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

Normal Logic Programs

1. $member(E, [E|\_])$
2. $member(E, [\_|L]) \leftarrow member(E, L)$

Normal Logic Programs

$$
\begin{array}{ll}
1. & member(E, [E|\_]) \\
2. & member(E, [\_|L]) \leftarrow member(E, L) \\
 & \\
3. & intersection([\,], \_, [\,]) \\
4. & intersection([H|L_1], L_2, [H|L_3]) \leftarrow occurs(H, L_2), \\
 & \qquad\qquad\qquad\qquad\qquad\qquad\qquad intersection(L_1, L_2, L_3) \\
5. & intersection([H|L_1], L_2, L_3) \leftarrow \neg occurs(H, L_2), \\
 & \qquad\qquad\qquad\qquad\qquad\qquad intersection(L_1, L_2, L_3)
\end{array}
$$

Clark's Completion

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

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

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

Clark's Completion

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

$$(\, p(\overline{x}) \; \leftrightarrow \; \bigvee_{k=1}^{m} \exists \overline{z}^k \; (\; \overline{x} \approx \overline{t}^k \wedge \overline{B}^k \;) \;)^\forall$$

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

The *Clark's completion* $\mathtt{Comp}(P)$ *of a program* P consists of:

- the conjunction of the completion formulas of each predicate $p \in \mathtt{Pred}_\Sigma(P)$
- the axioms of the *free equality theory* $\mathtt{FET}_\Sigma$ or Clark's equality theory

Clark's Completion: Example

1. $member(E, [E|\_])$
2. $member(E, [\_|L]) \leftarrow member(E, L)$

$$\begin{aligned}(\, member(x_1, x_2) \quad \leftrightarrow \quad [\, & \exists z_1 \cdot z_2 \, (\, x_1 \approx z_1 \wedge x_2 \approx cons(z_1, z_2) \,) \;\vee \\ & \exists z_1 \cdot z_2 \cdot z_3 \, (\, x_1 \approx z_1 \wedge x_2 \approx cons(z_2, z_3) \wedge \\ & \qquad\qquad member(z_1, z_3) \,) \,] \,)^{\forall}\end{aligned}$$

The Clark-Kunen Semantics

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

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

where $\leftarrow \overline{L}$ is a goal, φ 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 $\mathtt{Def}_P[L] = \langle\, H_k \leftarrow \overline{B}^k \mid 1 \leq k \leq m \,\rangle$, Shepherdson's operators are inductively defined as follows

$$
\begin{array}{lllllll}
\mathtt{T}_0^P[L] & = & false & \qquad & \mathtt{T}_{n+1}^P[L] & = & \bigvee_{k=1}^{m} \exists \overline{w}^k \; (\, \mathtt{T}_n^P[\overline{B}^k] \wedge \theta_k \,) \\
\mathtt{F}_0^P[L] & = & false & \qquad & \mathtt{F}_{n+1}^P[L] & = & \bigwedge_{k=1}^{m} \forall \overline{w}^k \; (\, \mathtt{F}_n^P[\overline{B}^k] \vee \neg\theta_k \,)
\end{array}
$$

where $\theta_k = \mathtt{mgu}(L, H_k)$

The Clark-Kunen Semantics: Shepherson's Operators II

Solving method features:

- Incremental
- Shared subformulas
- Lazy

The Clark-Kunen Semantics: Shepherson's Operators III

For any program P and any goal $\overline{L}$

$$\varphi \in \mathtt{COMP}[P, \overline{L}] \iff \text{there exists some } n \in \mathbb{N} \text{ such that } \mathtt{FET}_{\Sigma} \models (\ \mathtt{T}^{P}_{n}[\overline{L}] \wedge \varphi\ )$$

Unfold/fold Transformations

$$
\begin{array}{lll}
P_1: & 1. & q(X_1, X_2) \leftarrow \underline{member(Y, X_1)}, \neg member(Y, X_2) \\
 & 2. & member(E, [E|\_]) \leftarrow \\
 & 3. & member(E, [\_|L]) \leftarrow member(E, L)
\end{array}
$$

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

Unfold/fold Transformations

$P_1:$ 1. $q(X_1, X_2) \leftarrow \underline{member(Y, X_1)}, \neg member(Y, X_2)$
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. $q([\_|Z_1], Z_2) \leftarrow \underline{member(W, Z_1), \neg member(W, Z_2)}$
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

Unfold/fold Transformations

$$
\begin{array}{lll}
P_1: & 1. & q(X_1, X_2) \leftarrow \underline{member(Y, X_1)}, \neg member(Y, X_2) \\
 & 2. & member(E, [E|\_]) \leftarrow \\
 & 3. & member(E, [\_|L]) \leftarrow member(E, L)
\end{array}
$$

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

$$
\begin{array}{lll}
P_2: & 4. & q([Z_1|\_], Z_2) \leftarrow \neg member(Z_1, Z_2) \\
 & 5. & q([\_|Z_1], Z_2) \leftarrow \underline{member(W, Z_1), \neg member(W, Z_2)} \\
 & 2. & member(E, [E|\_]) \leftarrow \\
 & 3. & member(E, [\_|L]) \leftarrow member(E, L)
\end{array}
$$

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

$$
\begin{array}{lll}
P_3: & 4. & q([Z_1|\_], Z_2) \leftarrow \neg member(Z_1, Z_2) \\
 & 6. & q([\_|Z_1], Z_2) \leftarrow q(Z_1, Z_2) \\
 & 2. & member(E, [E|\_]) \leftarrow \\
 & 3. & member(E, [\_|L]) \leftarrow member(E, L)
\end{array}
$$

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. $p \leftarrow \underline{q},\ r$ 2. $q \leftarrow q$

Unfold/fold Transformations: Other Semantics

P_0: 1. $p \leftarrow \underline{q},\ r$ 2. $q \leftarrow q$

Unfolding of q in clause 1 using clause 2

P_1: 3. $p \leftarrow \underline{q,\ r}$ 2. $q \leftarrow q$

Unfold/fold Transformations: Other Semantics

P_0: $1.\ p \leftarrow \underline{q},\ r$ $2.\ q \leftarrow q$

Unfolding of q in clause 1 using clause 2

P_1: $3.\ p \leftarrow \underline{q,\ r}$ $2.\ q \leftarrow q$

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]

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

$$
\begin{array}{lll}
P_1: & 1. & q(X_1, X_2) \leftarrow \underline{member(Y, X_1)}, \neg member(Y, X_2) \\
 & 2. & member(E, [E|\_]) \leftarrow \\
 & 3. & member(E, [\_|L]) \leftarrow member(E, L)
\end{array}
$$

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

$$
\begin{array}{lll}
P_2: & 4. & q([Z_1|\_], Z_2) \leftarrow \neg member(Z_1, Z_2) \\
 & 5. & q([\_|Z_1], Z_2) \leftarrow \underline{member(W, Z_1), \neg member(W, Z_2)} \\
 & 2. & member(E, [E|\_]) \leftarrow \\
 & 3. & member(E, [\_|L]) \leftarrow member(E, L)
\end{array}
$$

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

$$
\begin{array}{lll}
P_3: & 4. & q([Z_1|\_], Z_2) \leftarrow \neg member(Z_1, Z_2) \\
 & 6. & q([\_|Z_1], Z_2) \leftarrow q(Z_1, Z_2) \\
 & 2. & member(E, [E|\_]) \leftarrow \\
 & 3. & member(E, [\_|L]) \leftarrow member(E, L)
\end{array}
$$

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

$$
\begin{array}{lll}
P_1: & 1. & q(X_1, X_2) \leftarrow \underline{member(Y, X_1)}, \neg member(Y, X_2) \\
 & & \text{Unfolding} \\
P_2: & 5. & q([\_|Z_1], Z_2) \leftarrow \underline{member(W, Z_1), \neg member(W, Z_2)} \\
 & & \text{Folding using clause 1} \\
P_3: & 6. & q([\_|Z_1], Z_2) \leftarrow q(Z_1, Z_2)
\end{array}
$$

New Decidable Conditions

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

$$
\begin{aligned}
P_1 :\ & 1.\ add(0, N, N) \leftarrow \\
& 2.\ add(s(N_1), N_2, s(N_3)) \leftarrow add(N_1, N_2, N_3) \\
& 3.\ add3(N_1, N_2, N_3, N_4) \leftarrow \underline{add(N_1, N_2, Y)},\ add(Y, N_3, N_4)
\end{aligned}
$$

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. $add(0, N, N) \leftarrow$

2. $add(s(N_1), N_2, s(N_3)) \leftarrow add(N_1, N_2, N_3)$

3. $add3(N_1, N_2, N_3, N_4) \leftarrow \underline{add(N_1, N_2, Y)},\ add(Y, N_3, N_4)$

Unfolding of $add(N_1, N_2, Y)$ using clauses 1 and 2

P_2 : 4. $add3(0, N_2, N_3, N_4) \leftarrow add(N_2, N_3, N_4)$

5. $add3(s(N_1), N_2, N_3, N_4) \leftarrow add(N_1, N_2, Y),\ add(s(Y), N_3, N_4)$

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 : 1. $add(0, N, N) \leftarrow$

2. $add(s(N_1), N_2, s(N_3)) \leftarrow add(N_1, N_2, N_3)$

3. $add3(N_1, N_2, N_3, N_4) \leftarrow \underline{add(N_1, N_2, Y)},\ add(Y, N_3, N_4)$

Unfolding of $add(N_1, N_2, Y)$ using clauses 1 and 2

P_2 : 4. $add3(0, N_2, N_3, N_4) \leftarrow add(N_2, N_3, N_4)$

5. $add3(s(N_1), N_2, N_3, N_4) \leftarrow add(N_1, N_2, Y),\ add(s(Y), N_3, N_4)$

Introduce the new predicate $add4_{/5}$

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

New Decidable Conditions

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

$$
\begin{array}{l}
P_1 : 1.\ add(0, N, N) \leftarrow \\
\quad\quad 2.\ add(s(N_1), N_2, s(N_3)) \leftarrow add(N_1, N_2, N_3) \\
\quad\quad 3.\ add3(N_1, N_2, N_3, N_4) \leftarrow \underline{add(N_1, N_2, Y)},\ add(Y, N_3, N_4)
\end{array}
$$

Unfolding of $add(N_1, N_2, Y)$ using clauses 1 and 2

$$
\begin{array}{l}
P_2 : 4.\ add3(0, N_2, N_3, N_4) \leftarrow add(N_2, N_3, N_4) \\
\quad\quad 5.\ add3(s(N_1), N_2, N_3, N_4) \leftarrow add(N_1, N_2, Y),\ add(s(Y), N_3, N_4)
\end{array}
$$

Introduce the new predicate $add4_{/5}$

$$
\begin{array}{l}
P_3 : 6.\ add4(N_1, N_2, N_3, N_4, N_5) \leftarrow \underline{add(N_1, N_2, Y_1),\ add(Y_1, N_3, Y_2)}, \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad add(Y_2, N_4, N_5)
\end{array}
$$

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

$$
\begin{array}{l}
P_4 : 7.\ add4(N_1, N_2, N_3, N_4, N_5) \leftarrow add3(N_1, N_2, N_3, Y_2), \\
\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad\quad add(Y_2, N_4, N_5)
\end{array}
$$

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

$$P_1 : 3.\ add3(N_1, N_2, N_3, N_4) \leftarrow \underline{add(N_1, N_2, Y)},\ add(Y, N_3, N_4)$$

$$P_3 : 6.\ add3(s(N_1), N_2, N_3, s(N_4)) \leftarrow \underline{add(N_1, N_2, Y),\ add(Y, N_3, N_4)}$$

$$P_5 : 8.\ add3(s(N_1), N_2, N_3, s(N_4)) \leftarrow add3(N_1, N_2, N_3, N_4)$$

Both literals $add(N_1, N_2, Y)$ and $add(Y, N_3, N_4)$ come from a previous unfolding step

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

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 : 1. $q(X_1, X_2) \leftarrow \underline{member(Y, X_1)}, \neg member(Y, X_2)$

Unfolding

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

Folding using clause 1

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

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 : 1. $q(X_1, X_2) \leftarrow \underline{member(Y, X_1)}, \neg member(Y, X_2)$

Unfolding

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

Folding using clause 1

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

Is $\neg member(W, Z_2)$ 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 $\overline{L}$ is non-failing on $\overline{z} \subseteq \mathtt{Var}(\overline{L})$ iff for all substitution σ of domain $\overline{x}$ and any fair literal selection rule there exists at least one derivation starting from $\leftarrow \overline{L}\sigma$ that does not fail
- Using Shepherdson's operators, $\overline{L}$ is non-failing on $\overline{z} \subseteq \mathtt{Var}(\overline{L})$ iff for all $n \in \mathbb{N}$

$$\mathtt{FET}_\Sigma \models \forall \overline{x}\, \exists \overline{y}\ (\ \neg \mathtt{F}_n^P[\overline{L}]\ )$$

Roughly speaking, $\neg member(W, Z_2)$ 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 $\mathtt{T}$, that successes are preserved
- A similar proof can be found in [Kanamori & Fujita/86]

Preservation of Failures

- Failures are preserved from P_{i+1} to P_i if the program P_{i+1} is obtained by folding from P_i

$$
\begin{array}{llll}
P_1: & 1.\quad p \leftarrow \underline{q},\ r & & 2.\quad q \leftarrow q \\
 & \text{Unfolding of } q \text{ using clause 3} & & \\
P_2: & 3.\quad p \leftarrow \underline{q,\ r} & & 2.\quad q \leftarrow q \\
 & \text{Folding of } \langle q,\ r\rangle \text{ using clause 1} & & \\
P_3: & 4.\quad p \leftarrow p & & 2.\quad q \leftarrow q
\end{array}
$$

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. $p \leftarrow \underline{q},\ r$ 2. $q \leftarrow q$

Unfolding of q using clause 3

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

Unfolding of q using clause 3

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

Unfolding of q using clause 3

P_2: 3. $p \leftarrow \underline{q, r}$ 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

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

Literals in the Scope of Universal Quantification

- Local variables are the main source of inefficiency when dealing with negation

$$q(X_1, X_2) \leftarrow member(Y, X_1), \neg member(Y, X_2)$$

$$\Downarrow$$

$$\big(\neg q(x_1, x_2) \quad \leftrightarrow \quad \exists \overline{z} \; (\; x_1 \approx z_1 \wedge x_2 \approx z_2 \wedge \\ \forall y \; [\; \neg member(y, z_1) \vee member(y, z_2) \;] \;) \; \big)^{\forall}$$

Modes and Well-Moded Programs

- 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

$$q(X_1, X_2) \leftarrow member(Y, X_1), \neg member(Y, X_2)$$

  - $member(Y, X_1)$: the first occurrence of $Y \Rightarrow$ the first argument of $member_{/2}$ is out
  - $\neg member(Y, X_2)$: the second occurrence of $Y \Rightarrow$ the first argument of $member_{/2}$ is `in`

Modes and Well-Moded Programs

A program P is *well-moded* iff for every clause $C \in P$

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

we have that

$$\mathtt{Var}(\overline{s}^i) \subseteq \bigcup_{j=0}^{i-1} \mathtt{Var}(\overline{t}^j)$$

The Basic Transformation Step

- 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

$$
\begin{array}{lll}
P_1: & 1. & q(X_1, X_2) \leftarrow \underline{member(Y, X_1)}, \neg member(Y, X_2) \\
 & 2. & member(E, [E|\_]) \leftarrow \\
 & 3. & member(E, [\_|L]) \leftarrow member(E, L)
\end{array}
$$

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

$$
\begin{array}{lll}
P_2: & 4. & q([Z_1|\_], Z_2) \leftarrow \neg member(Z_1, Z_2) \\
 & 5. & q([\_|Z_1], Z_2) \leftarrow \underline{member(W, Z_1), \neg member(W, Z_2)} \\
 & 2. & member(E, [E|\_]) \leftarrow \\
 & 3. & member(E, [\_|L]) \leftarrow member(E, L)
\end{array}
$$

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

$$
\begin{array}{lll}
P_3: & 4. & q([Z_1|\_], Z_2) \leftarrow \neg member(Z_1, Z_2) \\
 & 6. & q([\_|Z_1], Z_2) \leftarrow q(Z_1, Z_2) \\
 & 2. & member(E, [E|\_]) \leftarrow \\
 & 3. & member(E, [\_|L]) \leftarrow member(E, L)
\end{array}
$$

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

  $P_0 : \quad p \leftarrow \underline{q} \quad q \leftarrow r \quad r \leftarrow$

  Unfolding of q using its definition in P_0

  $P_1 : \quad p \leftarrow r \quad q \leftarrow \underline{r} \quad r \leftarrow$

  Folding of r the 1^{st} clause)

  $P_2 : \quad p \leftarrow r \quad q \leftarrow \underline{p} \quad r \leftarrow$

  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