Logic Programs with Access Modifiers

ADRIAN GIURCA, DOREL SAVULEA

University of Craiova,
Faculty of Mathematics and Computer Science
giurca@inf.ucv.ro, savulea@central.ucv.ro

Abstract. Many actual research concerning distributed architectures of logic programs are oriented to techniques of sharing clauses. All the work is mainly for minimizing the computational cost of distribution (see, for example [1], [2] and [8]). In this paper we propose an architecture of logic programs using access modifiers (ALP) as in object oriented programming. We define the notion of logic program with access modifiers and we give an algorithm to obtain a such programs. Finally we propose a structure of an ALP and a general architecture of a such programs.


1 Preliminaries

This paper is devoted to investigate the possibility for the treatment of logic programs using the methodology of object oriented programming. The main reasons to do that are: A programmer write a logic program for a dedicated goal i.e. the logic program will be frequently queried with some dominant predicates – the predicates that solve the main goal of the problem. Following this criteria, it is a natural way to fragment the logic program using the principle of dominant queries. That permit to study strategies for parallel processing (see for example [4], [9] or [7]) and for fragment distribution of logic programs (see [1], [2] and [8]).

To obtain a such fragmentation we use OOP access modifiers on predicates of the logic program.

In OOP, access modifiers (see [3], [5], [6]) allow some parts of a class to be public (that is, usable by the rest of the world) and other parts to be private implementation details (hidden to the rest of the world). This helps provide encapsulation, or the ability to hide details of implementation away from other classes that should not know the inner workings.

The access modifiers which we will use in program fragmentation are:

– Public Access. Many programmers have habit of making everything public in their applications. This point of view is acceptable for test applications but when we are developing real life applications you should expose only the data or functionality that is necessary by the user of our class. Classes, attributes and methods marked with public access modifier are available in the same class, all the classes from the same project and to all other projects as well. This access specifier is least restrictive. Restricting access
to classes and/or members is not only a good programming practice but is also necessary to avoid any accidental misuse of our classes.

- **Private Access.** Unlike public access modifier, the private access modifier is applicable to the attributes/methods from the class. It restricts access to the attributes and/or methods within the class itself.

- **Protected Access.** Using private access modifier permit us to hide attributes and/or methods from others but what we do if some one is inheriting from our class? In many cases we want that methods of base class should be available in derived class. This cannot be achieved with private modifier. The protected access modifier provide a such behavior. The members marked with protected access modifier can be accessed in the same class and all the classes inherited from it but they are not available to other classes.

In the following section of the paper, which is the main section, we propose a construction of logic programs with access modifiers.

## 2 Constructing Logic Programs with Access Modifiers

Usually, the public part of a logic program is a set of clauses which is constructed following the most queried predicates from the program. This part is necessary to be public because the user can use it to resolve his goals. The other predicates are used by the program himself to solve the user goal. The corresponding part of the logic program can be divided into two sections: a protected part which is a set of clauses concerning predicates that must use the public part of the program and a private part that do not use the public part of the program. Bellow we present a such construction.

Let $P$ be a Horn logic program. We make the following notations:

- $PRED(P)$ the set of all predicate symbols from $P$
- $PRED(P)_{\text{public}}$ a nonempty set of all public predicates from $P$.

**Definition 1.** Let $p \in PRED(P)$. The set of clauses from $P$ defined by $p$ defined by

$$
def(p) = \{p(t_1, \ldots, t_{n_p}) \leftarrow B_1, \ldots, B_{m_p} | p(t_1, \ldots, t_{n_p}) \leftarrow B_1, \ldots, B_{m_p} \in P\}$$

is named the definition of $p$.

**Example 1.** Let $P$ be the following Horn program:

$$P: \begin{cases}
join(X,Y) \leftarrow table(X,Z), table(Z,Y), X \neq Y. \\
property(X,Y) \leftarrow hasProperty(X,Y). \\
property(X,Y) \leftarrow join(X,Z), hasProperty(Z,Y). \\
table(t1,t2) \leftarrow \\
table(t1,t3) \leftarrow \\
table(t1,t4) \leftarrow \\
hasProperty(t2,prop1) \leftarrow \\
hasProperty(t3,prop2) \leftarrow \\
hasProperty(t4,prop3) \leftarrow
\end{cases}$$

Clearly, $PRED(P)_{\text{public}} \subseteq PRED(P)$. 

---

1 Clearly, $PRED(P)_{\text{public}} \subseteq PRED(P)$. 

Then \( \text{def}(\text{join}) = \{ \text{join}(X, Y) \leftarrow \text{table}(X, Z), \text{table}(Z, Y), X \neq Y. \} \).

\[
\begin{align*}
\text{def}(\text{property}) &= \left\{ \begin{array}{l}
\text{property}(X, Y) \leftarrow \text{hasProperty}(X, Y), \\
\text{property}(X, Y) \leftarrow \text{join}(X, Z), \text{hasProperty}(Z, Y).
\end{array} \right. \\
\text{def}(\text{hasProperty}) &= \left\{ \begin{array}{l}
\text{hasProperty}(t_2, \text{prop}1) \leftarrow, \\
\text{hasProperty}(t_3, \text{prop}2) \leftarrow, \\
\text{hasProperty}(t_4, \text{prop}3) \leftarrow.
\end{array} \right. \\
\text{def}(\text{table}) &= \left\{ \begin{array}{l}
\text{table}(t_1, t_2) \leftarrow, \\
\text{table}(t_1, t_3) \leftarrow, \\
\text{table}(t_1, t_4) \leftarrow.
\end{array} \right.
\end{align*}
\]

**Definition 2.**

1. The **public definition** of \( P \), denoted \( \text{public}(P) \), is the following set of clauses from \( P \)

\[
\text{public}(P) = \bigcup_{p \in \text{PRED}(P)_{\text{public}}} \text{def}(p)
\]

All clauses from \( \text{public}(P) \) are called **public clauses** of \( P \).

2. The **protected definition** of \( P \), denoted \( \text{protected}(P) \), is the following recursively defined set of clauses from \( P \)

   i) \( \text{protected}(P)^{(1)} = \bigcup_{\exists p(t_1 \ldots t_{k_p}) \leftarrow p_1(t_{11}^1, \ldots t_{n_1}^1), \ldots, p_k(t_{1_k}^k, \ldots t_{n_k}^k) \in P,} \exists p_1(t_{1}^1, \ldots t_{n_1}^1) \text{ such that } p_1 \in \text{PRED}(P)_{\text{public}}} \text{def}(p) \).

   ii) \( \text{protected}(P)^{(i)} = \bigcup_{\exists p(t_1 \ldots t_{k_p}) \leftarrow p_1(t_{11}^1, \ldots t_{n_1}^1), \ldots, p_k(t_{1_k}^k, \ldots t_{n_k}^k) \in P,} \exists p_i(t_{1}^i, \ldots t_{n_i}^i) \text{ such that } \text{def}(p_i) \in \text{protected}(P)^{(i-1)} \text{def}(p) \bigcup \text{protected}(P)^{(i-1)} \)

   \( i \geq 2. \)

   iii) \( \text{protected}(P) = \bigcup_{i \geq 1} \text{protected}(P)^{(i)}. \)

All clauses from \( \text{protected}(P) \) are called **protected clauses** of \( P \).
3. The **private definition** of $P$, denoted $\text{private}(P)$, is the following set of clauses from $P$:

$$\text{private}(P) = P - (\text{public}(P) \cup \text{protected}(P))$$

All clauses from $\text{private}(P)$ are called **private clauses** of $P$.

**Remark 1.** Using access modifiers on a logic program (as in the previous definition) we obtain a partition over the clauses of the program.

**Example 2.** Let $P$ be the Horn program from the previous example.

Let $P_{\text{RED}}(P)_{\text{public}} = \{\text{join}\}$ the declared public predicates from $P_{\text{RED}}(P)$. Then

$$\text{public}(P) = \{ \text{join}(X, Y) \leftarrow \text{table}(X, Z), \text{table}(Z, Y), X \neq Y. \}$$

$$\text{protected}(P) = \{ \text{property}(X, Y) \leftarrow \text{hasProperty}(X, Y). \}
\{ \text{property}(X, Y) \leftarrow \text{join}(X, Z), \text{hasProperty}(Z, Y). \}
\{ \text{table}(t1, t2) \leftarrow \}
\{ \text{table}(t1, t3) \leftarrow \}
\{ \text{table}(t1, t4) \leftarrow \}
\{ \text{hasProperty}(t2, \text{prop}1) \leftarrow \}
\{ \text{hasProperty}(t3, \text{prop}2) \leftarrow \}
\{ \text{hasProperty}(t4, \text{prop}3) \leftarrow \}
\text{private}(P) = \{ \}$$

**Remark 2.** There exists programs $P$ such that $\text{protected}(P) = \emptyset$ and/or $\text{private}(P) = \emptyset$.

**Proof.** Let $P$ be the following Horn program:

$$P : \{ \text{sum}(0, X, X) \leftarrow \text{naturalNumber}(X). \}
\{ \text{sum}(s(X), Y, s(Z)) \leftarrow \text{sum}(X, Y, Z). \}
\{ \text{naturalNumber}(0) \leftarrow \}
\{ \text{naturalNumber}(s(X)) \leftarrow \text{naturalNumber}(X). \}$$

Let $P_{\text{RED}}(P)_{\text{public}} = \{\text{sum}\}$ the declared public predicates from $P_{\text{RED}}(P)$. Then $\text{protected}(P) = \emptyset$.

Let now $P_{\text{RED}}(P)_{\text{public}} = \{\text{naturalNumber}\}$. Then $\text{private}(P) = \emptyset$.

Let $P'$ be the following Horn program:

$$P' : \{ \text{naturalNumber}(0) \leftarrow \}
\{ \text{naturalNumber}(s(X)) \leftarrow \text{naturalNumber}(X). \}$$

Let $P_{\text{RED}}(P')_{\text{public}} = \{\text{naturalNumber}\}$ the declared public predicates from $P_{\text{RED}}(P')$. Then $\text{protected}(P') = \emptyset$ and $\text{private}(P') = \emptyset$. 
**Proposition 1.** There exists $n \in \mathbb{N}$ such that

$$\text{protected}(P)^{(n+1)} = \text{protected}(P)^{(n)}$$

**Proof.** Let $P$ be a Horn program and $PRED(P)_{public}$ the set of public predicates from $P$.

All protected predicates from $P$ are in the set $PRED(P) - PRED(P)_{public}$. Following the Definition 2, we observe that there exists the chain of sets:

$$\text{protected}(P)^{(1)} \subseteq \text{protected}(P)^{(2)} \subseteq \ldots \subseteq (P - \text{public}(P)).$$

Because, $(P - \text{public}(P))$ is a finite set, then there exists $n \in \mathbb{N}$ such that $\text{protected}(P)^{(n+1)} = \text{protected}(P)^{(n)}$.

Let $P$ be a Horn logic program and $PRED(P)_{public}$ the set of public predicates from $P$.

The algorithm below obtain the sets of clauses $\text{public}(P)$, $\text{protected}(P)$ and $\text{private}(P)$:

**INPUT:**
A logic program.
$P$ - the list of all clauses of the logic program.
$PRED(P)$ - the list of all predicate symbols from the logic program.
$PRED(P)_{public}$ - the list of all declared public predicate symbols from the logic program.

**OUTPUT:**
$\text{public}(P)$ - the set of public clauses from the logic program.
$\text{protected}(P)$ - the set of protected clauses from the logic program.
$\text{private}(P)$ - the set of private clauses from the logic program.

/* Calculus of $\text{public}(P)$ */
public(P)={};
for i = 1, ll do
  p=PRED(P)_{public}[i];
  l2 = P.length;
  for j = 1, l2 do
    if ( p == head(P[j]) ) then
      public(P) = public(P).add(P[j]);
      P = P.remove(P[j]);
    endif;
endfor;
endfor;

/* Calculus of protected(P) */
k = 0;
protected(P)={};
protected(P)(k)={};
protected(P)(k+1)={};
for i = 1, l1 do
    p=PRED(P)_public[i];
    for j = 1, l2 do
        if ( p in tail(P[j]) ) then
            protected(P)(k+1).add(def(P[j]));
            P.remove(def(P[j]));
        endif;
    endfor;
endfor;
while ( ! (protected(P)(k) == protected(P)(k+1) ) do
    protected(P)(k+1) = protected(P)(k);
    k = k + 1;
    l3 = protected(P)(k).length;
    for i = 1, l3 do
        p = head(protected(P)(k)[i]);
        l4 = P.length;
        for j = 1, l4 do
            if ( p in tail(P[j]) ) then
                protected(P)(k+1).add(def(p));
                P.remove(def(p));
            endif;
        endfor;
    endfor;
endwhile;
protected(P) = protected(P)(k);

/* Calculus of private(P) */
private(P) = P;

/*
* head(C) - return the symbol predicate from the head
* of the clause C.
* tail(C) - return the list of predicate symbols from the
* atoms in the body of the clause C.
* def(p) - return the list of all clauses with
* the head symbol predicate p.
* add(), remove() are usual operations on sets.
*/
Corollary 1. The previous algorithm will terminate.

Proof. As a result of Proposition 1, the while clause from the algorithm will terminate.

3 The Knowledge Architecture Proposal

In this section we propose an global knowledge architecture based on logic programs with access modifiers (ALP). At this time, all ALP programs are accessible both to the Knowledge Manager and Users.

Also, in this prototype, only the Knowledge Manager can share knowledge between the ALP’s.

In the Figure 1 we present a such knowledge architecture.

![Diagram of the Global Knowledge Architecture](image)

**Fig. 1.** The Global Knowledge Architecture

An architectural design of an ALP may be as in Figure 2. In this diagram there exist a separation between public, protected and private predicates to restrict user access only to public predicates. Also, we have a concrete separation between the layer of private clauses and facts, which are obtained from a database. This correspond of deductive databases point of view.
Fig. 2. A proposed structure of an ALP

References