Software Verification and Synthesis by Program Transformation

Maurizio Proietti*

joint work with Fabio Fioravanti* and Alberto Pettorossi**

{fioravanti,adp,proietti}@iasi.rm.cnr.it

(*) IASI-CNR
Viale Manzoni 30
00185 Rome, Italy

(**) DISP - Universita' di Roma Tor Vergata
Via del Politecnico
00133 Rome, Italy

Goals of this work

- Establish a correspondence between
  Theorem Proving and ***Program Transformation***

- Exploit this correspondence for performing
  software verification
  by means of
  program transformers, ***program specializers***, ...

Theorem Proving vs Program Transformation

Truth-preserving ***Rules***:

- modus ponens
- generalization
- resolution

Semantics-preserving ***Rules*****:**

- unfold
- fold
- replacement

Strategies (***or Tactics***)
to construct proofs:

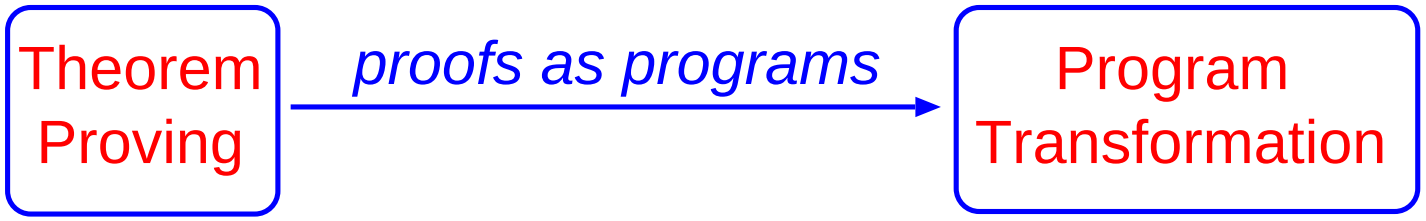
- depth-first
- breadth-first
- case analysis
- generalization
- induction

Strategies (***or Tactics***)
to derive "better" programs:

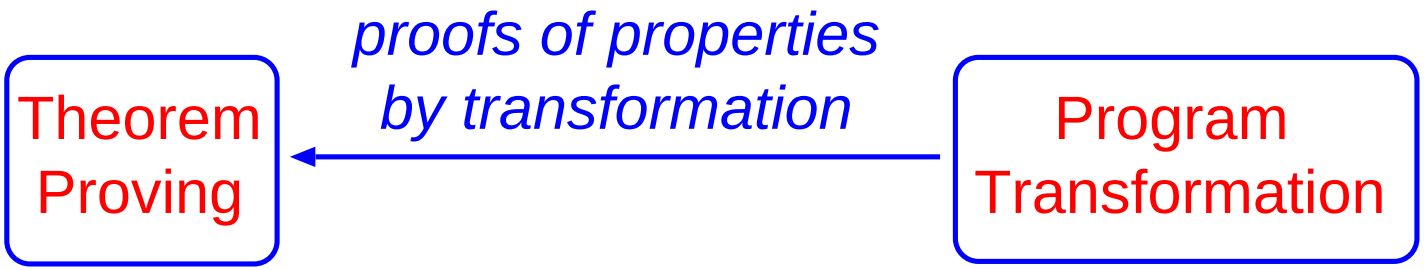
- specialization
- tupling
- composition (deforestation)
- generalization
- accumulation

Theorem Proving and Program Transformation

Can we establish a more formal correspondence?



Manna-Waldinger, Constable, Bundy



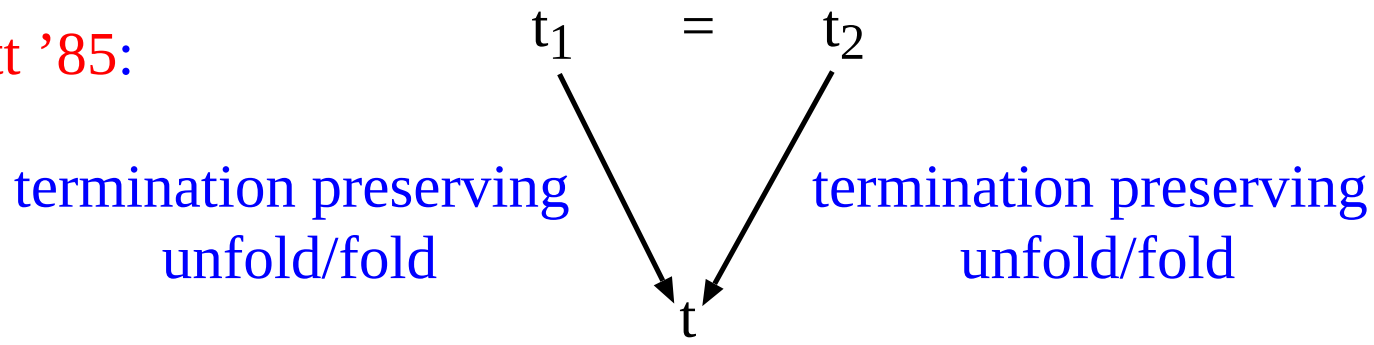
McCarthy, Kott, Courcelle, Pettorossi-Proietti, Roychoudury et al., Leuschel

Previous Work (Functional Programs)

- Recursion induction (Mc Carthy ’63): proving properties by rewriting + termination proofs

- Proofs by consistency or inductionless induction (Musser ’80, Huet-Hullot ’82, Jouannaud-Kounalis ’86): finding minimal counterexamples by rewriting and well-founded orderings

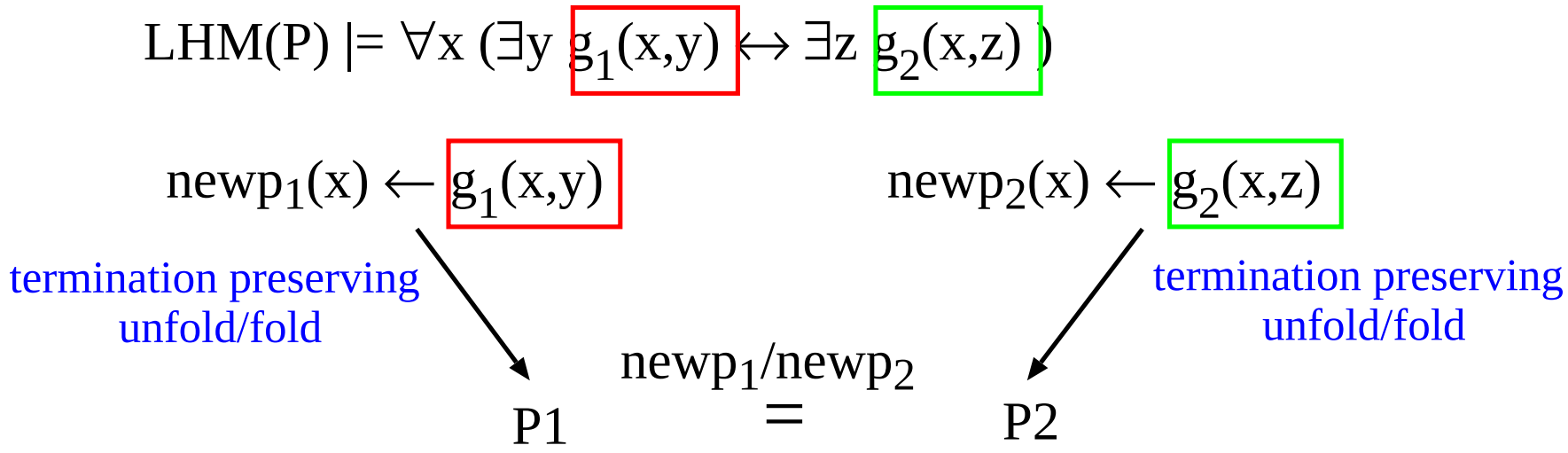
- Kott ’85:



- Courcelle ’86: completeness results, equivalence of polynomial systems (~ tree automata) by unfold/fold

Previous Work (Logic Programs)

- Pettorossi-Proietti ’94-’99: proving the equivalence of definite programs (no negation) by unfold/fold



- Roychoudury et al. ’99: as P-P, but using a more powerful folding rule (verification of parameterized finite state systems)

- Leuschel et al. ’99: Using partial deduction (= partial evaluation of LP) for the verification of infinite state systems (Petri nets)

Associativity of list concatenation

P: $a([\,],Y,Y) \leftarrow$

$a([H|X],Y,[H|Z]) \leftarrow a(X,Y,Z)$

$LHM(P) \models \forall A\ \forall B\ \forall C\ \forall D$

$(\exists X\ (a(A,B,X) \wedge a(X,C,D)) \leftrightarrow \exists Y\ (a(B,C,Y) \wedge a(A,Y,D)))$

$(A\ @\ B)\ @\ C = D$ $\qquad$ $A\ @\ (B\ @\ C) = D$

$newp_1(A,B,C,D) \leftarrow a(A,B,X) \wedge a(X,C,D)$

unfold ; fold

$newp_2(A,B,C,D) \leftarrow a(B,C,Y) \wedge a(A,Y,D)$

unfold ; fold

$newp_1([\,],B,C,D) \leftarrow a(B,C,D)$

$newp_1([\,H|A],B,C,[H|D]) \leftarrow newp_1(A,B,C,D)$

$newp_1 / newp_2$

$=$

$newp_2([\,],B,C,D) \leftarrow a(B,C,D)$

$newp_2([\,H|A],B,C,[H|D]) \leftarrow newp_2(A,B,C,D)$

This work: Proving properties of general logic programs

Given - a general logic program P with perfect model M(P)
- a closed first order formula φ in the language of P

Prove $M(P) \models \varphi$

Example

P: $even(0) \leftarrow$ $\quad$ $nat(0) \leftarrow$

$even(s(X)) \leftarrow \neg even(X)$ $\quad$ $nat(s(X)) \leftarrow nat(X)$

$odd(s(X)) \leftarrow \neg odd(X)$

prove $M(P) \models \forall X\ (nat(X) \rightarrow (even(X) \vee odd(X)))$

For ensuring the existence of M(P) we consider locally stratified programs: no recursion through negation in ground clauses

Overview of the Unfold/Fold Proof Method

Step 1. Introduce the statement $f \leftarrow \varphi$

Lloyd-Topor transformation

$Cls(f,\varphi)$

such that: - $P \cup Cls(f,\varphi)$ is locally stratified

- $M(P) \models \varphi$ iff $f \in M(P \cup Cls(f,\varphi))$

Step 2. Apply unfold/fold transformation rules that preserve $M(\ )$

$$P \cup Cls(f,\varphi) \rightarrow \cdots \rightarrow Q$$

such that $f \in M(P \cup Cls(f,\varphi))$ iff $f \in M(Q)$

if the clause $f \leftarrow$ belongs to Q then $M(P) \models \varphi$

if no clause for f belongs to Q then $M(P) \models \neg\varphi$

The unfold/fold rules should be applied according to a strategy

Plan

- Locally stratified programs and perfect models

- Lloyd-Topor transformation

- Unfold/fold transformation rules

- Transformation strategy

- Decision procedures via the UF proof method

- Program synthesis via the UF proof method

General Logic Programs

- An atom is a formula $p(t_1,...,t_k)$ where p is a predicate and $t_1,...,t_k$ are terms. A literal is either an atom or a negated atom. A clause is a formula $H \leftarrow L_1 \wedge \ldots \wedge L_n$, where H is an atom and $L_1,...,L_n$ are literals. H is the head and $L_1 \wedge \ldots \wedge L_n$ is the body. A (general logic) program is a set of clauses.

- Unlike definite programs (no negation in bodies) some programs have no least Herbrand model. For instance:

  $$p \leftarrow \neg q$$

  has two minimal models: $\{p\}$ and $\{q\}$

- We can associate a program with a unique model (possibly not least) by ordering the atoms: $q < p$.
  We compute the model bottom-up wrt $<$:
  1. The least model of the clauses with head q is $\emptyset$ and q is false in $\emptyset$.
  2. Assuming that q is false, the least model of $p \leftarrow \neg q$ is $\{p\}$.

Locally Stratified Programs

- A stratification is a function

$$\sigma: B_L \rightarrow W$$

where B_L is the set of all ground atoms in the first order language L used for writing programs and W is the set of countable ordinals

- A ground clause $H \leftarrow A_1 \wedge \ldots \wedge A_m \wedge \neg A_{m+1} \wedge \ldots \wedge \neg A_n$ is locally stratified wrt σ iff $\sigma(H) \geq \sigma(A_i)$ for i=1,...,m

$\sigma(H) > \sigma(A_i)$ for i=m+1,...,n

no recursion through negation

- A program P is locally stratified iff there exists a stratification σ such that every ground instance of a clause of P is locally stratified wrt σ

Locally Stratified Programs: Examples

- $odd(s(X)) \leftarrow \neg odd(X)$ is locally stratified

  ground instances: $odd(s(0)) \leftarrow \neg odd(0)$
  $odd(s(s(0))) \leftarrow \neg odd(s(0))$
  $\ldots$

  every clause is locally stratified wrt $\sigma(odd(s^n(0))) = n$

- $p \leftarrow \neg p$ is not locally stratified: there is no stratification σ such that $\sigma(p) > \sigma(p)$

Perfect Model

- Interpretation: a subset $\Im$ of B_L (i.e. $\Im \in \wp(B_L)$)

  $\Im \models A$ iff $A \in \Im$ and $\Im \models \neg A$ iff $A \notin \Im$

- Immediate consequence operator $T_{P,\alpha}: \wp(B_L) \rightarrow \wp(B_L)$

  where P is a locally stratified program and $\alpha \in W$ is an ordinal

$T_{P,\alpha}(\Im) = \{A \mid A \leftarrow L_1 \wedge \ldots \wedge L_n$ is a ground instance of a clause in P,
$\sigma(A)=\alpha$, and for i = 1,...,n if $\sigma(L_i)=\alpha$ then $\Im \models L_i$
else if $\sigma(L_i)=\beta<\alpha$ then $lfp(T_{P,\beta}) \models L_i\}$

$T_{P,\alpha}(\Im)$ is the set of atoms in stratum α that are one-step consequences (using the clauses in P) of the literals that are true in $\Im$ and of the literals that are true in any stratum $\beta<\alpha$

- For every ordinal $\alpha \in W$, $T_{P,\alpha}$ is a continuous operator on the lattice $\wp(B_L)$ of all interpretations. The perfect model of P is defined as:

$$M(P) = \bigcup_{\alpha \in W} lfp(T_{P,\alpha})$$

Perfect Model: An Example

P: $p \leftarrow \neg odd(X)$

$odd(s(X)) \leftarrow \neg odd(X)$

ground instances:

1. $odd(s(0)) \leftarrow \neg odd(0)$
2. $odd(s(s(0))) \leftarrow \neg odd(s(0))$

. . .

ω. $p \leftarrow \neg odd(0)$

ω. $p \leftarrow \neg odd(s(0))$

. . .

stratification:

$\sigma(p) = \omega$

$\sigma(odd(s^n(0))) = n$

$$M(P) = lfp(T_{P,0}) \cup lfp(T_{P,1}) \cup \ldots \cup lfp(T_{P,\omega})$$
$$= \varnothing \cup \{odd(s(0))\} \cup \ldots \cup \{p\}$$

Plan

- Locally stratified programs and perfect models

⇒ **Lloyd-Topor transformation**

- Unfold/fold transformation rules

- Transformation strategy

- Decision procedures via the UF proof method

- Program synthesis via the UF proof method

Lloyd-Topor Transformation

- Given a locally stratified program P and a closed first order formula φ using $\neg$, $\wedge$, $\exists$, and the predicate symbols occurring in P, introduce the statement: $f \leftarrow \varphi$, where f is a new predicate symbol

Lloyd-Topor transformation

$$f \leftarrow \varphi \quad \Rightarrow \quad Cls(f,\varphi)$$

such that:
- $P \cup Cls(f,\varphi)$ is locally stratified
- $M(P) \models \varphi$ iff $f \in M(P \cup Cls(f,\varphi))$

. . . Lloyd-Topor Transformation

Apply as long as possible the following transformations
(C[ψ] denotes a formula of the form: ... ∧ ψ ∧ ...)

$H \leftarrow C[\neg\neg\psi] \quad \Rightarrow \quad H \leftarrow C[\psi]$

$H \leftarrow C[\neg(\psi_1 \wedge \psi_2)] \quad \Rightarrow \quad \begin{cases} H \leftarrow C[\neg newp(X_1,...,X_k)] \\ newp(X_1,...,X_k) \leftarrow \psi_1 \wedge \psi_2 \end{cases}$

where $X_1,...,X_k$ are the free variables of $\psi_1 \wedge \psi_2$

$H \leftarrow C[\neg\, \exists X\, \psi] \quad \Rightarrow \quad \begin{cases} H \leftarrow C[\neg newp(X_1,...,X_k)] \\ newp(X_1,...,X_k) \leftarrow \psi \end{cases}$

where $X_1,...,X_k$ are the free variables of ψ

$H \leftarrow C[\exists X\, \psi] \quad \Rightarrow \quad H \leftarrow C[\psi\{X/Y\}]$ where Y is a new variable

LT Transformation: Less-or-Equal Example

P: $0 \leq N \leftarrow$ $\quad$ $\mathrm{nat}(0) \leftarrow$

$s(N1) \leq s(N2) \leftarrow N1 \leq N2$ $\quad$ $\mathrm{nat}(s(X)) \leftarrow \mathrm{nat}(X)$

φ: $\quad \forall N\ (\mathrm{nat}(N) \rightarrow N \leq s(N))$

Rewrite φ as: $\quad \neg\exists N(\mathrm{nat}(N) \wedge \neg N \leq s(N))$

Introduce the statement:

$$f \leftarrow \neg\ \exists N(\ \boxed{\mathrm{nat}(N) \wedge \neg\ N \leq s(N)}\ )$$

Cls(f,φ): $\quad \begin{cases} f \leftarrow \neg\ g \\ g \leftarrow \boxed{\mathrm{nat}(N) \wedge \neg\ N \leq s(N)} \end{cases}$

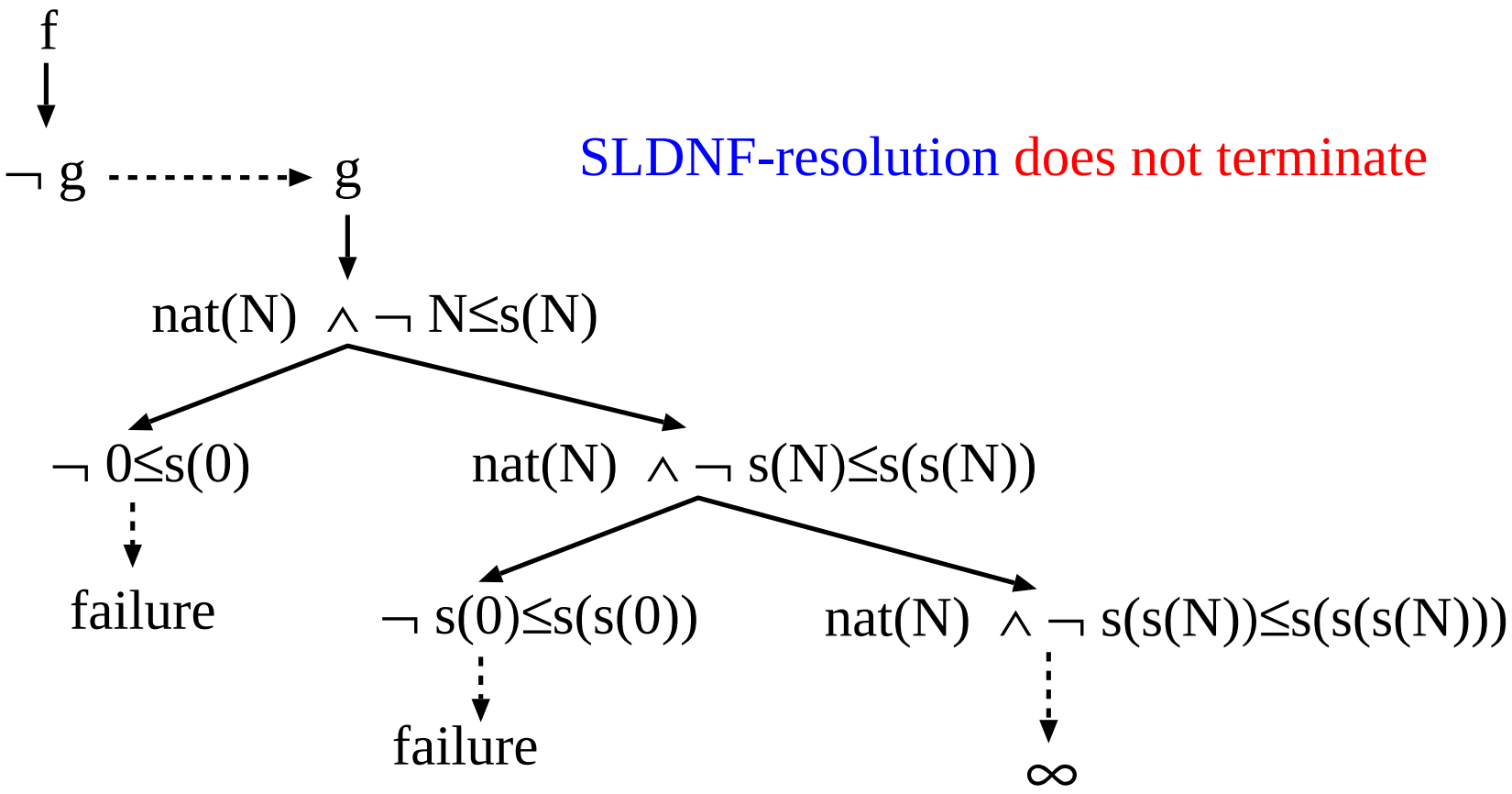
Locally stratified

Note:

$\forall N\ (g \leftarrow \mathrm{nat}(N) \wedge \neg\ N \leq s(N)) \equiv g \leftarrow \exists N(\mathrm{nat}(N) \wedge \neg\ N \leq s(N))$

Limitations of SLDNF-resolution

$$\text{Cls}(f,\varphi): \begin{cases} f \leftarrow \neg\, g \\ g \leftarrow \text{nat}(N) \wedge \neg\, N \leq s(N) \end{cases}$$



SLDNF-resolution does not terminate

We need to prove that, for all natural numbers N, ¬ N≤s(N) fails

Plan

- Locally stratified programs and perfect models

- Lloyd-Topor transformation

⇒ **Unfold/fold transformation rules**

- Transformation strategy

- Decision procedures via the UF proof method

- Program synthesis via the UF proof method

Overview of the Unfold/Fold Proof Method

Step 1. Introduce the statement $f \leftarrow \varphi$

$\downarrow$ Lloyd-Topor transformation

$Cls(f,\varphi)$

such that: - $M(P \cup Cls(f,\varphi))$ is locally stratified

- $M(P) \models \varphi$ iff $f \in M(P \cup Cls(f,\varphi))$

Step 2. Apply unfold/fold transformation rules that preserve $M(\ )$

$$P \cup Cls(f,\varphi) \rightarrow \cdots \rightarrow Q$$

such that $f \in M(P \cup Cls(f,\varphi))$ iff $f \in M(Q)$

if the clause $f \leftarrow$ belongs to Q then $M(P) \models \varphi$

if no clause for f belongs to Q then $M(P) \models \neg\varphi$

The unfold/fold rules should be applied according to a strategy

The Unfold/Fold Transformation Rules

- Construct a transformation sequence, that is, a sequence of programs

$$P_0 \rightarrow \cdots \rightarrow P_n$$

where P_{k+1} is derived from P_k by applying a transformation rule

- **Transformation Rules**:
  - R1. Definition Introduction
  - R2. Unfolding (w.r.t. positive or negative literals)
  - R3. Folding
  - R4. Tautologies

- The transformation rules preserve the Perfect Model

$$M(P_0 \cup Defs_n) = M(P_n)$$

where $Defs_n$ is the set of new clauses introduced by the Definition Introduction rule

Definition Introduction: Less-or-Equal Example

P_0:
$0 \leq N \leftarrow$
$s(N1) \leq s(N2) \leftarrow N1 \leq N2$
$nat(0) \leftarrow$
$nat(s(X)) \leftarrow nat(X)$

Definition Introduction (twice):
δ_1: $g \leftarrow nat(N) \wedge \neg\, N \leq s(N)$
δ_2: $f \leftarrow \neg g$

P_2:
$0 \leq N \leftarrow$
$s(N1) \leq s(N2) \leftarrow N1 \leq N2$
$nat(0) \leftarrow$
$nat(s(X)) \leftarrow nat(X)$
δ_1: $g \leftarrow nat(N) \wedge \neg\, N \leq s(N)$
δ_2: $f \leftarrow \neg g$

$Defs_2 = \{\delta_1, \delta_2\}$

R1. Definition Introduction

[Tamaki-Sato 84, Seki 91]

Introduce a new definition, that is, a clause

$$\delta:\ \mathrm{newp}(X_1,\ldots,X_h) \leftarrow L_1 \wedge \ldots \wedge L_n$$

where:

- newp is a new predicate symbol
- $X_1,\ldots,X_h$ are distinct variables occurring in $L_1 \wedge \ldots \wedge L_n$
- the predicate symbols of $L_1 \wedge \ldots \wedge L_n$ occur in P_k

$$P_{k+1} = P_k \cup \{\delta\}$$

Defs_k is the set of definitions introduced up to step k

No recursive definitions, no multiple clause definitions

Positive Unfolding: Less-or-Equal Example

P_2:
$0 \le N \leftarrow$
$s(N1) \le s(N2) \leftarrow N1 \le N2$
$nat(0) \leftarrow$
$nat(s(X)) \leftarrow nat(X)$
δ_1: $g \leftarrow nat(N) \wedge \neg\, N \le s(N)$
$f \leftarrow \neg g$

replace

Unfolding δ_1 wrt $nat(N)$

P_3:
$0 \le N \leftarrow$
$s(N1) \le s(N2) \leftarrow N1 \le N2$
$nat(0) \leftarrow$
$nat(s(X)) \leftarrow nat(X)$
λ_1: $g \leftarrow \neg\, 0 \le s(0)$ $\{N/0\}$
λ_2: $g \leftarrow nat(X) \wedge \neg\, s(X) \le s(s(X))$ $\{N/s(X)\}$
$f \leftarrow \neg g$

$Defs_3 = \{\delta_1, \delta_2\}$

$R2^+$. Positive Unfolding

[Tamaki-Sato 84, Seki 91]

Given a clause in P_k

$$\lambda:\ H \leftarrow G_1 \wedge \boxed{A} \wedge G_2$$

take all clauses in P_k whose head H_i unifies with A via an mgu θ_i

$$\boxed{H_1} \leftarrow \boxed{Body_1} \quad \ldots \quad \boxed{H_m} \leftarrow \boxed{Body_m}$$

and replace λ by all its resolvents w.r.t. the atom A

$$P_{k+1} = (P_k \setminus \{\lambda\}) \cup \left\{ \begin{array}{l} (H \leftarrow G_1 \wedge \boxed{Body_1} \wedge G_2)\ \theta_1 \\ \ldots \\ (H \leftarrow G_1 \wedge \boxed{Body_m} \wedge G_2)\ \theta_m \end{array} \right\}$$

If m=0 then delete λ from P_k

Negative Unfolding: Less-or-Equal Example

P_3:
$0 \le N \leftarrow$
$s(N1) \le s(N2) \leftarrow N1 \le N2$
$nat(0) \leftarrow$
$nat(s(X)) \leftarrow nat(X)$
λ_1: $g \leftarrow \neg\, 0 \le s(0)$
λ_2: $g \leftarrow nat(X) \wedge \neg\, s(X) \le s(s(X))$
$f \leftarrow \neg g$

Unfolding λ_1 wrt $\neg\, 0 \le s(0)$ and
λ_2 wrt $\neg\, s(X) \le s(s(X))$

replace

P_4:
$0 \le N \leftarrow$
$s(N1) \le s(N2) \leftarrow N1 \le N2$
$nat(0) \leftarrow$
$nat(s(X)) \leftarrow nat(X)$
~~$g \leftarrow \neg\, 0 \le s(0)$~~ {N/0}
λ_3: $g \leftarrow nat(X) \wedge \neg\, X \le s(X)$ {N1/X, N2/X}
$f \leftarrow \neg g$

$Defs_4 = \{\delta_1, \delta_2\}$

R2$^-$. Negative Unfolding

[Pettorossi-Proietti 00]

Given a clause in P_k $\quad \lambda:\ H \leftarrow G_1 \wedge \neg A \wedge G_2$

take all clauses in P_k whose head H_i unifies with A via an mgu θ_i

$$H_1 \leftarrow Body_1 \quad \ldots \quad H_m \leftarrow Body_m$$

if (1) $A = H_1\,\theta_1 = \ldots = H_m\,\theta_m$ (A is an instance of $H_1, \ldots, H_m$)

(2) $Body_1, \ldots, Body_m$ have no existential variables

then take the disjunctive normal form

$$Q_1 \vee \ldots \vee Q_r = DNF\,(\,G_1 \wedge \neg(Body_1\,\theta_1 \vee \ldots \vee Body_m\,\theta_m) \wedge G_2\,)$$

$$P_{k+1} = (P_k \setminus \{\lambda\}) \cup \{H \leftarrow Q_1, \ldots, H \leftarrow Q_r\}$$

If m=0 then delete $\neg A$ from the body of λ; if $A\theta \leftarrow$ is a clause in P_k then delete λ.

Folding: Less-or-Equal Example

P_4:
$0 \leq N \leftarrow$
$s(N1) \leq s(N2) \leftarrow N1 \leq N2$
$nat(0) \leftarrow$
$nat(s(X)) \leftarrow nat(X)$
λ_3: $g \leftarrow nat(X) \wedge \neg X \leq s(X)$
$f \leftarrow \neg g$

$Defs_4$:
δ_1: $g \leftarrow nat(N) \wedge \neg N \leq s(N)$
δ_2: $f \leftarrow \neg g$

replace

folding λ_3 wrt $nat(X) \wedge \neg X \leq s(X)$

P_5:
$0 \leq N \leftarrow$
$s(N1) \leq s(N2) \leftarrow N1 \leq N2$
$nat(0) \leftarrow$
$nat(s(X)) \leftarrow nat(X)$
λ_4: $g \leftarrow g$
$f \leftarrow \neg g$

R3. Folding

Given a clause in P_k $\quad \lambda:\ H \leftarrow G_1 \wedge G_2\theta \wedge G_3$

and a definition in $Defs_k$ $\quad \delta:\ Newp \leftarrow G_2$

if (1) $\theta = \theta_1 \circ \theta_2$ where:

- θ_1 and θ_2 share no variables
- θ_2 is a renaming of the existential variables of δ

and (2) δ has been (or will be) unfolded w.r.t. a positive literal (*)

then $P_{k+1} = (P_k \setminus \{\lambda\}) \cup \{H \leftarrow G_1 \wedge Newp\ \theta \wedge G_3\}$

Similar to [Tamaki-Sato 84, Seki 91], except for (*)

Folding: Condition (2)

P_0: $p(X) \leftarrow p(X)$
$p(X) \leftarrow q$
$q \leftarrow fail$

P_1: $newp \leftarrow \neg\, p(X)$ definition introduction

P_2: $newp \leftarrow \neg\, p(X) \wedge \neg\, q$ unfolding wrt $\neg\, p(X)$

P_3: $newp \leftarrow newp \wedge \neg\, q$ folding

$newp \in M(P_0 \cup Defs_3) = M(P_1)$ and $newp \notin M(P_3)$

Tautologies: Less-or-Equal Example

P_5: $0 \leq N \leftarrow$

$s(N1) \leq s(N2) \leftarrow N1 \leq N2$

$nat(0) \leftarrow$

$nat(s(X)) \leftarrow nat(X)$

~~$g \leftarrow g$~~

$f \leftarrow \neg g$

$H \leftarrow H \wedge G$ is a tautology

Finally, by unfolding $f \leftarrow \neg g$ we get (there is no clause for g):

P_7: $0 \leq N \leftarrow$

$s(N1) \leq s(N2) \leftarrow N1 \leq N2$

$nat(0) \leftarrow$

$nat(s(X)) \leftarrow nat(X)$

$f \leftarrow$

$f \leftarrow$ belongs to P_7

$\Rightarrow \quad f \in M(P_7)$

$\Rightarrow \quad M(P_0) \models \forall N\ (nat(N) \rightarrow N \leq s(N))$

Tautologies

$$P_{k+1} = (P_k \setminus Cs) \cup Ds$$

where $Cs \Rightarrow Ds$ is an instance of one of the following rewritings:

$\{H \leftarrow A \wedge \neg A \wedge G\} \quad \Rightarrow \quad \varnothing$

$\{H \leftarrow H \wedge G\} \quad \Rightarrow \quad \varnothing$

$\{H \leftarrow G_1 \wedge L_1 \wedge L_2 \wedge G_2\} \quad \Rightarrow \quad \{H \leftarrow G_1 \wedge L_2 \wedge L_1 \wedge G_2\}$

$\{H \leftarrow L \wedge L \wedge G\} \quad \Rightarrow \quad \{H \leftarrow L \wedge G\}$

$\{H \leftarrow G_1, \quad \Rightarrow \quad \{H \leftarrow G_1\}$
$H \leftarrow G_1 \wedge G_2\}$

$\{H \leftarrow A \wedge G_1 \wedge G_2, \quad \Rightarrow \quad \{H \leftarrow G_1 \wedge G_2,$
$H \leftarrow \neg A \wedge G_1\} \qquad\qquad H \leftarrow \neg A \wedge G_1\}$

A Derived Rule: Propositional Simplification

Suppose that in P_k a predicate p depends on nullary predicates only.
Then $p \in M(P_k)$ is decidable and,by unfolding and tautologies,

if $p \in M(P_k)$ then $P_{k+1} = (P_k \setminus D_p) \cup \{p \leftarrow\}$

if $p \notin M(P_k)$ then $P_{k+1} = (P_k \setminus D_p)$

where D_p is the set of clauses in P_k with head p

Correctness of the Unfold/Fold rules

Theorem: Let $P_0, \ldots, P_n$ be a transformation sequence. Let $Defs_n$ be the set of definitions introduced in that sequence. Then

$$M(P_0 \cup Defs_n) = M(P_n)$$

Plan

- Locally stratified programs and perfect models

- Lloyd-Topor transformation

- Unfold/fold transformation rules

⇒ **Transformation strategy**

- Decision procedures via the UF proof method

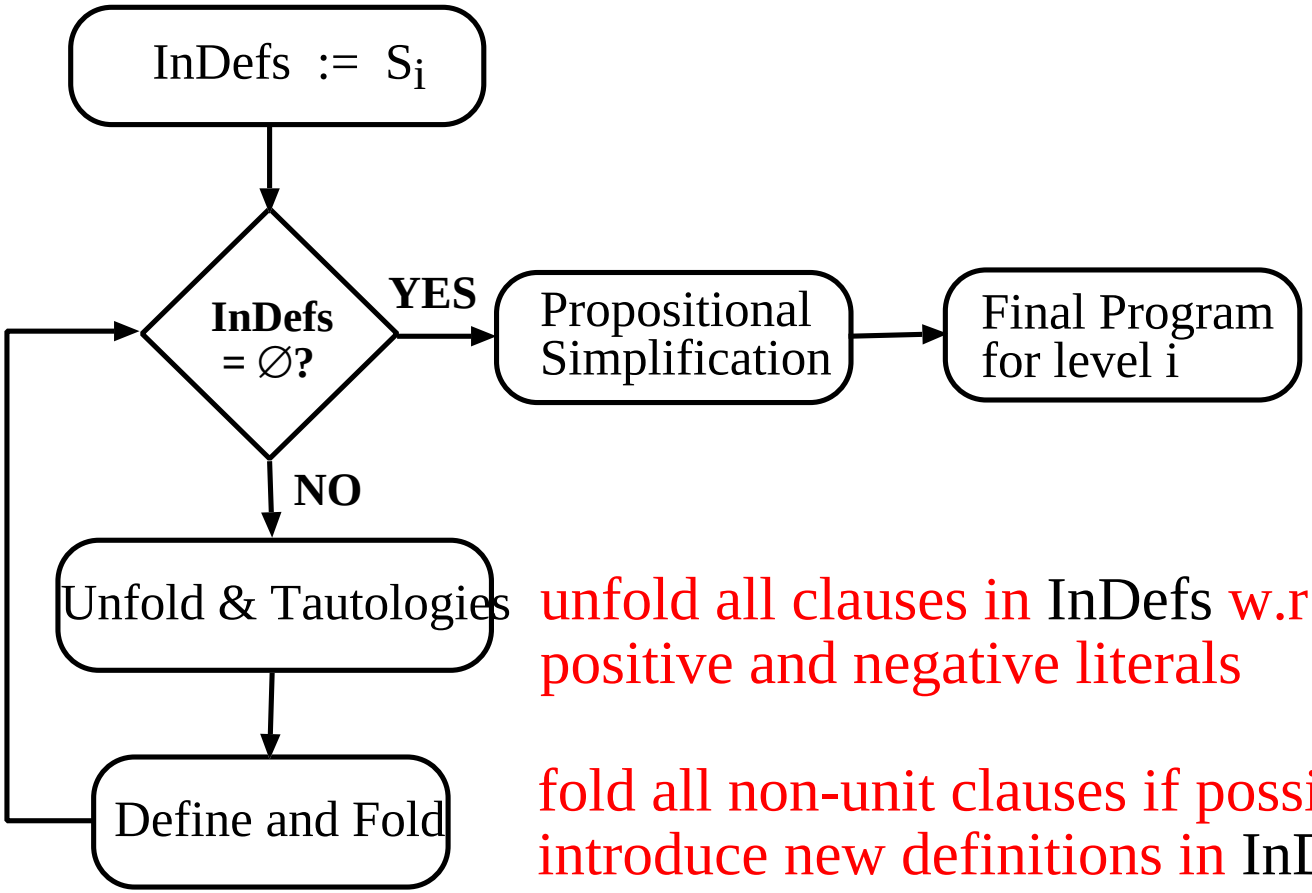
- Program synthesis via the UF proof method

The Unfold/Fold Transformation Strategy

$P \cup Cls(f,\varphi) = S_0 \cup \ldots \cup S_k$ is a finite partition into levels where:

- $S_0 = P$
- the predicates in S_i depend only on predicates in $S_0 \cup \ldots \cup S_{i-1}$

For i=1, . . . , k



unfold all clauses in InDefs w.r.t. all positive and negative literals

fold all non-unit clauses if possible; introduce new definitions in InDefs otherwise

Even-Odd Example

P: **even(0) ←** **nat(0) ←**
even(s(X)) ← ¬even(X) **nat(s(X)) ← nat(X)**
odd(s(X)) ← ¬odd(X)

We want to prove
φ: **∀X (nat(X) → (even(X) ∨ odd(X)))**

Step 1. (LT-transformation)

Introduce the statement: **f ← ¬ ∃X (nat(X) ∧ ¬even(X) ∧ ¬odd(X))**

Lloyd-Topor Transformation

Definition Introduction: **f ← ¬g** **level 2**
g ← nat(X) ∧ ¬even(X) ∧ ¬odd(X) **level 1**

Even-Odd Example

Step 2. (Transformation Strategy)

Level 1.

$g \leftarrow nat(X) \wedge \neg even(X) \wedge \neg odd(X)$

Unfold wrt
nat(X), ¬even(X), ¬odd(X)

$g \leftarrow nat(X) \wedge even(X) \wedge odd(X)$

Define &Fold

$h \leftarrow nat(X) \wedge even(X) \wedge odd(X)$
$g \leftarrow h$

Unfold wrt
nat(X), even(X), odd(X)

$h \leftarrow nat(X) \wedge \neg even(X) \wedge \neg odd(X)$
$g \leftarrow h$

Fold

$h \leftarrow g$
$g \leftarrow h$

Propositional Simplification

~~$h \leftarrow g$~~
~~$g \leftarrow h$~~

$M(\{h \leftarrow g,\ g \leftarrow h\}) = \emptyset$

Level 2.

$f \leftarrow \neg g$

Unfold wrt ¬g

$f \leftarrow$

The Transformation Strategy ...

may not terminate because at each loop one or more new definitions may be introduced.

For restricted classes of programs and formulas it always terminates

Plan

- Locally stratified programs and perfect models

- Lloyd-Topor transformation

- Unfold/fold transformation rules

- Transformation strategy

⇒ **Decision procedures via the UF proof method**

- Program synthesis via the UF proof method

Tree-typed Formulas

- Tree-typed formulas:

$$\varphi ::= p(X) \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \exists X\,(r(X) \wedge \varphi)$$

where r is defined by a tree program and p(X) is defined by a Monadic Regular program.

- Tree programs:

A set of clauses of the form:

$$r_0(t(X_1,\ldots,X_n)) \leftarrow r_1(X_1) \wedge \ldots \wedge r_n(X_n) \qquad n \geq 0$$

where t is a function symbol

Monadic Regular Programs

- MR programs have clauses of the form:

$$p_0(t(X_1,...,X_n)) \leftarrow p_1(Y_1) \wedge ... \wedge p_k(Y_k) \wedge \neg p_{k+1}(Y_{k+1}) \wedge ... \wedge \neg p_m(Y_m)$$

where t is a function symbol and $\{Y_1,...,Y_n\} \subseteq \{X_1,...,X_n\}$ $m,n \geq 0$

- MR programs are locally stratified

- Theorem: For a tree-typed formula φ and an MR program P the unfold/fold transformation strategy terminates on the input $P \cup Cls(f,\varphi)$. Moreover, if Q is the output program,

  either the clause $f \leftarrow$ belongs to Q

  or no clause for f belongs to Q

  Thus, the UF proof method is a decision procedure for $M(P) \models \varphi$

Tree-typed Formulas and MR Programs: Examples

- Tree-typed formula: $\forall X\ (\mathrm{nat}(X) \rightarrow (\mathrm{even}(X) \vee \mathrm{odd}(X)))$

  Tree program:

  $\mathrm{nat}(0) \leftarrow$

  $\mathrm{nat}(s(X)) \leftarrow \mathrm{nat}(X)$

  MR program:

  $\mathrm{even}(0) \leftarrow$

  $\mathrm{even}(s(X)) \leftarrow \neg\mathrm{even}(X)$

  $\mathrm{odd}(s(X)) \leftarrow \neg\mathrm{odd}(X)$

- Tree automata can be expressed as MR programs. Membership, complementation, intersection, containment, and equivalence of tree automata can be expressed as tree-typed formulas.

  For instance, the equivalence of two tree automata a and b can be expressed as:

  $$\forall T\ (\mathrm{tree}(T) \rightarrow (a(T) \leftrightarrow b(T)))$$

Natset-typed Formulas

- Terms: $t ::= n \mid s$
  Natural numbers: $n ::= 0 \mid N \mid s(n)$
  Sets: $s ::= [\,] \mid S \mid [\mathbf{t}|S] \mid [\mathbf{f}|S]$

A natural number k is represented by $s^k(0)$

A set of natural numbers is represented by a list $[b_0,...,b_m]$

$s^k(0) \in [b_0,...,b_m]$ iff $0 \le k \le m$ and $b_k = \mathbf{t}$

- Natset-typed formulas: $\varphi ::= p(t_1,...,t_n) \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \exists N\,(nat(N) \wedge \varphi) \mid \exists S\,(set(S) \wedge \varphi)$

where nat(X) and set(X) are defined by:

$nat(0) \leftarrow$
$nat(s(N)) \leftarrow nat(N)$

$set([\,]) \leftarrow$
$set([\mathbf{t}|S]) \leftarrow set(S)$
$set([\mathbf{f}|S]) \leftarrow set(S)$

and p is defined by a natset-typed program.

Natset-typed Programs

- Head terms: $h ::= 0 \mid s(N) \mid [\,] \mid [\mathbf{t}|S] \mid [\mathbf{f}|S]$

- Natset-typed clauses:

$$C ::= p_1(h_1,...,h_k) \leftarrow \mid p_1(h_1,...,h_k) \leftarrow p_2(X_1,...,X_m)$$

where: $p_1(h_1,...,h_k)$ is a linear atom (each variable occurs at least once), $X_i ::= N \mid S$, and $X_1,...,X_m$ are distinct variables occurring in $p_1(h_1,...,h_k)$

- Natset-typed programs are sets of natset-typed clauses.

- Theorem: For a natset-typed formula φ and a natset-typed program P the unfold/fold transformation strategy terminates on the input $P \cup Cls(f,\varphi)$. Moreover, if Q is the output program,

  either the clause $f \leftarrow$ belongs to Q

  or no clause for f belongs to Q

  Thus, the UF proof method is a decision procedure for $M(P) \models \varphi$

WS1S

- WS1S, the weak monadic second order theory of 1 successor [Buchi '60] can be decided by the unfold/fold proof method

- WS1S Formulas: $\varphi ::= n \in S \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid$
  $\exists N(nat(N) \wedge \varphi) \mid \exists S(set(S) \wedge \varphi)$

  where $n \in S$ is defined by the natset-typed program:

  Member: $0 \in [\mathbf{t}|S] \leftarrow$
  $s(N) \in [B|S] \leftarrow N \in S$

- WS1S expresses properties of the membership of finite sets of natural numbers

WS1S: Examples

Set equality: $\mathbf{S1=S2} \equiv \forall \mathbf{N}\ (\mathbf{nat(N)} \rightarrow \mathbf{N} \in \mathbf{S1} \leftrightarrow \mathbf{N} \in \mathbf{S2})$

Order over numbers:

S is 'downward' closed

$$\mathbf{N1} \leq \mathbf{N2} \equiv \forall \mathbf{S}\ (\mathbf{set(S)} \rightarrow (\mathbf{N2} \in \mathbf{S} \wedge \forall \mathbf{N3}\ (\mathbf{nat(N3)} \rightarrow \mathbf{s(N3)} \in \mathbf{S} \rightarrow \mathbf{N3} \in \mathbf{S}) \rightarrow \mathbf{N1} \in \mathbf{S}))$$

Every finite set of natural numbers has a maximum element:

$$\forall \mathbf{S}\ (\mathbf{set(S)} \rightarrow \exists \mathbf{N}(\mathbf{nat(N)} \wedge \mathbf{N} \in \mathbf{S} \wedge \neg \exists \mathbf{N1}\ (\mathbf{nat(N1)} \wedge \mathbf{N1} \in \mathbf{S} \wedge \neg \mathbf{N1} \leq \mathbf{N})))$$

There exixts no finite set which is nonempty and upwards closed:

$$\neg \exists \mathbf{S}\ (\mathbf{set(S)} \wedge \exists \mathbf{N1}\ (\mathbf{nat(N1)}\ \wedge \mathbf{N1} \in \mathbf{S}) \wedge \forall \mathbf{N2}\ (\mathbf{nat(N2)}\ \wedge \mathbf{N2} \in \mathbf{S} \rightarrow \mathbf{s(N2)} \in \mathbf{S}))$$

More complex examples: Verification of finite state systems
[Basin-Klarlund 98, Klarlund et al.96]

More Decision Procedures

By extending the encoding of the WS1S theory we can decide:

- WSkS, the monadic second order theory of k successors (membership to finite sets of strings over a k-symbol alphabet)
- CTL, the computational tree logic (a branching time temporal logic used for verifying finite state transition systems)

Plan

- Locally stratified programs and perfect models

- Lloyd-Topor transformation

- Unfold/fold transformation rules

- Transformation strategy

- Decision procedures via the UF proof method

⇒ **Program synthesis via the UF proof method**

Program Synthesis

- Apply the Unfold/Fold proof method starting from open formulas.

- Example (Maximum of a set).

$\varphi \equiv nat(N) \wedge set(S) \wedge N \in S \wedge \neg \exists N1\ (nat(N1) \wedge N1 \in S \wedge \neg N1 \leq N)$

N and S are free variables

Step 1. Apply the Lloyd-Topor transformation starting from the statement:

$max(S,N) \leftarrow nat(N) \wedge set(S) \wedge N \in S \wedge \neg\ \exists N1\ (nat(N1) \wedge N1 \in S \wedge \neg N1 \leq N)$

N and S are free variables

Cls(max,φ):

$$\begin{cases} max(S,N) \leftarrow set(S) \wedge nat(N) \wedge N \in S \wedge \neg\ newp(S,N) \\ newp(S,N) \leftarrow set(S) \wedge nat(N) \wedge nat(N1) \wedge N1 \in S \wedge \neg N1 \leq N \end{cases}$$

Set Maximum Example

Step 2. Apply the unfold/fold transformation strategy starting from:

Member $\cup$ Cls(max, φ)

and derive a program for computing the maximum of a set:

max([**t**|S],0) $\leftarrow$ new1(S)
max([**t**|S],s(N)) $\leftarrow$ max(S,N)
max([**f**|S],s(N)) $\leftarrow$ max(S,N)
new1([ ]) $\leftarrow$
new1([**f**|S]) $\leftarrow$ new1(S)

The Unfold/Fold Synthesis Method

Let φ be a first order formula with free variables $X_1, \ldots, X_n$.

Step 1. Introduce the statement $f(X_1,\ldots,X_n) \leftarrow \varphi$

Lloyd-Topor transformation

$Cls(f,\varphi)$

such that, for all ground terms $t_1,\ldots,t_n$,

$$M(Member) \models \varphi\{X_1/t_1,\ldots,X_n/t_n\}$$

iff

$$M(Member \cup Cls(f,\varphi)) \models f(t_1,\ldots,t_n)$$

Step 2. Apply unfold/fold transformation rules that preserve M( )

$$P \cup Cls(f,\varphi) \rightarrow \cdots \rightarrow Q$$

such that $f(t_1, \ldots, t_n) \in M(P \cup Cls(f,\varphi))$ iff $f(t_1, \ldots, t_n) \in M(Q)$

The unfold/fold rules should be applied according to a strategy

Optimizations of synthesized programs: Determinization

The program derived by the u/f strategy may be nondeterministic.

$p(s(X)) \leftarrow q(X)$
$p(s(X)) \leftarrow r(X)$
$q(0) \leftarrow$
$q(s(X)) \leftarrow r(X)$
$r(s(X)) \leftarrow q(X)$

$new(X) \leftarrow q(X)$
$new(X) \leftarrow r(X)$

multiple clause definition

$new(0) \leftarrow$
$new(s(X)) \leftarrow r(X)$
$new(s(X)) \leftarrow q(X)$

unfolding

$new(0) \leftarrow$
$new(s(X)) \leftarrow new(X)$

multiple clause folding

Optimizations of synthesized programs: Minimization

The programs derived during the unfold/fold strategy may contain equivalent predicates.

$p(0) \leftarrow$

$p(s(X)) \leftarrow \boxed{q(X)}$

$q(0) \leftarrow$

$q(s(X)) \leftarrow \boxed{p(X)}$

p and q have the same definition modulo predicate names.

$M(P) \models \forall X(p(X) \leftrightarrow q(X))$

$p(0) \leftarrow$

$p(s(X)) \leftarrow \boxed{p(X)}$

$q(0) \leftarrow$

~~$q(s(X)) \leftarrow \boxed{q(X)}$~~

goal replacement

if we are interested to p only
we can delete the clauses for q

Conclusions

- Theorem Proving and Program Synthesis can be done via Program Transformation
- There are theoretical connection between decision procedures for logical theories and termination of program transformation strategies
- Robust Software Construction requires several tasks: synthesis from specifications, verification, optimizing transformations, specialization. All of them can be performed in systems based on program transformers.
- The unfold/fold proof and synthesis methods from WS1S specifications have been implemented on the MAP transformation system, available at

  http://www.iasi.rm.cnr.it/~proietti/system.html

  Reasonable efficiency for small formulas.
- Ongoing work:
  - Theorem proving and synthesis via transformation of constraint logic programs
  - Verification and synthesis of infinite state systems