Learning from Entailment in First-Order Setting

B4M36SMU

In the last tutorial, we focused on relational lgg and rlgg. In this tutorial, we will have a look at a first-order logic version of the former (as the latter follows trivially).

LGG

Almost everything stays the same as in the relational lgg. The only new things that FOL adds are functions, i.e. functions symbols from which one can construct compound terms. Therefore, we have to define lgg such that it can handle even compound terms.

Constructing lgg of two terms: The basic step to compute lgg of two clauses is to compute firstly lgg of two terms t_1 and t_2.

- The simplest rule comes with resolving two constants: if these two constants are equal, then return the constant.
- The second simplest rule comes with resolving two variables or one constant and a variable, e.g. $t_1 = Adam$ and $t_2 = x$. In this situation, lgg returns a new variable for the tuple, in this case $(Adam, x)$.
- **(!)** The third rule to compute lgg of two terms is applicable when at least one of them is a compound term. If there is only one compound term, e.g. $t_1 = fatherOf(Adam)$ and $t_2 = x$, or function symbols of the terms differ, e.g. $t_1 = fatherOf(Adam)$ and $t_2 = motherOf(Adam)$, lgg returns a new variable for this tuple (similarly to the basic rule). Finally, if both terms have the same function symbol, e.g. $t_1 = fatherOf(Adam)$ and $t_2 = fatherOf(x)$; then lgg returns a compound term with the same function symbol applied to lgg-ed arguments of these compound terms, e.g. $fatherOf(LGG(Adam, x))$ which equals to $father(x_{Adam,x})$.
- However, it is important to note that each pair of terms (t_1, t_2) is anti-unified with exactly one variable, where t_1 is a term from l_1 and t_2 is a term from l_2.

We can sum up these rules in the following pseudocode[1]:

$$\begin{aligned}
LGG(A,t) &= \begin{cases} A & A = t \\ x_{A,t} & otherwise \end{cases} \\
LGG(x,y) &= x_{x,y} \\
LGG(f(\dots),x) &= x_{f(\dots),x} \\
LGG(f(t_1,\dots),f'(t'_1,\dots)) &= \begin{cases} x_{f(t_1,\dots),f'(t'_1,\dots)} & f \neq f' \\ f(LGG(t_1,t'_1),\dots) & otherwise \end{cases}
\end{aligned}$$

Where A is a constant, t is an arbitrary term, x and y are variables, and f/n is a function symbol of arity n. New variables are denoted either by tuples they represent, e.g. $x_{A,t}$.

[1]Recall that lgg is a symmetric operator, thus these rules behave similarly for symmetric inputs.

Constructing lgg of two clauses: Stays the same as in the previous tutorial, i.e.:

$$LGG(\gamma_1, \gamma_2) = \bigvee_{\substack{l_1 \in \gamma_1, l_2 \in \gamma_2 \\ compatibleLiterals(l_1, l_2)}} LGG(l_1, l_2)$$

$$LGG(p(t_1, t_2, \dots), p(t'_1, t'_2, \dots)) = p(LGG(t_1, t'_1), LGG(t_2, t'_2), \dots)$$

Where p is a predicate symbol, l is a literal and both γ_1 and γ_2 are clauses. The *compatibleLiterals* is a function which returns true iff both of its arguments have the same negation sign and the same predicate. Once again, recall from the lecture that lgg is a symmetric, commutative, and associative operator.

Exercise

- Does this equation $lgg(p(x, f(A), A), p(f(A), x, A)) = p(x', y', z')$ hold? If not, why?
- Compute unification and anti-unification (lgg) of $\gamma_1 = p(A, x, f(x))$ and $\gamma_2 = p(y, f(B), f(f(B)))$, [1]. Write down γ_1, γ_2 and the results obtained to visualize the lattice induced by θ-subsumption.
- Find two clauses (universally quantified disjunctions) γ_1 and γ_2 for which $\gamma_1 \models \gamma_2$ and $\gamma_1 \not\subseteq_\theta \gamma_2$ hold.

Solution:

- No. Although the resulting clause θ-subsumes the input ones, it is not the *least* – the correct result is $p(x, y, A)$.
- $lgg(\gamma_1, \gamma_2) = p(x, y, f(y))$, unification of γ_1 and γ_2 is $p(A, f(B), f(f(B)))$.

$$\begin{tikzcd}
 & p(x_{A,y}, x_{x,f(B)}, f(x_{x,f(B)})) \arrow[dl, dash, "{\theta_1 = \{x_{A,y} \mapsto A, x_{x,f(B)} \mapsto x\}}"'] \arrow[dr, dash, "{\theta_2 = \{x_{A,y} \mapsto y, x_{x,f(B)} \mapsto f(B)\}}"] & \\
p(A, x, f(x)) \arrow[dr, dash, "{\theta_3 = \{x \mapsto f(B)\}}"'] & & p(y, f(B), f(f(B))) \arrow[dl, dash, "{\theta_4 = \{y \mapsto A\}}"] \\
 & p(A, f(B), f(f(B))) &
\end{tikzcd}$$

Lgg of γ_1 and γ_2 is at the top of the lattice. Corresponding substitutions lie on edges to show how a clause higher in the lattice can θ-subsume a lower clause. At the bottom, the unification of γ_1 and γ_2 lies; see, the most general unifier (mgu) is $\theta_{mgu} = \{x \mapsto B, y \mapsto A\}$.

new variable	γ_1 (θ_1)	γ_2 (θ_2)
$x_{A,y}$	A	y
$x_{x,f(B)}$	x	$f(B)$

- We need self-resolving clauses, e.g. $\gamma_1 = n(x) \implies n(s(x))$ and $\gamma_2 = n(x) \implies n(s(s(x)))$.

References

[1] Shan-Hwei Nienhuys-Cheng and Ronald De Wolf. *Foundations of inductive logic programming.* Vol. 1228. Springer Science & Business Media, 1997.