Program Transformation and Its Applications to Software Synthesis and Verification

Maurizio Proietti

Joint work with:

Fabio Fioravanti (Univ. D'Annunzio, Pescara, Italy), Alberto Pettorossi (Univ. Tor Vergata, Rome, Italy), Valerio Senni (Univ. Tor Vergata, Rome, Italy)

IASI – 1 Ottobre 2010
Correctness of Software

Safety and business-critical applications need dependable software.

Traditional validation and testing methodologies are not always adequate: they do not guarantee that software artifacts meet their specifications in all cases.

Logic-based methods aim at mechanically proving the correctness of software wrt formal specifications.
Overview

• **Program Verification**: proof of program properties

• **Program Synthesis**: automatic derivation of programs from first order logic specifications

• **Program Transformation**: automatic improvement of programs
A Bit of History

**Automated Theorem Proving**
- Leibniz [1666] Calculus ratiocinator, Lingua characteristica universalis
- Frege [1879] First Order Logic
- Hilbert's program [early '900] Formalization of mathematics, prove the consistency of Arithmetics by finitist methods, the decision problem for FOL
- Presburger [1929] Decision procedure for the FO theory of addition

- Gödel [1931], Church-Turing [1936-7] Undecidability of Arithmetics and FOL

**Decidable theories**
- Tarski [1951] First order theory of real numbers is decidable
- Description logics [1990's] Ontologies, Semantic Web

**General methods (based on application strategies)**
- Robinson [1965] Resolution
- Kowalski [1974] Logic Programming
- Jaffar-Lassez [1987] Constraint Logic Programming
- CADE 2009: The Vampire resolution-based theorem prover solves 181/200 problems of the annual competition
A Bit of History

Verification

Turing [1936] Undecidability of the Halting Problem

Floyd [1967] Inductive assertions for flowchart programs
Hoare [1969] FOL axiomatization of the correctness of ALGOL programs
Pnueli [1977] Temporal logics for the verification of concurrent programs
Clarke-Emerson [1980] Model Checking

Synthesis

Waldinger [1969] Using resolution for synthesis of LISP programs
Clark-Hogger [1977-81] Synthesis of logic programs
Clarke-Emerson [1981] Synthesis of concurrent programs

Transformation

[1960's] Equivalence of flowchart schemas
Paterson-Hewitt [1970] Recursive schemas are more expressive than flowcharts
Program Transformation
Rule-based Program Transformation

Initial program $P_0 \rightarrow P_1 \rightarrow \cdots \rightarrow P_n$ Final program

where '$\rightarrow$' is an application of a transformation rule.

- Program transformation separates the correctness and the efficiency concerns during program development.
- $P_0$ can easily be proved correct wrt a given specification and semantics $M$.
- Each rule application preserves the semantics:
  $M(P_0) = M(P_1) = \cdots = M(P_n)$
- The application of the rules is guided by a strategy which guarantees that $P_n$ is more efficient than $P_0$. 
An Example: Approximate Matching

Classical matching: \[ S: \quad L \quad P \quad R \]

Approximate matching:

Given two lists of integers \( P = [x_1, \ldots, x_n] \) and \( S \), and an integer \( K \), \( \text{match}(P,S,K) \) iff there exists a subsequence \( Q = [y_1, \ldots, y_n] \) of \( S \) s.t., for \( i = 1, \ldots, n \), \( |x_i - y_i| \leq K \).

\[
\begin{align*}
P &: 2 \quad 0 \\
S &: 5 \quad 0 \quad 4 \quad 1 \quad 4 \quad 3 \quad 3 \quad 0 \quad 3 \quad 6 \quad 5 \quad 1 \quad 4 \\
Q &
\end{align*}
\]

\( \text{max-diff}(P,Q) \leq 2 \)

Constraint logic program for approximate matching:

\[
\begin{align*}
\text{Match:} \quad & \quad S = L :: Q :: R \\
\text{match}(P,S,K) :- \text{append}(L,Q,A), \text{append}(A,R,S), \text{max-diff}(P,Q,K). \\
& \quad \text{append}([],S,S). \\
& \quad \text{append}([X|S],T,[X|U]) :- \text{append}(S,T,U). \\
& \quad \text{max-diff}([],[],K). \\
& \quad \text{max-diff}([X|S],[Y|T],K) :- |X-Y| \leq K, \text{max-diff}(S,T,K).
\end{align*}
\]
Approximate Matching (2)

• Suppose that we want to use the Match program for queries of the form:
  \[ \text{match([2, 0], S, 2)} \]

• Add a new clause to Match:
  \[ \text{C: } \text{sp_match}(S) :- \text{match([2, 0], S, 2)} \]

• \{C\} ∪ Match has a **generate and test** behaviour: subsequences Q of S are generated and then the test \text{diff([2,0],Q,2)} is performed.

• Derive an efficient program by applying a sequence of transformation **rules** according to a transformation **strategy**:
  \[ \{C\} ∪ \text{Match} \rightarrow \cdots \rightarrow \text{Sp_Match} \]
Specialized Approximate Matching

Sp_Match:

\[
\text{sp}\_\text{match}(S) :- p1(S).
\]

\[
p1([X|S]) :- 0 \leq X \leq 4, \ p2(S).
\]

\[
p1([X|S]) :- (X < 0 \lor X > 4), \ p1(S).
\]

\[
p2([X|S]) :- -2 \leq X \leq 2, \ p3(S).
\]

\[
p2([X|S]) :- 2 < X \leq 4, \ p2(S).
\]

\[
p2([X|S]) :- (X < -2 \lor X > 4), \ p1(S).
\]

\[
p3(S).
\]

Sp_Match has a \( O(|S|) \) running time for an input sequence \( S \) and corresponds to a deterministic finite automaton.
Correctness of the Transformation Rules

- The transformation rules (e.g., unfolding, folding, constraint replacement, clause replacement) replace a set of clauses by an equivalent one.

\[
P_0: \begin{align*}
p &\iff r. \\ q &\iff r. \\ r &. \\
\end{align*}
\]
\[M(P_0) = \{p, q, r\} \]

\[
P_1: \begin{align*}
p &\iff r. \\ q &. \\ r &. \\
\end{align*}
\]
\[M(P_1) = \{p, q, r\} \]

- In general, replacement does not preserve the least model semantics:

\[
P_0: \begin{align*}
p &\iff q. \\ q &\iff r. \\ r &. \\
\end{align*}
\]
\[M(P_0) = \{p, q, r\} \]

\[
P_1: \begin{align*}
p &\iff p. \\ q &. \\ r &. \\
\end{align*}
\]
\[M(P_1) = \{q, r\} \]
Correctness of the Transformation Rules

Replacement of equivalent formulas is partially correct (or sound):

\[ M(P_0) \supseteq M(P_1) \supseteq \cdots \supseteq M(P_n) \]

if (i) P is a definite program (no negative literals in the premises)
(ii) \( M(P) \) is the least model of P

Correctness Issues

• Sufficient conditions for total correctness:
  \[ M(P_0) = M(P_1) = \cdots = M(P_n) \]

• General programs (negative literals in the premises)

• Various semantics: least model, perfect model, stable model, ...
Transformation Strategies

Transformation strategies are directed by **syntactical** features of programs

- Avoiding multiple visits of data structures and repeated computations by **eliminating multiple occurrences of variables** from bodies of clauses
- Avoiding the computation of unnecessary values by **eliminating existential variables** (variables occurring in the body and not in the head)
- Reducing nondeterminism by **avoiding multiple clauses** for the same predicate definition
- Specializing programs to the context of use by pre-computing **partially instantiated literals**
Approximate String Matching Revisited

Initial program \( \{C\} \cup \text{Match} \):

\[
\text{sp\_match}(S) :- \text{match}([2, 0], S, 2). \quad \text{(Partially instantiated literal)}
\]

\[
\text{match}(P,S,K) :- \text{append}(L,Q,A), \text{append}(A,R,S), \text{diff}(P,Q,K). \quad \text{(Existential and multiple occurrences of list variables)}
\]

Final program Sp\_Match:

\[
\text{sp\_match}(S) :- \text{p1}(S).
\]

\[
\text{p1}([X|S]) :- 0 \leq X \leq 4, \text{p2}(S).
\]

\[
\text{p1}([X|S]) :- (X<0 \lor X>4), \text{p1}(S).
\]

\[
\text{p2}([X|S]) :- -2 \leq X \leq 2, \text{p3}(S).
\]

\[
\text{p2}([X|S]) :- 2<X \leq 4, \text{p2}(S).
\]

\[
\text{p2}([X|S]) :- (X<-2 \lor X>4), \text{p1}(S).
\]

\[
\text{p3}(S).
\]

No multiple occurrence of list variables.
No existential variables.
Clauses are mutually exclusive.
Transformation strategies face undecidability limitations, e.g., the problem of checking whether or not from a given program we can derive a program without existential variables is undecidable.

Issues about Strategies

• In general, transformation strategies are based on heuristics and are evaluated in an experimental way

• For specific classes of programs the transformation strategies can be proved successful in terms of transformation times and speed-up.
Experimental Evaluation of Strategies

- Many transformation rules and strategies are implemented in the MAP system: http://www.iasi.cnr.it/~proietti/system.html

- Experimental results on matching and parsing problems

<table>
<thead>
<tr>
<th>Program</th>
<th>Query</th>
<th>Transformation Time (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>String Matching</td>
<td>match([aab],S)</td>
<td>0.07</td>
<td>6.8 x 10^3</td>
</tr>
<tr>
<td>Multi Matching</td>
<td>mmatch([[aaa],[aab]],S,N)</td>
<td>0.28</td>
<td>6.8 x 10^3</td>
</tr>
<tr>
<td>Reg. Expr. Matching</td>
<td>re_match(aa*b,S)</td>
<td>0.21</td>
<td>3.0 x 10^6</td>
</tr>
<tr>
<td>Context Free Parsing</td>
<td>parse(g,[s],W)</td>
<td>1.62</td>
<td>87.1</td>
</tr>
<tr>
<td>Approximate Matching</td>
<td>match([2,0,4], S, 2)</td>
<td>1.89</td>
<td>46</td>
</tr>
<tr>
<td>Approx. Multi Matching</td>
<td>mmatch([[1,1],[1,2]], S, 1)</td>
<td>2.11</td>
<td>45</td>
</tr>
</tbody>
</table>
Program Synthesis by Transformation
The Transformational Synthesis Method

**Program Synthesis:** Given a logic program P and a first order formula $\phi[X]$, derive a logic program Q defining a predicate $r(X)$ such that, for all ground terms $t$:

$$M(P) \models \phi[t] \text{ iff } M(Q) \models r(t)$$

Maximum of a nonempty list:

P:

\[
\text{member}(X, [Y|L]) :- X=Y. \\
\text{member}(X, [Y|L]) :- \text{member}(X,L).
\]

$\phi[L,M]$: \( \text{member}(M,L) \land \forall X (\text{member}(X,L) \rightarrow X \leq M) \)

Clause form of $\phi[L,M]$:

CF:

\[
\text{r}(L,M) :- \text{member}(M,L), \neg \text{greater}(L,M). \\
\text{greater}(L,M) :- \text{member}(X,L), X > M.
\]

P $\cup$ CF is correct: For all $L$, $M$, $M(P) \models \phi[L,M]$ iff $M(P \cup CF) \models r(L,M)$

… but inefficient: generate an element $M$ in $L$ and test $M$ is an upper bound \[ O(|L|^2) \]
The Transformational Synthesis Method (2)

Derive an efficient program by (i) eliminating multiple and existential variables and (ii) eliminating negation.

**CF:**

\[ r(L,M) :\text{:-} \text{member}(M,L), \neg \text{greater}(L,M). \]
\[ \text{greater}(L,M) :\text{:-} \text{member}(X,L), X > M. \]

Elimination of multiple and existential variables
Elimination of negation

**Q:**

\[ r([X|L],M) :\text{:-} s(X,L,M). \]
\[ s(X,[\ ],M) :\text{:-} M=A. \]
\[ s(X,[Y|L],M) :\text{:-} Y \leq X, s(X,L,M). \]
\[ s(X,[Y|L],M) :\text{:-} X \leq Y, s(Y,L,M). \]

**Q is correct:** For all \( L, M, \) \( M(P) \models \varphi[L,M] \) iff \( M(Q) \models r(L,M) \)

…and **efficient:** while visiting the list, keep the maximum so far \( [ O(|L|) ] \)
Issues in Transformational Synthesis

- Find suitable synthesis strategies based on heuristics (e.g., by composing several transformation strategies)
- Find specific classes of programs where the synthesis strategies can be proved successful in terms of synthesis times and speed-up.
Power of Transformational Synthesis

- **Weak monadic second order theory of 1 successor (WS1S) [Buchi ’60]**

\[
\begin{align*}
  n & ::= N \mid 0 \mid \text{succ}(n) \\
  \varphi & ::= n_1 > n_2 \mid n_1 = n_2 \mid n \in S \mid S_1 = S_2 \mid \neg \varphi \mid \varphi_1 \land \varphi_2 \mid \exists N \varphi \mid \exists S \varphi
\end{align*}
\]

where \(N\) is a variable ranging over natural numbers and \(S\) is a variable ranging over finite sets of natural numbers.

\[
\begin{align*}
  d^n & \prec \prec 2^n
\end{align*}
\]

- **WS1S is decidable in** \(2^{2^n}\) **time complexity, for some** \(d>0\).

- For every WS1S formula the transformational method synthesizes a program with **linear time complexity**.

\[
\begin{align*}
  d^n & \prec \prec 2^n
\end{align*}
\]

- The transformation strategy has **\(2^{2^n}\)** worst case time complexity.
Verification of Program Properties by Program Transformation
Theorem Proving by Program Transformation

Given a program $P$ and a closed first order formula $\varphi$, check whether or not

$$M(P) \models \varphi$$

The transformational proof method:

- **Closed formula** $\varphi$
- **Propositional program** defining a predicate $r$
- **Clause form** $\text{CF}$
- **Elimination of existential variables** $Q$

such that:

$$M(P) \models \varphi \iff M(Q) \models r$$

$M(Q) \models r$ is **decidable** in $O(|Q|)$ time
The Transformational Proof Method

Given a program

\[ P: \text{member}(X,[Y|L]) :- X=Y. \]
\[ \text{member}(X,[Y|L]) :- \text{member}(X,L). \]

and a closed first order formula ("every list of numbers has an upper bound")

\[ \varphi: \forall L (\text{list}(L) \rightarrow \exists U \forall X ( \text{member}(X,L) \rightarrow X \leq U )) \]

we want to prove:

\[ M(P) \vdash \varphi \]
Step 1. Clause-Form Transformation

\[ \varphi: \forall L \ (\text{list}(L) \rightarrow \exists U \ \forall X \ (\text{member}(X,L) \rightarrow X \leq U)) \]

\[ r \equiv \neg \exists L(\text{list}(L) \land \neg \exists U \neg \exists X (\text{member}(X,L) \land \neg X \leq U)) \]

Clause-Form:

CF: \[ r \equiv \neg a. \]

\[ a \equiv \text{list}(L), \neg \text{b}(L). \]

\[ b(L) \equiv \text{list}(L), \neg \text{c}(L,U). \]

\[ c(L,U) \equiv X > U, \text{list}(L), \text{member}(X,L). \]

\[ M(P) \models \varphi \quad \text{iff} \quad M(P \cup \text{CF}) \models r \]
Step 2. Elimination of Existential Variables

The strategy for the elimination of existential variables returns:

\[
Q: \quad r \iff \neg a.
\]

\[
a \iff d.
\]

\[
d \iff d.
\]

s.t. \quad M(P) \models \varphi \iff M(Q) \models r
Step 3. Computation of the Perfect Model

Q:  \[ r :\neg a. \]

S2: stratum 2
\[ a :\neg d. \]
\[ d :\neg d. \]

S1: stratum 1

1. Compute the least model of S1:

\[ T_{S1} \]

\[
\{ \} \quad \xrightarrow{T_{S1}} \quad \{ \}
\]

where \( T_{S1} \) is the immediate consequence operator (= one-step deduction)

\[
\{ \} \quad \text{is the least fixpoint of } T_{S1}, \text{ hence } M(S1) = \{ \}
\]

2. Transform S2 using M(S1):

\[ r. \]

\[ M(Q) = (M(\{r.\}) \cup M(S1)) = \{r\} \]

\[ \Rightarrow M(Q) \models r \quad \Rightarrow \quad M(P) \models \varphi \]
Power of the Proof Method

The transformational proof method is a decision procedure for WS1S.

**Experimental evaluation**

Examples run by the MAP transformation system
(www.iasi.cnr.it/~proietti/system.html)

Constraints are handled using the clp(R) module of SICStus Prolog
(implementing a variant of Fourier-Motzkin variable elimination)

<table>
<thead>
<tr>
<th>Property</th>
<th>Time (PM 1.73)</th>
</tr>
</thead>
<tbody>
<tr>
<td>∀L ∃U ∀Y ( member(Y,L) → Y ≤ U )</td>
<td>31 ms</td>
</tr>
<tr>
<td>∀L ∀Y ( (sumlist(L,Y) ∧ Y &gt; 0) → ∃X (member(X,L) → X &gt; 0) )</td>
<td>15 ms</td>
</tr>
<tr>
<td>∀L ∀M ∀N ( (ord(L) ∧ ord(M) ∧ sumzip(L,M,N)) → ord(N) )</td>
<td>16 ms</td>
</tr>
<tr>
<td>∀L ∀M ∀X ∀Y ( (leqlist(L,M) ∧ sumlist(L,X) ∧ sumlist(M,Y)) → X ≤ Y )</td>
<td>16 ms</td>
</tr>
</tbody>
</table>
Model Checking
Infinite State Systems by
Program Transformation
Verification of Infinite State Systems

• The behaviour of a concurrent system can be represented as a state transition system which generates infinite computation paths:

Properties are expressed in the Temporal Logic CTL, a propositional logic augmented with:

1. quantifiers over paths: $E$ (Exists), $A$ (All), and
2. temporal operators along paths: $F$ (Future, there exists a state in the path),
   $G$ (Globally, for all states of the path).

• If the set of states is finite, then CTL is decidable in polynomial time.

• CTL is undecidable for: (1) infinite state systems (e.g., integer variables) and (2) parameterized systems (families of finite-state systems).
The Bakery Protocol (Lamport)

Each process has: control state: \( s \in \{\text{think, wait, use}\} \) and counter: \( n \in \mathbb{N} \)

Process A

- **think**
  - \( n_A := n_B + 1 \)
  - If \( n_A < n_B \) or \( n_B = 0 \), go to **use**

Process B

- **think**
  - \( n_B := n_A + 1 \)
  - If \( n_B < n_A \) or \( n_A = 0 \), go to **use**

System: \( A \parallel B \)

- \( <\text{think},0,\text{think},0> \rightarrow <\text{wait},1,\text{think},0> \rightarrow <\text{wait},1,\text{wait},2> \rightarrow <\text{use},1,\text{wait},2> \rightarrow \ldots \)

Mutual Exclusion: \( <\text{think},0,\text{think},0> \models \neg \text{EF unsafe} \)

where, for all \( n_A, n_B \): \( <\text{use},\overline{n_A},\text{use},\overline{n_B}> \models \text{unsafe} \)
A system $S$ and the temporal logic are encoded by a constraint logic program $P_S$:

- the **transition relation** is encoded by a binary predicate $\text{trans}$:

  \[
  \begin{align*}
  \text{trans}(\langle \text{think},A,S,B\rangle,\langle \text{wait},A_1,S,B\rangle) & : - A_1= B+1. \\
  \text{trans}(\langle \text{wait},A,S,B\rangle,\langle \text{use},A,S,B\rangle) & : - A < B. \\
  \text{trans}(\langle \text{wait},A,S,B\rangle,\langle \text{use},A,S,B\rangle) & : - B = 0. \\
  \text{trans}(\langle \text{wait},A,S,B\rangle,\langle \text{use},A_1,S,B\rangle) & : - A_1 = 0.
  \end{align*}
  \]

- the **satisfaction relation** $\models$ is encoded by a binary predicate $\text{holds}$:

  \[
  \begin{align*}
  \text{holds}(\langle \text{use},A,\text{use},B\rangle, \text{unsafe}). \\
  \text{holds}(S, \text{not}(P)) & : - \neg \text{holds}(S, P). \\
  \text{holds}(S, \text{ef}(P)) & : - \text{holds}(S, P). \\
  \text{holds}(S, \text{ef}(P)) & : - \text{trans}(S,T), \text{holds}(T, \text{ef}(P)).
  \end{align*}
  \]

- the **property** to be verified is encoded by a predicate $\text{prop}$:

  \[
  \text{prop} : - \text{holds}(\langle \text{think},0,\text{think},0\rangle, \text{not}(\text{ef}(\text{unsafe}))) .
  \]
Protocol Verification Via Program Transformation

- The encoding is correct:
  \[ \langle \text{think}, 0, \text{think}, 0 \rangle \models \neg \text{EF unsafe} \iff M(P_S) \models \text{prop} \]

- Bottom-up construction of \( M(P_S) \) from facts does not terminate because \( M(P_S) \) is infinite. Top-down evaluation of \( P_S \) from \( \text{prop} \) does not terminate due to infinite computation paths.

- Transformation-based Verification Method:
  1) specialize \( P_S \) to the query \( \text{prop} \):

  \[ P_S \rightarrow \cdots \rightarrow Q \quad \text{s.t.} \quad M(P_S) \models \text{prop} \iff M(Q) \models \text{prop} \]

  2) keep only the clauses \( \text{dep}(\text{prop}, Q) \) on which the predicate \( \text{prop} \) syntactically depends:

  \[ \text{prop} \in M(Q) \iff \text{prop} \in M(\text{dep}(\text{prop}, Q)) \]

  3) construct bottom-up the model of \( \text{dep}(\text{prop}, Q) \).
<table>
<thead>
<tr>
<th>Protocol</th>
<th>Property</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bakery (mutual exclusion)</td>
<td>safety: ¬EF unsafe</td>
<td>0.05</td>
</tr>
<tr>
<td></td>
<td>liveness: AG(wait → AF use)</td>
<td>0.13</td>
</tr>
<tr>
<td>Ticket (mutual exclusion)</td>
<td>safety: ¬EF unsafe</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td>liveness: AG(wait → AF use)</td>
<td>0.10</td>
</tr>
<tr>
<td>Sleeping Barber</td>
<td>safety</td>
<td>0.03</td>
</tr>
<tr>
<td>Office Light Control</td>
<td>safety</td>
<td>0.10</td>
</tr>
<tr>
<td>Petri Net</td>
<td>safety</td>
<td>0.08</td>
</tr>
<tr>
<td>Berkeley RISC (cache coherence)</td>
<td>safety</td>
<td>0.07</td>
</tr>
<tr>
<td>Xerox Dragon (cache coherence)</td>
<td>safety</td>
<td>0.07</td>
</tr>
<tr>
<td>DEC Firefly (cache coherence)</td>
<td>safety</td>
<td>0.05</td>
</tr>
<tr>
<td>Illinois Univ. (cache coherence)</td>
<td>safety</td>
<td>0.06</td>
</tr>
<tr>
<td>MESI (cache coherence)</td>
<td>safety</td>
<td>0.07</td>
</tr>
<tr>
<td>MOESI (cache coherence)</td>
<td>safety</td>
<td>0.08</td>
</tr>
<tr>
<td>Synapse N+1 (cache coherence)</td>
<td>safety</td>
<td>0.04</td>
</tr>
<tr>
<td>IEEE Futurebus+ (cache coherence)</td>
<td>safety</td>
<td>0.22</td>
</tr>
</tbody>
</table>
Ongoing Work

- More expressive logics for path properties:
  - LTL / CTL* [PPS09]
  - $\omega$-regular languages [PPS10]

- Proving properties of logic programs on infinite structures
  (some work already in [PPS10] for infinite lists)

- Synthesis of reactive systems (e.g., protocols)

- Modelling and verification of Business Processes
  [joint work with Missikoff, Smith]