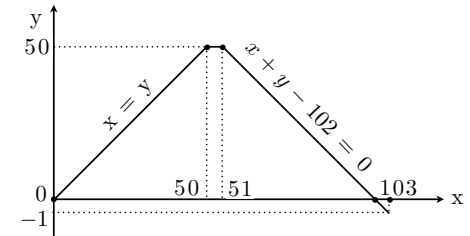


# A Binary Decision Tree Abstract Domain Functor

Junjie Chen and Patrick Cousot  
New York University

## A Motivating Example

```
x = 0; y = 0;
while(y >= 0) {
  if (x <= 50) y++;
  else y--;
  x++;
}
```



Intervals:  $x \geq 0 \wedge y \geq -1$

Convex Polyhedra:  $y \geq -1 \wedge x - y \geq 0 \wedge x + 52y \geq 0$

## Precision Problem

- A common believe (in data flow analysis) is that the problem is from the **imprecise joins**  $\sqcup$  ?
- **No**, e.g. in the Galois connection case, the abstraction of  $\cup$  is exact ( $\alpha$  preserves joins)
- The problem is from the **imprecise abstraction**:
  - convex abstractions do not take the **control flow** into account precisely enough

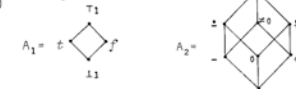
## Idea:

- The **reduced cardinal power**  $A_2^{A_1} = A_1 \rightarrow A_2$  [CC79]

THEOREM 10.2.0.1

The *reduced cardinal power* with base  $(A_2, t_2, \gamma_2)$  and exponent  $(A_1, t_1, \gamma_1)$  is  $(A, t, \gamma)$  where  $A = \sigma\{iso(A_1 \rightarrow A_2)\}$ ,  $\sigma_c\{iso(A_1 \rightarrow A_2) \rightarrow iso(A_1 \rightarrow A_2)\}$  is  $\lambda g. \prod\{f \in iso(A_1 \rightarrow A_2) : \gamma(f) = \gamma(g)\}$ ,  $\gamma_c\{iso(A_1 \rightarrow A_2) \rightarrow A\}$  is  $\lambda g. [\lambda X. \wedge\{Y_1(v)(X) \Rightarrow \gamma_2(g(v))(X) : v \in A_1\}]$ ,  $t = \lambda S. (\alpha \circ \tau(S) \circ \gamma)$  and  $\alpha \in (A \rightarrow iso(A_1 \rightarrow A_2))$  is  $\lambda P. [\sigma\{\lambda v. [\alpha_2(P \wedge \gamma_1(v))]\}]$ .  $A \triangleright \alpha, \gamma \triangleright A$  and  $\forall S \in L, t(S) \in \lambda g. [\lambda v. \sqcup_2\{t_2(S)(g(z)) : z \in A_1 \wedge t_1(S)(z) \in v\}]$  (with  $\sqcup_2^\emptyset = \perp_2$ ).

Example 10.2.0.2



$g \in (A_1 \rightarrow A_2)$  then  $\gamma(g) = (\gamma_1(t) \wedge \gamma_2(g(t))) \vee (\gamma_1(f) \wedge \gamma_2(g(f)))$

- with exponent  $A_1$  which is an abstraction of the control flow graph

# Operational trace Semantics

## Syntax

- Consider the following **abstract syntax** of command:

$$\begin{aligned} C \in \mathbb{C} ::= & \text{skip} \mid x = E \mid C_1 ; C_2 \mid \\ & \text{if } (B) \{C_1\} \text{ else } \{C_2\} \mid \\ & \text{while } (B) \{C\} \end{aligned}$$

- Actions** describe elementary **indivisible** program computation steps:

$$A \in \mathbb{A} ::= \text{skip} \mid x = E \mid B \mid \neg B$$

## States

- States** record the current **values** of variables in the **environment/memory** as well as a **label/control point** specifying what remains to be **executed**.

$$L \in \mathbb{L} ::= C \mid \text{stop} \quad \text{labels}$$

$$v \in \mathcal{V} \quad \text{values}$$

$$\rho \in \mathcal{E} \triangleq \mathbb{X} \rightarrow \mathcal{V} \quad \text{environments}$$

$$\sigma \in \Sigma \triangleq \mathbb{L} \times \mathcal{E} \quad \text{states}$$

## Traces

- Trace

$$\pi = \sigma_0 \xrightarrow{A_0} \sigma_1 \xrightarrow{A_1} \dots \xrightarrow{A_{n-2}} \sigma_{n-1}$$

- sequence  $\underline{\pi} = \sigma_0 \sigma_1 \dots \sigma_{n-1} \in \Sigma^n$  of **states**
- sequence  $\bar{\pi} = A_0 A_1 \dots A_{n-2} \in \mathbb{A}^{n-1}$  of **actions**.

# Trace Semantics

The **trace semantics** describes all possible **observations** of **executions** of the command  $C$ .

$$\mathcal{S}^t \llbracket x = E \rrbracket \triangleq \{ \langle x = E, \rho \rangle \xrightarrow{x=E} \langle \text{stop}, \rho[x := v] \rangle \mid \rho \in \mathcal{E} \wedge v \in \mathcal{E} \llbracket E \rrbracket \rho \}$$

$$\mathcal{S}^t \llbracket C_1; C_2 \rrbracket \triangleq \{ \langle \pi; C_2 \rangle \xrightarrow{A} \langle C_2, \rho \rangle \xrightarrow{A'} \pi' \mid \rho \in \mathcal{E} \wedge \pi \xrightarrow{A} \langle \text{stop}, \rho \rangle \in \mathcal{S}^t \llbracket C_1 \rrbracket \wedge \langle C_2, \rho \rangle \xrightarrow{A'} \pi' \in \mathcal{S}^t \llbracket C_2 \rrbracket \}$$

...

$$\begin{aligned} \mathcal{F}^{ti} \llbracket \text{while} (B) \{C\} \rrbracket X &\triangleq \{ \langle \text{while} (B) \{C\}, \rho \rangle \mid \rho \in \mathcal{E} \} \cup \\ &\{ \pi \xrightarrow{A} \langle \text{while} (B) \{C\}, \rho \rangle \xrightarrow{B} \langle C, \rho \rangle \xrightarrow{A'} \pi' \xrightarrow{A''} \langle \text{stop}, \rho' \rangle \}; \text{while} (B) \{C\} \mid \\ &\pi, \pi' \in \Pi^* \wedge \pi \xrightarrow{A} \langle \text{while} (B) \{C\}, \rho \rangle \in X \wedge \text{true} \in \mathcal{E} \llbracket B \rrbracket \rho \wedge \\ &\langle C, \rho \rangle \xrightarrow{A'} \pi' \xrightarrow{A''} \langle \text{stop}, \rho' \rangle \in \mathcal{S}^t \llbracket C \rrbracket \} \end{aligned}$$

$$\mathcal{S}^{ti} \llbracket \text{while} (B) \{C\} \rrbracket \triangleq \text{lfp}^{\subseteq} \mathcal{F}^{ti} \llbracket \text{while} (B) \{C\} \rrbracket$$

$$\mathcal{S}^b \llbracket \text{while} (B) \{C\} \rrbracket \triangleq \{ \pi \xrightarrow{A} \langle \text{while} (B) \{C\}, \rho \rangle \xrightarrow{\neg B} \langle \text{stop}, \rho \rangle \mid \pi \in \Pi^* \wedge \pi \xrightarrow{A} \langle \text{while} (B) \{C\}, \rho \rangle \in \mathcal{S}^{ti} \llbracket \text{while} (B) \{C\} \rrbracket \wedge \text{false} \in \mathcal{E} \llbracket B \rrbracket \rho \}$$

# The Control Flow Graph Abstraction

# Action Path Abstraction

Let  $\alpha^a(\pi) \triangleq \bar{\pi}$  collects the sequence of actions  $A_0 A_1 \dots A_{n-2}$ , then

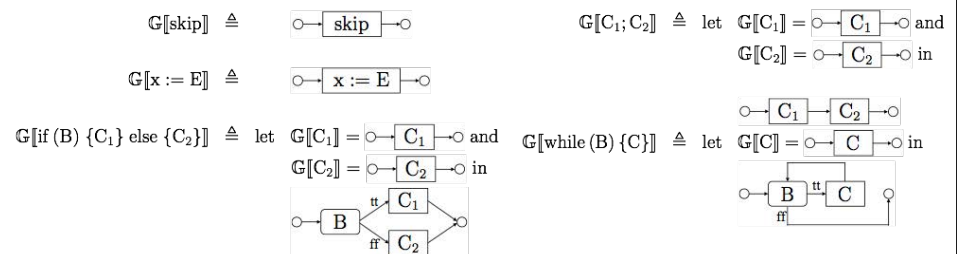
**Definition 1** (Action path abstraction). Given a set of traces  $\mathcal{S}$ ,

$$\begin{aligned} \alpha^a &\in \wp(\Pi) \rightarrow \wp(\mathbb{A}^*) \\ \alpha^a(\mathcal{S}) &\triangleq \{ \alpha^a(\pi) \mid \pi \in \mathcal{S} \} \end{aligned}$$

collects the sequences of actions executed along the traces of  $\mathcal{S}$ .

# Control Flow Graph

- A **control flow graph**  $(V, E)$  of a program is, as usual, a directed graph:
  - **nodes** are actions in the program
  - **edges** represent the possible flow of control.
- The CFG can be build by the structural (fixpoint) induction on the syntax of the command  $C$ :



# Action Path Semantics of CFG

$$\mathcal{G}^a \llbracket \circ \rightarrow \text{skip} \rightarrow \circ \rrbracket \triangleq \{\text{skip}\} \quad \mathcal{G}^a \llbracket \circ \rightarrow x := E \rightarrow \circ \rrbracket \triangleq \{x = E\}$$

$$\mathcal{G}^a \llbracket \circ \rightarrow B \begin{array}{l} \text{tt} \rightarrow C_1 \\ \text{ff} \rightarrow C_2 \end{array} \rightarrow \circ \rrbracket \triangleq \{B\} \cdot \mathcal{G}^a \llbracket \circ \rightarrow C_1 \rightarrow \circ \rrbracket \cup \{\neg B\} \cdot \mathcal{G}^a \llbracket \circ \rightarrow C_2 \rightarrow \circ \rrbracket$$

$$\mathcal{G}^a \llbracket \circ \rightarrow C_1 \rightarrow C_2 \rightarrow \circ \rrbracket \triangleq \mathcal{G}^a \llbracket \circ \rightarrow C_1 \rightarrow \circ \rrbracket \cdot \mathcal{G}^a \llbracket \circ \rightarrow C_2 \rightarrow \circ \rrbracket$$

$$\mathcal{F}^a \llbracket \circ \rightarrow B \begin{array}{l} \text{tt} \rightarrow C \\ \text{ff} \rightarrow \circ \end{array} \rrbracket X \triangleq \{\varepsilon\} \cup X \cdot \{B\} \cdot \mathcal{G}^a \llbracket \circ \rightarrow C \rightarrow \circ \rrbracket$$

$$\mathcal{G}^a \llbracket \circ \rightarrow B \begin{array}{l} \text{tt} \rightarrow C \\ \text{ff} \rightarrow \circ \end{array} \rrbracket \triangleq \text{lfp} \subseteq \mathcal{F}^a \llbracket \circ \rightarrow B \begin{array}{l} \text{tt} \rightarrow C \\ \text{ff} \rightarrow \circ \end{array} \rrbracket \cdot \{\neg B\}$$

$$= (\{B\} \cdot \mathcal{G}^{a*} \llbracket \circ \rightarrow C \rightarrow \circ \rrbracket)^* \cdot \{\neg B\}$$

The CFG is an abstraction of the trace semantics

- Soundness:

$$\alpha^a(\mathcal{S}^t \llbracket C \rrbracket) \subseteq \mathcal{G}^a \llbracket G \llbracket C \rrbracket \rrbracket$$

- The basis for most static analyses

# The Branch Condition Graph (abstracting the control flow graph)

# Condition Path Abstraction

- Let

$$\alpha^c(\text{skip}) \triangleq \varepsilon \quad \alpha^c(B) \triangleq B$$

$$\alpha^c(x = E) \triangleq \varepsilon \quad \alpha^c(\neg B) \triangleq \neg B$$

$$\alpha^c(\pi_1 \cdot \pi_2) \triangleq \alpha^c(\pi_1) \cdot \alpha^c(\pi_2)$$

- The condition path abstraction collects the sequences of **conditions** in the action paths  $\mathcal{A}$ :

$$\alpha^c \in \wp(\mathbb{A}^*) \mapsto \wp((\mathbb{A}^c)^*)$$

$$\alpha^c(\mathcal{A}) \triangleq \{\alpha^c(\bar{\pi}) \mid \bar{\pi} \in \mathcal{A}\}$$

# Loop Condition Elimination

- Let

$\mathbb{A}^B$ : the set of branch conditions

$\mathbb{A}^L$ : the set of loop conditions

- and

$$\alpha^d(A^b) \triangleq A^b, \quad \alpha^d(A^l) \triangleq \varepsilon$$

$$\alpha^d(\pi_{c_1} \cdot \pi_{c_2}) \triangleq \alpha^d(\pi_{c_1}) \cdot \alpha^d(\pi_{c_2})$$

- where  $A^b \in \mathbb{A}^B$  and  $A^l \in \mathbb{A}^L$ .
- The loop condition elimination collects the sequences of **branch conditions** from the condition paths :

$$\alpha^d \in \wp((\mathbb{A}^C)^*) \mapsto \wp((\mathbb{A}^B)^*)$$

$$\alpha^d(\mathcal{C}) \triangleq \{\alpha^d(\bar{\pi}_c) \mid \bar{\pi}_c \in \mathcal{C}\}$$

# Duplication Elimination

$$\text{erase}(d_1 d_2 d_3 \dots d_n, d) \triangleq \begin{cases} \text{if } d_1 == d \text{ then } \text{erase}(d_2 d_3 \dots d_n, d) \\ \text{else } d_1 \cdot \text{erase}(d_2 d_3 \dots d_n, d) \end{cases}$$

$$\text{fold}(d_1 d_2 \dots d_n) \triangleq \begin{cases} \text{if } d_1 d_2 \dots d_n == \varepsilon \text{ then } \varepsilon \\ \text{else } \text{fold}(\text{erase}(d_1 d_2 \dots d_{n-1}, d_n)) \cdot d_n \end{cases}$$

# Branch Condition Path Abstraction

- Let

$$\alpha^\ell(\bar{\pi}_d) = \text{fold}(\bar{\pi}_d)$$

eliminate duplications of each branch condition while keeping its last occurrence in  $\bar{\pi}_d$ ,

- The branch condition path abstraction collects **branch condition paths (sequences of branch conditions without duplications)**:

$$\alpha^\ell \in \wp((\mathbb{A}^B)^*) \mapsto \wp((\mathbb{A}^B)^* \setminus \mathbb{D})$$

$$\alpha^\ell(\mathcal{D}) \triangleq \{\alpha^\ell(\bar{\pi}_d) \mid \bar{\pi}_d \in \mathcal{D}\}$$

# Concretizations

- Action path abstraction

$$\gamma^a(\mathcal{A}) \triangleq \{\pi \mid \alpha^a(\pi) \in \mathcal{A}\}$$

- Condition path abstraction

$$\gamma^c(\mathcal{C}) \triangleq \{\bar{\pi} \mid \alpha^c(\bar{\pi}) \in \mathcal{C}\}$$

- Loop condition elimination

$$\gamma^d(\mathcal{D}) \triangleq \{\bar{\pi}_c \mid \alpha^d(\bar{\pi}_c) \in \mathcal{D}\}$$

- Branch condition path abstraction

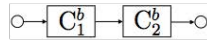
$$\gamma^\ell(\mathcal{B}) \triangleq \{\bar{\pi}_d \mid \alpha^\ell(\bar{\pi}_d) \in \mathcal{B}\}$$

# Branch Condition Graph

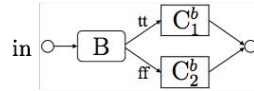
- A branch condition graph (BCG) of a program is a **directed acyclic graph**, in which each node corresponds to a **branch condition** occurring in the program and has **two outgoing edges** representing its **true** and **false branches**.

$$\mathcal{G}^b[\text{skip}] \triangleq \circ \longrightarrow \circ \quad \mathcal{G}^b[x := E] \triangleq \circ \longrightarrow \circ$$

$$\mathcal{G}^b[C_1; C_2] \triangleq \text{let } \mathcal{G}^b[C_1] = \circ \longrightarrow C_1^b \longrightarrow \circ \text{ and } \mathcal{G}^b[C_2] = \circ \longrightarrow C_2^b \longrightarrow \circ \text{ in}$$



$$\mathcal{G}^b[\text{if } (B) \{C_1\} \text{ else } \{C_2\}] \triangleq \text{let } \mathcal{G}^b[C_1] = \circ \longrightarrow C_1^b \longrightarrow \circ \text{ and } \mathcal{G}^b[C_2] = \circ \longrightarrow C_2^b \longrightarrow \circ$$



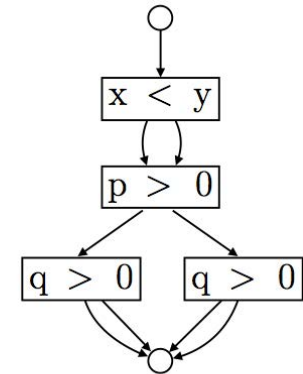
$$\mathcal{G}^b[\text{while } (B) \{C\}] \triangleq \text{let } \mathcal{G}^b[C] = \circ \longrightarrow C^b \longrightarrow \circ \text{ in } \circ \longrightarrow C^b \longrightarrow \circ$$

## Abstraction of Control Flow Graph!

# Example

```

while(i <= m) {
  if(x < y) x++;
  else y++;
  if(p > 0)
    if(q > 0) r = p + q;
    else r = p - q;
  else
    if(q > 0) r = q - p;
    else r = -(p + q);
  i++;
}
    
```



$$\begin{aligned}
 & (x < y) \cdot (p > 0) \cdot (q > 0), & \neg(x < y) \cdot (p > 0) \cdot (q > 0), \\
 & (x < y) \cdot (p > 0) \cdot \neg(q > 0), & \neg(x < y) \cdot (p > 0) \cdot \neg(q > 0), \\
 & (x < y) \cdot \neg(p > 0) \cdot (q > 0), & \neg(x < y) \cdot \neg(p > 0) \cdot (q > 0), \\
 & (x < y) \cdot \neg(p > 0) \cdot \neg(q > 0), & \neg(x < y) \cdot \neg(p > 0) \cdot \neg(q > 0).
 \end{aligned}$$

# Trace Semantics Partitioning

Given the trace semantics  $\mathcal{S}^t[P]$  of a program P



$$\alpha^{\ell} \circ \alpha^d \circ \alpha^c \circ \alpha^a(\mathcal{S}^t[P]) = \mathcal{B} \text{ where } |\mathcal{B}| = N$$



For each  $\pi_b \in \mathcal{B}$  and all pairs  $(\pi_{b_1}, \pi_{b_2}) \in \mathcal{B} \times \mathcal{B}$ , we have

- $\gamma^a \circ \gamma^c \circ \gamma^d \circ \gamma^{\ell}(\pi_b) \cap \mathcal{S}^t[P] \subseteq \mathcal{S}^t[P]$
- $\bigcup_{i \leq N} (\gamma^a \circ \gamma^b(\pi_{b_i}) \cap \mathcal{S}^t[P]) = \mathcal{S}^t[P]$
- $(\gamma^a \circ \gamma^b(\pi_{b_1}) \cap \mathcal{S}^t[P]) \cap (\gamma^a \circ \gamma^b(\pi_{b_2}) \cap \mathcal{S}^t[P]) = \emptyset$



$\mathcal{B}$  defines a partitioning on the trace semantics  $\mathcal{S}^t[P]$

# The Binary Decision Tree Abstraction

# Binary Decision Tree

**Definition 2.** A binary decision tree  $t \in \mathbb{T}(\mathcal{B}, \mathbb{D}_\ell)$  over the set  $\mathcal{B}$  of branch condition paths (with concretization  $\gamma^a \circ \gamma^c \circ \gamma^d \circ \gamma^\ell$ ) and the leaf abstract domain  $\mathbb{D}_\ell$  (with concretization  $\gamma_\ell$ ) is either  $\langle p \rangle$  with  $p$  is an element of  $\mathbb{D}_\ell$  and  $\mathcal{B}$  is empty or  $\llbracket B : t_t, t_f \rrbracket$  where  $B$  is the first element of all branch condition paths  $\pi_b \in \mathcal{B}$  and  $(t_t, t_f)$  are the left and right subtree of  $t$  represent its true and false branch such that  $t_t, t_f \in \mathbb{T}(\mathcal{B}_{\setminus \beta}, \mathbb{D}_\ell)$  ( $\beta \triangleq B$  or  $\neg B$  and  $\mathcal{B}_{\setminus \beta}$  denotes the removal of  $\beta$  and all branch conditions appearing before in each branch condition path in  $\mathcal{B}$ ).

$$\llbracket B_1 : \llbracket B_2 : \langle p_1 \rangle, \langle p_2 \rangle \rrbracket, \llbracket B_3 : \langle p_3 \rangle, \langle p_4 \rangle \rrbracket \rrbracket$$

# Concretization

**Definition 3.** Let  $\rho$  be the concrete environment assigning concrete values  $\rho(x)$  to variables  $x$  and  $\llbracket e \rrbracket \rho$  for the concrete value of the expression  $e$  in the concrete environment  $\rho$ , the concretization of a binary decision tree  $\gamma_t$  is either

$$\gamma_t(\langle p \rangle) \triangleq \gamma_\ell(p)$$

when the binary decision tree can be reduced to a leaf or

$$\gamma_t(\llbracket B : t_t, t_f \rrbracket) \triangleq \{ \rho \mid \llbracket B \rrbracket \rho = \text{true} \implies \rho \in \gamma_t(t_t) \wedge \llbracket B \rrbracket \rho = \text{false} \implies \rho \in \gamma_t(t_f) \}$$

when the binary decision tree is rooted at a decision node.

# Binary Decision Tree Abstract Domain Functor

**Definition 4.** A binary decision tree abstract domain functor is a tuple

$$\langle \mathbb{T}(\mathcal{B}, \mathbb{D}_\ell) \setminus_{\equiv_t}, \sqsubseteq_t, \perp_t, \top_t, \sqcup_t, \sqcap_t, \nabla_t, \Delta_t \rangle$$

on two parameters, a set  $\mathcal{B}$  of branch condition paths and a leaf abstract domain  $\mathbb{D}_\ell$  where

$$\begin{aligned} P, Q, \dots &\in \mathbb{T}(\mathcal{B}, \mathbb{D}_\ell) \setminus_{\equiv_t} \\ \sqsubseteq_t &\in \mathbb{T} \times \mathbb{T} \rightarrow \{\text{false}, \text{true}\} \\ \perp_t, \top_t &\in \mathbb{T}(\mathcal{B}, \mathbb{D}_\ell) \end{aligned}$$

abstract properties  
abstract partial order  
infimum, supremum  
( $\forall P \in \mathbb{T} : \perp_t \sqsubseteq_t P \sqsubseteq_t \top_t$ )

$$\begin{aligned} \sqcup_t, \sqcap_t &\in \mathbb{T} \times \mathbb{T} \rightarrow \mathbb{T} \\ \nabla_t, \Delta_t &\in \mathbb{T} \times \mathbb{T} \rightarrow \mathbb{T} \end{aligned}$$

abstract join, meet  
abstract widening, narrowing

# Binary Decision Tree Abstract Domain Functor

- The set  $\mathcal{B}$  of branch condition paths is built by the **syntactic analysis** from the **control flow** of the program. Hence the structure of the binary decision tree is **finite** and **does not change** in the data flow analysis.
- The leaf abstract domain  $\mathbb{D}_\ell$  for the leaves could be any numerical or symbolic **algebraic** abstract domains such as polyhedra, ...

# Binary Decision Tree Abstract Domain Functor

- The set  $\mathcal{B}$  of branch condition paths is built by the **syntactic analysis** from the **control flow** of the program. Hence the structure of the binary decision tree is **finite** and **does not change** in the data flow analysis.
- The leaf abstract domain  $\mathbb{D}_\ell$  for the leaves could be any numerical or symbolic **algebraic** abstract domains such as polyhedra, ...

→ very different from other proposals where the shape of the decision tree evolves during the analysis e.g. (among many others)

Patrick Cousot, Radhia Cousot, Laurent Mauborgne:  
A Scalable Segmented Decision Tree Abstract Domain. Essays in Memory of Amir Pnueli, LNCS 6200, 2010: 72-95.

# Inclusion and Equality Test

Given two binary decision tree  $t_1, t_2 \in \mathbb{T}(\mathcal{B}, \mathbb{D}_\ell) \setminus \{\perp_t, \top_t\}$ ,

- Inclusion test: comparing each pair  $(\ell_1, \ell_2)$  of leaves in  $(t_1, t_2)$  where  $\ell_1$  and  $\ell_2$  are defined by the same branch condition path  $\pi_b \in \mathcal{B}$ .

- $t_1 \sqsubseteq_t t_2$  if  $\ell_1 \sqsubseteq_\ell \ell_2$  for all pairs of  $(\ell_1, \ell_2)$ ,
- $t_1 \not\sqsubseteq_t t_2$  otherwise.

- Equality test:  $t_1 =_t t_2 \triangleq t_1 \sqsubseteq_t t_2 \wedge t_2 \sqsubseteq_t t_1$ .

- If the leaf abstract domain  $\mathbb{D}_\ell$  has  $=_\ell$ , we may use it directly.

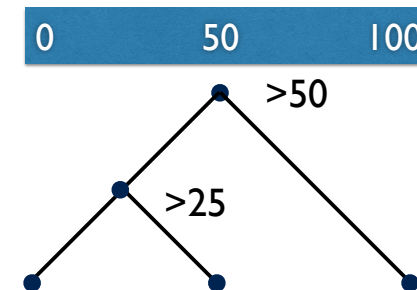
# Meet and Join

Given two binary decision tree  $t_1, t_2 \in \mathbb{T}(\mathcal{B}, \mathbb{D}_\ell) \setminus \{\perp_t, \top_t\}$ ,

- Meet: for each pair  $(\ell_1, \ell_2)$  of leaves in  $(t_1, t_2)$  where  $\ell_1$  and  $\ell_2$  are defined by the same branch condition path  $\pi_b \in \mathcal{B}$ .
  - $\ell = \ell_1 \sqcap_\ell \ell_2$  using the meet  $\sqcap_\ell$  in the leaf abstract domain  $\mathbb{D}_\ell$ .
- Join: for each pair  $(\ell_1, \ell_2)$  of leaves in  $(t_1, t_2)$  where  $\ell_1$  and  $\ell_2$  are defined by the same branch condition path  $\pi_b \in \mathcal{B}$ .
  - $\ell = (\ell_1 \sqcup_\ell \ell_2) \sqcap_\ell \mathbb{D}_\ell(\beta_1) \sqcap_\ell \mathbb{D}_\ell(\beta_2) \sqcap_\ell \dots \sqcap_\ell \mathbb{D}_\ell(\beta_n)$  where  $\pi_b = \beta_1 \cdot \beta_2 \cdot \dots \cdot \beta_n$  and  $\mathbb{D}_\ell(\beta)$  is the representation of  $\beta$  in  $\mathbb{D}_\ell$  (when  $\alpha_\ell$  exists in the leaf abstract domain  $\mathbb{D}_\ell$ , we can use  $\alpha_\ell(\beta)$  instead).

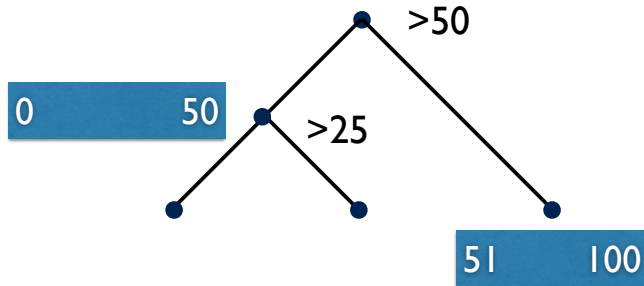
↑ pairwise join and distribute over leaves

# Distribution over leaves

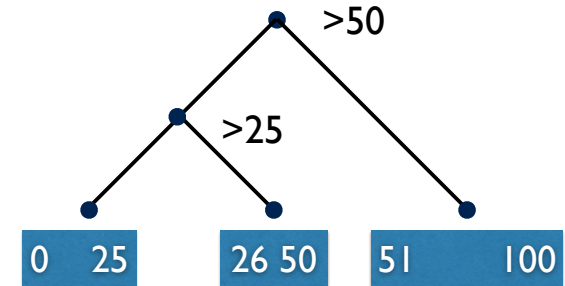




# Distribution over leaves



# Distribution over leaves



# Widening and Narrowing

Given two binary decision tree  $t_1, t_2 \in \mathbb{T}(\mathcal{B}, \mathbb{D}_\ell) \setminus \{\perp_t, \top_t\}$  and  $t_1 \sqsubseteq_t t_2$ ,

- Widening  $t_1 \nabla_\ell t_2$ : for each pair  $(\ell_1, \ell_2)$  of leaves in  $(t_1, t_2)$  where  $\ell_1$  and  $\ell_2$  are defined by the same branch condition path  $\underline{\pi}_b \in \mathcal{B}$ .
  - $\ell = (\ell_1 \nabla_\ell \ell_2) \sqcap_\ell \mathbb{D}_\ell(\beta_1) \sqcap_\ell \mathbb{D}_\ell(\beta_2) \sqcap_\ell \dots \sqcap_\ell \mathbb{D}_\ell(\beta_n)$  where  $\nabla_\ell$  is the widening in the leaf abstract domain  $\mathbb{D}_\ell$ ,  $\underline{\pi}_b = \beta_1 \cdot \beta_2 \cdot \dots \cdot \beta_n$  and  $\mathbb{D}_\ell(\beta)$  is the representation of  $\beta$  in  $\mathbb{D}_\ell$ .

↑ pairwise widen and distribute over leaves

- Narrowing  $t_2 \Delta_\ell t_1$ : for each pair  $(\ell_1, \ell_2)$  of leaves in  $(t_1, t_2)$  where  $\ell_1$  and  $\ell_2$  are defined by the same branch condition path  $\underline{\pi}_b \in \mathcal{B}$ .
  - $\ell = \ell_2 \Delta_\ell \ell_1$  using the narrowing  $\Delta_\ell$  in the leaf abstract domain  $\mathbb{D}_\ell$ .

# Reduction of Binary Decision Tree by an Abstract Property

Given a binary decision tree  $t \in \mathbb{T}(\mathcal{B}, \mathbb{D}_\ell)$  and an abstract property  $p$ , we define  $t \sqcap_t p$  as:

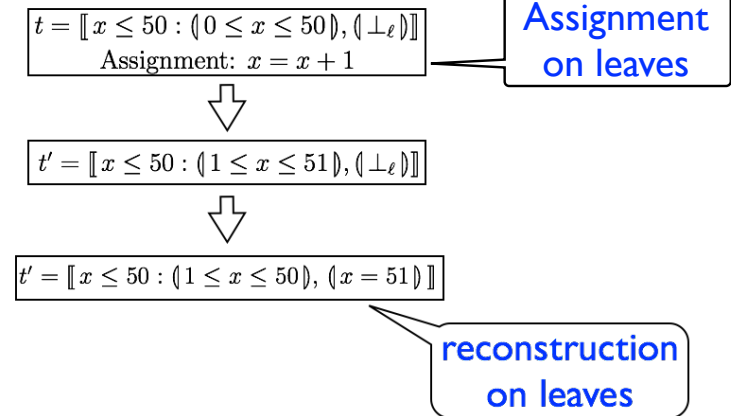
$$\begin{aligned}
 \perp_t \sqcap_t p &\triangleq \perp_t \\
 \top_t \sqcap_t p &\triangleq (p) \\
 t \sqcap_t \text{false} &\triangleq \perp_t \\
 t \sqcap_t \text{true} &\triangleq t \\
 (p') \sqcap_t p &\triangleq (p' \sqcap_\ell \mathbb{D}_\ell(p)) \\
 \llbracket B : t_l, t_r \rrbracket \sqcap_t p &\triangleq \llbracket B : t_l \sqcap_t \mathbb{D}_\ell(B) \sqcap_t \mathbb{D}_\ell(p), t_r \sqcap_t \mathbb{D}_\ell(\neg B) \sqcap_t \mathbb{D}_\ell(p) \rrbracket
 \end{aligned}$$

# Test Transfer Function

$$f_T[[B]]t \triangleq t \sqcap_t B$$

# Assignment Transfer Function

Given a binary decision tree  $t \in \mathbb{T}(\mathcal{B}, \mathbb{D}_\ell)$ , the assignment  $x = E$  can be performed at each leaf in  $t$  by using the assignment transfer function of  $\mathbb{D}_\ell$ .



# Reconstruction on Leaves

1. Collecting all leaf properties in  $t$ , let it be  $\{p_1, p_2, \dots, p_n\}$ ;
2. For each leaf in  $t$ , let  $\pi_b = \beta_1 \cdot \beta_2 \cdot \dots \cdot \beta_n$  be the branch condition path leading to it. We then calculate  $p'_i = p_i \sqcap_\ell (\mathbb{D}_\ell(\beta_1 \wedge \beta_2 \wedge \dots \wedge \beta_n))$ .
3. For each leaf in  $t$ , update it with  $p'_1 \sqcup_\ell p'_2 \sqcup_\ell \dots \sqcup_\ell p'_n$ .

↑ assign and redistribute over leaves

# Binary Decision Tree Construction

- In the pre-analysis
- On the fly during the analysis
  - Unification

# Tree Merging

1. Pick up a branch condition  $B$ .
2. Eliminate  $B$  ( $B$  or  $\neg B$ ) from each branch condition path in  $\mathcal{B}$ .
3. For each subtree of the form  $\llbracket B : t_t, t_f \rrbracket$ , if  $t_t$  and  $t_f$  have identical decision nodes, replace it by  $t_t \sqcup_t t_f$ .
4. Otherwise, there are decision nodes existing only in  $t_t$  or  $t_f$ . For each of those decision nodes, (recursively) eliminate it by merging its subtrees. When no such decision node exists, we get  $t'_t$  and  $t'_f$ , and they must have identical decision nodes, so  $\llbracket B : t_t, t_f \rrbracket$  can be replaced by  $t'_t \sqcup_t t'_f$ .

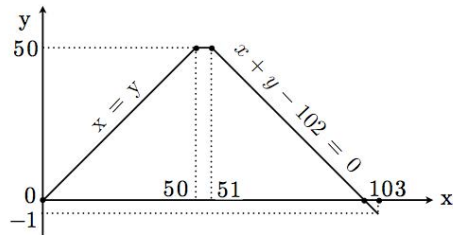
# A small example

# Example

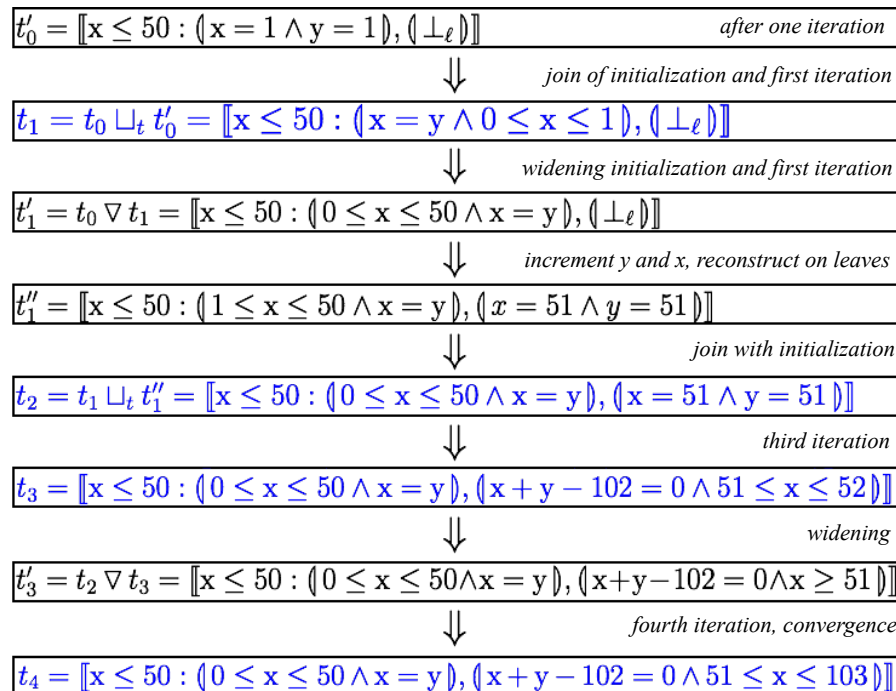
```

x = 0; y = 0;
while (y >= 0) {
  if (x <= 50) y++;
  else y--;
  x++;
}

```



- $y \geq -1 \wedge x - y \geq 0 \wedge x + 52y \geq 0$  by Apron
- We choose the polyhedra abstract domain as the leaf abstract domain
- We have  $\mathcal{B} = \{x \leq 50, \neg(x \leq 50)\}$
- Initially,  $t_0 = \llbracket x \leq 50 : (x = 0 \wedge y = 0), (\perp_\ell) \rrbracket$



# Conclusion

# Conclusion

- We need more precise abstractions than the Control Flow Graph (the usual starting point)
- Binary Decision Tree Abstraction:
  - Disjunctive refinement
  - Cost / precision ratio adjustable

# Thanks & Questions?