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 LP, we use a special treatment for the pred. symbols $=$, true, fail.

Def 6.1.1 (Constraints) Slide 40

Let (Σ, Δ) be a signature with $true, fail \in \Delta_0$ and $= \in \Delta_2$. Let $\Sigma' \subseteq \Sigma$, $\Delta' \subseteq \Delta$, where Δ' does not contain true, fail, or $=$. Then $(\Sigma, \Delta, \Sigma', \Delta')$ is a Constraint Signature. The atomic formulas from $At(\Sigma', \Delta', \mathcal{V}) \cup At(\Sigma, \{=\}, \mathcal{V}) \cup \{true, fail\}$ 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 fct. symbols in $t_1, \ldots, t_n$ are from Σ'.

Ex. 6.1.2 We consider a constraint signature for Slide 40

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

Slide 40

$\Sigma_0' = \mathbb{Z}$

$\Sigma_1' = \{-, abs\}$

$\Sigma_2' = \{+, -, *, /, mod, min, max\}$

$\Delta_2' = \{\#>=, \#=<, \#>, \#<, \#=, \#\backslash=\}$

↑ to distinguish it from the built-in Prolog predicate >=

These sets Σ' and Δ' are also called Σ_{FD} and Δ_{FD}, where "FD" stands for "Finite Domains".

Examples for Constraints:

true

fail

$X + Y \ \#> \ Z * 3$

$max(X, Y) \ \#= \ X \ mod \ 2$

$f(X) + 2 = Y + Z$

↑ $\in \Sigma \setminus \Sigma_{FD}$

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 φ is true iff $CT \models \varphi$.

Def 6.1.3 (Constraint Theory)

Let $(\Sigma, \Delta, \Sigma', \Delta')$ be a constraint signature

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

Ex 6.1.4 Let $S_{FD} = (\mathbb{Z}, \alpha)$ be the structure with the carrier $\mathbb{Z}$ and the "intuitive" meaning α of all fct. and pred. symbols from Σ_{FD} and Δ_{FD}:

$\alpha_n = n$ for all $n \in \mathbb{Z}$

$\alpha_+ =$ addition function on integers

$\alpha_{\#>} =$ ">" on integers

⋮

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

$\varphi \in CT_{FD}$ iff $S_{FD} \models \varphi$.

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. (⇒ Sect. 6.2)

Syntax of CLPs:

- true, fail = are pre-defined

Syntax of CLP:

- true, fail, = are pre-defined
- pred. symbols from Δ' can only occur in bodies of clauses
- pred. symbols from Δ' can only be applied to fct. symbols from Σ'

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 $\{B, \neg C_1, \ldots, \neg C_n\} \in \mathcal{P}$ 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 $t_j \in \mathcal{T}(\Sigma', \mathcal{V})$ for all $1 \leq j \leq m$.

Ex 6.16: We regard the signature $(\Sigma, \Delta, \Sigma_{FD}, \Delta_{FD})$. Instead of Prolog's predicates ">" and "is" we now use predicates from the constraint sub-signature Δ_{FD} (more efficient, bidirectional, ...).

Slide 41

```
fact(0,1).
fact(X,Y) :- X #> 0, X1 #= X-1, fact(X1, Y1),
             Y #= X * Y1.
```

Similar to LP, we now define the declarative and procedural semantics of CLP

Similar to LP, we now define the declarative and procedural semantics of CLP.

Declarative semantics: In addition to the prog. clauses of $\mathcal{P}$, one now also uses the constraint theory CT as additional axioms.

Def 6.1.7 (Declarative Semantics of CLP)

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

$$\mathbb{D}[\![\mathcal{P}, CT, G]\!] = \{\sigma(A_1 \wedge \dots \wedge A_k) \mid \mathcal{P} \cup CT \models \sigma(A_1 \wedge \dots \wedge A_k), \sigma \text{ is a ground substitution}\}$$

Ex. 6.1.8 Let $\mathcal{P}$ be the CLP for fact, consider CT_{FD} and $G = \{\neg fact(1, Z)\}$.

The only ground subst. σ with $\mathcal{P} \cup CT_{FD} \models \sigma(fact(1,Z))$ is $\sigma(Z) = 1$. Thus: $\mathbb{D}[\![\mathcal{P}, CT_{FD}, G]\!] = \{fact(1,1)\}$.

For $G' = \{\neg fact(X, 1)\}$ we have $\mathcal{P} \cup CT_{FD} \models \sigma_1(G')$ and $\mathcal{P} \cup CT_{FD} \models \sigma_2(G')$ for $\sigma_1(X) = 0$ and $\sigma_2(X) = 1$.

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

Clearly, LP is a special case of CLP.

Corollary 6.1.9

Corollary 6.1.9

Let $\mathcal{P}$ be a CLP with $\Sigma' = \emptyset$ and $\Delta' = \emptyset$.
Then for all queries G, we have
$\mathcal{D}[\![\mathcal{P}, \emptyset, G]\!] = \mathcal{D}[\![\mathcal{P}, G]\!]$.

To explain how CLPs are evaluated, we now define their procedural semantics. Problem: in addition to $\mathcal{P}$, 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 $\mathcal{P}$ (by SLD-resol.) 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 LPs.

Ex. 6.1.10 add-program Slide 42

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 $\overline{A_i = B}$

e.g. $p(X,0) = p(s(0),Y)$

is $X = s(0) \wedge 0 = Y$ $\rightarrow$

Since A_i and B are atomic formulas (and no terms), "=" cannot be applied to them. $A_i = B$ 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

$$X = s(0) \wedge X' = s(0) \wedge Y = 0 \wedge Z = s(0) \wedge U = s(s(0))$$

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 $\overline{A = B}$ as follows:

- $\overline{A = B}$ is the formula fail if $A = p(\ldots)$, $B = q(\ldots)$, $p \neq q$.
- $\overline{A = B}$ is the formula true if $A = B = p \in \Delta_0$.
- $\overline{A = B}$ ...

- $A = B$ is the formula $s_1 = t_1 \wedge \ldots \wedge s_n = t_n$ if $A = p(s_1, \ldots, s_n)$ and $B = p(t_1, \ldots, t_n)$

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

$$\{\forall X\ X = X, \text{true}\} \models \forall (CO \leftrightarrow \text{simplify}(CO))$$

↑ Those 2 axioms describe unification: $s = t$ is only entailed by these axioms if s and t are syntactically equal

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

$$\forall X_1, \ldots, X_n\ \varphi.$$

(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 $\{\forall X\ X = X, \text{true}\} \models \exists\, CO_2$

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 CLP. Now CO can contain both unification constraints (built with $=$, true, fail) and constraints built with predicates from Δ'.

Thus, to check satisfiability of CO, we now have to check $CT \cup \{\forall X\ X=X, \text{true}\} \models \exists\, CO$.

↖ We assume that this can be checked by a black-box constraint solver

Similarly, one can replace CO by Simplify(CO) provided that

$$CT \cup \{\forall X\ X=X, \text{true}\} \models \forall (CO \leftrightarrow \text{simplify}(CO))$$

↖ We assume that the constr. solver can perform such simplifications

Def 6.1.12 (Procedural Semantics of CLP)

Let S be a CLP and let CT be a corresponding constraint theory.

- A configuration is a pair (G, CO), where G is a query or $\square$ and CO is a conjunction of constraints.

- There is a computation step $(G_1, CO_1) \vdash_S (G_2, CO_2)$ iff ... see slide 43
- A computation of S for the query G is a finite or infinite sequence:
  $(G, true) \vdash_S (G_1, CO_1) \vdash_S (G_2, CO_2) \vdash_S \ldots$
- A computation that ends in $(\square, CO)$ is successful. The computed answer constraints are $simplify(CO)$, where $CT \cup \{\forall X\ X=X, true\} \models \forall (CO \leftrightarrow simplify(CO))$

The procedural semantics of S w.r.t. G is defined as ... see slide 43

Here: σ are all ground substitutions where the answer constraints hold.

Ex 6.1.13

```
fact(0,1).
fact(X,Y) :- X #> 0, X1 #= X-1,
             fact(X1,Y1), Y #= X*Y1.
```

We regard the query $G = \{\neg fact(1,Z)\}$.
To ease readability, we omit "$\neg$".

$(fact(1,Z), true)$

$\vdash_S (X \#>0, \ldots, Y \#= X*Y1, true \wedge \overline{fact(1,Z) = fact(X,Y)})$

:

See slide 44 (we applied simplify after each step)

Thus: $\llbracket P, \mathcal{S}, CT_{FD}, G \rrbracket = \{fact(1,1)\}$.

Thm 6.1.14 (Equivalence of Procedural and Declarative Semantics)

Let $\mathcal{S}$ be a CLP, let CT be a corresp. Constraint theory, let G be a query. Then:

$$\llbracket D, \mathcal{S}, CT, G \rrbracket = \llbracket P, \mathcal{S}, CT, G \rrbracket.$$

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

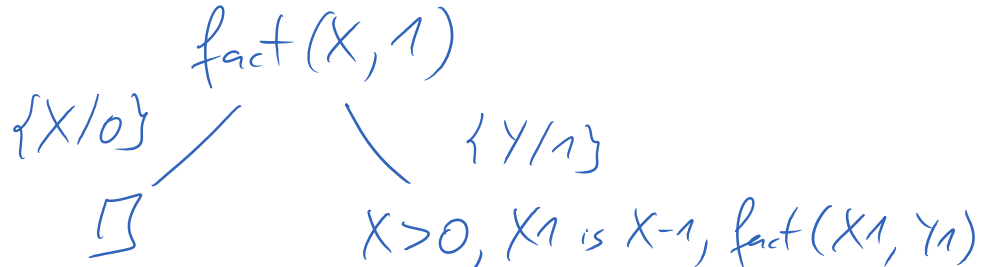
Indet. 2: restrict ourselves to canonical computations (always choose the leftmost literal of the query)

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 LP for fact.

?- fact(X,1).



Prog. error, because > may only be evaluated if both arguments are fully instantiated

CLP for fact:

fact(0,1).

fact(X,Y) :- X #> 0, X1 #= X-1, fact(X1,Y1), Y #= X * Y1.

?- fact(X,1).

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

$CT \cup \{\forall X\ X=X, true\} \not\models \exists\ CO$,

then we do not construct the corresponding node in the tree.

Slide 45

Now both solutions $X=0$ and $X=1$ can be found. Afterwards, we end in non-termination.

Problem: Literal $Y \#= X * Y1$ is at the end of the query. If one adds $Y \#= X * Y1$ to CO, it would become unsatisfiable:

$$Y \#= X * Y1 \wedge Y = 1 \wedge X1 \#= X - 1 \wedge X1 \#> 0$$

$\uparrow$ $\uparrow$

1 >1

unsatisfiable

Solution: Change the order of literals in our program:

```
fact(0,1).
fact(X,Y) :- X #>0, X1 #= X-1, Y #= X*Y1,
             fact(X1,Y1).
```

Now the SLD-tree is finite.

Slide 46