

Computational Logic

CLP Semantics and Fundamental Results

Constraint Domains

- Semantics parameterized by the constraint domain:
CLP(\mathcal{X}), where $\mathcal{X} \equiv (\Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T})$
- Signature Σ : set of predicate and function symbols, together with their arity
- $\mathcal{L} \subseteq \Sigma$ -formulae: constraints
- \mathcal{D} is the set of actual elements in the domain
- Σ -structure \mathcal{D} : gives the meaning of predicate and function symbols (and hence, constraints).
- \mathcal{T} a first-order theory (axiomatizes some properties of \mathcal{D})
- $(\mathcal{D}, \mathcal{L})$ is a *constraint domain*
- Assumptions:
 - ◇ \mathcal{L} built upon a first-order language
 - ◇ $= \in \Sigma$ is identity in \mathcal{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)

- $\Sigma = \{0, 1, +, *, =, <, \leq\}$, $D = \mathbf{R}$, \mathcal{D} interprets Σ as usual, $\mathfrak{R} = (\mathcal{D}, \mathcal{L})$
 - ◇ 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 $\Sigma' = \{0, 1, +, =, <, \leq\}$, $\mathfrak{R}_{Lin} = (\mathcal{D}', \mathcal{L}')$
 - ◇ Linear arithmetic
 - ◇ Eg.: $3x - y < 3$ ($\equiv x + x + x < 1 + 1 + 1 + y$)

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

Domains (II)

- $\Sigma = \{ \langle \text{constant and function symbols} \rangle, = \}$
- $D = \{ \text{finite trees} \}$
- \mathcal{D} interprets Σ 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
- $\mathcal{FT} = (\mathcal{D}, \mathcal{L})$
 - ◇ Constraints over the Herbrand domain
 - ◇ Eg.: $g(h(Z), Y) = g(Y, h(a))$
- $\text{LP} \equiv \text{CLP}(\mathcal{FT})$

Domains (III)

- $\Sigma = \{ \langle \text{constants} \rangle, \lambda, ., ::, = \}$
 - $D = \{ \text{finite strings of constants} \}$
 - \mathcal{D} interprets $.$ as string concatenation, $::$ as string length
 - ◇ Equations over strings of constants
 - ◇ Eg.: $X.A.X = X.A$
-

- $\Sigma = \{0, 1, \neg, \wedge, =\}$
- $D = \{ \text{true}, \text{false} \}$
- \mathcal{D} interprets symbols in Σ as boolean functions
- $\text{BOOL} = (D, \mathcal{L})$
 - ◇ Boolean constraints
 - ◇ Eg.: $\neg(x \wedge y) = 1$

CLP(\mathcal{X}) Programs

- Recall that:
 - ◇ Σ is a set of predicate and function symbols
 - ◇ $\mathcal{L} \subseteq \Sigma$ -formulae are the constraints
- Π : set of predicate symbols definable by a program
- Atom: $p(t_1, t_2, \dots, t_n)$, where t_1, t_2, \dots, t_n are terms and $p \in \Pi$
- Primitive constraint: $p(t_1, t_2, \dots, t_n)$, where t_1, t_2, \dots, 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 $a \leftarrow b_1, \dots, b_n$ 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: $\mathcal{D} \models \exists \tilde{x} c$,
 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: $\mathcal{D} \models c(x, \tilde{z}) \wedge c(y, \tilde{w}) \rightarrow x = y$
- 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:
 - ◇ $x * x < 0$ is inconsistent in \mathfrak{R} (because $\neg \exists x \in \mathfrak{R} : x * x < 0$)
 - ◇ $\mathcal{D} \models (x \wedge y = 1) \rightarrow (x \vee y = 1)$ in $\mathcal{BOO}\mathcal{L}$
 - ◇ In \mathcal{FT} , the projection of $x = f(y) \wedge y = f(z)$ on $\{x, z\}$ is $x = f(f(z))$
 - ◇ In \mathcal{WE} , $\mathcal{D} \models x.a.x = x.a \wedge y.b.y = y.b \rightarrow x = y$
- Prove the last assertion!

Properties of CLP Languages

- \mathcal{T} axiomatizes some of the properties of \mathcal{D}
- For a given Σ , let $(\mathcal{D}, \mathcal{L})$ be a constraint domain with signature Σ , and \mathcal{T} a Σ -theory.
- \mathcal{D} and \mathcal{T} correspond on \mathcal{L} if:
 - ◇ \mathcal{D} is a model of \mathcal{T} , and
 - ◇ for every constraint $c \in \mathcal{L}$, $\mathcal{D} \models \tilde{\exists}c$ iff $\mathcal{T} \models \tilde{\exists}c$.
- \mathcal{T} is *satisfaction complete* with respect to \mathcal{L} if for every constraint $c \in \mathcal{L}$, either $\mathcal{T} \models \tilde{\exists}c$ or $\mathcal{T} \models \neg\tilde{\exists}c$.
- $(\mathcal{D}, \mathcal{L})$ is *solution compact* if

$$\forall c \exists \{c_i\}_{i \in I} : \mathcal{D} \models \forall \tilde{x} \neg c(\tilde{x}) \longleftrightarrow \bigvee_{i \in I} c_i(\tilde{x})$$

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 $\text{CLP}(\mathcal{X})$
- We have to deal only with user predicates
- E.g.
 - ◇ $x \not\leq y$ in $\text{CLP}(\mathcal{R})$ is $x < y$
 - ◇ $x \neq y$ in $\text{CLP}(\mathcal{R})$ is $x < y \vee y < x$
 - ◇ \mathcal{R}_{Lin} with constraint $x \neq \pi$ is not s.c.
- How can we express $x \neq y$ in $\text{CLP}(\mathcal{FT})$?

Logical Semantics (I)

- Two common logical semantics exist.
- The first one interprets a rule

$$p(\tilde{x}) \leftarrow b_1, \dots, b_n$$

as the logic formula

$$\forall \tilde{x}, \tilde{y} \ p(\tilde{x}) \vee \neg b_1 \vee \dots \vee \neg b_n$$

Logical Semantics (II)

- The second one associates a logic formula to each predicate in Π
 - ◇ If the set of rules of P with p in the head is:

$$\begin{aligned} p(\tilde{x}) &\leftarrow B_1 \\ p(\tilde{x}) &\leftarrow B_2 \\ &\vdots \\ p(\tilde{x}) &\leftarrow B_n \end{aligned}$$

then the formula associated with p is:

$$\begin{aligned} \forall \tilde{x} p(\tilde{x}) \leftrightarrow & \exists \tilde{y}_1 B_1 \\ & \vee \exists \tilde{y}_2 B_2 \\ & \vdots \\ & \vee \exists \tilde{y}_n B_n \end{aligned}$$

- ◇ If p does not occur in the head of a rule of P , the formula is: $\forall \tilde{x} \neg p(\tilde{x})$
- ◇ 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 \mathcal{L}^* -formulas.
- A \mathcal{D} -interpretation of a formula is an interpretation of the formula with the same domain as \mathcal{D} and the same interpretation for the symbols in Σ as \mathcal{D} .
- It can be represented as a subset of $B_{\mathcal{D}}$ where

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

- A \mathcal{D} -model of a closed formula is a \mathcal{D} -interpretation which is a model of the formula.
- The usual logical semantics is based on the \mathcal{D} -models of P and the models of P^*, T .
- The least \mathcal{D} -model of a formula Q is denoted by $lm(Q, \mathcal{D})$.
- A *solution* to a query G is a valuation v such that $v(G) \subseteq lm(P, \mathcal{D})$.

Fixpoint Semantics

- Based on one-step consequence operator $T_P^{\mathcal{D}}$ (also called “immediate consequence operator”).
- Take as semantics $lfp(T_P^{\mathcal{D}})$, where:

$$T_P^{\mathcal{D}}(I) = \{p(\tilde{d}) \mid p(\tilde{x}) \leftarrow c, b_1, \dots, b_n \in P, a_i \in I, \\ \mathcal{D} \models v(c), v(\tilde{x}) = \tilde{d}, v(b_i) = a_i\}$$

- Theorems:
 1. $T_P^{\mathcal{D}} \uparrow \omega = lfp(T_P^{\mathcal{D}})$
 2. $lm(P, \mathcal{D}) = lfp(T_P^{\mathcal{D}})$

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 *consistent* and a function *infer*:
 - ◇ *consistent*(C) checks the consistency of a constraint store
 - ◇ Usually “*consistent*(C) iff $\mathcal{D} \models \exists c$ ”, but sometimes “if $\mathcal{D} \models \exists c$ then *consistent*(C)”
 - ◇ *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
 $\langle A \cup a, C, S \rangle \rightarrow_r fail$
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
 $\langle A, C, S \rangle \rightarrow_i \langle A, C', S' \rangle$ if $(C', S') = infer(C, S)$
 - ◇ In particular, may turn passive constraints into active ones
- Transition s : checks satisfiability
$$\langle A, C, S \rangle \rightarrow_s \begin{cases} \langle A, C, S \rangle & \text{if } consistent(C) \\ fail & \text{if } \neg consistent(C) \end{cases}$$

Top-Down Operational Semantics (III)

- Initial state: $\langle G, \emptyset, \emptyset \rangle$
- Derivation: $\langle A_1, C_1, S_1 \rangle \rightarrow \dots \rightarrow \langle A_i, C_i, S_i \rangle \rightarrow \dots$
- 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: $\exists_{-\tilde{x}} C \wedge S$, 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

Types of CLP(\mathcal{X}) Systems

- *Quick-checking* CLP(\mathcal{X}) system: its operational semantics can be described by $\rightarrow_{ris} \equiv \rightarrow_r \rightarrow_i \rightarrow_s$ and $\rightarrow_{cis} \equiv \rightarrow_c \rightarrow_i \rightarrow_s$
- 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
 - ◇ $infer(C, S) = (C \cup S, \emptyset)$
 - ◇ $consistent(C)$ 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:

$$SS(P) = \{p(\tilde{x}) \leftarrow c \mid \langle p(\tilde{x}), \emptyset, \emptyset \rangle \rightarrow^* \langle \emptyset, c', c'' \rangle, \mathcal{D} \models c \leftrightarrow \exists_{-\tilde{x}} c' \wedge c''\}$$

- 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. $[SS(P)]_{\mathcal{D}} = lm(P, \mathcal{D})$, where

$$[SS(P)]_{\mathcal{D}} = \{v(a) \mid (a \leftarrow c) \in SS(P), \mathcal{D} \models v(c)\}$$

2. $SS(P) = lfp(S_P^{\mathcal{D}})$
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, \dots, 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 $P^*, \mathcal{T} \models \neg G$.

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