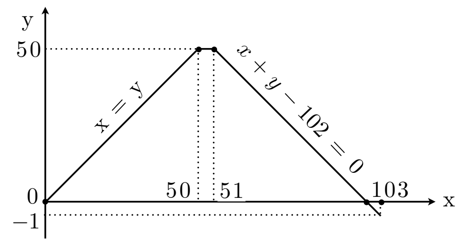
A Binary Decision Tree Abstract Domain Functor

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 (α preserves joins)
- The problem is from the imprecise abstraction:
  - convex abstractions do not take the control flow into account precisely enough

Idea:

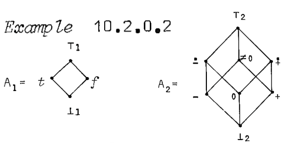
- The reduced cardinal power $A_2^{A_1} = A_1 \rightarrow A_2$ [CC79]

THEOREM 10.2.0.1

The *reduced cardinal power* with base (A_2, t_2, γ_2) and exponent (A_1, t_1, γ_1) is (A, t, γ) where $A = \sigma(iso(A_1 \to A_2))$, $\sigma \in (iso(A_1 \to A_2) \to iso(A_1 \to A_2))$ is $\lambda g.\sqcap\{f \in iso(A_1 \to A_2) : \gamma(f) = \gamma(g)\}$, $\gamma \in (iso(A_1 \to A_2) \to A)$ is $\lambda g.[\lambda X.\wedge\{\gamma_1(v)(X) \Rightarrow \gamma_2(g(v))(X) : v \in A_1\}]$, $t = \lambda S.(\alpha \circ \tau(S) \circ \gamma)$ and $\alpha \in (A \to iso(A_1 \to A_2))$ is $\lambda P.[\sigma(\lambda v.[\alpha_2(P \wedge \gamma_1(v))])]$.

$A \triangleright\!\!\!<\alpha,\gamma> A$ and $\forall S \in L$, $t(S) \sqsubseteq \lambda g.[\lambda v.\sqcup_2\{t_2(S)(g(z)) : z \in A_1 \wedge t_1(S)(z) \sqsubseteq v\}]$ (with $\sqcup_2 \emptyset = \bot_2$).

Example 10.2.0.2



$g \in (A_1 \to A_2)$ 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:

$$\begin{array}{lcll} \mathrm{C} & \in & \mathbb{C} & ::= \mathrm{skip} \mid \mathrm{x} = \mathrm{E} \mid \mathrm{C}_1 \, ; \mathrm{C}_2 \mid \\ & & & \quad \mathrm{if}\,(\mathrm{B})\ \{\mathrm{C}_1\}\ \mathrm{else}\ \{\mathrm{C}_2\} \mid \\ & & & \quad \mathrm{while}\,(\mathrm{B})\ \{\mathrm{C}\} \end{array}$$

- Actions describe elementary indivisible program computation steps:

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

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.

$$\begin{array}{lclll} \mathrm{L} & \in & \mathbb{L} & ::= \mathrm{C} \mid \mathrm{stop} & \text{labels} \\ v & \in & \mathcal{V} & & \text{values} \\ \rho & \in & \mathcal{E} & \triangleq \mathbb{X} \to \mathcal{V} & \text{environments} \\ \sigma & \in & \Sigma & \triangleq \mathbb{L} \times \mathcal{E} & \text{states} \end{array}$$

Traces

- Trace

$$\pi = \sigma_0 \xrightarrow{\mathrm{A}_0} \sigma_1 \xrightarrow{\mathrm{A}_1} \ldots \xrightarrow{\mathrm{A}_{n-2}} \sigma_{n-1}$$

- sequence $\underline{\pi} = \sigma_0\sigma_1\ldots\sigma_{n-1} \in \Sigma^n$ of states

  sequence $\overline{\pi} = \mathrm{A}_0\mathrm{A}_1\ldots\mathrm{A}_{n-2} \in \mathbb{A}^{n-1}$ of actions

Trace Semantics

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

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

$$\mathcal{S}^t[\![\mathrm{C}_1\,;\,\mathrm{C}_2]\!] \triangleq \{(\pi\,;C_2) \xrightarrow{\mathrm{A}} \langle \mathrm{C}_2, \rho\rangle \xrightarrow{\mathrm{A}'} \pi' \mid \rho \in \mathcal{E} \wedge \pi \xrightarrow{\mathrm{A}} \langle \mathrm{stop}, \rho\rangle \in \mathcal{S}^t[\![\mathrm{C}_1]\!] \wedge \langle \mathrm{C}_2, \rho\rangle \xrightarrow{\mathrm{A}'} \pi' \in \mathcal{S}^t[\![\mathrm{C}_2]\!]\}$$

...

$$\mathcal{F}^{ti}[\![\mathrm{while\,(B)\,\{C\}}]\!]X \triangleq \{\langle \mathrm{while\,(B)\,\{C\}}, \rho\rangle \mid \rho \in \mathcal{E}\} \cup \{\pi \xrightarrow{\mathrm{A}} \langle \mathrm{while\,(B)\,\{C\}}, \rho\rangle \xrightarrow{\mathrm{B}} (\langle \mathrm{C}, \rho\rangle \xrightarrow{\mathrm{A}'} \pi' \xrightarrow{\mathrm{A}''} \langle \mathrm{stop}, \rho'\rangle)\,;\,\mathrm{while\,(B)\,\{C\}} \mid \pi, \pi' \in \Pi^* \wedge \pi \xrightarrow{\mathrm{A}} \langle \mathrm{while\,(B)\,\{C\}}, \rho\rangle \in X \wedge \mathit{true} \in \mathcal{E}[\![\mathrm{B}]\!]\rho \wedge (\langle \mathrm{C}, \rho\rangle \xrightarrow{\mathrm{A}'} \pi' \xrightarrow{\mathrm{A}''} \langle \mathrm{stop}, \rho'\rangle) \in \mathcal{S}^t[\![\mathrm{C}]\!]\}$$

$$\mathcal{S}^{ti}[\![\mathrm{while\,(B)\,\{C\}}]\!] \triangleq \mathrm{lfp}^{\subseteq}\mathcal{F}^{ti}[\![\mathrm{while\,(B)\,\{C\}}]\!]$$

$$\mathcal{S}^t[\![\mathrm{while\,(B)\,\{C\}}]\!] \triangleq \{\pi \xrightarrow{\mathrm{A}} \langle \mathrm{while\,(B)\,\{C\}}, \rho\rangle \xrightarrow{\neg\mathrm{B}} \langle \mathrm{stop}, \rho\rangle \mid \pi \in \Pi^* \wedge \pi \xrightarrow{\mathrm{A}} \langle \mathrm{while\,(B)\,\{C\}}, \rho\rangle \in \mathcal{S}^{ti}[\![\mathrm{while\,(B)\,\{C\}}]\!] \wedge \mathit{false} \in \mathcal{E}[\![\mathrm{B}]\!]\rho\}$$

The Control Flow Graph Abstraction

Action Path Abstraction

Let $\alpha^a(\pi) \triangleq \overline{\pi}$ collects the sequence of actions $\mathrm{A}_0\mathrm{A}_1\ldots\mathrm{A}_{n-2}$, then

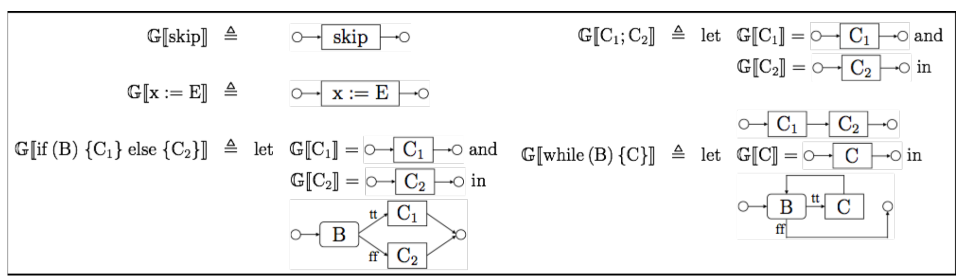
Definition 1 (Action path abstraction). Given a set of traces $\mathcal{S}$,

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

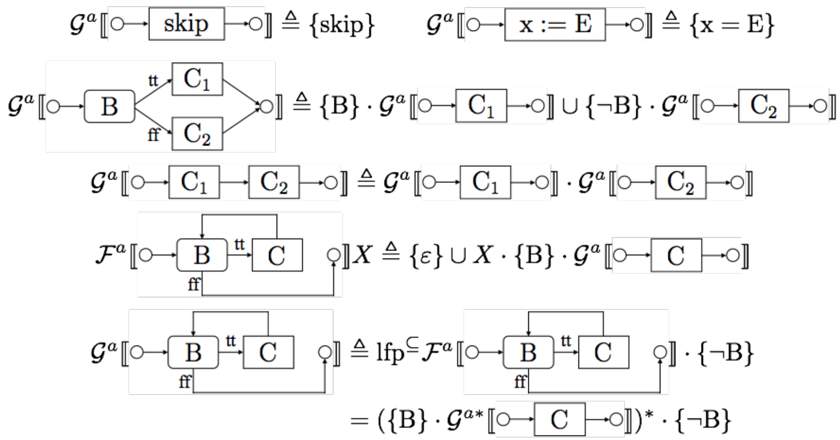
collects the sequences of actions executed along the traces of $\mathcal{S}$.

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



The CFG is an abstraction of the trace semantics

- Soundness:

$$\boxed{\alpha^a(\mathcal{S}^t[\![\mathrm{C}]\!]) \subseteq \mathcal{G}^a[\![\mathbb{G}[\![\mathrm{C}]\!]]\!]}$$

- The basis for most static analyses

The Branch Condition Graph (abstracting the control flow graph)

Condition Path Abstraction

- Let

$$\begin{array}{rcl} \alpha^c(\mathrm{skip}) & \triangleq & \varepsilon \\ \alpha^c(\mathrm{x} = \mathrm{E}) & \triangleq & \varepsilon \\ \alpha^c(\pi_1 \cdot \pi_2) & \triangleq & \alpha^c(\pi_1) \cdot \alpha^c(\pi_2) \end{array} \qquad \begin{array}{rcl} \alpha^c(\mathrm{B}) & \triangleq & \mathrm{B} \\ \alpha^c(\neg\mathrm{B}) & \triangleq & \neg\mathrm{B} \end{array}$$

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

$$\begin{array}{rcl} \alpha^c & \in & \wp(\mathbb{A}^*) \mapsto \wp((\mathbb{A}^C)^*) \\ \alpha^c(\mathcal{A}) & \triangleq & \{\alpha^c(\overline{\pi}) \mid \overline{\pi} \in \mathcal{A}\} \end{array}$$

Loop Condition Elimination

- Let

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

- and

$$\alpha^d(\mathrm{A}^b) \triangleq \mathrm{A}^b, \qquad \alpha^d(\mathrm{A}^l) \triangleq \varepsilon$$

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

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

$$\begin{array}{rcl} \alpha^d & \in & \wp((\mathbb{A}^C)^*) \mapsto \wp((\mathbb{A}^B)^*) \\ \alpha^d(\mathcal{C}) & \triangleq & \{\alpha^d(\overline{\pi}_c) \mid \overline{\pi}_c \in \mathcal{C}\} \end{array}$$

Duplication Elimination

$$\begin{array}{rcl} erase(d_1d_2d_3...d_n, d) & \triangleq & \text{if } \; d_1 == d \; \text{ then } \; erase(d_2d_3...d_n, d) \\ & & \text{else } \; d_1 \cdot erase(d_2d_3...d_n, d) \end{array}$$

$$\begin{array}{rcl} fold(d_1d_2...d_n) & \triangleq & \text{if } d_1d_2...d_n == \varepsilon \text{ then } \varepsilon \\ & & \text{else } fold(erase(d_1d_2...d_{n-1}, d_n)) \cdot d_n \end{array}$$

Branch Condition Path Abstraction

- Let

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

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

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

$$\begin{array}{rcl} \alpha^\ell & \in & \wp((\mathbb{A}^B)^*) \mapsto \wp((\mathbb{A}^B)^* \setminus \mathbb{D}) \\ \alpha^\ell(\mathcal{D}) & \triangleq & \{\alpha^\ell(\overline{\pi}_d) \mid \overline{\pi}_d \in \mathcal{D}\} \end{array}$$

Concretizations

- Action path abstraction

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

- Condition path abstraction

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

- Loop condition elimination

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

- Branch condition path abstraction

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

Branch Condition Graph

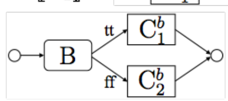
- A branch condition graph (BCG) of a program is a directed acyclic graph, in which each node corresponds to a branch condition occurring in the program and has two outgoing edges representing its true and false branches.

$$\mathbb{G}^b[\![\text{skip}]\!] \triangleq \circ\!\!\longrightarrow\!\!\circ \qquad \mathbb{G}^b[\![\text{x} := \text{E}]\!] \triangleq \circ\!\!\longrightarrow\!\!\circ$$

$$\mathbb{G}^b[\![\text{C}_1;\text{C}_2]\!] \triangleq \text{let } \mathbb{G}^b[\![\text{C}_1]\!] = \circ\!\!\longrightarrow\!\boxed{\text{C}_1^b}\!\longrightarrow\!\!\circ \text{ and } \mathbb{G}^b[\![\text{C}_2]\!] = \circ\!\!\longrightarrow\!\boxed{\text{C}_2^b}\!\longrightarrow\!\!\circ \text{ in}$$

$$\circ\!\!\longrightarrow\!\boxed{\text{C}_1^b}\!\longrightarrow\!\boxed{\text{C}_2^b}\!\longrightarrow\!\!\circ$$

$$\mathbb{G}^b[\![\text{if (B) } \{\text{C}_1\} \text{ else } \{\text{C}_2\}]\!] \triangleq \text{let } \mathbb{G}^b[\![\text{C}_1]\!] = \circ\!\!\longrightarrow\!\boxed{\text{C}_1^b}\!\longrightarrow\!\!\circ \text{ and } \mathbb{G}^b[\![\text{C}_2]\!] = \circ\!\!\longrightarrow\!\boxed{\text{C}_2^b}\!\longrightarrow\!\!\circ$$

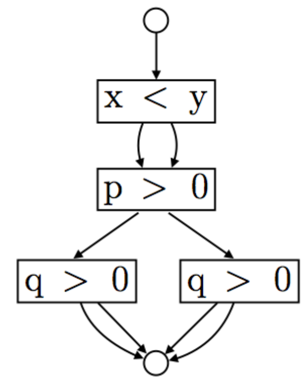
in 

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

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++;
}
```



$(x < y) \cdot (p > 0) \cdot (q > 0)$, $\quad \neg(x < y) \cdot (p > 0) \cdot (q > 0)$,
$(x < y) \cdot (p > 0) \cdot \neg(q > 0)$, $\quad \neg(x < y) \cdot (p > 0) \cdot \neg(q > 0)$,
$(x < y) \cdot \neg(p > 0) \cdot (q > 0)$, $\quad \neg(x < y) \cdot \neg(p > 0) \cdot (q > 0)$,
$(x < y) \cdot \neg(p > 0) \cdot \neg(q > 0)$, $\quad \neg(x < y) \cdot \neg(p > 0) \cdot \neg(q > 0)$.

Trace Semantics Partitioning

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

$\Downarrow$

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

$\Downarrow$

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

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

$\Downarrow$

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

Patrick Cousot. Semantic foundations of program analysis. In S.S. Muchnick & N.D. Jones, editors, *Program Flow Analysis: Theory and Applications*, Ch. 10, pages 303—342, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, U.S.A., 1981.

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 γ_ℓ) is either $(\!| p |\!)$ with p is an element of $\mathbb{D}_\ell$ and $\mathcal{B}$ is empty or $[\![\mathrm{B} : t_t, t_f]\!]$ where B is the first element of all branch condition paths $\underline{\pi_b} \in \mathcal{B}$ and (t_t, t_f) are the left and right subtree of t represent its true and false branch such that $t_t, t_f \in \mathbb{T}(\mathcal{B}_{\setminus\beta}, \mathbb{D}_\ell)$ ($\beta \triangleq \mathrm{B}$ or $\neg\mathrm{B}$ and $\mathcal{B}_{\setminus\beta}$ denotes the removal of β and all branch conditions appearing before in each branch condition path in $\mathcal{B}$).

$$[\![\mathrm{B}_1 : [\![\mathrm{B}_2 : (\!| p_1 |\!), (\!| p_2 |\!)]\!], [\![\mathrm{B}_3 : (\!| p_3 |\!), (\!| p_4 |\!)]\!]]\!]$$

Concretization

Definition 3. Let ρ be the concrete environment assigning concrete values $\rho(x)$ to variables x and $[\![e]\!]\rho$ for the concrete value of the expression e in the concrete environment ρ, the concretization of a binary decesion tree γ_t is either

$$\gamma_t((\!| p |\!)) \triangleq \gamma_\ell(p)$$

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

$$\gamma_t([\![\mathrm{B} : t_t, t_f]\!]) \triangleq \{\rho \mid [\![\mathrm{B}]\!]\rho = \mathit{true} \Longrightarrow \rho \in \gamma_t(t_t) \wedge [\![\mathrm{B}]\!]\rho = \mathit{false} \Longrightarrow \rho \in \gamma_t(t_f)\}$$

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

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

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

$P, Q, \ldots$	$\in$	$\mathbb{T}(\mathcal{B}, \mathbb{D}_\ell)\backslash_{\equiv_t}$	abstract properties
$\sqsubseteq_t$	$\in$	$\mathbb{T} \times \mathbb{T} \to \{\mathit{false}, \mathit{true}\}$	abstract partial order
$\bot_t, \top_t$	$\in$	$\mathbb{T}(\mathcal{B}, \mathbb{D}_\ell)$	infimum, supremum $(\forall P \in \mathbb{T} : \bot_t \sqsubseteq_t P \sqsubseteq_t \top_t)$
$\sqcup_t, \sqcap_t$	$\in$	$\mathbb{T} \times \mathbb{T} \to \mathbb{T}$	abstract join, meet
∇_t, Δ_t	$\in$	$\mathbb{T} \times \mathbb{T} \to \mathbb{T}$	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, $\cdots$

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, $\cdots$

⇢ 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 $t_1, t_2 \in \mathbb{T}(\mathcal{B}, \mathbb{D}_\ell) \setminus \{\bot_t, \top_t\}$,

- Inclusion test: comparing each pair (ℓ_1, ℓ_2) of leaves in (t_1, t_2) where ℓ_1 and ℓ_2 are defined by the same branch condition path $\underline{\pi_b} \in \mathcal{B}$.
  - $t_1 \sqsubseteq_t t_2$ if $\ell_1 \sqsubseteq_\ell \ell_2$ for all pairs of (ℓ_1, ℓ_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.

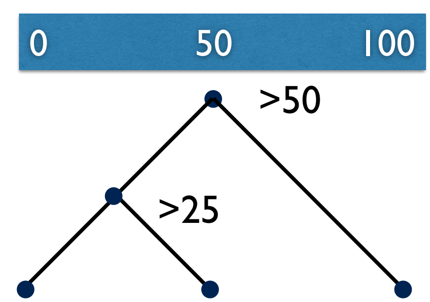
Meet and Join

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

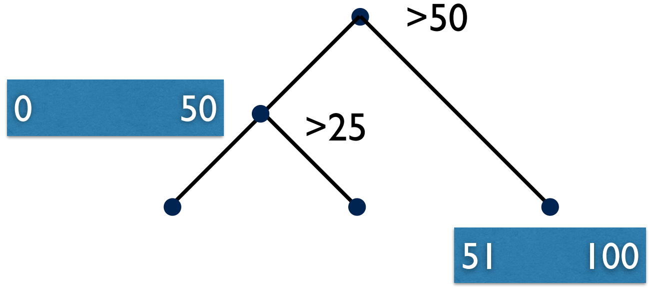
- Meet: for each pair (ℓ_1, ℓ_2) of leaves in (t_1, t_2) where ℓ_1 and ℓ_2 are defined by the same branch condition path $\underline{\pi_b} \in \mathcal{B}$.
  - $\ell = \ell_1 \sqcap_\ell \ell_2$ using the meet $\sqcap_\ell$ in the leaf abstract domain $\mathbb{D}_\ell$.
- Join: for each pair (ℓ_1, ℓ_2) of leaves in (t_1, t_2) where ℓ_1 and ℓ_2 are defined by the same branch condition path $\underline{\pi_b} \in \mathcal{B}$.
  - $\ell = (\ell_1 \sqcup_\ell \ell_2) \sqcap_\ell \mathbb{D}_\ell(\beta_1) \sqcap_\ell \mathbb{D}_\ell(\beta_2) \sqcap_\ell \ldots \sqcap_\ell \mathbb{D}_\ell(\beta_n)$ where $\underline{\pi_b} = \beta_1 \cdot \beta_2 \cdot \ldots \cdot \beta_n$ and $\mathbb{D}_\ell(\beta)$ is the representation of β in $\mathbb{D}_\ell$ (when α_ℓ exists in the leaf abstract domain $\mathbb{D}_\ell$, we can use $\alpha_\ell(\beta)$ instead).

↑ pairwise join and distribute over leaves

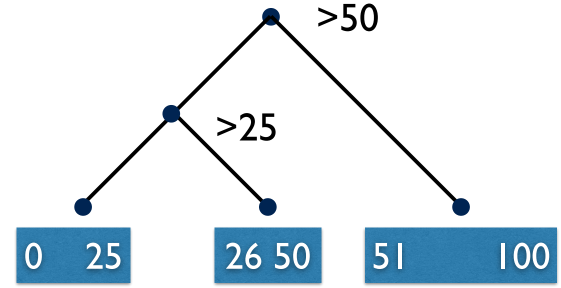
Distribution over leaves



Distribution over leaves



Distribution over leaves



Widening and Narrowing

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

- Widening $t_1 \triangledown_\ell t_2$: for each pair (ℓ_1, ℓ_2) of leaves in (t_1, t_2) where ℓ_1 and ℓ_2 are defined by the same branch condition path $\underline{\pi_b} \in \mathcal{B}$.
  - $\ell = (\ell_1 \triangledown_\ell \ell_2) \sqcap_\ell \mathbb{D}_\ell(\beta_1) \sqcap_\ell \mathbb{D}_\ell(\beta_2) \sqcap_\ell \ldots \sqcap_\ell \mathbb{D}_\ell(\beta_n)$ where $\triangledown_\ell$ is the widening in the leaf abstract domain $\mathbb{D}_\ell$, $\underline{\pi_b} = \beta_1 \cdot \beta_2 \cdot \ldots \cdot \beta_n$ and $\mathbb{D}_\ell(\beta)$ is the representation of β in $\mathbb{D}_\ell$.

↑ pairwise widen and distribute over leaves

- Narrowing $t_2 \vartriangle_t t_1$: for each pair (ℓ_1, ℓ_2) of leaves in (t_1, t_2) where ℓ_1 and ℓ_2 are defined by the same branch condition path $\underline{\pi_b} \in \mathcal{B}$.
  - $\ell = \ell_2 \vartriangle_\ell \ell_1$ using the narrowing $\vartriangle_\ell$ 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:

$$\begin{array}{lcl}
\bot_t \sqcap_t p & \triangleq & \bot_t \\
\top_t \sqcap_t p & \triangleq & (\!| p |\!) \\
t \sqcap_t \mathit{false} & \triangleq & \bot_t \\
t \sqcap_t \mathit{true} & \triangleq & t \\
(\!| p' |\!) \sqcap_t p & \triangleq & (\!| p' \sqcap_\ell \mathbb{D}_\ell(p) |\!) \\
[\![\mathrm{B} : t_l, t_r]\!] \sqcap_t p & \triangleq & [\![\mathrm{B} : t_l \sqcap_t \mathbb{D}_\ell(\mathrm{B}) \sqcap_t \mathbb{D}_\ell(p), t_r \sqcap_t \mathbb{D}_\ell(\neg \mathrm{B}) \sqcap_t \mathbb{D}_\ell(p)]\!]
\end{array}$$

Test Transfer Function

$$f_T[\![\mathrm{B}]\!]t \triangleq t \sqcap_t \mathrm{B}$$

Assignment Transfer Function

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

$t = [\![x \le 50 : (\!|0 \le x \le 50|\!), (\!|\perp_\ell|\!)]\!]$
Assignment: $x = x + 1$

Assignment on leaves

$\Downarrow$

$t' = [\![x \le 50 : (\!|1 \le x \le 51|\!), (\!|\perp_\ell|\!)]\!]$

$\Downarrow$

$t' = [\![x \le 50 : (\!|1 \le x \le 50|\!), (\!|x = 51|\!)]\!]$

reconstruction on leaves

Reconstruction on Leaves

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

↑ assign and redistribute over leaves

Binary Decision Tree Construction

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

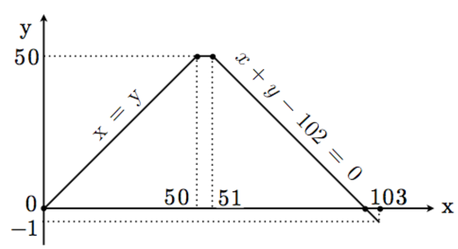
Tree Merging

1. Pick up a branch condition B.
2. Eliminate B (B or ¬B) from each branch condition path in $\mathcal{B}$.
3. For each subtree of the form $[\![B : t_t, t_f]\!]$, 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 $[\![B : t_t, t_f]\!]$ can be replaced by $t'_t \sqcup_t t'_f$.

A small example

Example

```
x = 0; y = 0;
¹while(y >= 0) {
    if (x <= 50) y++;
    else y−−;
    x++;
}
```



- $y >= -1 \wedge x - y >= 0 \wedge x + 52y >= 0$ by Apron
- We choose the polyhedra abstract domain as the leaf abstract domain
- We have $\mathcal{B} = \{x <= 50, \neg(x <= 50)\}$
- Initially, $t_0 = [\![x \le 50 : (\!|x = 0 \wedge y = 0|\!), (\!|\perp_\ell|\!)]\!]$

$t'_0 = [\![x \le 50 : (\!|x = 1 \wedge y = 1|\!), (\!|\perp_\ell|\!)]\!]$ *after one iteration*

$\Downarrow$ *join of initialization and first iteration*

$t_1 = t_0 \sqcup_t t'_0 = [\![x \le 50 : (\!|x = y \wedge 0 \le x \le 1|\!), (\!|\perp_\ell|\!)]\!]$

$\Downarrow$ *widening initialization and first iteration*

$t'_1 = t_0 \triangledown t_1 = [\![x \le 50 : (\!|0 \le x \le 50 \wedge x = y|\!), (\!|\perp_\ell|\!)]\!]$

$\Downarrow$ *increment y and x, reconstruct on leaves*

$t''_1 = [\![x \le 50 : (\!|1 \le x \le 50 \wedge x = y|\!), (\!|x = 51 \wedge y = 51|\!)]\!]$

$\Downarrow$ *join with initialization*

$t_2 = t_1 \sqcup_t t''_1 = [\![x \le 50 : (\!|0 \le x \le 50 \wedge x = y|\!), (\!|x = 51 \wedge y = 51|\!)]\!]$

$\Downarrow$ *third iteration*

$t_3 = [\![x \le 50 : (\!|0 \le x \le 50 \wedge x = y|\!), (\!|x + y - 102 = 0 \wedge 51 \le x \le 52|\!)]\!]$

$\Downarrow$ *widening*

$t'_3 = t_2 \triangledown t_3 = [\![x \le 50 : (\!|0 \le x \le 50 \wedge x = y|\!), (\!|x + y - 102 = 0 \wedge x \ge 51|\!)]\!]$

$\Downarrow$ *fourth iteration, convergence*

$t_4 = [\![x \le 50 : (\!|0 \le x \le 50 \wedge x = y|\!), (\!|x + y - 102 = 0 \wedge 51 \le x \le 103|\!)]\!]$

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?