Path-based Inductive Synthesis for Program Inversion

Saurabh Srivastava (a)
Sumit Gulwani (b)
Swarat Chaudhuri (c)
Jeffrey S. Foster (d)

(a) University of California, Berkeley
(b) Microsoft Research, Redmond
(c) Rice University
(d) University of Maryland, College Park
Program Inversion as Synthesis

• Task
  Given a program $P$, synthesis $P^{-1}$ such that $P^{-1}(P(x)) = x$

• Motivation: Many common program/inverse pairs
  • Compress/decompress, insert/delete, lossless encode/decode, encrypt/decrypt, rollback, many more
  • Only having to write one increases productivity, reduces bugs

• Problem
  • Existing synthesis techniques not well-suited for inversion
  • Dedicated inversion techniques limited in scope
PINS: Path-based Inductive Synthesis

- **Specification**
  - Program to be inverted
  - Template hints: Control flow, and expressions, predicates
  - Functional requirement: Program + Inverse = Identity

- **Engine**: SMT solver (Z3)

- **Algorithm**: Inspired by testing
  - Explore path through program + template
  - Ask engine for instantiations on path to match spec
  - Iterate, refining space
Small path-bound hypothesis

“Program behavior can be summarized by examining a carefully chosen, small, finite set of paths”

- Same hypothesis underlies program testing

- As in testing, two questions:
  1) Which paths?
     - Especially since the template describes “set of programs”
  2) How can we ensure the generated inverse is correct?
     - We check using: manual inspection, testing, bounded verification
Example of templates

In-place run-length encoding:

\[ A = [1,1,1,0,0,2,2,2,2] \]

\[ A = [1,0,2] \]
\[ N = [3,2,4] \]

\[ A' = [1,1,1,0,0,2,2,2,2] \]
Example of templates

In-place run-length encoding:

\[ A = [1,1,1,0,0,2,2,2,2] \]

Original encoder

\[ A = [1,0,2] \]
\[ N = [3,2,4] \]

\[ A' = [1,1,1,0,0,2,2,2,2] \]

\[ \text{assume}(n \geq 0); \]
\[ i, m := 0, 0; \]
\[ \text{// parallel assignment} \]
\[ \text{while } (i < n) \]
\[ r := 1; \]
\[ \text{while } (i+1 < n \text{ and } A[i] = A[i+1]) \]
\[ r, i := r + 1, i+1; \]
\[ A[m], N[m], m, i := A[i], r, m+1, i+1; \]
In-place run-length encoding:

\[ A = [1,1,1,0,0,2,2,2,2] \]

Assume \( n \geq 0 \);

\[ i, m := 0, 0; \quad \text{// parallel assignment} \]

\[ \text{while } (i < n) \]

\[ r := 1; \]

\[ \text{while } (i+1 < n \&\& A[i] = A[i+1]) \]

\[ r, i := r + 1, i+1; \]

\[ A[m], N[m], m, i := A[i], r, m+1, i+1; \]

Original encoder

\[ A = [1,0,2] \]

\[ N = [3,2,4] \]

Template decoder

\[ A' = [1,1,1,0,0,2,2,2,2] \]

\[ i', m' := e_1, e_2; \quad \text{// } e_i \in E \]

\[ \text{while } (p_1) \quad \text{// } p_i \in P \]

\[ r' := e_3; \]

\[ \text{while } (p_2) \]

\[ r', i', A' := e_4, e_5, e_6; \]

\[ m' := e_7; \]
Example of templates

In-place run-length encoding:

\[
A = [1,1,1,0,0,2,2,2,2]
\]

\[
assume(n\geq0);
\]
\[
i, m := 0, 0; \quad // parallel assignment
\]
\[
while (i < n)
\]
\[
r := 1;
\]
\[
while (i+1 < n \&\& A[i] = A[i+1])
\]
\[
r, i := r + 1, i+1;
\]
\[
A[m], N[m], m, i := A[i], r, m+1, i+1;
\]

\[
A = [1,0,2]
\]
\[
N = [3,2,4]
\]

\[
i', m' := e1, e2; \quad // e_i \in E
\]
\[
while (p1) \quad // p_i \in P
\]
\[
r' := e3;
\]
\[
while (p2)
\]
\[
r', i', A' := e4, e5, e6;
\]
\[
m' := e7;
\]

\[
E = \{
0, 1, m'+1, m'-1, r'+1, r'-1,
\}
\]
\[
i'+1, i'-1, A'[m']:=A[i'],
\]
\[
A'[i'] := A[m'], N[m']
\]
Example of templates

In-place run-length encoding:

\[
A = [1,1,1,0,0,2,2,2,2] \\
\text{assume}(n \geq 0); \\
i, m := 0, 0; \quad \text{/// parallel assignment} \\
\text{while } (i < n) \\
\quad r := 1; \\
\quad \text{while } (i+1 < n \&\& A[i] = A[i+1]) \\
\quad \quad r, i := r + 1, i+1; \\
\quad A[m], N[m], m, i := A[i], r, m+1, i+1;
\]

Original encoder

\[
A = [1,0,2] \\
N = [3,2,4]
\]

Template decoder

\[
A' = [1,1,1,0,0,2,2,2,2] \\
i', m' := e_1, e_2; \quad \text{/// } e_i \in E \\
\text{while } (p_1) \quad \text{/// } p_i \in P \\
\quad r' := e_3; \\
\quad \text{while } (p_2) \\
\quad \quad r', i', A' := e_4, e_5, e_6; \\
\quad m' := e_7;
\]

\[
E = \{ \\
0, 1, m'+1, m'-1, r'+1, r'-1, \\
i'+1, i'-1, A'[m']:=A[i'], \\
A'[i'] := A[m'], N[m']
\}
\]

\[
P = \{ \\
m'<m, r'>0, A'[i']=A[i'+1]
\}
Example of templates

In-place run-length encoding:

\[ A = [1,1,1,0,0,2,2,2,2] \]

\[ A' = [1,1,1,0,0,2,2,2,2] \]

\[ assume(n>=0); \]
\[ i, m := 0, 0; \] \hspace{1cm} // parallel assignment
\[ while (i < n) \]
\[ r := 1; \]
\[ while (i+1 < n && A[i] = A[i+1]) \]
\[ r, i := r + 1, i+1; \]
\[ A[m], N[m], m, i := A[i], r, m+1, i+1; \]

Original encoder

\[ A = [1,0,2] \]
\[ N = [3,2,4] \]

Template decoder

\[ A' = [1,1,1,0,0,2,2,2,2] \]

Template control flow, expressions \( E \), and predicates \( P \), semi-automatically mined from original

\[ E = \{ \]
\[ 0, 1, m'+1, m'-1, r'+1, r'-1, \]
\[ i'+1, i'-1, A'[m']:=A[i'], \]
\[ A'[i'] := A[m'], N[m'] \] \}

\[ P = \{ \]
\[ m'<m, r'>0, A'[i']=A'[i'+1] \] \}
Symbolic execution of program paths

\begin{verbatim}
assume(n>=0);
i, m := 0, 0;
while (i < n)
    r := 1;
    while (i+1 < n && A[i] = A[i+1])
        r, i := r + 1, i+1;
    A[m], N[m], m, i := A[i], r, m+1, i+1;

i', m' := e₁, e₂;
while (p₁)
    r' := e₃;
    while (p₂)
        r', i', A' := e₄, e₅, e₆;
    m' := e₇;
\end{verbatim}
Symbolic execution of program paths

```
assume(n>=0);
i, m := 0, 0;
while (i < n)
    r := 1;
    while (i+1 < n && A[i] = A[i+1])
        r, i := r + 1, i+1;
    A[m], N[m], m, i := A[i], r, m+1, i+1;

i', m' := e_1, e_2;
while (p_1)
    r' := e_3;
    while (p_2)
        r', i', A' := e_4, e_5, e_6;
m' := e_7;
```
Symbolic execution of program paths

```
assume(n>=0);
i, m := 0, 0;
while (i < n)
    r := 1;
    while (i+1 < n && A[i] = A[i+1])
        r, i := r + 1, i+1;
    A[m], N[m], m, i := A[i], r, m+1, i+1;

i’, m’ := e1, e2;
while (p1)
    r’ := e3;
    while (p2)
        r’, i’, A’ := e4, e5, e6;
        m’ := e7;
```

(n>=0) ∧
i’ = 0 ∧ m’ = 0 ∧
¬ (i’ < n0) ∧
¬ (p1)
⇒ identity
Symbolic execution of program paths

\[
\begin{align*}
\text{assume}(n \geq 0); \\
i, m &:= 0, 0; \\
\text{while } (i < n) \\
&\quad \left[ \\
&\quad \quad r := 1; \\
&\quad \quad \text{while } (i + 1 < n \&\& A[i] = A[i + 1]) \\
&\quad \quad \quad \left[ \\
&\quad \quad \quad \quad r, i := r + 1, i + 1; \\
&\quad \quad \quad A[m], N[m], m, i := A[i], r, m + 1, i + 1; \\
&\quad \quad \left[ \\
&\quad \quad \quad i', m' := e_1, e_2; \\
&\quad \quad \text{while } (p_1) \\
&\quad \quad \quad r' := e_3; \\
&\quad \quad \quad \text{while } (p_2) \\
&\quad \quad \quad \quad r', i', A' := e_4, e_5, e_6; \\
&\quad \quad \quad m' := e_7;
\right]
\right]
\end{align*}
\]
Symbolic execution of program paths

\[ \text{assume}(n \geq 0); \]
\[ i, m := 0, 0; \]
\[ \text{while } (i < n) \]
  \[ r := 1; \]
  \[ \text{while } (i+1 < n \land A[i] = A[i+1]) \]
    \[ r, i := r + 1, i+1; \]
  \[ A[m], N[m], m, i := A[i], r, m+1, i+1; \]
\[ i', m' := e_1, e_2; \]
\[ \text{while } (p_1) \]
  \[ r' := e_3; \]
  \[ \text{while } (p_2) \]
    \[ r', i', A' := e_4, e_5, e_6; \]
  \[ m' := e_7; \]

\( (n^0 \geq 0) \land \\
  i^1 = 0 \land m^1 = 0 \land \\
  \neg (i^1 < n^0) \land \\
  i'^1 = e_1^v \land m'^1 = e_2^v \land \\
  \neg (p_1^v) \Rightarrow \text{identity} \)
Symbolic execution of program paths

\[ \begin{align*}
\text{assume}(n \geq 0); & \quad (n^0 \geq 0) \\
i, m := 0, 0; & \quad i^1 = 0 \land m^1 = 0 \land (i^1 < n^0) \\
\text{while } (i < n) & \quad \Rightarrow \text{ identity}
\end{align*} \]

\[ \begin{align*}
r := 1; & \quad \neg (p_1^v) \\
\text{while } (i+1 < n \land A[i] = A[i+1]) & \quad r', i := r + 1, i+1; \\
A[m], N[m], m, i := A[i], r, m+1, i+1; & \quad \neg (p_2^v) \\
i', m' := e_1, e_2; & \quad m' := e_7;
\end{align*} \]

\[ \begin{align*}
\text{while } (p_1) & \quad r' := e_3; \\
\text{while } (p_2) & \quad (p_1^v) \\
\text{while } (p_2) & \quad (p_1^v') \\
\text{while } (p_2) & \quad (p_1^v'')
\end{align*} \]
Symbolic execution of program paths

assume(n>=0);
i, m := 0, 0;
while (i < n)
    r := 1;
    while (i+1 < n && A[i] = A[i+1])
       r, i := r + 1, i+1;
    A[m], N[m], m, i := A[i], r, m+1, i+1;

i’, m’ := e1, e2;
while (p1)
    r’ := e3;
    while (p2)
       r’, i’, A’ := e4, e5, e6;
    m’ := e7;
Symbolic execution of program paths

\begin{align*}
\text{assume}(n \geq 0); \\
i, m &:= 0, 0; \\
\text{while } (i < n) \\
\quad r &:= 1; \\
\quad &\text{while } (i+1 < n \land A[i] = A[i+1]) \\
\quad &\quad r, i := r + 1, i+1; \\
A[m], N[m], m, i &:= A[i], r, m+1, i+1; \\
i', m' &:= e_1, e_2; \\
\text{while } (p_1) \\
\quad r' &:= e_3; \\
\quad &\text{while } (p_2) \\
\quad &\quad r', i', A' := e_4, e_5, e_6; \\
\quad m' &:= e_7; \\
\end{align*}
Symbolic execution of program paths

assume(n>=0);
i, m := 0, 0;
while (i < n)
r := 1;
while (i+1 < n && A[i] = A[i+1])
\[ r, i := r + 1, i+1; \]
A[m], N[m], m, i := A[i], r, m+1, i+1;
i', m' := e_1, e_2;
while (p_1)
r' := e_3;
while (p_2)
\[ r', i', A' := e_4, e_5, e_6; \]
m' := e_7;

\[ (n^0>=0) \land 
   i^1 = 0 \land m^1 = 0 \land 
   \neg (i^1 < n^0) \land 
   i^{1'}=e_1^v \land m^{1'}=e_2^v \land 
   \neg (p_1^v) \Rightarrow \text{identity} \]

\[ (n^0>=0) \land 
   i^1 = 0 \land m^1 = 0 \land 
   \neg (i^1 < n^0) \land 
   i^{1'}=e_1^v \land m^{1'}=e_2^v \land 
   (p_1^v) \land 
   r^{1'}=e_3^v \land 
   \neg (p_2^v') \land 
   m^{1''}=e_7^v'' \land 
   \neg (p_1^{v''}) \Rightarrow \text{identity} \]
assume(n>=0);
i, m := 0, 0;
while (i < n)
    r := 1;
    while (i+1 < n && A[i] = A[i+1])
        r, i := r + 1, i+1;
    A[m], N[m], m, i := A[i], r, m+1, i+1;

i', m' := e₁, e₂;
while (p₁)
    r' := e₃;
    while (p₂)
        r', i', A' := e₄, e₅, e₆;
        m' := e₇;

(n>=0) \land
i' = 0 \land m' = 0 \land
\neg (i' < n) \land
i'' = e₁ \land m'' = e₂ \land
\neg (p₁)

(n>=0) \land
i' = 0 \land m' = 0 \land
\neg (i' < n) \land
i'' = e₁ \land m'' = e₂ \land
(p₁) \land
r'' = e₃ \land
\neg (p₂) \land

m'' = e₇ \land
\neg (p₁) \land

(n0>=0) \land
i' = 0 \land m' = 0 \land
\neg (i' < n0) \land
i'' = e₁ \land m'' = e₂ \land
(p₁)

i'' = e₃ \land
\neg (p₂) \land
m'' = e₇ \land
\neg (p₁)

⇒ identity

Symbolic execution of program paths
Symbolic execution of program paths

\[(n^0 \geq 0) \land i^1 = 0 \land m^1 = 0 \land \neg (i^1 < n^0) \land
\]
\[(i^1 = e_1^v \land m^1 = e_2^v \land \neg (p_1^v)) \Rightarrow \text{identity}\]

\[(n^0 \geq 0) \land i^1 = 0 \land m^1 = 0 \land \neg (i^1 < n^0) \land
\]
\[i^1 = e_1^v \land m^1 = e_2^v \land (p_1^v) \land
\]
\[r^1 = e_3^v \land \neg (p_2^v) \land m^2 = e_7^v \land \neg (p_1^v) \Rightarrow \text{identity}\]

\[(n^0 \geq 0) \land i^1 = 0 \land m^1 = 0 \land
\]
\[(i^1 < n^0) \land r^1 = 1 \land
\]
\[\neg (i^1 + 1 < n^0 \land A[i^1] = A[i^1 + 1]) \land A[m^1] = A[i^1] \land N[m^1] = r^1 \land m^2 = m^1 + 1 \land i^2 = i^1 + 1 \land
\]
\[(i^2 < n^0) \land i^1 = e_1^v \land m^1 = e_2^v \land \neg (p_1^v) \Rightarrow \text{identity}\]
Solving using SMT and SAT

\[(n^0 \geq 0) \land \]
\[i^1 = 0 \land m^1 = 0 \land \]
\[\neg (i^1 < n^0) \land \]
\[i^{i^1} = e_1^V \land m^{i^1} = e_2^V \land \]
\[\neg (p_1^V) \]
\[\Rightarrow \text{identity} \]

\[(n^0 \geq 0) \land \]
\[i^1 = 0 \land m^1 = 0 \land \]
\[\neg (i^1 < n^0) \land \]
\[i^{i^1} = e_1^V \land m^{i^1} = e_2^V \land \]
\[(p_1^V) \land \]
\[r^{i^1} = e_3^V \land \]
\[\neg (p_2^{V''}) \land \]
\[m^{i^2} = e_7^{V''} \land \]
\[\neg (p_1^{V'''}) \Rightarrow \text{identity} \]

\[(n^0 \geq 0) \land \]
\[i^1 = 0 \land m^1 = 0 \land \]
\[(i^1 < n^0) \land \]
\[r^1 = 1 \land \]
\[\neg (i^1 + 1 < n^0 \land A[i^1] = A[i^1 + 1]) \]
\[A[m^1] = A[i^1] \land N[m^1] = r^1 \land m^2 = m^1 + 1 \land i^2 = i^1 + 1 \land \]
\[\neg (i^2 < n^0) \land \]
\[i^{i^1} = e_1^V \land m^{i^1} = e_2^V \land \]
\[\neg (p_1^V) \]
\[\Rightarrow \text{identity} \]
Solving using SMT and SAT

\( \varphi_1(e_1,e_2,p_1) \Rightarrow \text{identity} \)

\( (n^0 \geq 0) \land \\
  i^1 = 0 \land m^1 = 0 \land \\
  (i^1 < n^0) \land \\
  r^1 = 1 \land \\
  \neg (i^1 + 1 < n^0 \land A[i^1] = A[i^1+1]) \\
  A[m^1] = A[i^1] \land N[m^1] = r^1 \land m^2 = m^1 + 1 \land i^2 = i^1 + 1 \\
  \neg (i^2 < n^0) \\
  i^1 = e_1^V \land m^1 = e_2^V \land \\
  \neg (p_1^V) \Rightarrow \text{identity} \)
Solving using SMT and SAT

\[ \varphi_1(e_1, e_2, p_1) \Rightarrow \text{identity} \]

\[ \varphi_2(e_1, e_2, p_1, e_3, e_7, p_2) \Rightarrow \text{identity} \]

\[ (n^0 \geq 0) \land 
   i^1 = 0 \land m^1 = 0 \land 
   (i^1 < n^0) \land 
   r^1 = 1 \land 
   \neg (i^1 + 1 < n^0 \land A[i^1] = A[i^1 + 1]) \land 
   A[m^1] = A[i^1] \land N[m^1] = r^1 \land m^2 = m^1 + 1 \land i^2 = i^1 + 1 \land 
   \neg (i^2 < n^0) \land 
   i^1 = e_1 \land m^1 = e_2 \land 
   \neg (p_1) \Rightarrow \text{identity} \]
Solving using SMT and SAT

\[
\varphi_1(e_1, e_2, p_1) \Rightarrow \text{identity}
\]

\[
\varphi_2(e_1, e_2, p_1, e_3, e_7, p_2) \Rightarrow \text{identity}
\]
Solving using SMT and SAT

\[ \exists e_i, p_j \forall \text{program vars} \]

\[ \varphi_1(e_1, e_2, p_1) \Rightarrow \text{identity} \]

\[ \varphi_2(e_1, e_2, p_1, e_3, e_7, p_2) \Rightarrow \text{identity} \]
Solving using SMT and SAT

\[ \exists e_i, p_j \forall \text{program vars} \]

- Naive approach:
  - Enumerate \( e_i, p_j \) and “validate”
  - Will not scale
    - \( 2^{11} \) to \( 2^{37} \) candidates our experiments
Efficient solving from prior work on verification using SAT/SMT

\[ \exists e_i, p_j \forall \text{vars} \]

\[ \land k \varphi_k(e_1, e_2, p_1) \Rightarrow \text{identity} \]

- Efficient solving strategy:
  - Verification solves \( \exists \text{Invariant} \forall \text{vars} \)
  - Reuse SMT-based verifier technology

- Core idea:
  - Predicates/expressions form a lattice
  - Efficient encoding using lattice instead of enumerating entire domain
  - See prior work in PLDI’09/POPL’10
The PINS Algorithm

\[
C = \text{termination (T)} \\
\text{while (true) \{} \\
\quad \text{solns} = \text{solve (C,P,E,Spec)} \\
\quad \text{if (empty(solns)) fail} \\
\quad \text{if (stabilized(solns)) return solns} \\
\quad s = \text{pickone (solns)} \\
\quad C = C \land \text{directed-path-explore (T,s)} \\
\text{\}}
\]
The PINS Algorithm

C holds the accumulated constraints

\[ C = \text{termination (T)} \]
while (true) {
    solns = solve (C,P,E,Spec)
    if (empty(solns)) fail
    if (stabilized(solns)) return solns
    s = pickone (solns)
    C = C \land \text{directed-path-explore (T,s)}
}

C = \text{termination (T)}
C holds the accumulated constraints

C = termination (T)
while (true) {
    solns = solve (C,P,E,Spec)
    if (empty(solns)) fail
    if (stabilized(solns)) return solns
    s = pickone (solns)
    C = C ∧ directed-path-explore (T,s)
}
The PINS Algorithm

\( C \) holds the accumulated constraints

Initialize with termination cnstr

( Simple linear constraints that ensure that symbolic execution terminates )

If no change to candidate set then they are likely not refutable

\[ C = \text{termination (T)} \]

while (true) {

\( \text{solns} = \text{solve (C,P,E,Spec)} \)

if (empty(\text{solns})) fail

if (stabilized(\text{solns})) return \text{solns}

\( s = \text{pickone (solns)} \)

\( C = C \land \text{directed-path-explore (T,s)} \)

}
The PINS Algorithm

\[ C = \text{termination} \ (T) \]
while (true) {
\[ \text{solns} = \text{solve} \ (C, P, E, \text{Spec}) \]
if (empty(\text{solns})) fail
if (stabilized(\text{solns})) return \text{solns}
\[ s = \text{pickone} \ (\text{solns}) \]
\[ C = C \land \text{directed-path-explore} \ (T, s) \]
}

\( C \) holds the accumulated constraints

Initialize with termination cnstr
(Simple linear constraints that ensure that symbolic execution terminates)

If no change to candidate set then they are likely not refutable
Else use one \( s \) to parameterize next path exploration
The PINS Algorithm

C holds the accumulated constraints

Initialize with termination cnstr

( Simple linear constraints that ensure
  that symbolic execution terminates )

If no change to candidate set then they are likely not refutable

Else use one s to parameterize next path exploration

Explore another path and add its constraint

C = termination (T)

while (true) {
    solns = solve (C,P,E,Spec)
    if (empty(solns)) fail
    if (stabilized(solns)) return solns
    s = pickone (solns)
    C = C ∧ directed-path-explore (T,s)
}
The PINS Algorithm

C = termination (T)

while (true) {
    solns = solve (C,P,E,Spec)
    if (empty(solns)) fail
    if (stabilized(solns)) return solns
    s = pickone (solns)
    C = C ∧ directed-path-explore (T,s)
}

We do not have a way of certifiably saying which remaining solutions are correct and which are not.

So how do we find a path that prunes the space further?

Else use one s to parameterize next path exploration

Explore another path and add its constraint.
Directed path exploration

$2^{|P| (# Pred Holes)} \times |E| (# Expr Holes)$

Template program $T$

Remaining search space

Explored paths
Directed path exploration

\[ 2^{\lvert P \rvert} (\# \text{ Pred Holes}) \times \lvert E \rvert (\# \text{ Expr Holes}) \]
Directed path exploration

\[ 2^{|P|} \times |E| \]

Template program \( T \)

Remaining search space

Explored paths

\( S \checkmark \) \( S \times \)
Directed path exploration

Remaining search space

Template program $T$

$x |E| (\# \text{Expr Holes})$

$2 |P| (\# \text{Pred Holes})$  $\Rightarrow \text{spec}$
Directed path exploration

Template program $T$

Remaining search space

Explored paths

⇒ spec

What if?

$2^{P}$ (number of pred holes) x $E$ (number of expr holes)
Directed path exploration

Remaining search space

Template program $T$

Explored paths

$\Rightarrow$ spec

$T$ instantiated with $S^\checkmark$

$\Rightarrow$ false

What if?

$2 |P| (# Pred Holes) \times |E| (# Expr Holes)$
Directed path exploration

Template program $T$

Explored paths

$\Rightarrow$ spec

$T$ Instantiated with $S\checkmark$

$\Rightarrow$ false

$\Rightarrow$ spec

$|\mathcal{P}|$ (# Pred Holes) $\times |\mathcal{E}|$ (# Expr Holes)

Remaining search space

What if?
Directed path exploration

What if?

Template program $T$

Explored paths

$\Rightarrow$ spec

$T$ Instantiated with $S^\checkmark$

$\Rightarrow$ false

$\Rightarrow$ spec

$\Rightarrow$ false
Directed path exploration

Remaining search space

Template program T

What if?

Explored paths

⇒ spec

T Instantiated with $S_\checkmark$

⇒ false
⇒ spec

$\Rightarrow$ false
⇒ spec

$2 \mid P \mid (# \text{ Pred Holes}) \times \mid E \mid (# \text{ Expr Holes})$
Directed path exploration

- $S_{\vee}$
- $S_{x}$
- $\Rightarrow$ spec
- $\Rightarrow$ false $\Rightarrow$ spec
- $\Rightarrow$ false $\Rightarrow$ spec
Directed path exploration

Pick any solution from remaining space; don’t care about its validity

- $S \checkmark$
- $S \times$

$\Rightarrow$ spec

- $\not\Rightarrow$ false $\Rightarrow$ spec
- $\not\Rightarrow$ false $\not\Rightarrow$ spec
Directed path exploration

Pick any solution from remaining space; don’t care about its validity

Directed path exploration
Instantiate template with picked solution, and now symbolically execute to find feasible path

$S\checkmark$

$S\times$

$\Rightarrow$ spec

$\Rightarrow$ false $\Rightarrow$ spec

$\Rightarrow$ false $\not\Rightarrow$ spec
Program inversion benchmarks

- Three domains
  - Lossless compression
  - Format conversion
  - Arithmetic

- Semi-automatic procedure to extract template T
  - Control-flow derived from original program
  - Expression/predicates mined

- Ran PINS to invert using template T
## Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Search space reduction</th>
<th>Number of Iterations</th>
<th>Time</th>
<th>Manual Check</th>
<th>Model Checker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run length</td>
<td>$2^{25} \rightarrow 1$</td>
<td>7</td>
<td>26s</td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>In place RL</td>
<td>$2^{30} \rightarrow 1$</td>
<td>7</td>
<td>36s</td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>LZ77</td>
<td>$2^{25} \rightarrow 2$</td>
<td>6</td>
<td>1810s</td>
<td>1 of 2 ok</td>
<td>validated</td>
</tr>
<tr>
<td>LZW</td>
<td>$2^{31} \rightarrow 2$</td>
<td>4</td>
<td>150s</td>
<td>2 of 2 ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Base 64</td>
<td>$2^{37} \rightarrow 4$</td>
<td>12</td>
<td>1377s</td>
<td>1 of 4 ok</td>
<td>too complex</td>
</tr>
<tr>
<td>UUencode</td>
<td>$2^{20} \rightarrow 1$</td>
<td>7</td>
<td>34s</td>
<td>ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Pkt Wrap</td>
<td>$2^{20} \rightarrow 1$</td>
<td>6</td>
<td>132s</td>
<td>ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Serialize</td>
<td>$2^{11} \rightarrow 1$</td>
<td>14</td>
<td>55s</td>
<td>ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Sum i</td>
<td>$2^{15} \rightarrow 1$</td>
<td>4</td>
<td>1s</td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>Vector rotate</td>
<td>$2^{16} \rightarrow 1$</td>
<td>3</td>
<td>4s</td>
<td>ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Vector shift</td>
<td>$2^{16} \rightarrow 1$</td>
<td>3</td>
<td>4s</td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>Vector scale</td>
<td>$2^{16} \rightarrow 1$</td>
<td>3</td>
<td>36s</td>
<td>ok</td>
<td>too complex</td>
</tr>
</tbody>
</table>
## Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Search space reduction</th>
<th>Number of Iterations</th>
<th>Time</th>
<th>Manual Check</th>
<th>Model Checker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run length</td>
<td>$2^{25} -&gt; 1$</td>
<td>7</td>
<td>26s</td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>In place RL</td>
<td>$2^{30} -&gt; 1$</td>
<td>7</td>
<td>36s</td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>LZ77</td>
<td>$2^{25} -&gt; 2$</td>
<td></td>
<td></td>
<td>1 of 2 ok</td>
<td>validated</td>
</tr>
<tr>
<td>LZW</td>
<td>$2^{31} -&gt; 2$</td>
<td></td>
<td></td>
<td>2 of 2 ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Base 64</td>
<td>$2^{37} -&gt; 4$</td>
<td></td>
<td>1377s</td>
<td>1 of 4 ok</td>
<td>too complex</td>
</tr>
<tr>
<td>UUencode</td>
<td>$2^{20} -&gt; 1$</td>
<td>7</td>
<td>34s</td>
<td>ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Pkt Wrap</td>
<td>$2^{20} -&gt; 1$</td>
<td>6</td>
<td>132s</td>
<td>ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Serialize</td>
<td>$2^{11} -&gt; 1$</td>
<td>14</td>
<td>55s</td>
<td>ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Sum i</td>
<td>$2^{15} -&gt; 1$</td>
<td>4</td>
<td>1s</td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>Vector rotate</td>
<td>$2^{16} -&gt; 1$</td>
<td>3</td>
<td>4s</td>
<td>ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Vector shift</td>
<td>$2^{16} -&gt; 1$</td>
<td>3</td>
<td>4s</td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>Vector scale</td>
<td>$2^{16} -&gt; 1$</td>
<td>3</td>
<td>36s</td>
<td>ok</td>
<td>too complex</td>
</tr>
</tbody>
</table>

*PINS narrowed the valid candidates to 1 in almost all cases*
## Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Search space reduction</th>
<th>Number of Iterations</th>
<th>Time</th>
<th>Manual Check</th>
<th>Model Checker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run length</td>
<td>$2^{25} \rightarrow 1$</td>
<td>7</td>
<td>26s</td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>In place RL</td>
<td>$2^{30} \rightarrow 1$</td>
<td>7</td>
<td>36s</td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>LZ77</td>
<td>$2^{25} \rightarrow 2$</td>
<td>6</td>
<td></td>
<td></td>
<td>validated</td>
</tr>
<tr>
<td>LZW</td>
<td>$2^{31} \rightarrow 2$</td>
<td>4</td>
<td></td>
<td></td>
<td>too complex</td>
</tr>
<tr>
<td>Base 64</td>
<td>$2^{37} \rightarrow 4$</td>
<td>12</td>
<td></td>
<td></td>
<td>too complex</td>
</tr>
<tr>
<td>UUencode</td>
<td>$2^{20} \rightarrow 1$</td>
<td>7</td>
<td>34s</td>
<td>ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Pkt Wrap</td>
<td>$2^{20} \rightarrow 1$</td>
<td>6</td>
<td>132s</td>
<td>ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Serialize</td>
<td>$2^{11} \rightarrow 1$</td>
<td>14</td>
<td>55s</td>
<td>ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Sum i</td>
<td>$2^{15} \rightarrow 1$</td>
<td>4</td>
<td>1s</td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>Vector rotate</td>
<td>$2^{16} \rightarrow 1$</td>
<td>3</td>
<td>4s</td>
<td>ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Vector shift</td>
<td>$2^{16} \rightarrow 1$</td>
<td>3</td>
<td>4s</td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>Vector scale</td>
<td>$2^{16} \rightarrow 1$</td>
<td>3</td>
<td>36s</td>
<td>ok</td>
<td>too complex</td>
</tr>
</tbody>
</table>

Directed path exploration is successful in finding a small set of paths that prune the space.
## Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Search space reduction</th>
<th>Number of Iterations</th>
<th>Time</th>
<th>Manual Check</th>
<th>Model Checker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run length</td>
<td>$2^{25} -&gt; 1$</td>
<td>7</td>
<td>26s</td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>In place RL</td>
<td>$2^{30} -&gt; 1$</td>
<td>7</td>
<td>36s</td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>LZ77</td>
<td>$2^{25} -&gt; 2$</td>
<td>6</td>
<td>1810s</td>
<td>1 of 2 ok</td>
<td>validated</td>
</tr>
<tr>
<td>LZW</td>
<td>$2^{31} -&gt; 2$</td>
<td>4</td>
<td>150s</td>
<td>2 of 2 ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Base 64</td>
<td>$2^{37} -&gt; 4$</td>
<td>12</td>
<td>1377s</td>
<td>1 of 4 ok</td>
<td>too complex</td>
</tr>
<tr>
<td>UUencode</td>
<td>$2^{20} -&gt; 1$</td>
<td>7</td>
<td>34s</td>
<td>ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Pkt Wrap</td>
<td>$2^{20} -&gt; 1$</td>
<td>6</td>
<td>132s</td>
<td>ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Serialize</td>
<td>$2^{11} -&gt; 1$</td>
<td>14</td>
<td>55s</td>
<td></td>
<td>too complex</td>
</tr>
<tr>
<td>Sum i</td>
<td>$2^{15} -&gt; 1$</td>
<td>4</td>
<td>1s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vector rotate</td>
<td>$2^{16} -&gt; 1$</td>
<td>3</td>
<td>4s</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vector shift</td>
<td>$2^{16} -&gt; 1$</td>
<td>3</td>
<td>4s</td>
<td></td>
<td>validated</td>
</tr>
<tr>
<td>Vector scale</td>
<td>$2^{16} -&gt; 1$</td>
<td>3</td>
<td>36s</td>
<td>ok</td>
<td>too complex</td>
</tr>
</tbody>
</table>

Symbolic execution is sometimes expensive; but mostly the paths are explored in reasonable time.
## Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Search space reduction</th>
<th>Number of Iterations</th>
<th>Time</th>
<th>Manual Check</th>
<th>Model Checker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run length</td>
<td>$2^{25} \rightarrow 1$</td>
<td>7</td>
<td>26s</td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>In place RL</td>
<td>$2^{30} \rightarrow 1$</td>
<td>7</td>
<td>36s</td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>LZ77</td>
<td>$2^{25} \rightarrow 2$</td>
<td>6</td>
<td>1810s</td>
<td>1 of 2 ok</td>
<td>validated</td>
</tr>
<tr>
<td>LZW</td>
<td>$2^{31} \rightarrow 2$</td>
<td>4</td>
<td>150s</td>
<td>2 of 2 ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Base 64</td>
<td>$2^{37} \rightarrow 4$</td>
<td>12</td>
<td>1377s</td>
<td>1 of 4 ok</td>
<td>too complex</td>
</tr>
<tr>
<td>UUencode</td>
<td>$2^{20} \rightarrow 1$</td>
<td>7</td>
<td>34s</td>
<td>ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Pkt Wrap</td>
<td>$2^{20} \rightarrow 1$</td>
<td>6</td>
<td>138s</td>
<td>ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Serialize</td>
<td>$2^{11} \rightarrow 1$</td>
<td>1</td>
<td></td>
<td>ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Sum i</td>
<td>$2^{15} \rightarrow 1$</td>
<td>4</td>
<td></td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>Vector rotate</td>
<td>$2^{16} \rightarrow 1$</td>
<td>3</td>
<td></td>
<td>ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Vector shift</td>
<td>$2^{16} \rightarrow 1$</td>
<td>3</td>
<td>4s</td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>Vector scale</td>
<td>$2^{16} \rightarrow 1$</td>
<td>3</td>
<td>36s</td>
<td>ok</td>
<td>too complex</td>
</tr>
</tbody>
</table>

Either only one remained or were easily examined.
### Results

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Search space reduction</th>
<th>Number of Iterations</th>
<th>Time</th>
<th>Manual Check</th>
<th>Model Checker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run length</td>
<td>(2^{25} \rightarrow 1)</td>
<td>7</td>
<td>26s</td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>In place RL</td>
<td>(2^{30} \rightarrow 1)</td>
<td>7</td>
<td>36s</td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>LZ77</td>
<td>(2^{25} \rightarrow 2)</td>
<td>6</td>
<td>1810s</td>
<td>1 of 2 ok</td>
<td>validated</td>
</tr>
<tr>
<td>LZW</td>
<td>(2^{31} \rightarrow 2)</td>
<td>4</td>
<td>150s</td>
<td>2 of 2 ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Base 64</td>
<td>(2^{37} \rightarrow 4)</td>
<td>12</td>
<td>1377s</td>
<td>1 of 4 ok</td>
<td>too complex</td>
</tr>
<tr>
<td>UUencode</td>
<td>(2^{20} \rightarrow 1)</td>
<td>7</td>
<td>34s</td>
<td>ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Pkt Wrap</td>
<td>(2^{20} \rightarrow 1)</td>
<td>6</td>
<td>132s</td>
<td>ok</td>
<td>too complex</td>
</tr>
<tr>
<td>Serialize</td>
<td>(2^{11} \rightarrow 1)</td>
<td>14</td>
<td></td>
<td></td>
<td>too complex</td>
</tr>
<tr>
<td>Sum i</td>
<td>(2^{15} \rightarrow 1)</td>
<td>5</td>
<td></td>
<td></td>
<td>validated</td>
</tr>
<tr>
<td>Vector rotate</td>
<td>(2^{16} \rightarrow 1)</td>
<td>10</td>
<td></td>
<td></td>
<td>too complex</td>
</tr>
<tr>
<td>Vector shift</td>
<td>(2^{16} \rightarrow 1)</td>
<td>3</td>
<td></td>
<td>ok</td>
<td>validated</td>
</tr>
<tr>
<td>Vector scale</td>
<td>(2^{16} \rightarrow 1)</td>
<td>3</td>
<td>36s</td>
<td>ok</td>
<td>too complex</td>
</tr>
</tbody>
</table>

**A testing-based approach is the only viable option, as most of the examples are too complex for even for bounded verification.**
Conclusions

- PINS seems very promising
  - First testing-based approach to program synthesis
  - To our knowledge, no other technique can invert these programs with as little guidance

- Supports small path-bound hypothesis for synthesis
  - Makes sense, since it works for testing (approximate verification), and we know verification and synthesis are related (see POPL’10 paper)

- PINS should be applicable to other domains too

http://www.cs.umd.edu/~saurabhs/vs3/PINS/
PINS approach vs CEGAR/CEGIS

Verifier

Constraint Solver

Counterexample

SATisfying Soln

Evidence of correctness

Constraint Solver

Correctness checker

picked

k SATisfying Solns

Correctness checker

Evidence of correctness

Verifier

Counterexample

Constraint Solver

SATisfying Soln
void main(int n, BitString A) {
    BitString *D;
    int *B;
    int i,p,k,j,r,size,x,go;
    IN(str(A,0,n-1),n);
    ASSUME(n >= 1);
    D[0] = "0";
    D[1] = "1";
    i = 0; p = 2; k = 0;
    while (i < n) {
        j = i; r = 0; size=-1;
        while (j < n && r != -1) {
            x = 0; r = -1;
            while (x < p) {
                if (D[x] == substr(A,i,j))
                    r = x;
                x++;
            }
            if (r != -1) {
                go = r; size = j-i+1;
            }
            j++;
        }
        B[k++] = go;
        D[p++] = substr(A,i,j-1);
        i += size;
    }
    OUT(B,k);
}

LZW compression

void main(int *A, int n) {
    int *P,*N,*C;
    int i,j,k,c,p,r;
    IN(BOUND(A,0,n),n);
    ASSUME(n >= 0);
    i = 0; k = 0;
    while (i < n) {
        c = 0; p = 0; j = 0;
        while (j < i) {
            r = 0;
            while (i+r < n-1 && A[j+r] == A[i+r])
                r++;
            if (c < r) {
                c = r; p = i-j;
            }
            j++;
        }
        P[k] = p; N[k] = c; C[k] = A[i+c];
        i = i+1+c;
        k++;
    }
    OUT(P,N,C,k);
}

LZW decompression