6.1 Syntax and Semantics of Constraint Logic Programs

Freitag, 26. Juni 2015 10:00

Goal: extend logic programming by constraints
$\Rightarrow$ for the signature (Σ, Δ) introduce a sub-signature $\Sigma' \subseteq \Sigma$, $\Delta' \subseteq \Delta$ for constraints.

Def. 6.1.1 (Constraint-Signature, Constraints)

see slide.

Constraints:
- atomic formulas over sub-signature (Σ', Δ'),
- $s = t$, where s and t are arbitrary terms,
- true, fail

the special predicates in Δ' may only be applied to the special fct. symbols in Σ'

$=$ can be applied to all fct. symbols

Ex 6.1.2. Constraint-Signature for integer numbers.

predicates $\#>=$ etc. are different from $>= \in \Delta_2$

$\#>= \in \Delta_2' \subseteq \Delta_2$.

This Constraint-Signature is pre-defined in Prolog and called FD (finite domain).

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

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

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

$f \notin \Sigma'$

Idea: There should be a constraint solver to handle constraints which has to be combined with the ordinary mechanism to evaluate logic programs.

To determine whether a constraint is true, one needs a constraint theory CT.

Def. 6.13 (Constraint Theory)

Let $(\Sigma, \Delta, \Sigma', \Delta')$ be a constraint signature. CT is <u>constraint theory</u> iff $CT \subseteq \mathcal{F}(\Sigma', \Delta', \mathcal{V})$ is satisfiable and only contains closed formulas.

no free variables, e.g. $\forall X \;\; X + 0 \;\#\!= X$

Idea: We assume that we have a constraint solver to decide $\varphi \in CT$ for all closed formulas $\varphi \in \mathcal{F}(\Sigma', \Delta', \mathcal{V})$.

Ex 6.14 For FD, CT_{FD} should contain all true closed formulas over integers.
(CT_{FD} is not decidable, not even semi-decidable.
→ see Sect. 6.2)

Def 6.15 (Syntax of LP with Constraints)

A non-empty finite set $\mathcal{P}$ of definite Horn clauses over a constraint signature $(\Sigma, \Delta, \Sigma', \Delta')$ is a <u>logic program with constraints</u> iff $\{true\} \in \mathcal{P}$, $\{X = X\} \in \mathcal{P}$, and for all constraints iff $\{true\} \in \mathcal{P}$, $\{X = X\} \in \mathcal{P}$, and for all other clauses $\{B, \neg C_1, \dots, \neg C_n\} \in \mathcal{P}$ we have:

(a) if $B = p(t_1, \dots, t_m)$, then $p \notin \Delta' \cup \{true, fail, =\}$

(b) if $C_i = p(t_1, \dots, t_m)$ and $p \in \Delta'$, then
$t_1, \dots, t_m \in \mathcal{T}(\Sigma', \mathcal{V})$.

Condition (b) also has to hold for all queries $\{\neg C_1, \dots, \neg C_n\}$.

Ex 6.16 factorial as a CLP

Semantics of CLP: declarative + procedural semantics

Declarative Semantics: entailment from

- clauses of the program $\mathcal{P}$
- constraint theory CT

Def 6.17 (Declarative Semantics of CLP)
Let $\mathcal{P}$ be a LP with constraints, let CT be the corresponding constraint theory. Let $G = \{\neg A_1, \dots, \neg A_k\}$ be a query. Then the declarative semantics of $\mathcal{P}$ and CT wrt G is defined as:

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

Ex 6.18 $\mathcal{P}$ from Ex 6.16.

$G = \{\neg fact(1, 7)\}$

$G' = \{\neg fact(X, 1)\}$

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

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

?- fact(X,1).

X = 0 ;

X = 1

?- fac(X,1).

X = 0 ;

prog. error

Main advantages of CLP:

- efficiency
- bi-directionality

<u>Corollary 6.19</u> Let $\Sigma' = \emptyset$, $\Delta' = \emptyset$.

Then $D[\![\mathcal{P}, \emptyset, G]\!] = D[\![\mathcal{P}, G]\!]$.

(i.e.: CLP is a proper extension of CP).

Now we have to define the procedural semantics, i.e., how to evaluate CLP.

Pure LP: binary SLD-resolution with prog. clauses of $\mathcal{P}$

Problem: CT can contain arbitrary formulas (not just definite Horn clauses). Constraint solver should be used to handle CT.

Idea: also represent the SLD-resolution steps as "constraints" (to have a uniform representation of

(evaluation steps with prog. clauses and with constraints)

these constraints are unification problems of the form:
"does the goal unify with the head of a clause?"

Ex 6.1.10. Illustrate how SLD-resolution steps can be represented as constraints.

add-program

Query: ?- add(s(0), s(0), U).

Idea: Do not perform the required unifications directly, but only collect the unification problems that have to be solved.

Configurations now have the form $(G, \underbrace{CO}_{\text{conjunction of unification problems } A = B})$

Start with initial configuration $(G, true)$.

In each step, check whether CO remains satisfiable (otherwise, one can't perform the desired resolution step).

Final configuration of successful computation:

$(\Box, CO)$

Now CO can be simplified to obtain the answer subst.:

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

In pure LP, "=" can only be applied to terms, not to formulas. Therefore, if A and B are atomic formulas, we write $\overline{A=B}$ as an abbreviation for a corresponding conjunction of equalities ~~between~~ terms:

Def 6.1.11 Let A, B be atomic formulas. Then we define the formula $\overline{A=B}$ as follows:

- $\overline{A=B}$ is fail, if $A = p(\ldots)$, $B = q(\ldots)$, $p \neq q$.
- $\overline{A=B}$ is true, if $A = p$, $B = p$
- $\overline{A=B}$ is the formula $s_1 = t_1 \wedge s_2 = t_2 \wedge \ldots \wedge s_n = t_n$ if $A = p(s_1, \ldots, s_n)$, $B = p(t_1, \ldots, t_n)$

Ex 6.1.12 add-example using definition of $\overline{A=B}$

A configuration (G_1, CO_1) should only be evaluated to (G_2, CO_2) if CO_2 is still satisfiable (under the axioms for $=$ and true).

Thus, we check:

$$\{\forall X\ X=X,\ true\} \models \underbrace{\exists\ CO_2}_{\text{existential closure of } CO_2}$$

existential closure of CO_2, i.e., all variables of CO_2 are existentially quantified

This variant of the procedural semantics of LP can easily be extended in order to handle constraints

- Now one can add both unification constraints (with =) and constraints built with Δ' (e.g. $X \mathbin{\#>} 0$)
- When checking satisfiability of constraints, one also has to regard CT: (G_1, CO_1) can be evaluated to (G_2, CO_2) only if:

  $$\{\forall X = X, true\} \cup CT \models \exists\, CO_2$$

  ← this needs the constraint solver

- After each evaluation step, one can simplify the constraints (here one has to take CT into account again)

Def 6.1.14 (Procedural Semantics of CLP)

Let $\mathcal{P}$ be a CLP and CT be the corresponding constraint theory.

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

Computation step: $(G_1, CO_1) \vdash_{\mathcal{P}} (G_2, CO_2)$

See slide

$P[\![\mathcal{P}, CT, G]\!]$: Here, the atoms of G are instantiated by all those ground subst. σ where $\sigma(CO)$ is true.

Ex 6.1.15 Procedural semantics of fact.

Here: „$\neg$" omitted in the queries

Procedural semantics of fact.

Here: · "$\neg$" omitted in the queries
· Simplified constraints after each eval. step } for readability

A <u>computation</u> for query G is a (finite or infinite) sequence of configurations:

$$(G, true) \vdash_{\mathcal{P}} (G_1, CO_1) \vdash_{\mathcal{P}} (G_2, CO_2) \vdash \dots$$

A computation is successful iff it ends in

$$(\square, CO).$$

The <u>answer constraints</u> are $simplify(CO)$

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

Simplification can also be used after each computation step.

<u>Thm 6.1.16</u> (Equivalence of declarative + procedural semantics for CLP)

Let $\mathcal{P}$ be a CLP and let CT be the corresp. constraint theory. Let G be a query.

Then: $D[\![\mathcal{P}, CT, G]\!] = P[\![\mathcal{P}, CT, G]\!]$.

CLP has the same indeterminisms as LP, and they are resolved in the same way:

Indet. 1: Which prog. clause is used for the next step?
$\Rightarrow$ top to bottom

Indet 2: Which literal of the goal is used for the next step?
$\Rightarrow$ left to right

$\Rightarrow$ Construct SLD trees by depth-first search from left to right.

Ex 6.1.17 $\quad$?- fac(X, 1).

fac(X,1)

{X/0}

$\Box$ $\qquad$ X>0, X1 is X-1, fac(X_1, Y_1), 1 is X * Y_1

1. Answer Subst: X = 0 ;

Then: prog. error, because X is not instantiated in X>0.

$\Rightarrow$ fac is not bidirectional

?- fact(X, 1)

Instead of labeling edges by unifiers,
we now label them by the constraints:

if $(G_1, CO_1) \vdash_\beta (G_2, CO_2)$,

then this results in the edge:

$$\begin{array}{c} G_1 \\ \big| \; CO_2 \leftarrow \text{or } simplify(CO_2) \\ G_2 \end{array}$$

CLP is bidirectional:

?- fact(X,1)

finds both solutions for X (but runs into non-termination afterwards).
If one exchanged the last 2 literals in the recursive fact-rule, it would terminate.