Inverse Narrowing
for the Inductive Inference
of Functional Logic Programs

J. Hernandez-Orallo & M.J. Ramirez-Quintana

DSIC, Universitat Politècnica de València
Camí de Vera s/n, 46022 València, Spain.
Email: {jorallo, mramirez}@dsic.upv.es

IV Advanced Seminar on Declarative Programming / Reunión CPD’98
Valencia, Monday, June 15th

---

1 Work partially supported by CICYT under grant TIC 95-0433-C03-03.
Inductive Synthesis of Declarative Programs

Sample

\begin{align*}
\text{even}(0) \\
\text{even}(ssssss0) \\
\text{even}(ss0) \\
\neg\text{even}(s0) \\
\ldots
\end{align*}

Induction

\begin{align*}
\text{Induced} \\
\text{Program}
\end{align*}

Induction

\begin{align*}
\text{even}(0). \\
\text{even}(ssX) :- \text{even}(X).
\end{align*}

Validation

Logic Programming \quad \cap \quad \text{Machine Learning} \quad \Rightarrow \quad \text{Inductive Logic Programming (ILP)}

Applications:
- Established: Scientific Theory Formation, Data Mining, Specific Industrial Applications (Traffic Control).
- Promising: NLP, Modelling, Program Synthesis.
Specific Trend

LP Extensions & Combinations
CLP, AILP, Planification (EvC, SitC), RL
Extend ILP to other Declarative Paradigms

• Functional Programming
  Based on rewriting (e.g. Haskell, ML). → Olson95

• Functional Logic Programming
  Based on residuation (e.g. Escher) → FlaGirLlo98
  Based on narrowing (e.g. Curry) → *

• Higher-Order Frameworks
  Different rewriting or unification mechanisms.

_advantages:
• Background knowledge can be richer: schemata, biases…
• More expressive power → More compact theories
• The relation between deduction & induction can be more deeply considered (incompleteness, information-gain…)

_drawbacks:
• Similar efforts and techniques could be scattered among different representation mechanisms.
• In general*, the deduction methods are less efficient or less well-established than resolution.
Language:

Inductive Functional Logic Programs:

Conditional Rewriting Systems (CRT)
with rules of the form:
\[ l = r \iff e_1, \ldots, e_n \text{ with } n \geq 0 \]

+ \[ \varepsilon \text{-unification} \]

Subsumes LP and Functional Programming.

Narrowing:

- *Sound and complete* \( \varepsilon \)-unification method.
- *More expressive power in comparison to functional languages.*
- *Better operational behavior in comparison to logic languages.*
- *Migration to HOL will be easier than directly from ILP.*
Narrowing

This work $\rightarrow$ unconditional case:
- unrestricted (ordinary) narrowing:

Narrowing ($\leftarrow\rightarrow$)  \textit{pattern-matching} $\rightarrow$ \textit{unification}

$t$ ‘narrows’ into $t'$ ($t \leftarrow\rightarrow \theta t'$) using program $P$ iff
- $u \in O_{nv}(t)$,
- $l = r$ is a new variant of a rule from $P$,
- $\theta = \text{mgu}(t_{u} \mid l)$, and
- $t' = \theta(t[r]_{u})$.

\textbf{Example:} program $P_1 = \{ r_1: X+0=X. \text{ } r_2: X+sY=s(X+Y) \}$
\begin{align*}
\Leftarrow & \text{s0 + Z = ss0} \quad \text{u = lhs}_\varepsilon \text{, rule } r_2, \text{ } \theta = \{ X/s0, Z/sY \} \\
\Leftarrow & \text{s(s0 + Y) = ss0} \quad \text{u = lhs}_1 \text{, rule } r_1, \text{ } \theta = \{ X'/s0, Y/0 \} \\
\Leftarrow & \text{ss0 = ss0} \quad \text{X' = X'' } \text{ } \theta = \{ X''/ss0 \} \\
\Leftarrow & \text{true} \quad \text{SOL: } \{ \text{ Z/s0 } \}
\end{align*}

Unrestricted narrowing is sound and complete wrt. canonical programs.
For this work, we shall only induce canonical programs.
Inductive framework

• Evidence $E$:
  - Positive sample $E^+$
  - Negative sample $E^-$

• Background Knowledge Theory $B$:

A program $P$ is a solution to the inductive (or learning) problem generated from $E$ iff:

\[
B \cup P \models E^+ \quad (posterior\ sufficiency\ or\ completeness)
\]
\[
B \cup P \not\models E^- \quad (posterior\ satisfiability\ or\ consistency)
\]

Additionally, it is usually supposed
\[
B \not\models E^+ \quad (prior\ necessity)
\]
\[
B \not\models E^- \quad (prior\ satisfiability)
\]

Also, to approach abduction in an ILP framework:

$P \not\models E^+$ and only facts can be in $P$. 
Hypotheses Selection

For every $E$ there are infinite many solutions

Criteria for generation and selection:

- The shortest one (the MDL principle)
  - problems $\rightarrow$ Non-computable.
  - It can leave extensional parts.

- The most specific one (Plotkin’s $\lgg$):
  - problem $\rightarrow$ $P = E^+$ is a solution.

- The least specific one:
  - problem $\rightarrow$ $P = T - E^-$ is a solution.

- The most efficient one:
  - problem $\rightarrow$ $P = E^+$ is usually the most efficient.

- The most ‘coherent’ one. No part must be left in an extensional way, i.e., all the data must be produced by the same ‘main set of rules’.
  - problem $\rightarrow$ it must be combined with other criteria to avoid ‘fantastic’ inductions.
Example 1

- Background Knowledge Theory $B$:
  $s(X) < s(Y) = X < Y$
  $0 < s(Y) = \text{true}$
  $X < 0 = \text{false}$

- Evidence $E$:
  
  | $E_1^+$ | $0 + 0 = 0$ |
  | $E_2^+$ | $s0 + s0 = ss0$ |
  | $E_3^+$ | $0 + s0 = s0$ |
  | $E_4^+$ | $s(s0 + s0) = sss0$ |
  | $E_5^+$ | $ss0 + s0 = ss0$ |
  | $E_6^+$ | $0 + s0 = s0$ |

- Possible solutions:

  $P_1 = E^+$  
  
  Very specific

  $P_2 = \{X+0 = X, 0+X = X, sX+s0 = ssX\}$

  Short and Coherent

  $P_3 = \{X+0 = X, X+sY = s(X+Y)\}$

  Short

  $P_4 = \{X+0 = X, X+s0 = sX\}$

  Coherent

  $P_5 = \{X+Y = X \Leftarrow Y = 0, X+sY = sX \Leftarrow Y = 0\}$

  Efficient

  $P_6 = \{X+0 = X, X+Y = Y+X \Leftarrow X<Y, sX+sY = ss(X+Y)\}$

  $P_7 = \{X+0 = X, 0+X = X, sX+sY = ss(X+Y)\}$

  $P_8 = \{X+0 = X, 0+X = X, sX+sY = s(X+sY)\}$

  ...
General heuristics

No unified criterion for all the applications. There is no such thing as “the right hypothesis”

The stop-criteria should be parametrised. ↓

The search is guided by an optimality factor weighting some selected criteria.

\[
Opt(P) = \alpha \cdot \text{LenF}(P) + \beta \cdot \text{CovF}^+(P) + \gamma \cdot \text{ConF}(P) + \delta \cdot \ldots
\]

 Advantages:
• The same generic algorithm can be used for different applications.
• Any information about the supposed ‘true’ hypothesis can help to select the different criteria and speed up the search.

 Drawbacks:
• The search cannot be fully optimised (it is difficult to prune if the search heuristics are variable)
• Hard-completeness results are difficult.
Used Criteria

\[
\text{Opt}(P) = \text{LenF}(P) + \text{CovF}^+(P) + \text{ConF}(P)
\]

\textbf{LenF} = syntactical length of rhs.

Different ‘weight’: 
- 1 : constants and functors
- 0.5 : variables

Example: \(\text{Weight}(\{ \text{ssX} + \text{sX} \Rightarrow \text{s(ssX} + 0) \} ) = 5.5\)

\[
\text{LenF}(P) = -\sum_{e \in P} \log_2 \text{Weight}(e)
\]

\[
\text{CovF}^+(P) = \frac{\text{card}(e \in E^+ : P \models e)}{\text{card}(E^+)}
\]

It allows approximate learning.

\[
\text{ConF}(P) = 1 \text{ if } P \text{ has only an equation, otherwise}
\]

\[
\text{ConF}(P) = 1 - \max(\text{card}(e \in E^+ : P_i \subset P \land P_i \models e)) / \text{card}(E^+)
\]

Example:

\(P_1 = \{ r_1, r_2, r_3 \}\)  Suppose \(\{ r_3 \}\) covers \(e_5\)

and \(\{ r_1, r_2 \}\) covers \(e_1, e_2, e_3, e_4\)

\(\text{ConF}(P_1) = 1/5 \Rightarrow e_5\) is clearly an exception.

Different Stop Criteria for different applications:

- If \(\text{CovF}^+ = 1\) and \(\text{ConF} > dc\) (desired consilience) \(\Rightarrow\) Appropriate for \textit{program synthesis} (perfect data and coherent programs)
- If \(dc = 0\) and \(\text{CovF}^+ = 1\) the criterion \(\approx\) MDL principle \(\Rightarrow\) No information at all about the source.
- If \(dc = 0.5\) and \(\text{CovF}^+ = 0.8\), learning a consilient theory in the presence of errors (with known error ratio = 0.2).
Main mechanisms

Inverse of matching/substitution $\rightarrow$ generalisation
Inverse of narrowing $\rightarrow$ “inverse narrowing”

Def. 1. Restricted Generalisation (RG)

Given an equation $e \equiv \{ t = s \}$, the equation $t' = s'$ is a restricted generalisation of $e$ iff it is a generalisation, i.e.

$$\exists \theta : t' \theta = t \land s' \theta = s$$

and it does not include fresh variables in the rhs.

$$\forall x (x \in \text{Var}(s') \Rightarrow x \in \text{Var}(t'))$$

Def. 2. Consistent Restricted Generalisation (CRG)

The equation $e = \{ l_1 = r_1 \}$ is a CRG w.r.t. $E^+$ and $E^-$ and the theory $T = B \cup P$ iff $e$ is a RG for some equation of $E^+$ and there does not exist a narrowing chain $(s \xrightarrow{T \cup e} t)$ such that:

$s=t \in E^-$. (consistency wrt. $E^-$)

Example: (following Example 1)

Clause $\{ X' + 0 = X' \}$ is a CRG of $E^+_{1}$

Clause $\{ X + s0 = sX \}$ is a CRG of $E^+_{2}, E^+_{3}, (E^+_{4}), E^+_{5}$
Inverse Narrowing

Def.3 Inverse Narrowing ($\leftarrow\rightarrow$)

t ‘conversely narrows’ into $t'$ ($t \leftarrow\rightarrow \theta \ t'$) iff

- $u \in O(t)$,
- $l = r$ is a new variant of a rule from $P$,
- $\theta = mgu(t|u, r)$, and
- $t' = \theta(t[l]_u)$.

Reversed Narrowing + CRG = Inverse Narrowing.

Example:

From the equation $e_a = \{X + s0 = sX\}$ select $t = sX$
We find a new variant $\{X' + 0 = X'\}$ from $P$.

Two occurrences:  $u_1 = 1$ gives $t'_1 = s(X + 0)$
$u_2 = \varepsilon$ gives $t'_2 = sX + 0$

giving two equations

$e_{a,1} = \{X + s0 = s(X+0)\}$
$e_{a,2} = \{X + s0 = sX+0\}$

It is obvious that both narrow into $e_a$ using $P$.
The same holds after CRG: $e'_{a,1} = \{X + sY = s(X+Y)\}$
Non-incremental Algorithm

Two main sets:

\( EH : \text{Set of equations, generated from all CRG of } E^+. \)

\( PH \subset \wp(EH) : \text{set of programs constructed from } EH. \)

Initially, \( PH = \{ \{ e \} : e \in EH \} \)

Programs are \textit{merged} using inverse narrowing followed by a CRG.

On each iteration, until all the data are ‘consiliated’:

- The two most optimal programs are selected, provided they cover most of the examples, and they have not been merged before.
- Inverse narrowing is made between all the possible occurrences using one equation of each program.
- The resulting programs which are consistent and canonical are added to \( PH. \) If not, they can be split.

Several parameters: \textit{min}, \textit{step}, \textit{inarcomb} are introduced to \textit{temporarily} prune the search tree.

Condition for using \( B: \) some example does not have any program which covers it with good optimality.
Example (non-incremental)

• Evidence $E$:

$E_1^+$ append([1,2],[3]) = [1,2,3]  \quad (E_1^-) \quad append([3],[4])=[4,3]

$E_2^+$ append([c],[a])=[c,a]  \quad (E_2^-) \quad append([1,2],[])=[1]

$E_3^+$ append([],[4])=[4]  \quad (E_3^-) \quad append([1,2,3],[4])=[1,2,3,4,5]

$E_4^+$ append([a,b],[])=[a,b]  \quad (E_4^-) \quad append([],[a,b])=[b,a]

$E_5^+$ append([a,b,c],[d,e])=[a,b,c,d,e]

• From each example, two ($min=2$) CRG’s are generated with the best optimality:

$CRG(E_1^+) = \{ e_1: append(.(X,.(Y,[])), Z) = .(X,.(Y,Z)),
\quad e_2: append(.(X,.(Y,Z)),.(W,Z)) = .(X,.(Y,.(W,Z))) \} $

$CRG(E_2^+) = \{ e_3: append(.(X,[]),Y)=.(X,Y),
\quad e_4: append(.(X,Y),.(Z,Y)) = .(X,.(Z,Y)) \} $

$CRG(E_3^+) = \{ e_5: append([],X)=X,
\quad e_6: append(X,.(Y,X)) = .(Y,X) \} $

$CRG(E_4^+) = \{ e_7: append(X,[])=X,
\quad e_8: append(.(X,.(Y,Z)),Z)=.X,.(Y,Z)) \} $

$CRG(E_5^+) = \{ e_9: append(.(Y,.(Z,.(W,V))),X)=.Y,.(Z,.(W,X))),
\quad e_{10}: append(.(Y,.((Z,.(W,[]))),X)=.Y,.(Z,.(W,X))) \} $

Constructed $EH$ and $PH$, the best solution is \{$e_1, e_3, e_5, e_9$\} covering $E^+$ (with dreadful optimality and no consilience at all).
Example (cont)

• 1st Iteration. 1st Inverse Narrowing Combination.

There is no pair of programs covering 5 or 4 examples. Thus, from those programs covering 3 examples, the most optimal ones are:

\[ P_1 = \{ \text{append}(X,.(Y,[]),Z) = .(X,.(Y,Z)) \} \text{ covering } E_1^+, E_4^+ \]
\[ P_2 = \{ \text{append}([],X) = X \} \text{ covering } E_3^+ \]

giving 3 consistent programs:

\[ P_a = \{ \text{append}(.(X,.(Y,W)), Z) = .(\text{append}(W,X),.(Y,Z)), \text{append}([],X)=X \} \]
\[ P_b = \{ \text{append}(.(X,.(Y,W)), Z) = .(X,.(\text{append}(W,Y),Z)), \text{append}([],X)=X \} \]
\[ P_c = \{ \text{append}(.(X,.(Y,W)), Z) = .(X,.(Y,\text{append}(W,Z))), \text{append}([],X)=X \} \]

Added to \( PH \). The best solution is the same as before.
Example (cont)

- 2\textsuperscript{nd} Iteration. 2\textsuperscript{nd} Inverse Narrowing Combination.

Now, we find two programs covering 4 examples:

\[ P'_1 = P_a = \{ e_{1,1}: \text{append}(.(X,.(Y,W)), Z) = .(\text{append}(W,X),.(Y,Z)), e_{1,2}: \text{append}([],X) = X \} \text{ covering } E_1^+, E_3^+, E_4^+ \]

\[ P'_2 = \{ e_{2,1}: \text{append}(.(X,[]), Y) = .(X,Y) \} \text{ covering } E_2^+ \]

Select the two rules with the highest optimality: \( e_{1,2} \) and \( e_{2,1} \).

After inverse narrowing and CRG, most of them are inconsistent. After ‘splitting’, only one of them results consistent and confluent (\( e_{1,1} \) is removed):

\[ P_d = \{ \text{append(}.(X,Z),Y) = .(X, \text{append}(Z,Y)), \text{append}([],X) = X \} \]

which covers \( E^+ \) and has good optimality.

Best Solution: \( P_d \) with consilience > 0.5, the stop criterion.

\textit{The example shows that if optimality is not used heuristically, the method is not feasible in practice.}
Incremental Algorithm

More interactive (the user can stop the sample).

For each new example which is being presented:

- **If it is a positive example**: $E^+_n$, check for every program $P_i \in PH$:
  1. **HIT** ($P_i \models E^+_n$): Just recompute the optimalities.
  2. **NOT COVERED** ($P_i \not\models E^+_n \wedge \text{lhs}(E^+_n)$ is $\downarrow$): = HIT
  3. **ANOMALY**: Remove all non confluent and inconsistent $P_i$ from $PH$ and prune $EH$.
   and we generate all the CRG’s of $E^+_n$ in $EH$ and extend $PH$ with all the new unary programs.

- **If it is a negative example**: $E^-_n$, we check the consistency for every program $P_i \in PH$ and we act as in either the HIT or as in the ANOMALY cases.

In any case, the iteration can be ‘reactivated’ until the best solution complies with the stop-criterion (or an iteration limit is exhausted).

The consilience criterion avoids extensional ‘patches’ for the NOT-COVERED case.
Example (incremental & BK)

Induce the power function from the product function:

\[ B = \{0 \times X = 0, \; sX \times Y = X \times Y + Y, \; X + 0 = X, \; X + sY = s(X + Y)\} \]
\[ BF = \{ \times \} // Only \text{ use } \times \text{ and the functors which appear in } E. \]

Example of 9 steps of an interactive session:

1. The first example \( E_1^+ = \{ ss0 \uparrow ss0 = ssss0 \} \) is processed. The first \( EH \) could be enormous and must be pruned.
2. The second example \( E_1^- = \{ ss0 \uparrow sss0 = sssssss0 \} \) does not make any program inconsistent.
3. The third example \( E_2^+ = \{ sss0 \uparrow ss0 = sssssss0 \} \) is a NOT COVERED case and generates new equations, like \( \{ X \uparrow Y = sssssX \} \) or \( \{ sX \uparrow X = sssssssX \} \).

   Poor optimality \( \Rightarrow \) inverse narrowing between \( \{ E_2^+ \} \) and \( B \).

   Program \( P_a = \{ X \uparrow ss0 = X \times X \} \) is generated covering all \( E^+ \) and with good optimality over other solutions.

   It is offered to the user. The user deems it to be too hasty.

4. Example \( E_2^- = \{ sss0 \uparrow sss0 = sssssss0 \} \) prunes some programs but \( P_a \) is still the best solution.
5. Example $E_3^+ = \{ sss0 \uparrow s0 = sss0 \}$ is NOT COVERED by all programs. New CRG’s are generated like $\{ X \uparrow s0 = X \}$ and $\{ ssX \uparrow X = ssX \}$. Until some limit of iterations, the algorithm stops because it does not find a consilient program. The best one is $P_5 = \{ X \uparrow s0 = X, X \uparrow ss0 = X \times X \}$.

6. Example $E_3^- = \{ ss0 \uparrow sss0 = sss0 \}$ eliminates some uninteresting programs.

7. Example $E_4^+ = \{ 0 \uparrow sss0 = 0 \}$ is NOT COVERED by all programs. New CRG’s are generated like $\{ 0 \uparrow X = 0 \}$.

8. Example $E_4^- = \{ sss0 \uparrow ss0 = ssss0 \}$ eliminates some uninteresting programs.

9. Example $E_5^+ = \{ ss0 \uparrow 0 = s0 \}$ is NOT COVERED by all programs. New CRG’s are generated like $\{ X \uparrow 0 = s0 \}$ or $\{ ss0 \uparrow 0 = s0 \}$. The first one is combined with $P_5$ which contained $\{ X \uparrow ss0 = X \times X \}$. This gives equations like $\{ X \uparrow sY = (X \uparrow Y) \times X \}, \{ X \uparrow sY = X \times (X \uparrow Y) \}$ and $\{ X \uparrow sY = (X \uparrow X) \times Y \}$. Some new programs are constructed using them. One has very good optimality → the algorithm offers it to the user…

Solution guessed at step 9:

\[
\begin{align*}
X \uparrow sY &= (X \uparrow Y) \times X \\
X \uparrow s0 &= s0
\end{align*}
\]

The user now considers it’s time to stop.

*Obviously, any future example can be NOT COVERED or even can make it inconsistent.*
Conclusions and Future Work

General framework for the induction of functional logic programs.

Two basic operators are introduced:
• Consistent Restricted Generalisation
• Inverse Narrowing

The selection criterion is parametrisable.

Adaptation to the incremental case is immediate due to the notion of consilience (a good solution is sought earlier than the MDL principle suggests).

Current work:
• conditional extension: based on balanced reinforcement to avoid exceptions as conditions.
• comparison with other ILP systems.

Future work:
• theoretical results on ‘completeness’ and complexity.
• study of different narrowing techniques (especially needed narrowing) to possibly integrate with Curry.
• higher-order logic.