

Introduction to Static Analysis

Chapter 5

山本 航平

この章でやること

これまでの章で学んだ解析手法を発展させる

- Abstract domains

より良い抽象ドメインの設計方法

- Iteration techniques

解析コストを下げる, 精度を上げる反復手法

- Scalability techniques

時間的, 空間的なコストを下げる解析手法

- Analysis direction

逆向きの解析手法

Contents

5.1 Construction of Abstract Domains

- Boolean の抽象化
- 性質の論理積
- 性質の論理和
- 抽象ドメインの設計方法

5.2 Advanced Iteration Techniques

5.3 Sparse Analysis

5.4 Modular Analysis

5.5 Backward Analysis

5.1 Construction of Abstract Domains

- ・適切な抽象ドメインを選択したい
- ・現実の静的解析では、複雑な性質を表現できる抽象ドメインが必要

やること

- ・ Boolean 変数の抽象化
- ・ 複数の性質の合成 (論理和, 論理積)
- ・ 抽象ドメインの構築方法

Contents

5.1 Construction of Abstract Domains

- Boolean の抽象化
- 性質の論理積
- 性質の論理和
- 抽象ドメインの設計方法

5.2 Advanced Iteration Techniques

5.3 Sparse Analysis

5.4 Modular Analysis

5.5 Backward Analysis

Non-Relational Boolean Abstraction

- ・整数データ以外も抽象化したい
- ・Boolean の抽象化を考える

Non-Relational Boolean Abstraction

抽象ドメイン $A_{\mathbb{B}} = \{\top, \underline{\text{true}}, \underline{\text{false}}, \perp\}$

具体化関数 $\gamma_{\mathbb{B}} : \gamma_{\mathbb{B}}(\perp) = \emptyset$

$\gamma_{\mathbb{B}}(\underline{\text{true}}) = \{\text{true}\}$

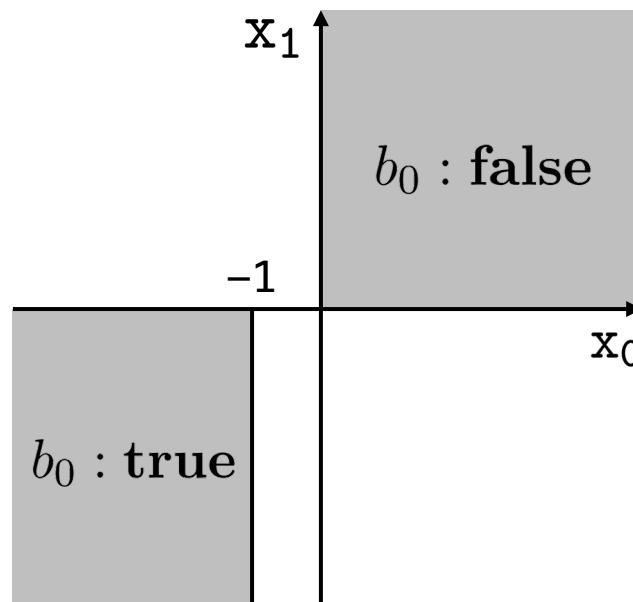
$\gamma_{\mathbb{B}}(\underline{\text{false}}) = \{\text{false}\}$

$\gamma_{\mathbb{B}}(\top) = \{\text{true}, \text{false}\}$

Example (Non-relational Boolean Abstraction)

- Boolean 変数の抽象ドメイン $\mathbb{A}_{\mathbb{B}} = \{\top, \underline{\text{true}}, \underline{\text{false}}, \perp\}$
- 数値変数の抽象化：区間抽象

```
1 int x0, x1;
2 bool b0, b1;
3 if (x1 < 0)
4     x1 = -x1;
5 b0 = x0 < 0;
6 b1 = x1 >= 0;
7 if (b0)
8     x1 = -x1;
```



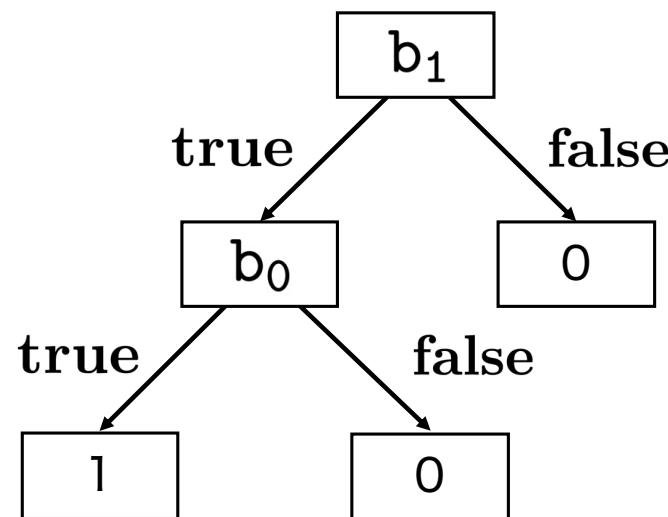
$b_0 \mapsto \top$
 $b_1 \mapsto \underline{\text{true}}$
 $x_0 \mapsto \top$
 $x_1 \mapsto \top$

Relational Boolean Abstraction

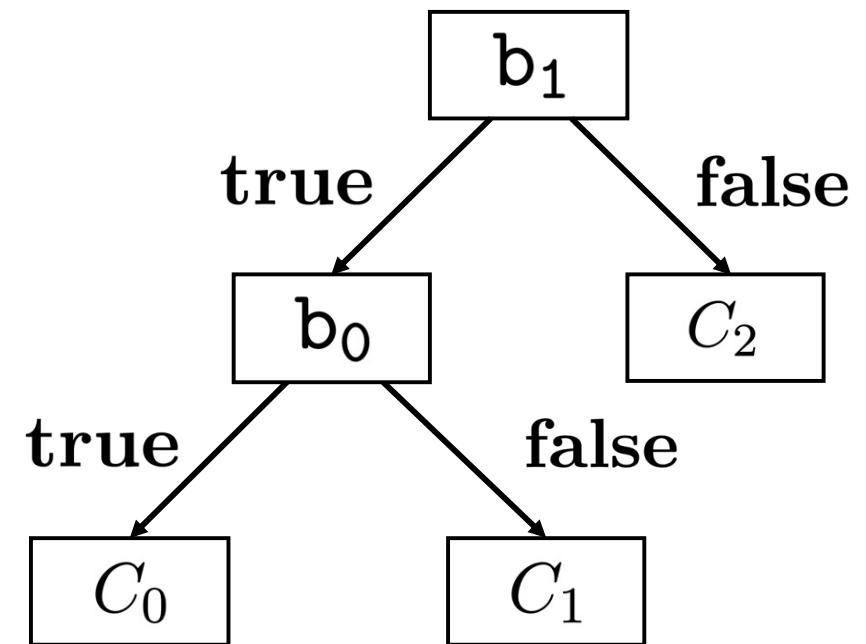
- Boolean 変数と数値変数の制約の関係

if b contains **true** : then numerical condition C_0 holds
otherwise : then numerical condition C_1 holds

- 決定木 (decision tree) として表せる



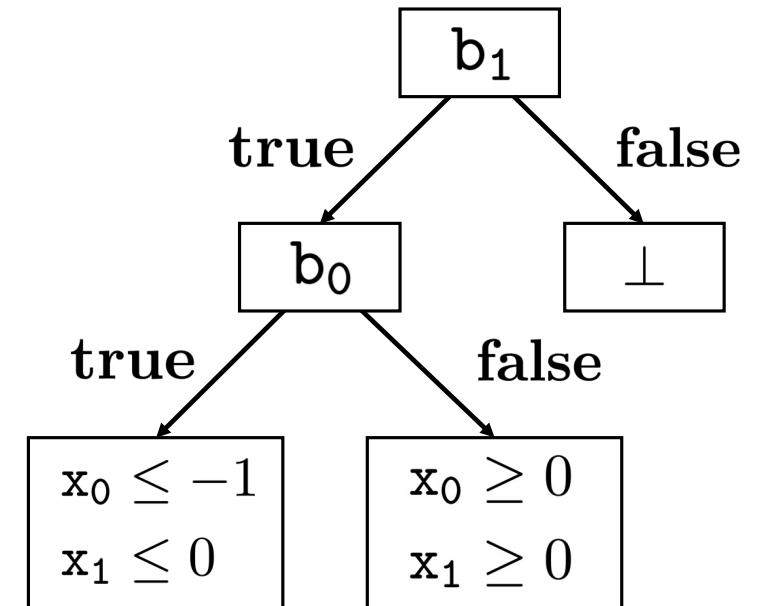
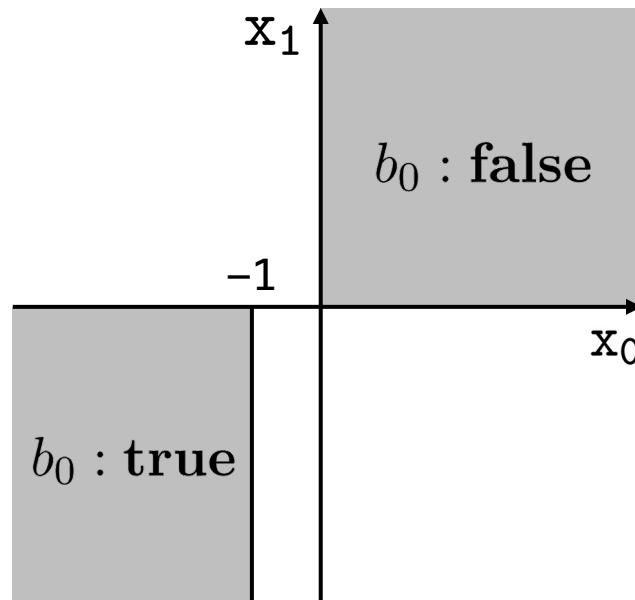
→
葉が数値制約



binary decision diagram (BDDs)

Example (Relational Boolean Abstraction)

```
1 int x0, x1;
2 bool b0, b1;
3 if (x1 < 0)
4     x1 = -x1;
5 b0 = x0 < 0;
6 b1 = x1 >= 0;
7 if (b0)
8     x1 = -x1;
```



決定木

Contents

5.1 Construction of Abstract Domains

- Boolean の抽象化
- 性質の論理積
- 性質の論理和
- 抽象ドメインの設計方法

5.2 Advanced Iteration Techniques

5.3 Sparse Analysis

5.4 Modular Analysis

5.5 Backward Analysis

5.1.2 Describing Conjunctive Properties

- 複数の抽象要素が表す論理的性質を論理積で結合する

Product domain

$$\text{抽象ドメイン } \mathbb{A}_\times = \mathbb{A}_0 \times \mathbb{A}_1$$

\mathbb{A}_\times の要素は 2 つの抽象ドメイン $\mathbb{A}_0, \mathbb{A}_1$ の要素の組

具体化関数 γ_\times :

$$\forall (a_0, a_1) \in \mathbb{A}_\times, \gamma_\times(a_0, a_1) = \gamma_0(a_0) \cap \gamma_1(a_1)$$

Example (Product domain)

\mathbb{A}_0 : 区間抽象

\mathbb{A}_1 : 合同抽象 ((0, 2) : 2で割ったときの余りが0)

Product domain $\mathbb{A}_{\times} = \mathbb{A}_0 \times \mathbb{A}_1$ を考える

ex.) $([0, 100], (0, 2))$

(0以上100以下) かつ (2で割ったときの余りが0) の値の抽象化

- (使用例) ポインタを扱うときに, 範囲とアライメントの制約をどちらも表せる

Reduction

区間制約 $[1, 3]$ と合同制約 $(0, 2)$ の組 $([1, 3], (0, 2))$ を考える

具体化すると, $\gamma_{\times}([1, 3], (0, 2)) = \gamma_0([1, 3]) \cap \gamma_1((0, 2)) = \{2\}$

抽象化すると, $([2, 2], (0, 2))$

→ より強い制約となった

- Reduction : 抽象要素に具体化関数と抽象化関数を適用させ,
情報を洗練させること

Reduced product

関係 \equiv に対する $\mathbb{A}_0 \times \mathbb{A}_1$ の同値類の集合 \mathbb{A}_{\bowtie}

$$(a_0, a_1) \equiv (a'_0, a'_1) \iff \gamma_{\times}(a_0, a_1) = \gamma_{\times}(a'_0, a'_1)$$

Example (Reduced product)

2章のように2つの変数 x, y を区間抽象で抽象化する

→ この2つの組は product domain と考えられる

- $M^\sharp(x) \neq \perp, M^\sharp(y) \neq \perp$ のとき

M^\sharp と \equiv に対して合同な抽象状態は存在しない

→ reduction できない

- $M^\sharp(x) = \perp$ のとき

$\gamma_{\mathcal{N}}(M^\sharp) = \emptyset = \gamma_{\mathcal{N}}(M_\perp^\sharp)$ ($M_\perp^\sharp : M_\perp^\sharp(x) = M_\perp^\sharp(y) = \perp$)

→ M^\sharp と M_\perp^\sharp は合同だから, M_\perp^\sharp に reduction できる

Coalescent Product

今の話を一般化すると, M^\sharp の変数が一つでも最小要素にマッピングされると, M_\perp^\sharp に reduction できる

Coalescent Product

最小要素 M_\perp^\sharp

$$\forall x \in \mathbb{X}, \quad M_\perp^\sharp(x) = \perp$$

要素 M^\sharp

$$\forall x \in \mathbb{X}, \quad M^\sharp(x) \neq \perp$$

Example (Reduced product)

区間抽象 $[a, b]$ と合同抽象 (n, p) の組を考える

- $([1, 5], (0, 2))$
→ $([1, 5], (0, 2)) \equiv ([2, 4], (0, 2))$
- $([1, 3], (0, 4))$
→ $([1, 3], (0, 4)) \equiv (\perp, \perp)$

Example (Coalescent product-based analysis)

Coalescent product が解析の精度を上げる例

```
1 int x, y;  
2 x = 8;  
3 y = 1;  
4 if (x < 0) {  
5     y = 0;  
6 } else {}
```

(3行目)

$\{x \mapsto [8, 8], y \mapsto [1, 1]\}$

- Reduction を使わないとき

true branch (4行目)	$\{x \mapsto \perp, y \mapsto [1, 1]\}$
(5行目)	$\{x \mapsto \perp, y \mapsto [0, 0]\}$
join (6行目)	$\{x \mapsto [8, 8], y \mapsto [0, 1]\}$

- Reduction を使うとき

true branch (4, 5行目)	$\{x \mapsto \perp, y \mapsto \perp\}$
join (6行目)	$\{x \mapsto [8, 8], y \mapsto [1, 1]\}$

Analysis with a Reduced Product Domain

- Reduced product による解析は静的解析の積よりも正確になる
 - 2つの解析を別々に行うよりも, reduced product に基づく一回の解析の方が精度が上がる
- 最適な reduced product の定義は難しい
 - すべての抽象要素を健全で最適な要素に変換するのは計算コストが大きい
 - 計算コストと解析精度のバランスを考える
 - 特定の場所のみで reduction を使用する

Contents

5.1 Construction of Abstract Domains

- Boolean の抽象化
- 性質の論理積
- 性質の論理和
- 抽象ドメインの設計方法

5.2 Advanced Iteration Techniques

5.3 Sparse Analysis

5.4 Modular Analysis

5.5 Backward Analysis

5.1.3 Describing Properties Involving Case Splits

```
1 int x, y, z;  
2 if (x > 0) {  
3     x = x + 1;  
4 } else {}  
5 z = y / (x - 1);
```

- ・5行目で x が1ではないことを確認したい

$$(-\infty, 0] \cup [2, +\infty)$$

- ・3章での抽象ドメインではこれを表すことができない
→ 区間抽象で x の抽象化は $(-\infty, +\infty)$

Disjunctive Completion

- 性質を論理和の形式で表す

$$A_0 \vee A_1 \vee \cdots \vee A_n$$

- Exact abstract join : join 操作をさらに正確にする

$$\gamma(M_0^\sharp) \cup \gamma(M_1^\sharp) = \gamma(M_0^\sharp \sqcup^\sharp M_1^\sharp)$$

3章での定義 : $\gamma(M_0^\sharp) \cup \gamma(M_1^\sharp) \subseteq \gamma(M_0^\sharp \sqcup^\sharp M_1^\sharp)$

- 多くの抽象ドメインでは exact abstract join を定義できない

Disjunctive Completion

Disjunctive completion :

Exact abstract join を定義できるように抽象要素を追加する

ex.) 区間抽象ドメインにすべての有限の論理和を追加

$$(-\infty, 0] \cup [2, +\infty), \quad (-\infty, -3] \cup [-1, 1] \cup [3, +\infty)$$

- 抽象状態の数が非常に大きくなり, コストが大きい

Reduced Cardinal Power

Case analysis による性質の表し方

$$(A_0 \Rightarrow B_0) \wedge (A_1 \Rightarrow B_1) \wedge \cdots \wedge (A_n \Rightarrow B_n)$$

ex.)

if b contains **true** then numerical condition $x > 10$ holds,
and if b contains **false** then numerical condition $x < 3$ holds.

Reduced Cardinal Power

実行可能な動作の集合 : \mathcal{E}

抽象ドメイン : $\mathbb{A}_0, \mathbb{A}_1$ (具体化関数 : $\gamma_0 : \mathbb{A}_0 \rightarrow \wp(\mathcal{E}), \gamma_1 : \mathbb{A}_1 \rightarrow \wp(\mathcal{E})$)

Cardinal Power Abstraction

- ・ 抽象ドメイン \mathbb{A}_\rightarrow は \mathbb{A}_0 から \mathbb{A}_1 への単調な関数
- ・ 具体化関数

$$\gamma_\rightarrow(a_\rightarrow) = \{m \in \mathcal{E} \mid \forall a_0 \in \mathbb{A}_0, m \in \gamma_0(a_0) \implies m \in \gamma_1(a_\rightarrow(a_0))\}$$

- ・ Reduction によってより正確な性質を表せる

State Partitioning

実行可能な動作の集合 : \mathcal{E}

抽象ドメイン : $\mathbb{A}_0, \mathbb{A}_1$ (具体化関数 : $\gamma_0 : \mathbb{A}_0 \rightarrow \wp(\mathcal{E}), \gamma_1 : \mathbb{A}_1 \rightarrow \wp(\mathcal{E})$)

Cardinal Power Abstraction

- ・抽象ドメイン \mathbb{A}_\rightarrow は \mathbb{A}_0 から \mathbb{A}_1 への単調な関数
- ・具体化関数

$$\gamma_\rightarrow(a_\rightarrow) = \{m \in \mathcal{E} \mid \forall a_0 \in \mathbb{A}_0, m \in \gamma_0(a_0) \implies m \in \gamma_1(a_\in(a_0))\}$$

State partitioning : \mathcal{E} をメモリ状態の集合 \mathbb{M} としたもの

\mathbb{A}_1 を区間抽象として \mathbb{A}_0 を考える

State Partitioning

```
1 int x, y, z;  
2 if (x > 0) {  
3     x = x + 1;  
4 } else {}  
5 z = y / (x - 1);
```

$\mathbb{A}_0 = \{\perp, [< 0], [= 0], [> 0], \top\}$ とするとき

$$\begin{aligned}\mathbb{A}_{\rightarrow} : [< 0] &\longmapsto (-\infty, -1] \\ [= 0] &\longmapsto [0, 0] \\ [> 0] &\longmapsto [2, +\infty)\end{aligned}$$

State Partitioning

- Flow-sensitive

プログラムポイントごとに抽象状態を計算し, そのポイントでのすべての状態を表現する

\mathbb{A}_{\rightarrow} をプログラムラベルから抽象状態への関数と考える (4章)

- Context-sensitive

複数の手続き呼び出しのコンテキストを識別する

\mathbb{A}_{\rightarrow} を calling context から抽象状態への関数と考える

- Dynamic partitioning : partition (\mathbb{A}_{\rightarrow}) を解析自身が決定する

Trace Partitioning

Trace partitioning : \mathcal{E} をすべてのプログラム実行（状態の列）の集合としたもの → \mathbb{A}_0 はプログラム実行の抽象化

- ・ 過去の情報も持っている
(State partitioning はある一点の状態のみを見る)
- ・ ex.) the "true" branch was executed \Rightarrow $x_0 \in [2, +\infty)$
 \wedge the "false" branch was executed \Rightarrow $x_0 \in (-\infty, 0]$

```
1 int x, y, z;
2 if (x > 0) {
3     x = x + 1;
4 } else {}
5 z = y / (x - 1);
```

Contents

5.1 Construction of Abstract Domains

- Boolean の抽象化
- 性質の論理積
- 性質の論理和
- 抽象ドメインの設計方法

5.2 Advanced Iteration Techniques

5.3 Sparse Analysis

5.4 Modular Analysis

5.5 Backward Analysis

5.1.4 Construction of an Abstract Domain

抽象ドメインを構築する上で考えること

- 抽象要素の表し方

ex.) 凸多角形抽象は線形不等式で表すか？図形的に表すか？

- 抽象要素の意味 (具体との関係)

具体化関数や抽象化関数

Reduction

→ Sound な解析をする上で重要

- 抽象での操作

Lattice 操作 (join, widening) や遷移関数 (post-condition の計算)

5.1.4 Construction of an Abstract Domain

抽象ドメインを構築する上で考えること

- ・複数の変数間での制約があるときは relational 抽象ドメインが必要
- ・複数の要素の論理積で表すときは product 抽象ドメインや reduced product 抽象ドメインが必要
- ・性質を論理和で表すときは disjunctive completion や partitioning 抽象ドメインが必要

Contents

5.1 Construction of Abstract Domains

5.2 Advanced Iteration Techniques

- Loop Unrolling
- Widening の精度を上げる
- 最小不動点を用いた抽象化の精度を上げる

5.3 Sparse Analysis

5.4 Modular Analysis

5.5 Backward Analysis

5.2 Advanced Iteration Techniques

- 抽象上でのループ実行は制約が弱くなっていく
→ 工夫をしてループ解析の正確さを向上させたい

3章の復習

$$[\![\text{while}(B)\{C\}]\!]_{\mathcal{P}}^{\sharp}(M) = \mathcal{F}_{\neg B}^{\sharp} \left(\text{abs_iter}([\![C]\!]_{\mathcal{P}}^{\sharp} \circ \mathcal{F}_B^{\sharp}, M^{\sharp}) \right)$$

`abs_iter`(F^{\sharp} , M^{\sharp})

$$M_0^{\sharp} = M^{\sharp}$$

$$M_{k+1}^{\sharp} = M_k^{\sharp} \sqcup^{\sharp} F^{\sharp}(M_k^{\sharp})$$

$$(\text{widening} : M_{k+1}^{\sharp} = M_k^{\sharp} \triangledown F^{\sharp}(M_k^{\sharp}))$$

これを収束するまで繰り返す

Contents

5.1 Construction of Abstract Domains

5.2 Advanced Iteration Techniques

- Loop Unrolling
- Widening の精度を上げる
- 最小不動点を用いた抽象化の精度を上げる

5.3 Sparse Analysis

5.4 Modular Analysis

5.5 Backward Analysis

5.2.1 Loop Unrolling

- 直感的に x の範囲は $[0, 1001]$
- 1回目のループでは x の範囲に制約がない
→ 最終的な解析結果も x の範囲は実数全体 $(-\infty, +\infty)$

```
1 int i = 1;
2 while (i > 0) {
3     if (x < 0 || x > 1000) {
4         x = 0;
5     } else {
6         x = x + 1;
7     }
8     input(i);
9 }
```

⇒ 反復の1回目を別に処理する

5.2.1 Loop Unrolling

- ループ直前の x の範囲は $[0, 1001]$
→ ループ解析結果も x の範囲は
 $[0, 1001]$ となる

```
1 if (x < 0 || x > 1000) {  
2     x = 0;  
3 } else {  
4     x = x + 1;  
5 }  
6 input(i);  
7 while (i > 0) {  
8     if (x < 0 || x > 1000) {  
9         x = 0;  
10    } else {  
11        x = x + 1;  
12    }  
13    input(i);  
14 }
```

5.2.1 Loop Unrolling

Loop unrolling の原理： join や widening の操作を遅らせる

最初の N 回の反復を展開する

$$M_0^\sharp = M^\sharp$$
$$M_{k+1}^\sharp = \begin{cases} F^\sharp(M_k^\sharp) & \text{if } k < N \\ M_k^\sharp \vee F^\sharp(M_k^\sharp) & \text{otherwise} \end{cases}$$

M_{\lim}^\sharp に収束したときの結果

$$\mathcal{F}_{\neg B}^\sharp(M_0^\sharp) \sqcup^\sharp \cdots \sqcup^\sharp \mathcal{F}_{\neg B}^\sharp(M_{N-1}^\sharp) \sqcup^\sharp \mathcal{F}_{\neg B}^\sharp(M_{\lim}^\sharp)$$

Contents

5.1 Construction of Abstract Domains

5.2 Advanced Iteration Techniques

- Loop Unrolling
- Widening の精度を上げる
- 最小不動点を用いた抽象化の精度を上げる

5.3 Sparse Analysis

5.4 Modular Analysis

5.5 Backward Analysis

Delaying Widening with Union

- Widening による解析結果の精度を上げたい

- ループに入る直前の x の範囲は $[0, 0]$
- 1回目のループ後の x の範囲は $[-1, 2]$

$$[0, 0] \nabla_{\mathcal{V}} [-1, 2] = (-\infty, +\infty)$$

→ 直感的に x の範囲は $[-1, +\infty)$

```
1 int x = 0;
2 while (rand()) {
3     if (rand()) {
4         x = -1;
5     } else {
6         x = x + 2;
7     }
8 }
```

ここでの widening の定義

$$[n, p] \nabla_{\mathcal{V}} [n, q] = \begin{cases} [n, p] & \text{if } p \geq q \\ [n, +\infty) & \text{if } p < q \end{cases}$$

Delaying Widening with Union

- Widening の適用を遅らせる

$$\begin{aligned} M_0^\sharp &= M^\sharp \\ M_{k+1}^\sharp &= M_k^\sharp \sqcup^\sharp F^\sharp(M_k^\sharp) && \text{if } k \leq N \\ M_{k+1}^\sharp &= M_k^\sharp \sqcap F^\sharp(M_k^\sharp) && \text{if } k > N \end{aligned}$$

- join 操作を1回目の反復から行う点で loop unrolling とは異なる

Delaying Widening with Union

- $N = 2$ としてwidening の適用を遅らせる

at rank $k = 0$ $[0, 0]$

at rank $k = 1$ $[0, 0] \sqcup^{\sharp} [-1, 2] = [-1, 2]$

at rank $k = 2$ $[-1, 2] \sqcup^{\sharp} [-1, 4] = [-1, 4]$

at rank $k = 3$ $[-1, 4] \nabla_{\mathcal{V}} [-1, 6] = [-1, +\infty)$

at rank $k = 4$ $[-1, +\infty) \nabla_{\mathcal{V}} [-1, +\infty) = [-1, +\infty)$

```
1 int x = 0;
2 while (rand()) {
3     if (rand()) {
4         x = -1;
5     } else {
6         x = x + 2;
7     }
8 }
```

Widening with Threshold

- Widening をある閾値 (threshold values) で一度止める

$$[n, p] \nabla_{\mathcal{V}} [n, q] = \begin{cases} [n, p] & \text{if } q \leq p \\ [n, B] & \text{if } p < q \leq B \\ [n, +\infty) & \text{if } B < q \end{cases}$$

ex.) threshold $B = 50$ とする

at rank $k = 0$ $[0, 0]$

at rank $k = 1$ $[0, 0] \nabla_{\mathcal{V}} [0, 1] = [0, 50]$

at rank $k = 2$ $[0, 50] \nabla_{\mathcal{V}} [0, 50] = [0, 50]$

```
1 int x = 0;
2 while (x <= 100) {
3     if (x >= 50) {
4         x = 10;
5     } else {
6         x = x + 1;
7     }
8 }
```

Contents

5.1 Construction of Abstract Domains

5.2 Advanced Iteration Techniques

- Loop Unrolling
- Widening の精度を上げる
- 最小不動点を用いた抽象化の精度を上げる

5.3 Sparse Analysis

5.4 Modular Analysis

5.5 Backward Analysis

5.2.3 Refinement of an Abstract Approximation

- 反復の状態が収束した後に結果の精度を上げる

3章の復習

$$[\![\text{while}(B)\{C\}]\!]_{\mathcal{P}}(M) = \mathcal{F}_{\neg B}(M_{\text{loop}})$$

$$M_{\text{loop}} = \bigcup_{i \geq 0} F^i(\gamma(M^\sharp)) = \text{lfp } G$$

$$G(X) = \gamma(M^\sharp) \cup F(X)$$

$$F = [\![C]\!]_{\mathcal{P}} \circ \mathcal{F}_B$$

$$\text{抽象化: } G^\sharp(X) = M^\sharp \sqcup^\sharp F^\sharp(X) \quad (G \circ \gamma \subseteq \gamma \circ G^\sharp)$$

5.2.3 Refinement of an Abstract Approximation

- M_{lim}^\sharp を M_{loop} の過大近似とする
- G は単調で $M_{\text{loop}} \subseteq \gamma(M_{\text{lim}}^\sharp)$ より, $G(M_{\text{loop}}) \subseteq G(\gamma(M_{\text{lim}}^\sharp))$
- M_{loop} は G の不動点 $G(M_{\text{loop}}) = M_{\text{loop}}$ より, $M_{\text{loop}} \subseteq G(\gamma(M_{\text{lim}}^\sharp))$
- $G \circ \gamma \subseteq \gamma \circ G^\sharp$ より, $G(\gamma(M_{\text{lim}}^\sharp)) \subseteq \gamma(G^\sharp(M_{\text{lim}}^\sharp))$
- 以上より, $M_{\text{loop}} \subseteq \gamma(G^\sharp(M_{\text{lim}}^\sharp))$

$$G^\sharp(X) = M^\sharp \sqcup^\sharp F^\sharp(X)$$

➡ M_{lim}^\sharp にもう一度反復の操作をしても健全性が保たれる
(この操作は解析の精度を上げることが多い)

5.2.3 Refinement of an Abstract Approximation

```
1 int x = 0;
2 while (rand() && x < 50) {
3     x = x + 1;
4 }
```

Widening を使うと
 $M_{\text{lim}}^\sharp = \{x \mapsto [0, +\infty)\}$
→ この精度を上げたい

もう一度反復の操作を行うと

$$F^\sharp(M_{\text{lim}}^\sharp) = \llbracket x = x + 1 \rrbracket_\mathcal{P}^\sharp \circ \mathcal{F}_{\text{rand}() \wedge x < 50}^\sharp(M_{\text{lim}}^\sharp) = \{x \mapsto [1, 50]\}$$

ループの開始状態 $\{x \mapsto [0, 0]\}$ と join して

$$\{x \mapsto [0, 50]\}$$

→ 最も精度の高い範囲になった

5.2.3 Refinement of an Abstract Approximation

この操作の制限

```
1 int x = 0;
2 int y = 0;
3 while (rand() && x < 50) {
4     if (rand())
5         y = x;
6     x = x + 1;
7 }
```

Widening を使うと

$$\{x \mapsto [0, +\infty), y \mapsto [0, +\infty)\}$$

同様の方法で精度を上げる

$$\{x \mapsto [0, 50], y \mapsto [0, +\infty)\}$$

→ y の範囲の精度が上がらない

- $\{x \mapsto [0, 50], y \mapsto [0, +\infty)\}$ を具体化したものは G の不動点
- この操作は結果を G の任意の不動点にする
→ 閾値を使って精度を上げるしかない



天皇賞 (秋)

Contents

5.1 Construction of Abstract Domains

5.2 Advanced Iteration Techniques

5.3 Sparse Analysis

- Spatial Sparsity
- Temporal Sparsity
- Def-Use Chain

5.4 Modular Analysis

5.5 Backward Analysis

5.3 Sparse Analysis

- ・ 解析精度を落とさずにコスト（メモリや時間消費）を減らしたい
- ・ 2つの特徴を利用して解析コストを削減する：sparse analysis
 - ・ Spatial sparsity プログラムの各部分はメモリのほんの一部分しかアクセスしない
 - ・ Temporal sparsity メモリの書き込みから使用までは間が空く

5.3 Sparse Analysis

Sparse analysis の特徴

- ・ 解析の基本的な定義に依存しない
→ 基本的な抽象解釈を定義した後にこの手法を付け足す
- ・ 各プログラムラベルでの状態を見たいときに効果的

解析結果をプログラムラベルから抽象メモリ状態とする

$$\mathbb{L} \rightarrow (\mathbb{A}^\sharp \rightarrow \mathbb{V}^\sharp)$$

Contents

5.1 Construction of Abstract Domains

5.2 Advanced Iteration Techniques

5.3 Sparse Analysis

- Spatial Sparsity
- Temporal Sparsity
- Def-Use Chain

5.4 Modular Analysis

5.5 Backward Analysis

5.3.1 Exploiting Spatial Sparsity

Spatial sparsity を利用した解析 (abstract garbage collection, frame rule)

- ・ プログラムの各部分で使われるメモリの一部分だけを保存する
→ 解析のメモリコストが抑えられる

Abstract semantic function

$$F_{\text{sparse}}^{\sharp} : (\mathbb{L} \rightarrow \mathbb{M}_{\text{sparse}}^{\sharp}) \rightarrow (\mathbb{L} \rightarrow \mathbb{M}_{\text{sparse}}^{\sharp})$$

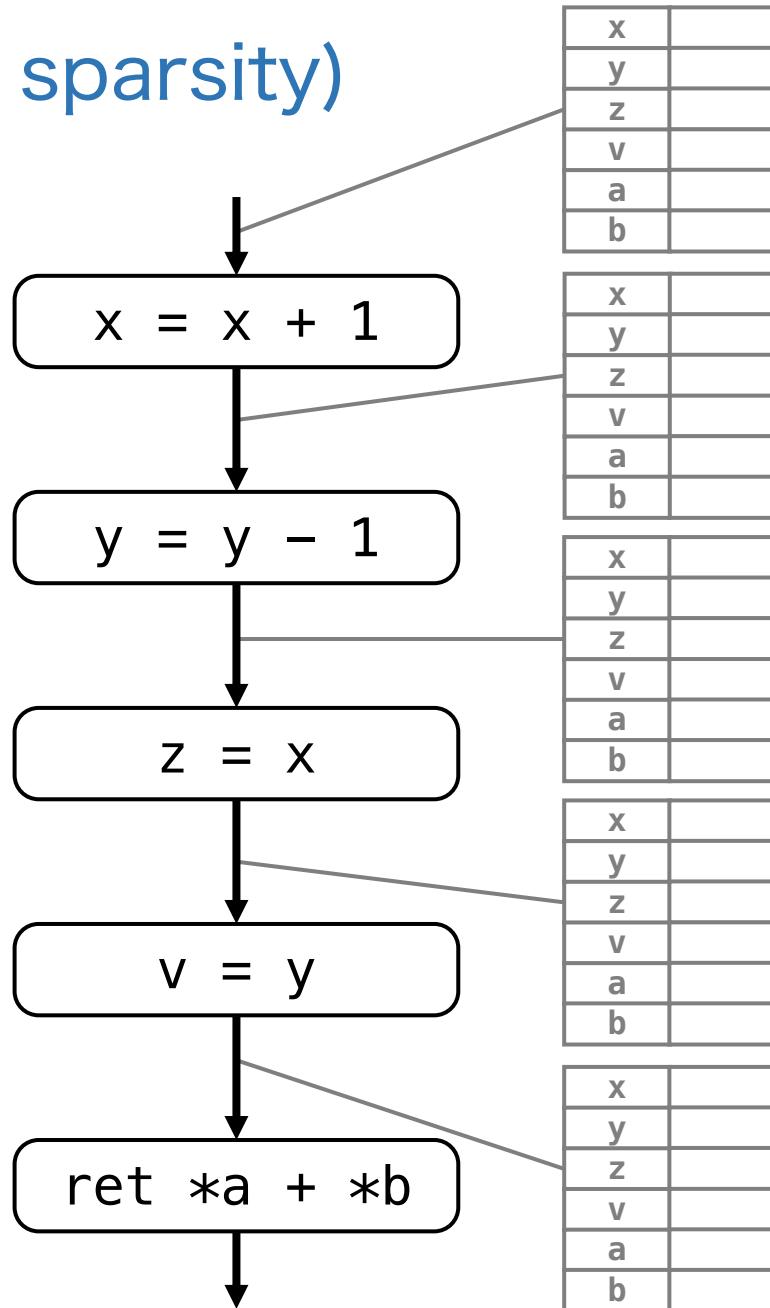
$$\mathbb{M}_{\text{sparse}}^{\sharp} = \{ M^{\sharp} \in \mathbb{M}^{\sharp} \mid \text{dom}(M^{\sharp}) = \text{Access}^{\sharp}(l), l \in \mathbb{L} \} \cup \{ \perp \}$$

$\text{dom}(M^{\sharp})$: M^{\sharp} のエントリ

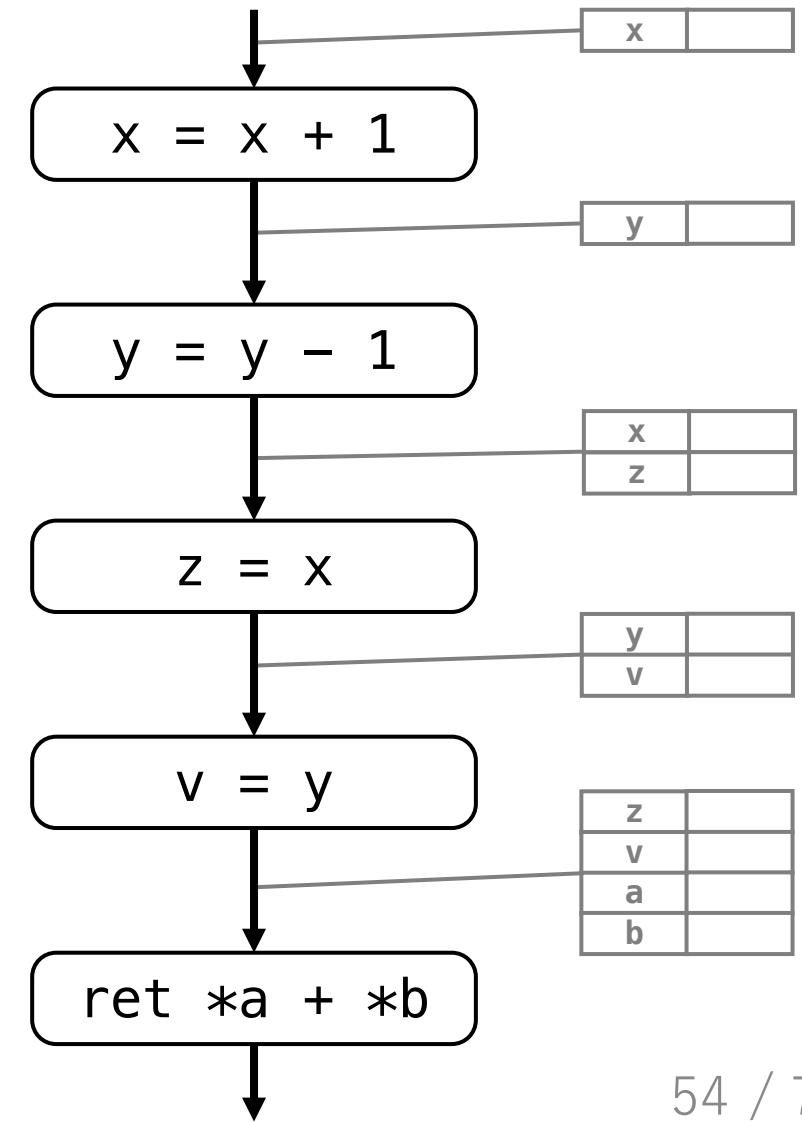
$\text{Access}^{\sharp}(l)$: l でアクセスされる抽象場所 (変数)
→ 健全な事前の解析で計算

Example (Spatial sparsity)

```
1 a = &v;  
2 b = &z;  
3 x = x + 1;  
4 y = y - 1;  
5 z = x;  
6 v = y;  
7 ret *a + *b;
```



Spatial sparsity



Contents

5.1 Construction of Abstract Domains

5.2 Advanced Iteration Techniques

5.3 Sparse Analysis

- Spatial Sparsity
- Temporal Sparsity
- Def-Use Chain

5.4 Modular Analysis

5.5 Backward Analysis

5.3.2 Exploiting Temporal Sparsity

Temporal sparsity を利用した解析

- ・メモリエントリ (変数) が変化したとき, その使用場所までスキップする
→ 解析の時間的コストが抑えられる
- ・def-use chain を事前の解析で求めておく
- ・変数が定義されたラベルからそれが読み込まれるラベルへの関係

$$(l, M^\#) \rightarrow^\# (l', M'^\#) \quad \text{for } l' \in \text{next}^\#(l, M^\#)$$

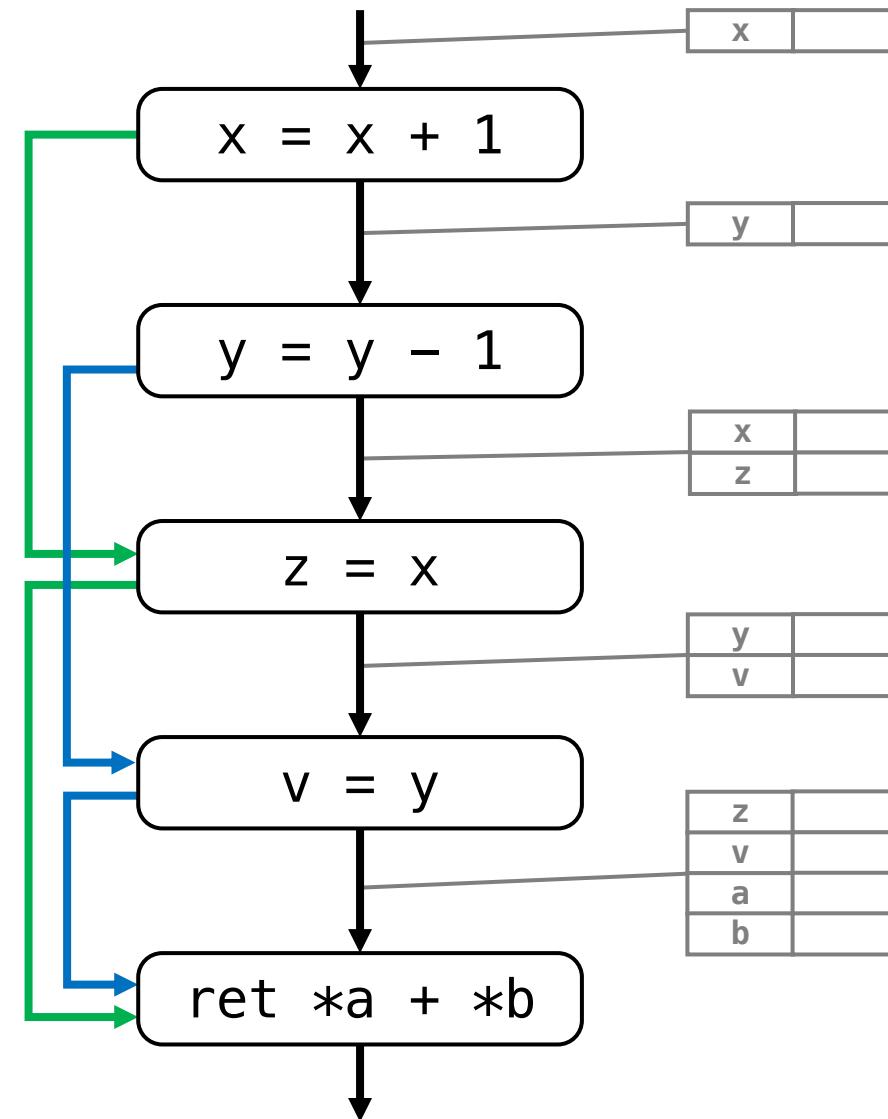
$$\text{next}^\#(l, M^\#)$$

変数 $dom(M^\#)$ の定義場所 l から使用場所 l' への def-use 関係

Example (Temporal sparsity)

```
1 a = &v;  
2 b = &z;  
3 x = x + 1;  
4 y = y - 1;  
5 z = x;  
6 v = y;  
7 ret *a + *b;
```

Spatial and
temporal sparsity



Contents

5.1 Construction of Abstract Domains

5.2 Advanced Iteration Techniques

5.3 Sparse Analysis

- Spatial Sparsity
- Temporal Sparsity
- Def-Use Chain

5.4 Modular Analysis

5.5 Backward Analysis

5.3.3 Precision-Preserving Def-Use Chain by Pre-Analysis

$D^\sharp(l), U^\sharp(l)$ non-sparse 解析でのラベル l で定義と使用される抽象場所 (変数) の集合

$D_{pre}^\sharp, U_{pre}^\sharp$ 事前解析によって計算される, 定義と使用の抽象場所 (変数) の集合

$D_{pre}^\sharp, U_{pre}^\sharp$ が安全 (safe) であるとは

- $D_{pre}^\sharp, U_{pre}^\sharp$ はそれぞれ D^\sharp, U^\sharp を過大近似する

$$\forall l \in \mathbb{L} : D_{pre}^\sharp(l) \supseteq D^\sharp(l) \quad \text{and} \quad U_{pre}^\sharp(l) \supseteq U^\sharp(l)$$

- D_{pre}^\sharp の偽定義は U_{pre}^\sharp に含まれる

$$\forall l \in \mathbb{L} : U_{pre}^\sharp(l) \supseteq D_{pre}^\sharp(l) \setminus D^\sharp(l)$$

5.3.3 Precision-Preserving Def-Use Chain by Pre-Analysis

Def-use chain information from pre-analysis

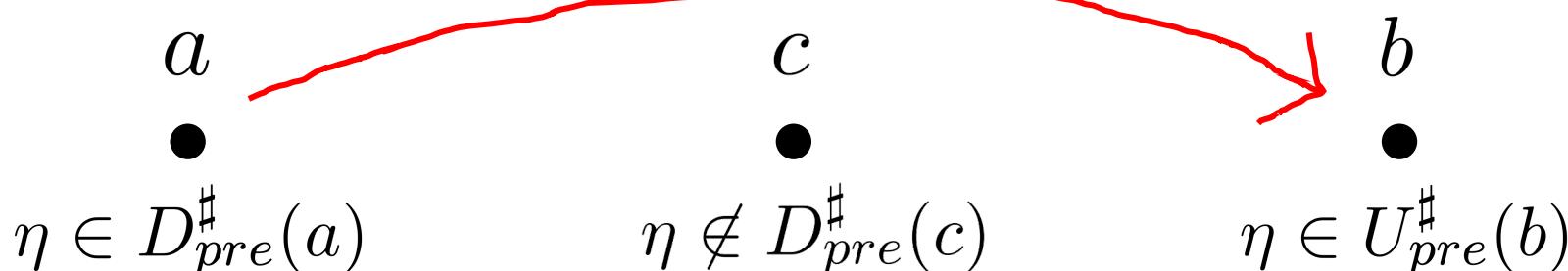
ラベル a, b が def-use chain を持つとは

- 抽象場所 (変数) η に対して以下が成り立つ

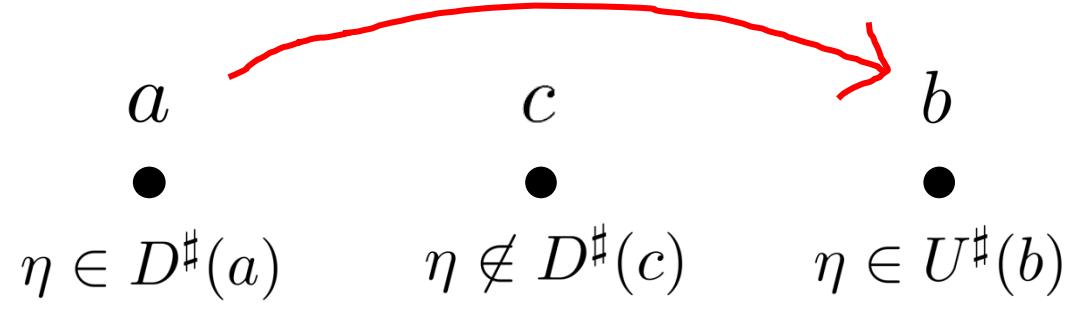
$$\eta \in D_{pre}^\sharp(a), \eta \in U_{pre}^\sharp(b)$$

- a, b 間の実行パス上のすべてのラベル c で η が再定義されない

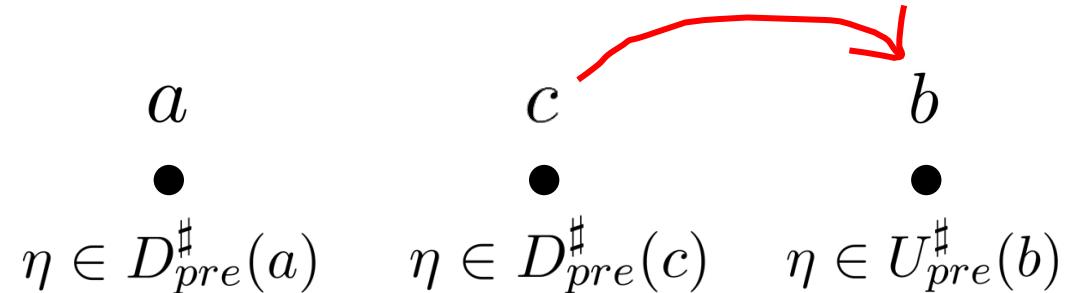
$$\eta \notin D_{pre}^\sharp(c)$$



5.3.3 Precision-Preserving Def-Use Chain by Pre-Analysis

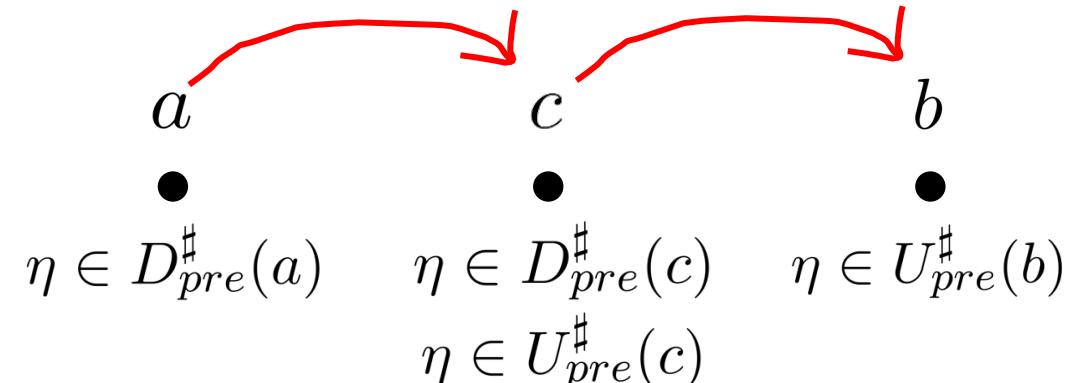


張られるべき def-use edge



偽定義によって間違った
def-use edge が張られる

D_{pre}^\sharp の偽定義は U_{pre}^\sharp に含まれる
 $\forall l \in \mathbb{L} : U_{pre}^\sharp(l) \supseteq D_{pre}^\sharp(l) \setminus D^\sharp(l)$



安全な def-use chain では
a から b への edge が
回復する

Contents

5.1 Construction of Abstract Domains

5.2 Advanced Iteration Techniques

5.3 Sparse Analysis

5.4 Modular Analysis

5.5 Backward Analysis

5.4 Modular Analysis

Modular Analysis : 関数などプログラムの構成要素を別々に解析し,
その結果をまとめる cf.) コンパイラ
→ スケーラビリティの向上

手順

1. Calling context (初期状態) をシンボル化する
2. 解析したい条件を満たすようなシンボルの制約を求める
3. それらの制約を合わせてプログラム全体が安全な動作をするか確認する

- ・スケーラビリティは各手続きの推論を処理する方法に依存
- ・ある手続きが更新されたとき, それに依存する部分のみを変更すれば良い

Example (Modular analysis)

解析の目標： buffer の範囲外アクセスの検知

```
1 void set_i(int *arr, int index) {  
2     arr[index] = 0;  
3 }  
4  
5 int *int_malloc_wrapper(int n) {  
6     return malloc(n * sizeof(int));  
7 }
```

```
8 void interprocedural(){  
9     int *arr = int_malloc_wrapper(9);  
10    int i;  
11    for (i = 0; i < 9; i++) {  
12        set_i(arr, i);  
13        set_i(arr, i + 1);  
14    }  
15 }
```

Example (Modular analysis)

解析の目標： buffer の範囲外アクセスの検知

```
1 void set_i(int *arr, int index) {  
2     arr[index] = 0;  
3 }
```

$\text{arr} \mapsto (\text{offset} : [\mathbf{s}_0, \mathbf{s}_1], \text{size} : [\mathbf{s}_2, \mathbf{s}_3])$

$\text{index} \mapsto [\mathbf{s}_4, \mathbf{s}_5]$

満たすべき条件

index が buffer の size を超えない

$$[\mathbf{s}_0 + \mathbf{s}_4, \mathbf{s}_1 + \mathbf{s}_5] < [\mathbf{s}_2, \mathbf{s}_3]$$

$$(0 \leq \mathbf{s}_0 + \mathbf{s}_4 \wedge \mathbf{s}_1 + \mathbf{s}_5 < \mathbf{s}_2)$$

Example (Modular analysis)

解析の目標： buffer の範囲外アクセスの検知

```
5 int *int_malloc_wrapper(int n) {  
6     return malloc(n * sizeof(int));  
7 }
```

$$n \mapsto [s_6, s_7]$$

$$\text{ret} \mapsto (\text{offset} : [0, 0], \text{size} : [s_6, s_7])$$

Example (Modular analysis)

実際の calling context が条件を満たすか確認する

```
8 void interprocedural(){
9     int *arr = int_malloc_wrapper(9);
10    int i;
11    for (i = 0; i < 9; i++) {
12        set_i(arr, i);
13        set_i(arr, i + 1);
14    }
15 }
```

arr \mapsto (offset : [0, 0], size : [9, 9])
index \mapsto [1, 9]
条件 $[0 + 1, 0 + 9] \not< [9, 9]$

n \mapsto [9, 9]
ret \mapsto (offset : [0, 0], size : [9, 9])

arr \mapsto (offset : [0, 0], size : [9, 9])
index \mapsto [0, 8]
条件 $[0 + 0, 0 + 8] < [9, 9]$

安全違反

Contents

- 5.1 Construction of Abstract Domains
- 5.2 Advanced Iteration Techniques
- 5.3 Sparse Analysis
- 5.4 Modular Analysis
- 5.5 Backward Analysis

5.5.1 Forward Semantics and Backward Semantics

- 今まで：事前条件から事後条件を過大近似（forward）
この章：事後条件から事前条件を過大近似（backward）
- 状態から結果への関数
- 結果からそれを導く状態の集合への関数（backward semantics）

$$\llbracket B \rrbracket(m) = \nu \in \{\text{true}, \text{false}\}$$

$$\llbracket B \rrbracket_{\text{bwd}}(\nu) = \{m \in \mathbb{M} \mid \llbracket B \rrbracket(m) = \nu\}$$

$$\mathcal{F}_B(M) = M \cap \llbracket B \rrbracket_{\text{bwd}}(\text{true})$$

5.5.1 Forward Semantics and Backward Semantics

- 実数式の backward semantics $\mathbb{V} \longrightarrow \wp(\mathbb{M})$

$$\llbracket E \rrbracket_{\mathbf{bwd}}(\nu) = \{m \in \mathbb{M} \mid \llbracket E \rrbracket(m) = \nu\}$$

- 命令の backward semantics $\wp(\mathbb{M}) \longrightarrow \wp(\mathbb{M})$

出力状態の集合から入力状態の集合への関数

$$\begin{aligned}\llbracket C \rrbracket_{\mathbf{bwd}}(M) &= \{m \in \mathbb{M} \mid \exists m' \in \llbracket C \rrbracket(\{m\}), m' \in M\} \\ &= \{m \in \mathbb{M} \mid \llbracket C \rrbracket(\{m\}) \cap M \neq \emptyset\}\end{aligned}$$

- 上記の定義は M になるかもしれない初期状態の集合
 \leftrightarrow 必ず M になる初期状態の集合を考えることもできる

5.5.2 Backward Analysis and Applications

- どのように与えられた状態に到達したかを確認する

```
1 int x, y;  
2 input(x);  
3 if (x > 0) {  
4     y = x;  
5 } else {  
6     y = -x;  
7 }
```

- 事後状態 $y \mapsto [2, 5]$ のとき
 input の直後 $x \mapsto [-5, -2] \cup [2, 5]$
- 事後状態 $y \mapsto (-\infty, -3]$ のとき $x \mapsto \perp$
→ これを満たす事前状態が存在しない

- ある動作が起こる必要条件やある動作が起こらない十分条件が得られる
→ プログラムの理解に役立つ

Definition of a Backward Analysis

命令の backward abstract semantics

- skip $\llbracket \text{skip} \rrbracket_{\text{bwd}}^\sharp(M^\sharp) = M^\sharp$
- 命令の連接 $\llbracket C_0; C_1 \rrbracket_{\text{bwd}}^\sharp(M^\sharp) = \llbracket C_0 \rrbracket_{\text{bwd}}^\sharp(\llbracket C_1 \rrbracket_{\text{bwd}}^\sharp(M^\sharp))$
- 条件分岐
 $\llbracket \text{if}(B)\{C_0\} \text{else}\{C_1\} \rrbracket_{\text{bwd}}^\sharp(M^\sharp) = \mathcal{F}_B^\sharp(\llbracket C_0 \rrbracket_{\text{bwd}}^\sharp(M^\sharp)) \sqcup^\sharp \mathcal{F}_{\neg B}^\sharp(\llbracket C_1 \rrbracket_{\text{bwd}}^\sharp(M^\sharp))$
- ループ

forward 解析の不動点を使った計算と同じようにできる

Definition of a Backward Analysis

命令の backward abstract semantics

- 代入 $x := E$

- E に x が含まれないとき (non-invertible)

$\mathcal{F}_{x=E}^\sharp$ を計算して x の制約をなくす

- E に x が含まれるとき (invertible)

事前状態の x の制約は事後状態の x の制約に影響する

Q $\llbracket x := y \rrbracket_{\mathbf{bwd}}^\sharp(\{x \mapsto [0, 10], y \mapsto [5, 15]\}) =$

$$\llbracket x := x - 5 \rrbracket_{\mathbf{bwd}}^\sharp(\{x \mapsto [2, 8]\}) =$$

Definition of a Backward Analysis

命令の backward abstract semantics

- 代入 $x := E$

- E に x が含まれないとき (non-invertible)

- $\mathcal{F}_{x=E}^\sharp$ を計算して x の制約をなくす

- E に x が含まれるとき (invertible)

- 事前状態の x の制約は事後状態の x の制約に影響する

Q $\llbracket x := y \rrbracket_{\mathbf{bwd}}^\sharp(\{x \mapsto [0, 10], y \mapsto [5, 15]\}) = \{x \mapsto \top, y \mapsto [5, 10]\}$

$$\llbracket x := x - 5 \rrbracket_{\mathbf{bwd}}^\sharp(\{x \mapsto [2, 8]\}) = \{x \mapsto [7, 13]\}$$

Definition of a Backward Analysis

while の例

```
1 while (x < 2) {  
2     x = x + 1;  
3 }
```

事後状態

$$M^\sharp = \{x \mapsto [2, 8]\}$$

3章の復習

$$[\![\text{while}(B)\{C\}]\!]_{\mathcal{P}}^\sharp(M) = \mathcal{F}_{\neg B}^\sharp \left(\text{abs_iter}([\![C]\!]_{\mathcal{P}}^\sharp \circ \mathcal{F}_B^\sharp, M^\sharp) \right)$$

$$\text{abs_iter}(F^\sharp, M^\sharp)$$

$$M_0^\sharp = M^\sharp$$

$$M_{k+1}^\sharp = M_k^\sharp \nabla F^\sharp(M_k^\sharp)$$

これを収束するまで繰り返す

$$\mathcal{F}_{x \geq 2}^\sharp(M^\sharp) = \{x \mapsto [2, 8]\}$$

$$[\![x := x + 1]\!]_{\text{bwd}}^\sharp(\{x \mapsto [2, 8]\}) = \{x \mapsto [1, 7]\}$$

2回目以降の反復でも
同じ状態に収束する

$$\mathcal{F}_{x < 2}^\sharp(\{x \mapsto [1, 7]\}) = \{x \mapsto [1, 1]\}$$

$$\{x \mapsto [2, 8]\} \nabla \{x \mapsto [1, 1]\} = \{x \mapsto (-\infty, 8]\}$$



5.3.3 Combining Forward and Backward Analysis

- forward 解析と backward 解析を組み合わせて解析の精度を上げる

```
1 ...  
2 if (y <= x) {  
3   ...  
4   if (x <= 4) {  
5     ...  
6     if (5 <= y) {  
7       ...
```

	区間抽象	凸多角形抽象
1	{ $x \mapsto \top, y \mapsto \top$ }	\top
3	{ $x \mapsto \top, y \mapsto \top$ }	$y \leq x$
5	{ $x \mapsto (-\infty, 4], y \mapsto \top$ }	$y \leq x \wedge x \leq 4$
7	{ $x \mapsto (-\infty, 4], y \mapsto [5, +\infty)$ }	\perp

- non-relational ドメインでは 7行目が実行されないことを表現できない

5.3.3 Combining Forward and Backward Analysis

- forward 解析と backward 解析を何度も反復させることで精度が上がる
- local iterations : 局所的な解析の精度を上げる手法

```
1 ...  
2 if (y <= x) {  
3 ...  
4 if (x <= 4) {  
5 ...  
6 if (5 <= y) {  
7 ...
```

	1st forward	1st backward	2nd forward
1	$\{x \mapsto \top, y \mapsto \top\}$	\perp	\perp
3	$\{x \mapsto \top, y \mapsto \top\}$	$\{x \mapsto (-\infty, 4], y \mapsto [5, +\infty)\}$	\perp
5	$\{x \mapsto (-\infty, 4], y \mapsto \top\}$	$\{x \mapsto (-\infty, 4], y \mapsto [5, +\infty)\}$	\perp
7	$\{x \mapsto (-\infty, 4], y \mapsto [5, +\infty)\}$	$\{x \mapsto (-\infty, 4], y \mapsto [5, +\infty)\}$	\perp

まとめ

これまでの章で学んだ解析手法を発展させる

- Abstract domains

より良い抽象ドメインの設計方法

- Iteration techniques

解析コストを下げる, 精度を上げる反復手法

- Scalability techniques

時間的, 空間的なコストを下げる解析手法

- Analysis direction

逆向きの解析手法