



Art of Invariant Generation applied to Symbolic Bound Computation

Part 2

Sumit Gulwani

(Microsoft Research, Redmond, USA)

Oregon Summer School

July 2009



Art of Invariant Generation

1. Program Transformations



- Reduce need for sophisticated invariant generation.
- E.g., control-flow refinement, loop-flattening/peeling, non-standard cut-points, quantitative attributes instrumentation.

2. Colorful Logic



- Language of Invariants
- E.g., arithmetic, uninterpreted fns, lists/arrays

3. Fixpoint Brush



- Automatic generation of invariants in some shade of logic, e.g., conjunctive/k-disjunctive/predicate abstraction.
- E.g., Iterative, Constraint-based, Proof Rules



Logic

- To validate correctness of loop-free programs, or programs annotated with loop invariants, decision procedures are enough.
 - Provided the logic is closed under WP.
 - $\langle \text{Pre}, S, \text{Post} \rangle$ is valid
 - iff $\text{Pre} \Rightarrow \text{WP}(S, \text{Post})$ is valid
 - iff $\text{Pre} \wedge \neg \text{WP}(S, \text{Post})$ is unsatisfiable
- To validate correctness of programs with loops, we need to automatically discover loop invariants using fixpoint computation techniques.
 - This requires algorithms that are more sophisticated than decision procedures.



Fixpoint Brush

We will briefly study fixpoint techniques for discovering loop invariants in conjunctive fragments of various logics.

- Iterative
 - Forward
 - Backward
- Constraint-based
- Proof-rules

Iterative Forward: Examples

- Start with Precondition and propagate facts forward using **Existential elimination** operator until fixpoint.
- Data-flow analysis
 - Join at merge points
 - Finite abstract domains
 - Abstract Interpretation
 - **Join** at merge points
 - Widening for infinite height abstract domains.
 - Model Checking
 - Analyze all paths precisely without join at merge points.
 - Finite abstractions (e.g., boolean abstraction) required
 - Counterexample guided abstraction refinement
 - BDD based data-structures for efficiency

Key Operators needed by Iterative Forward

- Fixpoint checking: **Decision Procedure.**

$\text{Decide}_T(\phi) = \text{Yes}$, iff ϕ is satisfiable

- Transfer Function for Assignment Node: **Existential Quantifier Elimination**

$\text{Eliminate}(\phi, V) = \text{strongest } \phi' \text{ such that } V \in \text{Vars}(\phi')$

- Transfer Function for Merge Node: **Join**

$\text{Join}(\phi_1, \phi_2) = \text{strongest } \phi \text{ s.t. } \phi_1 \Rightarrow \phi \text{ and } \phi_2 \Rightarrow \phi$

Eliminate and Join are usually harder than Decide.

Difference Constraints

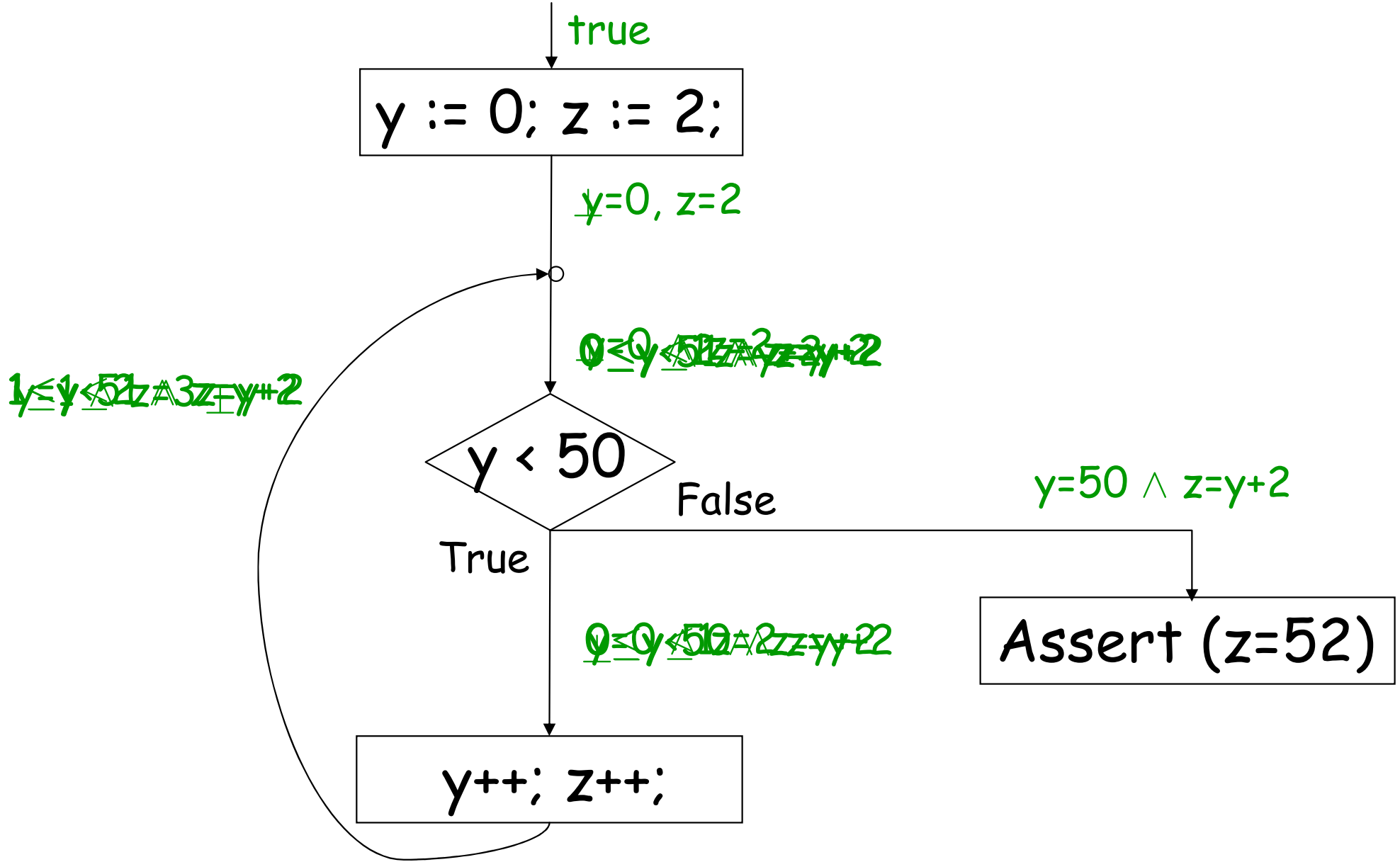
- Abstract element:
 - conjunction of $x_i - x_j \leq c_{ij}$
 - can be represented using matrix M , where $M[i][j] = c_{ij}$
- Decide(M):
 1. $M' := \text{Saturate}(M)$;
 2. Declare unsat iff $\exists i: M'[i][i] < 0$
- Join(M_1, M_2):
 1. $M'_1 := \text{Saturate}(M_1)$; $M'_2 := \text{Saturate}(M_2)$;
 2. Let M_3 be s.t. $M_3[i][j] = \text{Max} \{ M'_1[i][j], M'_2[i][j] \}$
 3. return M_3

Difference Constraints

- $\text{Eliminate}(M, x_i)$:
 1. $M' := \text{Saturate}(M)$;
 2. Let M_1 be s.t. $M_1[j][k] = \infty$ (if $j=i$ or $k=i$)
 $= M'[j][k]$ otherwise
 3. return M_1

- $\text{Widen}(M_1, M_2)$:
 1. $M'_1 := \text{Saturate}(M_1)$; $M'_2 := \text{Saturate}(M_2)$;
 2. Let M_3 be s.t. $M_3[i][j] = M_1[i][j]$ (if $M_1[i][j] = M_2[i][j]$)
 $= \infty$ (otherwise)
 3. return M_3

Example: Abstract Interpretation using Difference Constraints



Uninterpreted Functions

- Abstract element:
 - conjunction of $e_1=e_2$, where $e := y \mid F(e_1,e_2)$
 - can be represented using EDAGs
- Decide(G):
 1. $G' := \text{Saturate}(G)$;
 2. Declare unsat iff G contains $e_1 \neq e_2$ and G' has e_1, e_2 in the same congruence class.
- Eliminate(G, y):
 1. $G' := \text{Saturate}(G)$;
 2. Erase y ; (might need to delete some dangling expressions)
 3. return G'

Uninterpreted Functions

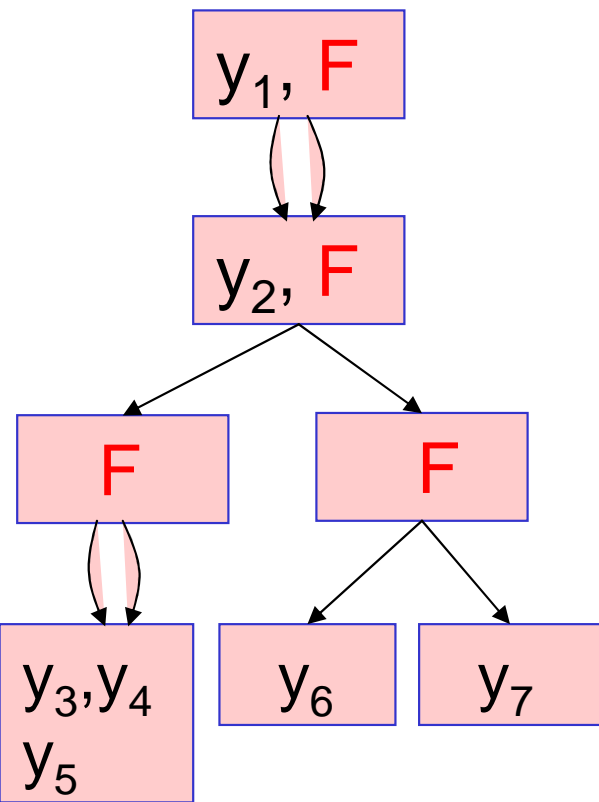
- $\text{Join}(G_1, G_2)$:
 1. $G'_1 := \text{Saturate}(G_1)$; $G'_2 := \text{Saturate}(G_2)$;
 2. $G := \text{Intersect}(G'_1, G'_2)$;
 3. return G ;

For each node $n = \langle U, F(n_i, n'_i) \rangle$ in G'_1

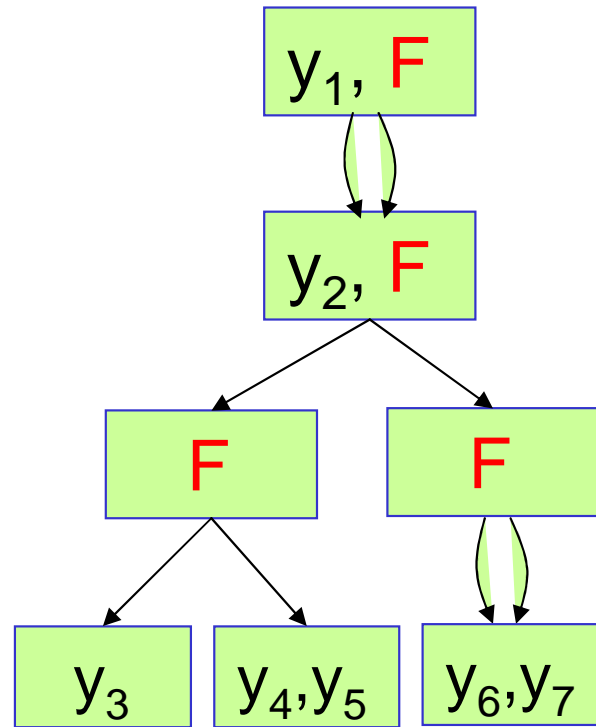
and node $m = \langle V, F(m_j, m'_j) \rangle$ in G'_2 ,

G contains a node $[n, m] = \langle U \cap V, F([n_i, m_j], [n'_i, m'_j]) \rangle$

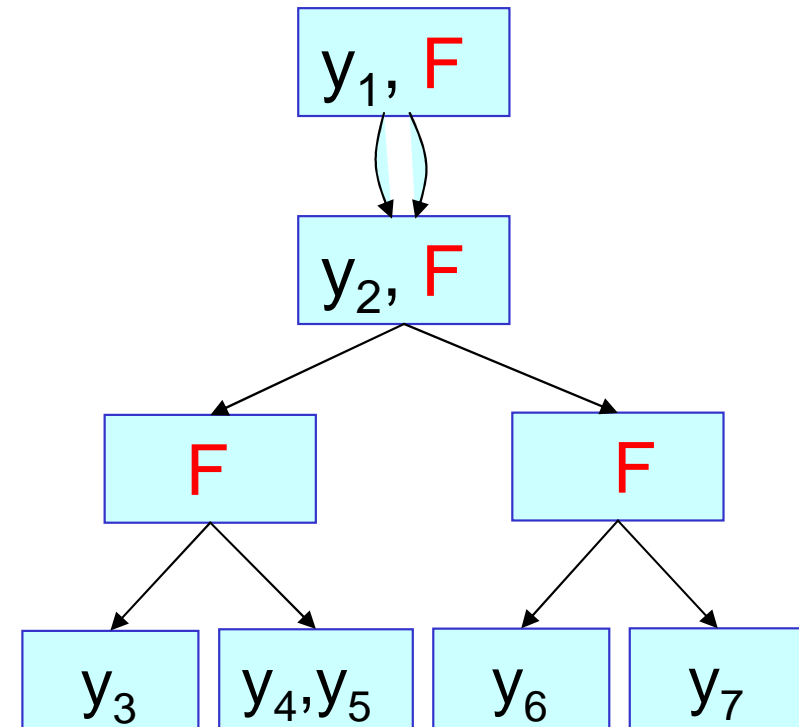
Uninterpreted Functions: Example of Join



G_1



G_2



$G = \text{Join}(G_1, G_2)$

Recap: Combination of Decision Procedures

- $\text{Decide}(E_{12})$:
 1. $\langle E_1, E_2 \rangle := \text{Purify\&Saturate}(E_{12})$;
 2. Return $\text{Decide}_{T_1}(E_1) \wedge \text{Decide}_{T_2}(E_2)$;

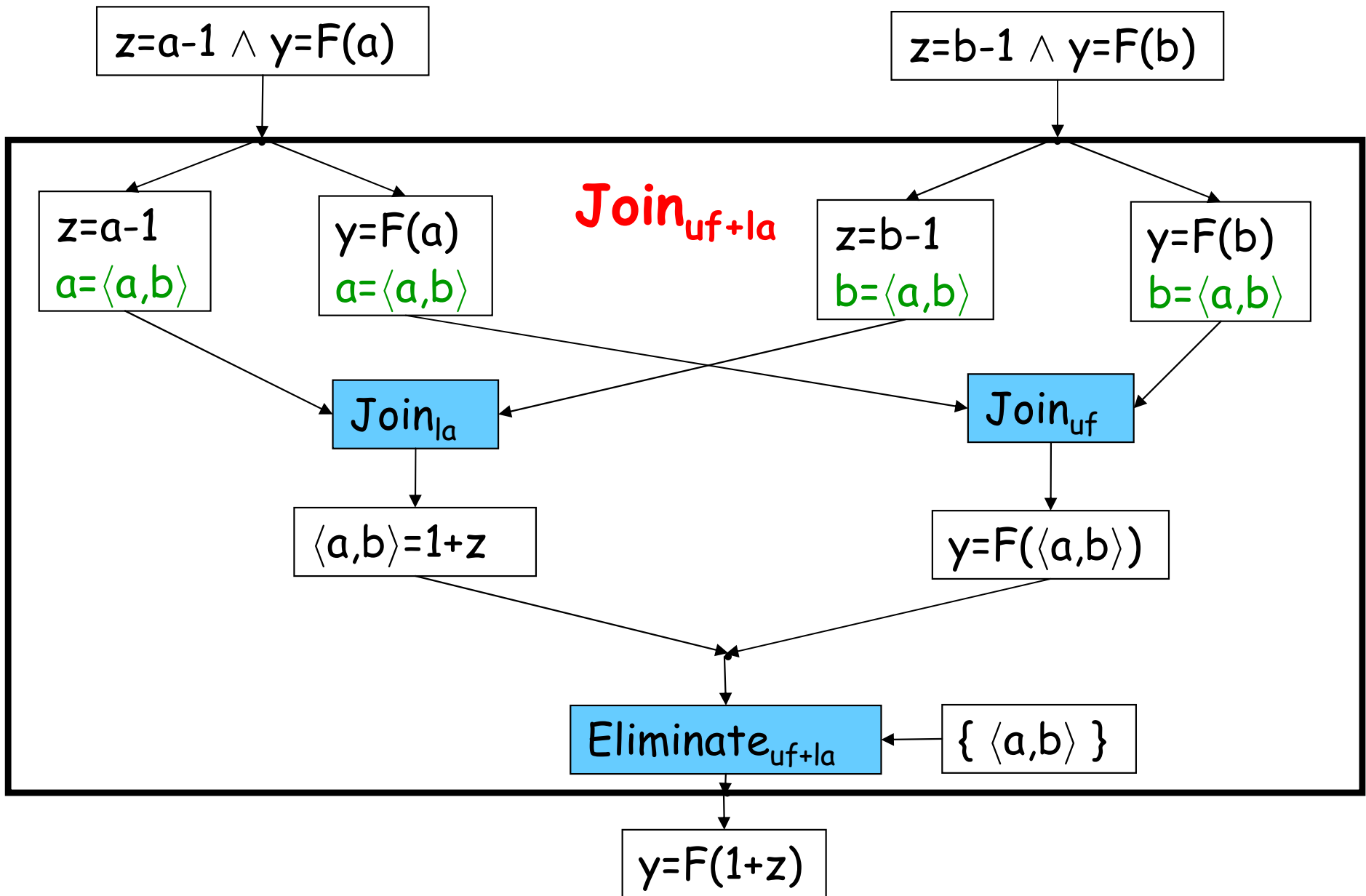
Combination: Join Algorithm (1st attempt)

- $\text{Join}_{T_{12}}(L_{12}, R_{12})$:
 1. $\langle L_1, L_2 \rangle := \text{Purify\&Saturate}(L_{12})$;
 $\langle R_1, R_2 \rangle := \text{Purify\&Saturate}(R_{12})$;
 2. $A_1 := \text{Join}_{T_1}(L_1, R_1)$;
 $A_2 := \text{Join}_{T_2}(L_2, R_2)$;
 3. Return $A_1 \wedge A_2$;

Combination: Join Algorithm

- $\text{Join}_{T_{12}}(L_{12}, R_{12})$:
 1. $\langle L_1, L_2 \rangle := \text{Purify\&Saturate}(L_{12})$;
 $\langle R_1, R_2 \rangle := \text{Purify\&Saturate}(R_{12})$;
 2. $D_L := \bigwedge \{v_i = \langle v_i, v_j \rangle \mid v_i \in \text{Vars}(L_1 \wedge L_2), v_j \in \text{Vars}(R_1 \wedge R_2)\}$;
 $D_R := \bigwedge \{v_j = \langle v_i, v_j \rangle \mid v_i \in \text{Vars}(L_1 \wedge L_2), v_j \in \text{Vars}(R_1 \wedge R_2)\}$;
 3. $L'_1 := L_1 \wedge D_L$; $R'_1 := R_1 \wedge D_R$;
 $L'_2 := L_2 \wedge D_L$; $R'_2 := R_2 \wedge D_R$;
 4. $A_1 := \text{Join}_{T_1}(L'_1, R'_1)$;
 $A_2 := \text{Join}_{T_2}(L'_2, R'_2)$;
 5. $V := \text{Vars}(A_1 \wedge A_2)$ - Program Variables;
 $A_{12} := \text{Eliminate}_{T_{12}}(A_1 \wedge A_2, V)$;
 6. Return A_{12} ;

Combination: Example of Join Algorithm

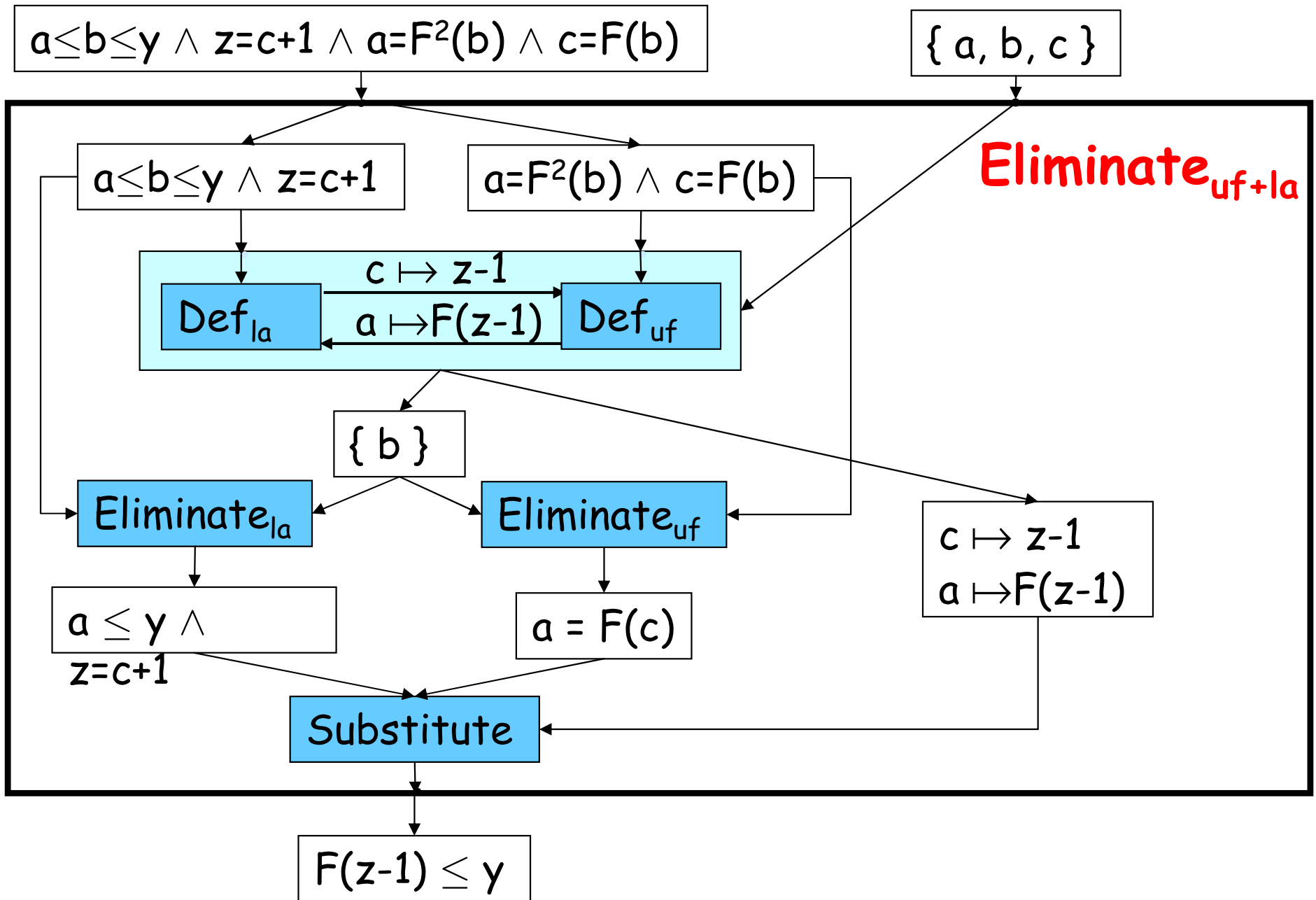


Combination: Existential Quantifier Elimination

- Elimintate_{T₁₂}(E₁₂, V):
 1. $\langle E_1, E_2 \rangle := \text{Purify\&Saturate}(E_{12});$
 2. $\langle D, \text{Defs} \rangle := \text{DefSaturate}(E_1, E_2, V \cup \text{Temp Variables});$
 3. $V' := V \cup \text{Temp Variables} - D;$
 $E'_1 := \text{Eliminate}_{T_1}(E_1, V');$
 $E'_2 := \text{Eliminate}_{T_2}(E_2, V');$
 4. $E := (E'_1 \wedge E'_2) [\text{Defs}(y)/y];$
 5. Return E;

DefSaturate(E₁, E₂, U) returns the set of all variables D that have definitions Defs in terms of variables not in U as implied by E₁ ∧ E₂.

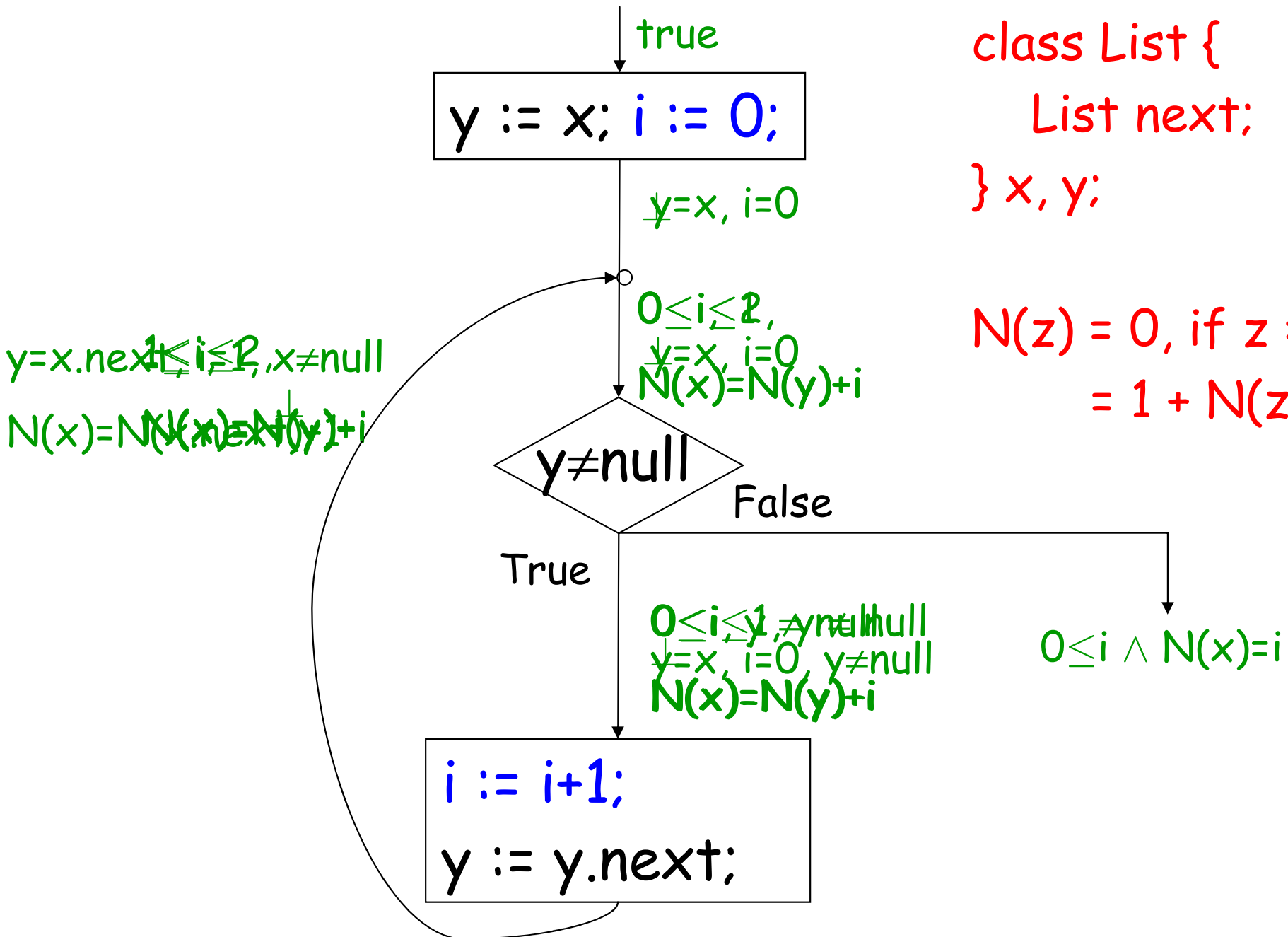
Combination: Example of Existential Elimination



Example: Abstract Interpretation over Combined Domain

```
class List {
    List next;
} x, y;
```

$N(z) = 0$, if $z = \text{null}$
 $= 1 + N(z.\text{next})$



Iterative Forward: References

- Uninterpreted Functions
 - A Polynomial-Time Algorithm for Global Value Numbering;
Gulwani, Necula; SAS '04
- Linear Arithmetic + Uninterpreted Functions
 - Combining Abstract Interpreters;
Gulwani, Tiwari; PLDI '06
- Theory of Arrays/Lists
 - Quantified Abstract Domains
 - Lifting Abstract Interpreters to Quantified Logical Domains;
Gulwani, McCloskey, Tiwari; POPL '08
 - Discovering Properties about Arrays in Simple Programs;
Halbwachs, Péron; PLDI '08
 - Shape Analysis
 - Parametric Shape Analysis via 3-Valued Logic;
Sagiv, Reps, Wilhelm; POPL '99, TOPLAS '02

Iterative Forward: References

- Theory of Arrays/Lists
 - Combination of Shape Analysis + Arithmetic
 - A Combination Framework for tracking partition sizes; Gulwani, Lev-Ami, Sagiv; POPL '09
- Non-linear Arithmetic
 - User-defined axioms + Expression Abstraction
 - A Numerical Abstract Domain Based on Expression Abstraction and Max Operator with Application in Timing Analysis; Gulavani, Gulwani; CAV '08
 - Polynomial Equalities
 - An Abstract Interpretation Approach for Automatic Generation of Polynomial Invariants; Rodriguez-Carbonell, Kapur; SAS '04



Fixpoint Brush

- Iterative
 - Forward
 - Backward
- Constraint-based
- Proof-rules

Iterative Backward

- Comparison with Iterative Forward
 - Positives: Can compute preconditions, Goal-directed
 - Negatives: Requires assertions or template assertions.
- Transfer Function for Assignment Node is easier.
 - Substitution takes role of existential elimination.
- Transfer Function for Conditional Node is challenging.
 - Requires abductive reasoning/under-approximations.
 - $\text{Abduct}(\phi, g) = \text{weakest } \phi' \text{ s.t. } \phi' \wedge g \Rightarrow \phi$
 - Case-split reasoning as opposed to Saturation based, and hence typically not closed under conjunctions.
 - Optimally weak solutions for negative unknowns as opposed to optimally strong solutions for positive unknowns.

Iterative Backward: References

- Program Verification using Templates over Predicate Abstraction;
Srivastava, Gulwani; PLDI '09
- Assertion Checking Unified;
Gulwani, Tiwari; VMCAI '07
- Computing Procedure Summaries for Interprocedural Analysis;
Gulwani, Tiwari; ESOP '07

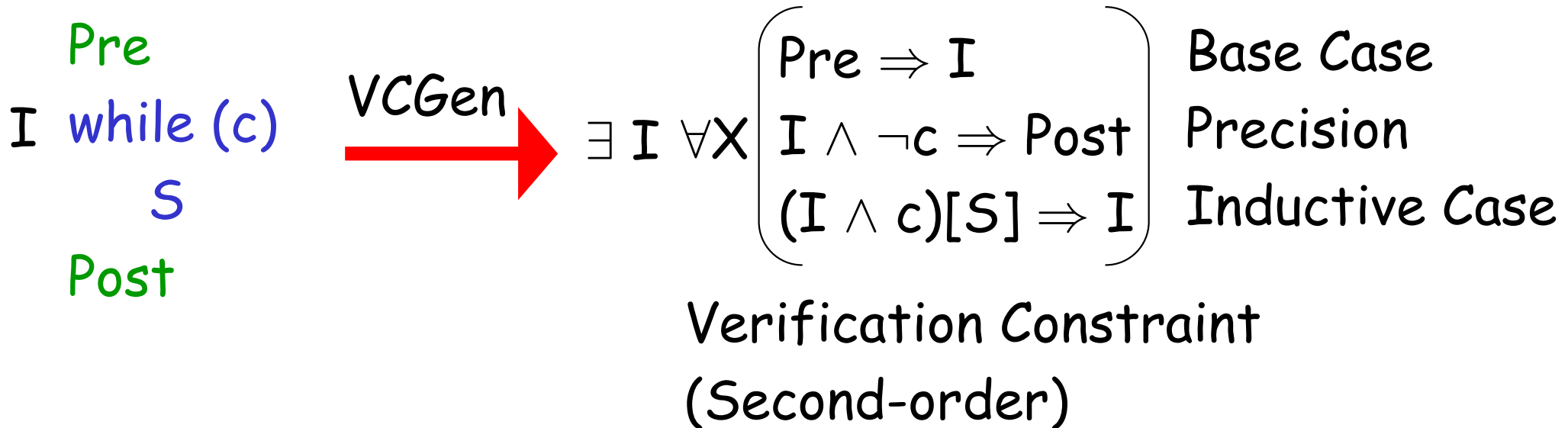


Fixpoint Brush

- Iterative
 - Forward
 - Backward
- Constraint-based
- Proof-rules

Constraint-based Invariant Generation

- Goal-directed invariant generation for verification of a Hoare triple (Pre, Program, Post)



- Key Idea: Reduce the second-order verification constraint to a first-order satisfiability constraint that can be solved using off-the-shelf SAT/SMT solvers
 - Choose a template for I (specific color/shade in some logic).
 - Convert \forall into \exists .

Key Idea in reducing \forall to \exists for various Domains

Trick for converting \forall to \exists is known for following domains:

- Linear Arithmetic
 - Farkas Lemma
- Linear Arithmetic + Uninterpreted Fns.
 - Farkas Lemma + Ackerman's Reduction
- Non-linear Arithmetic
 - Grobner Basis
- Predicate Abstraction
 - Boolean indicator variables + Cover Algorithm (Abduction)
- Quantified Predicate Abstraction
 - Boolean indicator variables + More general Abduction

Constraint-based Invariant Generation: References

- Linear Arithmetic
 - Constraint-based Linear-relations analysis; Sankaranarayanan, Sipma, Manna; SAS '04
 - Program analysis as constraint solving; Gulwani, Srivastava, Venkatesan; PLDI '08
- Linear Arithmetic + Uninterpreted Fns.
 - Invariant synthesis for combined theories; Beyer, Henzinger, Majumdar, Rybalchenko; VMCAI '07
- Non-linear Arithmetic
 - Non-linear loop invariant generation using Gröbner bases; Sankaranarayanan, Sipma, Manna; POPL '04
- Predicate Abstraction
 - Constraint-based invariant inference over predicate abstraction; Gulwani, Srivastava, Venkatesan; VMCAI '09
- Quantified Predicate Abstraction
 - Program verification using templates over predicate abstraction; Srivastava, Gulwani; PLDI '09

Constraint-based Invariant Generation

➤ Linear Arithmetic

- Linear Arithmetic + Uninterpreted Fns.
- Non-linear Arithmetic
- Predicate Abstraction
- Quantified Predicate Abstraction

Farkas Lemma

$$\forall X \wedge_k (e_k \geq 0) \Rightarrow e \geq 0$$

iff

$$\exists \lambda_k \geq 0 \forall X (e \equiv \lambda + \sum_k \lambda_k e_k)$$

Example

- Let's find Farkas witness $\lambda, \lambda_1, \lambda_2$ for the implication
$$x \geq 2 \wedge y \geq 3 \Rightarrow 2x + y \geq 6$$
- $\exists \lambda, \lambda_1, \lambda_2 \geq 0$ s.t. $\forall x, y [2x + y - 6 \equiv \lambda + \lambda_1(x - 2) + \lambda_2(y - 3)]$
- Equating coefficients of x, y and constant terms, we get:
$$2 = \lambda_1 \wedge 1 = \lambda_2 \wedge -6 = \lambda - 2\lambda_1 - 3\lambda_2$$
which implies $\lambda = 1, \lambda_1 = 2, \lambda_2 = 1$.

Solving 2nd order constraints using Farkas Lemma

$$\exists \mathbf{I} \forall \mathbf{X} \phi_1(\mathbf{I}, \mathbf{X})$$

- Second-order to First-order
 - Assume \mathbf{I} has some form, e.g., $\sum_j a_j x_j \geq 0$
 - $\exists \mathbf{I} \forall \mathbf{X} \phi_1(\mathbf{I}, \mathbf{X})$ translates to $\exists \mathbf{a}_j \forall \mathbf{X} \phi_2(\mathbf{a}_j, \mathbf{X})$
- First-order to “only existentially quantified”
 - Farkas Lemma helps translate \forall to \exists
 - $\forall \mathbf{X} (\wedge_k (e_k \geq 0) \Rightarrow e \geq 0)$ iff $\exists \lambda_k \geq 0 \forall \mathbf{X} (e \equiv \lambda + \sum_k \lambda_k e_k)$
 - Eliminate \mathbf{X} from polynomial equality by equating coefficients.
 - $\exists \mathbf{a}_j \forall \mathbf{X} \phi_2(\mathbf{a}_j, \mathbf{X})$ translates to $\exists \mathbf{a}_j \exists \lambda_k \phi_3(\mathbf{a}_j, \lambda_k)$
- “only existentially quantified” to SAT
 - Bit-vector modeling for integer variables

Example

I

```

[n=1 ∧ m=1]
x := 0; y := 0;
while (x < 100)
  x := x+n;
  y := y+m;
[y ≥ 100]
    
```

VCGen

```

n=m=1 ∧ x=y=0 ⇒ I
I ∧ x ≥ 100 ⇒ y ≥ 100
I ∧ x < 100 ⇒ I[x ← x+n,
y ← y+m]
    
```

Invariant Template

```

a0 + a1x + a2y + a3n + a4m ≥ 0
b0 + b1x + b2y + b3n + b4m ≥ 0
c0 + c1x + c2y + c3n + c4m ≥ 0
    
```

Satisfying Solution

$a_2=b_0=c_4=1, a_1=b_3=c_0=-1$

Loop Invariant

```

y ≥ x
m ≥ 1
n ≤ 1
    
```

```

a0 + a1x + a2y + a3n + a4m ≥ 0
b0 + b1x + b2y + b3n + b4m ≥ 0
    
```

$a_2=b_2=1, a_1=b_1=-1$

```

y ≥ x
m ≥ n
    
```

```

a0 + a1x + a2y + a3n + a4m ≥ 0
    
```

UNSAT

Invalid triple or Imprecise Template

Constraint-based Invariant Generation

- Linear Arithmetic
- Linear Arithmetic + Uninterpreted Fns.
- Non-linear Arithmetic
- Predicate Abstraction
- Quantified Predicate Abstraction

Solving 2nd order constraints using Boolean Indicator Variables + Cover Algorithm

$$\exists I \forall X \phi_1(I, X)$$

- Second-order to First-order (Boolean indicator variables)
 - Assume I has the form $P_1 \vee P_2$, where $P_1, P_2 \subseteq P$
 - Let $b_{i,j}$ denote presence of $p_j \in P$ in P_i
 - Then, I can be written as $(\bigwedge_j b_{1,j} \Rightarrow p_j) \vee (\bigwedge_j b_{2,j} \Rightarrow p_j)$
 - $\exists I \forall X \phi_1(I, X)$ translates to $\exists b_{i,j} \forall X \phi_2(b_{i,j}, X)$
 - Can generalize to k disjuncts
- First-order to “only existentially quantified”
 - Cover Algorithm helps translate \forall to \exists
 - $\exists b_{i,j} \forall X \phi_2(b_{i,j}, X)$ translates to SAT formula $\exists b_{i,j} \phi_3(b_{i,j})$

Example

I

```
[m > 0]
x := 0; y := 0;
while (x < m)
  x := x+1;
  y := y+1;
[y=m]
```

VCGen \rightarrow

```
m > 0  $\Rightarrow$  I[x $\leftarrow$ 0, y $\leftarrow$ 0]
I  $\wedge$  x  $\geq$  m  $\Rightarrow$  y=m
I  $\wedge$  x < m  $\Rightarrow$  I[x $\leftarrow$ x+1, y $\leftarrow$ y+1]
```

Suppose $P =$

```
x  $\leq$  y, x  $\geq$  y, x < y
x  $\leq$  m, x  $\geq$  m, x < m
y  $\leq$  m, y  $\geq$  m, y < m
```

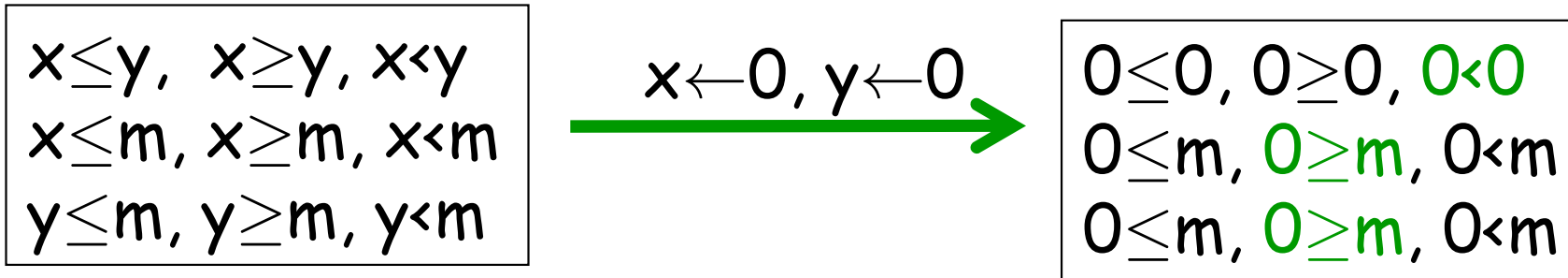
and $k = 1$

Then, I has the form P_1 , where $P_1 \subseteq P$

```
m > 0  $\Rightarrow$  P1[x $\leftarrow$ 0, y $\leftarrow$ 0] (1)
P1  $\wedge$  x  $\geq$  m  $\Rightarrow$  y=m (2)
P1  $\wedge$  x < m  $\Rightarrow$  P1[x $\leftarrow$ x+1, y $\leftarrow$ y+1] (3)
```

Example

(1) $m > 0 \Rightarrow P_1 [x \leftarrow 0, y \leftarrow 0]$



Hence, (1) $\equiv P_1$ doesn't contain $x < y, x \geq m, y \geq m$
 $\equiv \neg b_{x \geq m} \wedge \neg b_{x < y} \wedge \neg b_{y \geq m}$

(2) $P_1 \wedge x \geq m \Rightarrow y = m$

There are 3 maximally-weak choices for P_1

- (i) $y \leq m \wedge y \geq m$
 - (ii) $x < m$

(computed using Pred)

Hence, (2) $\equiv P_1$ contains at least one of above combinations
 $\equiv (b_{x < m}) \vee (b_{y \leq m} \wedge b_{y \geq m}) \vee (b_{x \leq y} \wedge b_{y \leq m})$

Example

- (1) $\neg b_{x \geq m} \wedge \neg b_{x < y} \wedge \neg b_{y \geq m}$
(2) $(b_{x < m}) \vee (b_{y \leq m} \wedge b_{y \geq m}) \vee (b_{x \leq y} \wedge b_{y \leq m})$
(3) $(b_{y \leq m} \Rightarrow (b_{y < m} \vee b_{y \leq x})) \wedge \neg b_{x < m} \wedge \neg b_{y < m}$

Obtained from solving local/small SMT queries

SAT Solver

$b_{y \leq x}, b_{y \leq m}, b_{x \leq y}$: true

rest: false

I: $(y = x \wedge y \leq m)$

```
[m > 0]
x := 0; y := 0;
I while (x < m)
    x := x+1;
    y := y+1;
[y = m]
```

Bonus Material



Where can we go?

Going beyond Invariant Generation with Constraint-based techniques...

Example: Bresenham's Line Drawing Algorithm

```
[ $0 < Y \leq X$ ]  
 $v_1 := 2Y - X$ ;  $y := 0$ ;  $x := 0$ ;  
while ( $x \leq X$ )  
  out[x] := y;  
  if ( $v_1 < 0$ )  $v_1 := v_1 + 2Y$ ;  
  else  $v_1 := v_1 + 2(Y - X)$ ;  $y++$ ;  
return out;  
[ $\forall k (0 \leq k \leq X \Rightarrow |out[k] - (Y/X)k| \leq \frac{1}{2})$ ]
```

Postcondition: The best fit line shouldn't deviate more than half a pixel from the real line, i.e., $|y - (Y/X)x| \leq 1/2$

Transition System Representation

$[0 < Y \leq X]$

$v_1 := 2Y - X; y := 0; x := 0;$

while ($x \leq X$)

$v_1 < 0: \text{out}' = \text{Update}(\text{out}, x, y) \wedge v_1' = v_1 + 2Y \wedge y' = y \wedge x' = x + 1$

$v_1 \geq 0: \text{out}' = \text{Update}(\text{out}, x, y) \wedge v_1' = v_1 + 2(Y - X) \wedge y' = y + 1 \wedge x' = x + 1$

$[\forall k (0 \leq k \leq X \Rightarrow |\text{out}[k] - (Y/X)k| \leq \frac{1}{2})]$

Or, equivalently,

Where,

$g_{\text{body1}}: v_1 < 0$

$g_{\text{body2}}: v_1 \geq 0$

$g_{\text{loop}}: x \leq X$

$s_{\text{entry}}: v_1' = 2Y - X \wedge y' = 0 \wedge x' = 0$

$s_{\text{body1}}: \text{out}' = \text{Update}(\text{out}, x, y) \wedge v_1' = v_1 + 2Y \wedge x' = x + 1 \wedge y' = y$

$s_{\text{body2}}: \text{out}' = \text{Update}(\text{out}, x, y) \wedge v_1' = v_1 + 2(Y - X) \wedge x' = x + 1 \wedge y' = y + 1$

$[\text{Pre}]$

$s_{\text{entry}};$

while (g_{loop})

$g_{\text{body1}}: s_{\text{body1}};$

$g_{\text{body2}}: s_{\text{body2}};$

$[\text{Post}]$

Verification Constraint Generation & Solution

Verification Constraint:

$$\begin{aligned} \text{Pre} \wedge s_{\text{entry}} &\Rightarrow I' \\ I \wedge g_{\text{loop}} \wedge g_{\text{body1}} \wedge s_{\text{body1}} &\Rightarrow I' \\ I \wedge g_{\text{loop}} \wedge g_{\text{body2}} \wedge s_{\text{body2}} &\Rightarrow I' \\ I \wedge \neg g_{\text{loop}} &\Rightarrow \text{Post} \end{aligned}$$

Given Pre, Post, g_{loop} , g_{body1} , g_{body2} , s_{body1} , s_{body2} , we can find solution for I using constraint-based techniques.

$$I: \quad 0 < Y \leq X \wedge v_1 = 2(x+1)Y - (2y+1)X \wedge 2(Y-X) \leq v_1 \leq 2Y \wedge \forall k (0 \leq k \leq x \Rightarrow |\text{out}[k] - (Y/X)k| \leq \frac{1}{2})$$

The Surprise!

Verification Constraint:

$$\begin{aligned} \text{Pre} \wedge s_{\text{entry}} &\Rightarrow I' \\ I \wedge g_{\text{loop}} \wedge g_{\text{body1}} \wedge s_{\text{body1}} &\Rightarrow I' \\ I \wedge g_{\text{loop}} \wedge g_{\text{body2}} \wedge s_{\text{body2}} &\Rightarrow I' \\ I \wedge \neg g_{\text{loop}} &\Rightarrow \text{Post} \end{aligned}$$

- What if we treat each g and s as unknowns like I ?
- We get a solution that has $g_{\text{body1}} = g_{\text{body2}} = \text{false}$.
 - This doesn't correspond to a valid transition system.
 - We can fix this by encoding $g_{\text{body1}} \vee g_{\text{body2}} = \text{true}$.
- We now get a solution that has $g_{\text{loop}} = \text{true}$.
 - This corresponds to a non-terminating loop.
 - We can fix this by encoding *existence of a ranking function*.
- We now discover each g and s along with I .
 - We have gone from Invariant Synthesis to *Program Synthesis*.



Fixpoint Brush

- Iterative
 - Forward
 - Backward
- Constraint-based
 - Proof-rules

Proof Rules

while (cond(X))

$\pi: X' := F(X);$

Bounding Loop Iterations

If $(\text{cond}(X) \wedge X'=F(X)) \Rightarrow (e>0 \wedge e[X'/X] \leq e-1),$

Then $\text{Bound}(\pi) \leq e$

- Candidate expressions for e : Look inside $\text{Cond}(X)$.

Bounding values of monotonically increasing variables

If $(\text{cond}(X) \wedge X'=F(X)) \Rightarrow y' \leq y+c,$

Then $y^{\text{out}} \leq y^{\text{in}} + c \times \text{Bound}(\pi)$

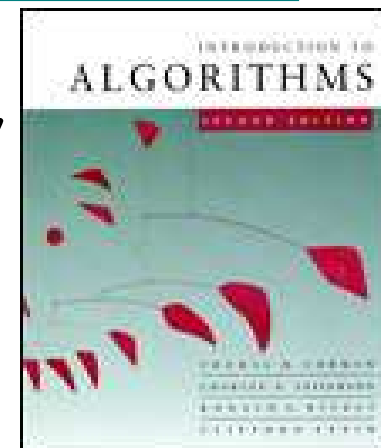
- Candidate constants for c : 1, 2

Recurrence Solving Techniques vs. Our Fixpoint Brush

Undergraduate Textbook on Algorithms by Cormen, Leiserson, Rivest, Stein describes 3 fundamental methods for recurrence solving:

Example of a recurrence: $T(n) = T(n-1) + 2n \wedge T(0) = 0$

- Iteration Method
 - Expands/unfolds the recurrence relation
 - Similar to Iterative approach
- Substitution Method
 - Assumes a template for a closed-form
 - Similar to Constraint-based approach
- Masters Theorem
 - Provides a cook-book solution for $T(n) = aT(n/b) + f(n)$
 - Similar to Proof-Rules approach





Fixpoint Brush

- Iterative
 - Forward
 - Backward
- Constraint-based
- Proof-rules
- Learning
 - Iterative, but w/o monotonic increase or decrease.
 - Distance to fixed-point decreases in each iteration.

Learning: References

- Program Verification as Probabilistic Inference;
Gulwani, Jovic; POPL '07
- Learning Regular Sets from Queries and Counterexamples;
Angluin; Information and Computing '87
 - Learning Synthesis of interface specifications for Java classes;
Alur, Madhusudan, Nam; POPL '05.
 - Learning meets verification;
Leucker; FMCO '06
- May/Must Analyses?
- Game-theoretic Analyses?



Fixpoint Brush: Summary

- Iterative
 - Forward
 - Join, Existential Elimination
 - Backward
 - Abduct
- Constraint-based
 - Exotic; Works well for small code-fragments
 - \forall to \exists
- Proof-rules
 - Scalable
 - Requires understanding design patterns
- Learning