A Binary Decision Tree Abstract Domain Functor

Junjie Chen(✉) and Patrick Cousot

New York University, New York, USA
junjie@cs.nyu.edu

Abstract. We present an abstract domain functor whose elements are binary decision trees. It is parameterized by decision nodes which are a set of boolean tests appearing in the programs and by a numerical or symbolic abstract domain whose elements are the leaves. We first define the branch condition path abstraction which forms the decision nodes of the binary decision trees. It also provides a new prospective on partitioning the trace semantics of programs as well as separating properties in the leaves. We then discuss our binary decision tree abstract domain functor by giving algorithms for inclusion test, meet and join, transfer functions and extrapolation operators. We think the binary decision tree abstract domain functor may provide a flexible way of adjusting the cost/precision ratio in path-dependent static analysis.

1 Introduction

In past decades, abstract interpretation [5] has been widely and successfully applied to the static analysis and verification of programs. Abstract domains, one of the key concepts in abstract interpretation, aim at collecting information about the set of all possible values of the program variables. The biggest advantage of using abstract domains instead of logic predicates is that they are fully automatic and can easily scale up. Intervals [4], octagons [14] and polyhedra [6] are the most commonly used numerical abstract domains. These abstract domains are inferring a conjunction of linear constraints to maintain the information of all possible values of program variables and/or the possible relationships between them. The absence of disjunctions may cause rough approximations and produce much less precise results, gradually leading to false alarms or even worse to the complete failure to prove the desired program property.

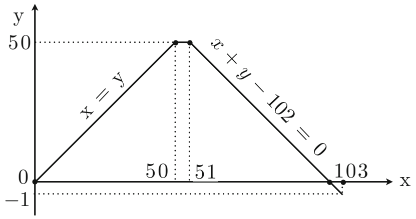
Let us consider the following example which is modified from the one in [9]:

Work supported in part by NSF grant CNS-1446511.

S. Blazy and T. Jensen (Eds.): SAS 2015, LNCS 9291, pp. 36–53, 2015.
DOI: 10.1007/978-3-662-48288-9_3

Example 1. A motivating example.

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



We know that the strongest invariant at program point ^{l} is (0 <= x <= 50 ∧ x = y) ∨ (51 <= x <= 103 ∧ $x + y - 102 = 0$). When we use the APRON numerical abstract domain library [12] to generate the invariant at program point ^{l}, we get x >= 0 ∧ y >= −1 with the box (interval) abstract domain and y >= −1∧x−y >= 0∧x+52y >= 0 with the polka (convex polyhedra) abstract domain. Both analyses are very imprecise compared to the strongest one. This is because the true and false branches of "if (x <= 50)" have different behaviors and those abstract domains do not consider them separately. □

Hence, we propose the binary decision tree abstract domain that takes those branches into consideration.

2 Action Path Semantics

We consider the following *abstract syntax* of commands which describes the abstract syntax trees (AST) representing the syntactic structure of source code:

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

The *trace semantics* $\mathcal{S}^t[\![\mathrm{C}]\!]$ of a command C describes all possible observations of executions of the command C. A *trace* π of length $|\pi| \triangleq n \geq 1$ is a pair $\pi = \langle \overline{\pi}, \underline{\pi} \rangle$ of a finite sequence $\overline{\pi} = \sigma_0\sigma_1...\sigma_{n-1}$ of states separated by a finite sequence $\underline{\pi} = \mathrm{A}_0\mathrm{A}_1...\mathrm{A}_{n-2}$ of actions. *States* record the current values of variables in the environment/memory as well as a label/control point specifying what remains to be executed while *actions* record which elementary indivisible elementary program steps are computed during the execution of commands. An action A $\in \mathbb{A}$ is either no operation "skip", an assignment "x = E" or a test which output is either true (tt) or false (ff). We use action "B" to record that the Boolean expression B evaluated to true (tt), while action "¬B" records that the Boolean expression B evaluated to false (ff).

The *action path abstraction* $\alpha^a(\mathcal{S})$ collects the set of action paths, that is sequences of actions performed along the traces of a trace semantics $\mathcal{S}$. Given a trace $\pi = \langle \overline{\pi}, \underline{\pi} \rangle$, $\alpha^a(\pi) \triangleq \underline{\pi}$ collects the sequence of actions $\underline{\pi}$ executed along that trace, which may be empty ε for traces reduced to a single state.

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

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

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

Note that α^a preserves both arbitrary unions and non-empty intersections. We then have the following theorem:

Theorem 1 (Homomorphic Abstraction). *Given a function $h : C \mapsto A$, let $\alpha_h(X) = \{h(x) \mid x \in X\}$ and $\gamma_h(Y) = \{x \mid h(x) \in Y\}$, then α_h and γ_h form a Galois connection:*

$$(\wp(C), \subseteq) \underset{\alpha_h}{\overset{\gamma_h}{\leftrightarrows}} (\wp(A), \subseteq) \tag{1}$$

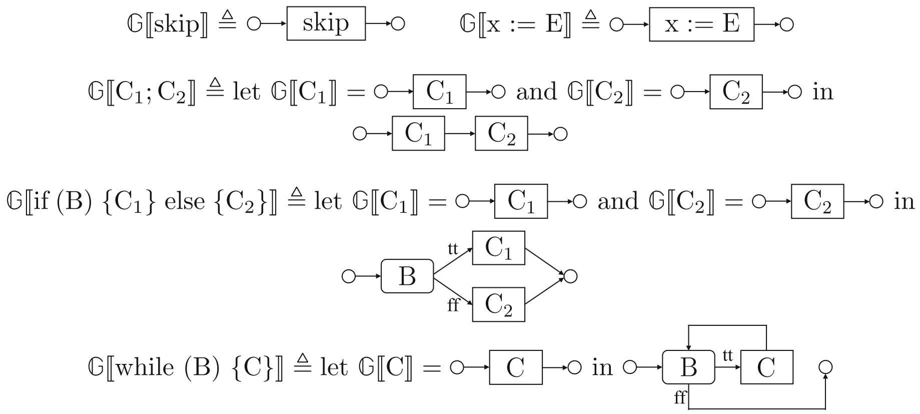
Proof. For all $X \in \wp(C)$ and $Y \in \wp(A)$,

$$\begin{aligned}
& \alpha_h(X) \subseteq Y \\
\iff & \{h(x) \mid x \in X\} \subseteq Y && \text{(definition of } \alpha_h \text{)} \\
\iff & \forall x \in X : h(x) \in Y && \text{(definition of } \subseteq \text{)} \\
\iff & X \subseteq \{x \mid h(x) \in Y\} && \text{(definition of } \subseteq \text{)} \\
\iff & X \subseteq \gamma_h(Y) && \text{(definition of } \gamma_h \text{)}
\end{aligned}$$

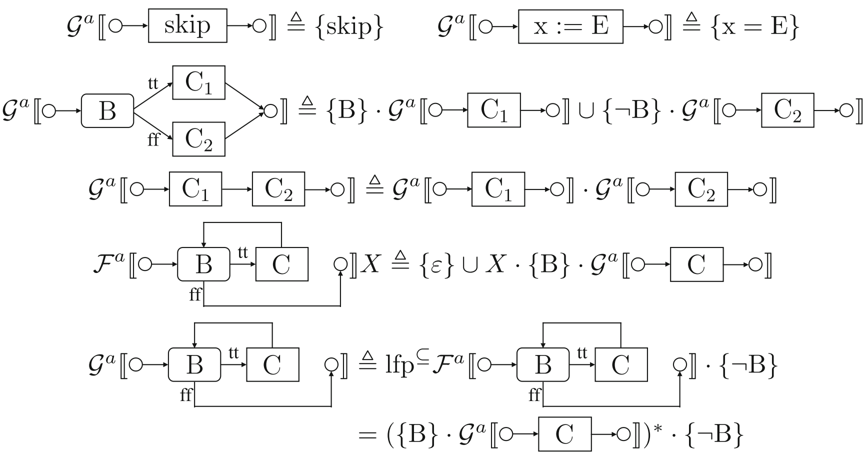
□

Hence, by defining $\gamma^a(\mathcal{A}) \triangleq \{\pi \mid \alpha^a(\pi) \in \mathcal{A}\}$, we will have α^a and γ^a form the Galois connection by Theorem 1 where h is α^a.

A *control flow graph* (CFG) is a directed graph, in which nodes correspond to the actions in the program and the edges represent the possible flow of control. The CFG $\mathbb{G}[\![\mathrm{C}]\!]$ of command C can be built by structural induction on the syntax of the command C:



Then the *action path semantics* $\mathcal{G}^a[\![\mathbb{G}[\![\mathrm{C}]\!]]\!]$ of CFG $\mathbb{G}[\![\mathrm{C}]\!]$ of command C can be defined as:


$\mathcal{G}^a[\![\circ \rightarrow \boxed{\text{skip}} \rightarrow \circ]\!] \triangleq \{\text{skip}\}$
$\mathcal{G}^a[\![\circ \rightarrow \boxed{\mathrm{x} := \mathrm{E}} \rightarrow \circ]\!] \triangleq \{\mathrm{x} = \mathrm{E}\}$
B
tt
ff
C_1
C_2
$\triangleq \{\mathrm{B}\} \cdot \mathcal{G}^a[\![\circ \rightarrow \boxed{\mathrm{C}_1} \rightarrow \circ]\!] \cup \{\neg\mathrm{B}\} \cdot \mathcal{G}^a[\![\circ \rightarrow \boxed{\mathrm{C}_2} \rightarrow \circ]\!]$
$\mathcal{G}^a[\![\circ \rightarrow \boxed{\mathrm{C}_1} \rightarrow \boxed{\mathrm{C}_2} \rightarrow \circ]\!] \triangleq \mathcal{G}^a[\![\circ \rightarrow \boxed{\mathrm{C}_1} \rightarrow \circ]\!] \cdot \mathcal{G}^a[\![\circ \rightarrow \boxed{\mathrm{C}_2} \rightarrow \circ]\!]$
$\mathcal{F}^a$
$X \triangleq \{\varepsilon\} \cup X \cdot \{\mathrm{B}\} \cdot \mathcal{G}^a[\![\circ \rightarrow \boxed{\mathrm{C}} \rightarrow \circ]\!]$
$\mathcal{G}^a$
$\triangleq \mathrm{lfp}^{\subseteq} \mathcal{F}^a$
$\cdot \{\neg\mathrm{B}\}$
$= (\{\mathrm{B}\} \cdot \mathcal{G}^a[\![\circ \rightarrow \boxed{\mathrm{C}} \rightarrow \circ]\!])^* \cdot \{\neg\mathrm{B}\}$

The following Theorem 2 states that the action path semantics of the control flow graph of a program is an over-approximation, hence a sound abstraction, of the action paths that would be collected directly from the trace semantics.

Theorem 2. $\alpha^a(\mathcal{S}^t[\![\mathrm{C}]\!]) \subseteq \mathcal{G}^a[\![\mathbb{G}[\![\mathrm{C}]\!]]\!]$.

Proof. The proof can be done by the structural induction on the syntax of the command C. More details can be found in the Appendix of [2]. □

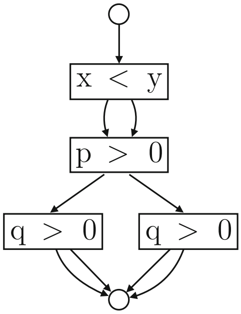
3 Branch Condition Path Abstraction

We introduce branch condition graphs $\mathbb{G}^b[\![\mathrm{C}]\!]$ of command C which can be viewed as further abstractions of the control flow graphs $\mathbb{G}[\![\mathrm{C}]\!]$. We define the branch condition path semantics $\mathcal{G}^b[\![\mathbb{G}^b[\![\mathrm{C}]\!]]\!]$ as an abstract interpretation α^b of the action path semantics $\mathcal{G}^a[\![\mathbb{G}[\![\mathrm{C}]\!]]\!]$ of the control flow graph $\mathbb{G}[\![\mathrm{C}]\!]$ of command C.

3.1 Branch Condition Graph

A *branch condition* is the test B occurring in a command "if (B) $\{\mathrm{C}_1\}$ else $\{\mathrm{C}_2\}$" while a *loop condition* is the test B occurring in a command "while (B) {C}". 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. An edge from node A to node B means that the branch condition corresponding to node B occurs after the branch condition corresponding to node A in the program and there are no other branch conditions occurring between them. A trace from the entry point to the exit point of a BCG is called *branch condition path*. We use B to denote the true branch while ¬B denotes the false branch.

Example 2. Consider the following branch condition graph:



Its branch condition paths include:

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

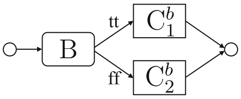
□

The branch condition graph $\mathbb{G}^b[\![\mathrm{C}]\!]$, like the CFG, can be defined by structural induction on the syntax of the command C:

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

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

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

in 

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

Note that the concatenation of $\circ\!\longrightarrow\!\circ$ and $\circ\!\longrightarrow\!\circ$ is still $\circ\!\longrightarrow\!\circ$

3.2 Branch Condition Path Abstraction

We abstract finite action paths $\mathrm{A}_1 \cdot \mathrm{A}_2 \cdot \ldots \cdot \mathrm{A}_n, n \geq 0$ by the finite branch condition path $\mathrm{A}_1^b \cdot \mathrm{A}_2^b \cdot \ldots \cdot \mathrm{A}_m^b, m \leq n$ where $\mathrm{A}_1^b = \mathrm{A}_p, \mathrm{A}_2^b = \mathrm{A}_q, \ldots, \mathrm{A}_m^b = \mathrm{A}_r, 1 \leq p < q < \ldots < r \leq n$ are distinct branch conditions. The branch condition path is empty ε when there are no branch conditions occurred in the action path. We say that two branch conditions $\mathrm{A}_1^b, \mathrm{A}_2^b$ are equal if and only if A_1^b and A_2^b occur at the same program point. Moreover, each branch condition in the branch condition path must be the last occurrence in the action path being abstracted, that is, if A_i^b is a branch condition in the branch condition path $\mathrm{A}_1^b \cdot \mathrm{A}_2^b \cdot \ldots \cdot \mathrm{A}_m^b$ abstracting the action path $\mathrm{A}_1 \cdot \mathrm{A}_2 \cdot \ldots \cdot \mathrm{A}_n$ where $\mathrm{A}_i^b = \mathrm{A}_j$, then $\forall k : j < k \leq n, \mathrm{A}_i^b \neq \mathrm{A}_k$. Note that we only consider finite action paths hence safety properties.

Condition Path Abstraction. The condition path abstraction collects the set of finite sequences of conditions performed along the action path $\underline{\pi}$ and ignores any skip and assignment in $\underline{\pi}$. Given an action path $\underline{\pi}$, $\alpha^c(\underline{\pi})$ collects the sequence of conditions in the action path, which may be empty ε when there are no conditions occurred in the action path, by the following induction rules:

$$\begin{aligned} \alpha^c(\mathrm{skip}) &\triangleq \varepsilon & \alpha^c(\mathrm{B}) &\triangleq \mathrm{B} \\ \alpha^c(\mathrm{x} = \mathrm{E}) &\triangleq \varepsilon & \alpha^c(\neg\mathrm{B}) &\triangleq \neg\mathrm{B} \\ \alpha^c(\underline{\pi}_1 \cdot \underline{\pi}_2) &\triangleq \alpha^c(\underline{\pi}_1) \cdot \alpha^c(\underline{\pi}_2) \end{aligned}$$

Note that $\varepsilon \cdot \underline{\pi_c} = \underline{\pi_c} \cdot \varepsilon = \underline{\pi_c}$. Let $\mathbb{A}^C$ be the set of conditions and $(\mathbb{A}^C)^*$ be the set of finite, possible empty, condition paths. Given a set of action paths $\mathcal{A}$, $\alpha^c(\mathcal{A})$ collects the sequences of conditions in the action paths $\mathcal{A}$:

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

It follows that α^c preserves arbitrary unions and non-empty intersections. By defining $\gamma^c(\mathcal{C}) \triangleq \{\underline{\pi} \mid \alpha^c(\underline{\pi}) \in \mathcal{C}\}$, we will have:

Corollary 1.

$$(\wp(\mathbb{A}^*), \subseteq) \underset{\alpha^c}{\overset{\gamma^c}{\longleftrightarrow}} (\wp((\mathbb{A}^C)^*), \subseteq) \tag{2}$$

Proof. By Theorem 1 where h is α^c. □

Loop Condition Elimination. Given a finite condition path $\underline{\pi_c}$, $\alpha^d(\underline{\pi_c})$ collects the finite sequence of branch conditions (with duplications) by eliminating all loop conditions in $\underline{\pi_c}$. This sequence may be empty ε when there are no branch conditions occurred in $\underline{\pi_c}$. Let $\mathbb{A}^B$ be the set of branch conditions and $\mathbb{A}^L$ be the set of loop conditions, thus $\mathbb{A}^C \triangleq \mathbb{A}^B \cup \mathbb{A}^L$ and $\mathbb{A}^B \cap \mathbb{A}^L = \emptyset$. Note that we distinguish those conditions by the program points where they occur, not by themselves.

For all $\mathrm{A}^b \in \mathbb{A}^B$ and $\mathrm{A}^l \in \mathbb{A}^L$, we have

$$\alpha^d(\mathrm{A}^b) \triangleq \mathrm{A}^b \qquad \text{and} \qquad \alpha^d(\mathrm{A}^l) \triangleq \varepsilon. \tag{3}$$

Then given two condition paths $\underline{\pi_c}_1$ and $\underline{\pi_c}_2$, we have

$$\alpha^d(\underline{\pi_c}_1 \cdot \underline{\pi_c}_2) \triangleq \alpha^d(\underline{\pi_c}_1) \cdot \alpha^d(\underline{\pi_c}_2). \tag{4}$$

Note that $\varepsilon \cdot \underline{\pi_d} = \underline{\pi_d} \cdot \varepsilon = \underline{\pi_d}$. Let $(\mathbb{A}^B)^*$ be the set of finite, possible empty, sequences of branch conditions. Given a set of condition paths $\mathcal{C}$, $\alpha^d(\mathcal{C})$ collects the sequences of branch conditions (with duplications) from the condition paths $\mathcal{C}$:

$$\begin{aligned}\alpha^d &\in \wp((\mathbb{A}^C)^*) \mapsto \wp((\mathbb{A}^B)^*)\\ \alpha^d(\mathcal{C}) &\triangleq \{\alpha^d(\underline{\pi_c}) \mid \underline{\pi_c} \in \mathcal{C}\}\end{aligned}$$

It's easy to see that α^d preserves both arbitrary unions and non-empty intersections. By defining $\gamma^d(\mathcal{D}) \triangleq \{\underline{\pi_c} \mid \alpha^d(\underline{\pi_c}) \in \mathcal{D}\}$, we will have:

Corollary 2.

$$(\wp((\mathbb{A}^C)^*), \subseteq) \underset{\alpha^d}{\overset{\gamma^d}{\leftrightarrows}} (\wp((\mathbb{A}^B)^*), \subseteq) \tag{5}$$

Proof. By Theorem 1 where h is α^d. □

Duplication Elimination. The branch condition paths are the sequences of branch conditions without duplications. In this part, we introduce the abstraction function that eliminates duplications in any sequence of branch conditions.

We first define two functions that are used in the abstraction function α^ℓ. Given a sequence *seq* and an element d of *seq*, $erase(seq, d)$ eliminates all elements in *seq* that is equal to d:

$$\begin{aligned}erase(d_1d_2d_3...d_n, d) \triangleq \text{if } d_1 = d \text{ then } & erase(d_2d_3...d_n, d)\\ \text{else } & d_1 \cdot erase(d_2d_3...d_n, d)\end{aligned} \tag{6}$$

Note that $erase(seq, d)$ may return the empty sequence ε. Then $fold(seq)$ eliminates the duplications of each element in *seq* starting from the last element:

$$\begin{aligned}fold(d_1d_2...d_n) \triangleq\ & \text{if } d_1d_2...d_n = \varepsilon \text{ then } \varepsilon\\ & \text{else } fold(erase(d_1d_2...d_{n-1}, d_n)) \cdot d_n\end{aligned} \tag{7}$$

Hence, given a sequence of branch conditions $\underline{\pi_d}$, $\alpha^\ell(\underline{\pi_d}) = fold(\underline{\pi_d})$ eliminates duplications of each branch condition while keeping its last occurrence in $\underline{\pi_d}$. Let $\mathbb{D}$ be the set of sequences of branch conditions that have duplications. Given a set of sequences of branch conditions $\mathcal{D}$, $\alpha^\ell(\mathcal{D})$ collects branch condition paths (sequences of branch conditions without duplications):

$$\begin{aligned}\alpha^\ell &\in \wp((\mathbb{A}^B)^*) \mapsto \wp((\mathbb{A}^B)^* \setminus \mathbb{D})\\ \alpha^\ell(\mathcal{D}) &\triangleq \{\alpha^\ell(\underline{\pi_d}) \mid \underline{\pi_d} \in \mathcal{D}\}\end{aligned}$$

Similarly, we have α^ℓ preserves both arbitrary unions and non-empty intersections. By defining $\gamma^\ell(\mathcal{B}) \triangleq \{\underline{\pi_d} \mid \alpha^\ell(\underline{\pi_d}) \in \mathcal{B}\}$, we will have:

Corollary 3.

$$(\wp((\mathbb{A}^B)^*), \subseteq) \underset{\alpha^\ell}{\overset{\gamma^\ell}{\leftrightarrows}} (\wp((\mathbb{A}^B)^* \setminus \mathbb{D}), \subseteq) \tag{8}$$

Proof. By Theorem 1 where h is α^ℓ. □

Branch Condition Path Abstraction. The branch condition path abstraction $\alpha^b[\![\mathcal{A}]\!]$ collects the branch condition paths, which is the set of sequences of branch conditions with no duplications along the action paths in $\mathcal{A}$. It can be defined by the composition of $\alpha^c, \alpha^d, \alpha^\ell$ defined in the previous sections as:

$$\begin{aligned}\alpha^b &\in \wp(\mathbb{A}^*) \mapsto \wp((\mathbb{A}^B)^* \setminus \mathbb{D})\\ \alpha^b(\mathcal{A}) &\triangleq \alpha^\ell \circ \alpha^d \circ \alpha^c(\mathcal{A})\end{aligned}$$

Respectively, the concretization function $\gamma^b(\mathcal{B})$ can be defined by the composition of $\gamma^c, \gamma^d, \gamma^\ell$ as:

$$\begin{aligned}\gamma^b &\in \wp((\mathbb{A}^B)^* \setminus \mathbb{D}) \mapsto \wp(\mathbb{A}^*)\\ \gamma^b(\mathcal{B}) &\triangleq \gamma^c \circ \gamma^d \circ \gamma^\ell(\mathcal{B})\end{aligned}$$

It follows that α^b and γ^b form a Galois connection:

$$(\wp(\mathbb{A}^*), \subseteq) \underset{\alpha^b}{\overset{\gamma^b}{\leftrightarrows}} (\wp((\mathbb{A}^B)^* \setminus \mathbb{D}), \subseteq) \tag{9}$$

Proof. The composition of Galois connections is still a Galois connection. □

Example 3. In Example 1, let $\mathcal{A}$ be all possible action path semantics of its CFG, then $\alpha^b(\mathcal{A}) = \{\mathrm{x} \leq 50, \neg(\mathrm{x} \leq 50)\}$.

4 Binary Decision Tree Abstract Domain Functor

We introduce the binary decision tree abstract domain functor to represent and manipulate invariants in the form of binary decision trees. The abstract property will be represented by the disjunction of leaves which are separated by the values of binary decisions, i.e., boolean tests, which will be organized in the decision nodes of the binary decision trees.

4.1 Definition

Given the trace semantics $\mathcal{S}^t[\![\mathrm{P}]\!]$ of a program P, $\alpha^b \circ \alpha^a(\mathcal{S}^t[\![\mathrm{P}]\!])$ abstracts $\mathcal{S}^t[\![\mathrm{P}]\!]$ into a finite set $\mathcal{B}$ of branch condition paths where $\mathcal{B} = \{\underline{\pi_b}_1, ..., \underline{\pi_b}_N\}$. Then for $\underline{\pi_b}_i \in \mathcal{B}$, we have $\gamma^a \circ \gamma^b(\underline{\pi_b}_i) \cap \mathcal{S}^t[\![\mathrm{P}]\!] \subseteq \mathcal{S}^t[\![\mathrm{P}]\!]$ and $\bigcup_{i \leq N}(\gamma^a \circ \gamma^b(\underline{\pi_b}_i) \cap \mathcal{S}^t[\![\mathrm{P}]\!]) = \mathcal{S}^t[\![\mathrm{P}]\!]$. Moreover, for all distinct pairs $(\underline{\pi_b}_1, \underline{\pi_b}_2) \in \mathcal{B} \times \mathcal{B}$, we have $(\gamma^a \circ \gamma^b(\underline{\pi_b}_1) \cap \mathcal{S}^t[\![\mathrm{P}]\!]) \cap (\gamma^a \circ \gamma^b(\underline{\pi_b}_2) \cap \mathcal{S}^t[\![\mathrm{P}]\!]) = \emptyset$. Each branch condition path $\underline{\pi_b}_i$ defines a subset of the trace semantics $\mathcal{S}^t[\![\mathrm{P}]\!]$ of a program P. If we can generate the invariants for each program point only using the information of one subset of the trace semantics, then for each program point, we will get a finite set of invariants. It follows that the disjunction of such set of invariants forms the invariant of that program point. Hence, we encapsulate the set of branch condition paths into the decision nodes of a binary decision tree where each top-down path (without leaf) of the binary decision tree represents a branch condition path, and store in each leaf nodes the invariant generated from the information of the subset of the trace semantics defined by the corresponding branch condition path.

We denote the binary decision tree in the parenthesized form

$$[\![\, \mathrm{B}_1 : [\![\, \mathrm{B}_2 : (\!|\, P_1 \,|\!), (\!|\, P_2 \,|\!) \,]\!], [\![\, \mathrm{B}_3 : (\!|\, P_3 \,|\!), (\!|\, P_4 \,|\!) \,]\!] \,]\!]$$

where $\mathrm{B}_1, \mathrm{B}_2, \mathrm{B}_3$ are decisions (branch conditions) and P_1, P_2, P_3, P_4 are invariants. It encodes the fact that either if B_1 and B_2 are both true then P_1 holds, or if B_1 is true and B_2 is false then P_2 holds, or if B_1 is false and B_3 is true then P_3 holds, or if B_1 and B_3 are both false then P_4 holds. The parenthesized representation of binary trees uses $(\!|\, \ldots \,|\!)$ for leaves and $[\![\, \mathrm{B} : t_l, t_r \,]\!]$ for the decision B and t_l (resp. t_r) represents its left subtree (resp. right subtree). In first order logic, the above binary decision tree would be be written as $(\mathrm{B}_1 \wedge \mathrm{B}_2 \wedge P_1) \vee (\mathrm{B}_1 \wedge \neg\mathrm{B}_2 \wedge P_2) \vee (\neg\mathrm{B}_1 \wedge \mathrm{B}_3 \wedge P_3) \vee (\neg\mathrm{B}_1 \wedge \neg\mathrm{B}_3 \wedge P_4)$ with an implicit universal quantification over free variables.

Let $D(\mathcal{B})$ denote the set of all branch conditions appearing in $\mathcal{B}$. Let $\beta = \mathrm{B}$ or $\neg\mathrm{B}$ and $\mathcal{B}_{\setminus\beta}$ denote the removal of β and all branch conditions appearing before in each branch condition path in $\mathcal{B}$, then we define the binary decision tree as:

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 $\gamma^a \circ \gamma^b$) 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 $[\![\, B : t_t, t_f \,]\!]$ where $B \in D(\mathcal{B})$ 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 $t_t, t_f \in \mathbb{T}(\mathcal{B}_{\setminus\beta}, \mathbb{D}_\ell)$.* □

Example 4. In Example 1, the binary decision tree at program point ^{l} will be $t = [\![\mathrm{x} \leq 50 : (\!|\, 0 \leq \mathrm{x} \leq 50 \wedge \mathrm{x} = \mathrm{y} \,|\!), (\!|\, 51 \leq \mathrm{x} \leq 103 \wedge \mathrm{x} + \mathrm{y} - 102 = 0 \,|\!)]\!]$.

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, we can then define the concretization of the binary decision tree as

Definition 3. *The concretization of a binary decesion tree γ_t is either*

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

when the binary decision tree is reduced to a leaf or

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

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

Given $t_1, t_2 \in \mathbb{T}(\mathcal{B}, \mathbb{D}_\ell)$, we say that $t_1 \equiv_t t_2$ if and only if $\gamma_t(t_1) = \gamma_t(t_2)$. Let $\mathbb{T}(\mathcal{B}, \mathbb{D}_\ell)\backslash_{\equiv_t}$ be the quotient by the equivalence relation $\equiv_t$. The binary decision tree abstract domain functor is defined as:

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

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

on two parameters, a set $\mathcal{B}$ of branch condition paths and a leaf abstract domain $\mathbb{D}_\ell$ where ($\mathbb{T}$ is short for $\mathbb{T}(\mathcal{B}, \mathbb{D}_\ell)\backslash_{\equiv_t}$)

$$\begin{array}{ll}
P, Q, ... \in \mathbb{T} & \textit{abstract properties} \\
\sqsubseteq_t \in \mathbb{T} \times \mathbb{T} \to \{\textit{false}, \textit{true}\} & \textit{abstract partial order} \\
\bot_t, \top_t \in \mathbb{T} & \textit{infimum, supremum} \\
 & (\forall P \in \mathbb{T} : \bot_t \sqsubseteq_t P \sqsubseteq_t \top_t) \qquad (10) \\
\sqcup_t, \sqcap_t \in \mathbb{T} \times \mathbb{T} \to \mathbb{T} & \textit{abstract join, meet} \\
\triangledown_t, \vartriangle_t \in \mathbb{T} \times \mathbb{T} \to \mathbb{T} & \textit{abstract widening, narrowing}
\end{array}$$

□

The set $\mathcal{B}$ of branch condition paths can be constructed from the CFG of the program. It can be done either in the pre-analysis or on the fly during the analysis. The static analyzer designer should allow to change the maximal length of branch condition paths in $\mathcal{B}$ so as to be able to adjust the cost/precision ratio of the analysis. The leaf abstract domain $\mathbb{D}_\ell$ for the leaves could be any numerical or symbolic abstract domains such as intervals, octagons and polyhedra, array domains, etc., or even the reduced product of two or more abstract domains. A list of available abstract domains that can be used at the leaves would be another option of the static analyzer designer. We can use any of these options to build a particular instance of the binary decision tree abstract functor. The advantage of this modular approach is that we can change those options to adjust the cost/precision ratio without having to change the structure of the analyzer.

4.2 Binary Operations

Inclusion and Equality. Given two binary decision tree $t_1, t_2 \in \mathbb{T}(\mathcal{B}, \mathbb{D}_\ell) \setminus \{\bot_t, \top_t\}$, we can check $t_1 \sqsubseteq_t t_2$ by 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 $\underline{\pi_b} \in \mathcal{B}$. If each pair (ℓ_1, ℓ_2) satisfies $\ell_1 \sqsubseteq_\ell \ell_2$, we can conclude that $t_1 \sqsubseteq_t t_2$; otherwise, we have $t_1 \not\sqsubseteq_t t_2$.

```
include(t1, t2 : binary decision trees)
{
    if (t1 == ⦇l1⦈ && t2 == ⦇l2⦈) then return t1 $\sqsubseteq_\ell$ t2;

    let t1 = ⟦B: t1l, t1r⟧ and t2 = ⟦B: t2l, t2r⟧;
    return include(t1l, t2l) & include(t1r, t2r);
}
```

Example 5. We have $[\![\mathrm{x} \leq 50 : (\!| \mathrm{x} = 0 \wedge \mathrm{y} = 0 |\!), (\!| \bot_\ell |\!)]\!] \sqsubseteq [\![\mathrm{x} \leq 50 : (\!| 0 \leq \mathrm{x} \leq 1 \wedge \mathrm{x} = \mathrm{y} |\!), (\!| \bot_\ell |\!)]\!]$ and $[\![\mathrm{x} \leq 50 : (\!| \mathrm{x} = 0 \wedge \mathrm{y} = 0 |\!), (\!| \bot_\ell |\!)]\!] \not\sqsubseteq [\![\mathrm{x} \leq 50 : (\!| \mathrm{x} = 1 \wedge \mathrm{y} = 1 |\!), (\!| \bot_\ell |\!)]\!]$.

The equality of t_1 and t_2 can be tested by the fact $t_1 =_t t_2 \triangleq t_1 \sqsubseteq_t t_2 \wedge t_2 \sqsubseteq_t t_1$. When the leaf abstract domain $\mathbb{D}_\ell$ has $=_\ell$, we can also check the equality 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 $\underline{\pi_b} \in \mathcal{B}$.

Meet and Join. Given two binary decision tree $t_1, t_2 \in \mathbb{T}(\mathcal{B}, \mathbb{D}_\ell)$, the meet $t = t_1 \sqcap_t t_2$ can be computed using the meet $\sqcap_\ell$ in the leaf abstract domain $\mathbb{D}_\ell$. Let ℓ_1, ℓ_2 are leaves of t_1, t_2 respectively, where the same branch condition path $\underline{\pi_b} \in \mathcal{B}$ leads to ℓ_1 and ℓ_2, then $\ell = \ell_1 \sqcap_\ell \ell_2$ is the leaf of t led by the same branch condition path $\underline{\pi_b} \in \mathcal{B}$. After computing each leaf $\ell = \ell_1 \sqcap_\ell \ell_2$ in t, we then get $t = t_1 \sqcap_t t_2$.

```
meet(t1, t2 : binary decision trees)
{
    if (t1 == $(\!|$l1$|\!)$ && t2 == $(\!|$l2$|\!)$) then return t1 $\sqcap_\ell$ t2;

    let t1 = $[\![$B: t1l, t1r$]\!]$ and t2 = $[\![$B: t2l, t2r$]\!]$;
    return $[\![$B: meet(t1l, t2l), meet(t1r, t2r)$]\!]$;
}
```

Similar to the meet, we can compute the join $t = t_1 \sqcup_t t_2$ using the join $\sqcup_\ell$ in the leaf abstract domain $\mathbb{D}_\ell$. But instead of computing the join $\ell_1 \sqcup_\ell \ell_2$ for each pair (ℓ_1, ℓ_2) of leaves in (t_1, t_2) where ℓ_1 and ℓ_2 are led by the same branch condition path $\underline{\pi_b} \in \mathcal{B}$, we also use the branch conditions in $\underline{\pi_b}$ as bound to prevent precision loss. Let $\underline{\pi_b} = \beta_1 \cdot \beta_2 \cdot ... \cdot \beta_n$ where $\beta_i = \mathrm{B}_i$ or $\neg \mathrm{B}_i, i = 1, ..., n$, we have $\ell = (\ell_1 \sqcup_\ell \ell_2) \sqcap_\ell \mathbb{D}_\ell(\beta_1) \sqcap_\ell \mathbb{D}_\ell(\beta_2) \sqcap_\ell ... \sqcap_\ell \mathbb{D}_\ell(\beta_n)$ ($\mathbb{D}_\ell(\beta)$ means 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).

```
join(t1, t2 : binary decision trees, bound = $\top$)
{
    if (t1 == $(\!|$l1$|\!)$ && t2 == $(\!|$l2$|\!)$) then return (t1 $\sqcup_\ell$ t2) $\sqcap_\ell$ bound;

    let t1 = $[\![$B: t1l, t1r$]\!]$ and t2 = $[\![$B: t2l, t2r$]\!]$;
    return $[\![$B: join(t1l, t2l, bound $\sqcap_\ell$ $\mathbb{D}_\ell$(B)),
        join(t1r, t2r, bound $\sqcap_\ell$ $\mathbb{D}_\ell$($\neg$B))$]\!]$;
}
```

Example 6. Let $t_1 = [\![\mathrm{x} \leq 50 : (\!|\, \mathrm{x} = 0 \wedge \mathrm{y} = 0 \,|\!), (\!|\, \bot_\ell \,|\!)]\!], t_2 = [\![\mathrm{x} \leq 50 : (\!|\, \mathrm{x} = 1 \wedge \mathrm{y} = 1 \,|\!), (\!|\, \bot_\ell \,|\!)]\!], t_3 = [\![\mathrm{x} \leq 50 : (\!|\, 0 \leq \mathrm{x} \leq 1 \wedge \mathrm{x} = \mathrm{y} \,|\!), (\!|\, \bot_\ell \,|\!)]\!]$, we have $t_1 \sqcap_t t_2 = \bot_t, t_1 \sqcap_t t_3 = t_1$ and $t_1 \sqcup_t t_2 = t_3, t_2 \sqcap_t t_3 = t_3$.

4.3 Transfer Functions

We define transfer functions for both tests and assignments. The tests either occur in a loop head or occur in the branch. Hence, we define both loop test transfer function and branch test abstract function for the binary decision tree abstract domain.

Loop Test Transfer Function. The transfer function for the loop tests is simple. Given a binary decision tree $t \in \mathbb{T}(\mathcal{B}, \mathbb{D}_\ell)$ and a loop test B, we first define $t \sqcap_t \mathrm{B}$ as:

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

Then the transfer function $f_L[\![\mathrm{B}]\!]t$ for the loop test B of the binary decision tree t can be defined as:

$$f_L[\![\mathrm{B}]\!]t \;\triangleq\; t \sqcap_t \mathrm{B}.$$

Example 7. Let t be the binary decision tree in Example 4, then $f_L[\![\mathrm{y} >= 0]\!]t = [\![\mathrm{x} \leq 50 : (\!|\, 0 \leq \mathrm{x} \leq 50 \wedge \mathrm{x} = \mathrm{y} \,|\!), (\!|\, 51 \leq \mathrm{x} \leq 102 \wedge \mathrm{x} + \mathrm{y} - 102 = 0 \,|\!)]\!]$.

Branch Test Transfer Function. The binary decision tree can be constructed in two different ways. On one hand, it can be generated immediately after the set $\mathcal{B}$ of branch condition paths has been generated in the pre-analysis. In this way, all leaves of the binary decision tree will be set to $\top_\ell$ for the first program point and $\bot_\ell$ for others ($\top_\ell, \bot_\ell \in \mathbb{D}_\ell$) at the beginning. On the other hand, both binary decision tree and $\mathcal{B}$ can be constructed on the fly during the static analysis. In this last case, we have $\mathcal{B} = \emptyset$ and the binary decision tree $t = (\!|\, \top_\ell \,|\!)$ for the first program point and $t = (\!|\, \bot_\ell \,|\!)$ for others at the beginning.

In the latter case, the branch test transfer function should first construct the new binary decision tree from the old one by splitting on the branch condition when it has been first met in the analysis. Given a binary decision condition $t \in \mathbb{T}(\mathcal{B}, \mathbb{D}_\ell)$ and a branch test B that's been first met, there are two situations. One situation is that the branch condition B is independent, that is, it does not occur inside any scope of a branch. In this situation, the new binary decision tree

t' can be constructed by replacing each leaf p in the binary tree t with a subtree $[\![\mathrm{B} : (\!| p \sqcap_\ell \mathbb{D}_\ell(\mathrm{B}) |\!), (\!| p \sqcap_\ell \mathbb{D}_\ell(\neg\mathrm{B}) |\!)]\!]$. We also have $\mathcal{B}' = \{\underline{\pi_b} \cdot \mathrm{B} \mid \underline{\pi_b} \in \mathcal{B}\} \cup \{\underline{\pi_b} \cdot \neg\mathrm{B} \mid \underline{\pi_b} \in \mathcal{B}\}$. The other situation is that the branch condition B is inside a scope of a branch. Let B' be the condition of the branch and there is no other branch scope between B and B', if B is inside the true branch of B', then the new binary decision tree t' can be constructed by replacing each left leaf p of B' in the binary tree t with a subtree $[\![\mathrm{B} : (\!| p \sqcap_\ell \mathbb{D}_\ell(\mathrm{B}) |\!), (\!| p \sqcap_\ell \mathbb{D}_\ell(\neg\mathrm{B}) |\!)]\!]$. We also have $\mathcal{B}' = \{\underline{\pi_b} \cdot \mathrm{B}' \cdot \mathrm{B} \mid \underline{\pi_b} \cdot \mathrm{B}' \in \mathcal{B}\} \cup \{\underline{\pi_b} \cdot \mathrm{B}' \cdot \neg\mathrm{B} \mid \underline{\pi_b} \cdot \mathrm{B}' \in \mathcal{B}\} \cup (\mathcal{B} \setminus \{\underline{\pi_b} \cdot \mathrm{B}' \mid \underline{\pi_b} \cdot \mathrm{B}' \in \mathcal{B}\})$. If B is inside the false branch of B', the right leaves of B' instead of left leaves should be replaced by the same subtrees and $\mathcal{B}' = \{\underline{\pi_b} \cdot \neg\mathrm{B}' \cdot \mathrm{B} \mid \underline{\pi_b} \cdot \neg\mathrm{B}' \in \mathcal{B}\} \cup \{\underline{\pi_b} \cdot \neg\mathrm{B}' \cdot \neg\mathrm{B} \mid \underline{\pi_b} \cdot \neg\mathrm{B}' \in \mathcal{B}\} \cup (\mathcal{B} \setminus \{\underline{\pi_b} \cdot \neg\mathrm{B}' \mid \underline{\pi_b} \cdot \neg\mathrm{B}' \in \mathcal{B}\})$.

Then in both ways, the branch test transfer function will do the same thing as loop test transfer function. Given the branch test B and the binary decision tree $t \in \mathbb{T}(\mathcal{B}, \mathbb{D}_\ell)$, we have:

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

Example 8. Let t be the binary decision tree in Example 4, then $f_B[\![\mathrm{x} <= 50]\!]t = [\![\mathrm{x} \leq 50 : (\!| 0 \leq \mathrm{x} \leq 50 \wedge \mathrm{x} = \mathrm{y} |\!), (\!| \perp_\ell |\!)]\!]$.

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 $\mathbb{D}_\ell$. E.g., let $t = [\![x \leq 50 : (\!| 0 \leq x \leq 50 |\!), (\!| \perp_\ell |\!)]\!]$ and given an assignment $x = x + 1$, after performing the assignment transfer function of Polyhedra abstract domain on each leaf of t, we will get $t' = [\![x \leq 50 : (\!| 1 \leq x \leq 51 |\!), (\!| \perp_\ell |\!)]\!]$. Generally, the branch condition paths in $\mathcal{B}$ are used as labels separating the abstract properties in disjunctions which are gathered in the leaves. But this is not always the case. For example, in the join operator, we use the branch conditions in $\mathcal{B}$ to reduce the result of the join. After performing the assignment transfer function of leaf abstract domain $\mathbb{D}_\ell$ on each leaf, we may also need to manipulate the leaves using the branch condition paths in $\mathcal{B}$. Let's check the above result t' after the assignment, it appears that some leaves in the new binary decision tree may not satisfy some branch conditions in the branch condition paths which are leading to them. For example, $1 \leq x \leq 51$ is not satisfying the branch condition $x \leq 50$. We know the violation part is actually satisfying the negation of those branch conditions. Hence we need to use the branch condition $x \leq 50$ to separate $1 \leq x \leq 51$ into $1 \leq x \leq 50 \vee x = 51$ and update the corresponding leaves. For example, we have $t'' = [\![x \leq 50 : (\!| 1 \leq x \leq 50 |\!), (\!| x = 51 |\!)]\!]$.

We call this procedure *reconstruction on leaves*. Given a binary decision tree t after an assignment, we define the procedure as follow:

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 $p'_1 \sqcup_\ell p'_2 \sqcup_\ell ... \sqcup_\ell p'_n$.

Correctness. Let $p = p_1 \vee p_2 \vee ... \vee p_n$ be the disjunction of all properties in leaves before reconstruction on leaves. For each leaf ℓ_i in t, we have $\ell_i = (p_1 \sqcap_\ell (\mathbb{D}_\ell(\beta_1^i \wedge \beta_2^i \wedge ... \wedge \beta_n^i))) \sqcup_\ell ... \sqcup_\ell (p_n \sqcap_\ell (\mathbb{D}_\ell(\beta_1^i \wedge \beta_2^i \wedge ... \wedge \beta_n^i))) = (p_1 \sqcup_\ell ... \sqcup_\ell p_n) \sqcap_\ell (\mathbb{D}_\ell(\beta_1^i \wedge \beta_2^i \wedge ... \wedge \beta_n^i))$ after reconstruction on leaves. We then have the disjunction of all properties in leaves after reconstruction on leaves is $p' = \ell_1 \vee ... \vee \ell_n = (p_1 \sqcup_\ell ... \sqcup_\ell p_n) \sqcap_\ell (\mathbb{D}_\ell(\beta_1^1 \wedge \beta_2^1 \wedge ... \wedge \beta_n^1)) \vee ... \vee (p_1 \sqcup_\ell ... \sqcup_\ell p_n) \sqcap_\ell (\mathbb{D}_\ell(\beta_1^n \wedge \beta_2^n \wedge ... \wedge \beta_n^n)) = (p_1 \sqcup_\ell ... \sqcup_\ell p_n) \sqcap_\ell ((\mathbb{D}_\ell(\beta_1^1 \wedge \beta_2^1 \wedge ... \wedge \beta_n^1)) \vee ... \vee (\mathbb{D}_\ell(\beta_1^n \wedge \beta_2^n \wedge ... \wedge \beta_n^n))) = (p_1 \sqcup_\ell ... \sqcup_\ell p_n) \sqcap_\ell true = p_1 \sqcup_\ell ... \sqcup_\ell p_n \equiv p$. This shows that the reconstruction on leaves procedure will not change the result of the assignment transfer function.

4.4 Extrapolation Operators

When the leaf abstract domain $\mathbb{D}_\ell$ has strictly increasing and/or strictly decreasing infinite chains, widening and/or narrowing operators are required in the binary decision tree abstract domain to accelerate the convergence of fixpoint iterates.

Widening. Given two binary decision tree $t_1, t_2 \in \mathbb{T}(\mathcal{B}, \mathbb{D}_\ell)$, the widening $t = t_1 \triangledown_t t_2$ can be computed using the widening $\triangledown_\ell$ in the leaf abstract domain $\mathbb{D}_\ell$ similar to the join operator, that is, computing the widening $\ell_1 \triangledown_\ell \ell_2$ for each pair (ℓ_1, ℓ_2) of leaves in (t_1, t_2) where ℓ_1 and ℓ_2 are led by the same branch condition path $\underline{\pi_b} \in \mathcal{B}$ while the branch conditions in $\underline{\pi_b}$ are also used as the threshold. Let $\underline{\pi_b} = \beta_1 \cdot \beta_2 \cdot ... \cdot \beta_n$ where $\beta_i = \mathrm{B}_i$ or $\neg\mathrm{B}_i, i = 1, ..., n$, we have each leaf $\ell = (\ell_1 \triangledown_\ell \ell_2) \sqcap_\ell \mathbb{D}_\ell(\beta_1) \sqcap_\ell \mathbb{D}_\ell(\beta_2) \sqcap_\ell ... \sqcap_\ell \mathbb{D}_\ell(\beta_n)$.

```
widening(t1, t2 : binary decision trees, bound = $\top$)
{
    if (t1 == $(\!|$l1$|\!)$ && t2 == $(\!|$l2$|\!)$) then return (t1 $\triangledown_\ell$ t2) $\sqcap_\ell$ bound;

    let t1 = $[\![$B: t1l, t1r$]\!]$ and t2 = $[\![$B: t2l, t2r$]\!]$;
    return $[\![$B: widening(t1l, t2l, bound $\sqcap_\ell$ $\mathbb{D}_\ell$(B)),
        widening(t1r, t2r, bound $\sqcap_\ell$ $\mathbb{D}_\ell$($\neg$B))$]\!]$;
}
```

Narrowing. The narrowing operator in the binary decision tree abstract domain is very similar to its meet operator. Given two binary decision tree $t_1, t_2 \in \mathbb{T}(\mathcal{B}, \mathbb{D}_\ell)$, the narrowing $t = t_1 \vartriangle_t t_2$ can be computed using the narrowing $\vartriangle_\ell$ in the leaf abstract domain $\mathbb{D}_\ell$. Let ℓ_1, ℓ_2 are leaves of t_1, t_2 respectively, where the same branch condition path $\underline{\pi_b} \in \mathcal{B}$ leads to ℓ_1 and ℓ_2, then $\ell = \ell_1 \vartriangle_\ell \ell_2$ is the leaf of t led by the same branch condition path $\underline{\pi_b} \in \mathcal{B}$. After computing each leaf $\ell = \ell_1 \vartriangle_\ell \ell_2$ in t, we then get $t = t_1 \vartriangle_t t_2$.

```
narrowing(t1, t2 : binary decision trees)
{
    if (t1 == $\llparenthesis$l1$\rrparenthesis$ && t2 == $\llparenthesis$l2$\rrparenthesis$) then return t1 $\triangle_\ell$ t2;

    let t1 = $\llbracket$B: t1l, t1r$\rrbracket$ and t2 = $\llbracket$B: t2l, t2r$\rrbracket$;
    return $\llbracket$B: narrowing(t1l, t2l), narrowing(t1r, t2r)$\rrbracket$;
}
```

Example 9. Let $t_1 = \llbracket \mathrm{x} \leq 50 : \llparenthesis\, \mathrm{x} = 0 \wedge \mathrm{y} = 0 \,\rrparenthesis, \llparenthesis\, \bot_\ell \,\rrparenthesis \rrbracket$ and $t_2 = \llbracket \mathrm{x} \leq 50 : \llparenthesis\, \mathrm{x} = \mathrm{y} \wedge 0 \leq \mathrm{x} \leq 1 \,\rrparenthesis, \llparenthesis\, \bot_\ell \,\rrparenthesis \rrbracket$. It's easy to see that $t_1 \subseteq t_2$. In polyhedra, we have $(\mathrm{x} = 0 \wedge \mathrm{y} = 0) \triangledown_t (\mathrm{x} = \mathrm{y} \wedge 0 \leq \mathrm{x} \leq 1) = \mathrm{x} \geq 0 \wedge \mathrm{x} = \mathrm{y}$. Hence, we have $t_1 \triangledown_t t_2 = \llbracket \mathrm{x} \leq 50 : \llparenthesis\, 0 \leq \mathrm{x} \leq 50 \wedge \mathrm{x} = \mathrm{y} \,\rrparenthesis, \llparenthesis\, \bot_\ell \,\rrparenthesis \rrbracket$.

4.5 Other Operators

Although the number of branch conditions in a program is always finite, it may still be a very large number. A large number of branch conditions means a large binary decision tree, with a potentially exponential growth which is not acceptable in practice. Hence, we need limit the size (depth) of the binary decision trees.

One method is to eliminate decision nodes by merging their subtrees when the binary decision tree grows too deep. This can be done as follow:

1. Pick up a branch condition B. We can simply use the one in the root, or the nearest one to the leaves, or by random. We can also design a ranking function based on the information from the analysis for each branch condition to estimate how likely it is to be eliminated with minimal information loss. Then we always choose the most likely one.
2. Eliminate B (B or ¬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 $t'_t \sqcup_t t'_f$.

Another method is to generate a smaller $\mathcal{B}$ by abstracting the branch condition paths in $\mathcal{B}$ into shorter ones. We may partition the set of branch conditions by its appearance inside or outside loops and then only keep the ones appeared inside the loops in $\mathcal{B}$. We may also only keep the branch conditions which have some particular form, such as $ax \leqslant b$, etc.

The second method is different from the first one because it can be done in the pre-analysis or on the fly before splitting trees, thus no merging is needed during the analysis. This reduces the cost of the analysis, thus improves its efficiency. But because all the branch conditions being eliminated are not based on the information that is collected during the static analysis, the result may be less precise than the one generated from the first method. Moreover, eliminating branch conditions and merging their subtrees allow us to dynamically change the binary decision trees on the fly. This provides a more flexible way of adjusting the cost/precision ratio of the static analysis.

5 Example

Let us come back to Example 1. We choose the polyhedra abstract domain as the leaf abstract domain and we have $\mathcal{B} = \{\mathrm{x} <= 50, \neg(\mathrm{x} <= 50)\}$. Initially, we set $t = \llparenthesis \bot_\ell \rrparenthesis$ in the program point ^{l}. After the assignment "x = 0; y = 0;", we have "$t = \llparenthesis \mathrm{x} = 0 \wedge \mathrm{y} = 0 \rrparenthesis$". Let t_i be the abstract property at program point ^{l} after the i-th iteration, then $t_0 = \llparenthesis \mathrm{x} = 0 \wedge y = \mathrm{y} \rrparenthesis$. In the first iteration, we have to construct the binary decision tree when first reaching the branch test "x <= 50". In this case, we have $t'_0 = \llbracket \mathrm{x} \leq 50 : \llparenthesis \mathrm{x} = 0 \wedge \mathrm{y} = 0 \rrparenthesis, \llparenthesis \bot_\ell \rrparenthesis \rrbracket$. At the end of the first iteration, we get $t''_0 = \llbracket \mathrm{x} \leq 50 : \llparenthesis \mathrm{x} = 1 \wedge \mathrm{y} = 1 \rrparenthesis, \llparenthesis \bot_\ell \rrparenthesis \rrbracket$. Then $t_1 = t_0 \cup_t t''_0 = \llbracket \mathrm{x} \leq 50 : \llparenthesis \mathrm{x} = \mathrm{y} \wedge 0 \leq \mathrm{x} \leq 1 \rrparenthesis, \llparenthesis \bot_\ell \rrparenthesis \rrbracket$. Afterwards, we apply the widening and get $t'_1 = t_0 \triangledown t_1 = \llbracket \mathrm{x} \leq 50 : \llparenthesis 0 \leq \mathrm{x} \leq 50 \wedge \mathrm{x} = \mathrm{y} \rrparenthesis, \llparenthesis \bot_\ell \rrparenthesis \rrbracket$. In the second iteration, the assignment "x++;" leads to reconstruction on leaves, hence we get $t''_1 = \llbracket \mathrm{x} \leq 50 : \llparenthesis 1 \leq \mathrm{x} \leq 50 \wedge \mathrm{x} = \mathrm{y} \rrparenthesis, \llparenthesis x = 51 \wedge y = 51 \rrparenthesis \rrbracket$. Then $t_2 = t_1 \cup_t t''_1 = \llbracket \mathrm{x} \leq 50 : \llparenthesis 0 \leq \mathrm{x} \leq 50 \wedge \mathrm{x} = \mathrm{y} \rrparenthesis, \llparenthesis \mathrm{x} = 51 \wedge \mathrm{y} = 51 \rrparenthesis \rrbracket$. After the third iteration, $t_3 = \llbracket \mathrm{x} \leq 50 : \llparenthesis 0 \leq \mathrm{x} \leq 50 \wedge \mathrm{x} = \mathrm{y} \rrparenthesis, \llparenthesis \mathrm{x} + \mathrm{y} - 102 = 0 \wedge 51 \leq \mathrm{x} \leq 52 \rrparenthesis \rrbracket$. We then apply the widening and get $t'_3 = t_2 \triangledown t_3 = \llbracket \mathrm{x} \leq 50 : \llparenthesis 0 \leq \mathrm{x} \leq 50 \wedge \mathrm{x} = \mathrm{y} \rrparenthesis, \llparenthesis \mathrm{x} + \mathrm{y} - 102 = 0 \wedge \mathrm{x} \geq 51 \rrparenthesis \rrbracket$. One more iteration yields $t_4 = \llbracket \mathrm{x} \leq 50 : \llparenthesis 0 \leq \mathrm{x} \leq 50 \wedge \mathrm{x} = \mathrm{y} \rrparenthesis, \llparenthesis \mathrm{x} + \mathrm{y} - 102 = 0 \wedge 51 \leq \mathrm{x} \leq 103 \rrparenthesis \rrbracket$. It follows that the program analysis converges. Hence t_4 is the invariant at program point ^{l}.

6 Related Work

A systematic characterization of the least bases for the disjunctive completion of abstract domains can be found in [8]. The trace partitioning using control flows was first introduced in [3]. A static analysis framework via trace partitioning was proposed by [11]. In this framework, the control flow is used to choose which disjunctions to keep but it lacks the merge of partitions, which may lead to exponential cost. In [13], a trace partitioning domain, where the partitioning of traces are based on the history of the control flow, has been proposed. The main difference between their partitionings and ours is we only use (part of) branch conditions while they are considering all conditions and other information.

Decision trees have been used for the disjunctive refinement of an abstract domain such as [10] for the interval abstract domain based on decision trees. A general segmented decision tree abstract domain, where disjunctions are determined by values of variables is introduced in [7]. Moreover, [16] proposed a general disjunctive refinement of an abstract domain based on decision trees extended with linear constraints for program termination. The difference between those works and ours is their partitionings are mainly based on the value of some variables while ours are directly based on the branch conditions.

There also exist several works on directly allowing disjunction in the domain, i.e., powerset domain [1]. In [15], the disjunctions are computed on an elaboration, which can be viewed as a multiply duplication, of the programs CFG structure. Moreover, our binary decision tree abstract domain functor can also be useful to scale traditional path-sensitive program analysis [17].

7 Conclusion

In this paper, we have introduced a series of abstractions which generates a set of branch condition paths. Those branch condition paths define a kind of trace partitioning on the concrete level (trace semantics of program). By using such information for trace partitioning, we proposed a binary decision tree abstract domain functor that allows finite disjunction of abstract properties generated by existing abstract domains[1]. We also discussed the implementation of our binary decision tree abstract domain functor by providing algorithms for inclusion test, meet and join, transfer functions and extrapolation operators. Although we bound the number of disjunctions only to the number of branch conditions in the program, the cost of our domain may still be excessive. Thus we also discussed how to limit the number of disjunctions. Our binary decision tree abstract domain functor may provide a flexible way of adjusting the cost/precision ratio for static analysis.

References

1. Bagnara, R., Hill, P.M., Zaffanella, E.: Widening operators for powerset domains. STTT **9**(3–4), 413–414 (2007)
2. Chen, J.: SMT-based and disjunctive relational abstract domains for static analysis. Ph.D. thesis, New York University (May 2015)
3. Cousot, P.: Semantic foundations of program analysis. In: Muchnick, S., Jones, N. (eds.) Program Flow Analysis: Theory and Applications, pp. 303–342. Prentice-Hall Inc., Englewood Cliffs (1981). Chapter. 10
4. Cousot, P., Cousot, R.: Static determination of dynamic properties of programs. In: Proceedings of the Second International Symposium on Programming, pp. 106–130. Dunod, Paris, France (1976)
5. Cousot, P., Cousot, R.: Abstract interpretation frameworks. J. Logic. Comput. **2**(4), 511–547 (1992)
6. Cousot, P., Halbwachs, N.: Automatic discovery of linear restraints among variables of a program. In: Conference Record of the Fifth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, pp. 84–97. ACM Press, New York, NY, Tucson, Arizona (1978)

[1] Note that all examples in this paper are numerical, but this does not mean that our abstract domain functor is limited to numerical abstract domains. Symbolic abstract domains are also allowed.

7. Cousot, P., Cousot, R., Mauborgne, L.: A scalable segmented decision tree abstract domain. In: Manna, Z., Peled, D.A. (eds.) Time for Verification. LNCS, vol. 6200, pp. 72–95. Springer, Heidelberg (2010)
8. Giacobazzi, R., Ranzato, F.: Optimal domains for disjunctive abstract intepretation. Sci. Comput. Program. **32**(1–3), 177–210 (1998)
9. Gopan, D., Reps, T.: Guided static analysis. In: Riis Nielson, H., Filé, G. (eds.) SAS 2007. LNCS, vol. 4634, pp. 349–365. Springer, Heidelberg (2007)
10. Gurfinkel, A., Chaki, S.: Boxes: a symbolic abstract domain of boxes. In: Cousot, R., Martel, M. (eds.) SAS 2010. LNCS, vol. 6337, pp. 287–303. Springer, Heidelberg (2010)
11. Handjieva, M., Tzolovski, S.: Refining static analyses by trace-based partitioning using control flow. In: Levi, G. (ed.) SAS 1998. LNCS, vol. 1503, pp. 200–214. Springer, Heidelberg (1998)
12. Jeannet, B., Miné, A.: Apron: a library of numerical abstract domains for static analysis. In: Bouajjani, A., Maler, O. (eds.) CAV 2009. LNCS, vol. 5643, pp. 661–667. Springer, Heidelberg (2009)
13. Mauborgne, L., Rival, X.: Trace partitioning in abstract interpretation based static analyzers. In: Sagiv, M. (ed.) ESOP 2005. LNCS, vol. 3444, pp. 5–20. Springer, Heidelberg (2005)
14. Miné, A.: The octagon abstract domain. High. Ord. Symbolic Comput. (HOSC) **19**(1), 31–100 (2006). http://www.di.ens.fr/ mine/publi/article-mine-HOSC06.pdf
15. Sankaranarayanan, S., Ivančić, F., Shlyakhter, I., Gupta, A.: Static analysis in disjunctive numerical domains. In: Yi, K. (ed.) SAS 2006. LNCS, vol. 4134, pp. 3–17. Springer, Heidelberg (2006)
16. Urban, C., Miné, A.: A decision tree abstract domain for proving conditional termination. In: Müller-Olm, M., Seidl, H. (eds.) SAS 2014. LNCS, vol. 8723, pp. 302–318. Springer, Heidelberg (2014)
17. Winter, K., Zhang, C., Hayes, I.J., Keynes, N., Cifuentes, C., Li, L.: Path-sensitive data flow analysis simplified. In: Groves, L., Sun, J. (eds.) ICFEM 2013. LNCS, vol. 8144, pp. 415–430. Springer, Heidelberg (2013)

15. Urban, C., Miné, A.: A decision tree abstract domain for proving conditional termination. In: Müller-Olm, M., Seidl, H. (eds.) Static Analysis - 21st International Symposium, SAS 2014, Munich, Germany, September 11-13, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8723, pp. 302–318. Springer (2014)
16. Winter, K., Zhang, C., Hayes, I.J., Keynes, N., Cifuentes, C., Li, L.: Path-sensitive data flow analysis simplified. In: Groves, L., Sun, J. (eds.) Formal Methods and Software Engineering - 15th International Conference on Formal Engineering Methods, ICFEM 2013, Queenstown, New Zealand, October 29 - November 1, 2013, Proceedings. Lecture Notes in Computer Science, vol. 8144, pp. 415–430. Springer (2013)

A Trace Semantics

Let $\mathit{\Pi}$ denote a set of traces. We define the trace semantics $\mathcal{S}^t\llbracket \mathrm{C} \rrbracket$ of commands C by induction on the syntactic structure of the commands:

$$
\begin{aligned}
&\mathcal{S}^t\llbracket \mathrm{skip} \rrbracket \triangleq \{\langle \mathrm{skip}, \rho\rangle \xrightarrow{\mathrm{skip}} \langle \mathrm{stop}, \rho\rangle \mid \rho \in \mathcal{E}\}\\
&\mathcal{S}^t\llbracket \mathrm{x} = \mathrm{E} \rrbracket \triangleq \{\langle \mathrm{skip}, \rho\rangle \xrightarrow{\mathrm{x=E}} \langle \mathrm{stop}, \rho[\mathrm{x} := v]\rangle \mid \rho \in \mathcal{E} \wedge v \in \mathcal{E}\llbracket \mathrm{E} \rrbracket\rho\}\\
&\mathcal{S}^t\llbracket \mathrm{C}_1 \,;\, \mathrm{C}_2 \rrbracket \triangleq \{(\pi \fatsemi C_2) \xrightarrow{\mathrm{A}} \langle \mathrm{C}_2, \rho\rangle \xrightarrow{\mathrm{A}'} \pi' \mid \rho \in \mathcal{E} \wedge\\
&\qquad\qquad \pi \xrightarrow{\mathrm{A}} \langle \mathrm{stop}, \rho\rangle \in \mathcal{S}^t\llbracket \mathrm{C}_1 \rrbracket \wedge \langle \mathrm{C}_2, \rho\rangle \xrightarrow{\mathrm{A}'} \pi' \in \mathcal{S}^t\llbracket \mathrm{C}_2 \rrbracket\}\\
&\mathcal{S}^t\llbracket \mathrm{if}\,(\mathrm{B})\;\{\mathrm{C}_1\}\;\mathrm{else}\;\{\mathrm{C}_2\} \rrbracket \triangleq\\
&\qquad \{\langle \mathrm{if}\,(\mathrm{B})\;\{\mathrm{C}_1\}\;\mathrm{else}\;\{\mathrm{C}_2\}, \rho\rangle \xrightarrow{\mathrm{B}} \langle \mathrm{C}_1, \rho\rangle \xrightarrow{\mathrm{A}} \pi \mid\\
&\qquad\qquad \rho \in \mathcal{E} \wedge \mathit{true} \in \mathcal{E}\llbracket \mathrm{B} \rrbracket\rho \wedge \langle \mathrm{C}_1, \rho\rangle \xrightarrow{\mathrm{A}} \pi \in \mathcal{S}^t\llbracket \mathrm{C}_1 \rrbracket\}\\
&\quad \cup \{\langle \mathrm{if}\,(\mathrm{B})\;\{\mathrm{C}_1\}\;\mathrm{else}\;\{\mathrm{C}_2\}, \rho\rangle \xrightarrow{\neg\mathrm{B}} \langle \mathrm{C}_2, \rho\rangle \xrightarrow{\mathrm{A}} \pi \mid\\
&\qquad\qquad \rho \in \mathcal{E} \wedge \mathit{false} \in \mathcal{E}\llbracket \mathrm{B} \rrbracket\rho \wedge \langle \mathrm{C}_2, \rho\rangle \xrightarrow{\mathrm{A}} \pi \in \mathcal{S}^t\llbracket \mathrm{C}_2 \rrbracket\}\\
&\mathcal{F}^{ti}\llbracket \mathrm{while}\,(\mathrm{B})\;\{\mathrm{C}\} \rrbracket X \triangleq \{\langle \mathrm{while}\,(\mathrm{B})\;\{\mathrm{C}\}, \rho\rangle \mid \rho \in \mathcal{E}\} \cup\\
&\quad \{\pi \xrightarrow{\mathrm{A}} \langle \mathrm{while}\,(\mathrm{B})\;\{\mathrm{C}\}, \rho\rangle \xrightarrow{\mathrm{B}} (\langle \mathrm{C}, \rho\rangle \xrightarrow{\mathrm{A}'} \pi' \xrightarrow{\mathrm{A}''} \langle \mathrm{stop}, \rho'\rangle) \fatsemi \mathrm{while}\,(\mathrm{B})\;\{\mathrm{C}\} \mid\\
&\qquad \pi, \pi' \in \mathit{\Pi} \wedge \pi \xrightarrow{\mathrm{A}} \langle \mathrm{while}\,(\mathrm{B})\;\{\mathrm{C}\}, \rho\rangle \in X \wedge \mathit{true} \in \mathcal{E}\llbracket \mathrm{B} \rrbracket\rho \wedge\\
&\qquad (\langle \mathrm{C}, \rho\rangle \xrightarrow{\mathrm{A}'} \pi' \xrightarrow{\mathrm{A}''} \langle \mathrm{stop}, \rho'\rangle) \in \mathcal{S}^t\llbracket \mathrm{C} \rrbracket\}\\
&\mathcal{S}^{ti}\llbracket \mathrm{while}\,(\mathrm{B})\;\{\mathrm{C}\} \rrbracket \triangleq \mathrm{lfp}^{\subseteq} \mathcal{F}^{ti}\llbracket \mathrm{while}\,(\mathrm{B})\;\{\mathrm{C}\} \rrbracket\\
&\mathcal{S}^t\llbracket \mathrm{while}\,(\mathrm{B})\;\{\mathrm{C}\} \rrbracket \triangleq \{\pi \xrightarrow{\mathrm{A}} \langle \mathrm{while}\,(\mathrm{B})\;\{\mathrm{C}\}, \rho\rangle \xrightarrow{\neg\mathrm{B}} \langle \mathrm{stop}, \rho\rangle \mid\\
&\pi \in \mathit{\Pi} \wedge \pi \xrightarrow{\mathrm{A}} \langle \mathrm{while}\,(\mathrm{B})\;\{\mathrm{C}\}, \rho\rangle \in \mathcal{S}^{ti}\llbracket \mathrm{while}\,(\mathrm{B})\;\{\mathrm{C}\} \rrbracket \wedge \mathit{false} \in \mathcal{E}\llbracket \mathrm{B} \rrbracket\rho\}
\end{aligned}
$$

B Proof for Theorem 1

The proof is by structural induction on the syntax of the command C.

- **skip command**

$$
\begin{aligned}
&\alpha^a(\mathcal{S}^t\llbracket \mathrm{skip} \rrbracket)\\
=\ &\alpha^a(\{\langle \mathrm{skip}, \rho\rangle \xrightarrow{\mathrm{skip}} \langle \mathrm{skip}, \rho\rangle \mid \rho \in \mathcal{E}\}) && \wr \text{definition of } \mathcal{S}^t\llbracket \mathrm{skip} \rrbracket \int\\
=\ &\{\mathrm{skip}\} && \wr \text{definition of } \alpha^a \int\\
\triangleq\ &\mathcal{G}^a\llbracket \circ\!\!\rightarrow \boxed{\mathrm{skip}} \rightarrow\!\!\circ \rrbracket && \wr \text{definition of } \mathcal{G}^a\llbracket \circ\!\!\rightarrow \boxed{\mathrm{skip}} \rightarrow\!\!\circ \rrbracket \int
\end{aligned}
$$

- **Assignment**

$$\alpha^a(\mathcal{S}^t[\![\mathrm{x} = \mathrm{E}]\!])$$

$$= \alpha^a(\{\langle \mathrm{skip}, \rho\rangle \xrightarrow{\mathrm{x=E}} \langle \mathrm{stop}, \rho[\mathrm{x} := v]\rangle \mid \rho \in \mathcal{E} \wedge v \in \mathcal{E}[\![\mathrm{E}]\!]\rho\})$$

$\wr$definition of $\mathcal{S}^t[\![\mathrm{x} = \mathrm{E}]\!]$$\wr$

$$= \{\mathrm{x} = \mathrm{E}\}$$

$\wr$definition of $\alpha^a$$\wr$

$$\triangleq \mathcal{G}^a[\![\circ\!\!\rightarrow \boxed{\mathrm{x} := \mathrm{E}} \!\!\rightarrow\!\circ]\!]$$

$\wr$definition of $\mathcal{G}^a[\![\circ\!\!\rightarrow \boxed{\mathrm{x} := \mathrm{E}} \!\!\rightarrow\!\circ]\!]$$\wr$

- **Conditional**

$$\alpha^a(\mathcal{S}^t[\![\mathrm{if\ (B)\ \{C_1\}\ else\ \{C_2\}}]\!])$$

$$= \alpha^a(\{\langle \mathrm{if\ (B)\ \{C_1\}\ else\ \{C_2\}}, \rho\rangle \xrightarrow{\mathrm{B}} \langle \mathrm{C}_1, \rho\rangle \xrightarrow{\mathrm{A}} \pi \mid \rho \in \mathcal{E} \wedge \mathit{true} \in \mathcal{E}[\![\mathrm{B}]\!]\rho \wedge \langle \mathrm{C}_1, \rho\rangle \xrightarrow{\mathrm{A}} \pi \in \mathcal{S}^t[\![\mathrm{C}_1]\!]\} \cup \{\langle \mathrm{if\ (B)\ \{C_1\}\ else\ \{C_2\}}, \rho\rangle \xrightarrow{\neg\mathrm{B}} \langle \mathrm{C}_2, \rho\rangle \xrightarrow{\mathrm{A}} \pi \mid \rho \in \mathcal{E} \wedge \mathit{false} \in \mathcal{E}[\![\mathrm{B}]\!]\rho \wedge \langle \mathrm{C}_2, \rho\rangle \xrightarrow{\mathrm{A}} \pi \in \mathcal{S}^t[\![\mathrm{C}_2]\!]\})$$

$\wr$definition of $\mathcal{S}^t[\![\mathrm{if\ (B)\ \{C_1\}\ else\ \{C_2\}}]\!]$$\wr$

$$= \alpha^a(\{\langle \mathrm{if\ (B)\ \{C_1\}\ else\ \{C_2\}}, \rho\rangle \xrightarrow{\mathrm{B}} \langle \mathrm{C}_1, \rho\rangle \xrightarrow{\mathrm{A}} \pi \mid \rho \in \mathcal{E} \wedge \mathit{true} \in \mathcal{E}[\![\mathrm{B}]\!]\rho \wedge \langle \mathrm{C}_1, \rho\rangle \xrightarrow{\mathrm{A}} \pi \in \mathcal{S}^t[\![\mathrm{C}_1]\!]\}) \cup \alpha^a(\{\langle \mathrm{if\ (B)\ \{C_1\}\ else\ \{C_2\}}, \rho\rangle \xrightarrow{\neg\mathrm{B}} \langle \mathrm{C}_2, \rho\rangle \xrightarrow{\mathrm{A}} \pi \mid \rho \in \mathcal{E} \wedge \mathit{false} \in \mathcal{E}[\![\mathrm{B}]\!]\rho \wedge \langle \mathrm{C}_2, \rho\rangle \xrightarrow{\mathrm{A}} \pi \in \mathcal{S}^t[\![\mathrm{C}_2]\!]\})$$

$\wr$since $\alpha^a(S \cup S') = \alpha^a(S) \cup \alpha^a(S')$$\wr$

$$\subseteq \{\mathrm{B} \cdot \alpha^a(\langle \mathrm{C}_1, \rho\rangle \xrightarrow{\mathrm{A}} \pi \mid \langle \mathrm{C}_1, \rho\rangle \xrightarrow{\mathrm{A}} \pi \in \mathcal{S}^t[\![\mathrm{C}_1]\!]\})\} \cup \{\neg\mathrm{B} \cdot \alpha^a(\langle \mathrm{C}_2, \rho\rangle \xrightarrow{\mathrm{A}} \pi \mid \langle \mathrm{C}_2, \rho\rangle \xrightarrow{\mathrm{A}} \pi \in \mathcal{S}^t[\![\mathrm{C}_2]\!]\})\}$$

$\wr$definition of α^a and ignoring the test$\wr$

$$= \{\mathrm{B}\} \cdot \{\alpha^a(\pi') \mid \pi' \in \mathcal{S}^t[\![\mathrm{C}_1]\!]\})\} \cup \{\neg\mathrm{B}\} \cdot \{\alpha^a(\pi'') \mid \pi'' \in \mathcal{S}^t[\![\mathrm{C}_2]\!]\})\}$$

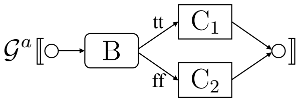
$\wr$by defining the set of action paths concatenation $S \cdot S' \triangleq \{\varpi \cdot \varpi' \mid \varpi \in S \wedge \varpi' \in S'\}$, letting $\pi' = \langle \mathrm{C}_1, \rho\rangle \xrightarrow{\mathrm{A}} \pi$, and $\pi'' = \langle \mathrm{C}_2, \rho\rangle \xrightarrow{\mathrm{A}} \pi$$\wr$

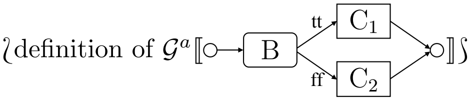
$$= \{\mathrm{B}\} \cdot \alpha^a(\mathcal{S}^t[\![\mathrm{C}_1]\!]) \cup \{\neg\mathrm{B}\} \cdot \alpha^a(\mathcal{S}^t[\![\mathrm{C}_2]\!])$$

$\wr$definition of $\alpha^a$$\wr$

$$\subseteq \{\mathrm{B}\} \cdot \mathcal{G}^a[\![\circ\!\!\rightarrow \boxed{\mathrm{C}_1} \!\!\rightarrow\!\circ]\!] \cup \{\neg\mathrm{B}\} \cdot \mathcal{G}^a[\![\circ\!\!\rightarrow \boxed{\mathrm{C}_2} \!\!\rightarrow\!\circ]\!]$$

$\wr$by structural induction hypothesis and the set of action paths concatenation $\cdot$ is $\subseteq$-increasing in both of its arguments.$\wr$

$\triangleq$ 

$\wr$definition of 

$\wr$

- **Sequence**

$$\alpha^a(\mathcal{S}^t[\![\mathrm{C}_1\ ;\ \mathrm{C}_2]\!])$$

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

$\wr$definition of $\mathcal{S}^t[\![\mathrm{C}_1\ ;\ \mathrm{C}_2]\!]$$\wr$

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

$\wr$since $\alpha^a(\pi \mathbin{;} C) = \alpha^a(\pi)$$\wr$

$$= \{\alpha^a(\pi) \cdot \mathrm{A} \cdot \mathrm{A}' \cdot \alpha^a(\pi') \mid \pi \xrightarrow{\mathrm{A}} \langle \mathrm{stop}, \rho\rangle \in \mathcal{S}^t[\![\mathrm{C}_1]\!] \wedge \langle \mathrm{C}_2, \rho\rangle \xrightarrow{\mathrm{A}'} \pi' \in \mathcal{S}^t[\![\mathrm{C}_2]\!]\}$$

$\wr$since $\alpha^a(\pi \xrightarrow{\mathrm{A}} \pi') = \alpha^a(\pi) \cdot \mathrm{A} \cdot \alpha^a(\pi')$$\wr$

$$= \{\alpha^a(\pi'') \cdot \alpha^a(\pi''') \mid \alpha^a(\pi'') \in \mathcal{S}^t[\![\mathrm{C}_1]\!] \wedge \alpha^a(\pi''') \in \mathcal{S}^t[\![\mathrm{C}_2]\!]\}$$

$\wr$letting $\alpha^a(\pi'') = \pi \xrightarrow{\mathrm{A}} \langle \mathrm{stop}, \rho\rangle$ and $\alpha^a(\pi''') = \langle \mathrm{C}_2, \rho\rangle \xrightarrow{\mathrm{A}'} \pi'$$\wr$

$= \ \alpha^a(\mathcal{S}^t[\![\mathrm{C}_1]\!]) \cdot \alpha^a(\mathcal{S}^t[\![\mathrm{C}_2]\!])$

$\wr$definition of α^a and set of action paths concatenation$\wr$

$\subseteq \ \mathcal{G}^a[\![\circ\!\!\longrightarrow\!\boxed{\mathrm{C}_1}\!\longrightarrow\!\circ]\!] \cdot \mathcal{G}^a[\![\circ\!\!\longrightarrow\!\boxed{\mathrm{C}_2}\!\longrightarrow\!\circ]\!]$

$\wr$by structural induction hypothesis and the set of action paths concatenation $\cdot$ is $\subseteq$-increasing in both of its arguments.$\wr$

$\triangleq \ \mathcal{G}^a[\![\circ\!\!\longrightarrow\!\boxed{\mathrm{C}_1}\!\longrightarrow\!\boxed{\mathrm{C}_2}\!\longrightarrow\!\circ]\!]$ $\wr$definition of $\mathcal{G}^a[\![\circ\!\!\longrightarrow\!\boxed{\mathrm{C}_1}\!\longrightarrow\!\boxed{\mathrm{C}_2}\!\longrightarrow\!\circ]\!]$$\wr$

– **Iteration, transformer**

$\alpha^a(\mathcal{F}^{ti}[\![\mathrm{while\,(B)\,\{C\}}]\!](X))$

$= \ \alpha^a(\{\langle \mathrm{while\,(B)\ \{C\}}, \rho\rangle \mid \rho \in \mathcal{E}\} \cup \{\pi \xrightarrow{\mathrm{A}} \langle \mathrm{while\,(B)\,\{C\}}, \rho\rangle \xrightarrow{\mathrm{B}} (\langle \mathrm{C}, \rho\rangle \xrightarrow{\mathrm{A}'} \pi' \xrightarrow{\mathrm{A}''} \langle \mathrm{stop}, \rho'\rangle) \fatsemi \mathrm{while\,(B)\,\{C\}} \mid \pi, \pi' \in \mathit{\Pi} \wedge \pi \xrightarrow{\mathrm{A}} \langle \mathrm{while\,(B)\,\{C\}}, \rho\rangle \in X \wedge \mathit{true} \in \mathcal{E}[\![\mathrm{B}]\!]\rho \ \wedge (\langle \mathrm{C}, \rho\rangle \xrightarrow{\mathrm{A}'} \pi' \xrightarrow{\mathrm{A}''} \langle \mathrm{stop}, \rho'\rangle) \in \mathcal{S}^t[\![\mathrm{C}]\!]\})$

$\wr$definition of $\mathcal{F}^{ti}[\![\mathrm{while\,(B)\,\{C\}}]\!]$$\wr$

$= \ \alpha^a(\{\langle \mathrm{while\,(B)\ \{C\}}, \rho\rangle \mid \rho \in \mathcal{E}\}) \cup \alpha^a(\{\pi \xrightarrow{\mathrm{A}} \langle \mathrm{while\,(B)\,\{C\}}, \rho\rangle \xrightarrow{\mathrm{B}} (\langle \mathrm{C}, \rho\rangle \xrightarrow{\mathrm{A}'} \pi' \xrightarrow{\mathrm{A}''} \langle \mathrm{stop}, \rho'\rangle) \fatsemi \mathrm{while\,(B)\,\{C\}} \mid \pi, \pi' \in \mathit{\Pi} \wedge \pi \xrightarrow{\mathrm{A}} \langle \mathrm{while\,(B)\,\{C\}}, \rho\rangle \in X \wedge \mathit{true} \in \mathcal{E}[\![\mathrm{B}]\!]\rho \wedge (\langle \mathrm{C}, \rho\rangle \xrightarrow{\mathrm{A}'} \pi' \xrightarrow{\mathrm{A}''} \langle \mathrm{stop}, \rho'\rangle) \in \mathcal{S}^t[\![\mathrm{C}]\!]\})$

$\wr$since $\alpha^a(S \cup S') = \alpha^a(S) \cup \alpha^a(S')$$\wr$

$\subseteq \ \{\varepsilon\} \cup \{\alpha^a(\pi \xrightarrow{\mathrm{A}} \langle \mathrm{while\,(B)\,\{C\}}, \rho\rangle \xrightarrow{\mathrm{B}} (\langle \mathrm{C}, \rho\rangle \xrightarrow{\mathrm{A}'} \pi' \xrightarrow{\mathrm{A}''} \langle \mathrm{stop}, \rho'\rangle) \fatsemi \mathrm{while\,(B)\,\{C\}}) \mid \pi, \pi' \in \mathit{\Pi} \wedge \pi \xrightarrow{\mathrm{A}} \langle \mathrm{while\,(B)\,\{C\}}, \rho\rangle \in X \wedge (\langle \mathrm{C}, \rho\rangle \xrightarrow{\mathrm{A}'} \pi' \xrightarrow{\mathrm{A}''} \langle \mathrm{stop}, \rho'\rangle) \in \mathcal{S}^t[\![\mathrm{C}]\!]\}$

$\wr$definition of α^a and ignoring the result $\mathit{true} \in \mathcal{E}[\![\mathrm{B}]\!]\rho$ of tests$\wr$

$= \ \{\varepsilon\} \cup \{\alpha^a(\pi \xrightarrow{\mathrm{A}} \langle \mathrm{while\,(B)\,\{C\}}, \rho\rangle) \cdot \mathrm{B} \cdot \alpha^a(\langle \mathrm{C}, \rho\rangle \xrightarrow{\mathrm{A}'} \pi' \xrightarrow{\mathrm{A}''} \langle \mathrm{stop}, \rho'\rangle) \mid \pi, \pi' \in \mathit{\Pi} \wedge \pi \xrightarrow{\mathrm{A}} \langle \mathrm{while\,(B)\,\{C\}}, \rho\rangle \in X \wedge (\langle \mathrm{C}, \rho\rangle \xrightarrow{\mathrm{A}'} \pi' \xrightarrow{\mathrm{A}''} \langle \mathrm{stop}, \rho'\rangle) \in \mathcal{S}^t[\![\mathrm{C}]\!]\}$

$\wr$since $\alpha^a(\pi \xrightarrow{\mathrm{A}} \pi') = \alpha^a(\pi) \cdot \mathrm{A} \cdot \alpha^a(\pi')$ and $\alpha^a(\pi \fatsemi C) = \alpha^a(\pi)$$\wr$

$= \ \{\varepsilon\} \cup \{\alpha^a(\pi'') \cdot \mathrm{B} \cdot \alpha^a(\pi''') \mid \pi'' \in X \wedge \pi''' \in \mathcal{S}^t[\![\mathrm{C}]\!]\}$

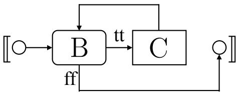
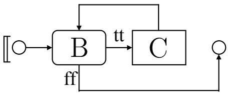
$\wr$letting $\pi'' = \pi \xrightarrow{\mathrm{A}} \langle \mathrm{while\,(B)\,\{C\}}, \rho\rangle$ and $\pi''' = \langle \mathrm{C}, \rho\rangle \xrightarrow{\mathrm{A}'} \pi' \xrightarrow{\mathrm{A}''} \langle \mathrm{stop}, \rho'\rangle$$\wr$

$= \ \{\varepsilon\} \cup \alpha^a(X) \cdot \{\mathrm{B}\} \cdot \alpha^a(\mathcal{S}^t[\![\mathrm{C}]\!])$

$\wr$definition of α^a and set of action paths concatenation$\wr$

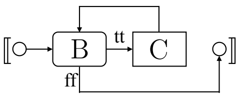
$\subseteq \ \{\varepsilon\} \cup \alpha^a(X) \cdot \{\mathrm{B}\} \cdot \mathcal{G}^a[\![\circ\!\!\longrightarrow\!\boxed{\mathrm{C}}\!\longrightarrow\!\circ]\!]$

$\wr$by structural induction hypothesis and the set of action paths concatenation $\cdot$ is $\subseteq$-increasing in both of its arguments.$\wr$

$= \ \mathcal{F}^a[\![$  $]\!](\alpha^a(X))$ $\wr$definition of $\mathcal{F}^a[\![$  $]\!]$$\wr$

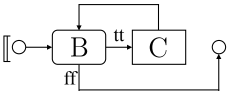
We have proved the semi-commutation property:

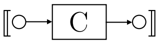
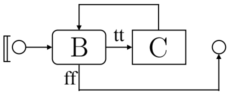
$\forall X \in \wp(\Pi)$: (10)

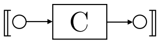
$\alpha^a(\mathcal{F}^{ti}[\![\text{while (B) \{C\}}]\!](X)) \subseteq \mathcal{F}^a[\![$  $]\!](\alpha^a(X))$

- **Union preservation**

  Let us show that the iteration transformer preserves unions.

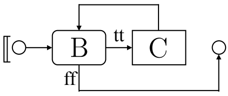
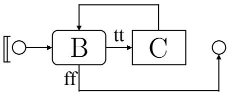
$\mathcal{F}^a[\![$  $]\!](\bigcup_{i\in\Delta} S_i)$

$= \{\varepsilon\} \cup (\bigcup_{i\in\Delta} S_i) \cdot \{\text{B}\} \cdot \mathcal{G}^a[\![$  $]\!]$ ≀definition of $\mathcal{F}^a[\![$  $]\!]$≀

$= \{\varepsilon\} \cup \bigcup_{i\in\Delta}(S_i \cdot \{\text{B}\} \cdot \mathcal{G}^a[\![$  $]\!])$

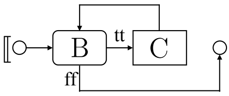
≀since the concatenation $\cdot$ of sets of action paths preserves unions≀

$= \bigcup_{i\in\Delta}(\{\varepsilon\} \cup S_i \cdot \{\text{B}\} \cdot \mathcal{G}^a[\![$ $]\!])$ ≀definition of $\cup$≀

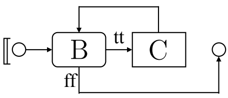
$= \bigcup_{i\in\Delta} \mathcal{F}^a[\![$  $]\!](S_i)$ ≀definition of $\mathcal{F}^a[\![$  $]\!]$≀

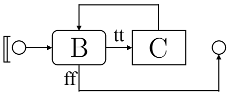
It follows that is $\subseteq$-increasing.

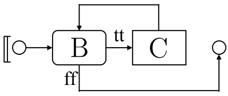
- **Fixpoint approximation**

A consequence of the semi-commutation property is that the abstraction of the concrete iterates $\langle S_t^n, n \in \mathbb{N}\rangle$ of $\mathcal{F}^{ti}[\![\text{while (B) \{C\}}]\!]$ from $\emptyset$ by α^a s over-approximated by the iterates $\langle S_a^n, n \in \mathbb{N}\rangle$ of $\mathcal{F}^a[\![$  $]\!]$ in the abstract: $\forall n \in \mathbb{N} : \alpha^a(S_t^n) \subseteq S_a^n$. The proof is by recurrence on $n \in \mathbb{N}$.

- $\alpha^a(S_t^0) = \alpha^a(\emptyset) = \emptyset = S_a^0$
- $\alpha^a(S_t^{n+1}) = \alpha^a(\mathcal{F}^{ti}[\![\text{while (B) \{C\}}]\!](S_t^n))$ ≀definition of iteration≀

$\subseteq \mathcal{F}^a[\![$  $]\!](\alpha^a(S_t^n))$ ≀semi-commutation property≀

$\subseteq \mathcal{F}^a[\![$  $]\!](S_a^n)$

≀by induction hypothesis $\alpha^a(S_t^n) \subseteq S_a^n$ and $\mathcal{F}^a[\![$  $]\!]$ is $\subseteq$-increasing≀

$= S_a^{n+1}$ ≀definition of iteration≀

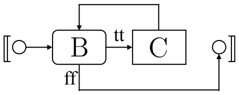
It follows that we have the following fixpoint approximation,

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

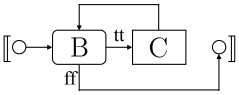
$$= \alpha^a(\bigcup_{n\in\mathbb{N}} S_t^n) \qquad \wr\text{since } \mathcal{F}^{ti}[\![\mathrm{while}\,(\mathrm{B})\,\{\mathrm{C}\}]\!] \text{ preserves unions}\int$$

$$= \bigcup_{n\in\mathbb{N}} \alpha^a(S_t^n) \qquad \wr\text{since } \alpha^a \text{ preserves unions}\int$$

$$\subseteq \bigcup_{n\in\mathbb{N}} S_a^n \qquad \wr\text{since } \forall n \in \mathbb{N} : \alpha^a(S_t^n) \subseteq S_a^n \text{ and definition of } \cup\int$$

$= \mathrm{lfp}^{\subseteq}\mathcal{F}^a[\![$ 

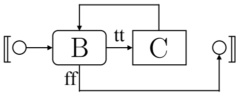
$]\!]$

$\wr$since $\mathcal{F}^a[\![$ 

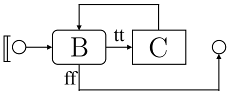
$]\!]$ preserves unions$\int$

- **Iterates and least fixpoint of the iteration transformer**

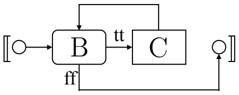
  Given a set $\mathcal{A}$ of action paths, let us define its natural powers as $\mathcal{A}^0 \triangleq \{\varepsilon\}$, $\mathcal{A}^1 \triangleq \mathcal{A}$, $\mathcal{A}^n = \underbrace{\mathcal{A}\cdot \ldots \cdot\mathcal{A}}_{n \text{ times}}$ for $n > 1$. The rule of powers $\mathcal{A}^{p+q} = \mathcal{A}^p \cdot \mathcal{A}^q$ is trivial.

  Let us calculate the iterates $\langle F^n, n \in \mathbb{N}\rangle$ of $\mathcal{F}^a[\![$ 

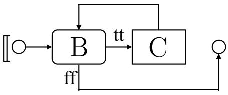
$]\!]$ from $F^0 = \emptyset$.

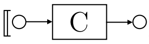
- $F^1 = \mathcal{F}^a[\![$ 

$]\!](F^0) = \{\varepsilon\}$

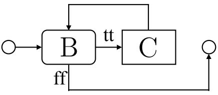
  $\wr$definition of iteration, $\mathcal{F}^a[\![$ 

$]\!]$ and $\forall\mathcal{A} : \emptyset\cdot\mathcal{A} = \mathcal{A}\int$

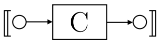
- $F^2 = \mathcal{F}^a[\![$ 

$]\!](F^1) = \{\varepsilon\} \cup \{\varepsilon\}\cdot\{\mathrm{B}\}\cdot\mathcal{G}^a[\![$ 

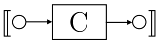
$]\!]$

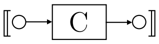
  $\wr$definition of iteration and $\mathcal{F}^a[\![$ 

$]\!]\int$

  $= \{\varepsilon\} \cup \{\mathrm{B}\}\cdot\mathcal{G}^a[\![$ 

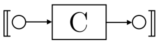
$]\!]$ $\wr$since $\forall\mathcal{A} : \{\varepsilon\}\cdot\mathcal{A} = \mathcal{A}\int$

  $= (\{\mathrm{B}\}\cdot\mathcal{G}^a[\![$ 

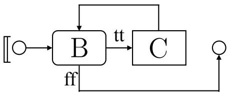
$]\!])^0 \cup (\{\mathrm{B}\}\cdot\mathcal{G}^a[\![$ 

$]\!])^1$

  $\wr$definition of natural powers $\mathcal{A}^0 \triangleq \{\varepsilon\}$ and $\mathcal{A}^1 \triangleq \mathcal{A}\int$

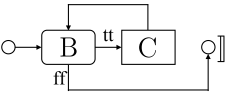
- $F^n = \bigcup_{i=0}^{n-1}(\{\mathrm{B}\}\cdot\mathcal{G}^a[\![$ 

$]\!])^i$ $\wr$induction hypothesis$\int$

- $F^{n+1} = \mathcal{F}^a[\![$ 

$]\!](F^n)$ $\wr$definition of iterates$\int$

$$= \{\varepsilon\} \cup (\bigcup_{i=0}^{n-1}(\{\mathrm{B}\}\cdot \mathcal{G}^a\llbracket \circ\!\!\rightarrow\!\boxed{\mathrm{C}}\!\rightarrow\!\circ\rrbracket)^i)\cdot\{\mathrm{B}\}\cdot\mathcal{G}^a\llbracket \circ\!\!\rightarrow\!\boxed{\mathrm{C}}\!\rightarrow\!\circ\rrbracket$$

≀definition of $\mathcal{F}^a\llbracket$  $\rrbracket$ and induction hypothesisʃ

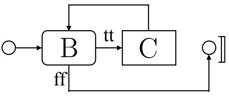
$= (\{\mathrm{B}\} \cdot \mathcal{G}^a\llbracket \circ\!\!\rightarrow\!\boxed{\mathrm{C}}\!\rightarrow\!\circ\rrbracket)^0 \cup \bigcup_{i=0}^{n-1}(((\{\mathrm{B}\} \cdot \mathcal{G}^a\llbracket \circ\!\!\rightarrow\!\boxed{\mathrm{C}}\!\rightarrow\!\circ\rrbracket)^i \cdot \{\mathrm{B}\} \cdot \mathcal{G}^a\llbracket \circ\!\!\rightarrow\!\boxed{\mathrm{C}}\!\rightarrow\!\circ\rrbracket)$

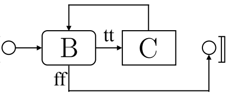
≀definition of natural powers $\mathcal{A}^0 = \{\varepsilon\}$ and $\bigcup_{i\in\delta}(\mathcal{A}^i)\cdot\mathcal{A} = \bigcup_{i\in\delta}(\mathcal{A}^i\cdot\mathcal{A})$ʃ

$$= \bigcup_{i=0}^{(n+1)-1}(\{\mathrm{B}\}\cdot \mathcal{G}^a\llbracket \circ\!\!\rightarrow\!\boxed{\mathrm{C}}\!\rightarrow\!\circ\rrbracket)^i$$

≀definition of natural powers $\mathcal{A}^{n+1} = \mathcal{A}^n\cdot\mathcal{A}$ and definition of $\cup$ʃ

- $\forall n \in \mathbb{N} : F^n = \bigcup_{i=0}^{n-1}(\{\mathrm{B}\}\cdot \mathcal{G}^a\llbracket \circ\!\!\rightarrow\!\boxed{\mathrm{C}}\!\rightarrow\!\circ\rrbracket)^i$ ≀by recurrenceʃ

$\mathrm{lfp}^{\subseteq}\mathcal{F}^a\llbracket$  $\rrbracket$

$= \bigcup_{n\in\mathbb{N}} F^n$ ≀since $\mathcal{F}^a\llbracket$  $\rrbracket$ preserves unionsʃ

$= \bigcup_{n\in\mathbb{N}}(\{\mathrm{B}\}\cdot \mathcal{G}^a\llbracket \circ\!\!\rightarrow\!\boxed{\mathrm{C}}\!\rightarrow\!\circ\rrbracket)^n$ ≀definition of natural powers $\mathcal{A}^n$ and $\cup$ʃ

which we can write as $(\{\mathrm{B}\}\cdot \mathcal{G}^a\llbracket \circ\!\!\rightarrow\!\boxed{\mathrm{C}}\!\rightarrow\!\circ\rrbracket)^*$.

- **Iteration**

$\alpha^a(\mathcal{S}^t\llbracket\mathrm{while\,(B)\,\{C\}}\rrbracket)$

$= \alpha^a(\mathrm{let}\ \mathcal{S}^{ti}\llbracket\mathrm{while\,(B)\,\{C\}}\rrbracket \triangleq \mathrm{lfp}^{\subseteq}\mathcal{F}^{ti}\llbracket\mathrm{while\,(B)\,\{C\}}\rrbracket\ \mathrm{in}\ \{\pi \xrightarrow{\mathrm{A}} \langle\mathrm{while\,(B)\,\{C\}},\rho\rangle \xrightarrow{\neg\mathrm{B}} \langle\mathrm{stop},\rho\rangle \mid \pi \in \mathit{\Pi} \wedge \pi \xrightarrow{\mathrm{A}} \langle\mathrm{while\,(B)\,\{C\}},\rho\rangle \in \mathcal{S}^{ti}\llbracket\mathrm{while\,(B)\,\{C\}}\rrbracket \wedge \mathit{false} \in \mathcal{E}\llbracket\mathrm{B}\rrbracket\rho\})$

≀definition of $\mathcal{S}^t\llbracket\mathrm{while\,(B)\,\{C\}}\rrbracket$ʃ

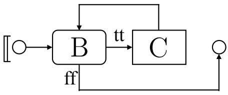
$= \mathrm{let}\ \mathcal{S}^{ti}\llbracket\mathrm{while\,(B)\,\{C\}}\rrbracket \triangleq \mathrm{lfp}^{\subseteq}\mathcal{F}^{ti}\llbracket\mathrm{while\,(B)\,\{C\}}\rrbracket\ \mathrm{in}\ \alpha^a(\{\pi \xrightarrow{\mathrm{A}} \langle\mathrm{while\,(B)\,\{C\}},\rho\rangle \xrightarrow{\neg\mathrm{B}} \langle\mathrm{stop},\rho\rangle \mid \pi \in \mathit{\Pi} \wedge \pi \xrightarrow{\mathrm{A}} \langle\mathrm{while\,(B)\,\{C\}},\rho\rangle \in \mathcal{S}^{ti}\llbracket\mathrm{while\,(B)\,\{C\}}\rrbracket \wedge \mathit{false} \in \mathcal{E}\llbracket\mathrm{B}\rrbracket\rho\})$

≀since $f(\mathrm{let}\ ...\ \mathrm{in}\ ...) = \mathrm{let}\ ...\ \mathrm{in}\ f(...)$ʃ

$= \mathrm{let}\ \mathcal{S}^{ti}\llbracket\mathrm{while\,(B)\,\{C\}}\rrbracket \triangleq \mathrm{lfp}^{\subseteq}\mathcal{F}^{ti}\llbracket\mathrm{while\,(B)\,\{C\}}\rrbracket\ \mathrm{in}\ \alpha^a(\mathcal{S}^{ti}\llbracket\mathrm{while\,(B)\,\{C\}}\rrbracket)\cdot\neg\mathrm{B}$

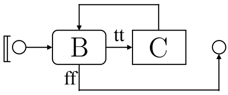
≀definition of α^a and ignoring the result $\mathit{false} \in \mathcal{E}\llbracket\mathrm{B}\rrbracket\rho$ of testʃ

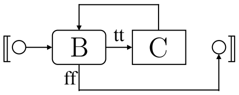
$= \alpha^a(\mathrm{lfp}^{\subseteq}\mathcal{F}^{ti}\llbracket\mathrm{while\,(B)\,\{C\}}\rrbracket)\cdot\{\neg\mathrm{B}\}$ ≀definition of let ... in ...ʃ

$\subseteq\ \mathrm{lfp}^{\subseteq}\mathcal{F}^a[\![$ 

$]\!] \cdot \{\neg\mathrm{B}\}$

$\wr$by fixpoint approximation and $\cdot$ preserves unions so is $\subseteq$-increasing$\int$

$=\ \mathcal{G}^a[\![$ 

$]\!]$ $\wr$definition of $\mathcal{G}^a[\![$ 

$]\!]\int$

C Proofs in Section 3.2

Theorem 2 (Homomorphic Abstraction). *Given a function $h : C \mapsto A$, let $\alpha_h(X) = \{h(x) \mid x \in X\}$ and $\gamma_h(Y) = \{x \mid h(x) \in Y\}$, then α_h and γ_h form a Galois connection:*

$$(\wp(C), \subseteq) \underset{\alpha_h}{\overset{\gamma_h}{\leftrightarrows}} (\wp(A), \subseteq) \tag{11}$$

Proof. For all $X \in \wp(C)$ and $Y \in \wp(A)$,

$$\begin{aligned}
& \alpha_h(X) \subseteq Y \\
\iff\ & \{h(x) \mid x \in X\} \subseteq Y && \wr\text{definition of } \alpha_h\textstyle\int \\
\iff\ & \forall x \in X : h(x) \in Y && \wr\text{definition of } \subseteq\textstyle\int \\
\iff\ & X \subseteq \{x \mid h(x) \in Y\} && \wr\text{definition of } \subseteq\textstyle\int \\
\iff\ & X \subseteq \gamma_h(Y) && \wr\text{definition of } \gamma_h\textstyle\int
\end{aligned}$$

□

Proof of (1). By Theorem 2 where h is α^c.

Proof of (4). By Theorem 2 where h is α^d.

Proof of (7). By Theorem 2 where h is α^ℓ.

Proof of (8). The composition of Galois connections is still a Galois connection. For all $\mathcal{A} \in \wp(\mathbb{A}^*)$ and $\mathcal{B} \in \wp((\mathbb{A}^B)^* \setminus \mathbb{D})$,

$$\begin{aligned}
& \alpha^b(\mathcal{A}) \subseteq \mathcal{B} \\
\iff\ & \alpha^\ell \circ \alpha^d \circ \alpha^c(\mathcal{A}) \subseteq \mathcal{B} && \wr\text{definition of } \alpha^b\textstyle\int \\
\iff\ & \alpha^d \circ \alpha^c(\mathcal{A}) \subseteq \gamma^\ell(\mathcal{B}) && \wr\text{by } (\wp((\mathbb{A}^B)^*), \subseteq) \underset{\alpha^\ell}{\overset{\gamma^\ell}{\leftrightarrows}} (\wp((\mathbb{A}^B)^* \setminus \mathbb{D}), \subseteq)\textstyle\int \\
\iff\ & \alpha^c(\mathcal{A}) \subseteq \gamma^d \circ \gamma^\ell(\mathcal{B}) && \wr\text{by } (\wp((\mathbb{A}^C)^*), \subseteq) \underset{\alpha^d}{\overset{\gamma^d}{\leftrightarrows}} (\wp((\mathbb{A}^B)^*), \subseteq)\textstyle\int \\
\iff\ & \mathcal{A} \subseteq \gamma^c \circ \gamma^d \circ \gamma^\ell(\mathcal{B}) && \wr\text{by } (\wp(\mathbb{A}^*), \subseteq) \underset{\alpha^c}{\overset{\gamma^c}{\leftrightarrows}} (\wp((\mathbb{A}^C)^*), \subseteq)\textstyle\int \\
\iff\ & \mathcal{A} \subseteq \gamma^b(\mathcal{B}) && \wr\text{definition of } \gamma^b\textstyle\int
\end{aligned}$$