Program Transformation and Constraint-based Verification

Valerio Senni

Department of Computer Science, Systems and Production
University of Rome ‘Tor Vergata’, Italy

joint work with
E. De Angelis, F. Fioravanti,
A. Pettorossi, M. Proietti

CPmeetsCAV 2012, Turunç

Rule-Based Program Transformation - Origins

$$\begin{array}{lccccccl} \text{initial} & P_0 & \longmapsto & \ldots & \longmapsto & P_n & \text{final} \\ & M(P_0) & = & \ldots & = & M(P_n) & \end{array}$$

rule-based
model-preserving
(local) rewriting

An approach to developing correct & efficient programs [Burstall-Darlington 77]

$$\begin{array}{lcccl} \text{'easy-to-prove-correct'} & P_0 & \longmapsto^* & P_n & \text{correct \& efficient} \\ & M(P_0) & = & M(P_n) & \end{array}$$

$$\longmapsto^* \quad \equiv \quad \text{optimization}$$

Separate **correctness** concerns from **efficiency** concerns

$\longmapsto^*$ constructed according to a strategy

Constraint Logic Programming

Programs as sets of rules (clauses) of the form:

$\mathrm{H} \leftarrow c \wedge \mathrm{B}$ (meaning, H holds **if** c is satisfiable in $\mathcal{T}$ and B holds)

Example:

```
ordered([])
ordered([x])
ordered([x1,x2|L]) ← x1 ≤ x2 ∧ ordered([x2|L])
```

$\underbrace{x_1 \leq x_2}_{\text{solver for } \mathcal{T}}$ $\wedge$ $\underbrace{\texttt{ordered([}x_2\texttt{|L])}}_{\text{resolution}}$

Query evaluation:

$d \wedge \mathrm{G} \Longrightarrow^{k}_{\rho} c_1 \wedge \ldots \wedge c_n$ (with $c_1 \wedge \ldots \wedge c_n$ $\mathcal{T}$-satisfiable) (and $\rho = \vartheta_1 \cdot \ldots \cdot \vartheta_k$)

$\Longrightarrow^{1}_{\vartheta}$ **is**

1. $d \wedge \mathrm{G} = d \wedge \mathrm{A} \wedge \mathrm{R}$ current goal
2. $(\mathrm{A} = \mathrm{H})\vartheta$ find unifying head
3. $(d \wedge c \wedge \mathrm{B} \wedge \mathrm{R})\vartheta$ rewrite

Transformation of Constraint Logic Programs

A program as a first order theory: **theory transformation**
(changing the axioms of a theory,
while preserving the model)

Syntax	*Semantics*
logic programs	least Herbrand model
+ negation	perfect model, stable models
+ constraints	least/perfect $\mathcal{D}$-model

Rules

Definition Introduction

$$\longmapsto \quad newp(x) \leftarrow c(x) \wedge p_1(x) \wedge \ldots \wedge p_n(x)$$

Unfolding

$$\boxed{\begin{array}{l} p(x) \leftarrow d_1 \wedge B_1 \\ \quad\vdots \\ p(x) \leftarrow d_n \wedge B_n \end{array}} \quad H \leftarrow c \wedge p(x) \wedge R \quad \longmapsto \quad \begin{array}{l} H \leftarrow c \wedge d_1 \wedge B_1 \wedge R \\ \quad\vdots \\ H \leftarrow c \wedge d_n \wedge B_n \wedge R \end{array}$$

Folding

$$\boxed{p(x) \leftarrow d \wedge B} \quad H \leftarrow c \wedge B \wedge R \quad \overset{c \sqsubseteq d}{\longmapsto} \quad H \leftarrow c \wedge p(x) \wedge R$$

Clause Removal

1. $\begin{array}{l} H \leftarrow c \wedge B \\ H \leftarrow d \end{array} \quad \longmapsto \quad H \leftarrow d$ if $c \sqsubseteq d$ (c entails d)
2. $H \leftarrow c \wedge B \quad \longmapsto \quad \emptyset$ if c is unsatisfiable

Rearrangement, Addition/Deletion, Constraint Rewriting

An Introductory Example

Classical matching: S: L **P** R S = L ++ (**P** ++ R)

P: 2 0

Approximate matching: S: 5 0 4 1 4 3 3 0 3 6 5 1 4

Q

Q is near to **P** with tolerance **K=2**

P_I:

near_match(P,K,S) ← append(L,T,S) ∧ append(Q,R,T) ∧ near(P,K,Q)

near([ ],K,[ ]) ←

near([X|Xs],K,[Y|Ys]) ← X≥Y ∧ X-Y≤K ∧ near(Xs,K,Ys)

near([X|Xs],K,[Y|Ys]) ← X<Y ∧ Y-X≤K ∧ near(Xs,K,Ys)

= element-wise |X-Y| ≤ K

Assume to fix P = [2,0] and K = 2. We introduce the new definition:

snm(S) ← near_match([2,0],2,S) *definitions as patterns*

An Introductory Example

snm(S) ← near_match([2,0],2,S)

An Introductory Example

snm(S) $\leftarrow$ near_match([2,0],2,S)

Unfold near_match([2,0],2,S) (a resolution step):

snm(S) $\leftarrow$ a(L,T,S) $\wedge$ a(Q,R,T) $\wedge$ n([2,0],2,Q)

An Introductory Example

snm(S) ← near_match([2,0],2,S)

Unfold near_match([2,0],2,S) (a resolution step):

snm(S) ← a(L,T,S) ∧ a(Q,R,T) ∧ n([2,0],2,Q)

Unfold*

snm([X|S]) ← 0 ≤ X ≤ 2 ∧ a(Q,R,S) ∧ n([0],2,Q)
snm([X|S]) ← 2 < X ≤ 4 ∧ a(Q,R,S) ∧ n([0],2,Q)
snm([X|S]) ← a(L,T,S) ∧ a(Q,R,T) ∧ n([2,0],2,Q)

recall :

a([ ],X,X) ←
a([X|Xs],Y,[X|Zs]) ← a(Xs,Y,Zs)

An Introductory Example

recall :

a([ ],X,X) ←

a([X|Xs],Y,[X|Zs]) ← a(Xs,Y,Zs)

snm(S) ← near_match([2,0],2,S)

Unfold near_match([2,0],2,S) (a resolution step):

snm(S) ← a(L,T,S) ∧ a(Q,R,T) ∧ n([2,0],2,Q)

Unfold*

1. snm([X|S]) ← $0 \le X \le 2$ ∧ a(Q,R,S) ∧ n([0],2,Q)
2. snm([X|S]) ← $2 < X \le 4$ ∧ a(Q,R,S) ∧ n([0],2,Q)

snm([X|S]) ← a(L,T,S) ∧ a(Q,R,T) ∧ n([2,0],2,Q)

By merging 1 and 2 and reasoning by cases we can determinize

snm([X|S]) ← $0 \le X \le 4$ ∧ a(Q,R,S) ∧ n([0],2,Q)
snm([X|S]) ← $0 \le X \le 4$ ∧ a(L,T,S) ∧ a(Q,R,T) ∧ n([2,0],2,Q)
snm([X|S]) ← $X < 0$ ∧ a(L,T,S) ∧ a(Q,R,T) ∧ n([2,0],2,Q)
snm([X|S]) ← $X > 4$ ∧ a(L,T,S) ∧ a(Q,R,T) ∧ n([2,0],2,Q)

mutually exclusive

An Introductory Example

snm(S) ← near_match([2,0],2,S)

Unfold near_match([2,0],2,S) (a resolution step):

Fold (inverse of Unfold)

snm(S) ← a(L,T,S) ∧ a(Q,R,T) ∧ n([2,0],2,Q)

Unfold*

1. snm([X|S]) ← 0 ≤ X ≤ 2 ∧ a(Q,R,S) ∧ n([0],2,Q)
2. snm([X|S]) ← 2 < X ≤ 4 ∧ a(Q,R,S) ∧ n([0],2,Q)

snm([X|S]) ← a(L,T,S) ∧ a(Q,R,T) ∧ n([2,0],2,Q)

By merging 1 and 2 and reasoning by cases we can determinize

snm([X|S]) ← 0 ≤ X ≤ 4 ∧ a(Q,R,S) ∧ n([0],2,Q)
snm([X|S]) ← 0 ≤ X ≤ 4 ∧ a(L,T,S) ∧ a(Q,R,T) ∧ n([2,0],2,Q)
snm([X|S]) ← X < 0 ∧ a(L,T,S) ∧ a(Q,R,T) ∧ n([2,0],2,Q)
snm([X|S]) ← X > 4 ∧ a(L,T,S) ∧ a(Q,R,T) ∧ n([2,0],2,Q)

mutually exclusive

tupling predicates that share variables

An Introductory Example

snm(S) ← near_match([2,0],2,S)

Unfold near_match([2,0],2,S) (a resolution step):

snm(S) ← a(L,T,S) ∧ a(Q,R,T) ∧ n([2,0],2,Q)

Unfold*

1. snm([X|S]) ← 0 ≤ X ≤ 2 ∧ a(Q,R,S) ∧ n([0],2,Q)
2. snm([X|S]) ← 2 < X ≤ 4 ∧ a(Q,R,S) ∧ n([0],2,Q)

snm([X|S]) ← a(L,T,S) ∧ a(Q,R,T) ∧ n([2,0],2,Q)

By merging 1 and 2 and reasoning by cases we can determinize

snm([X|S]) ← 0 ≤ X ≤ 4 ∧ a(Q,R,S) ∧ n([0],2,Q)
snm([X|S]) ← 0 ≤ X ≤ 4 ∧ a(L,T,S) ∧ a(Q,R,T) ∧ n([2,0],2,Q)
snm([X|S]) ← X < 0 ∧ a(L,T,S) ∧ a(Q,R,T) ∧ n([2,0],2,Q)
snm([X|S]) ← X > 4 ∧ a(L,T,S) ∧ a(Q,R,T) ∧ n([2,0],2,Q)

new1

Definition + Fold

mutually exclusive

An Introductory Example

By Folding, we get

$\mathrm{snm}([X|S]) \leftarrow 0 \leq X \leq 4 \wedge \mathbf{new1}(S)$
$\mathrm{snm}([X|S]) \leftarrow X < 0 \wedge \mathrm{snm}(S)$
$\mathrm{snm}([X|S]) \leftarrow X > 4 \wedge \mathrm{snm}(S)$

where **new1** is defined as follows

$\mathrm{new1}(S) \leftarrow a(Q,R,S) \wedge n([0],2,Q)$
$\mathrm{new1}(S) \leftarrow a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$

An Introductory Example

By Folding, we get

$snm([X|S]) \leftarrow 0 \leq X \leq 4 \wedge \mathbf{new1}(S)$
$snm([X|S]) \leftarrow X < 0 \wedge snm(S)$
$snm([X|S]) \leftarrow X > 4 \wedge snm(S)$

where **new1** is defined as follows

$new1(S) \leftarrow a(Q,R,S) \wedge n([0],2,Q)$
$new1(S) \leftarrow a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$

Unfold* + case-split

$new1([X|S]) \leftarrow -2 \leq X \leq 2$
$new1([X|S]) \leftarrow 2 < X \leq 4 \wedge a(Q,R,S) \wedge n([0],2,Q)$
$new1([X|S]) \leftarrow 2 < X \leq 4 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$
$new1([X|S]) \leftarrow X < -2 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$
$new1([X|S]) \leftarrow X > 4 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$

mutually exclusive

An Introductory Example

By Folding, we get

$snm([X|S]) \leftarrow 0 \leq X \leq 4 \wedge \textbf{new1}(S)$
$snm([X|S]) \leftarrow X < 0 \wedge snm(S)$
$snm([X|S]) \leftarrow X > 4 \wedge snm(S)$

where **new1** is defined as follows

$new1(S) \leftarrow a(Q,R,S) \wedge n([0],2,Q)$
$new1(S) \leftarrow a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$

Unfold* + case-split

$new1([X|S]) \leftarrow -2 \leq X \leq 2$
$new1([X|S]) \leftarrow 2 < X \leq 4 \wedge a(Q,R,S) \wedge n([0],2,Q)$
$new1([X|S]) \leftarrow 2 < X \leq 4 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$
$new1([X|S]) \leftarrow X < -2 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$ → **snm**
$new1([X|S]) \leftarrow X > 4 \wedge a(L,T,S) \wedge a(Q,R,T) \wedge n([2,0],2,Q)$ → **snm**

Fold (the two boxed bodies of the 2nd and 3rd clauses → the bodies of new1(S))

mutually exclusive

Fold

An Introductory Example

The final program P_F :

$snm([X|S]) \leftarrow 0 \leq X \leq 4 \wedge new1(S)$
$snm([X|S]) \leftarrow X < 0 \wedge snm(S)$
$snm([X|S]) \leftarrow X > 4 \wedge snm(S)$

$new1([X|S]) \leftarrow -2 \leq X \leq 2$
$new1([X|S]) \leftarrow 2 < X \leq 4 \wedge new1(S)$
$new1([X|S]) \leftarrow X < -2 \wedge snm(S)$
$new1([X|S]) \leftarrow X > 4 \wedge snm(S)$

An Introductory Example

The final program P_F :

snm([X|S]) ← 0 ≤ X ≤ 4 ∧ new1(S)
snm([X|S]) ← X < 0 ∧ snm(S)
snm([X|S]) ← X > 4 ∧ snm(S)

new1([X|S]) ← -2 ≤ X ≤ 2
new1([X|S]) ← 2 < X ≤ 4 ∧ new1(S)
new1([X|S]) ← X < -2 ∧ snm(S)
new1([X|S]) ← X > 4 ∧ snm(S)

Correctness:

For all S

$M(P_I) \models$ near_match([2,0],2,S)

iff

$M(P_F) \models$ snm(S)

where M( ) denotes the least *D-model*

An Introductory Example

The final program P_F :

snm([X|S]) ← 0 ≤ X ≤ 4 ∧ new1(S)
snm([X|S]) ← X < 0 ∧ snm(S)
snm([X|S]) ← X > 4 ∧ snm(S)

new1([X|S]) ← -2 ≤ X ≤ 2
new1([X|S]) ← 2 < X ≤ 4 ∧ new1(S)
new1([X|S]) ← X < -2 ∧ snm(S)
new1([X|S]) ← X > 4 ∧ snm(S)

Correctness:

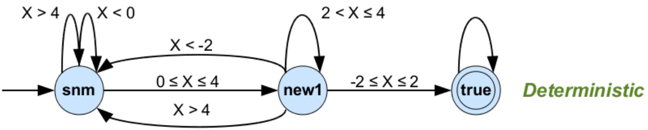
For all S

$M(P_I) \models$ near_match([2,0],2,S)

iff

$M(P_F) \models$ snm(S)

where M( ) denotes the least *D-model*



Transformation Strategies

Directed by syntactic features of programs:

- Specializing programs to the context of use (pre-computing)

  snm(S) ← near_match(**[2,0]**,**2**,S)

- Avoiding the computation of unnecessary values

  near_match(P,K,S) ← a(**L**,T,S) ∧ a(Q,R,T) ∧ near(P,K,Q)

- Avoding multiple visits of data structures and repeated computations

  near_match(P,K,S) ← a(L,**T**,S) ∧ a(**Q**,R,**T**) ∧ near(P,K,**Q**)

- Reducing nondeterminism (avoid multiple matchings per call-pattern)

  snm([X|S]) ← **0≤X≤4** ∧ a(Q,R,S) ∧ n([0],2,Q)
  snm([X|S]) ← **0≤X≤4** ∧ a(L,T,S) ∧ a(Q,R,T) ∧ near([2,0],2,Q)

  ⇓

  snm([X|S]) ← **0≤X≤4** ∧ new1(S)

Transformation Strategies

Directed by syntactic features of programs:

- Specializing programs to the context of use (pre-computing)

  snm(S) ← near_match(**[2,0]**,**2**,S)

- Avoiding the computation of unnecessary values

  near_match(P,K,S) ← a(**L**,T,S) ∧ a(Q,R,T) ∧ near(P,K,Q)

- Avoding multiple visits of data structures and repeated computations

  near_match(P,K,S) ← a(L,**T**,S) ∧ a(**Q**,R,**T**) ∧ near(P,K,**Q**)

- Reducing nondeterminism (avoid multiple matchings per call-pattern)

  snm([X|S]) ← **0≤X≤4** ∧ a(Q,R,S) ∧ n([0],2,Q)
  snm([X|S]) ← **0≤X≤4** ∧ a(L,T,S) ∧ a(Q,R,T) ∧ near([2,0],2,Q)

  ⇓

  snm([X|S]) ← **0≤X≤4** ∧ new1(S)

Transformation Strategies

Directed by syntactic features of programs:

- Specializing programs to the context of use (pre-computing)

  snm(S) $\leftarrow$ near_match(**[2,0]**,**2**,S)

- Avoiding the computation of unnecessary values

  near_match(P,K,S) $\leftarrow$ a(**L**,T,S) $\wedge$ a(Q,R,T) $\wedge$ near(P,K,Q)

- Avoding multiple visits of data structures and repeated computations

  near_match(P,K,S) $\leftarrow$ a(L,**T**,S) $\wedge$ a(**Q**,R,**T**) $\wedge$ near(P,K,**Q**)

- Reducing nondeterminism (avoid multiple matchings per call-pattern)

  snm([X|S]) $\leftarrow$ **0≤X≤4** $\wedge$ a(Q,R,S) $\wedge$ n([0],2,Q)
  snm([X|S]) $\leftarrow$ **0≤X≤4** $\wedge$ a(L,T,S) $\wedge$ a(Q,R,T) $\wedge$ near([2,0],2,Q)

  $\Downarrow$

  snm([X|S]) $\leftarrow$ **0≤X≤4** $\wedge$ new1(S)

Transformation Strategies

Directed by syntactic features of programs:

- Specializing programs to the context of use (pre-computing)

  snm(S) ← near_match(**[2,0]**,**2**,S)

- Avoiding the computation of unnecessary values

  near_match(P,K,S) ← a(**L**,T,S) ∧ a(Q,R,T) ∧ near(P,K,Q)

- Avoding multiple visits of data structures and repeated computations

  near_match(P,K,S) ← a(L,**T**,S) ∧ a(**Q**,R,**T**) ∧ near(P,K,**Q**)

- Reducing nondeterminism (avoid multiple matchings per call-pattern)

  snm([X|S]) ← **0≤X≤4** ∧ a(Q,R,S) ∧ n([0],2,Q)
  snm([X|S]) ← **0≤X≤4** ∧ a(L,T,S) ∧ a(Q,R,T) ∧ near([2,0],2,Q)

  ⇓

  snm([X|S]) ← **0≤X≤4** ∧ new1(S)

Rule-Based Program Transformation - More

P_0	$\longmapsto$	$\ldots$	$\longmapsto$	P_n	*What do we preserve?*
$M(P_0)$	$=$	$\ldots$	$=$	$M(P_n)$	a model
$A \in M(P_0)$	iff	$\ldots$	iff	$A \in M(P_n)$	selected predicates
$M(P_0) \models \varphi$	iff	$\ldots$	iff	$M(P_n) \models \varphi$	a class of formulas

$\longmapsto^* \equiv$ deduction

Depending on the choice of the set of **rules** and the **transformation strategy**

Applications

Theorem Proving

[Kott,P,P,Roychoudhury,Seki]

$T \cup \{p \leftarrow \varphi\} \longmapsto^* S \cup \{p \leftarrow\}$

Program Verification

[Albert,Gallagher,Puebla]

$[\![P]\!] \cup \{p \leftarrow \varphi\} \longmapsto^* Q \cup \{p \leftarrow\}$

where $[\![P]\!]$ is an encoding of the program semantics (e.g., an interpreter)

Program Synthesis

[Darlington,Deville,Flener,Hogger,Lau,Manna,Waldinger]

$T \cup \{p(x) \leftarrow \varphi(x)\} \longmapsto^* P$

s.t. $T \models \varphi(a)$ iff $p(a) \in M(P)$

Improving Infinite-state Systems Model Checking

Specialization-based Model Checking

Program Specialization

$\mathcal{P} : I_1 \times I_2 \longrightarrow O$

By partial evaluation

$\mathcal{P}_1 : I_2 \longrightarrow O$ A faster residual program

Take advantage of static knowledge

Specialization-based Symbolic Model Checking

$\mathcal{CTL} : TS \times \varphi \times Parameters \longrightarrow$ yes/no

By partial evaluation

$\mathcal{MC}^{\varphi}_{TS} : Parameters \longrightarrow$ yes/no An ad-hoc model checker

Specialization-based Model Checking

Program Specialization

$\mathcal{P} : I_1 \times I_2 \longrightarrow O$

By partial evaluation

$\mathcal{P}_1 : I_2 \longrightarrow O$ A faster residual program

Take advantage of static knowledge

Specialization-based Symbolic Model Checking

$\mathcal{CTL} : TS \times \varphi \times Parameters \longrightarrow$ yes/no

By partial evaluation

$\mathcal{MC}^{\varphi}_{TS} : Parameters \longrightarrow$ yes/no An ad-hoc model checker

Specialization Strategy

Input: P and a clause δ_0: $p_{sp}(x) \leftarrow c(x) \wedge p(x)$

Output: *SpecP* s.t. $p_{sp}(z) \in M(P \cup \{\delta_0\})$ iff $p_{sp}(z) \in M(SpecP)$

$SpecP := \emptyset$;

$Defs := \{\delta_0\}$;

while $\exists\, \delta \in Defs$ **do**

$\Gamma := \mathit{Unfold}\ \delta$

$\Delta := \mathit{Simplify}\ \Gamma$

$(\Phi, NewDefs) := \mathit{Generalize\&Fold}\ \Delta$

$Defs := (Defs - \{\delta\}) \cup NewDefs$

$SpecP := SpecP \cup \Phi$

od

Specialization Strategy

Input: P and a clause δ_0: $p_{sp}(x) \leftarrow \underline{c(x) \wedge p(x)}$

Output: $SpecP$ s.t. $p_{sp}(z) \in M(P \cup \{\delta_0\})$ iff $p_{sp}(z) \in M(SpecP)$

$SpecP := \emptyset$;

$Defs := \{\delta_0\}$;

while $\exists\, \delta \in Defs$ **do**

$\Gamma := \mathit{Unfold}\ \delta$

$\Delta := \mathit{Simplify}\ \Gamma$

$(\Phi, NewDefs) := \mathit{Generalize\&Fold}\ \Delta$

$Defs := (Defs - \{\delta\}) \cup NewDefs$

$SpecP := SpecP \cup \Phi$

od

Specialization Strategy

Input: P and a clause δ_0: $p_{sp}(x) \leftarrow \underline{c(x) \wedge p(x)}$

Output: $SpecP$ s.t. $p_{sp}(z) \in M(P \cup \{\delta_0\})$ iff $p_{sp}(z) \in M(SpecP)$

$SpecP := \emptyset$;

$Defs := \{\delta_0\}$;

while $\exists\, \delta \in Defs$ **do**

$\Gamma := Unfold\ \delta$ (Propagate Context)

$\Delta := Simplify\ \Gamma$

$(\Phi, NewDefs) := Generalize\&Fold\ \Delta$ (Apply Induction)

$Defs := (Defs - \{\delta\}) \cup NewDefs$

$SpecP := SpecP \cup \Phi$

od

Specialization Strategy

Input: P and a clause δ_0: $p_{sp}(x) \leftarrow \underline{c(x) \wedge p(x)}$

Output: *SpecP* s.t. $p_{sp}(z) \in M(P \cup \{\delta_0\})$ iff $p_{sp}(z) \in M(SpecP)$

$SpecP := \emptyset$;

$Defs := \{\delta_0\}$;

while $\exists\, \delta \in Defs$ **do**

$\Gamma := Unfold\ \delta$

$\Delta := Simplify\ \Gamma$

$(\Phi, NewDefs) := Generalize\&Fold\ \Delta$ ⟸

$Defs := (Defs - \{\delta\}) \cup NewDefs$

$SpecP := SpecP \cup \Phi$

od

We ensure *termination* by using wqos and generalization operators [à la Cousot-Halbwachs 78]

Specialization Strategy

Input: P and a clause δ_0: $p_{sp}(x) \leftarrow \underline{c(x) \wedge p(x)}$

Output: *SpecP* s.t. $p_{sp}(z) \in M(P \cup \{\delta_0\})$ iff $p_{sp}(z) \in M(SpecP)$

$SpecP := \emptyset$;

$Defs := \{\delta_0\}$;

while $\exists\, \delta \in Defs$ **do**

$\Gamma := \mathit{Unfold}\ \delta$

$\Delta := \mathit{Simplify}\ \Gamma$

$(\Phi, NewDefs) := \mathit{Generalize\&Fold}\ \Delta$ ⟸

$Defs := (Defs - \{\delta\}) \cup NewDefs$

$SpecP := SpecP \cup \Phi$

od

We ensure *termination* by using wqos and generalization operators [à la Cousot-Halbwachs 78]

- **Automated**
- **Terminating**

Application to Backward Reachability

Infinite-State System : $\langle Var, I, T, U \rangle$ (constraints on $\mathcal{Z}_{lin}$)

Given a set S of states, $\text{PRE}(T,S) = \{X \mid \exists X' \in S \;\; s.t. \;\; T(X,X')\}$

Goal: Check that $\text{PRE}^{\omega}(T,U) \cap I = \emptyset$ (undecidable)

Problem: The computation of $\text{PRE}^{\omega}(T,U)$ may not terminate

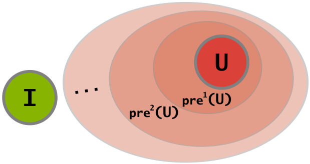
Application to Backward Reachability

Infinite-State System : $\langle Var, I, T, U \rangle$ (constraints on $\mathcal{Z}_{lin}$)

Given a set S of states, $\text{PRE}(T,S) = \{X \mid \exists X' \in S \;\; s.t. \;\; T(X,X')\}$

Goal: Check that $\text{PRE}^{\omega}(T,U) \cap I = \emptyset$ (undecidable)

Problem: The computation of $\text{PRE}^{\omega}(T,U)$ may not terminate



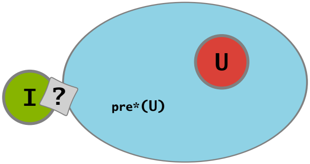
Application to Backward Reachability

Infinite-State System : $\langle Var, I, T, U \rangle$ (constraints on $\mathcal{Z}_{lin}$)

Given a set S of states, $\text{PRE}(T,S) = \{X \mid \exists X' \in S \;\; s.t. \;\; T(X,X')\}$

Goal: Check that $\text{PRE}^{\omega}(T,U) \cap I = \emptyset$ (undecidable)

Problem: The computation of $\text{PRE}^{\omega}(T,U)$ may not terminate



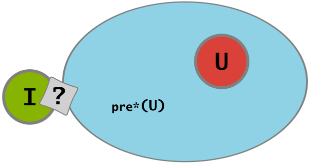
Application to Backward Reachability

Infinite-State System : $\langle Var, I, T, U \rangle$ (constraints on $\mathcal{Z}_{lin}$)

Given a set S of states, $\mathrm{PRE}(T,S) = \{X \mid \exists X' \in S \;\; s.t. \;\; T(X,X')\}$

Goal: Check that $\mathrm{PRE}^{\omega}(T,U) \cap I = \emptyset$ (undecidable)

Problem: The computation of $\mathrm{PRE}^{\omega}(T,U)$ may not terminate



knowledge of the ***target*** (I)
is not taken into account
in the construction of $\mathrm{PRE}^{\omega}(T,U)$

Encoding (Backward) Reachability in CLP

$$Sys = \langle Var, I, T, U \rangle \quad \text{where} \quad \begin{array}{rl} I: & init_1(X) \vee \ldots \vee init_k(X) \\ T: & t_1(X, X') \vee \ldots \vee t_m(X, X') \\ U: & u_1(X) \vee \ldots \vee u_n(X) \end{array}$$

$$\begin{array}{rrcl} I_1: & not\_safe & \leftarrow & init_1(X) \wedge bwReach(X) \\ & & \vdots & \\ I_k: & not\_safe & \leftarrow & init_k(X) \wedge bwReach(X) \\ T_1: & bwReach(X) & \leftarrow & t_1(X, X') \wedge bwReach(X') \\ & & \vdots & \\ T_m: & bwReach(X) & \leftarrow & t_m(X, X') \wedge bwReach(X') \\ U_1: & bwReach(X) & \leftarrow & u_1(X) \\ & & \vdots & \\ U_n: & bwReach(X) & \leftarrow & u_n(X) \end{array}$$

see also [Fribourg 97, Delzanno-Podelski 99]

full CTL encoded similarly [e.g. LOPSTR10]

Source-to-Source Specialization

System S $\Downarrow$ $P_S \in \mathrm{CLP}(\mathcal{Z})$ $\overset{\textit{specialize}}{\Longrightarrow}$ $\mathit{SpecP}_S \in \mathrm{CLP}(\mathcal{Z})$ $\Uparrow$ Specialized System SpecS

Input: $S = \langle \mathit{Var}, I, T, U \rangle$

BACKWARD we specialize w.r.t. the *Initial States*

$$\mathrm{PRE}^{\omega}(T, U) \cap I = \emptyset \quad \textbf{iff} \quad \mathrm{PRE}^{\omega}_{T,I}(U) = \emptyset$$

Output: $\mathit{SpecS} = \langle \mathit{SpecVar}, \mathit{SpecI}, \mathit{SpecT}, \mathit{SpecU} \rangle$

The standard operator PRE^{ω} behaves on *SpecS* as $\mathrm{PRE}^{\omega}_{T,I}$

Source-to-Source Specialization

System S $\Downarrow$ $P_S \in \mathrm{CLP}(\mathcal{Z})$ $\overset{\textit{specialize}}{\Longrightarrow}$ $SpecP_S \in \mathrm{CLP}(\mathcal{Z})$ $\Uparrow$ Specialized System $SpecS$

Input: $S = \langle Var, I, T, U \rangle$

BACKWARD we specialize w.r.t. the *Initial States*

$$\mathrm{PRE}^{\omega}(T,U) \cap I = \emptyset \quad \textbf{iff} \quad \mathrm{PRE}^{\omega}_{T,I}(U) = \emptyset$$

Output: $SpecS = \langle SpecVar, SpecI, SpecT, SpecU \rangle$

The standard operator PRE^{ω} behaves on $SpecS$ as $\mathrm{PRE}^{\omega}_{T,I}$

Source-to-Source Specialization

$$
\begin{array}{ccc}
\text{System } S & & \text{Specialized System } SpecS \\
\Downarrow & & \Uparrow \\
P_S \in \mathrm{CLP}(\mathcal{Z}) & \overset{specialize}{\Longrightarrow} & SpecP_S \in \mathrm{CLP}(\mathcal{Z})
\end{array}
$$

Input: $S = \langle Var, I, T, U \rangle$

BACKWARD we specialize w.r.t. the *Initial States*

$$\mathrm{PRE}^{\omega}(T, U) \cap I = \emptyset \quad \textbf{iff} \quad \mathrm{PRE}^{\omega}_{T,I}(U) = \emptyset$$

Output: $SpecS = \langle SpecVar, SpecI, SpecT, SpecU \rangle$

> The standard operator PRE^{ω} behaves on *SpecS* as $\mathrm{PRE}^{\omega}_{T,I}$

	default		F	
EXAMPLES	*Sys*	*SpSys*	*Sys*	*SpSys*
Bakery2	0.03	0.05	0.06	0.04
Bakery3	0.70	0.25	∞	3.68
MutAst	1.46	0.37	0.22	0.59
Peterson	56.49	0.10	∞	13.48
Ticket	∞	0.03	0.02	0.19
Berkeley RISC	0.01	0.04	0.01	0.02
DEC Firefly	0.01	0.02	0.01	0.07
IEEE Futurebus	0.26	0.68	∞	∞
Illinois Cache Coherence	0.01	0.03	∞	0.07
Barber	0.62	0.21	∞	0.08
CSM	56.39	7.69	∞	125.32
Consistency	∞	0.11	∞	324.14
Insertion Sort	0.03	0.06	0.18	0.02
Selection Sort	∞	0.21	∞	0.33
Reset Petri Net	∞	0.02	∞	0.01
Train	42.24	59.21	∞	0.46
No. of verified properties	12	16	6	15
	BACKWARD		FORWARD	

Timings : *Sys* $=$ **fixpoint only**
SpSys $=$ **specialization** $+$ **fixpoint**

fixpoint computed using ALV [Bultan et al. 09], based on the Omega library

'$\perp$' = 'Unable to verify' and '∞' = 'Timeout' (10 minutes)

	default		F	
EXAMPLES	*Sys*	*SpSys*	*Sys*	*SpSys*
Bakery2	0.03	0.05	0.06	0.04
Bakery3	0.70	0.25	∞	3.68
MutAst	1.46	0.37	0.22	0.59
Peterson	56.49	0.10	∞	13.48
Ticket	∞	0.03	0.02	0.19
Berkeley RISC	0.01	0.04	0.01	0.02
DEC Firefly	0.01	0.02	0.01	0.07
IEEE Futurebus	0.26	0.68	∞	∞
Illinois Cache Coherence	0.01	0.03	∞	0.07
Barber	0.62	0.21	∞	0.08
CSM	56.39	7.69	∞	125.32
Consistency	∞	0.11	∞	324.14
Insertion Sort	0.03	0.06	0.18	0.02
Selection Sort	∞	0.21	∞	0.33
Reset Petri Net	∞	0.02	∞	0.01
Train	42.24	59.21	∞	0.46
No. of verified properties	12	16	6	15

BACKWARD FORWARD

- Specialization *improves precision*
- Overall, it *does not deteriorate verification time*
- Applicable in both *forward and backward* analyses

Specialization-based Software Model Checking

$$
\begin{array}{lcl}
a & ::= & n \mid x \mid a_1 + a_2 \mid a_1 - a_2 \mid a_1 \times a_2 \\
b & ::= & \textbf{true} \mid \textbf{false} \mid a_1 \ op \ a_2 \mid !\, b \mid b_1 \ \&\& \ b_2 \mid b_1 \ \| \ b_2 \\
t & ::= & * \mid b \\
c & ::= & skip \mid x = a \mid c_1 ; c_2 \mid \textbf{if } t \textbf{ then } c_1 \textbf{ else } c_2 \mid \textbf{while } t \textbf{ do } c \textbf{ od}
\end{array}
$$

CLP interpreter for the operational semantics of SIMP

```
t(s(skip,S), E)
t(s(asgn(var(X),A),E),s(skip,E1))  ←  aeval(A,S,V), update(var(X),V,S,E1)
t(s(comp(C0,C1),S), s(C1,S1))  ←  t(s(C0,S),S1)
t(s(comp(C0,C1),S), s(comp(C0',C1),S'))  ←  t(s(C0,S), s(C0',S'))
t(s(ite(B,C0,_),S), s(C0,S))  ←  beval(B,S)
t(s(ite(B,_,C1),S), s(C1,S))  ←  beval(not(B),S)
t(s(ite(ndc,S1,_),E),s(S1,E))
t(s(ite(ndc,_,S2),E),s(S3,E))
t(s(while(B,C),S),s(ite(B,comp(C,while(B,C)),skip),S))
```

see also [Peralta,Gallagher SAS98+LOPSTR99]

Summarizing

Pros/Cons:

- improvement of termination but increase in size
- φ-preserving and terminating
- independent of the verification tool/independent of the verification tool

Features:

- control on **termination/precision** (*wqos* and *generalizations*)
- $\mathcal{Z}$ **hard** to solve: *precise relaxation* to $\mathcal{R}$
- **logics**: *CTL** and *ω-regular languages*
- residual program **size**: *controlling polyvariance*

Reasoning on Data Structures + Constraints

Context & Related Work

- Transformation and Theorem Proving:
  - equivalence proofs by unfold/fold [Kott-82, PP-99, Roychoudhury et al-99]
  - first order theorem proving by unfold/fold [PP-00]

- Deforestation [Wadler-90] used for quantifier elimination
  A technique for the elimination of intermediate data structures

- No existential variables entails No intermediate data structures
  Existential variables occur in the body of a clause and not in the head

  E.g., X existential in $p \leftarrow q(X) \wedge r(X)$

  Since $\forall X(\, p \leftarrow q(X) \wedge r(X)\,) \equiv p \leftarrow \exists X(q(X) \wedge r(X))$

Context & Related Work

- Transformation and Theorem Proving:
  - equivalence proofs by unfold/fold [Kott-82, PP-99, Roychoudhury et al-99]
  - first order theorem proving by unfold/fold [PP-00]

- Deforestation [Wadler-90] used for quantifier elimination
  A technique for the elimination of intermediate data structures

- No existential variables entails No intermediate data structures
  Existential variables occur in the body of a clause and not in the head

  E.g., X existential in $p \leftarrow q(X) \wedge r(X)$

  Since $\forall X(p \leftarrow q(X) \wedge r(X)) \equiv p \leftarrow \exists X(q(X) \wedge r(X))$

Context & Related Work

- Transformation and Theorem Proving:
  - equivalence proofs by unfold/fold [Kott-82, PP-99, Roychoudhury et al-99]
  - first order theorem proving by unfold/fold [PP-00]

- Deforestation [Wadler-90] used for quantifier elimination

  A technique for the elimination of intermediate data structures

- No existential variables entails No intermediate data structures

  Existential variables occur in the body of a clause and not in the head

  E.g., X existential in $p \leftarrow q(X) \wedge r(X)$

  Since $\forall X(\, p \leftarrow q(X) \wedge r(X)\,) \equiv p \leftarrow \exists X(q(X) \wedge r(X))$

Proving Properties of LR programs

LR programs:

- linear constraints on $\mathcal{R}/\mathcal{Q}$
- linear recursion + negation
- no existential variables

Example:

P: $member(X,[Y|L]) \leftarrow X = Y$
$member(X,[Y|L]) \leftarrow member(X,L)$

$\varphi: \ \forall L \exists U \forall X\, (member(X,L) \rightarrow X \leq U)$

Checking $M(P) \models \varphi$, for any LR-program P and closed formula φ, is undecidable (Peano arithmetic can be encoded)

Quantifier elimination cannot be algorithmic

Proving Properties of LR programs

LR programs:

- linear constraints on $\mathcal{R}/\mathcal{Q}$
- linear recursion + negation
- no existential variables

Example:

P: $member(X, [Y|L]) \leftarrow X = Y$
$member(X, [Y|L]) \leftarrow member(X, L)$

$\varphi: \ \forall L \exists U \forall X \, (member(X, L) \rightarrow X \leq U)$

Checking $M(P) \models \varphi$, for any LR-program P and closed formula φ, is undecidable (Peano arithmetic can be encoded)

Quantifier elimination cannot be algorithmic

QE for CLP(Rlin) via Program Transformation

Step 1. Using a variant of Lloyd-Topor transformation,
we obtain a clause form CF for φ s.t. $M(P) \models \varphi$ iff $M(P \cup CF) \models f$

$$\forall L \quad \exists U \quad \forall Y\ (\text{member}(Y, L) \rightarrow Y \leq U)$$

$$p \leftarrow \underbrace{\neg\ \exists L\ \underbrace{\neg\ \exists U\ \neg\ \underbrace{\exists Y(\text{member}(Y, L) \wedge Y > U)}_{s}}_{r}}_{q}$$

CF:

D_4: $p \leftarrow \neg q$
D_3: $q \leftarrow list(L) \wedge \neg r(L)$
D_2: $r(L) \leftarrow list(L) \wedge \neg s(L, U)$
D_1: $s(L, U) \leftarrow Y > U \wedge list(L) \wedge member(Y, U)$

- stratified
- non LR-clauses (existential variables, nonlinear)

Resolution does not terminate on the query ***p***

Quantifiers can be eliminated by deriving LR-programs

QE for CLP(Rlin) via Program Transformation

Step 1. Using a variant of Lloyd-Topor transformation, we obtain a clause form CF for φ s.t. $M(P) \models \varphi$ iff $M(P \cup CF) \models f$

$$\forall L \quad \exists U \quad \forall Y\ (\text{member}(Y,L) \rightarrow Y \leq U)$$

$$p \leftarrow \neg\ \underbrace{\exists L\ \neg\ \underbrace{\exists U\ \neg\ \underbrace{\exists Y(\text{member}(Y,L) \wedge Y > U)}_{s}}_{r}}_{q}$$

CF:

D_4: $p \leftarrow \neg q$

D_3: $q \leftarrow list(L) \wedge \neg r(L)$

D_2: $r(L) \leftarrow list(L) \wedge \neg s(L, U)$

D_1: $s(L,U) \leftarrow Y > U \wedge list(L) \wedge member(Y, U)$

- stratified
- non LR-clauses (existential variables, nonlinear)

Resolution does not terminate on the query p

Quantifiers can be eliminated by deriving LR-programs

QE for CLP(Rlin) via Program Transformation

Step 1. Using a variant of Lloyd-Topor transformation, we obtain a clause form CF for φ s.t. $M(P) \models \varphi$ iff $M(P \cup CF) \models f$

$$\forall L \quad \exists U \quad \forall Y\ (\text{member}(Y, L) \rightarrow Y \leq U)$$

$$p \leftarrow \underbrace{\neg\ \exists L\ \underbrace{\neg\ \exists U\ \neg\ \underbrace{\exists Y(\text{member}(Y, L) \wedge Y > U)}_{s}}_{r}}_{q}$$

CF:

D_4: $p \leftarrow \neg q$
D_3: $q \leftarrow list(L) \wedge \neg r(L)$
D_2: $r(L) \leftarrow list(L) \wedge \neg s(L, U)$
D_1: $s(L, U) \leftarrow Y > U \wedge list(L) \wedge member(Y, U)$

- stratified
- non LR-clauses (existential variables, nonlinear)

Resolution does not terminate on the query p

Quantifiers can be eliminated by deriving LR-programs

QE for CLP(Rlin) via Program Transformation

Step 1. Using a variant of Lloyd-Topor transformation, we obtain a clause form CF for φ s.t. $M(P) \models \varphi$ iff $M(P \cup CF) \models f$

$$\forall L \quad \exists U \quad \forall Y\, (\text{member}(Y, L) \rightarrow Y \leq U)$$

$$\boldsymbol{p} \leftarrow \underbrace{\neg\ \exists L\ \underbrace{\neg\ \exists U\ \underbrace{\neg\ \exists Y(\text{member}(Y, L) \wedge Y > U)}_{\boldsymbol{s}}}_{\boldsymbol{r}}}_{\boldsymbol{q}}$$

CF:

D_4: $\boldsymbol{p} \leftarrow \neg \boldsymbol{q}$
D_3: $\boldsymbol{q} \leftarrow \mathit{list}(L) \wedge \neg \boldsymbol{r}(L)$
D_2: $\boldsymbol{r}(L) \leftarrow \mathit{list}(L) \wedge \neg \boldsymbol{s}(L, U)$
D_1: $\boldsymbol{s}(L, U) \leftarrow Y > U \wedge \mathit{list}(L) \wedge \mathit{member}(Y, U)$

- stratified
- non LR-clauses (existential variables, nonlinear)

Resolution does not terminate on the query $\boldsymbol{p}$

Quantifiers can be eliminated by deriving LR-programs

QE for CLP(Rlin) via Program Transformation - Cont.

Step 2. Apply a strategy that combines unfold/fold transformations and constraint reasoning on $\mathcal{R}_{lin}$

Goal: from $P \cup CF$ derive a propositional program Prop s.t. $M(P \cup CF) \models f$ iff $M(Prop) \models f$

Prop:

$$p \leftarrow \neg\, q$$
$$q \leftarrow newp$$
$$newp \leftarrow newp$$

As a consequence, $M(P) \models \varphi$ iff $M(Prop) \models p$

The transformation from $P \cup CF$ to Prop consists in eliminating all existential variables from CF

QE for CLP(Rlin) via Program Transformation - Cont.

Step 2. Apply a strategy that combines unfold/fold transformations and constraint reasoning on $\mathcal{R}_{lin}$

Goal: from $P \cup CF$ derive a propositional program Prop s.t. $M(P \cup CF) \models f$ iff $M(Prop) \models f$

Prop:

$$\begin{aligned} p &\leftarrow \neg\, q \\ q &\leftarrow newp \\ newp &\leftarrow newp \end{aligned}$$

As a consequence, $M(P) \models \varphi$ iff $M(Prop) \models p$

The transformation from $P \cup CF$ to Prop consists in eliminating all existential variables from CF

The Transformational Proof Method

D_1: $s(L,U) \leftarrow X > U \wedge list(L) \wedge member(X,L)$

The Transformational Proof Method

D_1: s(L,U) $\leftarrow$ X > U $\wedge$ list(L) $\wedge$ member(X,L)

Unfold: s([X|T],U) $\leftarrow$ X > U $\wedge$ list(T)

s([Y|T],U) $\leftarrow$ X > U $\wedge$ list(T) $\wedge$ member (X,T)

The Transformational Proof Method

D_1: $s(L,U) \leftarrow X > U \wedge list(L) \wedge member(X,L)$

Unfold: $s([X|T],U) \leftarrow X > U \wedge list(T)$

$s([Y|T],U) \leftarrow X > U \wedge list(T) \wedge member(X,T)$

Fold: $s([X|T],U) \leftarrow X > U \wedge list(T)$

$s([X|T],U) \leftarrow s(T,U)$ LR-clauses

The Transformational Proof Method

D₁: $s(L,U) \leftarrow X > U \wedge list(L) \wedge member(X,L)$

Unfold: $s([X|T],U) \leftarrow X > U \wedge list(T)$

$s([Y|T],U) \leftarrow X > U \wedge list(T) \wedge member(X,T)$

Fold: $s([X|T],U) \leftarrow X > U \wedge list(T)$

$s([X|T],U) \leftarrow s(T,U)$ LR-clauses

D_4: $p \leftarrow \neg q$

D_3: $q \leftarrow list(L) \wedge \neg r(L)$

D_2: $r(L) \leftarrow list(L) \wedge \neg s(L, U)$

$\mathbf{D_1}$: $s([X|T],U) \leftarrow X > U \wedge list(T)$

$s([X|T],U) \leftarrow s(T,U)$

The Transformational Proof Method

D_2: $r(L) \leftarrow list(L) \wedge \neg s(L,U)$

The Transformational Proof Method

D_2: $r(L) \leftarrow \boxed{list(L)} \wedge \boxed{\neg\, s(L,U)}$

Unfold: $r([\,])$
$r([X|Xs]) \leftarrow X \leq U \wedge list(Xs) \wedge \neg\, s(Xs,U)$

The Transformational Proof Method

D_2: $r(L) \leftarrow list(L) \wedge \neg s(L,U)$

Unfold: $r([\,])$

$r([X|Xs]) \leftarrow X \leq U \wedge list(Xs) \wedge \neg s(Xs,U)$ BAD folding

The Transformational Proof Method

D₂: $r(L) \leftarrow list(L) \land \neg s(L,U)$

Unfold: $r([\,])$
$r([X|Xs]) \leftarrow X \leq U \land list(Xs) \land \neg s(Xs,U)$

Define: $new_1(X,L) \leftarrow X \leq U \land list(L) \land \neg s(L,U)$

The Transformational Proof Method

D_2: $r(L) \leftarrow list(L) \wedge \neg s(L,U)$

Unfold: $r([\,])$
$r([X|Xs]) \leftarrow X \leq U \wedge list(Xs) \wedge \neg s(Xs,U)$

Define: $new_1(X,L) \leftarrow X \leq U \wedge list(L) \wedge \neg s(L,U)$

Fold: $r([\,]) \leftarrow$
$r([X|Xs]) \leftarrow new_1(X,Xs)$

LR-clauses

The Transformational Proof Method

D_2: $r(L) \leftarrow list(L) \land \neg s(L,U)$

Unfold: $r([\,])$

$r([X|Xs]) \leftarrow X \leq U \land list(Xs) \land \neg s(Xs,U)$

Define: $new_1(X,L) \leftarrow X \leq U \land \boxed{list(L)} \land \boxed{\neg s(L,U)}$

Fold: $r([\,]) \leftarrow$

$r([X|Xs]) \leftarrow new_1(X,Xs)$

LR-clauses

Unfold: $new_1(X,[\,]) \leftarrow$

$new_1(X,[Y|Ys]) \leftarrow X \leq U \land Y \leq U \land list(Ys) \land \neg s(Ys,U)$

The Transformational Proof Method

D_2: $r(L) \leftarrow list(L) \wedge \neg s(L,U)$

Unfold: $r([\,])$

$r([X|Xs]) \leftarrow X \leq U \wedge list(Xs) \wedge \neg s(Xs,U)$

Define: $new_1(X,L) \leftarrow X \leq U \wedge list(L) \wedge \neg s(L,U)$

Fold: $r([\,]) \leftarrow$

$r([X|Xs]) \leftarrow new_1(X,Xs)$ LR-clauses

Unfold: $new_1(X,[\,]) \leftarrow$

$new_1(X,[Y|Ys]) \leftarrow \underline{X \leq U \wedge Y \leq U} \wedge list(Ys) \wedge \neg s(Ys,U)$

$\equiv (Y<X \wedge X \leq U) \vee (X \leq Y \wedge Y \leq U)$ (linear order)

The Transformational Proof Method

D_2: $r(L) \leftarrow list(L) \wedge \neg s(L,U)$

Unfold: $r([\,])$

$r([X|Xs]) \leftarrow X \leq U \wedge list(Xs) \wedge \neg s(Xs,U)$

Define: $new_1(X,L) \leftarrow X \leq U \wedge list(L) \wedge \neg s(L,U)$

Fold: $r([\,]) \leftarrow$

$r([X|Xs]) \leftarrow new_1(X,Xs)$ LR-clauses

Unfold: $new_1(X,[\,]) \leftarrow$

$new_1(X,[Y|Ys]) \leftarrow \underline{X \leq U \wedge Y \leq U} \wedge list(Ys) \wedge \neg s(Ys,U)$

$\equiv (Y < X \wedge X \leq U) \vee (X \leq Y \wedge Y \leq U)$ (linear order)

Replace: $new_1(X,[Y|Ys]) \leftarrow Y < X \wedge X \leq U \wedge list(Ys) \wedge \neg s(Ys,U)$

$new_1(X,[Y|Ys]) \leftarrow X \leq Y \wedge Y \leq U \wedge list(Ys) \wedge \neg s(Ys,U)$

The Transformational Proof Method

D_2: $r(L) \leftarrow list(L) \wedge \neg s(L,U)$

Unfold: $r([\,])$

$r([X|Xs]) \leftarrow X \leq U \wedge list(Xs) \wedge \neg s(Xs,U)$

Define: $new_1(X,L) \leftarrow \boxed{X \leq U \wedge list(L) \wedge \neg s(L,U)}$

Fold: $r([\,]) \leftarrow$

$r([X|Xs]) \leftarrow new_1(X,Xs)$ LR-clauses

Unfold: $new_1(X,[\,]) \leftarrow$

$new_1(X,[Y|Ys]) \leftarrow X \leq U \wedge Y \leq U \wedge list(Ys) \wedge \neg s(Ys,U)$

$\equiv (Y<X \wedge X \leq U) \vee (X \leq Y \wedge Y \leq U)$ (linear order)

Replace: $new_1(X,[Y|Ys]) \leftarrow Y<X \wedge \boxed{X \leq U \wedge list(Ys) \wedge \neg s(Ys,U)}$

$new_1(X,[Y|Ys]) \leftarrow X \leq Y \wedge \boxed{Y \leq U \wedge list(Ys) \wedge \neg s(Ys,U)}$

Fold: $new_1(X,[\,]) \leftarrow$

$new_1(X,[Y|Ys]) \leftarrow Y<X \wedge \boxed{new_1(X,Ys)}$

$new_1(X,[Y|Ys]) \leftarrow X \leq Y \wedge \boxed{new_1(Y,Ys)}$ LR-clauses

The Transformational Proof Method

D_2: $r(L) \leftarrow list(L) \wedge \neg s(L,U)$

Unfold:
$r([\,])$
$r([X|Xs]) \leftarrow X \leq U \wedge list(Xs) \wedge \neg s(Xs,U)$

Define: $new_1(X,L) \leftarrow X \leq U \wedge list(L) \wedge \neg s(L,U)$

Fold:
$r([\,]) \leftarrow$
$r([X|Xs]) \leftarrow new_1(X,Xs)$

LR-clauses

Unfold:
$new_1(X,[\,]) \leftarrow$
$new_1(X,[Y|Ys]) \leftarrow X \leq U \wedge Y \leq U \wedge list(Ys) \wedge \neg s(Ys,U)$

$\equiv (Y<X \wedge X \leq U) \vee (X \leq Y \wedge Y \leq U)$ (linear order)

Replace:
$new_1(X,[Y|Ys]) \leftarrow Y<X \wedge X \leq U \wedge list(Ys) \wedge \neg s(Ys,U)$
$new_1(X,[Y|Ys]) \leftarrow X \leq Y \wedge Y \leq U \wedge list(Ys) \wedge \neg s(Ys,U)$

Fold:
$new_1(X,[\,]) \leftarrow$
$new_1(X,[Y|Ys]) \leftarrow Y<X \wedge \boxed{new_1(X,Ys)}$
$new_1(X,[Y|Ys]) \leftarrow X \leq Y \wedge \boxed{new_1(Y,Ys)}$

LR-clauses

The Transformational Proof Method

D_4: $\boldsymbol{p} \leftarrow \neg\, \boldsymbol{q}$

D_3: $\boldsymbol{q} \leftarrow list(L) \wedge \neg\, \boldsymbol{r}(L)$

D_2: $r([\,]) \leftarrow$
$r([X|Xs]) \leftarrow new_1(X,Xs)$

$new_1(X,[\,]) \leftarrow$
$new_1(X,[Y|Ys]) \leftarrow Y<X \wedge new_1(X,Ys)$
$new_1(X,[Y|Ys]) \leftarrow X \leq Y \wedge new_1(Y,Ys)$

D_1: $s([X|T],U) \leftarrow X > U \wedge list(T)$
$s([X|T],U) \leftarrow s(T,U)$

The Transformational Proof Method

D₄: $p \leftarrow \neg\, q$

D₃: $q \leftarrow new_2$

$new_2 \leftarrow new_2$

D₂: $r([\,]) \leftarrow$
$r([X|Xs]) \leftarrow new_1(X,Xs)$

$new_1(X,[\,]) \leftarrow$
$new_1(X,[Y|Ys]) \leftarrow Y<X \wedge new_1(X,Ys)$
$new_1(X,[Y|Ys]) \leftarrow X \leq Y \wedge new_1(Y,Ys)$

D₁: $s([X|T],U) \leftarrow X > U \wedge list(T)$
$s([X|T],U) \leftarrow s(T,U)$

Using:

- domain axioms
- quantifier elimination on constraints

The Transformational Proof Method

D$_4$: $p \leftarrow \neg\, q$

D$_3$: $q \leftarrow new_2$

$new_2 \leftarrow new_2$

D$_2$: $r([\,]) \leftarrow$
$r([X|Xs]) \leftarrow new_1(X,Xs)$

$new_1(X,[\,]) \leftarrow$
$new_1(X,[Y|Ys]) \leftarrow Y{<}X \wedge new_1(X,Ys)$
$new_1(X,[Y|Ys]) \leftarrow X{\leq}Y \wedge new_1(Y,Ys)$

D$_1$: $s([X|T],U) \leftarrow X > U \wedge list(T)$
$s([X|T],U) \leftarrow s(T,U)$

Using:

- domain axioms
- quantifier elimination on constraints

Since $M(Prop) \models p$, we conclude $M(P) \models \varphi$

Some Observations

Summary:

- a heuristic
- a decision procedure for wS1S

Challenge:

property driven, automatic inference of needed axioms

Optimizing Test-case Generation

Filter Promotion

N-Queens

$$
\begin{array}{lll}
\text{queens}(X,C) & \leftarrow & \text{generate}(N,C) \wedge \text{check}(C) \\
\text{generate}(N,C) & \leftarrow & \text{permutation}(N,C) \\
\text{check}([\,]) & \leftarrow & \\
\text{check}([Q|Qs]) & \leftarrow & \neg\,\text{attack}(Q,Qs) \wedge \\
 & & \text{check}(Qs)
\end{array}
$$

Filter Promotion

N-Queens

queens(X,C) ← generate(N,C) ∧ check(C)

generate(N,C) ← permutation(N,C)

check([ ]) ←
check([Q|Qs]) ← ¬ attack(Q,Qs) ∧
check(Qs)

Q			
		Q	
	Q		
			Q

X

Filter Promotion

N-Queens

$$\text{queens}(X,C) \leftarrow \text{generate}(N,C) \wedge \text{check}(C)$$

$$\text{generate}(N,C) \leftarrow \text{permutation}(N,C)$$

$$\text{check}([\,]) \leftarrow$$
$$\text{check}([Q|Qs]) \leftarrow \neg\, \text{attack}(Q,Qs) \wedge \text{check}(Qs)$$

		Q	
Q			
			Q
	Q		

✓

Filter Promotion

N-Queens

queens(X,C) ← generate(N,C) ∧ check(C)

generate(N,C) ← permutation(N,C)

check([ ]) ←
check([Q|Qs]) ← ¬ attack(Q,Qs) ∧
check(Qs)

		Q	
Q			
			Q
	Q		

✓

queens(X,C) ← queens(N,[ ],C)

queens(0,Qs,Qs) ←
queens(N,SafeQs,Qs) ← place(SafeQs,**Q**) ∧
¬ attack(**Q**,SafeQs) ∧ ⟸ promoted
M>0 ∧ M is N-1 ∧
queens(M,[**Q**|SafeQs],Qs)

Filter Promotion

N-Queens

```
queens(X,C)     ← generate(N,C) ∧ check(C)
generate(N,C) ← permutation(N,C)
check([ ])        ←
check([Q|Qs])  ← ¬ attack(Q,Qs) ∧
                        check(Qs)
```

		Q	
Q			
			Q
	Q		

✓

```
queens(X,C)                  ← queens(N,[ ],C)
queens(0,Qs,Qs)            ←
queens(N,SafeQs,Qs) ← place(SafeQs,Q) ∧
                                   ¬ attack(Q,SafeQs) ∧
                                   M>0 ∧ M is N-1 ∧
                                   queens(M,[Q|SafeQs],Qs)
```

		Q	
Q			
			Q

Filter Promotion

N-Queens

queens(X,C) ← generate(N,C) ∧ check(C)

generate(N,C) ← permutation(N,C)

check([ ]) ←
check([Q|Qs]) ← ¬ attack(Q,Qs) ∧
check(Qs)

		Q	
Q			
			Q
	Q		

✓

queens(X,C) ← queens(N,[ ],C)

queens(0,Qs,Qs) ←
queens(N,SafeQs,Qs) ← place(SafeQs,**Q**) ∧
¬ attack(**Q**,SafeQs) ∧
M>0 ∧ M is N-1 ∧
queens(M,[**Q**|SafeQs],Qs)

		Q	
Q			
			Q
Q			

✗

Filter Promotion

N-Queens

```
queens(X,C)      ← generate(N,C) ∧ check(C)
generate(N,C)    ← permutation(N,C)
check([ ])       ←
check([Q|Qs])    ← ¬ attack(Q,Qs) ∧
                   check(Qs)
```

		Q	
Q			
			Q
	Q		

✓

```
queens(X,C)            ← queens(N,[ ],C)
queens(0,Qs,Qs)        ←
queens(N,SafeQs,Qs)    ← place(SafeQs,Q) ∧
                         ¬ attack(Q,SafeQs) ∧
                         M>0 ∧ M is N-1 ∧
                         queens(M,[Q|SafeQs],Qs)
```

		Q	
Q			
			Q
	Q		

✓

Filter Promotion

N-Queens

```
queens(X,C)      ← generate(N,C) ∧ check(C)
generate(N,C)    ← permutation(N,C)
check([ ])       ←
check([Q|Qs])    ← ¬ attack(Q,Qs) ∧
                   check(Qs)
```

		Q	
Q			
			Q
	Q		

✓

$\downarrow_*$ **Derive automatically, by transformation**

```
queens(X,C)            ← queens(N,[ ],C)
queens(0,Qs,Qs)        ←
queens(N,SafeQs,Qs)    ← place(SafeQs,Q) ∧
                         ¬ attack(Q,SafeQs) ∧
                         M>0 ∧ M is N-1 ∧
                         queens(M,[Q|SafeQs],Qs)
```

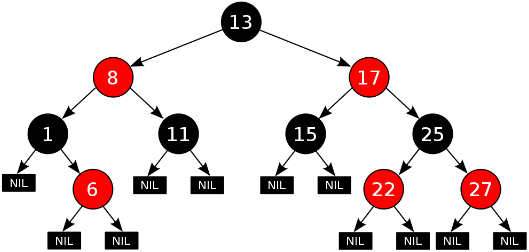
		Q	
Q			
			Q
	Q		

Enumeration of Complex Data-Structures

Red-Black Trees

Nodes are:

- colored (red or black)
- marked (by a key)



A binary tree satisfying the following invariants:

(I_1) **red** nodes have **black children**

(I_2) every **root-to-leaf path** has the same **number of black nodes**

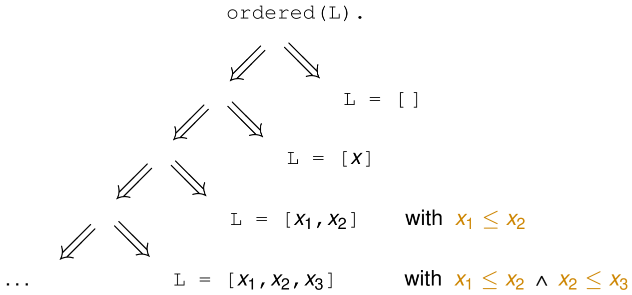
(I_3) keys are **ordered** left-to-right

imperative vs **declarative** languages for test-case generation

CLP Evaluation for Test Generation

ordered([])
ordered([x])
ordered([x_1, x_2 | L]) $\leftarrow \underbrace{x_1 \leq x_2}_{\text{solver for } \mathcal{T}} \wedge \underbrace{\texttt{ordered}([x_2 \mid L])}_{\text{resolution}}$

As a generator:



- Constraint-based
  [DeMillo-Offutt '91, Meudec (ATGen) '01, Euclide '09]
- Constraint Logic Programming -based
  [PathCrawler '05, Charreteur-Botella-Gotlieb '01, jPET '11]

A CLP-based Encoding of Red-Black Trees

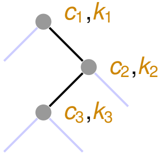
```
rbtree(T,MinSize,MaxSize,NumKeys) ←
    % Preamble
      ...DOMAINS...,
    % Symbolic Definition
      lbt(T,S,Keys,[]),                  % data structure shape
      pi(T,D), ci(T,Colors,[]),          % filters
      ordered(T,0,NumKeys),              % filters
    % Instantiation
      fd_labeling(Keys), fd_labeling(Colors).
```

lbt(T,S,Keys,[])	if	**T** is labeled binary tree with **S** nodes
pi(T,D)	if	the tree **T** satisfies the **path invariant**
ci(T,Colors)	if	the tree **T** satisfies the **color invariant**
ordered(T,0,NumKeys)	if	the labels (keys) in **T** are **ordered** left-to-right
fd_labeling(X)	if	the variable **X** is instantiated to a feasible value

Shape Rejection

`Tree ::= e | Color × Key × Tree × Tree` with `Color` in $\{0, 1\}$ (red, black) and `Key` in $\{0, \ldots, \text{MaxKey}\}$

Size 3 (a possible solution):



structure shape

$\wedge$

$$c_1 = c_1 + c_2 \;\wedge\; c_1 + c_2 = c_1 + c_2 + c_3$$

path invariant

$\wedge$

$$c_1 + c_2 > 0 \;\wedge\; c_2 + c_3 > 0$$

color invariant

$\wedge$

$$k_1 < k_2 \;\wedge\; k_1 < k_3 \;\wedge\; k_3 < k_2$$

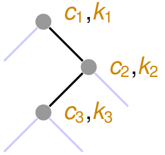
ordering

Shape Rejection

`Tree ::= e | Color` $\times$ `Key` $\times$ `Tree` $\times$ `Tree`

with `Color` in $\{0, 1\}$ (red, black)
and `Key` in $\{0, \ldots, \text{MaxKey}\}$

Size 3 (a possible solution):



$\bigwedge$

$$c_1 = c_1 + c_2 \;\wedge\; c_1 + c_2 = c_1 + c_2 + c_3$$

$\bigwedge$

$$c_1 + c_2 > 0 \;\wedge\; c_2 + c_3 > 0$$

$\bigwedge$

$$k_1 < k_2 \;\wedge\; k_1 < k_3 \;\wedge\; k_3 < k_2$$

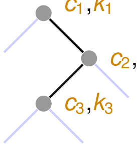
structure shape | path invariant | color invariant | ordering

lbt(T,S,Keys,[ ]) $\wedge$ pi(T,D) $\wedge$ ci(T,Colors) $\wedge$ ordered(T,...)

Shape Rejection

`Tree ::= e | Color × Key × Tree × Tree` with **Color** in $\{0, 1\}$ (red, black) and **Key** in $\{0, \ldots, \text{MaxKey}\}$

Size 3 (a possible solution):



$\bigwedge$

$$c_1 = c_1 + c_2 \;\wedge\; c_1 + c_2 = c_1 + c_2 + c_3$$

$\bigwedge$

$$c_1 + c_2 > 0 \;\wedge\; c_2 + c_3 > 0$$

$\bigwedge$

$$k_1 < k_2 \;\wedge\; k_1 < k_3 \;\wedge\; k_3 < k_2$$

UNFEASIBLE

structure shape — path invariant — color invariant — ordering

lbt(T,S,Keys,[ ]) $\wedge$ pi(T,D) $\wedge$ ci(T,Colors) $\wedge$ ordered(T,...)

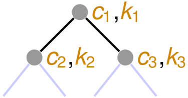
No instantiation possible $\Rightarrow c_2 = 0 \wedge c_3 = 0$

Shape Concretization

`Tree ::= e |` **Color** × **Key** × `Tree` × `Tree`

with **Color** in $\{0, 1\}$ (red, black)
and **Key** in $\{0, \ldots, \text{MaxKey}\}$

Size 3 (another solution) :



$$\bigwedge \quad c_1 + c_2 = c_1 + c_3 \quad \bigwedge \quad \begin{array}{c} c_1 + c_2 > 0 \\ \wedge \\ c_1 + c_3 > 0 \end{array} \quad \bigwedge \quad \begin{array}{c} k_2 < k_1 \\ \wedge \\ k_1 < k_3 \end{array}$$

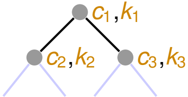
FEASIBLE

Instantiations:

Shape Concretization

`Tree ::= e | Color × Key × Tree × Tree` with **`Color`** in $\{0, 1\}$ (red, black) and **`Key`** in $\{0, \ldots, \text{MaxKey}\}$

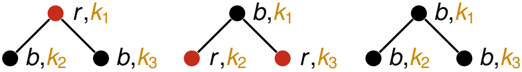
Size 3 (another solution) :



$$\bigwedge \quad c_1 + c_2 = c_1 + c_3 \quad \bigwedge \quad \begin{matrix} c_1 + c_2 > 0 \\ \wedge \\ c_1 + c_3 > 0 \end{matrix} \quad \bigwedge \quad \begin{matrix} k_2 < k_1 \\ \wedge \\ k_1 < k_3 \end{matrix}$$

FEASIBLE

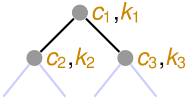
Instantiations:



Shape Concretization

`Tree ::= e | Color × Key × Tree × Tree` with **Color** in $\{0, 1\}$ (red, black) and **Key** in $\{0, \dots, \text{MaxKey}\}$

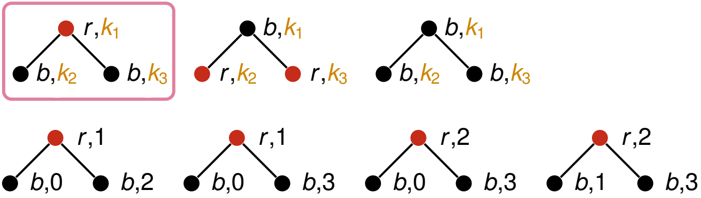
Size 3 (another solution) :



$$\wedge \quad c_1 + c_2 = c_1 + c_3 \quad \wedge \quad \begin{matrix} c_1 + c_2 > 0 \\ \wedge \\ c_1 + c_3 > 0 \end{matrix} \quad \wedge \quad \begin{matrix} k_2 < k_1 \\ \wedge \\ k_1 < k_3 \end{matrix}$$

FEASIBLE

Instantiations:

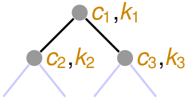


Shape Concretization

`Tree ::= e |` **Color** × **Key** × `Tree` × `Tree`

with **Color** in $\{0, 1\}$ (red, black)
and **Key** in $\{0, \ldots, \text{MaxKey}\}$

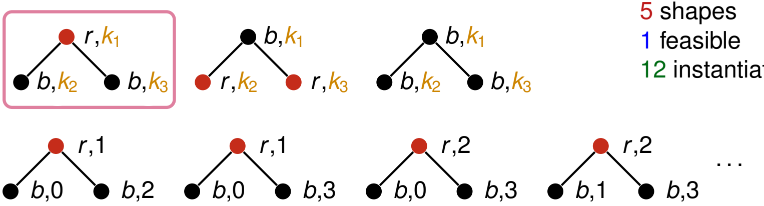
Size 3 (another solution) :



$\bigwedge \quad c_1 + c_2 = c_1 + c_3 \quad \bigwedge \quad \begin{matrix} c_1 + c_2 > 0 \\ \wedge \\ c_1 + c_3 > 0 \end{matrix} \quad \bigwedge \quad \begin{matrix} k_2 < k_1 \\ \wedge \\ k_1 < k_3 \end{matrix}$

FEASIBLE

Instantiations:

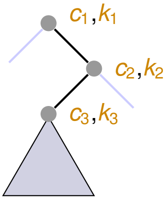


5 shapes
1 feasible
12 instantiations

we let the solver choose an instantiation order to minimize backtracking

Optimizing Generators by Transformation

Size 20



No way of placing the remaining 17 nodes to build a feasible tree

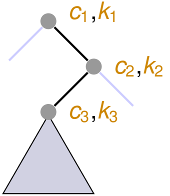
$c_1 = c_1 + c_2 \wedge c_1 = c_1 + c_2 + c_3 + X \wedge$
$X \geq 0 \wedge c_1 + c_2 > 0 \wedge c_2 + c_3 > 0$ UNFEASIBLE

Yet, there would be 35357670 **shapes** (and corresponding feasiblity tests)

Idea: apply filter earlier

Optimizing Generators by Transformation

Size 20



No way of placing the remaining
17 nodes to build a feasible tree

$$c_1 = c_1 + c_2 \wedge c_1 = c_1 + c_2 + c_3 + X \wedge$$
$$X \geq 0 \wedge c_1 + c_2 > 0 \wedge c_2 + c_3 > 0$$

UNFEASIBLE

Yet, there would be 35357670 **shapes**
(and corresponding feasiblity tests)

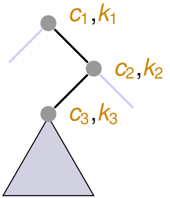
Idea: apply filter earlier

The Synchronization transformation strategy

- automates optimization
- **filter promotion + tupling + folding** (local optimization inductively propagated)
- force the generator to have the desired behavior
- reduces don't care nondeterminism (less **failures**)
- guided by the **inductive traversal** of (a slice of) the data-structure

Optimizing Generators by Transformation

Size 20



No way of placing the remaining
17 nodes to build a feasible tree

$c_1 = c_1 + c_2 \wedge c_1 = c_1 + c_2 + c_3 + X \wedge$
$X \geq 0 \wedge c_1 + c_2 > 0 \wedge c_2 + c_3 > 0$ UNFEASIBLE

Yet, there would be 35357670 **shapes**
(and corresponding feasiblity tests)

Idea: apply filter earlier

The Synchronization transformation strategy

- automates optimization
- **filter promotion + tupling + folding** (local optimization inductively propagated)
- force the generator to have the desired behavior
- reduces don't care nondeterminism (less **failures**)
- guided by the **inductive traversal** of (a slice of) the data-structure

Related techniques : compiling control, co-routining

Experiments

Size	***RB Trees***	Time		
		Original	Synchronized	Korat
6	20	0	0	0.47
7	35	0.01	0	0.63
8	64	0.02	0	1.49
9	122	0.08	0	4.51
10	260	0.29	0.01	21.14
11	586	1.07	0.03	116.17
12	1296	3.98	0.06	-
13	2708	14.85	0.12	-
14	5400	55.77	0.26	-
15	10468	-	0.55	-
...	...	...	...	...
20	279264	-	25.90	-

Size = number of nodes
RB Trees = number of structures found
Time = (in seconds) for generating all the structures

Zero means less than 10 ms, and
(-) means more than 200 seconds

Summarizing

- improved **declarativeness**
- heaparrays, disjoint sets, various lists/trees, ...
- baseline good, room for optimization
- a language of **composable**/**refinable** generators
- automatic extraction of CLP generators from **contracts** (a form of program synthesis)
- graph-like structures