Computational Logic
CLP Semantics and Fundamental Results
Constraint Domains

- Semantics parameterized by the constraint domain:
  CLP(\mathcal{X}), where \mathcal{X} \equiv (\Sigma, D, \mathcal{L}, \mathcal{T})

- Signature \Sigma: set of predicate and function symbols, together with their arity

- \mathcal{L} \subseteq \Sigma--formulae: constraints

- D is the set of actual elements in the domain

- \Sigma--structure D: gives the meaning of predicate and function symbols (and hence, constraints).

- \mathcal{T} a first–order theory (axiomatizes some properties of D)

- (D, \mathcal{L}) is a constraint domain

- Assumptions:
  - \mathcal{L} built upon a first–order language
  - =\in \Sigma is identity in D
  - There are identically false and identically true constraints in \mathcal{L}
  - \mathcal{L} is closed w.r.t. renaming, conjunction and existential quantification
Domains (I)

- \( \Sigma = \{0, 1, +, *, =, <, \leq\} \), \( D = \mathbb{R} \), \( D \) interprets \( \Sigma \) as usual, \( \mathbb{R} = (D, \mathcal{L}) \)
  - Arithmetic over the reals
  - Eg.: \( x^2 + 2xy < \frac{y}{x} \wedge x > 0 \) (\( \equiv xxx + xxy + xxy < y \wedge 0 < x \))

- Question: is 0 needed? How can it be represented?

- Let us assume \( \Sigma' = \{0, 1, +, =, <, \leq\} \), \( \mathbb{R}_{Lin} = (D', \mathcal{L}') \)
  - Linear arithmetic
  - Eg.: \( 3x - y < 3 \) (\( \equiv x + x + x < 1 + 1 + 1 + y \))

- Let us assume \( \Sigma'' = \{0, 1, +, =\} \), \( \mathbb{R}_{LinEq} = (D'', \mathcal{L}'') \)
  - Linear equations
  - Eg.: \( 3x + y = 5 \wedge y = 2x \)
Domains (II)

- $\Sigma = \{ \langle \text{constant and function symbols} \rangle, = \}$
- $D = \{ \text{finite trees} \}$
- $D$ interprets $\Sigma$ as tree constructors
- Each $f \in \Sigma$ with arity $n$ maps $n$ trees to a tree with root labeled $f$ and whose subtrees are the arguments of the mapping
- Constraints: syntactic tree equality
- $\mathcal{FT} = (D, \mathcal{L})$
  - Constraints over the Herbrand domain
  - Eg.: $g(h(Z), Y) = g(Y, h(a))$
- $LP \equiv CLP(\mathcal{FT})$
Domains (III)

• $\Sigma = \{ <\text{constants}>, \lambda, \_, ::, = \}$
• $D = \{ \text{finite strings of constants} \}$
• $D$ interprets \_ as string concatenation, :: as string length
  ◦ Equations over strings of constants
  ◦ Eg.: $X.A.X = X.A$

• $\Sigma = \{ 0, 1, \neg, \land, = \}$
• $D = \{ \text{true, false} \}$
• $D$ interprets symbols in $\Sigma$ as boolean functions
• $BOOL = (D, L)$
  ◦ Boolean constraints
  ◦ Eg.: $\neg(x \land y) = 1$
CLP(\mathcal{X}) Programs

- Recall that:
  - $\Sigma$ is a set of predicate and function symbols
  - $\mathcal{L} \subseteq \Sigma$–formulae are the constraints
- $\Pi$: set of predicate symbols definable by a program
- Atom: $p(t_1, t_2, \ldots, t_n)$, where $t_1, t_2, \ldots, t_n$ are terms and $p \in \Pi$
- Primitive constraint: $p(t_1, t_2, \ldots, t_n)$, where $t_1, t_2, \ldots, t_n$ are terms and $p \in \Sigma$ is a predicate symbol
- Every constraint is a (first–order) formula built from primitive constraints
- The class of constraints will vary (generally only a subset of formulas are considered constraints)
- A CLP program is a collection of rules of the form $a \leftarrow b_1, \ldots, b_n$ where $a$ is an atom and the $b_i$’s are atoms or constraints
- A fact is a rule $a \leftarrow c$ where $c$ is a constraint
- A goal (or query) $G$ is a conjunction of constraints and atoms
Basic Operations on Constraints

- Constraint domains are expected to support some basic operations on constraints
  
  1. Consistency (or satisfiability) test: $\mathcal{D} \models \exists c,$
  
  2. Implication or entailment: $\mathcal{D} \models c_0 \rightarrow c_1,$
  
  3. Projection of a constraint $c_0$ onto variables $\tilde{x}$ to obtain a constraint $c_1$ such that $\mathcal{D} \models c_1 \leftrightarrow \exists_{\tilde{x}} c_0,$

  4. Detection of uniqueness of variable value: $\mathcal{D} \models c(x, \tilde{z}) \land c(y, \tilde{w}) \rightarrow x = y$

- Actually, only the first one is really required

- In actual implementations, some of these operations—in particular the test of consistency—may be incomplete

- Examples:
  
  - $x \ast x < 0$ is inconsistent in $\mathbb{R}$ (because $\neg \exists x \in \mathbb{R} : x \ast x < 0$)
  
  - $\mathcal{D} \models (x \land y = 1) \rightarrow (x \lor y = 1)$ in $\mathbb{BOOL}$

  - In $\mathcal{FT}$, the projection of $x = f(y) \land y = f(z)$ on $\{x, z\}$ is $x = f(f(z))$

  - In $\mathbb{WE}$, $\mathcal{D} \models x.a.x = x.a \land y.b.y = y.b \rightarrow x = y$

- Prove the last assertion!
Properties of CLP Languages

- $\mathcal{T}$ axiomatizes some of the properties of $\mathcal{D}$

- For a given $\Sigma$, let $(\mathcal{D}, \mathcal{L})$ be a constraint domain with signature $\Sigma$, and $\mathcal{T}$ a $\Sigma$–theory.

- $\mathcal{D}$ and $\mathcal{T}$ correspond on $\mathcal{L}$ if:
  - $\mathcal{D}$ is a model of $\mathcal{T}$, and
  - for every constraint $c \in \mathcal{L}$, $\mathcal{D} \models \exists c$ iff $\mathcal{T} \models \exists c$.

- $\mathcal{T}$ is satisfaction complete with respect to $\mathcal{L}$ if for every constraint $c \in \mathcal{L}$, either $\mathcal{T} \models \exists c$ or $\mathcal{T} \models \neg \exists c$.

- $(\mathcal{D}, \mathcal{L})$ is solution compact if
  \[
  \forall c \exists \left\{c_i\right\}_{i \in I} : \mathcal{D} \models \forall \tilde{x} \neg c(\tilde{x}) \iff \bigvee_{i \in I} \bigwedge_{i \in I} c_i(\tilde{x})
  \]

  i.e., any negated constraint in $\mathcal{L}$ can be expressed as a (in)finite disjunction of constraints.
Solution Compactness

- Important to lift SLDNF results to CLP(\(\mathcal{X}\))
- We have to deal only with user predicates
- E.g.
  - \(x \not\geq y\) in CLP(\(\mathcal{R}\)) is \(x < y\)
  - \(x \neq y\) in CLP(\(\mathcal{R}\)) is \(x < y \lor y < x\)
  - \(\mathcal{R}_{Lin}\) with constraint \(x \neq \pi\) is not s.c.
- How can we express \(x \neq y\) in CLP(\(\mathcal{F}_T\))?
Logical Semantics (I)

- Two common logical semantics exist.
- The first one interprets a rule
  \[ p(\tilde{x}) \leftarrow b_1, \ldots, b_n \]
  as the logic formula
  \[ \forall \tilde{x}, \tilde{y} \; p(\tilde{x}) \lor \neg b_1 \lor \ldots \lor \neg b_n \]
Logical Semantics (II)

- The second one associates a logic formula to each predicate in \(\Pi\)
  - If the set of rules of \(P\) with \(p\) in the head is:
    
    \[
    \begin{align*}
    p(\vec{x}) & \leftarrow B_1 \\
    p(\vec{x}) & \leftarrow B_2 \\
    \vdots \\
    p(\vec{x}) & \leftarrow B_n
    \end{align*}
    \]

    then the formula associated with \(p\) is:
    
    \[
    \forall \vec{x} \ p(\vec{x}) \leftrightarrow \exists \vec{y}_1 B_1 \\
    \quad \lor \exists \vec{y}_2 B_2 \\
    \quad \vdots \\
    \quad \lor \exists \vec{y}_n B_n
    \]

  - If \(p\) does not occur in the head of a rule of \(P\), the formula is: \(\forall \vec{x} \neg p(\vec{x})\)
  - The collection of all such formulas is the *Clark completion* of \(P\) (denoted by \(P^*\))

- These two semantics differ on the treatment of the treatment of the negation
A *valuation* is a mapping from variables to $D$, and the natural extension which maps terms to $D$ and formulas to closed $L^*$–formulas.

A $D$–interpretation of a formula is an interpretation of the formula with the same domain as $D$ and the same interpretation for the symbols in $\Sigma$ as $D$.

It can be represented as a subset of $B_D$ where

$$B_D = \{ p(\tilde{d}) \mid p \in \Pi, \tilde{d} \in D^k \}$$

A $D$–model of a closed formula is a $D$–interpretation which is a model of the formula.

The usual logical semantics is based on the $D$–models of $P$ and the models of $P^*, \mathcal{T}$.

The least $D$–model of a formula $Q$ is denoted by $lm(Q, D)$.

A *solution* to a query $G$ is a valuation $v$ such that $v(G) \subseteq lm(P, D)$. 

---

Logical Semantics (III)
Fixpoint Semantics

- Based on one-step consequence operator $T^D_P$ (also called “immediate consequence operator”).

- Take as semantics $lfp(T^D_P)$, where:

  $$T^D_P(I) = \{ p(\tilde{d}) \mid p(\tilde{x}) \leftarrow c, b_1, \ldots, b_n \in P, a_i \in I, \quad D \models v(c), v(\tilde{x}) = \tilde{d}, v(b_i) = a_i \}$$

- Theorems:

  1. $T^D_P \uparrow \omega = lfp(T^D_P)$
  2. $lm(P, D) = lfp(T^D_P)$
Top–Down Operational Semantics (I)

- General framework for operational semantics
- Formalized as a transition system on states
- State: a 3–tuple $\langle A, C, S \rangle$, or fail, where
  - $A$ is a multiset of atoms and constraints,
  - $C \cup S$ multiset of constraints,
  - $C$, active constraints (awake)
  - $S$, passive constraints (asleep)
- Computation and Selection rules depend on $A$
- Transition system: parameterized by a predicate $\text{consistent}$ and a function $\text{infer}$:
  - $\text{consistent}(C)$ checks the consistency of a constraint store
  - Usually “$\text{consistent}(C)$ iff $D \models \exists c$”, but sometimes “if $D \models \exists c$ then $\text{consistent}(C)$”
  - $\text{infer}(C, S)$ computes a new set of active and passive constraints
Top–Down Operational Semantics (II)

- Transition $r$: computation step; rewriting using user predicates
  \[
  \langle A \cup a, C, S \rangle \rightarrow_r \langle A \cup B, C, S \cup (a = h) \rangle
  \]
  if $h \leftarrow B \in P$, and $a$ and $h$ have the same predicate symbol, or
  \[
  \langle A \cup a, C, S \rangle \rightarrow_r \text{fail}
  \]
  if there is no rule $h \leftarrow B$ of $P$ such that $a$ and $h$ have the same predicate symbol
  ($a = h$ is a set of argument–wise equations) if $a$ is a predicate symbol selected by the computation rule

- Transition $c$: selects constraints
  \[
  \langle A \cup c, C, S \rangle \rightarrow_c \langle A, C, S \cup c \rangle
  \]
  if $c$ is a constraint selected by the computation rule

- Transition $i$: infers new constraints
  \[
  \langle A, C, S \rangle \rightarrow_i \langle A, C', S' \rangle \text{ if } (C', S') = \text{infer}(C, S)
  \]
  ◦ In particular, may turn passive constraints into active ones

- Transition $s$: checks satisfiability
  \[
  \langle A, C, S \rangle \rightarrow_s \begin{cases} 
  \langle A, C, S' \rangle & \text{if consistent}(C) \\
  \text{fail} & \text{if } \neg\text{consistent}(C')
  \end{cases}
  \]
Top–Down Operational Semantics (III)

- Initial state: $\langle G, \emptyset, \emptyset \rangle$

- Derivation: $\langle A_1, C_1, S_1 \rangle \rightarrow \ldots \rightarrow \langle A_i, C_i, S_i \rangle \rightarrow \ldots$

- Final state: $E \rightarrow E$

- **Successful derivation**: final state $\langle \emptyset, C, S \rangle$

- A derivation *flounders* if finite and the final state is $\langle A, C, S \rangle$ with $A \neq \emptyset$

- A derivation is *failed* if it is finite and the final state is fail

- Answer: $\exists_{\tilde{x}} C \land S$, where $\tilde{x}$ are the variables in the initial goal

- A derivation is *fair* if it is failed or, for every $i$ and every $a \in A_i$, $a$ is rewritten in a later transition

- A computation rule is fair if it gives rise only to fair derivations
Top–Down Operational Semantics (IV)

• Computation tree for goal $G$ and program $P$:
  ◦ Nodes labeled with states
  ◦ Edges labeled with $\rightarrow_r$, $\rightarrow_c$, $\rightarrow_i$ or $\rightarrow_s$
  ◦ Root labeled by $\langle G, \emptyset, \emptyset \rangle$
  ◦ All sons of a given node have the same label
  ◦ Only one son with transitions $\rightarrow_c$, $\rightarrow_i$ or $\rightarrow_s$
  ◦ A son per program clause with transition $\rightarrow_r$
Consider the program
\[ p(X + 3, X) \leftarrow X < 3. \]
\[ p(X + 3, X) \leftarrow X > 3, p(X, Y). \]
and the goal \( \leftarrow p(5, X) \)

A possible computation tree is:

- Dotted rectangle: previous state was final as well
Types of CLP(\(\mathcal{X}\)) Systems

- **Quick-checking** CLP(\(\mathcal{X}\)) system: its operational semantics can be described by 
  \[ r_i \rightarrow s \equiv r \rightarrow r \rightarrow i \rightarrow s \quad \text{and} \quad c_i \rightarrow s \equiv c \rightarrow c \rightarrow i \rightarrow s \]
- i.e., always selects either an atom or a constraint, infers and checks consistency

- **Progressive** CLP system: for all \(\langle A, C, S \rangle\) with \(A \neq \emptyset\), every derivation from that state either fails or contains a \(\rightarrow r\) or \(\rightarrow c\) transition

- **Ideal** CLP system:
  - Quick-checking
  - Progressive
  - \(\text{infer}(C, S) = (C \cup S, \emptyset)\)
  - \(\text{consistent}(C)\) holds iff \(\mathcal{D} \models \exists c\)
Soundness and Completeness Results

- **Success set:** the set of queries plus constraints which have a successful derivation in the program:
  \[ SS(P) = \{ p(\bar{x}) \leftarrow c \mid \langle p(\bar{x}), \emptyset, \emptyset \rangle \rightarrow^* \langle \emptyset, c', c'' \rangle, D \models c \leftrightarrow \exists_{-\bar{x}} c' \land c'' \} \]

- Consider a program \( P \) in the CLP language determined by a 4–tuple \( (\Sigma, D, L, T) \) and executing on an *ideal* CLP system. Then:
  
  1. \([SS(P)]_D = \text{lm}(P, D)\), where
     \[ [SS(P)]_D = \{ v(a) \mid (a \leftarrow c) \in SS(P), D \models v(c) \} \]
  2. \( SS(P) = \text{lfp}(S^D_P) \)
  3. (Soundness) if the goal \( G \) has a successful derivation with answer constraint \( c \), then \( P, T \models c \rightarrow G \)
  4. (Completeness) if \( P, T \models c \rightarrow G \) then there are derivations for the goal \( G \) with answer constraints \( c_1, \ldots, c_n \) such that \( T \models c \rightarrow \bigvee_{i=1}^n c_i \)
  5. Assume \( T \) is satisfaction complete w.r.t. \( L \). Then the goal \( G \) is finitely failed for \( P \) iff \( P^*, T \models \neg G \).
Negation in CLP(\(\mathcal{X}\))

- Most LP results can be lifted to CLP(\(\mathcal{X}\))
- In particular, negation as failure (à la SLDNF) is still valid using:
  - Satisfiability instead of unification
  - Variable elimination instead of groundness
- Added bonus: if the system is `solution compact`, then negated constraints can be expressed in terms of primitive constraints
- Less chances of a floundered / incorrect computation