

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

$DP_{\top}(\phi) = \text{Yes}$, 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

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 constraints 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.

Uninterpreted Functions

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

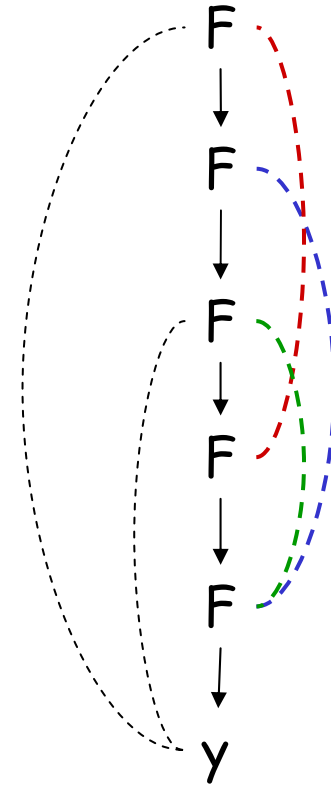
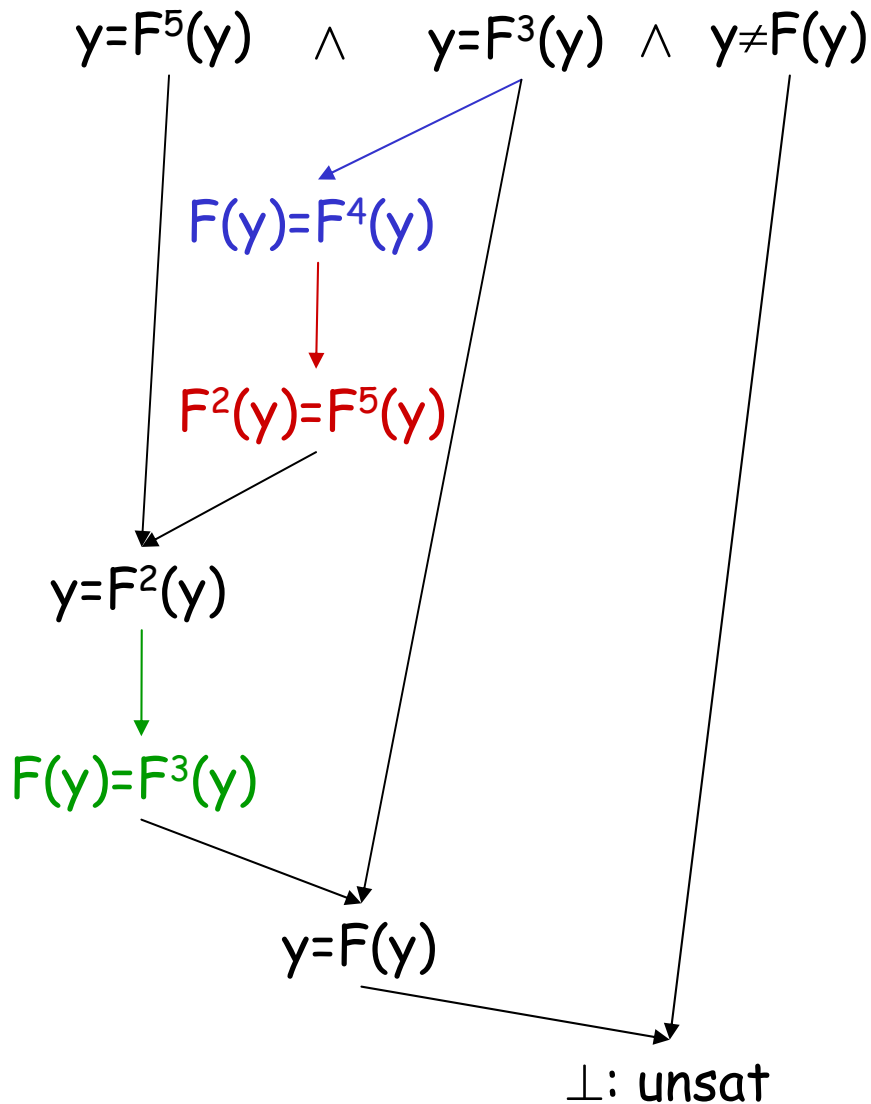
Atomic fact $g := e_1 = e_2 \mid e_1 \neq e_2$

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.

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 $O(n^4 \times (W_1(n) + W_2(n)))$.

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

Convex Theory

A theory is convex if the following holds.

Let $G = g_1 \wedge \dots \wedge g_n$

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

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

$$\text{But } 2 \leq y \leq 3 \not\Rightarrow y=2 \text{ and } 2 \leq y \leq 3 \not\Rightarrow y=3$$

- Theory of Arrays

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

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

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

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 $\phi: y_1=y_2$.
 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

$$y_1 \leq 4y_3 \leq F(2y_2 - y_1) \wedge y_1 = F(y_1) \wedge y_2 = F(F(y_1)) \wedge y_1 \neq 4y_3$$

Purification

$$a_1 = 2y_2 - y_1$$

$$y_1 \leq 4y_3 \leq a_2 \wedge y_1 \neq 4y_3$$

$$y_1 = y_2$$

$$y_1 = a_2$$

\perp : unsat

$$y_1 = y_2$$

$$y_1 = a_1$$

$$y_1 = a_2$$

Saturation

$$a_2 = F(a_1)$$

$$y_1 = F(y_1) \wedge y_2 = F(F(y_1))$$

$$y_1 = a_1$$

Outline

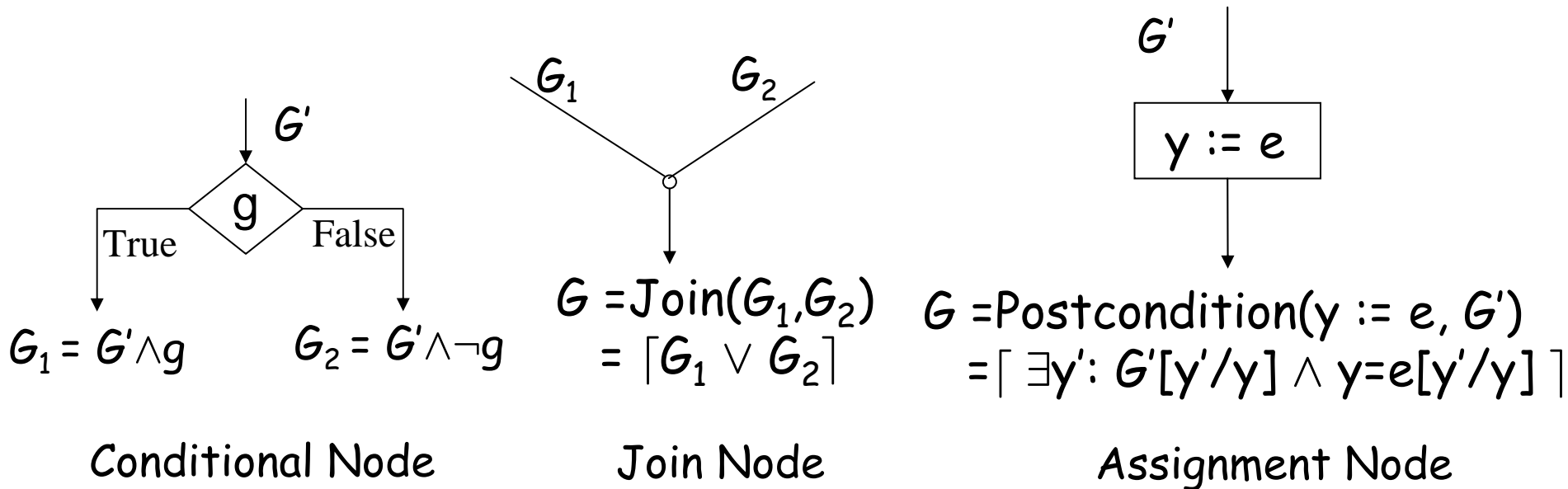
- Decision Procedures
 - Linear Arithmetic
 - Uninterpreted Functions
 - Combination of Linear Arithmetic and Uninterpreted Fns
- Logical Abstract Interpretation
 - Linear Arithmetic
 - Uninterpreted Functions
 - Combination of Linear Arithmetic and Uninterpreted Fns

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

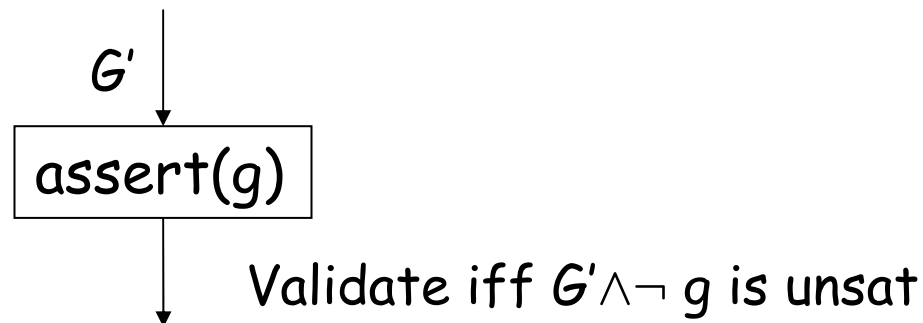
- An abstract interpreter computes abstract values or facts at each program point from facts at preceding program points using appropriate transfer fns.



- 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 \perp , 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

Difference Constraints

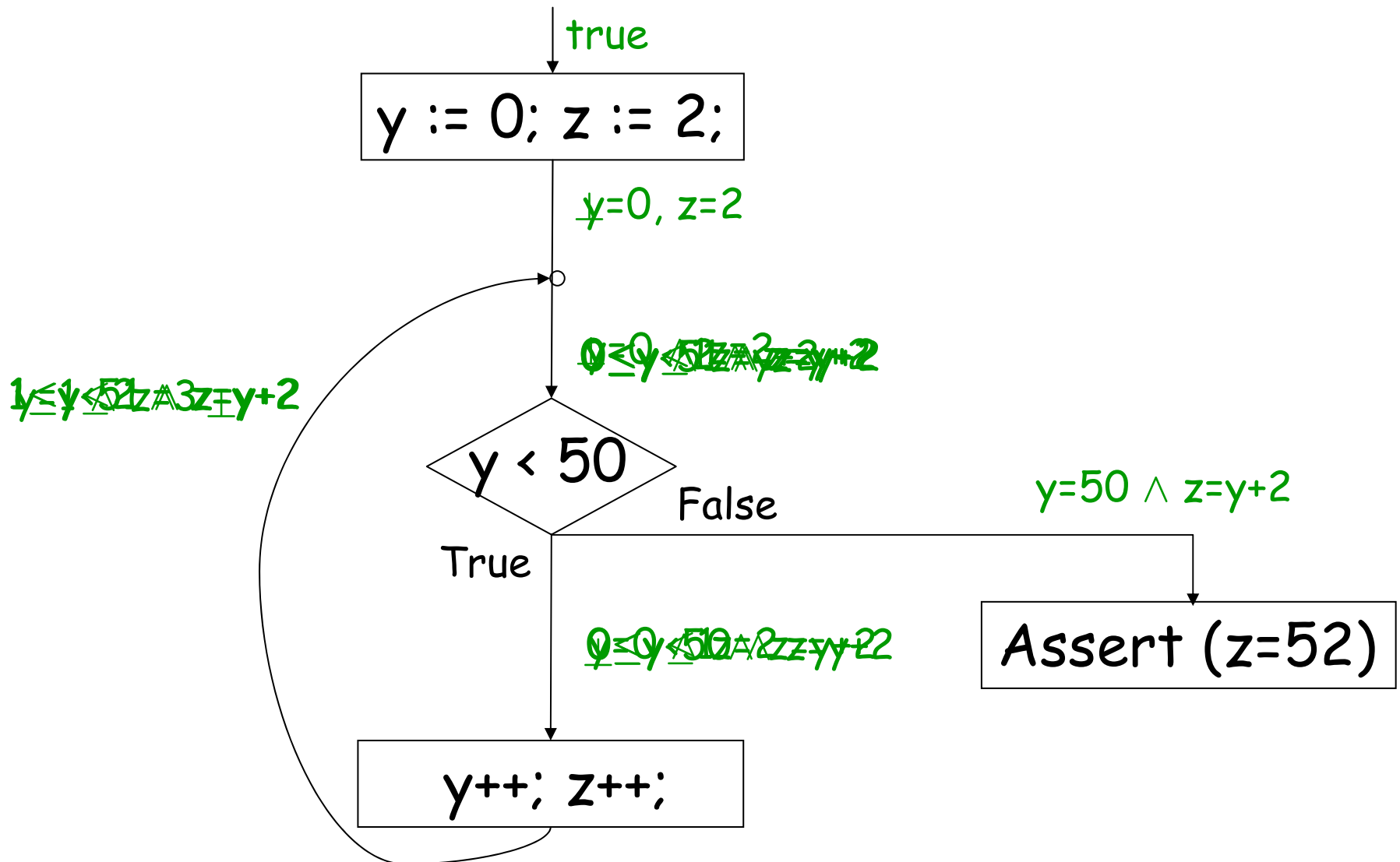
- Abstract element:
 - conjunction of $x_i - x_j \leq c_{ij}$
 - can be represented using matrix M , where $M[i][j] = c_{ij}$
- Decide(M):
 1. $M' := \text{Saturate}(M)$
 2. Declare unsat iff $\exists i: M'[i][i] < 0$
- Join(M_1, M_2):
 1. $M'_1 := \text{Saturate}(M_1); M'_2 := \text{Saturate}(M_2);$
 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

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

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

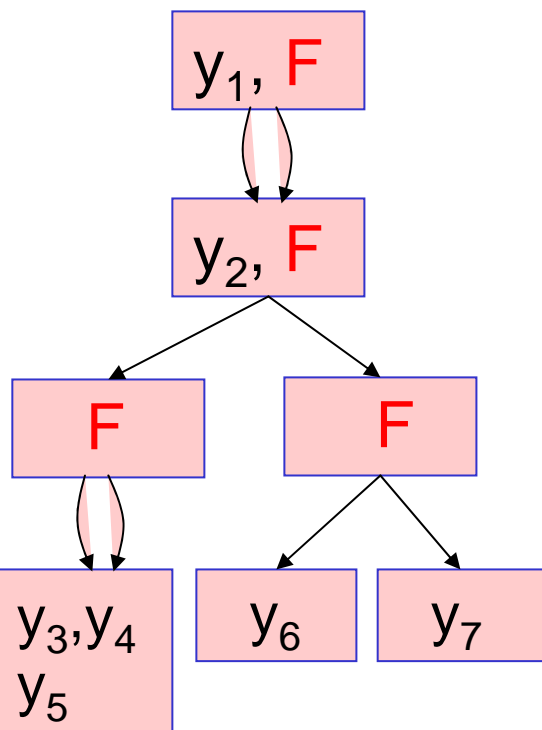
- $\text{Join}(G_1, G_2)$:
 1. $G'_1 := \text{Saturate}(G_1)$; $G'_2 := \text{Saturate}(G_2)$;
 2. $G := \text{Intersect}(G'_1, G'_2)$;
 3. return G ;

For each node $n = \langle U, \{n_i, n'_i\} \rangle$ in G'_1

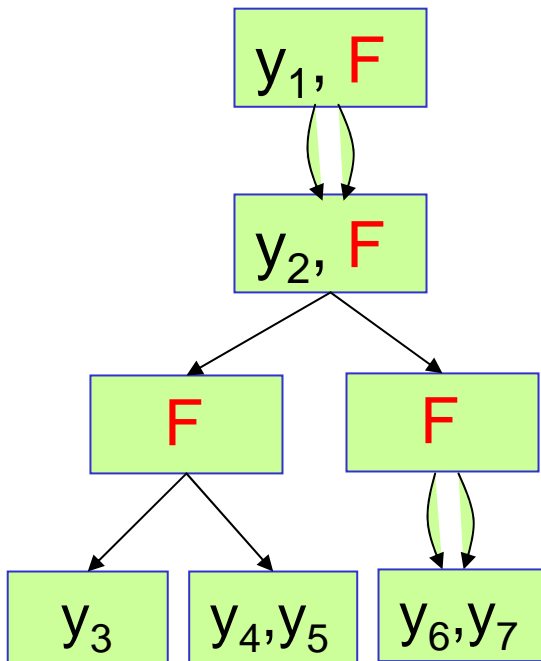
and node $m = \langle V, \{m_j, m'_j\} \rangle$ in G'_2 ,

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

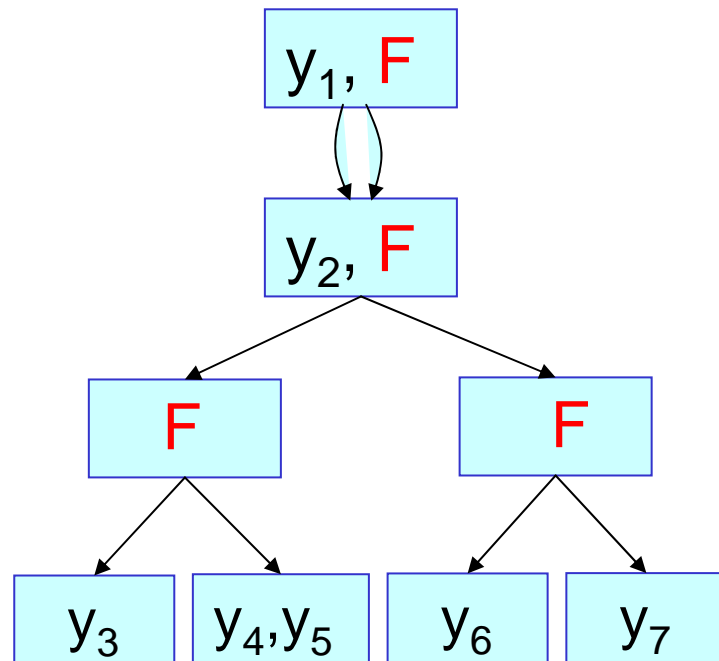
Uninterpreted Functions: Examples of Join



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

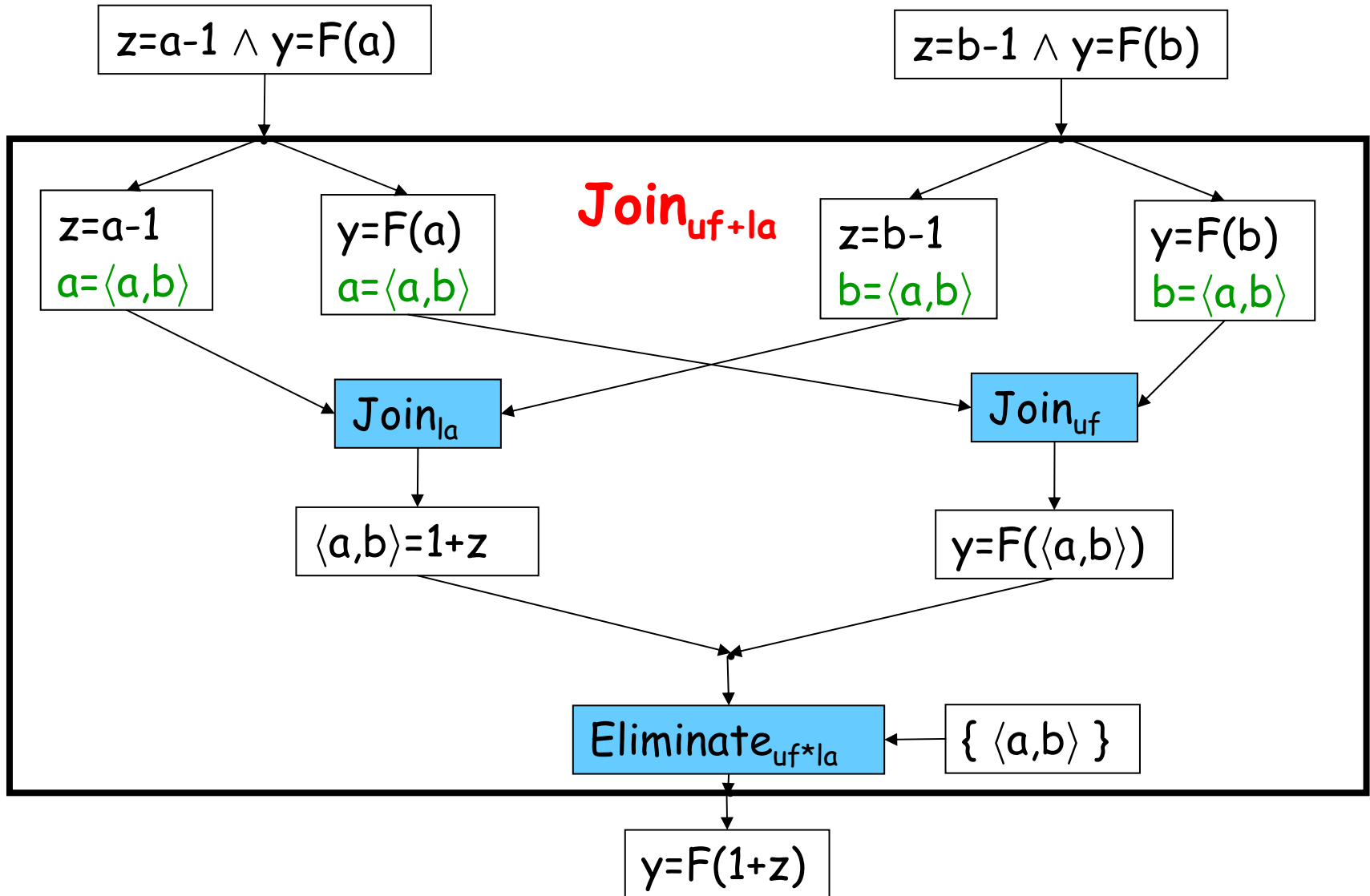
Decision Procedure

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

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. $L'_1 := L_1 \wedge D_L$; $R'_1 := R_1 \wedge D_L$;
 $L'_2 := L_2 \wedge D_R$; $R'_2 := R_2 \wedge D_R$;
 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} ;

Join Algorithm: Example

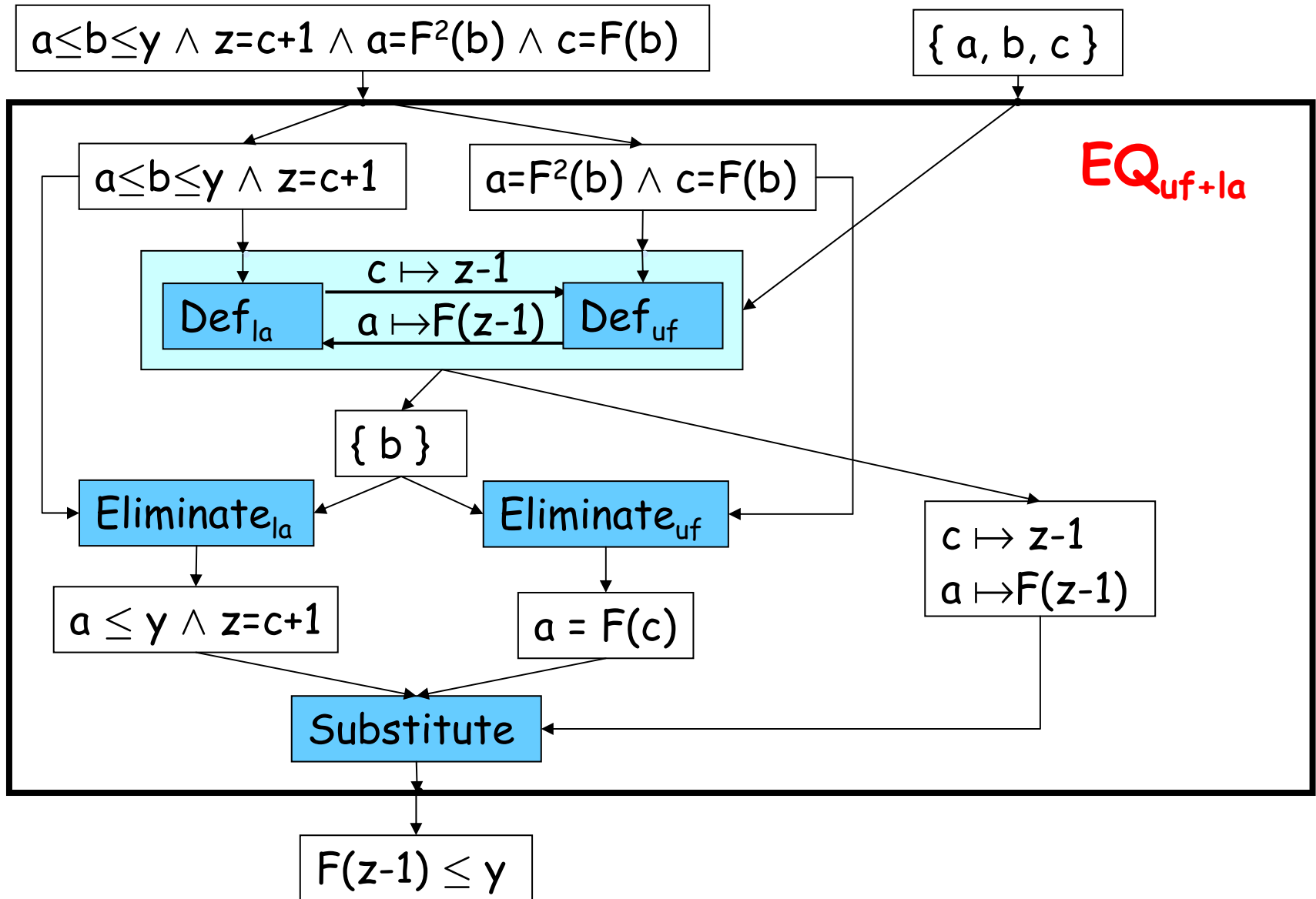


Existential Quantifier Elimination Algorithm

- Eliminate_{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.

Elimination Algorithm: Example



References

- Uninterpreted Functions
 - "A polynomial time algorithm for global value numbering", SAS 2004
 - "Join algorithms for the theory of uninterpreted functions", FSTTCS 2004
- Combination of Linear Arithmetic and Uninterpreted Functions
 - "Combining Abstract Interpreters", PLDI 2006