Fondements de la Programmation: Constraint Logic Programming

Giovanni Coray - Vincenzo Pallotta

June 25, 2000

References:

- "Constraint Logic Programming: A survey" - J. Jaffar and M. Maher, The Journal of Logic Programming Vol. 19 & 20, pp. 503-582, May 1994.
- "Programming with constraints: An Introduction" - K. Mariot and P.J. Stuckey, MIT Press, 1998.

Extensions of Logic Programming

Pure L.P. has been extended towards several directions:

- functional programming
- object-oriented
- ...why not import constraint programming?

PROLOG has built-in predicate without a truly declarative semantics:

- equality on different domains (e.g. numbers) doesn't behave as a relation.

New built-in domains:

- PROLOG II adds Rational Trees as an extension of PROLOG terms.

Need of a more general representation of solutions:

- substitutions or solved equations is a quite limited way of presenting solutions
- implicit answers for representing sets of solution (even denumerable):
  - the constraint: $X < Y, Y \geq 3.14$ on $\Re$ is not represantable by a substitution!

Towards CLP: unification revisited

Classic LP is based on computing the MGU of terms (finding the solution of the equation $t_1 = t_2$).
Unification makes:

1. verifies the existence of solutions (essential)
2. iteratively simplifies the equation obtaining a solved form, hence explicitly representing sets of solutions (not essential, it is only a better presentation of solutions).

- The Herbrand Universe is the data domain for classic LP.
- Unification computes over Herbrand universe constructing sets of solvable equations (sets of constraints).
- We can replace the Herbrand Universe with other domains!
- CLP(?) is a schema with a domain parameter.
- LP is actually CLP(Herbrand Universe) which uses unification as an algorithm for solving equality constraints.

Constraint Logic Programming

We define a class of logic programming languages CLP(χ) where χ represents the constraint structure. This parameter is defined as:

$$\chi = \langle \Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T} \rangle$$

where

- $\mathcal{D}$ is a Σ-structure (i.e. the underlying domain)
- Σ is a signature, (built-in predicates and function symbols used to build constraint over Σ-structure $\mathcal{D}$.
- $\mathcal{L}$ is a class of Σ-formule (e.g the class of expressible constraints).
- $\mathcal{T}$ is a Σ-theory (e.g. an axiomatization of (non necessarly complete) of $\mathcal{D}$.

A *constraint domain* is the pair $(\mathcal{D}, \mathcal{L})$ (which will be denoted simply by $\mathcal{D}$). We assune that:

- terms and constraints in $\mathcal{L}$ are first order objects,
- the binary predicate symbol $=$ is always contained in Σ and it is interpreted as the identity among elements in $\mathcal{D}$,
- there are at least two special constraints in $\mathcal{L}$ which are respectively indentically true and false in $\mathcal{D}$,
- the class of constraints $\mathcal{L}$ is closed with respect to variable renaming, constraint conjunction and existential quantification.

Example: CLP($\Re$)

Linear Constraints over $\Re$

We define the parameter $\mathcal{R} = \langle \Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T} \rangle$ where:

- $\Sigma = \{0, 1, =, <, \leq, +, *\}$
- $\mathcal{D} = \Re$
- $\mathcal{L}$ is the constraints set generated by primitive constraints (e.g. by combining constraints like: $E_1 = E_2$ or $E_1 < E_2$ or $E_1 \leq E_2$ using logical connectives and where $E_1 E_2$ are expression built using the symbols $0, 1, +, *$. For instance the constraint: $\exists x : (3x + y = 2)$ is an abbreviation of $\exists x : (x + x + x + y = 1 + 1)$.
- $\mathcal{T}$ is an axiomatization capable of computing solution for linear constraint systems in $\mathcal{L}$, for instance the symplex algorithm.

The CLP language

Consider:

$$\chi = \langle \Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T} \rangle$$

$CLP(\chi)$ clauses are of the form:

$$H \leftarrow C_1, \ldots, C_n | B_1, \ldots, B_k$$

where:

- $C_1, \ldots, C_n$ are the constraints, that is atomic formulae in $\mathcal{L}$
- $B_1, \ldots, B_k$ are defined predicates.

A goal is of the form:

$$\leftarrow C_1, \ldots, C_n | B_1, \ldots, B_k$$

with $C_1 \wedge \ldots \wedge C_n$ satisfiable in $\mathcal{T}$.

CLP resolution

Consider the goal:

$$\leftarrow C_1, \ldots, C_n | G_1, \ldots, G_i, \ldots, G_k$$

and the CLP clause:

$$H^1 \leftarrow C_1^1, \ldots, C_{n_1}^1 | B_1, \ldots, B_{k_1}$$

we can rewrite the goal in the following way:

$$\begin{aligned} \leftarrow C_1, \ldots, C_n, (G_i = H^1), C_1^1, \ldots, C_{n_1}^1 | \\ G_1, \ldots, G_{i-1}, B_1, \ldots, B_{k_1}, G_{i+1}, \ldots, G_k \end{aligned}$$

with $C_1, \ldots, C_n, G_i = H^1, C_1^1, \ldots, C_{n_1}^1$ satsifiable in $\mathcal{T}$.

- Resolution without unification.
- In LP=CLP(Herbrand) unification is replaced by constraint satisfaction on Herbrand Universe with term equality.
- The whole state of computation is contained in the rewritten goal.

CLP derivations

A derivation is a finite sequence of goals $G_1, \ldots, G_k$. It successfully terminates if

$$G_k \equiv \leftarrow C_1, \ldots, C_n$$

and $C_1 \wedge \ldots \wedge C_n$ satisfiable in $\mathcal{T}$. Otherwise if finitely fails. $C_1 \wedge \ldots \wedge C_n$ is called the *answer constraint*.

Example in CLP($\Re$)

Program:

$$
\begin{aligned}
p(X,Y,Z) \quad &\leftarrow \\
& \quad Z = X * X1 + Y * Y1, \\
& \quad X1 \geq 0, \\
& \quad Y1 \geq 0 | \\
& \quad q(X,Y,X1,Y1). \\
q(X,Y,X,Y) \quad &\leftarrow \quad true.
\end{aligned}
$$

Goal:

$$\leftarrow p(0.2, 0.1, Z)$$

Derivation steps:

$$
\begin{aligned}
\leftarrow \quad & p(0.2,0.1,Z) = p(X',Y,',Z'), \\
& Z' = X' * X1' + Y' * Y1', \\
& X1' \geq 0, \\
& Y1' \geq 0 | \\
& q(X',Y',X1',Y1'). \\
\leftarrow \quad & p(0.2,0.1,Z) = p(X',Y,',Z'), \\
& Z' = X' * X1' + Y' * Y1', \\
& X1' \geq 0, Y1' \geq 0, \\
& q(X',Y',X1',Y1') = q(X'',Y'',X'',Y'').
\end{aligned}
$$

Constraint solvers in CLP($\Re$)

Program:

$$
\begin{aligned}
p(X,Y,Z) \quad &\leftarrow \\
& \quad Z = X * X1 + Y * Y1, \\
& \quad X1 \geq 0, \\
& \quad Y1 \geq 0 | \\
& \quad q(X,Y,X1,Y1). \\
q(X,Y,X,Y) \quad &\leftarrow
\end{aligned}
$$

Goal:

$$\leftarrow X = 0.2, Y = 0.1 | p(X,Y,Z)$$

A smart constraint solver may simplify the answer constraint and get the numerical solution:

$$\leftarrow Z = 0.05.$$

Otherwise only a part of the answer constraint can be explicitly solved, obtaining:

$$\leftarrow Z = X * X + Y * Y, X > Y.$$

It is possible to specify input constraints and check if they are satisfied by the goal:

$$
\begin{gathered}
\leftarrow X > Y, p(X,Y,Z). \\
\vdots \\
\leftarrow X > Y, Z = X * X + Y * Y.
\end{gathered}
$$

Another example (1)

The program:

$$
\begin{aligned}
sumto(N,S) \quad &\leftarrow \quad N = 0, S = 0. \\
sum(N,S) \quad &\leftarrow \quad N \geq 1, N \leq S, N1 = N - 1, S1 = S - N | \\
& \qquad sum(N1, S1).
\end{aligned}
$$

defines the relation:

$$Sumto = \{(N,S) \in N^2 | S = \sum_{i=1}^{N} i\}$$

The goal: $\leftarrow S \leq 3 | sumto(N,S)$ leads to 3 succesfull derivations:

$$(N = 0, S = 0), (N = 1, S = 1), (N = 2, S = 3).$$

The derivation:

$$
\begin{aligned}
\leftarrow \quad & S \leq 3 | sumto(N,S) \\
\leftarrow \quad & S \leq 3, N \geq 1, N1 = N - 1, S1 = S - N | sumto(N1, S1) \\
\leftarrow \quad & S \leq 3, N \geq 1, N \leq S, N1 = N - 1, S1 = S - N, \\
& N1 \geq 1, N1 \leq S1, N1' = N1 - 1, S1' = S1 - N1 | \\
& sumto(N1', S1').
\end{aligned}
$$

then we could chose to select the first clause obtaining:

$$
\begin{aligned}
\leftarrow \quad & S \leq 3, N \geq 1, N \leq S, N1 = N - 1, S1 = S - N, \\
& N1 \geq 1, N1 \leq S1, N1' = N1 - 1, S1' = S1 - N1, \\
& N1' = 0, S1' = 0.
\end{aligned}
$$

which semplifies to $N = 2, S = 3$.

Another example (2)

Otherwise, if we choose the second clause we obtain:

$$
\begin{aligned}
\leftarrow \quad & S \leq 3, N \geq 1, N \leq S, N1 = N - 1, S1 = S - N, \\
& N1 \geq 1, N1 \leq S1, N1' = N1 - 1, S1' = S1 - N1, \\
& N1' \geq 1, N1' \leq S1', N1'' = N1' - 1, S1'' = S1' - N1' | \\
& sumto(N1'', S1'').
\end{aligned}
$$

which is has an unsatisfiable constraint and thus the computation terminates.

- Constraints are used to specify both input and output.
- Constraints are dynamically generated.
- To avoid infinite computation it is better to check the consistency of resulting constraint at each step of derivation.

Logic Programming revisited

In the program:

$$p(f(X), X, Y, g(X,Y)) \leftarrow q(X,Y).$$
$$q(X, h(X)) \leftarrow .$$

unification can be made explicit by means of term equality obtaining:

$$\begin{aligned} p(X,X,Z,W) &\leftarrow X = f(X1), Y = X1, \\ &\quad W = g(X1,Z) | q(X,Z). \\ q(X,Y) &\leftarrow Y = h(X). \end{aligned}$$

Goals are transformed in the same way:

$$\begin{aligned} \leftarrow p(X,a,Z,W) \equiv\ & \leftarrow Y = a | p(X,Y,Z,W). \\ & \vdots \\ & \leftarrow Y = a, X = f(a), \\ & \quad Z = h(a), W = g(a, h(a)). \end{aligned}$$

- Constraint solver = unification algorithm
  - check solvability
  - incrementally
  - finds a normal form

Are constraints only "implementation"?

Can we reverse the problem and consider CLP as a special case of LP?
Let's consider the CLP program P_1:

$$\begin{aligned} r(X,Y,Z) &\leftarrow X + Y = Z. \\ p(X,Y,Z) &\leftarrow X + Y = Z + 1 | \\ &\quad r(X,Y,Z). \end{aligned}$$

If we consider constraints like $X + Z = Z$ defined by the logic program $+(X, Y, Z)$, we obtain an equivalent logic program P_2:

$$\begin{aligned} &+(0, N, N). \\ +(s(N), M, s(K)) &\leftarrow +(N, M, K). \\ r(X,Y,Z) &\leftarrow +(X,Y,Z). \\ p(X,Y,Z) &\leftarrow +(X,Y,s(Z)), \\ &\quad r(X,Y,Z). \end{aligned}$$

but the goals:

- $\leftarrow r(X,Y,Z)$.
  - returns a constraint answer in P_1
  - returns an infinite number of answers in P_2
- $\leftarrow p(X,Y,Z)$.
  - in P_1 the constraint $X + Y = Z + 1, X + Y = Z$ is recognized as unsatisfiable and the goal finitely fails
  - in P_2 the evaluation of $+(X,Y,s(Z)), +(X,Y,Z)$ doesn't terminate.

LP vs CLP

- In LP there exists no global consistency checking: atoms in congiuntive goals are processed separately.
- In pure LP (i.e. CLP($\mathcal{FT}$): Finite Trees=Herbrand terms+unification) global constraints are collected and represented within the MGU by the unification algorithm. Its application has a global effect.
- CLP inherits (and generalize to different domains) almost of the semantic features of LP.
  - existence of a canonical computational domain (Herbrand universe → Many sorted algebraic structure)
  - existence of a fix-point semantics
  - existence of a minimal model semantics
  - soundness and completeness for successful derivations (positive programs)
  - (soundness and completeness for negation as a finite failure).

Algebraic Semantics

Let's consider CLP(χ) where $\chi = \langle \Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T} \rangle$.

- The structure $\mathcal{D}$ is:
  - a collection of sets $\mathcal{D}_i$ (on for each sort)
  - an mapping from each function and predicate symbols to functions and relations over $\mathcal{D}_i$.
- *$\mathcal{D}$-base*: $\{p(t_1, ..., t_n) \leftarrow C_1, \ldots, C_k | \ p$ is a defined predicate symbol (not belonging to Σ) and $t_1, ..., t_n$ are elements in $\mathcal{D}$.
- A subset I of the $\mathcal{D}$-base is called *$\mathcal{D}$-interpretation*. The clause $H \leftarrow C_1, \ldots, C_n | B_1, \ldots, B_k$ is true in I if for each variables instantiation ϑ (substitution) which is a solution for $C_1, \ldots, C_n$ then $\{B_1\vartheta, \ldots, B_k\vartheta\} \subseteq I$ implies $H\vartheta \in I$.
- I is a $\mathcal{D}$-model of a program P if all its clauses are true in I ($I \models P$).
- The least $\mathcal{D}$-model of P is

$$lm(P, \mathcal{D}) = \bigcap_{\substack{I \subseteq \mathcal{D} - base \\ I \models P}} I$$

Clark completion

We need to transform the LP part of a CLP program in order to construct its logical semantics.

- For each predicate p in P we consider the set of clause defining p:

$$\begin{array}{c} p(\overline{X}) \leftarrow B_1 \\ \vdots \\ p(\overline{X}) \leftarrow B_n \end{array}$$

  then the associate formula is

$$\begin{array}{rcl} \forall \overline{X}(p(\overline{X}) & \leftrightarrow & \exists \overline{Y_1} B_1 \vee \\ & & \vdots \\ & & \vee \exists \overline{Y_n} B_n) \end{array}$$

  where $\overline{Y_i}$ is the set of variables in B_i except those in $\overline{X}$.

- If p doesn't occur in the head of a clause in P then the formula is

$$\forall \overline{X} \neg p(\overline{X})$$

- The obtained theory is called Clark completion and it is denoted by P^*.

Logical Semantics

For a given constraint structure $\chi = \langle \Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T} \rangle$ we can say that:

- the theory $\mathcal{T}$ *correspond to* $\mathcal{D}$ if
  - $\mathcal{D}$ is a model of $\mathcal{T}$ (all formulas of $\mathcal{T}$ evaluate to true under the interpretation provided by $\mathcal{D}$),
  - for each constraint $C \in \mathcal{L}$: $\mathcal{D} \models \exists C$ iff $\mathcal{T} \models \exists C$ (sound and completeness)

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

- The structure $\mathcal{D}$ plays the same role of Herbrand models with respect to generic models in LP ($\Rightarrow$ we can compute in $\mathcal{D}$).

- For a given CLP program P then $lm(P^*, \mathcal{D})$ is the intersection of all the models of the theory:

$$\{p \leftarrow C | P^*, \mathcal{T} \models p \leftarrow C\}$$

- In ordinary LP the corresponding theory of equality constraints on Herbrand terms is the Clark's Equality Theory.

Fix-point Semantics

Immediate consequence operator: Let P a constraint logic program in CLP(χ), with $\chi = \langle \Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T} \rangle$ and let I be a $\mathcal{D}$-interpretation, then:

$$\begin{array}{rcl} \mathcal{T}_P^{\mathcal{D}}(I) & = & \{p(\overline{s}) \leftarrow C' \mid \\ & & \quad p(\overline{X}) \leftarrow C | p_1(\overline{s_1}), \ldots, p_n(\overline{s_n}) \in P \wedge \\ & & \quad \forall i = 1, ..., n : (p_i(\overline{Y_1}) \leftarrow C_i \in I) \wedge \\ & & \quad \mathcal{D} \models C \Leftrightarrow (\overline{X} = \overline{s} \wedge C' \wedge \\ & & \quad \displaystyle\bigwedge_{i=1}^{n} (\overline{s_i} = \overline{Y_i} \wedge C_i))\} \end{array}$$

The least fix point of $\mathcal{T}_P^{\mathcal{D}}$ can be computed as:

$$lfp(\mathcal{T}_P^{\mathcal{D}}) = \mathcal{T}_P^{\mathcal{D}} \uparrow \omega = lim_{n \to \infty} \mathcal{T}_P^{\mathcal{D}}(\emptyset)^n$$

Computing the $lfp(\mathcal{T}_P^{\mathcal{D}})$ we obtain exactly $lm(P, \mathcal{D}) = lm(P^*, \mathcal{D})$.

Summary of CLP operations

1. **Consistency** (or satisfiability)

$$\mathcal{D} \models \exists C$$

2. **Implication** (or entailment)

$$\mathcal{D} \models C \Rightarrow \bigvee_{i=1}^{n} C_i$$

3. **Projection** onto variables $\overline{X}$:
   find simplest constraint C' such that

$$\mathcal{D} \models C' \Leftrightarrow \exists \overline{Y}(C)$$

   where $\overline{Y}$ are all variables in C except $\overline{X}$.

4. **Determination** (of a variable):
   detect that

$$\mathcal{D} \models \exists Z(\forall X, \overline{Y}(C(X, \overline{Y}) \Rightarrow X = Z$$

   that is, X is *determined* (or *grounded*) by C.

Further properties

Definition. *A constraint domain* $\chi = \langle \Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T} \rangle$ *is* solution compact *iff*

1. for all constraints $D \in \mathcal{L}$ there exist constraints $\{C_i\}_{i \in \mathcal{I}} \subseteq \mathcal{L}$ such that
$$\mathcal{D} \models \forall \overline{X}(D \Leftrightarrow \bigwedge_{i \in \mathcal{I}} C_i(\overline{X})$$

2. for all constraints $C \in \mathcal{L}$ there exist constraints $\{C_i\}_{i \in \mathcal{I}} \subseteq \mathcal{L}$ such that
$$\mathcal{D} \models \forall \overline{X}(\neg C(\overline{X}) \Leftrightarrow \bigvee_{i \in \mathcal{I}} C_i(\overline{X})$$

Example: Let $\Re_{lin}^{\neq \pi}$ be $\Re_{lin}$ with the addition of a unary predicate $\neq \pi$ with the usual interpretation. $\Re_{lin}^{\neq \pi}$ is not solution compact since the negation of the constraint $X \neq \pi$ is $X = \pi$ which cannot be expressed as a disjunction of constraints in $\Re_{lin}^{\neq \pi}$.

Definition. *A constraint domain* $\chi = \langle \Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T} \rangle$ *has the Indipendence of Negated Constraints property iff for all constraints* $C, C_1, \ldots, C_n \in \mathcal{L}$*,*

$$\mathcal{D} \models \exists (C \wedge \neg C_1 \wedge \cdots \wedge \neg C_n)$$

iff for all $i = 1, \ldots, n$

$$\mathcal{D} \models \exists (C \wedge \neg C_i)$$

Operational Semantics

It is an extension of the SLD-refutation procedure and can be viewed as a transition system over states represented by triples $\langle A, C, S \rangle$, where

- A is a multi-set of atoms and constraints (atoms to be considered)
- C, S are multi-sets of constraints (C are *active* and S are *passive* constraints).
- $fail$ is a special failure state which correspond to $\langle A, C, false \rangle$.

The system is defined by the following labelled rules:

$$\frac{A' = A \setminus \{a\} \wedge a \in A \wedge \exists h : (h \leftarrow B \in P \wedge (a = h))}{\langle A, C, S \rangle \rightarrow_r \langle A' \cup B, C, S \cup \{(a = h)\} \rangle}$$

$$\frac{a \in A \wedge \neg \exists (h \leftarrow B) \in P : (a = h)}{\langle A, C, S \rangle \rightarrow_r fail}$$

$$\frac{a \in A \wedge c \in \mathcal{L}}{\langle A, C, S \rangle \rightarrow_c \langle A \setminus \{a\}, C, S \cup \{a\} \rangle}$$

$$\frac{(C', S') = infer(C, S)}{\langle A, C, S \rangle \rightarrow_i \langle A, C', S' \rangle}$$

$$\frac{consistent(C)}{\langle A,C,S \rangle \rightarrow_s \langle A,C,S \rangle} \qquad \frac{\neg consistent(C)}{\langle A,C,S \rangle_s \rightarrow fail}$$

Operational Semantics

- In the first three rules the atom a is selected from A using an arbitrary *selection rule*.
- A *computation rule* in a given state choose the appropriate transition rule and applies a selection rule when necessary.
- The external function $infer(C, S)$ computes a new pair of active and passive constrain sets such that no information will be lost or added
$$\mathcal{D} \models (C \wedge S) \Leftrightarrow (C' \wedge S')$$
At this step we assume also a simplification of constraints which crucially depends on the chosen algorithm.
- The predicate $consistent(C)$ performs a consistency chech of the constraints set defined by
$$consistent(C) \textbf{ iff } \mathcal{D} \models \exists \overline{X} : C$$
with $\overline{X} \in FV(C)$.

Success Set: The Succes Set for a CLP program is defined by:

$$SS(P) = \{p(\overline{X}) \leftarrow c \mid \left\langle p(\overline{X}), true, true \right\rangle \rightarrow^* \left\langle true, c', c'' \right\rangle \wedge \mathcal{D} \models (c \Leftrightarrow \exists \overline{X} : c' \wedge c'')\}$$

Derivations and Computed Answer Constraints

- An *initial state* is a state of the form $\langle A, true, true \rangle$.
- A *successful final state* is a state which cannot be further reduced and it has the form $\langle true, C, S \rangle$ with $S \neq false$.
- A *failed final* state is a state of the form $\langle A, C, false \rangle$.
- A *successful derivation* for a goal A is a finite sequence of states starting from the initial state $\langle A, true, true \rangle$ obtained by the application of the above transition rules and ending in a successful final state $\langle true, C, S \rangle$. $\exists \overline{Y}(C \wedge S)$ is called the *computed answer constraint* for the goal A ($\overline{Y}$ are variables in C and S but not in A).
- A derivation is *failed* if is finite and it ends in a failed final state.
- A derivivation $\{\langle A_i, C_i, S_i \rangle\}_{i \in \mathcal{I}}$ is *fair* if for every $i \in \mathcal{I}$ and $a \in A_i$, a is eventually subject to the application of one of the first three rules.
- A computation rule is fair if it gives rise solely to fair derivations.

CLP systems

Definition. *A CLP systems consists of:*

1. a constraint domain χ
2. a computation rule
3. definition of $infer$ and $consistent$

Let consider the following macro-transitions:

- $\rightarrow_{ris} = \rightarrow_r \rightarrow_i \rightarrow_s$
- $\rightarrow_{cis} = \rightarrow_c \rightarrow_i \rightarrow_s$

Definition. *A CLP system is*

- *quick checking* if uses only the macro-transitions $\rightarrow_{ris}$ and $\rightarrow_{cis}$.
- *progressive* if for every state $\Delta = \langle A, C, S \rangle$ with non-empty A, every transition sequence starting with Δ eventually either:
  - fails,
  - contains a $\rightarrow_r$ transition, or
  - contains a $\rightarrow_c$ transition.

Definition. *A CLP system is* ideal *if:*

- it is quick-checking
- it is progressive
- $infer(C, S) = (C', true)$
- $consistent$ is complete

All major CLP systems are quick-checking and progressive, but most are not ideal.

Soundness

Consider a CLP program P, a goal Q, and a constraint domain $\chi = \langle \Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T} \rangle$ where $\mathcal{D}$ correspond to $\mathcal{T}$.

Theorem. *(Soundness)*

1. $SS(P) = lfp(T_P^{\mathcal{D}})$
2. If Q has a successful derivation with a computed answer constraint C, then $P^*, \mathcal{T} \models Q \Leftarrow C$.
3. If $\mathcal{T}$ is satisfaction complete, then
   (a) if Q finitely fails, then $P^*, \mathcal{T} \models \neg Q$
   (b) if Q has finite number of successful derivations with computer answer constraints $C_1, \ldots, C_n$ then $C_1, \ldots, C_n$

$$P^*, \mathcal{T} \models Q \Leftrightarrow \bigvee_{i=1}^{n} C_i$$

Completeness

Consider a CLP program P, a goal Q, and a constraint domain $\chi = \langle \Sigma, \mathcal{D}, \mathcal{L}, \mathcal{T} \rangle$ where $\mathcal{D}$ correspond to $\mathcal{T}$. Suppose that the underlying CLP system is progressive.

Theorem. *(Completeness)*

1. If $P^*, \mathcal{T} \models Q \Leftarrow C$ then there exist a finite number of successful derivation for Q with computed answer constraints $C_1, \ldots, C_n$ such that
$$\mathcal{T} \models C \Rightarrow \bigvee_{i=1}^{n} C_i$$
2. Suppose that Indipendence of Negated Constraints holds. If $P^*, \mathcal{T} \models Q \Leftarrow C$, then there exists a succesful derivation for Q with computed answer constraint C' such that
$$\mathcal{T} \models C \Rightarrow C'$$
3. Suppose $\mathcal{T}$ is satisfaction complete. If $P^*, \mathcal{T} \models Q \Leftrightarrow \bigvee_{i=1}^{n} C_i$ then there exist a finite number of successful derivation for Q with computed answer constraints $C'_1, \ldots, C'_m$ such that
$$\mathcal{T} \models \bigvee_{i=1}^{n} C_i \Leftrightarrow \bigvee_{i=1}^{m} C'_i$$
4. Suppose the underlying CLP system is ideal. If $P^*, \mathcal{T} \models \neg Q$ then Q finitely fails.