Derivation of Logic Programs
by Generating Eureka Properties

Francis Alexandre* and Khaled Bsaïes**
* Centre de recherches en informatique de Nancy
  Université Henri Poincaré
** CRIN-CNRS et INRIA Lorraine
  BP 230, 54506 Vandoeuvre, France
e-mail: alexandre@loria.fr

Abstract. This paper presents a method for constructing logic programs by generating so-called eureka properties. The main problem in classic transformations using folding and unfolding is to achieve a folding step. The main interest of the folding is to synthesize some recursive definitions. The success of a folding is often linked to the invention of an eureka. This eureka may be a definition of a predicate (so-called eureka definition or eureka predicate) or a formula which expresses a property of the specification (so-called eureka property). The unfold/fold technique is particularly suited to find some eureka definitions.

However, in many cases the invention of an eureka property is necessary to improve the efficiency. There has been relatively little work on the use and the synthesis of eureka property. The technique presented here allows to synthesize some recursive definitions. In the first part of the technique some schemata of the form $p(t) \leftarrow \Delta, p(\overline{t})$ are obtained by the unfold/fold transformations. Then the main part of the technique is to determine $\Delta$, such that $p(t) \leftarrow \Delta, p(\overline{t})$ is valid under the least Herbrand model of the specification. $\Delta$ is determined by using a proof system, which is able to prove some implicative formulae, which are extended definite clauses.
1 Introduction

Program transformation is an important methodology for constructing correct and efficient programs. It is particularly true for logic programs [9]. In this context, we start with a specification in a Horn clauses form, the objective is to transform it into a more efficient definite program. This approach is particularly useful, if we adopt a top-down approach for writing the specification (this is called compositional style in [9]). A specification built in such a manner, has some qualities of clarity and readability, but in counterpart it generally contains some intermediate predicates or several occurrences of same variables that are the cause of the inefficiency. The paradigm “generate and test” is also a powerful way to write specifications, but these specifications need to be transformed. The main transformation in logic programming are those defined in [13], namely folding, unfolding, introduction of new predicates and goal replacement. The fold transformation plays a special role, because it allows to get a more efficient definition. The main strategies are tupling, loop absorption and generalization [11]. These strategies are particularly suitable to find some sequences of transformation (unfolding and introduction of new predicates (so-called eureka predicates)) in order to achieve a folding. But in some cases, it is necessary to perform a goal replacement in order to achieve a folding, then the problem is to find a property (so-called eureka property). A such property may be a classical property like associativity, commutativity or an other property that must be invented. In this paper we describe a strategy that solves partially the folding problem (when eureka properties are needed) and in some sense this makes more powerful the system of Tamaki and Sato [13]. The method consists in three steps, the generation of schemata by using the unfold/fold transformations, the instantiation of the schemata by using some transformations of the theorem prover [12], this step generates some formulae, the third step is the proof of these formulæ (we do not investigate this last point in this paper).

The paper is organized as follows: section 2 gives some basic definitions and a brief description of the rules used for the strategy. Section 3 is a general description of the strategy. In section 4, an example is developped. Then section 5 is the conclusion and a brief discussion.

2 Description of the Transformations

2.1 Basic Definitions

We assume the reader familiar with the basic notions concerning the logic programming [8] and with the unfold/fold transformations. If \( \epsilon \) is an expression we denote \( V(\epsilon) \) the variables occurring in \( \epsilon \). In the following we denote the atoms by \( A, B, C, ... \), the conjunctions of atoms by \( \land, \lor, \land \), and the substitutions by \( \theta, \sigma, ... \). A substitution is also denoted by a set of the form \( \{x_1/t_1, ..., x_n/t_n\} \), the set of variables \( \{x_1, ..., x_n\} \) is called the domain of the substitution. \text{mgu} \ is an abbreviation for \text{most general unifier}.
A program $P$ is a set of definite clauses, $\mathcal{M}(P)$ denotes the least Herbrand model of $P$. We consider some particular formulae called implicative formulae. These formulae are particularly important because they formalize the notion of program property.

**Definition 1 (Implicative formula).** An implicative formula is a first order formula of the form:

$$
\forall x_1, ..., \forall x_n (\exists y_1, ..., \exists y_m, \Delta \iff \exists z_1, ..., \exists z_k \; A) \text{ where:}
$$

$\Delta$ and $A$ are conjunctions of atoms.

$\{x_1, ..., x_n\} = V(\Delta) \cap V(A)$,

$\{y_1, ..., y_m\} = V(\Delta) \setminus V(A)$ and

$\{z_1, ..., z_k\} = V(A) \setminus V(\Delta)$

A such formula is abbreviated by $\Delta \iff A$ if it is unambiguous.

The existentially quantified variables are also called internal variables, the other variables are universally quantified.

**Definition 2 (Property of a program).** Let $P$ a program and $\pi : \Delta \iff A$ an implicative formula, $\pi$ is a property of $P$ if $\pi$ is valid under the least Herbrand model of $P$.

The replacement of atoms in the bodies of the clauses may be seen as the application of a program property. For example if one consider the property $\text{plus}(x, y, t) \iff \text{plus}(y, z, u) \iff \text{plus}(x, u, r)$, which expresses a part of the associativity of $\text{plus}$, it is possible to replace in a program an instance of its left hand side by an instance of its right hand side.

### 2.2 Description of the Transformations

We use an adaptation of the transformations of the theorem prover described in [12]. This theorem prover allows to prove that a set of implicative formulae is valid under the least Herbrand model of a definite program. In the following all the formulae we consider are implicative formulae.

Let $P$ be a definite program and $\pi : \Delta \iff \Gamma$ a formula. To prove that $\pi$ is valid under $\mathcal{M}(P)$ the general idea is to apply some transformations on the two parts of the formula $\Delta$ and $\Gamma$ in order to obtain some formulae in the form $\Gamma \iff \Gamma'$, $\Delta$ or $\Gamma' \iff \text{false}$ (i.e. the formula true). We describe the deduction rules in the form of inference rules $\frac{\pi}{S'}(NAME)$, in the following $\pi$ is a formula and $S'$ is a formula or a set of formulae, $NAME$ is the name of the rule.

The process of deduction is the following. At each stage of the deduction, we have a set of formulae $S$. We choose a formula $\pi$ of $S$ and we apply a deduction rule on $\pi$, then we replace the chosen formula $\pi$ by the set $S'$ of formulae generated by the deduction rule, then we get the set $S' = (S \setminus \{\pi\}) \cup S''$ of formulae. For each deduction rule $S'$ is given as a function of $S$. 


Definition 3 (Simplification).

\[
\frac{\pi : \Delta, A \leftarrow \Gamma, B}{(\Delta \leftarrow \Gamma) \theta} \quad (SIMP)
\]

\(\theta\) is substitution \(\{x_i/t_i\}\) such that \(A\theta = B\), the domain of \(\theta\) is a subset of the internal variables set of \(A\) and the variables of \(t_i\) occur in \(\Gamma, B\). We have \(S' = (S \setminus \{\pi\}) \cup \{(\Delta \leftarrow \Gamma)\theta\}\).

The simplification allows to suppress some atoms in the both sides of an formula. This rule may be seen as a heuristic rule, because it consists in instantiating some internal variables with terms. This rule does not "preserve the equivalence", \(S\) may be valid under \(M(P)\) while \(S'\) is not, on the other hand if \(S'\) is valid then \(S\) is also valid.

Definition 4 (Unfold-right).

\[
\frac{\pi : \Gamma \leftarrow \Delta, A}{\{\pi_i : (\Gamma \leftarrow \Delta_i)\theta_i, i \in [1,\ldots,k]\}} \quad (UNF_R)
\]

\(E = \{c_1, \ldots, c_k\}\) is the set of clauses of the program \(P\) such that \(c_i : B_i \leftarrow \Delta_i\) and there exists \(\theta_i = \text{mgu}(B_i, A)\).

If \(E = \emptyset\), then we generate the formula \(\Gamma \leftarrow \text{false}\), that can be reduced to the formula \text{true}. We have \(S' = (S \setminus \{\pi\}) \cup \{\pi_i, i \in [1,\ldots,k]\}\).

Let us note that to prove \(\pi\) it is necessary to prove all the formulae \(\pi_i\).

Definition 5 (Unfold-left).

\[
\frac{\pi : \Gamma, A \leftarrow \Delta}{\{\pi_i : (\Gamma, \Delta_i)\theta_i \leftarrow \Delta, i \in [1,\ldots,k]\}} \quad (UNF_L)
\]

\(E = \{c_1, \ldots, c_k\}\) is the set of the clauses of the program \(P\) such that \(c_i : B_i \leftarrow \Delta_i\) and there exists \(\theta_i = \text{mgu}(B_i, A)\), and \(\theta_i\) are existential mgu (i.e. the domain of \(\theta_i\) contains only existentially variables of \(A\) and variables of \(B\)). We have \(S' = (S \setminus \{\pi\}) \cup \{\pi_i\} \text{ for some } i_0 \in [1,\ldots,k]\).

To prove the initial formula \(\pi\), it is sufficient to prove one of the formulae \(\pi_i\) generated by the unfold-left.

3 Description of the method

Let \(P\) be a given program and \(c : p(\overline{t}) \leftarrow \Gamma\) a definite clause such that the symbol \(p\) does not occur in \(P\) and in \(\Gamma\), and \(\overline{t} = t_1, \ldots, t_n\). The method allows to synthesize a recursive definition of \(p\), it consists in two parts. The first part is the generation of some schemata, the second part is the invention of the eureka properties from a schema.
3.1 Generation of the schemata

An interesting feature of the unfold/fold transformations is that they automatically generate some schemata. More precisely, the unfolding introduces some constructors into the clause heads.

For example if we consider the clause

(1) \( \text{div}(a, b, q, r) \leftarrow \text{times}(q, b, t), \text{plus}(t, r, a), \text{less}(r, b) \)

defining the Euclidean division, the unfolding of the atom \( \text{times}(q, b, t) \) with the following clauses

(4) \( \text{times}(0, x, 0) \leftarrow \)
(5) \( \text{times}(s(x), y, z) \leftarrow \text{times}(x, y, u), \text{plus}(u, y, z) \)

produces the clauses:

(8) \( \text{div}(a, b, 0, r) \leftarrow \text{plus}(0, r, a), \text{less}(r, b) \)
(9) \( \text{div}(a, b, s(x), r) \leftarrow \text{times}(x, b, u), \text{plus}(u, b, t), \text{plus}(t, r, a), \text{less}(r, b) \)

which contain the constructors 0 and s in their head. By this way we have got a first part (the head of the clause) of the schema.

The second part may be got by a folding. The folding of (1) in (9) is impossible. If we consider the clause (1') and (9') obtained by removing in (1) and (9) the atoms built with the predicate \( \text{plus} \), it is possible to fold (1)' in (9)'.

(1)' \( \text{div}(a, b, q, r) \leftarrow \text{times}(q, b, t), \text{less}(r, b) \)
(9)' \( \text{div}(a, b, s(x), r) \leftarrow \text{times}(x, b, u), \text{less}(r, b) \)

We get the clause \( \text{div}(a, b, s(x), r) \leftarrow \text{div}(v1, b, x, r) \) that gives the schema \( \text{div}(a, b, s(x), r) \leftarrow \Delta, \text{div}(v1, b, x, r) \).

By the same way and by unfolding the atoms \( \text{plus}(t, r, a) \) and \( \text{less}(r, b) \), it is possible to get the following schemata

\( \text{div}(a, b, q, r) \leftarrow \Delta, \text{div}(a, b, v2, r) \),
\( \text{div}(a, b, q, s(r)) \leftarrow \Delta, \text{div}(v1, b, v2, r) \).

3.2 Invention of the eureka properties

Let \( p(\overline{u}) \leftarrow \Delta, p(\overline{u}_1), \ldots, p(\overline{u}_k) \) be a schema, the problem is now to instantiate this schema, namely to find a conjunction \( \Delta \) such that \( p(\overline{t}) \leftarrow \Delta, p(\overline{u}_1), \ldots, p(\overline{u}_k) \) is valid under \( M(P \cup \{c\}) \).

The general strategy consists in transforming the schema

\( p(\overline{u}) \leftarrow \Delta, p(\overline{u}_1), \ldots, p(\overline{u}_k) \)

into a formula of the form

\( A \leftarrow \Delta \theta \)
where $\Lambda$ is a conjunction of atoms and $\theta$ is a substitution. Then we must determine $\Delta$ verifying $\Lambda \leftarrow \Delta \theta$. Some informations about the predicate symbols and the variables of $\Delta$ allow us to lead the proof.

4 Example

In this section we give an example of the strategy, we stress on the invention of properties.

Example 6. Let $DIV$ be the following program that defines the Euclidean division.

(1) $\text{div}(a, b, q, r) \leftarrow \text{times}(q, b, t), \text{plus}(t, r, a), \text{less}(r, b)$

(2) $\text{plus}(0, x, x) \leftarrow$

(3) $\text{plus}(s(x), y, s(z)) \leftarrow \text{plus}(x, y, z)$

(4) $\text{times}(0, x, 0) \leftarrow$

(5) $\text{times}(s(x), y, z) \leftarrow \text{times}(x, y, u), \text{plus}(u, y, z)$

(6) $\text{less}(0, s(x)) \leftarrow$

(7) $\text{less}(s(x), s(y)) \leftarrow \text{less}(x, y)$

$\text{div}(a, b, q, r)$ holds if and only if $q$ and $r$ are respectively the quotient and the remainder of the division of $a$ by $b$. $\text{times}$ and $\text{plus}$ respectively define the multiplication and the addition. $\text{less}(x, y)$ holds if and only if $x < y$.

1. Generation of the schemata.
2. Instantiation of the schema.
   We choose the schema $\text{div}(a, b, s(q), r) \leftarrow \Delta$, $\text{div}(u, b, q, r)$, and we assume that $V(\Delta) \subseteq \{a, b, q, r, u\}$.
   $\text{div}(a, b, s(q), r) \leftarrow \Delta$, $\text{div}(u, b, q, r)$

\[ \Downarrow \text{unfold-right} \]

$\text{div}(a, b, s(q), r) \leftarrow \Delta$, $\text{times}(q, b, t), \text{plus}(t, r, u), \text{less}(r, b)$

\[ \Downarrow \text{unfold-left} \]

$\text{times}(s(q), b, v), \text{plus}(v, r, a), \text{less}(r, b) \leftarrow \Delta$, $\text{times}(q, b, t), \text{plus}(t, r, u), \text{less}(r, b)$

\[ \Downarrow \text{simplification of the atom less(r, b)} \]

$\text{times}(s(q), b, v), \text{plus}(v, r, a) \leftarrow \Delta$, $\text{times}(q, b, t), \text{plus}(t, r, u)$
\[\Downarrow\text{unfold-left}\]

\[\times(q, b, w), \, \text{plus}(w, b, v), \, \text{plus}(v, r, a) \leftarrow \Delta, \times(q, b, t), \, \text{plus}(t, r, u)\]

\[\Downarrow\text{simplification by using the existential substitution }\{w/t\}\]

\[\text{plus}(t, b, v), \, \text{plus}(v, r, a) \leftarrow \Delta, \, \text{plus}(t, r, u) \quad (C)\]

It is impossible to apply the simplification rule in the formula \((C)\), because \(t, b, r\) and \(a\) are universally quantified.

\[\Downarrow\text{unfold-right (the atom }\text{plus}(t, r, u))\]

\[
\begin{align*}
\{ & \text{plus}(0, b, v), \, \text{plus}(v, r, a) \leftarrow \Delta \theta_1, \\
& \text{plus}(s(t), b, v), \, \text{plus}(v, r, a) \leftarrow \Delta \theta_2, \, \text{plus}(t, r, u) \}
\end{align*}
\]

where \(\theta_1 = \{t/0, u/r\}\) and \(\theta_2 = \{t/s(t), u/s(u)\}\).

By considering the terminal case, the formula:

\[\text{plus}(0, b, v), \, \text{plus}(v, r, a) \leftarrow \Delta \theta_1\]

\[\Downarrow\text{unfold-left with the substitution }\{v/b\}\text{ that does not change }\Delta \theta_1\]

\[\text{plus}(b, r, a) \leftarrow \Delta \theta_1\]

We look for \(\Delta\) such that \(\Delta \theta_1 = \text{plus}(b, r, a)\)

\(\Delta\{t/0, u/r\} = \text{plus}(b, r, a)\) implies \(\Delta\{u/r\} = \text{plus}(b, r, a)\). We get the two following solutions for \(\Delta\):

\[\Delta_1 = \text{plus}(b, u, a)\]

\[\Delta_2 = \text{plus}(b, r, a)\]

By replacing these values in \((C)\) we get the following formulae:

\[
\begin{align*}
\text{plus}(t, b, v), \, \text{plus}(v, r, a) & \leftarrow \text{plus}(b, u, a), \, \text{plus}(t, r, u) \\
\text{plus}(t, b, v), \, \text{plus}(v, r, a) & \leftarrow \text{plus}(b, r, a), \, \text{plus}(t, r, u)
\end{align*}
\]

Only the first formula is valid (it may be proved by an inductive theorem prover, for example this one described in [12]), it expressions a property that combines the associativity and the commutativity of the predicate \(\text{plus}\).

By replacing \(\Delta\) in the initial schema we get the following definition of \(\text{div}\) (in the general case):

\[\text{div}(a, b, s(q), r) \leftarrow \text{plus}(b, u, a), \, \text{div}(u, b, q, r),\]

this definition is correct if we assume that the formula \(\text{plus}(t, b, v), \, \text{plus}(v, r, a) \leftarrow \text{plus}(b, u, a), \, \text{plus}(t, r, u)\) has been proved and it is obviously much more efficient than the initial one.

Moreover this technique avoid to use the goals replacement rule.
5 Conclusion

We have described a method that makes more powerful the system of logic programs transformations based on unfold-fold transformations. We have successfully experimented our method on several simple examples, by using an implementation of the theorem prover defined in [12]. This method is intended to work particularly when eureka properties are needed to achieve a folding. A property of a logic program is an implicative formula, that is more general than a Horn clause. For this reason it is natural and necessary to consider a system of deduction for such formulae, when we must prove some properties of definite programs.

Because of limited place we only cite some works related to the problem of eureka properties and we do not establish a precise comparison. In the domain of logic program transformation, there are few works. The technique described in [10] allows to synthesize some definite clauses from an implicative formula, but the objective is not the same. In [1, 2] a method based on well-moded logic programs generates some eureka properties. In the domain of logic program synthesis [3], the related works are those described in [7] where some schemes are used. The “proofs-as-programs” approach [4–6] is also very close and requires a comparison with the approach described here.

Acknowledgements

We thank the referees for their helpful comments.

References