6. Constraint Logic Programming

Goal: Extend Prolog by Constraints (Formulas over some theory) which have to be handled by dedicated constraint solvers.

6.1: Syntax + Semantics of CLP (Theory)
6.2: Integration of CLP in Prolog (Practice)

6.1. Syntax + Semantics of CLP

Constraints are essentially atomic formulas over a certain subsignature. To combine them with CP, we use a special treatment for the pred. symbols =, true, fail.

Def 6.1.1 (Constraints)

Let $(\Sigma, \Delta)$ be a signature with true, fail $\in \Delta_0$ and $=$ $\in \Delta_2$. Let $\Sigma' \subseteq \Sigma$, $\Delta' \subseteq \Delta$, where $\Delta'$ does not contain true, fail, or $\Rightarrow$. Then $(\Sigma, \Delta, \Sigma', \Delta')$ is a Constraint Signature. The atomic formulas from $\text{At}(\Sigma', \Delta', \mathcal{V})$, $\text{At}(\Sigma, \{=\}, \mathcal{V})$, $\text{true}$, $\text{false}$] are called Constraints.

Constraint: true, fail, $t_1 = t_2$ (with arbitrary $t_1, t_2$) $p(t_1, \ldots, t_n)$ with $p \in \Delta'$ and all fact. symbols in $t_1, \ldots, t_n$ are from $\Sigma'$.

Ex. 6.1.2 We consider a constraint signature for
Ex. 6.1.2 We consider a constraint signature for integer arithmetic: \((\Sigma, \Delta, \Sigma', \Delta')\) with

\[\Sigma_0^1 = \mathbb{Z}\]

\[\Sigma_n^1 = \{-, \, \text{abs}\}\]

\[\Sigma_2^1 = \{+, -, \times, \div, \mod, \min, \max\}\]

\[\Delta_2 = \{ #>=, #=<, #>, #<, #=, #\leq\}\]

\[\text{to distinguish it from the built-in Prolog predicate } \geq\]

These sets \(\Sigma^1\) and \(\Delta^1\) are also called \(\Sigma_{FD}\) and \(\Delta_{FD}\)

where "FD" stands for "Finite Domains".

Examples for Constraints:

- \text{true}
- \text{fail}
- \text{X+Y} \ #> \ 2 \ # 3
- \text{max}(X,Y) \ #= \ X \ \text{mod} \ 2
- \text{f}(X) + 2 \ = \ Y + 2
  \[\in \Sigma \backslash \Sigma_{FD}\]

Intuition: We assume that we have a black box constraint solver which can determine whether a constraint is true. To define when a constraint is "true", we need a constraint theory \(CT\). A constraint \(\psi\) is true iff \(CT \models \psi\).

\textbf{Def 6.13 (Constraint Theory)}

Let \((\Sigma, \Delta, \Sigma', \Delta')\) be a constraint signature
Let \((\Sigma, \Delta, \Sigma', \Delta')\) be a constraint signature. If \(CT \subseteq \mathcal{F}(\Sigma', \Delta', \mathcal{U})\) is satisfiable and only contains closed formulas, then \(CT\) is called a constraint theory.

**Ex 6.14** Let \(S_{FD} = (\mathbb{Z}, \alpha)\) be the structure with the carrier \(\mathbb{Z}\) and the "intuitive" meaning \(\alpha\) of all fun. and pred. symbols from \(\Sigma_{FD}\) and \(\Delta_{FD}\):

\[
\begin{align*}
\alpha_n &= n \text{ for all } n \in \mathbb{Z} \\
\alpha_+ &= \text{addition function on integers} \\
\alpha_{\#} &= \succ \text{ on integers}
\end{align*}
\]

The intuitive constraint theory \(CT_{FD}\) for the signature of Ex. 6.1.2 is the set of all closed formulas \(\Phi \in \mathcal{F}(\Sigma_{FD}, \Delta_{FD}, \mathcal{U})\) where:

\[
\Phi \in CT_{FD} \iff S_{FD} \models \Phi.
\]

By Gödel's incompleteness theorem, \(CT_{FD}\) is not even semi-decidable.

For integer arithmetic (with + and \#), there can't be any sound and complete terminating automatic constraint solver. \((\Rightarrow \text{Sect. 6.2})\)

**Syntax of CLPs:**

- \texttt{true, false} are one-ordered
Syntax of CLP:
- `true`, `fail`, `=` are pre-defined
- pred. symbols from $\Delta'$ can only occur in bodies of clauses
- pred. symbols from $\Delta'$ can only be applied to
  fact. symbols from $\Sigma'$

**Def 6.15** (Syntax of CLP)
A non-empty finite set $\mathcal{P}$ of definite Horn clauses
over a constraint signature $(\Sigma, \Delta, \Sigma', \Delta')$ is a
CLP, if $\{true\} \in \mathcal{P}$, $\{X = X\} \in \mathcal{P}$, and for
all other clauses $\{B, \ldots, C_n\} \in \mathcal{P}$ we have:
- If $B = p(t_1, \ldots, t_m)$, then $p \in \Delta' \cup \{true, fail, =\}$
- If $C_i = p(t_1, \ldots, t_m)$ and $p \in \Delta'$, then
tj $\in \Sigma'(\Sigma', \nu)$ for all $1 \leq j \leq m$.

**Ex 6.16**: We regard the signature $(\Sigma, \Delta, \Sigma_F, \Delta_F D)$. Instead of Roby's predicates "\>" and "\is", we now use predicates from the constraint sus-signature $\Delta_F D$
(more efficient, bidirectional ...).

\[
\text{fact}(0, 1),
\]

\[
\text{fact}(X, Y) \iff X \# > 0, X1 \# = X - 1, \text{fact}(X1, Y1),
\]

\[
\text{fact}(X, Y) \iff \# = X \times Y.
\]

Similar to LP, we now define the declarative and

\[
\text{LP 2017 Seite 4}
\]
Similar to LP, we now define the declarative and procedural semantics of CLP.

**Declarative semantics:** In addition to the prog. clauses of $S$, one now also uses the constraint theory $CT$ as additional axioms.

**Def 6.1.7 (Declarative Semantics of CLP)**

Let $S$ be a CLP, let $CT$ be the corresponding constraint theory, let $G = \{ \neg A_1, \ldots, \neg A_n \}$. Then the declarative semantics of $S$ and $CT$ w.r.t. $G$ is

\[
\mathcal{D}[S, CT, G] = \{ \sigma(A_1, \ldots, A_n) \mid SuCT \models \sigma(A_1, \ldots, A_n), \sigma \text{ is a ground substitution} \}
\]

**Ex. 6.1.8** Let $S$ be the CLP for fact, consider $CT_{FD}$ and $G = \{ \neg \text{fact}(1, 2) \}$.

The only ground subst. $\sigma$ with $SuCT_{FD} \models \sigma(\text{fact}(1, 2))$ is $\sigma(2) = 1$. Thus: $\mathcal{D}[S, CT_{FD}, G] = \{ \text{fact}(1, 1) \}$.

For $G' = \{ \neg \text{fact}(X, 1) \}$ we have $SuCT_{FD} \models \sigma_n(6')$ and $SuCT_{FD} \models \sigma_2(6')$ for $\sigma_n(X) = 0$ and $\sigma_2(X) = 1$.

Thus: $\mathcal{D}[S, CT_{FD}, G'] = \{ \text{fact}(0, 1), \text{fact}(1, 1) \}$.

Clearly, LP is a special case of CLP.

**Corollary 6.1.9**
Corollary 6.1.9

Let \( S \) be a CLP with \( \Sigma' = \emptyset \) and \( \Delta' = \emptyset \). Then for all queries \( G \), we have

\[
\text{\( [\Omega, \emptyset, \text{\( G \)}}) = \text{\( [\Omega, \emptyset, \text{\( G \)}}) .
\]

To explain how CLPs are evaluated, we now define their procedural semantics. Problem: in addition to \( S \), we also have the constraint theory \( CT \). \( CT \) cannot be handled by SLD-resolution, but by the black-box constraint solver.

How can we combine the handling of \( S \) (by SLD-resolution) and of \( CT \) (by constraint solver)?

Solution: use a uniform representation where

unification is also represented by a constraint.

Such a representation could also be used for ordinary CLPs.

Ex. 6.1.10 add-program

procedural semantics as before VS.

procedural semantics with unification constraints:

Instead of performing the unification between
Instead of performing the unification between $A_i$ and $B$, we only collect the constraint $A_i = B$.

\[ p(X, 0) = p(s(0), Y) \]

is $X = s(0) \land O = Y \implies A_i = B$ should be the conjunction of equalities needed to make $A_i$ and $B$ equal.

We start with the constraint true and in the example we end with a conjunction of constraints that is equivalent to

\[ X = s(0) \land X' = s(0) \land Y = 0 \land Z = s(0) \land U = s(s(0)) \]

If the resulting conjunction of constraints is not satisfiable, then one of the corresponding unification steps is not possible.

Advantage: Can easily be combined with CT

Def 6.1.11 (Equality between Atoms)

Let $A$, $B$ be atomic formulas. Then we define the formula $A = B$ as follows:

- $A = B$ is the formula fail if $A = p(\ldots)$, $B = q(\ldots)$, $p \neq q$.
- $A = B$ is the formula true if $A = B = p \in \Delta$.
\( A \equiv B \) is the formula \( s_1 = t_1 \land \ldots \land s_n = t_n \)

if \( A = \rho(s_1, \ldots, s_n) \) and \( B = \rho(t_1, \ldots, t_n) \)

The conjunction of constraints \( CO \) that is computed can be simplified into equivalent constraints:

\[ \forall X \; X = X, \text{true} \vdash A \left( CO \iff \text{simplify}(CO) \right) \]

These 2 axioms describe unification: \( s = t \) is only entailed by these axioms if \( s \) and \( t \) are syntactically equal.

For any quantifier-free formula \( \varphi \) with variables \( X_1, \ldots, X_n \), let \( \forall \varphi \) denote the universal closure of \( \varphi \):

\[ \forall X_1, \ldots, X_n \; \varphi, \]

(Similarly, \( \exists \varphi \) is the existential closure \( \exists X_1, \ldots, X_n \; \varphi \).

So we now perform computations on configurations of the form \( (G, CO) \) where \( CO \) is a conjunction of constraints.

One should only perform a computation step

\( (G_1, CO_1) \vdash (G_2, CO_2) \) if the new resulting conjunction \( CO_2 \) is still satisfiable (under the axioms for unifiability), i.e., if \( \forall X \; X = X, \text{true} \vdash \exists \varphi \).
This ensures that one does not perform comp. steps where unification would fail.

Now we extend the procedural semantics with unification constraints to the proc. semantics of CCL. Now CO can contain both unification constraints (built with \( =, \text{true}, \text{fail} \)) and constraints built with predicates from \( \Delta \).

Thus, to check satisfiability of \( CO \), we now have to check \( CT \cup \{ \forall X \; X = X, \text{true} \} \vdash \exists Z \; CO \).

\[ \text{\textasciitilde we assume that this can be checked by a black-box constraint solver} \]

Similarly, one can replace \( CO \) by \( \text{Simplify} \; (CO) \) provided that

\[ CT \cup \{ \forall X \; X = X, \text{true} \} \vdash A (CO \iff \text{Simplify} \; (CO)) \]

\[ \text{\textasciitilde we assume that the constr. solver can perform such simplifications} \]

**Def 6.1.12 (Procedural Semantics of CCL)**

Let \( S \) be a CCL and let \( CT \) be a corresponding constraint theory.

- A configuration is a pair \( (G, CO) \), where \( G \) is a query or \( \Box \) and \( CO \) is a conjunction of constraints.
• There is a computation step \((G_1, C_0) \xrightarrow{\sigma} (G_2, C_0)\)
  iff \(\ldots\) see slide 43

• A computation of \(\Sigma\) for the query \(G\) is a finite or infinite sequence:
  \((G, \text{true}) \xrightarrow{\sigma} (G_1, C_0) \xrightarrow{\sigma} (G_2, C_0) \xrightarrow{\sigma} \ldots\)

• A computation that ends in \((\square, C_0)\) is successful. The computed answer constraints are simplify \(C_0\), where \(C_0 \cup \{\forall X \pi \exists Y \phi\} \vdash A(C_0 \rightarrow \text{simplify}(C_0))\)

The procedural semantics of \(\Sigma\) w.r.t. \(G\) is defined as
  \(\ldots\) see slide 43
  Here: \(\sigma\) are all ground substitutions where the answer constraints hold.

\[\text{Ex 6.1.13} \quad \text{fact}(0, 1).\]

\[\text{fact}(X, Y) :\neg X \# > 0, \ X \# = X - 1, \ \text{fact}(X, Y), \ Y \# = X \# Y.\]

We regard the query \(G = \{\neg \text{fact}(1, 2)\}\), To ease readability, we omit “\(\neg\)”.

\((\text{fact}(1, 2), \text{true})\)

\(\xrightarrow{\sigma} (X \# > 0, \ldots, Y \# = X \# Y, \text{true} \land \text{fact}(1, 2) = \text{fact}(X, Y))\)
see slide 44 (we applied simplify after each step)

Thus: \( P \equiv P, \text{CT}_0, G \equiv \{ \text{fact}(1,1) \} \). 

**Thm 6.1.14** (Equivalence of Procedural and Declarative Semantics)

Let \( P \) be a CLP, let CT be a corresp. constraint theory, let \( G \) be a query. Then:

\[ P \equiv P, \text{CT}, G \equiv P \equiv P, \text{CT}, G \]

To solve the indeterminismus in the computation, we proceed as for ordinary LP.

**Indet. 2:** restrict ourselves to canonical computations (always choose the leftmost literal of the query)

**Indet. 1:** build up SLD-tree by depth-first search (i.e., treat prog. clauses from top to bottom).

**Ex 6.1.15** SLD-Trees for LP and CLP
SLD-Tree for ordinary CLP for fact:

?- fact(X, Y).

\[ \text{fact}(X, Y) \]
\[
\{X/1, Y/2\} \]
\[
\{Y/1\} \]

\[ X > 0, X1 \text{ is } X - 1, \text{ fact}(X1, Y1) \]
\[ Y \text{ is } X \times Y1 \]

Prog. error, because \( Y \) may only be evaluated if both arguments are fully instantiated.

CLP for fact:

\[ \text{fact}(0, Y) \]
\[ \text{fact}(X, Y) \leftarrow X > 0, X1 \neq X - 1, \text{ fact}(X1, Y1) \]
\[ Y \neq X \times Y1 \]

?- fact(X, Y).

For SLD-trees in CLP, label the edges by the conjunction of constraints \( CO \), where we simplify \( CO \) after each step. If for \( CO \) we have

\[ CT \cup \{ \forall X \; X = X, \text{true} \} \nmid 3 \quad CO \]

then we do not construct the corresponding node in the tree.
Now both solutions $X=0$ and $X=1$ can be found. Afterwards, we end in non-termination.

Problem: Literal $Y \neq X \lor Y \neq 0$ is at the end of the query. If one adds $Y \neq X \lor Y \neq 0$ to CO, it would become unsatisfiable:

\[
Y \neq X \lor Y \neq 0 \\
\uparrow \quad \uparrow \\
Y > 1 \\
\text{unsatisfiable}
\]

Solution: Change the order of literals in our program:

\[
\text{fact } (0, 1).
\text{fact } (X, Y) : - X \neq 0, X \neq X - 1, Y \neq X \lor Y \neq 0, \\
\text{fact } (X \neq, Y \neq).
\]

Now the SLD-tree is finite.