A Binary Decision Tree Abstract Domain Functor

Junjie Chen and Patrick Cousot
New York University

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 \( \sqcup \) is exact (\( \alpha \) preserves joins)

• The problem is from the imprecise abstraction:
  • convex abstractions do not take the control flow into account precisely enough

A Motivating Example

\[
\begin{align*}
x &= 0; \quad y = 0; \\
\text{while}(y \geq 0) \{ \\
\quad \text{if } (x \leq 50) \quad y++; \\
\quad \text{else} \quad y--; \\
\quad x++; \\
\}
\end{align*}
\]

Intervals: \( x \geq 0 \land y \geq -1 \)

Convex Polyhedra: \( y \geq -1 \land x - y \geq 0 \land x + 52y \geq 0 \)

Idea:

• The reduced cardinal power \( A_2^{A_1} = A_1 \to A_2 \) [CC79]

\[ x = 0; \quad y = 0; \]
\[ \text{while}(y \geq 0) \{ \]
\[ \quad \text{if } (x \leq 50) \quad y++; \]
\[ \quad \text{else} \quad y--; \]
\[ \quad x++; \]
\[ \} \]

The reduced cardinal power with base \( (A_2, t_2, y_2) \) and exponent \( (A_1, t_1, y_1) \) where \( A \equiv \{ \text{let } \alpha \equiv \text{f}(\text{f(a)} \rightarrow \text{g}(\text{a})) \}

\[ x = 0; \quad y = 0; \]
\[ \text{while}(y \geq 0) \{ \]
\[ \quad \text{if } (x \leq 50) \quad y++; \]
\[ \quad \text{else} \quad y--; \]
\[ \quad x++; \]
\[ \} \]

\( \] with exponent \( A_1 \) which is an abstraction of the control flow graph
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.

Syntax

- Consider the following abstract syntax of command:

$$C \in \mathcal{C} ::= \text{skip} \mid x = E \mid C_1 ; C_2 \mid$$

if (B) \{C_1\} else \{C_2\} |

while (B) \{C\}

- Actions describe elementary indivisible program computation steps:

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

Traces

- Trace

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

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

sequence $$\overline{\pi} = A_0 A_1 \ldots A_{n-2} \in A^{n-1}$$ of actions
Trace Semantics

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

\[ S'[x = E] \triangleq \left\{ (x = E, \rho) \xrightarrow{\text{assign}} (\text{stop}, \rho|\text{x := v}) \mid \rho \in \mathcal{E} \land v \in \mathcal{E}[E]\rho \right\} \]

\[ S'[C_1; C_2] \triangleq \left\{ (\pi; C_2) \xrightarrow{\|} (C_2, \rho) \xrightarrow{\pi'} \left( (\text{stop}, \rho') \in S'[C_1] \land (C_2, \rho) \xrightarrow{\pi'} \pi' \in S'[C_2] \right) \right\} \]

... 

The Control Flow Graph Abstraction

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:

\[
\begin{align*}
G[\text{skip}] & \triangleq \longrightarrow_{\text{skip}} \\
G[x := E] & \triangleq \longrightarrow_{x := E} \\
G[\text{if} (B) \{C_1\} \text{ else } \{C_2\}] & \triangleq \text{let } G[C_1] = \longrightarrow_{C_1} \text{ and } G[C_2] = \longrightarrow_{C_2} \text{ in } G[C] \\
G[\text{while} (B) \{C\}] & \triangleq \text{let } G[C] = \longrightarrow_{C} \text{ and } G[C] \text{ in } C \text{ while } (B) \{C\}
\end{align*}
\]
Action Path Semantics of CFG

\[ G^a[\text{skip}] \triangleq \{\text{skip}\} \]
\[ G^a[x := E] \triangleq \{x = E\} \]
\[ G^a[B \cdot C] \triangleq G^a[C] \cup \{-B\} \cdot G^a[B] \]
\[ G^a[B \cdot C] \triangleq G^a[C] \cdot G^a[B] \]
\[ G^a[\text{if} \frac{B}{C}] \triangleq \{\text{false}\} \cup \{B\} \cdot G^a[C] \]
\[ G^a[\text{if} \frac{B}{C}] \triangleq \text{false} \cdot G^a[C] \]
\[ = \{(B \cdot G^a[C])^* \cdot \{-B\}\} \]

The CFG is an abstraction of the trace semantics

\[ \alpha^a(S^e[C]) \subseteq G^a[G[C]] \]

- Soundness:

- The basis for most static analyses

Condition Path Abstraction

- Let
  \[ \alpha^c(\text{skip}) \triangleq \varepsilon \]
  \[ \alpha^c(x = E) \triangleq \varepsilon \]
  \[ \alpha^c(B) \triangleq B \]
  \[ \alpha^c(\neg B) \triangleq \neg B \]
  \[ \alpha^c(\pi_1 \cdot \pi_2) \triangleq \alpha^c(\pi_1) \cdot \alpha^c(\pi_2) \]

- The condition path abstraction collects the sequences of conditions in the action paths \( A \):

  \[ \alpha^c \in \varphi(A^*) \mapsto \varphi((A^C)^*) \]
  \[ \alpha^c(A) \triangleq \{\alpha^c(\pi) | \pi \in A\} \]
Loop Condition Elimination

- Let $A^B$: the set of branch conditions
  $A^L$: the set of loop conditions

- and

  $\alpha^d(A^b) \triangleq A^b$, \hspace{1cm} $\alpha^d(A^l) \triangleq \varepsilon$

  $\alpha^d(\pi_{e_1} \cdot \pi_{e_2}) \triangleq \alpha^d(\pi_{e_1}) \cdot \alpha^d(\pi_{e_2})$

- where $A^b \in A^B$ and $A^l \in A^L$.

- The loop condition elimination collects the sequences of branch conditions from the condition paths:

  $\alpha^d \in \varphi((A^B)^*) \mapsto \varphi((A^B)^*)$

  $\alpha^d(C) \triangleq \{\alpha^d(\pi_c) \mid \pi_c \in C\}$

Branch Condition Path Abstraction

- Let

  $\alpha^\ell(\pi_d) = fold(\pi_d)$

  eliminate duplications of each branch condition while keeping its last occurrence in $\pi_d$.

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

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

  $\alpha^\ell(D) \triangleq \{\alpha^\ell(\pi_d) \mid \pi_d \in D\}$

Duplication Elimination

- $erase(d_1d_2d_3...d_n, d) \triangleq$ if $d_1 == d$ then $erase(d_2d_3...d_n, d)$
  else $d_1 \cdot erase(d_2d_3...d_n, d)$

- $fold(d_1d_2...d_n) \triangleq$ if $d_1d_2...d_n == \varepsilon$ then $\varepsilon$
  else $fold(erase(d_1d_2...d_{n-1}, d_n)) \cdot d_n$

Concretizations

- Action path abstraction

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

- Condition path abstraction

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

- Loop condition elimination

  $\gamma^d(D) \triangleq \{\pi_c \mid \alpha^d(\pi_c) \in D\}$

- Branch condition path abstraction

  $\gamma^\ell(B) \triangleq \{\pi_d \mid \alpha^\ell(\pi_d) \in B\}$
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.

\[ G^b[\text{skip}] \triangleq \bigcirc \quad G^b[x := E] \triangleq \bigcirc \]

\[ G^b[C_1 ; C_2] \triangleq \text{let } G^b[C_1] = \bigcirc C_1 \bigcirc \text{ and } G^b[C_2] = \bigcirc C_2 \bigcirc \text{ in} \]

\[ G^b[\text{if } (B) \{ C_1 \} \text{ else } C_2] \triangleq \text{let } G^b[C_1] = \bigcirc C_1 \bigcirc \text{ and } G^b[C_2] = \bigcirc C_2 \bigcirc \text{ in} \]

\[ G^b[\text{while } (B) \{ C \}] \triangleq \text{let } G^b[C] = \bigcirc C \bigcirc \text{ in} \]

\[ A \text{bstraction of Control Flow Graph!} \]

Example

\[
\begin{align*}
\text{while } (i \leq m) \{ \\
\quad \text{if } (x < y) \ x++; \\
\quad \text{else } y++; \\
\quad \text{if } (p > 0) \\
\quad \quad \text{if } (q > 0) \ r = p + q; \\
\quad \quad \text{else } r = p - q; \\
\quad \text{else} \\
\quad \quad \text{if } (q > 0) \ r = q - p; \\
\quad \quad \text{else } r = -(p + q); \\
\quad i++; \\
\}\end{align*}
\]

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

Trace Semantics Partitioning

Given the trace semantics \( S^P[P] \) of a program \( P \)

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

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

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

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

The Binary Decision Tree Abstraction
Binary Decision Tree

**Definition 2.** A binary decision tree \( t \in \mathbb{T}(\mathcal{B}, \mathbb{D}) \) over the set \( \mathcal{B} \) of branch condition paths (with concretization \( \gamma^a \circ \gamma^c \circ \gamma^d \circ \gamma^f \)) and the leaf abstract domain \( \mathbb{D} \) (with concretization \( \gamma_t \)) is either \( \langle p \rangle \) with \( p \) is an element of \( \mathbb{D} \) and \( \mathcal{B} \) is empty or \( \langle \mathcal{B} : t_1, t_f \rangle \) where \( \mathcal{B} \) is the first element of all branch condition paths \( \pi_k \in \mathcal{B} \) and \( (t_1, t_f) \) are the left and right subtree of \( t \) represent its true and false branch such that \( t_1, t_f \in \mathbb{T}(\mathcal{B}_\beta, \mathbb{D}_t) \) (\( \beta \triangleq B \) or \( -B \) and \( \mathcal{B}_\beta \) denotes the removal of \( \beta \) and all branch conditions appearing before in each branch condition path in \( \mathcal{B} \)).

\[
\left[ B_1 : \left[ B_2 : (p_1), (p_2) \right], \left[ B_3 : (p_3), (p_4) \right] \right]
\]

Concretization

**Definition 3.** Let \( \rho \) 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 \( \rho \), the concretization of a binary decision tree \( \gamma_t \) is either

\[
\gamma_t(\langle p \rangle) \triangleq \gamma_t(p)
\]

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

\[
\gamma_t(\langle \mathcal{B} : t_1, t_f \rangle) \triangleq \{ \rho \mid \mathbb{B}\rho = true \implies \rho \in \gamma_t(t_1) \land \mathbb{B}\rho = false \implies \rho \in \gamma_t(t_f) \}
\]

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

Binary Decision Tree

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

\[
\langle \mathbb{T}(\mathcal{B}, \mathbb{D}) \rangle_{\subseteq_t, \sqsubseteq_t, \sqcup_t, \sqcap_t, \sqvee_t, \sqwedge_t} \quad \mathbb{D}_t \quad \mathcal{B}
\]

on two parameters, a set \( \mathcal{B} \) of branch condition paths and a leaf abstract domain \( \mathbb{D}_t \) where

- \( P, Q, \ldots \in \mathbb{T}(\mathcal{B}, \mathbb{D}) \)
- abstract properties
- \( \subseteq_t \in \mathbb{T} \times \mathbb{T} \rightarrow \{ \text{false}, \text{true} \} \)
- abstract partial order
- \( \sqsubseteq_t, \sqcup_t \in \mathbb{T}(\mathcal{B}, \mathbb{D}) \)
- infimum, supremum
- \( \forall P \in \mathbb{T} : \sqsubseteq_t \subseteq_t P \subseteq_t \sqcup_t \)
- abstract join, meet
- \( \sqvee_t, \sqwedge_t \in \mathbb{T} \times \mathbb{T} \rightarrow \mathbb{T} \)
- abstract widening, narrowing

- 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 \( \mathbb{D}_t \) for the leaves could be any numerical or symbolic algebraic abstract domains such as polyhedra, ⋯
Meet and Join

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

- **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 $\pi_b \in \mathcal{B}$.
  
  $$\ell = \ell_1 \cap \ell_2$$

- **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 $\pi_b \in \mathcal{B}$.

  $$\ell = (\ell_1 \cup \ell_2) \cap_{\ell} \mathcal{D}_\ell(\beta_1) \cap_{\ell} \mathcal{D}_\ell(\beta_2) \cap_{\ell} \ldots \cap_{\ell} \mathcal{D}_\ell(\beta_n)$$

  where $\pi_b = \beta_1 \cdot \beta_2 \cdot \ldots \beta_n$ and $\mathcal{D}_\ell(\beta)$ is the representation of $\beta$ in $\mathcal{D}_\ell$ (when $\alpha_\ell$ exists in the leaf abstract domain $\mathcal{D}_\ell$, we can use $\alpha_\ell(\beta)$ instead).

↑ pairwise join and distribute over leaves
**Widening and Narrowing**

Given two binary decision tree $t_1, t_2 \in T(\mathcal{B}, \mathcal{D}_t) \setminus \{\perp_t, \top_t\}$ and $t_1 \sqsubseteq t_2$,

- **Widening** $t_1 \triangledown t_2$: 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 $\pi_\ell \in \mathcal{B}$.
  - $\ell = (\ell_1 \triangledown \ell_2) \triangledown \ell_1 \mathcal{D}_t(\beta_1) \triangledown \ell_1 \mathcal{D}_t(\beta_2) \triangledown \ell_1 \mathcal{D}_t(\beta_n)$ where $\triangledown \ell$ is the widening in the leaf abstract domain $\mathcal{D}_t$, $\pi_b = \beta_1 \cdot \beta_2 \cdot \ldots \cdot \beta_n$ and $\mathcal{D}_t(\beta)$ is the representation of $\beta$ in $\mathcal{D}_t$.

- **Pairwise widen and distribute over leaves**

- **Narrowing** $t_2 \triangledown t_1$: 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 $\pi_\ell \in \mathcal{B}$.
  - $\ell = \ell_2 \triangledown \ell_1$ using the narrowing $\Delta_\ell$ in the leaf abstract domain $\mathcal{D}_t$.

---

**Reduction of Binary Decision Tree by an Abstract Property**

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

- $\perp_t \cap_t p \triangleq \perp_t$
- $\top_t \cap_t p \triangleq \{p\}$
- $t \cap_t \text{false} \triangleq \perp_t$
- $t \cap_t \text{true} \triangleq t$
- $\{p'\} \cap_t p \triangleq \{p' \cap_t \mathcal{D}_t(p)\}$
- $[B : t_1, t_r] \cap_t p \triangleq [B : t_1 \cap_t \mathcal{D}_t(B) \cap_t \mathcal{D}_t(p) \cap_t \mathcal{D}_t(\neg B) \cap_t \mathcal{D}_t(p)]$
Test Transfer Function

\[ f_T[B]t \triangleright t \cap t \cup B \]

Assignment Transfer Function

Given a binary decision tree \( t \in T(B, D_t) \), the assignment \( x = E \) can be performed at each leaf in \( t \) by using the assignment transfer function of \( D_t \).

\[
\begin{align*}
  t &= \llbracket x \leq 50 \land \{0 \leq x \leq 50\}, \{\bot\} \rrbracket \\
  \text{Assignment: } x &\leftarrow x + 1 \\
  t' &= \llbracket x \leq 50 \land \{1 \leq x \leq 51\}, \{\bot\} \rrbracket \\
  t'' &= \llbracket x \leq 50 \land \{1 \leq x \leq 50\}, \{x = 51\} \rrbracket
\end{align*}
\]

Reconstruction on Leaves

1. Collecting all leave properties in \( t \), let it be \( \{p_1, p_2, \ldots, p_n\} \);

2. For each leaf in \( t \), let \( \pi_b = \beta_1 \cdot \beta_2 \cdot \ldots \cdot \beta_n \) be the branch condition path leading to it. We then calculate \( p'_i = p_i \cap \ell \cap D_t(\beta_1 \wedge \beta_2 \wedge \ldots \wedge \beta_n) \).

3. For each leaf in \( t \), update it with \( p'_1 \cup \ell \cup p'_2 \cup \ell \cup \ldots \cup \ell \cup p'_n \).

↑ 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 $B$.
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 $t_t \sqcup t_f$.
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 $t'_t \sqcup t'_f$.

Example

$x = 0; y = 0;$
\begin{verbatim}
while(y >= 0) {
  if (x <= 50) y++; 
  else y--; 
  x++; 
}
\end{verbatim}

- $y >= -1 \land x - y >= 0 \land x + 52y >= 0$ by Apron
- We choose the polyhedra abstract domain as the leaf abstract domain
- We have $B = \{x <= 50, \neg(x <= 50)\}$
- Initially, $t_0 = [x \leq 50 : (x = 0 \land y = 0), (\bot_t)]$

A small example

$t'_0 = [x \leq 50 : (x = 1 \land y = 1), (\bot_t)]$
\hspace{1cm} after one iteration
\hspace{1cm} join of initialization and first iteration
\hspace{1cm} $t_1 = t_0 \sqcup t_0' = [x \leq 50 : (x = y \land 0 \leq x \leq 1), (\bot_t)]$
\hspace{1cm} $t'_0 \sqcup t'_0$
\hspace{1cm} widening initialization and first iteration
\hspace{1cm} $t'_1 = t_0 \lor t_1 = [x \leq 50 : (0 \leq x \leq 50 \land x = y), (\bot_t)]$
\hspace{1cm} $t'_1 \lor t'_1$
\hspace{1cm} increment $y$ and $x$, reconstruct on leaves
\hspace{1cm} $t'_2 = [x \leq 50 : (1 \leq x \leq 50 \land x = y), (x = 51 \land y = 51)]$
\hspace{1cm} $t'_2$
\hspace{1cm} join with initialization
\hspace{1cm} $t_2 = t_1 \sqcup t_2' = [x \leq 50 : (0 \leq x \leq 50 \land x = y), (x = 51 \land y = 51)]$
\hspace{1cm} $t_2 \sqcup t_2'$
\hspace{1cm} third iteration
\hspace{1cm} $t_3 = [x \leq 50 : (0 \leq x \leq 50 \land x = y), (x + y - 102 = 0 \land 51 \leq x \leq 52)]$
\hspace{1cm} $t_3$
\hspace{1cm} widening
\hspace{1cm} $t'_3 = t_2 \lor t_3 = [x \leq 50 : (0 \leq x \leq 50 \land x = y), (x + y - 102 = 0 \land x \geq 51)]$
\hspace{1cm} $t'_3 \lor t'_3$
\hspace{1cm} fourth iteration, convergence
\hspace{1cm} $t_4 = [x \leq 50 : (0 \leq x \leq 50 \land x = y), (x + y - 102 = 0 \land 51 \leq x \leq 103)]$
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?