Constraint Domains

- Semantics parameterized by the constraint domain: $\text{CLP}(\mathcal{X})$, where $\mathcal{X} \equiv (\Sigma, D, L, T)$
- Signature $\Sigma$: set of predicate and function symbols, together with their arity
- $L \subseteq \Sigma$–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).
- $T$ a first–order theory (axiomatizes some properties of $D$)
- $(D, L)$ is a constraint domain
- Assumptions:
  - $\mathcal{L}$ built upon a first–order language
  - $\equiv \in \Sigma$ is identity in $D$
  - There are identically false and identically true constraints in $L$
  - $L$ is closed w.r.t. renaming, conjunction and existential quantification
Domains (I)

- $\Sigma = \{0, 1, +, *, =, <, \leq\}$, $D = \mathbb{R}$, $D$ interprets $\Sigma$ as usual, $R = (D, \mathcal{L})$
  - Arithmetic over the reals
  - Eg.: $x^2 + 2xy < \frac{y}{2} \land x > 0$ ($\equiv xxx + xxy + xxy < y \land 0 < x$)
- Question: is 0 needed? How can it be represented?

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

- Let us assume $\Sigma'' = \{0, 1, +, =\}$, $R_{LinEq} = (D'', \mathcal{L}'')$
  - Linear equations
  - Eg.: $3x + y = 5 \land y = 2x$

Domains (II)

- $\Sigma = \{<\text{constant and function symbols}>, =\}$
- $D = \{\text{finite trees}\}$
- $D$ 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
- $\mathcal{F}T = (D, \mathcal{L})$
  - Constraints over the Herbrand domain
  - Eg.: $g(h(Z), Y) = g(Y, h(a))$
- $\text{LP} \equiv \text{CLP}(\mathcal{F}T)$
Domains (III)

- \( \Sigma = \{ \text{<constants>, } \lambda, \cdot, ::, = \} \)
- \( D = \{ \text{finite strings of constants} \} \)
- \( D \) interprets \( \cdot \) 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
- \( \mathbb{BOOL} = (D, L) \)
  - Boolean constraints
  - Eg.: \( \neg(x \land y) = 1 \)

CLP(\( \mathcal{X} \)) Programs

- Recall that:
  - \( \Sigma \) is a set of predicate and function symbols
  - \( L \subseteq \Sigma \)–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 $\bar{x}$ to obtain a constraint $c_1$ such that $\mathcal{D} \models c_1 \leftarrow \exists_{-\bar{x}} c_0$,
  4. Detection of uniqueness of variable value: $\mathcal{D} \models c(x, \bar{z}) \land c(y, \bar{w}) \rightarrow x = y$

- 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:
  - $x \star x < 0$ is inconsistent in $\mathcal{R}$ (because $\neg \exists x \in \mathcal{R} : x \star x < 0$)
  - $\mathcal{D} \models (x \land y = 1) \rightarrow (x \lor y = 1)$ in $BOOL$
  - In $\mathcal{F}T$, the projection of $x = f(y) \land y = f(z)$ on $\{x, z\}$ is $x = f(f(z))$
  - In $\mathcal{W}E$, $\mathcal{D} \models x.a = x.a \land y.b = y.b \rightarrow x = y$

- 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
  $$\forall c \exists \{c_i\}_{i \in I} : \mathcal{D} \models \forall \bar{x}. \neg c(\bar{x}) \iff \bigvee_{i \in I} c_i(\bar{x})$$
  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.
  - \(x \not< y\) in CLP(\(\mathcal{X}\)) is \(x < y\)
  - \(x \neq y\) in CLP(\(\mathcal{X}\)) is \(x < y \lor y < x\)
  - \(R_{Lin}\) with constraint \(x \neq \pi\) is not s.c.
- How can we express \(x \neq y\) in CLP(\(F\mathcal{T}\))?

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:
    \[
    p(\bar{x}) \leftarrow B_1 \\
    p(\bar{x}) \leftarrow B_2 \\
    \vdots \\
    p(\bar{x}) \leftarrow B_n
    \]
  - then the formula associated with \( p \) is:
    \[
    \forall \bar{x} \ p(\bar{x}) \leftarrow \exists \bar{y}_1 B_1 \\
    \lor \exists \bar{y}_2 B_2 \\
    \vdots \\
    \lor \exists \bar{y}_n B_n
    \]
  - If \( p \) does not occur in the head of a rule of \( P \), the formula is: \( \forall \bar{x} \neg p(\bar{x}) \)
  - The collection of all such formulas is the Clark completion of \( P \) (denoted by \( P^* \))
- These two semantics differ on the treatment of the negation

Logical Semantics (III)

- 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
  \[
  B_D = \{ p(\bar{d}) \mid p \in \Pi, \bar{d} \in D^k \}
  \]
- 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 \( P^*, T \).
- The least \( D \)–model of a formula \( Q \) is denoted by \( \text{lm}(Q, D) \).
- A solution to a query \( G \) is a valuation \( v \) such that \( v(G) \subseteq \text{lm}(P, D) \).
Fixpoint Semantics

- Based on one-step consequence operator $T^D_P$ (also called “immediate consequence operator”).
- Take as semantics $\text{lfp}(T^D_P)$, where:
  \[
  T^D_P(I) = \{p(\vec{d}) \mid p(\vec{x}) \leftarrow c, b_1, \ldots, b_n \in P, a_i \in I, \ D \models v(c), v(\vec{x}) = \vec{d}, v(b_i) = a_i\}
  \]
- Theorems:
  1. $T^D_P \uparrow \omega = \text{lfp}(T^D_P)$
  2. $\text{lm}(P, D) = \text{lfp}(T^D_P)$

Top–Down Operational Semantics (I)

- General framework for operational semantics
- Formalized as a transition system on states
- State: a 3–tuple $(A, C, S)$, or $\text{fail}$, 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
  
  $$(A \cup a, C, S) \xrightarrow{r} (A \cup B, C, S \cup \{a = h\})$$
  
  if $h \leftarrow B \in P$, and $a$ and $h$ have the same predicate symbol, or
  $$(A \cup a, C, S) \xrightarrow{r} \text{fail}$$
  
  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
  
  $$(A \cup c, C, S) \xrightarrow{c} (A, C, S \cup c)$$
  
  if $c$ is a constraint selected by the computation rule

- Transition $i$: infers new constraints
  
  $$(A, C, S) \xrightarrow{i} (A, C', S')$$ if $(C', S') = \text{infer}(C, S)$
  
  ○ In particular, may turn passive constraints into active ones

- Transition $s$: checks satisfiability
  
  $$(A, C, S) \xrightarrow{s} \begin{cases} (A, C, S) & \text{if } \text{consistent}(C) \\ \text{fail} & \text{if } \neg\text{consistent}(C) \end{cases}$$

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: $\exists x C \land S$, where $x$ 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 $\rightarrow_r$

Computation Tree: Example

- 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:

  $\langle \{5, X\}, \emptyset, \emptyset \rangle$

  
  $\langle \{X < 3\}, \emptyset, \{5 = X + 3\} \rangle$

  $\langle \{X > 3, p(X, Y)\}, \emptyset, \{5 = X + 3\} \rangle$

  $\langle \{X < 3\}, \{X = 2\}, \emptyset \rangle$

  $\langle \{X > 3, p(X, Y)\}, \{X = 2\}, \emptyset \rangle$

  $\langle \emptyset, \{X = 2\}, \{X < 3\} \rangle$

  $\langle \emptyset, \{X = 2\}, \emptyset \rangle$

  $\langle \emptyset, \{X = 2\}, \emptyset \rangle$

  $\langle \{p(X, Y), \{X = 2\}, \{X > 3\} \rangle$

  $\langle \{p(X, Y), \{X = 2, X > 3\}, \emptyset \rangle$

  $\langle \emptyset, \{X = 2\}, \emptyset \rangle$

  $\langle \emptyset, \{X = 2\}, \emptyset \rangle$

  $\langle \emptyset, \{X = 2\}, \emptyset \rangle$

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

- **Quick-checking** CLP(\textsc{X}) system: its operational semantics can be described by $\rightarrow_{rs} \equiv \rightarrow_r \rightarrow_s$ and $\rightarrow_{cs} \equiv \rightarrow_c \rightarrow_s$
  
  I.e., always selects either an atom or a constraint, infers and checks consistency

- **Progressive** CLP system: for all $A; C; S$ 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: $SS(P) = \{ p(\bar{x}) \leftarrow c \mid \langle p(\bar{x}), \emptyset, \emptyset \rangle \rightarrow^* \langle \emptyset, c', c'' \rangle, \mathcal{D} \models c \rightarrow \exists x c' \land c'' \}$
  
  Consider a program $P$ in the CLP language determined by a 4–tuple $(\Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T})$ and executing on an ideal CLP system. Then:

  1. $[SS(P)]_{\mathcal{D}} = lmf(P, D)$, where $[SS(P)]_{\mathcal{D}} = \{ v(a) \mid (a \leftarrow c) \in SS(P), \mathcal{D} \models v(c) \}$
  2. $SS(P) = lfp(S^P_P)$
  3. (Soundness) if the goal $G$ has a successful derivation with answer constraint $c$, then $P, \mathcal{T} \models c \rightarrow G$
  4. (Completeness) if $P, \mathcal{T} \models c \rightarrow G$ then there are derivations for the goal $G$ with answer constraints $c_1, \ldots, c_n$ such that $\mathcal{T} \models c \rightarrow \bigvee_{i=1}^n c_i$
  5. Assume $\mathcal{T}$ is satisfaction complete w.r.t. $\mathcal{L}$. Then the goal $G$ is finitely failed for $P$ iff $P^*, \mathcal{T} \models \neg G$.  

\textbf{Translation of the Content:}

Types of CLP(\textsc{X}) Systems

- Quick-checking
  - Operational semantics: $\rightarrow_{rs} \equiv \rightarrow_r \rightarrow_s$ and $\rightarrow_{cs} \equiv \rightarrow_c \rightarrow_s$
  - Always selects either an atom or a constraint, infers and checks consistency

- Progressive
  - Derivation: every derivation from a state either fails or contains a $\rightarrow_r$ or $\rightarrow_c$

- Ideal
  - Characteristics:
    - Quick-checking
    - Progressive
    - $\text{infer}(C, S) = (C \cup S, \emptyset)$
    - $\text{consistent}(C)$ holds if $\mathcal{D} \models \exists c$

Soundness and Completeness Results

- Success set: the set of queries plus constraints with successful derivations:
  - $SS(P) = \{ p(\bar{x}) \leftarrow c \mid \langle p(\bar{x}), \emptyset, \emptyset \rangle \rightarrow^* \langle \emptyset, c', c'' \rangle, \mathcal{D} \models c \rightarrow \exists x c' \land c'' \}$

- Program execution on an ideal CLP system:
  - Soundness: if goal $G$ has a successful derivation with answer constraint $c$, then $P, \mathcal{T} \models c \rightarrow G$
  - Completeness: if $P, \mathcal{T} \models c \rightarrow G$, then derivations exist with answer constraints $c_1, \ldots, c_n$
  - Satisfaction completeness: if $\mathcal{T} \models \neg G$ for $P$, then $P^*, \mathcal{T} \models \neg G$.  

Negation in CLP(\(\lambda'\))

- Most LP results can be lifted to CLP(\(\lambda'\))
- 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