Unfold/fold Transformation of Logic Programs under Program Completion Semantics

Javier Álvez

Departamento de Sistemas Informáticos y Computación
Universidad Complutense de Madrid

Departamento de Lenguajes y Sistemas Informáticos
Universidad del País Vasco

May, 2009
1 Introduction

2 Folding Transformation Rule

3 Strategies of Transformations: Local Variable Elimination

4 Conclusions and Future Work
Normal Logic Programs

1. $\text{member}(E, [E|\_])$
2. $\text{member}(E, [\_|L]) \leftarrow \text{member}(E, L)$
Normal Logic Programs

1. \textit{member}(E, [E|\_])
2. \textit{member}(E, [\_|L]) \leftarrow \textit{member}(E, L)

3. \textit{intersection}([\_], [\_], [\_])
4. \textit{intersection}([H|L_1], L_2, [H|L_3]) \leftarrow \textit{occurs}(H, L_2), \textit{intersection}(L_1, L_2, L_3)

5. \textit{intersection}([H|L_1], L_2, L_3) \leftarrow \neg\textit{occurs}(H, L_2), \textit{intersection}(L_1, L_2, L_3)
Clark’s Completion

Let $\text{Def}_P[p(x)] = \langle p(t^k) \leftarrow B^k \mid 1 \leq k \leq m \rangle$, the completion formula of the predicate $p \in \text{Pred}_\Sigma(P)$ is

\[
(p(x) \leftrightarrow \bigvee_{k=1}^{m} \exists \bar{z}^k (x \approx t^k \land B^k)) \forall
\]

where $\bar{z}^k = \text{Var}(t^k \cdot B^k)$. 
Clark’s Completion

Let $\text{Def}_P[p(x)] = \{ p(t^k) \leftarrow B^k \mid 1 \leq k \leq m \}$, the completion formula of the predicate $p \in \text{Pred}_\Sigma(P)$ is

$$( p(x) \leftrightarrow \bigvee_{k=1}^{m} \exists \overline{z}^k \left( x \approx t^k \land B^k \right) ) ^\forall$$

where $\overline{z}^k = \text{Var}(t^k \cdot B^k)$.

The Clark’s completion $\text{Comp}(P)$ of a program $P$ consists of:

- the conjunction of the completion formulas of each predicate $p \in \text{Pred}_\Sigma(P)$
- the axioms of the free equality theory $\text{FET}_\Sigma$ or Clark’s equality theory
Clark’s Completion: Example

1. \( \text{member}(E, [E|-]) \)
2. \( \text{member}(E, [-|L]) \leftarrow \text{member}(E, L) \)

\[
( \text{member}(x_1, x_2) \leftrightarrow [ \exists z_1 \cdot z_2 \ ( x_1 \approx z_1 \land x_2 \approx \text{cons}(z_1, z_2) ) \lor \exists z_1 \cdot z_2 \cdot z_3 \ ( x_1 \approx z_1 \land x_2 \approx \text{cons}(z_2, z_3) \land \text{member}(z_1, z_3) ) ] )^\forall
\]
The Clark-Kunen Semantics

The Clark-Kunen semantics of a program $P$ is defined by

$$\text{COMP}[P, \leftarrow \overline{L}] = \{ \varphi \mid \text{Comp}(P) \models_3 (\overline{L} \land \varphi)^\forall \}$$

where $\leftarrow \overline{L}$ is a goal, $\varphi$ is a general equality constraint and $\models_3$ stands for the three-valued logical consequence relation.
The Clark-Kunen Semantics: Motivation

- Constructive negation [Chan/88]
  - Extender in [Chan/89, Stuckey/95, Drabent/95, Fages/97] to a complete and sound operational semantics w.r.t Clark's completion
The Clark-Kunen Semantics: Shepherson’s Operators

Shepherdson’s operators provide a logical characterization of the logical consequences of program completion in three-valued logic.

Let $\text{Def}_P[L] = \langle H_k \leftarrow \overline{B}^k \mid 1 \leq k \leq m \rangle$, Shepherdson’s operators are inductively defined as follows:

- $T^P_0[L] = false$
- $T^P_{n+1}[L] = \bigvee_{k=1}^{m} \exists \overline{w}^k \left( T^P_n[\overline{B}^k] \land \theta_k \right)$
- $F^P_0[L] = false$
- $F^P_{n+1}[L] = \bigwedge_{k=1}^{m} \forall \overline{w}^k \left( F^P_n[\overline{B}^k] \lor \neg \theta_k \right)$

where $\theta_k = \text{mgu}(L, H_k)$
The Clark-Kunen Semantics: Shepherson’s Operators II

Solving method features:

- Incremental
- Shared subformulas
- Lazy
For any program $P$ and any goal $\overline{L}$

$$\varphi \in \text{COMP}[P, \overline{L}] \iff \text{there exists some } n \in \mathbb{N} \text{ such that } \text{FET}_\Sigma \models \left( T^P_n[\overline{L}] \land \varphi \right)$$
Unfold/fold Transformations

\[ P_1 : \]
1. \[ q(X_1, X_2) \leftarrow \text{member}(Y, X_1), \neg \text{member}(Y, X_2) \]
2. \[ \text{member}(E, [E|\_]) \leftarrow \]
3. \[ \text{member}(E, [\_|L]) \leftarrow \text{member}(E, L) \]

Unfolding of \( \text{member}(Y, X_1) \) using clauses 2 and 3
Unfold/fold Transformations

\[ P_1 : \quad 1. \quad q(X_1, X_2) \leftarrow \text{member}(Y, X_1), \neg \text{member}(Y, X_2) \]
\[ \quad 2. \quad \text{member}(E, [E\mid\_]) \leftarrow \]
\[ \quad 3. \quad \text{member}(E, [\_\mid L]) \leftarrow \text{member}(E, L) \]

Unfolding of \( \text{member}(Y, X_1) \) using clauses 2 and 3

\[ P_2 : \quad 4. \quad q([Z_1\mid\_], Z_2) \leftarrow \neg \text{member}(Z_1, Z_2) \]
\[ \quad 5. \quad q([\_\mid Z_1], Z_2) \leftarrow \text{member}(W, Z_1), \neg \text{member}(W, Z_2) \]
\[ \quad 2. \quad \text{member}(E, [E\mid\_]) \leftarrow \]
\[ \quad 3. \quad \text{member}(E, [\_\mid L]) \leftarrow \text{member}(E, L) \]

Folding of \( \langle \text{member}(W, Z_1), \neg \text{member}(W, Z_2) \rangle \) using clause 1
Unfold/fold Transformations

\[ P_1 : \]
1. \( q(X_1, X_2) \leftarrow member(Y, X_1), \neg member(Y, X_2) \)
2. \( member(E, [E|_]) \leftarrow \)
3. \( member(E, [-|L]) \leftarrow member(E, L) \)

Unfolding of \( member(Y, X_1) \) using clauses 2 and 3

\[ P_2 : \]
4. \( q([Z_1|_-], Z_2) \leftarrow \neg member(Z_1, Z_2) \)
5. \( q([-|Z_1], Z_2) \leftarrow member(W, Z_1), \neg member(W, Z_2) \)
2. \( member(E, [E|_]) \leftarrow \)
3. \( member(E, [-|L]) \leftarrow member(E, L) \)

Folding of \( \langle member(W, Z_1), \neg member(W, Z_2) \rangle \) using clause 1

\[ P_3 : \]
4. \( q([Z_1|_-], Z_2) \leftarrow \neg member(Z_1, Z_2) \)
6. \( q([-|Z_1], Z_2) \leftarrow q(Z_1, Z_2) \)
2. \( member(E, [E|_]) \leftarrow \)
3. \( member(E, [-|L]) \leftarrow member(E, L) \)
Unfold/fold Transformations: Motivation

- Efficiency
Unfold/fold Transformations: Motivation

- Efficiency
  - Local variable elimination
Unfold/fold Transformations: Motivation

- Efficiency
  - Local variable elimination
- Proving equivalence
Unfold/fold Transformations: Other Semantics

\[ P_0: \]
1. \( p \leftarrow q, r \)  
2. \( q \leftarrow q \)
Unfold/fold Transformations: Other Semantics

\[ P_0: \]
1. \( p \leftarrow q, r \)  
2. \( q \leftarrow q \)

Unfolding of \( q \) in clause 1 using clause 2

\[ P_1: \]
3. \( p \leftarrow q, r \)  
2. \( q \leftarrow q \)
Unfold/fold Transformations: Other Semantics

\[ P_0: \]
1. \( p \leftarrow q, r \)
2. \( q \leftarrow q \)

Unfolding of \( q \) in clause 1 using clause 2

\[ P_1: \]
3. \( p \leftarrow q, r \)
2. \( q \leftarrow q \)

Folding of \( q, r \) in clause 3 using clause 1

\[ P_2: \]
4. \( p \leftarrow p \)
2. \( q \leftarrow q \)
Logic Program Transformation

- Among the systems in the literature, the conditions that ensure the correctness of the rule unfolding are similar
- Regarding the rule folding, there are two main proposals:
  - *Reversible folding* systems [Gardner & Shepherdson/91, Maher/88]
  - *À la Tamaki-Sato* systems [Bossi & Etalle/94, Sato/92]
1 Introduction

2 Folding Transformation Rule
   - Previous Work
   - New Decidable Conditions
   - Correctness
   - Results

3 Strategies of Transformations: Local Variable Elimination

4 Conclusions and Future Work
Previous Work

Two main approaches:

- **Reversible folding systems:**
  - Only the clauses in the current program can be used as folder clauses
  - Systems: [Maher/88] and [Gardner & Shepherdson/91]

- **À la Tamaki-Sato systems:**
  - Split predicates into *old* and *new* ones
  - *Old* predicates cannot depend on *new* ones and only the clauses with a *new* predicate in the head can be used as folder clauses
  - The predicate in the head of the folded clause is *old* or all the literals to be folded comes from unfolding
  - Systems: [Seki/91], [Bossi & Etalle/94] and [Sato/92]
Previous Work II

\[ P_1 : \]
1. \( q(X_1, X_2) \leftarrow \text{member}(Y, X_1), \neg \text{member}(Y, X_2) \)
2. \( \text{member}(E, [E|\_]) \leftarrow \)
3. \( \text{member}(E, [\_|L]) \leftarrow \text{member}(E, L) \)

Unfolding of \( \text{member}(Y, X_1) \) using clauses 2 and 3

\[ P_2 : \]
4. \( q([Z_1|\_], Z_2) \leftarrow \neg \text{member}(Z_1, Z_2) \)
5. \( q([\_|Z_1], Z_2) \leftarrow \text{member}(W, Z_1), \neg \text{member}(W, Z_2) \)
2. \( \text{member}(E, [E|\_]) \leftarrow \)
3. \( \text{member}(E, [\_|L]) \leftarrow \text{member}(E, L) \)

Folding of \( \langle \text{member}(W, Z_1), \neg \text{member}(W, Z_2) \rangle \) using clause 1

\[ P_3 : \]
4. \( q([Z_1|\_], Z_2) \leftarrow \neg \text{member}(Z_1, Z_2) \)
6. \( q([\_|Z_1], Z_2) \leftarrow q(Z_1, Z_2) \)
2. \( \text{member}(E, [E|\_]) \leftarrow \)
3. \( \text{member}(E, [\_|L]) \leftarrow \text{member}(E, L) \)
Our Aim

- To propose a new folding rule with two main properties:
  - The folder clause can be taken from any program in the transformation sequence
  - Some literals that do not come from unfolding can also be folded

\[ P_1 : \begin{align*} 
q(X_1, X_2) &\leftarrow \text{member}(Y, X_1), \neg\text{member}(Y, X_2) 
\end{align*} \]

Unfolding

\[ P_2 : \begin{align*} 
q([-|Z_1], Z_2) &\leftarrow \text{member}(W, Z_1), \neg\text{member}(W, Z_2) 
\end{align*} \]

Folding using clause 1

\[ P_3 : \begin{align*} 
q([-|Z_1], Z_2) &\leftarrow q(Z_1, Z_2) 
\end{align*} \]
New Decidable Conditions

**Condition A**: the literal introduced by folding does not depend on the head of the folded clause in the current program

\[ P_1 : 1. \text{add}(0, N, N) \leftarrow \]
\[ 2. \text{add}(s(N_1), N_2, s(N_3)) \leftarrow \text{add}(N_1, N_2, N_3) \]
\[ 3. \text{add3}(N_1, N_2, N_3, N_4) \leftarrow \text{add}(N_1, N_2, Y), \text{add}(Y, N_3, N_4) \]
**New Decidable Conditions**

**Condition A:** the literal introduced by folding does not depend on the head of the folded clause in the current program

\[ P_1 : 1. \text{add}(0, N, N) \leftarrow \]
\[ 2. \text{add}(s(N_1), N_2, s(N_3)) \leftarrow \text{add}(N_1, N_2, N_3) \]
\[ 3. \text{add3}(N_1, N_2, N_3, N_4) \leftarrow \text{add}(N_1, N_2, Y), \text{add}(Y, N_3, N_4) \]

Unfolding of \( \text{add}(N_1, N_2, Y) \) using clauses 1 and 2

\[ P_2 : 4. \text{add3}(0, N_2, N_3, N_4) \leftarrow \text{add}(N_2, N_3, N_4) \]
\[ 5. \text{add3}(s(N_1), N_2, N_3, N_4) \leftarrow \text{add}(N_1, N_2, Y), \text{add}(s(Y), N_3, N_4) \]
New Decidable Conditions

**Condition A:** the literal introduced by folding does not depend on the head of the folded clause in the current program

\[
P_1 : \begin{align*}
1. \ & \text{add}(0, N, N) \leftarrow \\
2. \ & \text{add}(s(N_1), N_2, s(N_3)) \leftarrow \text{add}(N_1, N_2, N_3) \\
3. \ & \text{add3}(N_1, N_2, N_3, N_4) \leftarrow \text{add}(N_1, N_2, Y), \ \text{add}(Y, N_3, N_4)
\end{align*}
\]

Unfolding of \(\text{add}(N_1, N_2, Y)\) using clauses 1 and 2

\[
P_2 : \begin{align*}
4. \ & \text{add3}(0, N_2, N_3, N_4) \leftarrow \text{add}(N_2, N_3, N_4) \\
5. \ & \text{add3}(s(N_1), N_2, N_3, N_4) \leftarrow \text{add}(N_1, N_2, Y), \ \text{add}(s(Y), N_3, N_4)
\end{align*}
\]

Introduce the new predicate \(\text{add4/5}\)

\[
P_3 : \begin{align*}
6. \ & \text{add4}(N_1, N_2, N_3, N_4, N_5) \leftarrow \text{add}(N_1, N_2, Y_1), \ \text{add}(Y_1, N_3, Y_2), \\
& \quad \text{add}(Y_2, N_4, N_5)
\end{align*}
\]
New Decidable Conditions

**Condition A:** the literal introduced by folding does not depend on the head of the folded clause in the current program

\[ P_1 : \]
1. \( \text{add}(0, N, N) \leftarrow \)
2. \( \text{add}(s(N_1), N_2, s(N_3)) \leftarrow \text{add}(N_1, N_2, N_3) \)
3. \( \text{add3}(N_1, N_2, N_3, N_4) \leftarrow \text{add}(N_1, N_2, Y), \text{add}(Y, N_3, N_4) \)

Unfolding of \( \text{add}(N_1, N_2, Y) \) using clauses 1 and 2

\[ P_2 : \]
4. \( \text{add3}(0, N_2, N_3, N_4) \leftarrow \text{add}(N_2, N_3, N_4) \)
5. \( \text{add3}(s(N_1), N_2, N_3, N_4) \leftarrow \text{add}(N_1, N_2, Y), \text{add}(s(Y), N_3, N_4) \)

Introduce the new predicate \( \text{add4/5} \)

\[ P_3 : \]
6. \( \text{add4}(N_1, N_2, N_3, N_4, N_5) \leftarrow \text{add}(N_1, N_2, Y_1), \text{add}(Y_1, N_3, Y_2), \text{add}(Y_2, N_4, N_5) \)

Folding of \( \langle \text{add}(N_1, N_2, Y), \text{add}(Y_1, N_3, Y_2) \rangle \) using clause 3

\[ P_4 : \]
7. \( \text{add4}(N_1, N_2, N_3, N_4, N_5) \leftarrow \text{add3}(N_1, N_2, N_3, Y_2), \text{add}(Y_2, N_4, N_5) \)
New Decidable Conditions II

Condition B: the literals to be folded \( \overline{L} \) can be partitioned into \( \overline{M}, \overline{N} \) where

- the literals in \( \overline{N} \) come from unfolding,
- \( \overline{M} \) is non-failing on the variables of the literal introduced by folding
New Decidable Conditions II

**Condition B:** the literals to be folded $\overline{L}$ can be partitioned into $\overline{M}$, $\overline{N}$ where

- the literals in $\overline{N}$ come from unfolding,
- $\overline{M}$ is non-failing on the variables of the literal introduced by folding

\[ P_1 : 3. \; \text{add3}(N_1, N_2, N_3, N_4) \leftarrow \text{add}(N_1, N_2, Y), \; \text{add}(Y, N_3, N_4) \]

\[ P_3 : 6. \; \text{add3}(s(N_1), N_2, N_3, s(N_4)) \leftarrow \text{add}(N_1, N_2, Y), \; \text{add}(Y, N_3, N_4) \]

\[ P_5 : 8. \; \text{add3}(s(N_1), N_2, N_3, s(N_4)) \leftarrow \text{add3}(N_1, N_2, N_3, N_4) \]

Both literals $\text{add}(N_1, N_2, Y)$ and $\text{add}(Y, N_3, N_4)$ come from a previous unfolding step
**Condition B:** the literals to be folded $\overline{L}$ can be partitioned into $\overline{M}$, $\overline{N}$ where

- the literals in $\overline{N}$ come from unfolding,
- $\overline{M}$ is non-failing on the variables of the literal introduced by folding
New Decidable Conditions III

**Condition B:** the literals to be folded $\overline{L}$ can be partitioned into $\overline{M}$, $\overline{N}$ where

- the literals in $\overline{N}$ come from unfolding,
- $\overline{M}$ is non-failing on the variables of the literal introduced by folding

\[
P_1 : \quad 1. \quad q(X_1, X_2) \leftarrow \text{member}(Y, X_1), \neg\text{member}(Y, X_2)
\]

Unfolding

\[
P_2 : \quad 5. \quad q([\_|Z_1], Z_2) \leftarrow \text{member}(W, Z_1), \neg\text{member}(W, Z_2)
\]

Folding using clause 1

\[
P_3 : \quad 6. \quad q([\_|Z_1], Z_2) \leftarrow q(Z_1, Z_2)
\]
New Decidable Conditions III

**Condition B**: the literals to be folded \( \overline{L} \) can be partitioned into \( \overline{M} \), \( \overline{N} \) where

- the literals in \( \overline{N} \) come from unfolding,
- \( \overline{M} \) is non-failing on the variables of the literal introduced by folding

\[
P_1 : \quad 1. \quad q(X_1, X_2) \leftarrow \text{member}(Y, X_1), \neg \text{member}(Y, X_2)
\]
Unfolding

\[
P_2 : \quad 5. \quad q([\_|Z_1], Z_2) \leftarrow \text{member}(W, Z_1), \neg \text{member}(W, Z_2)
\]
Folding using clause 1

\[
P_3 : \quad 6. \quad q([\_|Z_1], Z_2) \leftarrow q(Z_1, Z_2)
\]

Is \( \neg \text{member}(W, Z_2) \) non-failing on \( Z_2 \)?
Non-failure Analysis

- Non-failure analysis problem is shown to be decidable in [Debray & López-García & Hermenegildo/97].

- A goal $L$ is non-failing on $z \subseteq \text{Var}(L)$ iff for all substitution $\sigma$ of domain $x$ and any fair literal selection rule there exists at least one derivation starting from $\leftarrow L\sigma$ that does not fail.

- Using Shepherdson’s operators, $L$ is non-failing on $z \subseteq \text{Var}(L)$ iff for all $n \in \mathbb{N}$

$$\text{FET}_\Sigma \models \forall x \ \exists y \ \left( \neg F_n^P[L] \right)$$

Roughly speaking, $\neg \text{member}(W, Z_2)$ is non-failing on $Z_2$ since for every list $Z_2$ there always exists a term $W$ such that $W$ is not a member of $Z_2$. 
Preservation of Successes

- It is easy to prove, by induction on the number of iterations of the Shepherdson’s operator $T$, that successes are preserved.
- A similar proof can be found in [Kanamori & Fujita/86]
Failures are preserved from $P_{i+1}$ to $P_i$ if the program $P_{i+1}$ is obtained by folding from $P_i$

$P_1$: 
1. $p \leftarrow q, r$
2. $q \leftarrow q$

Unfolding of $q$ using clause 3

$P_2$: 
3. $p \leftarrow q, r$
2. $q \leftarrow q$

Folding of $\langle q, r \rangle$ using clause 1

$P_3$: 
4. $p \leftarrow p$
2. $q \leftarrow q$
For proving that failures are preserved from $P_i$ to $P_{i+1}$ if the program $P_{i+1}$ is obtained by folding from $P_i$, we use either Condition A or B

$P_1$: 
1. $p \leftarrow q, r$
2. $q \leftarrow q$

Unfolding of $q$ using clause 3

$P_2$: 
3. $p \leftarrow q, r$
2. $q \leftarrow q$

Folding of $\langle q, r \rangle$ using clause 1

$P_3$: 
4. $p \leftarrow p$
2. $q \leftarrow q$
Preservation of Failures

- For proving that failures are preserved from $P_i$ to $P_{i+1}$ if the program $P_{i+1}$ is obtained by folding from $P_i$, we use either Condition A or B

  $P_1$:
  1. \( p \leftarrow q, r \)
  2. \( q \leftarrow q \)
  
  Unfolding of $q$ using clause 3

  $P_2$:
  3. \( p \leftarrow q, r \)
  2. \( q \leftarrow q \)
  
  Folding of \( \langle q, r \rangle \) using clause 1

  $P_3$:
  4. \( p \leftarrow p \)
  2. \( q \leftarrow q \)

- the literal introduced by folding depends on the head of the folded clause in the current program $\Rightarrow$ Condition A does not hold
Preservation of Failures

For proving that failures are preserved from $P_i$ to $P_{i+1}$ if the program $P_{i+1}$ is obtained by folding from $P_i$, we use either Condition A or B

$P_1$: 1. $p \leftarrow q, r$ 2. $q \leftarrow q$

Unfolding of $q$ using clause 3

$P_2$: 3. $p \leftarrow q, r$ 2. $q \leftarrow q$

Folding of $\langle q, r \rangle$ using clause 1

$P_3$: 4. $p \leftarrow p$ 2. $q \leftarrow q$

- the literal introduced by folding depends on the head of the folded clause in the current program $\Rightarrow$ Condition A does not hold
- $r$ does not come from unfolding and is failing $\Rightarrow$ Condition B does not hold
Results

- New conditions $A$ and $B$ preserve the Clark-Kunen semantics
- The resulting system enables more transformations than previous unfold/fold systems in the literature
Unfold/fold Transformation of Logic Programs under Program Completion Semantics

Strategies of Transformations: Local Variable Elimination

1. Introduction

2. Folding Transformation Rule

3. Strategies of Transformations: Local Variable Elimination

4. Conclusions and Future Work
Strategies of Transformation

- Till now, we have focused on applicability conditions and correctness
- New objective: automatic transformation of programs
- Example of application: elimination of local variables
Local variables are the main source of inefficiency when dealing with negation

\[ q(X_1, X_2) \leftarrow \text{member}(Y, X_1), \neg\text{member}(Y, X_2) \]

\[ \Downarrow \]

\[ (\neg q(x_1, x_2) \leftrightarrow \exists \exists ( x_1 \approx z_1 \land x_2 \approx z_2 \land \forall y [ \neg\text{member}(y, z_1) \lor \text{member}(y, z_2) ] ) ) \forall \]
In the logic programming paradigm, arguments can be used as both input and output.

However, several works are devoted to modes in logic programming:
- Dataflow
- Program analysis

We assign modes according to local variables:

\[ q(X_1, X_2) \leftarrow \text{member}(Y, X_1), \neg\text{member}(Y, X_2) \]

- \text{member}(Y, X_1): the first occurrence of \( Y \) ⇒ the first argument of \text{member}_{/2} \text{ is out}
- \( \neg\text{member}(Y, X_2) \): the second occurrence of \( Y \) ⇒ the first argument of \text{member}_{/2} \text{ is in}
A program $P$ is \textit{well-moded} iff for every clause $C \in P$

$$C = H(\overline{\text{t}}^0 \triangleright \overline{s}^{n+1}) \leftarrow L_1(\overline{s}^1 \triangleright \overline{t}^1), \ldots, L_n(\overline{s}^n \triangleright \overline{t}^n)$$

we have that

$$\text{Var}(\overline{s}^i) \subseteq \bigcup_{j=0}^{i-1} \text{Var}(\overline{t}^j)$$
The basic transformation step eliminates the local variables that are shared by two consecutive literals.

It consists in an unfold/fold transformation sequence: unfolding of each literal (or just one) and folding the resulting literals.

We provide decidable conditions ensuring that the basic transformation step is applicable.

We can use the basic transformation step for automatically removing local variables for a subclass of normal logic programs.
Example

$P_1:$
1. $q(X_1, X_2) \leftarrow \text{member}(Y, X_1), \neg \text{member}(Y, X_2)$
2. $\text{member}(E, [E|]) \leftarrow$
3. $\text{member}(E, [-|L]) \leftarrow \text{member}(E, L)$

Unfolding of $\text{member}(Y, X_1)$ using clauses 2 and 3

$P_2:$
4. $q([Z_1|], Z_2) \leftarrow \neg \text{member}(Z_1, Z_2)$
5. $q([-|Z_1], Z_2) \leftarrow \text{member}(W, Z_1), \neg \text{member}(W, Z_2)$
2. $\text{member}(E, [E|]) \leftarrow$
3. $\text{member}(E, [-|L]) \leftarrow \text{member}(E, L)$

Folding of $\langle \text{member}(W, Z_1), \neg \text{member}(W, Z_2) \rangle$ using clause 1

$P_3:$
4. $q([Z_1|], Z_2) \leftarrow \neg \text{member}(Z_1, Z_2)$
6. $q([-|Z_1], Z_2) \leftarrow q(Z_1, Z_2)$
2. $\text{member}(E, [E|]) \leftarrow$
3. $\text{member}(E, [-|L]) \leftarrow \text{member}(E, L)$
1 Introduction

2 Folding Transformation Rule
   • Previous Work
   • New Decidable Conditions
   • Correctness
   • Results

3 Strategies of Transformations: Local Variable Elimination

4 Conclusions and Future Work
Conclusions and Future Work

- Transformation rules highly depend on the considered semantics.
- A strong semantics restricts the application of transformation rules.
- Is it interesting to use a weaker semantics?
Example: a General Transformation for Well-Moded Programs

- We propose an alternative transformation when the basic transformation step is not applicable.
- This transformation eliminates all the local variables from well-moded programs.
- The transformation is based in the *call stack* technique for turning recursion into tail recursion.
- The transformation preserves a semantic notion that is weaker than the Clark-Kunen semantics.
Future Work

- Regarding the rule unfolding

  \( P_0 : \quad p \leftarrow q \quad q \leftarrow r \quad r \leftarrow \)

  Unfolding of \( q \) using its definition in \( P_0 \)

  \( P_1 : \quad p \leftarrow r \quad q \leftarrow r \quad r \leftarrow \)

  Folding of \( r \) the 1\(^{st} \) clause

  \( P_2 : \quad p \leftarrow r \quad q \leftarrow p \quad r \leftarrow \)

  Unfolding \( p \) using its definition in \( P_0 \)

  \( P_3 : \quad p \leftarrow r \quad q \leftarrow q \quad r \leftarrow \)

- We want to look for new conditions for the rule unfolding that ensure the preservation of the Clark-Kunen semantics when using definitions in previous programs