Modification of SLDNF-resolution for Built-in Predicates

Semyon Nigiyan
Yerevan State University
Yerevan, Armenia
e-mail: nigiyan@ysu.am

Lusine Sargsyan
Yerevan State University
Yerevan, Armenia
e-mail: lusinas@rambler.ru

ABSTRACT
In this paper the general logic programs with built-in predicates (logic programs with built-in predicates which use negation), general logic goals with built-in predicates (logic queries with built-in predicates which use negation) are regarded. Modification of *SLDNF*-resolution for built-in predicates is introduced. The soundness of modified *SLDNF*-resolution is proved.

Keywords
Logic, program, negation, built-in predicates, completion, *SLDNF*-resolution, soundness.

Consider three nonintersecting sets Φ, Π and X. Φ is a set of functional symbols each possessing an arity. For any $n\geq 0$, Φ contains a countable number of symbols of arity n. X is a countable set of variables. Terms are composed of elements of sets Φ and X.

1. Each 0-ary symbol of Φ is a term.
2. Each variable of X is a term.
3. If $t_1,\dots,t_n$ $(n>0)$ are terms and f is n-ary symbol of Φ, then $f(t_1,\dots,t_n)$ is a term.
4. No other terms exist.

The set of all terms with no variables is denoted by M. The set M is called the Herbrand universe.

$\Pi=\Pi_1\cup\Pi_2$, where Π_1 is the set of predicate symbols and for any $n\geq 0$, Π_1 contains a countable number of symbols of arity n, Π_2 is the set of built-in predicate symbols (built-in predicates), each k-ary $(k>0)$ built-in predicate is a calculable mapping $M^k\rightarrow\{true, false\}$. The atoms are defined as usual:

1. Each 0-ary symbol of Π is an atom.
2. If $t_1,\dots,t_n$ $(n>0)$ are terms and p is n-ary symbol of Π, then $p(t_1,\dots,t_n)$ is an atom.
3. No other atoms exist.

A formula of the first-order predicate logic over logical operations $\neg$, $\&$, $\vee$, $\supset$, $\sim$ and quantifiers $\exists$ and $\forall$ is defined conventionally [1]. A predicate term is an atom, which uses predicate symbol from Π_1. A literal is a predicate term or the negation of a predicate term. A ground literal is a literal not containing variables. A condition is an atom or the negation of an atom, which use predicate symbol from Π_2. By $Var(L)$ we denote the set of all variables involving in L, where L is a literal, a condition, or a term.

The substitution σ is a set $\{t_1/x_1,\dots,t_n/x_n\}$, where t_i is a term, x_i is a variable, $t_i\neq x_i$, $i\neq j \Rightarrow x_i\neq x_j$, $i,j=1,\dots,n$, $n\geq 0$. The following notations are introduced: $Arg(\sigma)=\{x_1,\dots,x_n\}$, $Var(\sigma)=Var(t_1)\cup\dots\cup Var(t_n)$. The composition of substitutions is defined traditionally.

Describe the studied interpretations (Herbrand interpretations). The object set of interpretations is the set M. The functional symbols are interpreted in the following way: with each 0-ary symbol of Φ we associate that symbol itself. With each n-ary $(n>0)$ symbol $f\in\Phi$ we associate the mapping $M^n\rightarrow M$ that maps n-tuple $(t_1,\dots,t_n)\in M^n$ to a term $f(t_1,\dots,t_n)$. With each 0-ary symbol of Π_1 we associate one of the elements of the set $\{true, false\}$, and with each n-ary $(n>0)$ symbol of Π_1 we associate some mapping $M^n\rightarrow\{true, false\}$. Denote the described set of interpretations by H. Note that interpretations from H may be different only in mappings corresponding to symbols from Π_1.

Let F be a closed formula and I be an interpretation from H. The value of a formula F on the interpretation I is defined in the natural way and denoted by $Val_I(F)$. The formula F is termed identically true if F takes the value *true* on any interpretation from H. If F and F' are closed formulas and the formula $F\supset F'$ is identically true, we will say that F' is a logical consequence of F and denote this fact by $F|=F'$.

A general logic program with built-in predicates (or, simple program) P is a sequence $S_1,\dots,S_n$ of the clauses, $n>0$. A clause $S\in\{S_1,\dots,S_n\}$ has the form A:-$L_1,\dots,L_m$, where A is a predicate term, each $L\in\{L_1,\dots,L_m\}$ is a literal or a condition, $m\geq 0$. Atom A is called the head of the clause S, the sequence $L_1,\dots,L_m$ is called the body of S, and number m is termed the length of the body of S. If $m=0$, S is termed a fact; if $m>0$, S is termed a rule. With the program P we associate the formula $comp(P)$:

$$F(p_1)\&\dots\&F(p_u),$$

where $p_1,\dots,p_u$ are the predicate symbols from program P, $p_i\in\Pi_1$, $i=1,\dots,u$, $u\geq 1$, and every $F(p)$, where $p\in\{p_1,\dots,p_u\}$, is defined in the following way:

If p is a 0-ary predicate symbol and p is a fact of program P, then $F(p)$ is p, else if p does not appear in the head of any clause of program P, then $F(p)$ is $\neg p$, else if definition of p is: p:-$B_1,\dots,p$:-B_v, where B_i is the body of the clause p:-B_i, $i=1,\dots,v$, $v\geq 1$, then $F(p)$ is:

$$p\sim E_1\vee\dots\vee E_v,$$

where E_i is $\exists y_1\dots\exists y_d(L_1\&\dots\&L_m)$, $y_1,\dots,y_d$ $(d\geq 0)$ are the variables of the rule p:-B_i, and B_i is $L_1,\dots,L_m$, $m\geq 1$, $i=1,\dots,v$.

If p is an n-ary $(n>0)$ predicate symbol and p does not appear in the head of any clause of program P, then $F(p)$ is $\forall x_1\dots\forall x_n\neg p(x_1,\dots,x_n)$, else if definition of p is: A_1:-$B_1,\dots,A_v$:-B_v, where B_i is the body of the clause A_i:-B_i, $i=1,\dots,v$, $v\geq 1$, then $F(p)$ is:

$$\forall x_1\dots\forall x_n\,(p(x_1,\dots,x_n)\sim E_1\vee\dots\vee E_v),$$

where $x_1,\dots,x_n$ are variables not appearing in the clauses A_1:-$B_1,\dots,A_v$:-B_v, and each E_i has a form $\exists y_1\dots\exists y_d((x_1=t_1)\&\dots\&(x_n=t_n)\&L_1\&\dots\&L_m)$, $y_1,\dots,y_d$ $(d\geq 0)$ are the variables of the rule A_i:-B_i, A_i is $p(t_1,\dots,t_n)$ and B_i is $L_1,\dots,L_m$, $m\geq 0$, $i=1,\dots,v$.

A general goal with built-in predicates (or, simple goal) Q has the form ?- $L_1,\dots,L_k$, where L_i is a literal or a condition, $i=1,\dots,k$, $k\geq 0$; number k is termed the length of the goal Q. If $k=0$, Q is termed an empty goal. The nonempty goal Q is identified with the formula:

$$\exists y_1\dots\exists y_s(L_1\&\dots\&L_k),$$

where $y_1,\dots,y_s$ are the variables involved in the $L_1,\dots,L_k$, $k\geq 1$, $s\geq 0$. The set $\{y_1,\dots,y_s\}$ we denote by $Var(Q)$.

The definition of general logic program and general goal without built-in predicates you can find in [1].

Computation rule R for general logic programs and general goals with built-in predicates based on modified *SLD*-resolution (see [2]) and is defined using functions Sel_R and Sub_R. (The definition of computation rule for general logic programs and general goals without built-in predicates you can find in [1]).

Let Q be a goal $?\text{-}\ L_1,\ldots,L_k,\ k\geq 1,\ Sel_R(Q)\in\{1,\ldots,k\}$. Let $Sel_R(Q)=j\ (1\leq j\leq k)$. If L_j is a literal $Sub_R(Q)$ is undefined. Let L_j be a condition. Then $Sub_R(Q)$ is a set of substitutions and for any $\sigma\in Sub_R(Q)$ the following conditions are satisfied:

a) $Arg(\sigma)\subset Var(L_j)$,

b) $Var(\sigma)\cap(Var(Q)\backslash Var(L_j))=\varnothing$,

c) $Val(L_j\sigma\gamma)=true$ for any substitution γ such that $Var(L_j\sigma\gamma)=\varnothing$.

And for any substitution δ such that $Var(L_j\delta)=\varnothing$ and $Val(L_j\delta)=true$, there exist $\sigma\in Sub_R(Q)$ and γ such that $L_j\delta=L_j\sigma\gamma$.

We specify an appropriate class of computation rules. A computation rule R (for *SLDNF*-resolution with built-in predicates) is safe if the following conditions are satisfied:

1. R only selects negative literals which are ground.
2. Having selected a ground negative literal $\neg A$ in some goal, R attempts to finish the construction of a finitely failed *SLDNF*-tree with root $?\text{-}A$ before continuing with the remainder of the computation.

Let P be a program, Q be a nonempty goal and R be a safe computation rule. An *SLDNF*-derivation $Q_1,Q_2,\ldots$ of (P,Q) via R, where $Q_1=Q$, is defined as follows:

Suppose Q_i is $?\text{-}L_1,\ldots,L_k\ (k\geq 1)$ and R selects the positive literal $L_j\ (1\leq j\leq k)$. Suppose $A\text{:-}K_1,\ldots,K_m\ (m\geq 0)$ is the input clause and L_j and A have most general unifier (mgu) σ. The derived goal Q_{i+1} is $?\text{-}L_1\sigma,\ldots,L_{j-1}\sigma,K_1\sigma,\ldots,K_m\sigma,L_{j+1}\sigma,\ldots,L_k\sigma$.

Suppose Q_i is $?\text{-}L_1,\ldots,L_k\ (k\geq 1)$ and R selects the ground negative literal L_j, where L_j is $\neg A$, $(1\leq j\leq k)$. An attempt is made to construct an *SLDNF*-tree with $?\text{-}A$ at the root. If the goal $?\text{-}A$ succeeds, then the subgoal $\neg A$ fails and so the goal Q_i also fails. If A fails finitely, then the subgoal $\neg A$ succeeds and the derived goal Q_{i+1} is $?\text{-}L_1,\ldots,L_{j-1},L_{j+1},\ldots,L_k$.

Suppose Q_i is $?\text{-}L_1,\ldots,L_k\ (k\geq 1)$ and R selects the condition $L_j\ (1\leq j\leq k)$. Let $Sub_R(Q_i)\neq\varnothing$ and $\delta\in Sub_R(Q_i)$, the derived goal Q_{i+1} is $?\text{-}L_1\delta,\ldots,L_{j-1}\delta,L_{j+1}\delta,\ldots,L_k\delta$.

Let P be a program, Q be a nonempty goal and R be a safe computation rule. Then the *SLDNF*-tree for (P,Q) via R is defined as follows:

1. Each node of the tree is a goal.
2. The root node is Q.
3. Let $?\text{-}L_1,\ldots,L_k\ (k\geq 1)$ is a node of the tree and suppose that the literal selected by R is the positive literal $L_j\ (1\leq j\leq k)$. Then this node has a descendent for each input clause $A\text{:-}K_1,\ldots,K_m\ (m\geq 0)$, such that L_j and A are unifiable. The descendent is $?\text{-}L_1\sigma,\ldots,L_{j-1}\sigma,K_1\sigma,\ldots,K_m\sigma,L_{j+1}\sigma,\ldots,L_k\sigma$, where $\sigma=mgu(L_j,A)$.
4. Let $?\text{-}L_1,\ldots,L_k\ (k\geq 1)$ is a node of the tree and suppose that the literal selected by R is the ground negative literal $L_j\ (1\leq j\leq k)$. If the subgoal L_j is successful, then the single descendent of the node is $?\text{-}L_1,\ldots,L_{j-1},L_{j+1},\ldots,L_k$.
If the subgoal L_j fails, then the node has no descendents.
5. Let $?\text{-}L_1,\ldots,L_k\ (k\geq 1)$ is a node of the tree and suppose that R selects the condition $L_j\ (1\leq j\leq k)$. If $Sub_R(Q_i)\neq\varnothing$ then this node has a descendent for each $\delta\in Sub_R(Q_i)$. The descendent is $?\text{-}L_1\delta,\ldots,L_{j-1}\delta,L_{j+1}\delta,\ldots,L_k\delta$.
If $Sub_R(Q_i)=\varnothing$, then this node has no descendents.
6. Nodes which are the empty goal have no descendents.

Let P be a program, Q be a nonempty goal and R be a safe computation rule. A finitely failed *SLDNF-tree* for (P,Q) via R is one which is finite and contains no branches, which end in the empty goal.

Theorem 1. Let P be a program, Q be a nonempty goal and R be a safe computation rule. Then, if (P,Q) has a finitely failed *SLDNF-tree* via R, then $comp(P)\models\neg Q$.

Theorem 2. Let P be a program, Q be a goal $?\text{-}L_1,\ldots,L_k\ (k\geq 1)$ and R be a safe computation rule. Then, if (P,Q) has an *SLDNF*-derivation, which ends in the empty goal, and $\sigma_1,\ldots,\sigma_s\ (s>0)$ is the sequence of substitutions using in this derivation, then $comp(P)\models\forall y_1\ldots\forall y_m((L_1\&\ldots\&L_k)\sigma_1\ldots\sigma_s)$, where $y_1,\ldots,y_m$ are variables appearing in $(L_1\&\ldots\&L_k)\sigma_1\ldots\sigma_s$.

REFERENCES

[1] J.W. Lloyd, “Foundations of Logic Programming”, *Berlin: Springer-Verlag*, 1984.

[2] S.A. Nigiyan, “Horn Programming with Built-in Predicates”, *Programming and Computer Software*, Vol.22, N1, pp. 19-25, 1996.