Logical Abstract Interpretation

Sumit Gulwani
Microsoft Research, Redmond

Final Goal of the class

Automatically verify partial correctness of programs like the following using abstract interpretation.

```
Void Init(int* A, int n) {
    for (i := 0; i<n; i++;)
        A[i] := 0;
    for (j := 0; j<n; j++;)
        Assert(A[j] = 0);
}
```

Outline

- Decision Procedures
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
- Logical Abstract Interpretation
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
  - Universally Quantified Formulas
- Hardness of Assertion Checking
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns

Decision Procedures

$DP_T(\phi)$ = Yes, if ϕ is satisfiable

= No, if ϕ is unsatisfiable

Without loss of generality, we can assume that ϕ is a conjunction of atomic facts.

- Why?
  - $DP(\phi_1 \vee \phi_2)$ is sat iff $DP(\phi_1)$ is sat or $DP(\phi_2)$ is sat
- What is the trade-off?
  - Converting ϕ into DNF may incur exponential blow-up

Outline

- Decision Procedures
  - ➢ Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
- Logical Abstract Interpretation
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
  - Universally Quantified Formulas
- Hardness of Assertion Checking
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns

Linear Arithmetic

Expressions $e := y \mid c \mid e_1 \pm e_2 \mid c \times e$

Atomic facts $g := e \geq 0 \mid e \neq 0$

Note that $e=0$ can be represented as $e \geq 0 \wedge e \leq 0$

$e>0$ can be represented as $e-1 \geq 0$

(over integer LA)

- The decision problem for integer LA is NP-hard.
- The decision problem for rational LA is PTime.
  - PTime algorithms are complicated to implement. Popular choice is an exponential algorithm called "Simplex"
  - We will study a PTime algorithm for a special case.

Difference Constraints

- A special case of Linear Arithmetic
- Constraints of the form $x \leq c$ and $x-y \leq c$
  - We can represent $x \leq c$ by $x-u \leq c$, where u is a special zero variable. Wlog, we will assume henceforth that we only have constraints $x-y \leq c$
- Reasoning required: $x-y \leq c_1 \wedge y-z \leq c_2 \Rightarrow x-z \leq c_1+c_2$
- $O(n^3)$ (saturation-based) decision procedure
  - Represent contraints by a matrix $M_{n \times n}$
    - where $M[i][j] = c$ represents $x_i - x_j \leq c$
  - Perform transitive closure of M
    - $M[i][j] = \min \{ M[i][j], M[i][k]+M[k][j] \}$
  - ϕ is unsat iff $\exists i: M[i][i] < 0$

Outline

- Decision Procedures
  - Linear Arithmetic
  - ➢ Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
- Logical Abstract Interpretation
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
  - Universally Quantified Formulas
- Hardness of Assertion Checking
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns

Uninterpreted Functions

Expressions $e := x \mid F(e_1,e_2)$

Atomic fact $g := e_1=e_2 \mid e_1 \neq e_2$

Axiom $\forall e_1,e_2,e_1',e_2': e_1=e_1' \wedge e_2=e_2' \Rightarrow F(e_1,e_2)=F(e_1',e_2')$

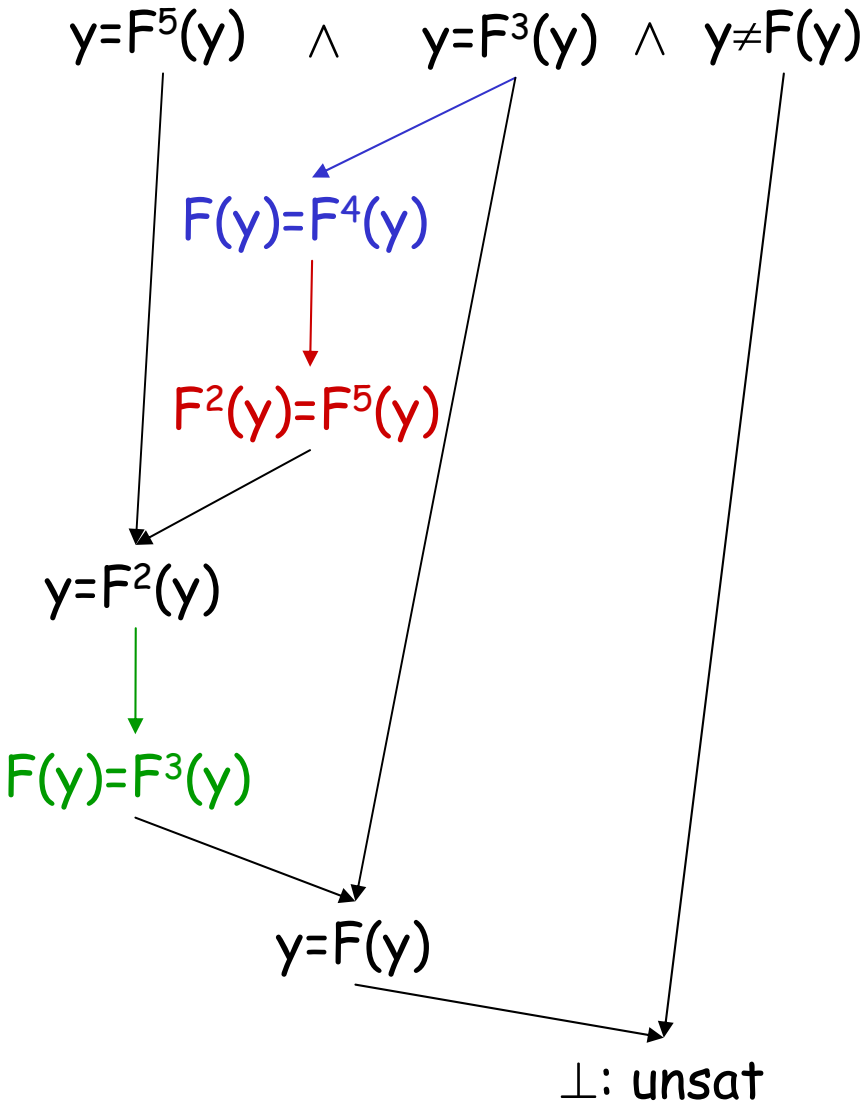
(called congruence axiom)

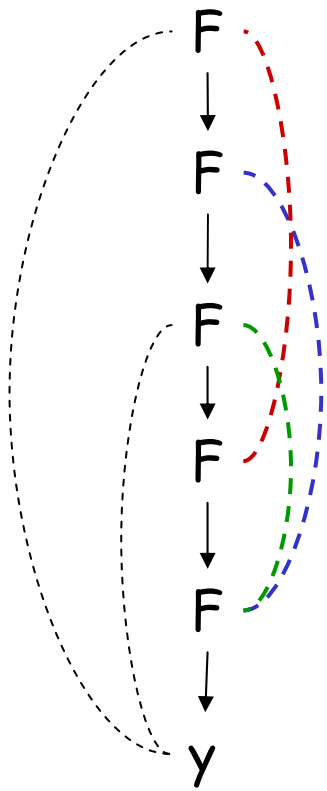
(saturation-based) Decision Procedure

- Represent equalities $e_1=e_2 \in G$ in Equivalence DAG (EDAG)
  - Nodes of an EDAG represent congruence classes of expressions that are known to be equal.
- Saturate equalities in the EDAG by following rule:
  - If $C(e_1)=C(e_1') \wedge C(e_2)=C(e_2')$, Merge $C(F(e_1,e_2))$, $C(F(e_1',e_2'))$

  where $C(e)$ denotes congruence class of expression e
- Declare unsatisfiability iff $\exists\, e_1 \neq e_2$ in G s.t. $C(e_1) = C(e_2)$

Uninterpreted Functions: Example





Uninterpreted Functions: Complexity

- Complexity of congruence closure : $O(n \log n)$, where n is the size of the input formula
  - In each step, we merge 2 congruence classes. The total number of steps required is thus n, where n is a bound on the original number of congruence classes.
  - The complexity of each step can be $O(\log n)$ by using union-find data structure

Outline

- Decision Procedures
  - Linear Arithmetic
  - Uninterpreted Functions
  - ➢ Combination of Linear Arithmetic and Uninterpreted Fns
- Logical Abstract Interpretation
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
  - Universally Quantified Formulas
- Hardness of Assertion Checking
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns

Combination of Linear Arithmetic and Uninterpreted Functions

Expressions $e := y \mid c \mid e_1 \pm e_2 \mid c \times e \mid F(e_1,e_2)$

Atomic Facts $g := e \geq 0 \mid e \neq 0$

Axioms: Combined axioms of linear arithmetic + uninterpreted fns.

Decision Procedure: Nelson-Oppen methodology for combining decision procedures

Combining Decision Procedures

- Nelson-Oppen gave an algorithm in 1979 to combine decision procedures for theories T_1 and T_2, where:
  - T_1 and T_2 have disjoint signatures
    - except equality
  - T_1, T_2 are stably infinite
- Complexity is $O(2^{n^2} \times (W_1(n)+W_2(n))$.
- If T_1, T_2 are convex, complexity is $O(n^3 \times (W_1(n)+W_2(n))$.

The theories of linear arithmetic and uninterpreted functions satisfy all of the above criterions.

Convex Theory

A theory is convex if the following holds.

Let $G = g_1 \wedge \ldots \wedge g_n$
If $G \Rightarrow e_1{=}e_2 \vee e_3{=}e_4$, then $G \Rightarrow e_1{=}e_2$ or $G \Rightarrow e_3{=}e_4$

Examples of convex theory:

- Rational Linear Arithmetic
- Uninterpreted Functions

Examples of Non-convex Theory

- Theory of Integer Linear Arithmetic

  $2 \leq y \leq 3 \Rightarrow y=2 \vee y=3$

  But $2 \leq y \leq 3 \not\Rightarrow y=2$ and $2 \leq y \leq 3 \not\Rightarrow y=3$

- Theory of Arrays

  $y=sel(upd(M,a,0),b) \Rightarrow y=0 \vee y=sel(M,b)$

  But $y=sel(upd(M,a,0),b) \Rightarrow y \neq 0$ and

  $y=sel(upd(M,a,0),b) \Rightarrow y \neq sel(M,b)$

Stably Infinite Theory

- A theory T is stably infinite if for all quantifier-free formulas ϕ over T, the following holds:

  If ϕ is satisfiable, then ϕ is satisfiable over an infinite model.

- Examples of stably infinite theories
  - Linear arithmetic, Uninterpreted Functions

- Examples of non-stably infinite theories
  - A theory that enforces finite # of distinct elements. Eg., a theory with the axiom: $\forall x,y,z\ (x=y \vee x=z \vee y=z)$. Consider the quantifier free formula ϕ: $y_1=y_2$.

    ϕ is satisfiable but doesn't have an infinite model.

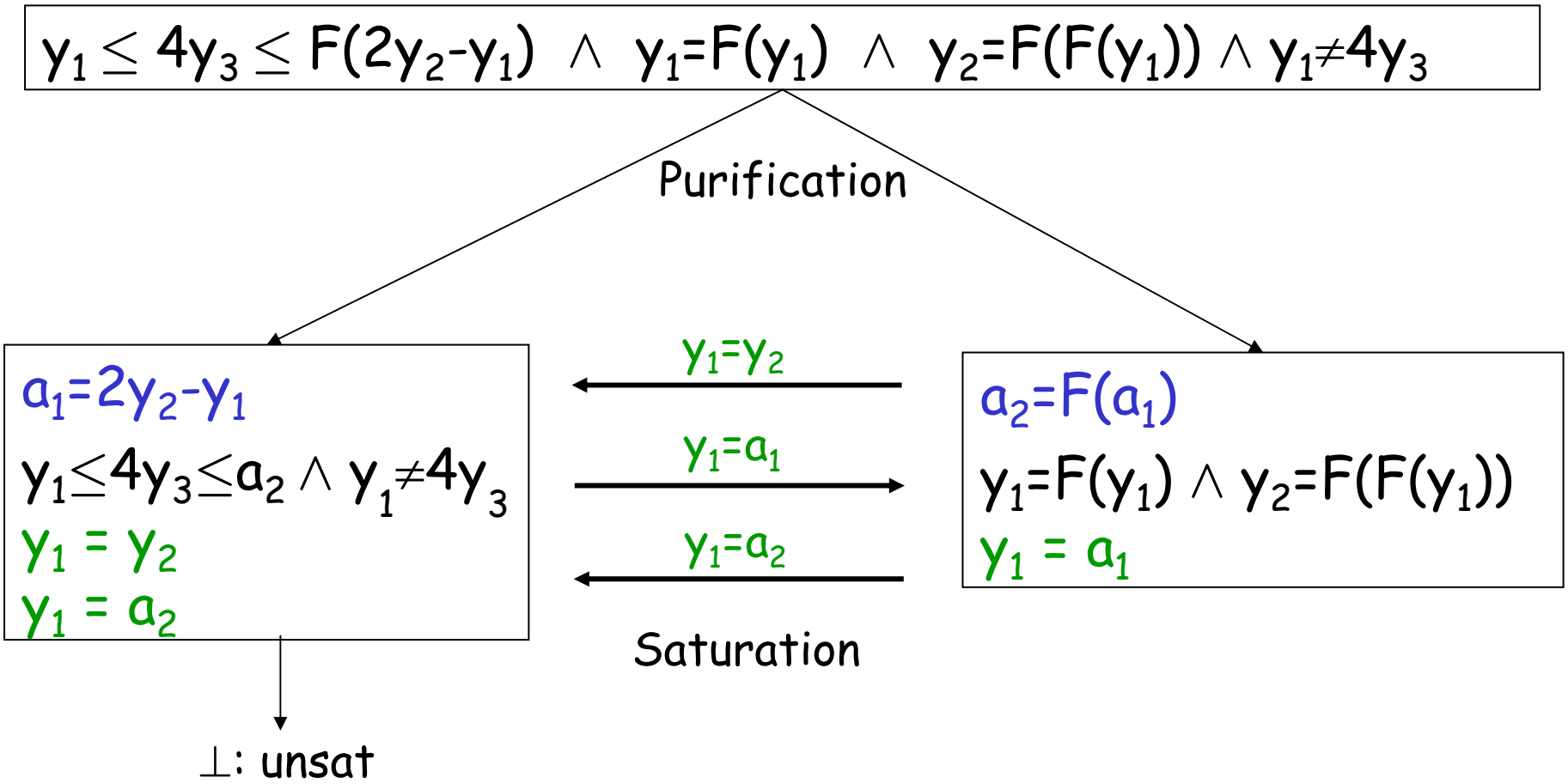
Nelson-Oppen Methodology

- Purification: Decompose ϕ into $\phi_1 \wedge \phi_2$ such that ϕ_i contains symbols from theory T_i.
  - This can be done by introducing dummy variables.

- Exchange variable equalities between ϕ_1 and ϕ_2 until no more equalities can be deduced.
  - Sharing of disequalities is not required because of stably-infiniteness.
  - Sharing of disjunctions of equalities is not required because of convexity.

- ϕ is unsat iff ϕ_1 is unsat or ϕ_2 is unsat.

Combining Decision Procedures: Example



Outline

- Decision Procedures
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
- ➢ Logical Abstract Interpretation
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
  - Universally Quantified Formulas
- Hardness of Assertion Checking
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns

Logical Abstract Interpretation

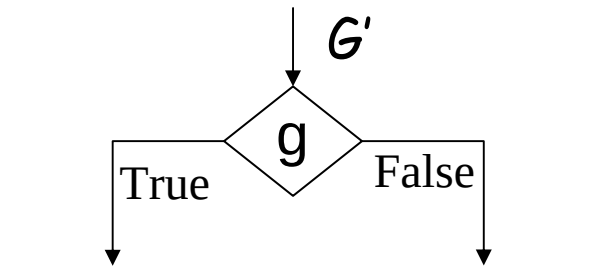
- Abstract Interpretation of a program involves interpreting the program over abstract values from some abstract domain D equipped with a partial order $\preceq$

- Logical Abstract Interpretation refers to the case when
  - D = logical formulas over theory T
  - $\preceq$ = logical implication relationship, i.e., $E \preceq E'$ iff $E \Rightarrow_T E'$

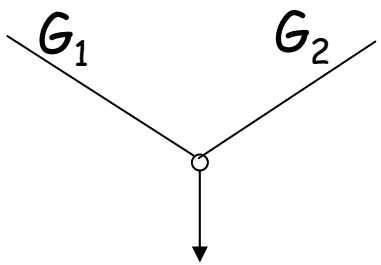
- We will study following examples of logical interpretation
  - D consists of finite conjunctions of atomic facts over T.
    - Linear Arithmetic
    - Uninterpreted Functions
    - Combination of Linear Arithmetic and Uninterpreted Functions
  - D consists of universally quantified formulas over T.

Transfer Functions for Logical Abstract Interpreter

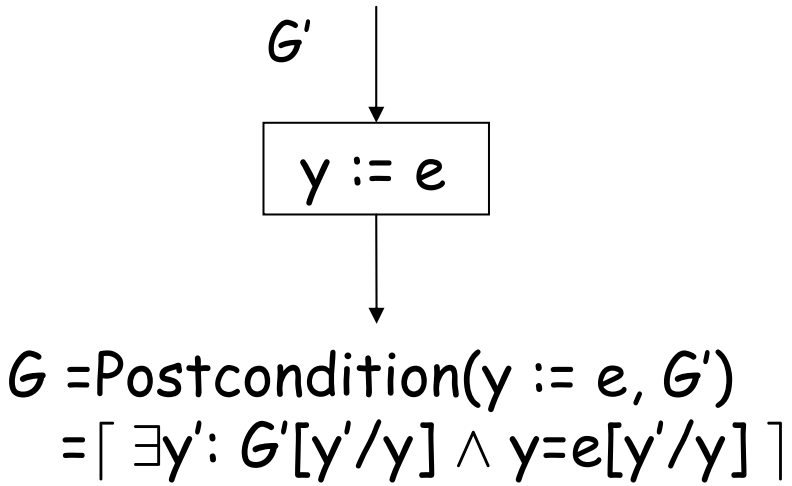
- An abstract interpreter computes abstract values or facts at each program point from facts at preceding program points using appropriate transfer fns.



Conditional Node



Join Node

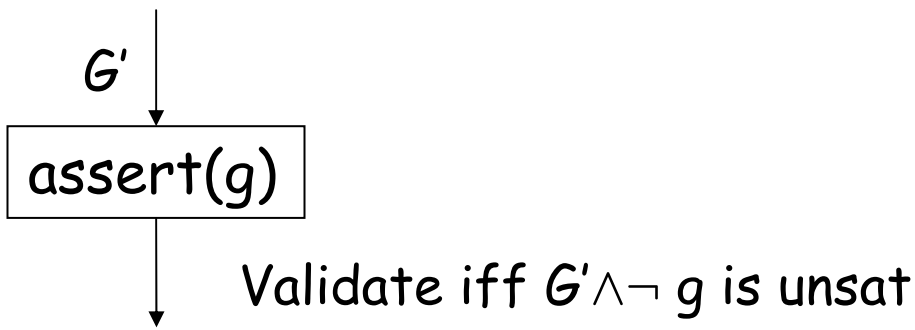


Assignment Node

- Transfer functions for a logical abstract interpreter thus involve providing operators for over-approximating disjunction and existential quantifier elimination.

Fixed-point Computation

- In presence of loops, fixed-point computation is required. The process is accelerated by using a widening operator, which takes the facts in the current and previous iteration (at some point inside a loop) and generates something weaker than the current fact.
  - A widening operator should guarantee convergence in a bounded number of steps.
  - Widening is typically applied at loop header points.
- Facts generated after fixed-point are invariants and can be used to validate assertions using decision procedures.



Initialization

- The fact at program entry is initialized to $\top$, which in our setting is the logical formula "true".
  - This denotes that we make no assumptions about inputs, and whatever we prove will be valid for all inputs.
- The facts at all other program points are initialized to $\bot$, which in our setting is the logical formula "false".
  - This denotes our optimistic assumption of unreachability of program locations (unless we can prove them reachable in the process of fixed-point computation).

Outline

- Decision Procedures
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
- Logical Abstract Interpretation
  - ➢ Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
  - Universally Quantified Formulas
- Hardness of Assertion Checking
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns

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' := Saturate(M)
  2. Declare unsat iff $\exists i$: $M'[i][i] < 0$

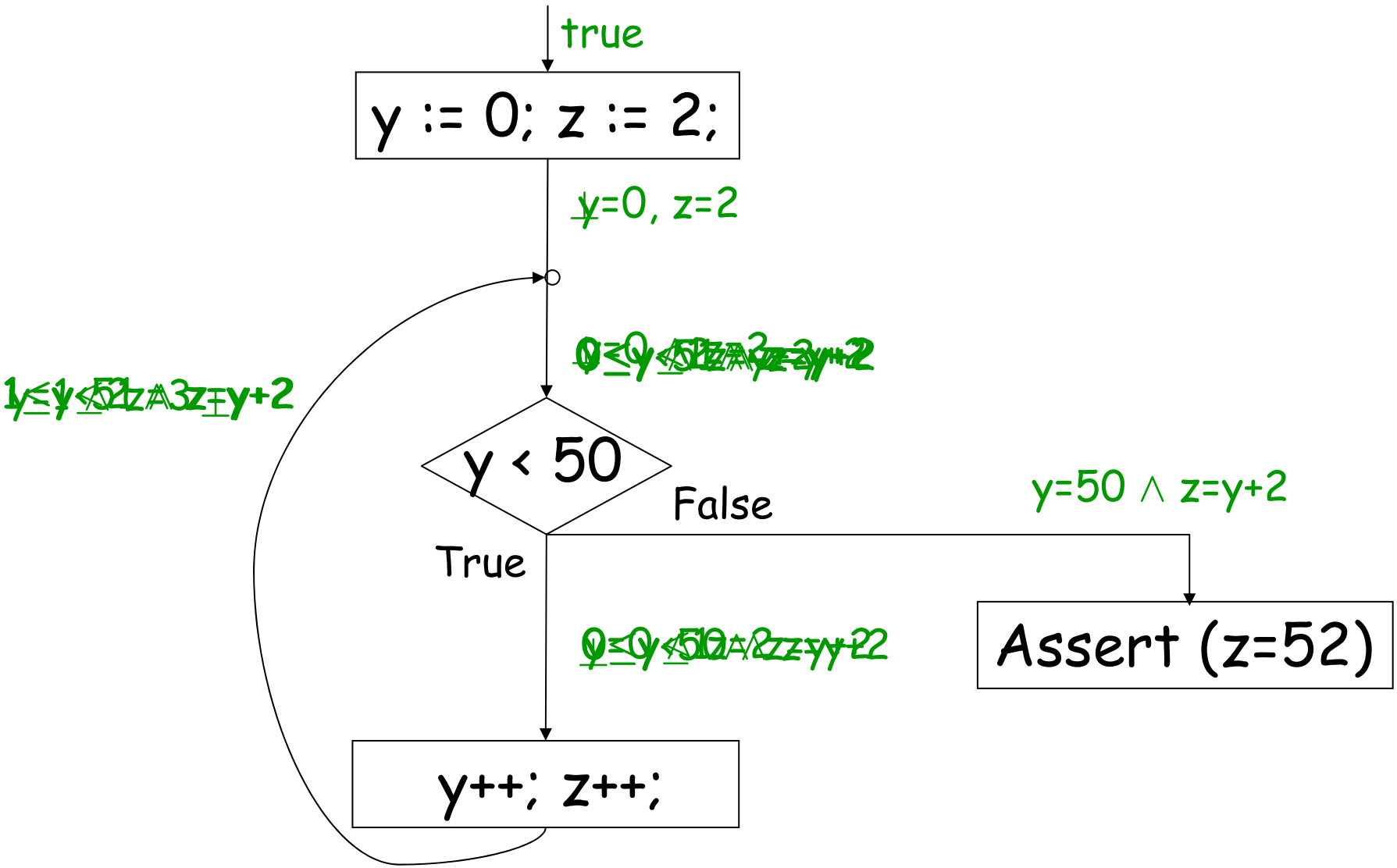
- Join(M_1, M_2):
  1. M'_1 := Saturate(M_1); M'_2 := Saturate(M_2);
  2. Let M_3 be s.t. $M_3[i][j]$ = Max { $M'_1[i][j]$, $M'_2[i][j]$ }
  3. return M_3

Difference Constraints

- Eliminate(M, x_i):
  1. M' := 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

- Widen(M_1, M_2):
  1. M'_1 := Saturate(M_1); M'_2 := 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

Difference Constraints: Example



Outline

- Decision Procedures
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
- Logical Abstract Interpretation
  - Linear Arithmetic
  - ➢ Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
  - Universally Quantified Formulas
- Hardness of Assertion Checking
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns

Uninterpreted Functions

- Abstract element:
  - conjunction of $e_1=e_2$, where e := y | $F(e_1,e_2)$
  - can be represented using EDAGs

- Decide(G):
  1. G' := 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.

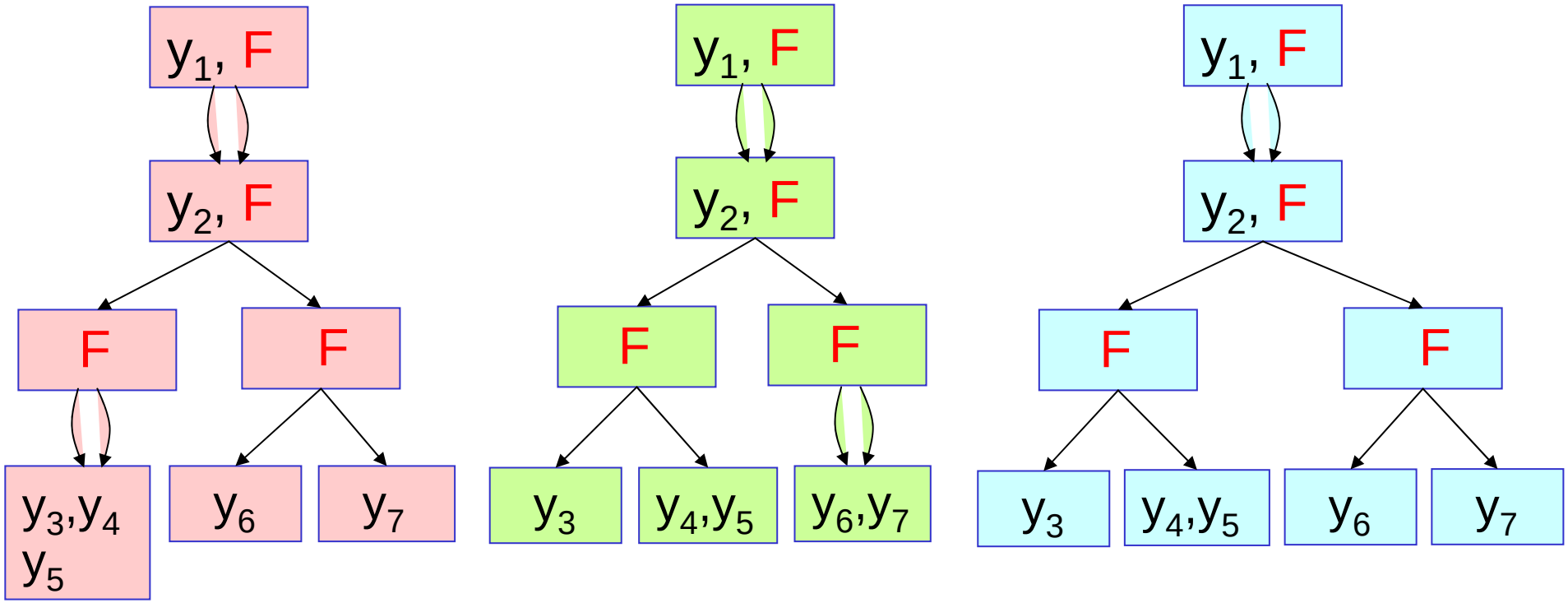
- Eliminate(G, y):
  1. G' := Saturate(G);
  2. Erase y; (might need to delete some dangling expressions)
  3. return G'

Uninterpreted Functions

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

For each node $n = \langle U, \{n_i, n'_i\}\rangle$ in G'_1
and node $m = \langle V, \{m_j, m'_j\}\rangle$ in G_2',
G contains a node $[n,m] = \langle U \cap V, \{[n_i, m_j], [n'_i, m'_j]\}\rangle$

Uninterpreted Functions: Example of Join



G_1

G_2

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

Outline

- Decision Procedures
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
- Logical Abstract Interpretation
  - Linear Arithmetic
  - Uninterpreted Functions
  - ➢ Combination of Linear Arithmetic and Uninterpreted Fns
  - Universally Quantified Formulas
- Hardness of Assertion Checking
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns

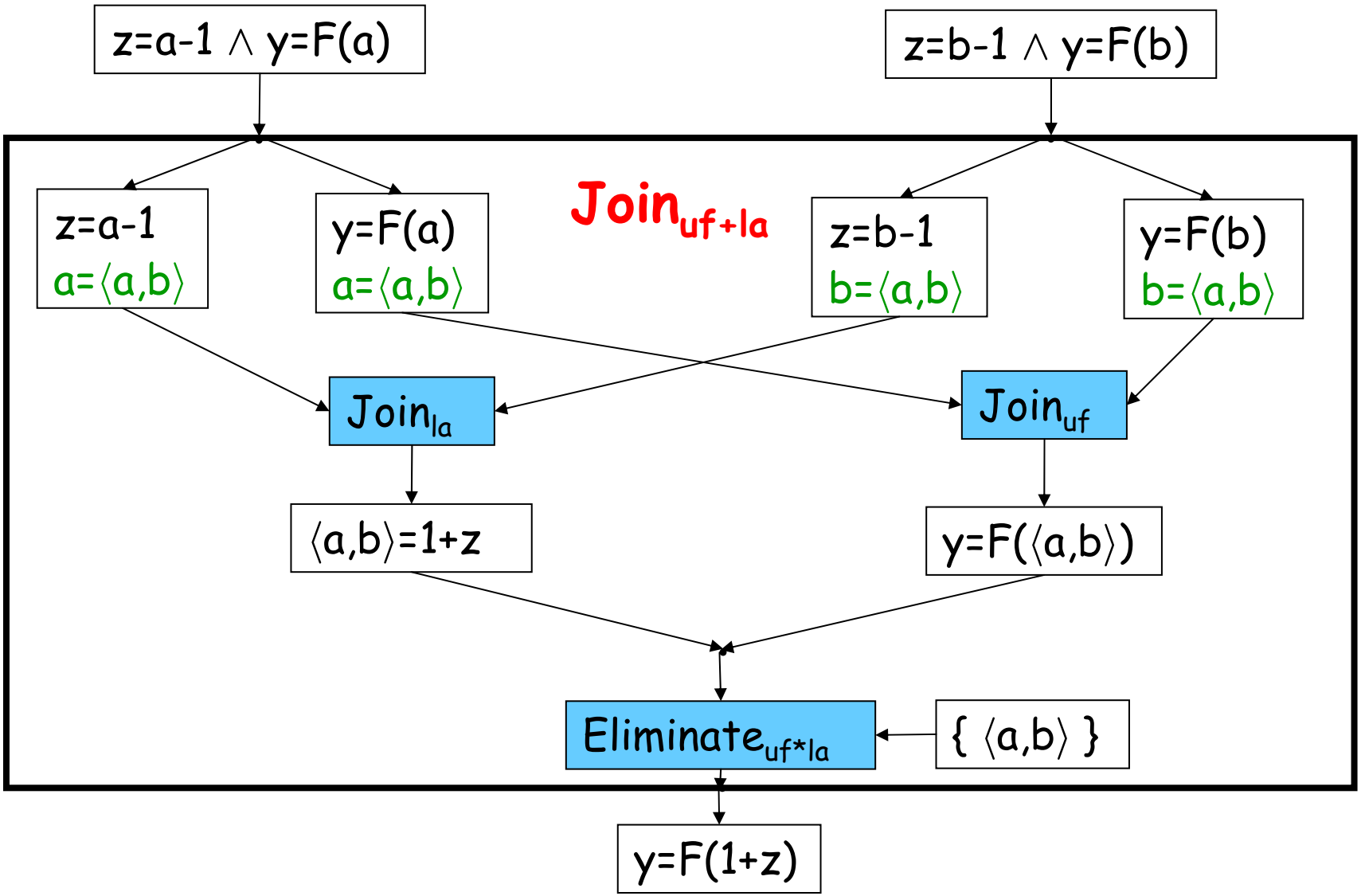
Combination: Decision Procedure

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

Combination: Join Algorithm

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

Combination: Example of Join Algorithm

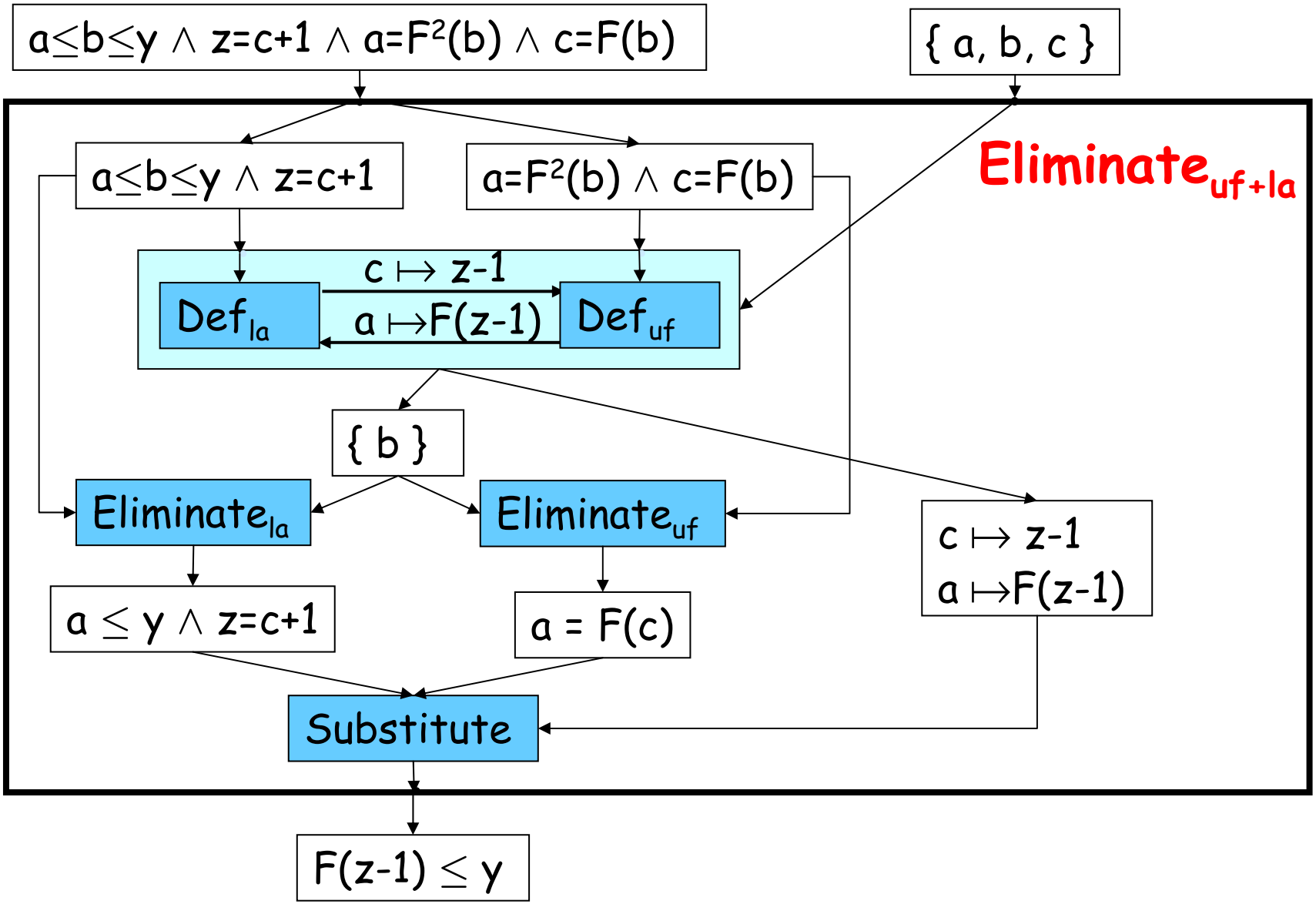


Combination: Existential Quantifier Elimination

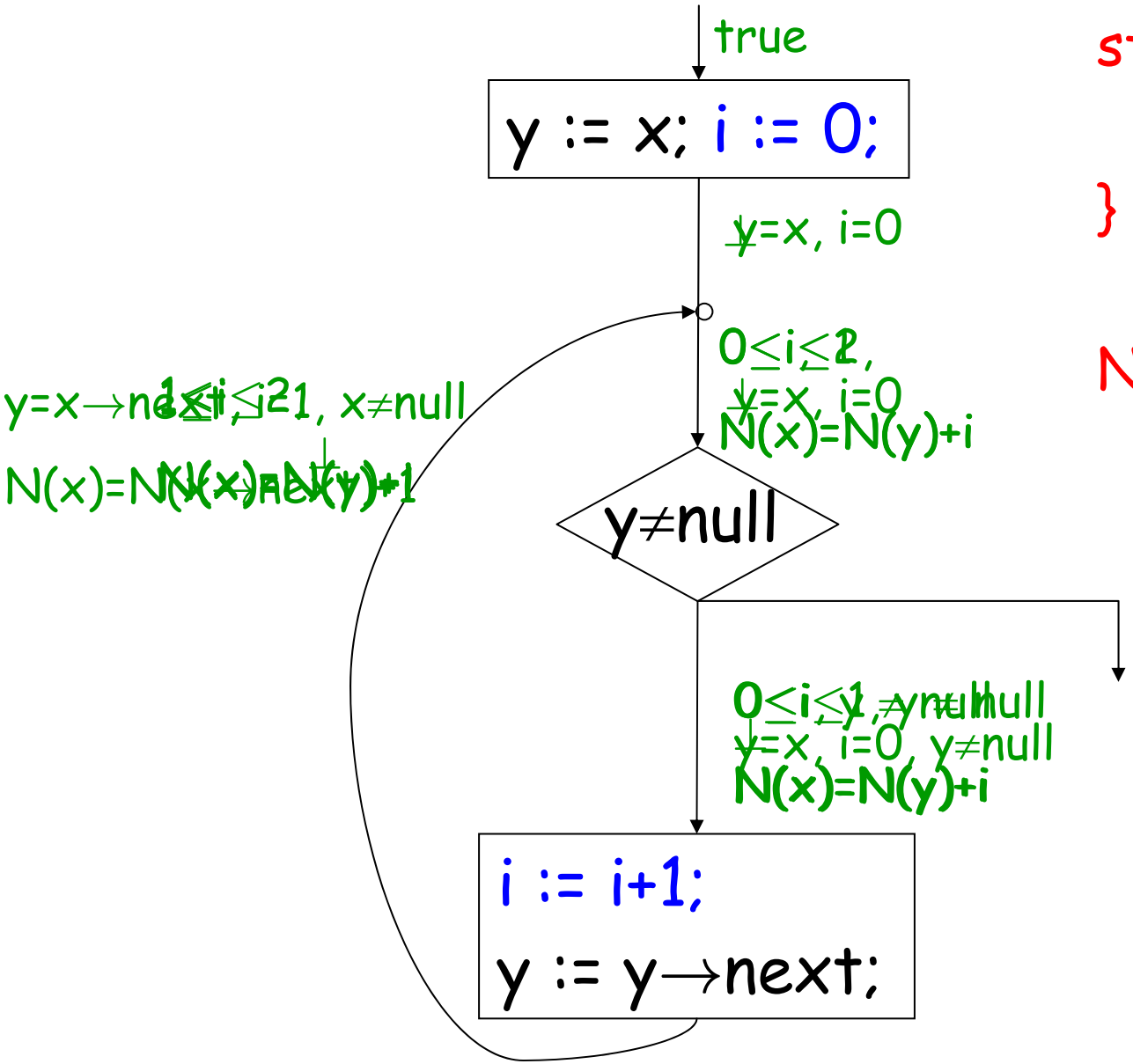
- $\text{Elimintate}_{T_{12}}(E_{12}, V)$:
  1. $\langle E_1, E_2 \rangle$:= Purify&Saturate(E_{12});
  2. $\langle D, Defs \rangle$:= DefSaturate(E_1, E_2, $V \cup$ Temp Variables);
  3. V' := $V \cup$ Temp Variables – D;
     E'_1 := $\text{Eliminate}_{T_1}(E_1, V')$;
     E'_2 := $\text{Eliminate}_{T_2}(E_2,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

DefSaturate(E_1, E_2, U) returns the set of all variables D that have definitions Defs in terms of variables not in U as implied by $E_1 \wedge E_2$.

Combination: Example of Existential Elimination



Abstract Interpretation over Combined Domain: Example




```
struct List {
   struct List* next;
} x, y;
```

N(z) = 0, if z = null
= 1 + N(z→next)

Outline

- Decision Procedures
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
- Logical Abstract Interpretation
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
  - ➢ Universally Quantified Formulas
- Hardness of Assertion Checking
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns

Universally Quantified Abstract Domain

- Abstract element is of the form $E \wedge \bigwedge_i(\forall X: A_i \Rightarrow B_i)$
  - where E, A_i, B_i are from some underlying base domain(s) D.

- The partial order $\preceq$ is a refinement of the more natural implication relationship.

  $E \wedge \bigwedge_i(\forall X: A_i \Rightarrow B_i) \preceq E' \wedge \bigwedge_j(\forall X: A'_j \Rightarrow B'_j)$ iff
  - $E \Rightarrow E'$
  - $\forall j \; \exists i$ s.t. $E \wedge A'_j \Rightarrow A_i$ and $E \wedge B_i \Rightarrow B'_j$

- Another way to state the above thing would be to say that the partial order is still the implication relationship but transfer functions are incomplete.

Quantified Abstract Domain: Join Algorithm

- Consider a simpler case first.
  Let $(E \wedge \forall X{:}A \Rightarrow B) = \text{Join}(E^1 \wedge \forall X{:}A^1 \Rightarrow B^1,\ E^2 \wedge \forall X{:}A^2 \Rightarrow B^2)$. Then,
  - $(E^1 \wedge \forall X{:}A^1 \Rightarrow B^1) \preceq (E \wedge \forall X{:}A \Rightarrow B)$
  - $(E^2 \wedge \forall X{:}A^2 \Rightarrow B^2) \preceq (E \wedge \forall X{:}A \Rightarrow B)$

  Or, equivalently,
  - $E^1 \Rightarrow E$ and $E^2 \Rightarrow E$. Thus, $E = \text{Join}(E^1, E^2)$.
  - $E^1 \wedge A \Rightarrow A^1$ and $E^2 \wedge A \Rightarrow A^2$, i.e., $A \Rightarrow (E^1 \Rightarrow A^1 \wedge E^2 \Rightarrow A^2)$. Thus, $A = \lfloor E^1 \Rightarrow A^1 \wedge E^2 \Rightarrow A^2 \rfloor$.
  - $E^1 \wedge B^1 \Rightarrow B$ and $E^2 \wedge B^2 \Rightarrow B$. Thus, $B = \text{Join}(E^1 \wedge B^1, E^2 \wedge B^2)$.

- $\text{Join}(E^1 \wedge \bigwedge_i(\forall X{:}\ A^1{}_i \Rightarrow B^1{}_i), E^2 \wedge \bigwedge_i(\forall X{:}\ A^2{}_i \Rightarrow B^2{}_i))$:
  1. result := $\text{Join}_D(E^1, E^2)$;
  2. Forall i,j:
  3. $A := \lfloor E^1 \Rightarrow A^1{}_i \wedge E^2 \Rightarrow A^2{}_j \rfloor$; $B := \text{Join}_D(E^1 \wedge B^1{}_i, E^2 \wedge B^2{}_j)$;
  4. result := result $\wedge \forall X{:}A \Rightarrow B$
  5. return result;

Quantified Abstract Domain: Example of Join

Let $G_1 = (i=0 \wedge \forall k: k=0 \Rightarrow F[k]=i)$

$G_2 = (i=1 \wedge \forall k: 0 \leq k \leq 1 \Rightarrow F[k]=0)$

Then $Join(G_1, G_2) = 0 \leq i \leq 1 \wedge \forall k: A \Rightarrow B$, where

$A = \lfloor (i=0 \Rightarrow k=0) \wedge (i=1 \Rightarrow 0 \leq k \leq 1) \rfloor = 0 \leq k \leq i$

$B = Join_D(i=0 \wedge F[k]=i, i=1 \wedge F[k]=0) = F[k]=0$

Quantified Abstract Domain: Eliminate

Let $(E' \wedge \forall X{:}A' \Rightarrow B') = \text{Eliminate}(E \wedge \forall X{:}A \Rightarrow B, s)$. Then,

$(E \wedge \forall X{:}A \Rightarrow B) \preceq (E' \wedge \forall X{:}A' \Rightarrow B')$ among other things.

For simplicity, assume that s doesn't affect terms in A,B involving X. Then,

- $E \Rightarrow E'$ and E' doesn't contain any term affected by change to s.
  - Thus, $E' = \text{Eliminate}_D(E,s)$.
- $E \wedge A' \Rightarrow A$ and A doesn't contain any term affected by change to s.
  - Thus, $A' = \lfloor \forall s{:}E \Rightarrow A \rfloor$.
- $E \wedge B \Rightarrow B'$ and B' doesn't contain any term affected by change to s.
  - Thus, $B' = \text{Eliminate}_D(E \wedge B, s)$.

Quantified Abstract Domain: Eliminate

- Eliminate(G, s):
  1. Let G be $E \wedge \forall X{:}A \Rightarrow B$
     - Psuedo-code can be easily extended for multiple $\forall$
  2. T := { e | e occurs in A or B; $Vars(e) \cap X \neq \emptyset$ }
  3. $A := A \wedge \bigwedge_{e \in T} NotEffect(\langle s,G \rangle, e)$;
  4. $E' := Eliminate_D(E,s)$;
  5. $B' := Eliminate_D(B \wedge E,s)$;
  6. $A' := \lfloor \forall s{:}E \Rightarrow A \rfloor$;
  7. return $(E' \wedge \forall X{:}\ A' \Rightarrow B')$

- NotEffect($\langle s,G \rangle$, e) denotes a constraint g s.t. $G \wedge g$ implies that s does not affect e.

Quantified Abstract Domain: Example of Eliminate

Let $G = (F[0]>10 \wedge \forall k: 0 \le k < F[0] \Rightarrow F[k]>F[0])$

Then Eliminate(G, F[0]) = true $\wedge \forall k: A' \Rightarrow B'$, where

T = { k, F[k] }

NotEffect(<F[0],G>, F[k]) = $k \neq 0$

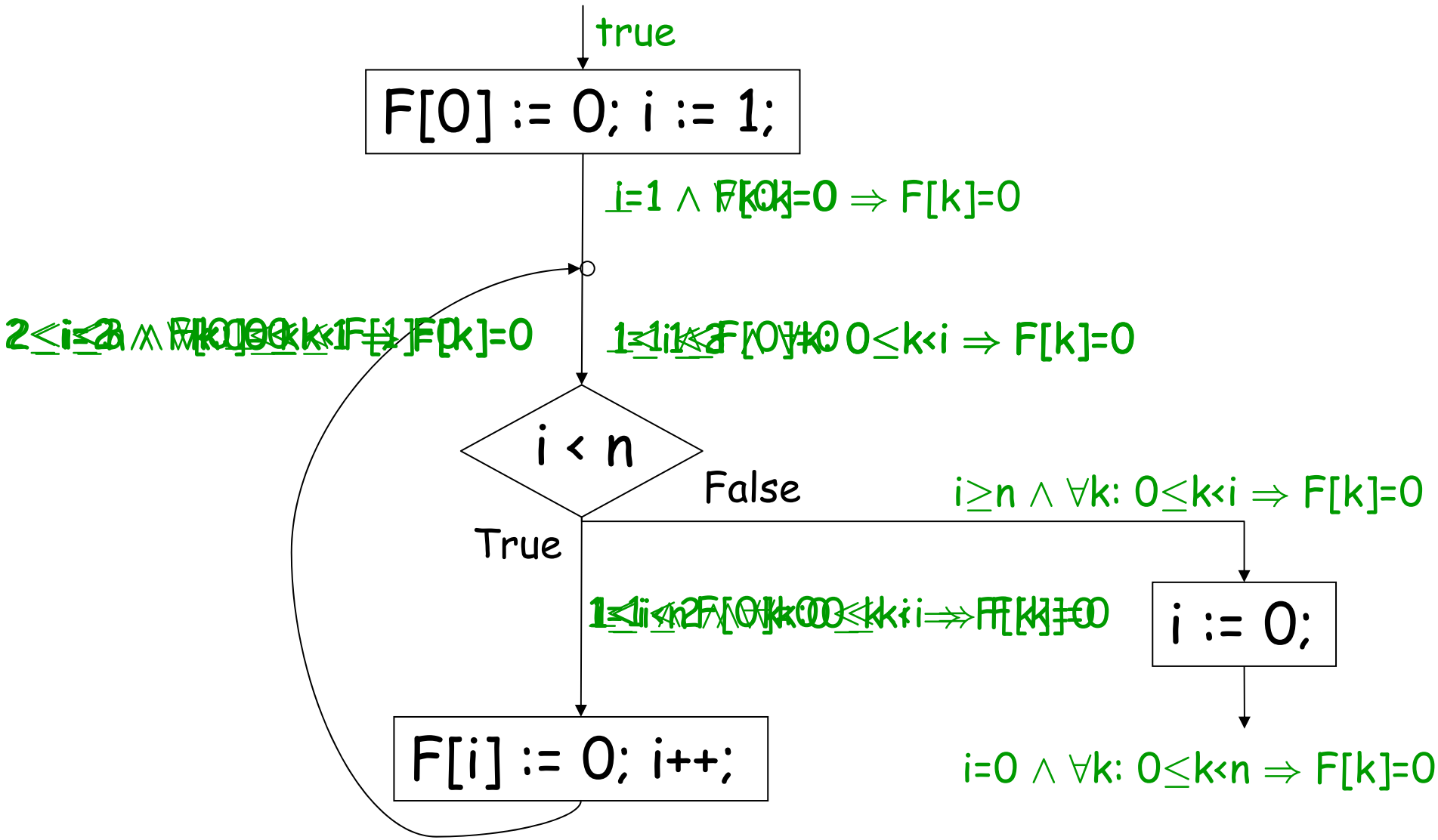
NotEffect(<F[0],G>, F[k]) = true

$A_1 = 0 \le k < F[0] \wedge k \neq 0 \wedge \text{true} = 1 \le k < F[0]$

$A' = \lfloor \forall F[0]: F[0]>10 \Rightarrow 1 \le k < F[0] \rfloor = 1 \le k < 10$

B' = Eliminate($F[k]>F[0] \wedge F[0]>10$, F[0]) = $F[k]<10$

Quantified Abstract Domain : Example



References

- Uninterpreted Functions
  - "A polynomial time algorithm for global value numbering" SAS 2004, S. Gulwani, G. Necula
  - "Join algorithms for the theory of uninterpreted fns" FSTTCS 2004, S. Gulwani, A. Tiwari, G. Necula

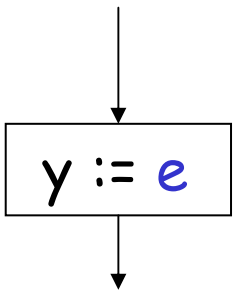
- Combination of Linear Arithmetic and Uninterpreted Fns
  - "Combining Abstract Interpreters" PLDI 2006, S. Gulwani, A. Tiwari

- Universally Quantified Abstract Domain
  - "Lifting Abstract Interpreters to Quantified Logical Domains" POPL 2008, S. Gulwani, B. McCloskey, A. Tiwari

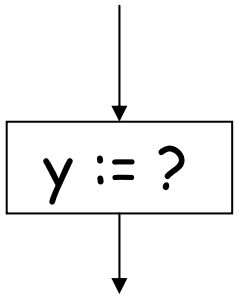
Outline

- Decision Procedures
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
- Logical Abstract Interpretation
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
  - Universally Quantified Formulas
- ➢ Hardness of Assertion Checking
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns

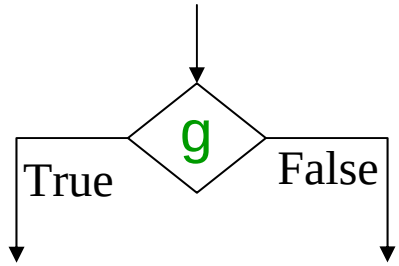
Abstract Program Model / Problem Statement



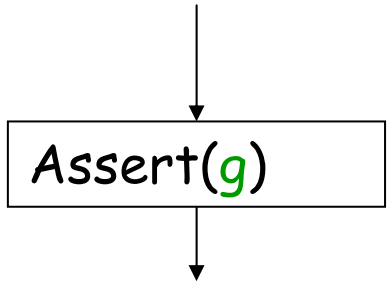
Assignment



Non-deterministic Assignment



Conditional



Assertion

Linear Arithmetic

$e := y \mid c \mid e_1 \pm e_2 \mid c\ e$ $\qquad g := e \geq 0$

Uninterpreted Functions

$e := y \mid F(e_1,e_2)$ $\qquad g := e_1 = e_2$

Combination

$e := y \mid c \mid e_1 \pm e_2 \mid c\ e \mid F(e_1,e_2)$ $\qquad g := e \geq 0$

Outline

- Decision Procedures
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
- Logical Abstract Interpretation
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
  - Universally Quantified Formulas
- Hardness of Assertion Checking
  - ➢ Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns

Assertion Checking: Linear Arithmetic

- Non-deterministic Conditionals
  - Equality Assertions: PTIME
    - Perform abstract interpretation over linear equalities.
    - "Affine relationships among variables of a program", Karr 76
  - Inequality assertions: ?

- Deterministic Conditionals: Undecidable
  - Even with equality conditionals and equality assertions
  - PCP Problem can be reduced to it.
  - "A Note on Karr's Algorithm", H. Seidl, M. Muller-Olm, ICALP 2004

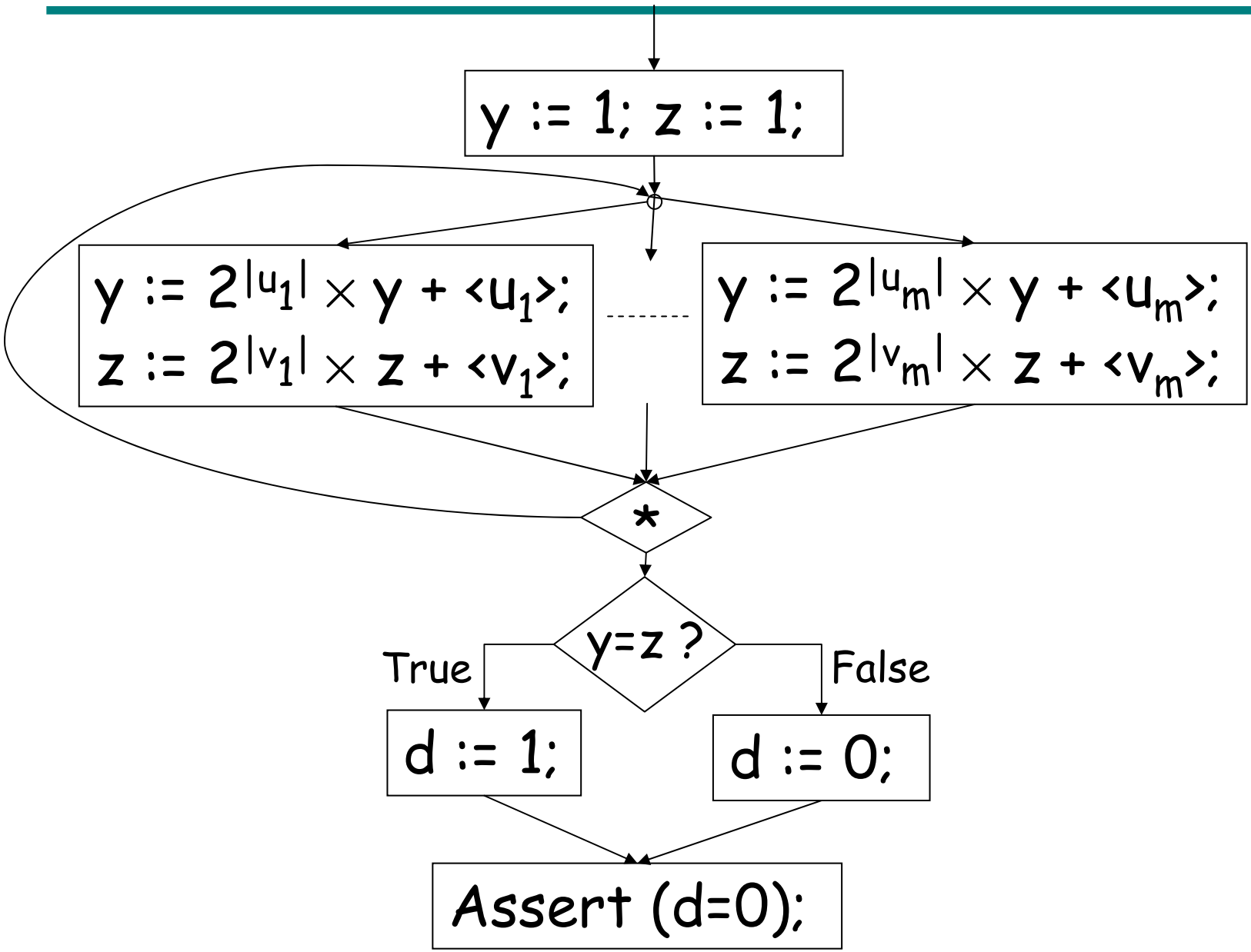
Reducing PCP Problem to Assertion Checking

- The following problem (PCP Problem) is undecidable.
  Given pairs: (u_1,v_1), …, (u_m,v_m), where $u_i, v_i \in \{0,1\}^*$
  Decide: $\exists$ a non-empty sequence i_1, …, i_n such that

$$u_{i_1} \ldots u_{i_n} = v_{i_1} \ldots v_{i_n}$$

- Given a PCP instance, we will construct an assertion checking problem over linear arithmetic such that the assertion holds iff the solution to PCP instance is false.

Reducing PCP Problem to Assertion Checking



Outline

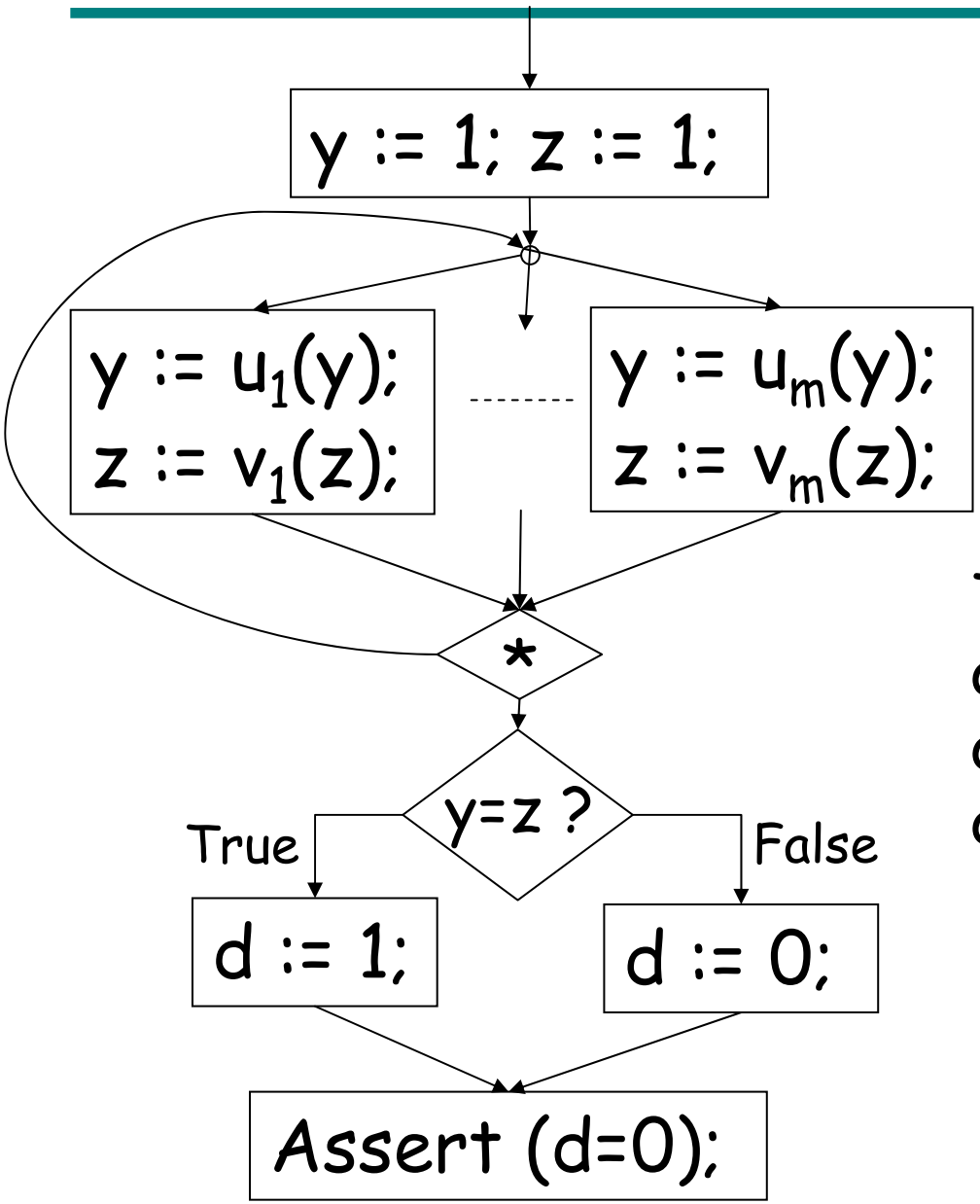
- Decision Procedures
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
- Logical Abstract Interpretation
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
  - Universally Quantified Formulas
- Hardness of Assertion Checking
  - Linear Arithmetic
  - ➢ Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns

Assertion Checking: Uninterpreted Functions

- Non-deterministic Conditionals: PTIME
  - Abstract interpretation over uninterpreted fns.

- Deterministic Conditionals: Undecidable
  - PCP Problem can be reduced to it.
  - "Checking Herbrand Equalities and Beyond", H. Seidl, M. Muller-Olm, VMCAI 2005

Reducing PCP Problem to Assertion Checking



Think of u_i, v_i as sequences of applications of unary fns, one corresponding to 0 and other corresponding to 1.

Outline

- Decision Procedures
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
- Logical Abstract Interpretation
  - Linear Arithmetic
  - Uninterpreted Functions
  - Combination of Linear Arithmetic and Uninterpreted Fns
  - Universally Quantified Formulas
- Hardness of Assertion Checking
  - Linear Arithmetic
  - Uninterpreted Functions
  - ➢ Combination of Linear Arithmetic and Uninterpreted Fns

Assertion Checking: Combination of Linear Arithmetic & Uninterpreted Functions

- Deterministic Conditionals: Undecidable
  - No surprise since problem is undecidable for individual cases of linear arithmetic and uninterpreted fns.

- Non-deterministic Conditionals: ? At least coNP-hard
  - Even for equality conditionals.
  - A surprising result since assertion checking for individual cases of linear arithmetic (equalities) and uninterpreted fns is PTIME, but not for combination.
  - In contrast, decision procedures for linear arithmetic and uninterpreted fns can be combined in PTIME using Nelson-Oppen methodology.
  - "Assertion checking over combination of linear arithmetic and uninterpreted fns", S. Gulwani, A. Tiwari, ESOP 2006

Reducing Unsatisfiability to Assertion Checking

ψ: boolean 3-SAT instance with m clauses

```
IsUnsatisfiable(ψ) {
   for j=1 to m
        c_j := 0;
   for i=1 to k do
       if (*)
            ∀ j s.t. var i occurs positively in clause j, c_j := 1;
       else
            ∀ j s.t. var i occurs negatively in clause j, c_j := 1;
   y = c_1 + c_2 + ... + c_m;
   Assert (y=0 ∨ y=1 ... ∨ y=m-1);
}
```

Encoding disjunction

- The check $y=1 \vee y=2$ can be encoded by the assertion $F(y) = F(1)+F(2)-F(3-y))$.

- The above trick can be recursively applied to construct an assertion that encodes $y=0 \vee y=1 \vee \ldots \vee y=m-1$
  - Eg., $y=0 \vee y=1 \vee y=2$ can be encoded by encoding $F(y)=F(0) \vee F(y)=F(1)+F(2)-F(3-y)$

Conclusion

- We showed how logical reasoning traditionally used in theorem proving can be exploited in an abstract interpretation setting.
  - We focused on conjunctive and universally quantified invariants over the domain of linear arithmetic and uninterpreted fns.

- There are several other interesting issues in program analysis that we did not address:
  - Destructive updates
    - Points-to analysis, Shape analysis
  - Path-sensitive analysis
    - Disjunctive invariants
  - Inter-procedural analysis
    - Procedure summaries