Tractable Constraints in Finite Semilattices
Jakob Rehof, Torben Mogensen

Presented by Divya Muthukumaran
Constraint Satisfaction Problem

- Constraint Satisfaction Problem (CSP) Instance:
  - $\mathcal{N}$: Finite set of variables; e.g., \{a, b, c, d\}
  - $\mathcal{D}$: Domain of values; e.g., \{0, 1\}
  - $\mathcal{C}$: Set of constraints
    - \{C(S_1), C(S_2),..., C(S_c)\},
      - $S_i$: Ordered subset of $\mathcal{N}$; e.g., \{a, b, c\}
      - $C(S_i)$: Mutually compatible values for variables in $S_i$
  - **Solution to CSP**: Assignment of values to variables in $\mathcal{N}$, consistent with all constraints in $\mathcal{C}$
Example

• Assignment of values to variables $N=\{a,b,c,d\}$
• $C=\{C_0, C_1, C_2, C_3\}$
  – $C_0 = \{(1,1,1,1),(1,0,1,1),(0,1,1,0),(1,0,1,0)\}$
  – $C_1 = \{(0,1,1,0),(1,0,0,1),(1,0,1,0),(1,0,1,1)\}$
  – $C_2 = \{(1,1,1,1),(1,1,1,0),(0,1,1,1),(1,0,1,0)\}$
  – $C_3 = \{(1,0,0,1),(1,0,1,0),(1,0,1,1),(0,1,1,1)\}$
Tractability of the CSP

• [Mackworth77] CSP is NP-Complete.
• In practice, problems have special properties
  – Allow them to be solved efficiently
• **Tractable**: A CSP is tractable if there is a PTIME solution to it.
• Identifying restrictions to the general problem that ensures tractability
  – Structure of Constraints
  – Nature of Constraints
  – Restrictions on domains
Quest for tractability

• [Schaefer78] Studied the CSP problem for Boolean variables
  – States the necessary and sufficient conditions under which a set S of Boolean relations yield polynomial-time problems when the relations of S are used to constrain some of the propositional variables.
  – Identified four classes of sets of Boolean relations for which CSP is in P and proves that all other sets of relations generate an NP-complete problem.

• [Jeavons95] Generalization of Schaefer’s results
  – Identified four classes of tractable constraints, ensuring tractability in whatever way these classes were combined
  – All of them were characterized by a simple algebraic closure condition
    • Tractability is very closely linked to algebraic properties
Jeavons’ Classification

• **Class 0**: Any set of constraints, allows some constant value $d$ to be assigned to every variable.

• **Class I**: Any set of binary constraints which are 0/1/all.

• **Class II**: Any set of constraints on ordered domains, each constraint is closed under an ACI operation.

• **Class III**: Any set of constraints in which each constraint corresponds to a set of linear equations.
Tractable constraints in a POSET

• [Pratt-Tiuryn96]
  – The structure of posets are important for tractability
  – Some structures are intractable – Example: Crowns

• [Rehof-Mogensen99]
  – Tractable constraints in finite semi-lattices
    • Shows how to solve certain classes of constraints over finite domains efficiently
    • Characterize those that are not tractable
    • Can help programmers identify when an analysis
Tractable constraints in Finite Semilattices

• Deals with Definite Inequalities:
  – Evolved from the notion of Horn clauses
  – Two point Boolean lattices -> arbitrary finite semi-lattices

• Developed an algorithm ‘D’ with properties
  – Algorithm runs in linear time for any fixed finite semilattice
  – Can serve as a general-purpose off-the-shelf solver for a whole range of program analyses
Only Definite Constraints?

• The algorithm only applies to definite constraints
• Can other constraints be transformed into definite constraints?
• If yes, then
  – What is the cost of this transformation?
Monotone Function Problem

• $P$: Poset
• $F$: Finite set of monotone functions $f$ with arity $a^f$.
• $\phi = (P,F)$ is a monotone function problem
• $T_\phi$: Is the set of $\phi$ terms of range,
  - $T_\phi ::= \alpha \mid c \mid f(T_1,\ldots,T_{af})$
• $A$ – Collection of constants and variables
• $\rho : V \rightarrow P$,
  - $\rho$: Valuation of all variables
  - $\rho(\alpha)$: value assigned to $\alpha$
Constraint Satisfiability

• **Constraint Set C over ϕ**
  – Set of inequalities \( \tau \leq \tau' \mid \tau, \tau' \in T_\phi \)

• \( \rho \) is a valuation of C in P
  – \( \rho \in P^m \), satisfies C iff the constraint holds under the valuation
    • \( \rho (\tau) \leq \rho (\tau') \) holds for every \( \tau \leq \tau' \) in C

• C is satisfiable only if there is a \( \rho \in P^m \) that satisfies C

• \( \phi\text{-SAT} : \) Given C over \( \phi \), is C satisfiable?
More Definitions....

• **Definite Constraint Set:**
  
  – A constraint set in which every inequality is of the form
    \[ \tau \leq A \]
  
  – \( C = \{ \tau_i \leq A_i \} \) can be written \( C = C_{\text{var}} \cup C_{\text{cnst.}} \).

• **Simple terms**
  
  – Has no nested function applications

• **L-Normalization:**
  
  – \( C' \cup \{ f(\ldots g(\tau)) \leq A \} \rightarrow_L C' \cup \{ f(\ldots v_m \ldots) \leq A, g(\tau) \leq v_m \} \)
  
  – Monotonicity guarantees that this is equivalent to the original constraint set
• $\rho(\beta) = \bot$ for all $\beta \in V$
• $WL = \{\tau \leq \beta \mid L, \rho \text{ does not entail } \tau \leq \beta\}$
• While $WL \neq \emptyset$
  – $\tau \leq \beta = \text{POP}(WL)$
  – If $L, \rho \text{ does not entail } \tau \leq \beta$
    • $\rho(\beta) = \rho(\beta) \lor \rho(\tau)$
    • For each $\tau' \leq \alpha \in C$ with $\beta \in \text{Vars}(\tau')$
      – $WL = WL \cup \{\tau' \leq \alpha\}$
• For each $\tau \leq L \in C$
  – If $L, \rho \text{ does not entail } \tau \leq L$
    • raise exception
• return $\rho$
• $\rho(\beta) = \bot$ for all $\beta \in V$

• $WL = \{\tau \leq \beta \mid L, \rho \text{ does not entail } \tau \leq \beta\}$

• While $WL \neq \emptyset$
  – $\tau \leq \beta = \text{POP}(WL)$
  – If $L, \rho$ does not entail $\tau \leq \beta$
    • $\rho(\beta) = \rho(\beta) \lor \rho(\tau)$
    • For each $\tau' \leq \alpha \in C$ with $\beta \in \text{Vars}(\tau')$ \[ \rho \text{ does not entail } \tau \leq \beta \]
      – $WL = WL \cup \{\tau' \leq \alpha\}$

• For each $\tau \leq c \in C$
  – If $L, \rho$ does not entail $\tau \leq c$
    • raise exception

• return $\rho$
RM Example

- $C = \{L_1 \leq \beta_0, \ L_2 \land \beta_0 \leq \beta_1, \ \beta_0 \land \beta_1 \leq \beta_2\}$
- $\beta_0 = \bot \quad \beta_1 = \bot \quad \beta_2 = \bot$
  
  - $L_1 \leq \beta_0 \Rightarrow \beta_0 = L_1$

- $\beta_0 = L_1 \quad \beta_1 = \bot \quad \beta_2 = \bot$
  
  - $L_2 \land \beta_0 \leq \beta_1 \Rightarrow \beta_1 = L_1 \land L_2$

- $\beta_0 = L_1 \quad \beta_1 = L_1 \land L_2 \quad \beta_2 = \bot$
  
  - $\beta_0 \land \beta_1 \leq \beta_2 \Rightarrow \beta_2 = L_1 \land L_2$

- $\beta_0 = L_1 \quad \beta_1 = L_1 \land L_2 \quad \beta_2 = L_1 \land L_2$
• $\rho(\beta) = \bot$ for all $\beta \in V$

• $WL = \{\tau \leq \beta | L, \rho \text{ does not entail } \tau \leq \beta\}$

• While $WL \neq \emptyset$
  
  – $\tau \leq \beta = \text{POP}(WL)$

  – If $L, \rho$ does not entail $\tau \leq \beta$
    
    • $\rho(\beta) = \rho(\beta) \lor \rho(\tau)$
    
    • For each $\tau' \leq \alpha \in C$ with $\beta \in Vars(\tau')$
      
      – $WL = WL \cup \{\tau' \leq \alpha\}$

• For each $\tau \leq c \in C$
  
  – If $L, \rho$ does not entail $\tau \leq c$
    
    • raise exception

• return $\rho$

Valuation at $\beta$ strictly increases
It can only increase till $h(L)$
Total number of constraints added to $WL$ each time bounded by $|C|
Total checks done is bound by $h(L).|C|$
Extensions

• **To a finite meet-semilattice:**
  – Add top element to \( P \)
  – If any atom is valued at top then FAIL

• **Relational constraints (RC):**
  – Inequality constraints special case of RC’s
  – A RCP is a pair \( \Gamma=\{P,S\} \) with \( P: \) finite poset, \( S: \) finite set of relations over \( P \)
  – A RCP is satisfiable if there exists a valuation \( \rho \) of \( C \) in \( P \) s.t.
    \[ (\rho(A_1), \ldots, \rho(A_{a_R})) \in R \text{ for every } R(A_1, \ldots, A_{a_R}) \]
Relational Constraints

• How many relational constraint problems can be efficiently solved using algorithm D?
  – How many problems can be transformed into definite inequality problems and what is the cost of the transformation?
  – Characterize the class of relational problem that can be solved by the algorithm D as follows
  – Let $\Gamma = \{P, S\}$ where $P$ : meet-semilattice, then it can be represented as a definite inequality problem iff $\Gamma$ is meet-closed.
  – C over $\Gamma$ can be represented by a definite a simple constraint set $C'$ with $|C'| \leq m(m+2).|C|$
Boolean Representation

• Translating sets of **definite inequalities** to **propositional formulae**
  – Direct correspondence between solutions to the propositional system and solutions to the lattice inequalities.

• Translation to Boolean constraints will **expand exponentially** in the arity of functions in F
  – This conversion should only be done when the function arities are small.

• Satisfiability of translation: Each constraint in the translation is of the form
  – $a_1 \land a_2 \land a_3 \land ... \land a_m \leq a_0$ where $a_i$ are atoms ranging over $\{0,1\}$.
  – Isomorphic to **Horn-clauses**, can be solved in time linear in the size of the constraint set using the algorithm for **HORNSAT**
Extensibility

- Can algorithm be extended to cover more relations than the meet-closed ones?
- Proved that no such extension is possible for any meet-semilattice L
  - “Algorithm D is complete for a maximal tractable class of problems i.e. meet closed ones”
Program flow as constraints

• Check if program enforces information safety.
• Information security policy specified as a lattice.
• Variables in program assigned labels from lattice.
• Generate flow constraints from program.
Program Flow security as Constraints

• *Security enforcing compilers* verify that a program correctly enforces a security policy.
Program Flow security as Constraints

• *Security enforcing compilers* verify that a program correctly enforces a security policy.

• Programmer specifies a policy as a *security lattice*. 
Program Flow security as Constraints

- **Security enforcing compilers** verify that a program correctly enforces a security policy.
- Programmer specifies a policy as a *security lattice*.
  - Lattice $L$ governs security, contains levels $l$ related by $\leq$.
  - If $l \leq l'$, then $l$ is allowed to flow to $l'$.
  - *Information Flow Security*: Information at a level $l$ can only affect information for all $l'$ such that $l \leq l'$. 
Program Flow security as Constraints

• *Security enforcing compilers* verify that a program correctly enforces a security policy.

• Programmer specifies a policy as a *security lattice*.

• Compiler performs source code analysis to identify *information flows*.
  – If *a* flows to *b*, the constraint $L(a) \preceq L(b)$ is generated.
  – Type system for constraints.
Program Flow security as Constraints

• **Security enforcing compilers** verify that a program correctly enforces a security policy.

• Programmer specifies a policy as a security lattice.

• Compiler performs source code analysis to identify *information flows*.

• Flags *information flow errors*.
  
  – There exists a constraint $L(a) \leq L(b)$ that is not satisfied.
Program Flow security as Constraints

• Constraint type system:
  – \( v = e \iff L(e) \leq L(v) \)

• Method calls:
  – Actual Call: \( x(a_1, a_2, \ldots, a_n) \)
  – Method Signature: \( x(f_1, f_2, \ldots, f_n) \)
  – \( L(a_i) \leq L(f_i) \) for \( 1 \leq i \leq n \)

• Similar idea for returns.
Context sensitivity

Example:

```c
int sum(int x, int y) {
    int z;
    z = x * y;
    return z;
}
```

```c
int main{
    int a __secret__, b, c, d, p, q __public__;
    p = sum(a, b);
    q = sum(c, d); }
```

Constraints

- Secret $\leq$ L(a)
- L(a) $\leq$ L(x), L(c) $\leq$ L(x)
- L(b) $\leq$ L(y), L(d) $\leq$ L(y)
- L(x) $\leq$ L(z), L(y) $\leq$ L(z)
- L(z) $\leq$ L(p), L(z) $\leq$ L(q)
- L(q) $\leq$ Public

• Constraints will fail if contexts are not separated.
Context sensitivity

Example:

```c
int sum(int x, int y) {
    int z;
    z=x*y;
    Return z; }
```

```c
int main{
    int a __secret__, b, c, d, p, q __public__;
    p=sum(a, b);
    q=sum(c, d); }
```

- Constraints will not fail; valuation exists.

Constraints

- Secret $\preccurlyeq L(a)$
- $L(a) \preccurlyeq L(x_1)$, $L(c) \preccurlyeq L(x_2)$
- $L(b) \preccurlyeq L(y_1)$, $L(d) \preccurlyeq L(y_2)$
- $L(x_1) \preccurlyeq L(z_1)$, $L(y_1) \preccurlyeq L(z_1)$
- $L(x_2) \preccurlyeq L(z_2)$, $L(y_2) \preccurlyeq L(z_2)$
- $L(z_1) \preccurlyeq L(p)$, $L(z_2) \preccurlyeq L(q)$
- $L(q) \preccurlyeq$ Public