \implies (p \sqcup [\![\varphi_1]\!]_{PP}, o) \\
o \vDash_{Oct} \neg\varphi_2 \implies (p \sqcup [\![\neg\varphi_1]\!]_{PP}, o) \\
(p, o) \text{ otherwise}
\end{cases}
\end{aligned}
$$

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

Interval propagators completion [TMT20]

Consider the constraint $\varphi \triangleq D_1 > 1 \wedge T_1 + T_2 \leq D_1 \wedge T_1 - T_2 \leq 3$.

- $D_1 > 1$ 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.

Interval propagators completion

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 $IPC(Box \times Octagon)$ with a propagator for $T_1 + T_2 \leq D_1$:

$$
\begin{aligned}
&[\![T_1 + T_2 \leq D_1]\!] = \lambda a.a \\
&\quad \sqcup_A \ [\![T_1 + T_2 \leq \lceil t \rceil_a]\!]_A && \text{Send an over-approximation to octagon.} \\
&\quad \sqcup_A \ [\![\lfloor T_1 + T_2 \rfloor_a \leq D_1]\!]_A && \text{Send an over-approximation to box.}
\end{aligned}
$$

Interval propagators completion

$$\llbracket T_1 + T_2 \leq D_1 \rrbracket = \lambda a.a$$

$\sqcup_A \ \llbracket T_1 + T_2 \leq \lceil t \rceil_a \rrbracket_A$	Send an over-approximation to octagon.
$\sqcup_A \ \llbracket \lfloor T_1 + T_2 \rfloor_a \leq D_1 \rrbracket_A$	Send an over-approximation to box.

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

$$\begin{tikzcd}
 & \Phi \arrow[dl, "\llbracket\rrbracket^\sharp"'] \arrow[dr, "\llbracket\rrbracket^\flat"] & \\
A^\sharp \arrow[rr, shift left, "\gamma"] & & C^\flat \arrow[ll, shift left, "\alpha"]
\end{tikzcd}$$

- Φ is the set of all first-order logical formulas.
- $C^\flat$ is the concrete domain.
- $A^\sharp$ is the abstract domain.

Abstract constraint reasoning

$$
\begin{tikzcd}
 & x > 2.25 \land x < 2.75 \arrow[dl, "\llbracket\,\rrbracket^{\sharp}"'] \arrow[dr, "\llbracket\,\rrbracket^{\flat}"] & \\
A^{\sharp} \arrow[rr, shift left, "\gamma"] & & C^{\flat} \arrow[ll, shift left, "\alpha"]
\end{tikzcd}
$$

- $x > 2.25 \land x < 2.75 \in \Phi$ is a logical formula.

Abstract constraint reasoning

$$
\begin{tikzcd}
 & x > 2.25 \land x < 2.75 \arrow[dl, "\llbracket\rrbracket^{\sharp}"'] \arrow[dr, "\llbracket\rrbracket^{\flat}"] & \\
A^{\sharp} \arrow[rr, shift left, "\gamma"] & & \mathcal{P}(\mathbb{R}) \arrow[ll, shift left, "\alpha"]
\end{tikzcd}
$$

e.g. $\{x \in \mathbb{R} \,|\, x > 2.25 \land x < 2.75\}$

- $x > 2.25 \land x < 2.75 \in \Phi$ is a logical formula.
- $\{x \in \mathbb{R} \,|\, x > 2.25 \land x < 2.75\}$ is the *concrete solutions set* of this formula.

Abstract constraint reasoning

$$\begin{tikzcd}
 & x > 2.25 \land x < 2.75 \arrow[dl, "\llbracket\rrbracket^\sharp"'] \arrow[dr, "\llbracket\rrbracket^\flat"] & \\
\mathbb{F} \times \mathbb{F} \arrow[rr, shift left, "\gamma"] & & \mathcal{P}(\mathbb{R}) \arrow[ll, shift left, "\alpha"]
\end{tikzcd}$$

e.g. $\{x \in \mathbb{R} \,|\, x > 2.25 \land x < 2.75\}$

- It is not possible to represent all real numbers in a machine.
- We rely on the abstract domain of floating point intervals $\mathbb{F} \times \mathbb{F}$.

Abstract constraint reasoning

$$\begin{tikzcd}
 & x > 2.25 \land x < 2.75 \arrow[dl, "\llbracket\rrbracket^{\sharp}"'] \arrow[dr, "\llbracket\rrbracket^{\flat}"] & \\
\mathbb{F} \times \mathbb{F} \arrow[rr, shift left, "\gamma"] & & \mathcal{P}(\mathbb{R}) \arrow[ll, shift left, "\alpha"]
\end{tikzcd}$$

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*: $\llbracket x > 2.25 \land x < 2.75 \rrbracket^{\sharp}_{\uparrow} = [2.25..2.75] \in \mathbb{F}^2$ (2.25 and 2.75 are not solutions).
- *Under-approximation*: $\llbracket x > 2.25 \land x < 2.75 \rrbracket^{\sharp}_{\downarrow} = [2.375..2.625] \in \mathbb{F}^2$ (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 $Asn = [X \to V]$, the set of all assignments of the variables to values.
- The **concrete domain** is the following lattice $D^\flat = \langle \mathcal{P}(Asn), \supseteq \rangle$.

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

$$\llbracket . \rrbracket^\flat : \Phi \to D^\flat$$
$$\llbracket \varphi \rrbracket^\flat = \{a \in Asn \mid A \vDash_a \varphi\}$$

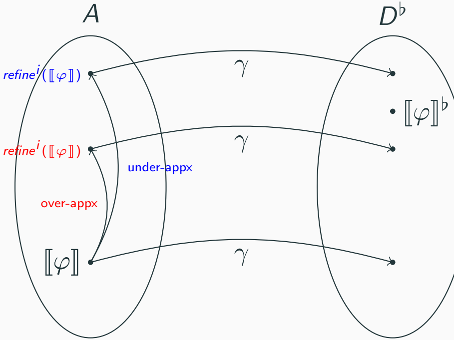
Example:

$$\llbracket x \in \{1,2\}, y \in \{1,3\}, x \geq y \rrbracket^\flat = \{\{x \mapsto 1, y \mapsto 1\}, \{x \mapsto 2, y \mapsto 1\}\}$$

Two core properties

Using this formal framework, we establish two important properties of abstract domains:

$$\exists i \in \mathbb{N},\ (\gamma \circ \mathit{refine}^i \circ [\![.]\!])(\varphi) \subseteq [\![\varphi]\!]^\flat \qquad \text{(under-approximation)}$$
$$\forall i \in \mathbb{N},\ (\gamma \circ \mathit{refine}^i \circ [\![.]\!])(\varphi) \supseteq [\![\varphi]\!]^\flat \qquad \text{(over-approximation)}$$



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.

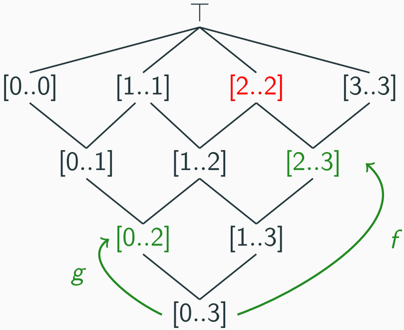
Perspective: Towards automatic creation of the abstract domain

$$\begin{tikzcd}
 & \varphi \arrow[dl, "\llbracket\,\rrbracket^\sharp"'] \arrow[dr, "\llbracket\,\rrbracket^\flat"] & \\
A_1^\sharp \times \ldots \times A_n^\sharp \arrow[rr, shift left, "\gamma"] & & C^\flat \arrow[ll, shift left, "\alpha"]
\end{tikzcd}$$

- 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

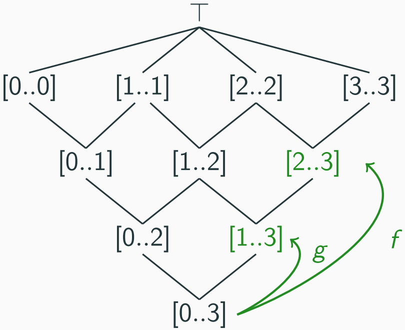
Constraint solving on GPUs (Ongoing research project with Frédéric Pinel)



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

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

Indeed, it is possible that after `f || g`, we have [1..3] instead of [2..3].

Parallel execution without synchronization and atomics

Key idea: With lattice data structure and fixpoint of refinement, **our model is tolerant to race conditions**.

- Key idea: we execute `f || g` until we reach a fixpoint.
- Assume a race condition, then `f || g` $= [1..3]$.
- But `f || g` is not at a fixed point, so it is reexecuted.
- The second time, `f || g` $= [2..3]$, because g is at a local fixpoint and cannot write in `lb` anymore.

Turbo: a pure GPU constraint solver

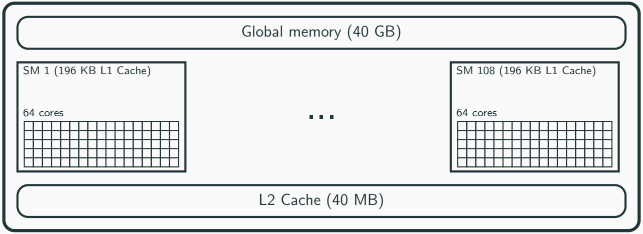
We have experimented this idea with Turbo[1], a constraint solver with both propagation and search on the GPU.

- Almost no synchronization (2 `__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.

[1] `https://github.com/ptal/turbo/`

An architecture for constraint solving on GPU

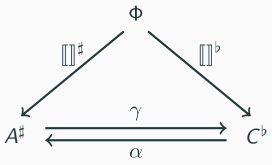


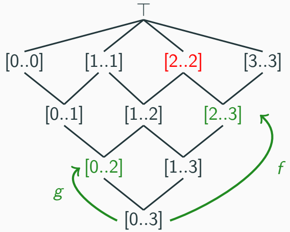
- OR-parallelism across SM.
- AND-parallelism inside each SM.
- Enable the usage of cache L1 for fast memory access.

Conclusion

Conclusion

- Abstract interpretation a “*grand unification theory*” among the fields of constraint reasoning?
- Not there yet, but interesting theory and promising results!





References

[CC77] Patrick Cousot and Radhia Cousot. “Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints”. In: *POPL 77'*. ACM, 1977, pp. 238–252. DOI: 10.1145/512950.512973.

[CCM13] Patrick Cousot, Radhia Cousot, and Laurent Mauborgne. “Theories, Solvers and Static Analysis by Abstract Interpretation”. In: *J. ACM* 59.6 (Jan. 2013). DOI: 10.1145/2395116.2395120.

[Cou20] Patrick Cousot. “The Symbolic Term Abstract Domain”. In: *TASE* (Dec. 2020). URL: https://sei.ecnu.edu.cn/tase2020/file/video-slides-PCousot-TASE-2020.pdf.

[Cou77] Patrick Cousot. *Asynchronous iterative methods for solving a fixed point system of monotone equations in a complete lattice*. Research Report 88. Grenoble, France: Laboratoire IMAG, Université scientifique et médicale de Grenoble, Sept. 1977, p. 15.

[CRK18] Aziem Chawdhary, Ed Robbins, and Andy King. "Incrementally closing octagons". In: *Formal Methods in System Design* (Jan. 2018). DOI: 10.1007/s10703-017-0314-7.

[DHK13] Vijay D'Silva, Leopold Haller, and Daniel Kroening. "Abstract Conflict Driven Learning". In: *POPL '13*. ACM, 2013, pp. 143–154. DOI: 10.1145/2429069.2429087.

[Min06] A. Miné. "The octagon abstract domain". In: *Higher-Order and Symbolic Computation (HOSC)* 19.1 (2006), pp. 31–100. DOI: 10.1007/s10990-006-8609-1.

[Pel+13] Marie Pelleau et al. "A constraint solver based on abstract domains". In: *VMCAI 13'*. Springer, 2013, pp. 434–454. DOI: 10.1007/978-3-642-35873-9_26.

[Tal+19] Pierre Talbot et al. "Combining Constraint Languages via Abstract Interpretation". In: *31st IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2019)*. 2019, pp. 50–58. DOI: 10.1109/ICTAI.2019.00016.

[Tal+21] Pierre Talbot et al. "Abstract Constraint Programming". In: (2021). draft. URL: http://hyc.io/papers/abstract-cp.pdf.

[TMT20] Pierre Talbot, Éric Monfroy, and Charlotte Truchet. "Modular Constraint Solver Cooperation via Abstract Interpretation". In: *Theory and Practice of Logic Programming* 20.6 (2020), pp. 848–863. DOI: 10.1017/S1471068420000162.