

EFFICIENT STATE MERGING IN SYMBOLIC EXECUTION

[PLDI'12]

Volodymyr Kuznetsov
et. al.

0. ABSTRACTION

記号実行はテストケース生成とバグ検出に実用的

- しかし、現実世界のプログラムでは状態爆発でスケールしない
 - 解法の一つが**State Merge (状態マージ)**
 - 状態 = パス数は減るが、ソルバの負担は増大
 - 結果マージ前よりも非効率になる可能性も有

アプローチ：

- **Query Count Estimation** (クエリ数推定)
 - ソルバクエリへの影響が少ない（記号）変数を静的に推定
- **Dynamic State Merging** (動的状態マージ)
 - 任意の探索戦略に相互作用するマージ可能状態の検索

評価：

- 96個のGNU Core Utilitiesでスピードアップ

1. INTRODUCTION

EXE[5], KLEE[6], S2E[7], DART[18]などが記号実行の優位性を提示

記号実行の優位な特徴

- ①動的解析の要素（システムコールなどを具体化可能）
- ②静的解析の要素（各実行を集約可能）
- ③セマンティクスについて完全（抽象化がなければ誤検出なし）
- ④SAT, SMTソルバがメッチャ良くなつた（特徴かこれ？）

が、状態爆発によりスケールしない

- 一般に分岐数に対して指数関数的

さらに、正確性を保つためには全状態を保つ必要有

[5] EXE: Automatically generating inputs of death. [CCS'06]

[6] KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. [SOSP'08]

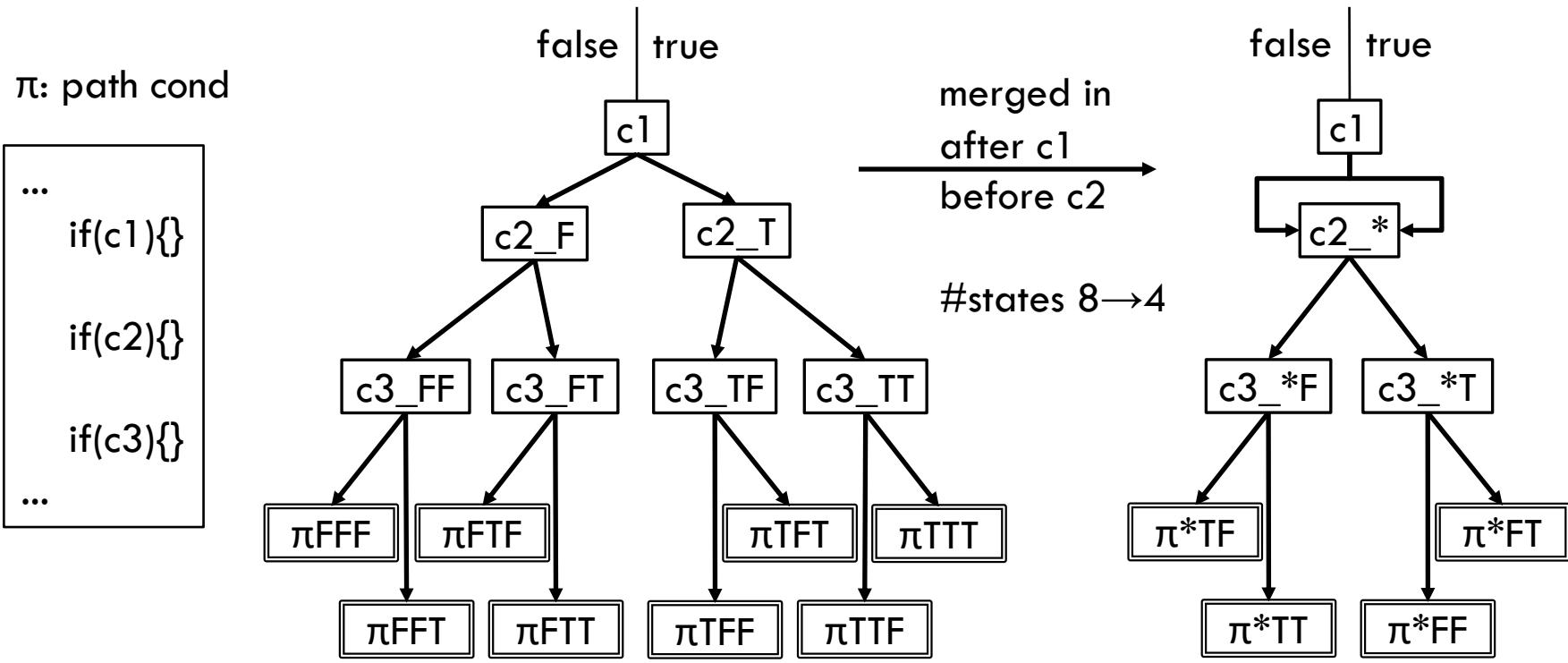
[7] S2E: A platform for in-vivo multi-path analysis of software systems. [ASPLOS'11]

[18] DART: Directed automated random testing. [PLDI'05]

1. INTRODUCTION

State Merge (状態マージ) :

- ・状態爆発に対する解法の一つ
- ・特定のプログラムロケーションでエンジン状態を統合
 - ・マージにより状態数は最大で半減 → 指数的に状態数を減らせる



- ・しかし、欠点も有り

1. INTRODUCTION

マージの欠点

- ソルバへの負担が増加

$a = (b > 0) ? 0 : 2;$

$c = (a > 0) ? a : -1;$

$S1 = \{\sigma[a \rightarrow 0], \pi = (b > 0)\}^*, S1' = \{\sigma[a \rightarrow 2], \pi = !(b > 0)\}^*$

$S2_S1 = \{\sigma[a \rightarrow 0, c \rightarrow -1], \pi = (b > 0) \wedge (a > 0)\}^* // \text{unsat}$

$S2'_S1 = \{\sigma[a \rightarrow 0, c \rightarrow -1], \pi = (b > 0) \wedge !(a > 0)\}^*$

$S2_S1' = \{\sigma[a \rightarrow 2, c \rightarrow 2], \pi = !(b > 0) \wedge (a > 0)\}^*$

$S2'_S1' = \{\sigma[a \rightarrow 2, c \rightarrow 2], \pi = !(b > 0) \wedge !(a > 0)\}^* // \text{unsat}$

- ite (if-then-else) 式を解決するためには別途クエリが必要
 - #query (no merge, eager) : $2 + 2 * 2 = 6$

1. INTRODUCTION

マージの欠点

- ソルバへの負担が増加

```
a = ( b > 0 )? 0: 2;  
//merge 1
```

```
c = ( a > 0 )? a: -1;  
//merge 2
```

$S1 = \{\sigma[a \rightarrow 0], \pi = (b > 0)\}^*$, $S1' = \{\sigma[a \rightarrow 2], \pi = !(b > 0)\}^*$

- ite (if-then-else) 式を解決するためには別途クエリが必要
 - #query (no merge, eager) : $2 + 2 * 2 = 6$

1. INTRODUCTION

マージの欠点

- ソルバへの負担が増加

```
a = ( b > 0 )? 0: 2;
//merge 1
```

```
c = ( a > 0 )? a: -1;
//merge 2
```

$$\begin{aligned} S1 &= \{\sigma[a \rightarrow 0], \pi = (b > 0)\}^*, S1' = \{\sigma[a \rightarrow 2], \pi = !(b > 0)\}^* \\ S1m &= \{\sigma[a \rightarrow \text{ite}(b > 0, 0, 2)^*], \pi = (b > 0) \vee !(b > 0)^* = \text{true}\} \end{aligned}$$

- ite (if-then-else) 式を解決するためには別途クエリが必要
 - #query (no merge, eager) : $2 + 2^* 2 = 6$

1. INTRODUCTION

マージの欠点

- ソルバへの負担が増加

```
a = ( b > 0 )? 0: 2;
//merge 1

c = ( a > 0 )? a: -1;
//merge 2
```

$$\begin{aligned} S1 &= \{\sigma[a \rightarrow 0], \pi = (b>0)^*\}, S1' = \{\sigma[a \rightarrow 2], \pi = !(b>0)^*\} \\ S1m &= \{\sigma[a \rightarrow \text{ite}(b>0, 0, 2)^*], \pi = (b>0) \vee !(b>0)^* = \text{true}\} \\ &\downarrow \\ S2 &= \{\sigma[a \rightarrow \text{ite}(b>0, 0, 2)^*, c \rightarrow a^*], \pi = (a>0)^{**}\}, \\ S2' &= \{\sigma[a \rightarrow \text{ite}(b>0, 0, 2)^*, c \rightarrow -1], \pi = !(a>0)^{**}\} \end{aligned}$$

- ite (if-then-else) 式を解決するためには別途クエリが必要
 - #query (no merge, eager) : $2+2^*2 = 6$

1. INTRODUCTION

マージの欠点

- ソルバへの負担が増加

```
a = ( b > 0 )? 0: 2;
//merge 1
```

```
c = ( a > 0 )? a: -1;
//merge 2
```

$S1 = \{\sigma[a \rightarrow 0], \pi = (b>0)^*\}, S1' = \{\sigma[a \rightarrow 2], \pi = !(b>0)^*\}$

$S1m = \{\sigma[a \rightarrow \text{ite}(b>0, 0, 2)^*], \pi = (b>0) \vee !(b>0)^* = \text{true}\}$

\downarrow

$S2 = \{\sigma[a \rightarrow \text{ite}(b>0, 0, 2)^*, c \rightarrow a^*], \pi = (a>0)^{**}\},$

$S2' = \{\sigma[a \rightarrow \text{ite}(b>0, 0, 2)^*, c \rightarrow -1], \pi = !(a>0)^{**}\}$

$S2m = \{\sigma[a \rightarrow \text{ite}(b>0, 0, 2)^*, c \rightarrow \text{ite}(a>0, a, -1)^{***}],$

$\pi = (a>0) \vee !(a>0)^{***} = \text{true}\}$

- ite (if-then-else) 式を解決するためには別途クエリが必要
 - #query (no merge, eager) : $2+2*2 = 6$

1. INTRODUCTION

マージの欠点

- ソルバへの負担が増加

```
a = ( b > 0 )? 0: 2;
//merge 1

c = ( a > 0 )? a: -1;
//merge 2
```

$S1 = \{\sigma[a \rightarrow 0], \pi = (b>0)^*\}, S1' = \{\sigma[a \rightarrow 2], \pi = !(b>0)^*\}$

$S1m = \{\sigma[a \rightarrow \text{ite}(b>0, 0, 2)^*], \pi = (b>0) \vee !(b>0)^* = \text{true}\}$

\downarrow

$S2 = \{\sigma[a \rightarrow \text{ite}(b>0, 0, 2)^*, c \rightarrow a^*], \pi = (a>0)^{**}\},$

$S2' = \{\sigma[a \rightarrow \text{ite}(b>0, 0, 2)^*, c \rightarrow -1], \pi = !(a>0)^{**}\}$

$S2m = \{\sigma[a \rightarrow \text{ite}(b>0, 0, 2)^*, c \rightarrow \text{ite}(a>0, a, -1)^{***}],$

$\pi = (a>0) \vee !(a>0)^{***} = \text{true}\}$

- ite (if-then-else) 式を解決するためには別途クエリが必要
 - #query (no merge, eager) : $2 + 2^* 2 = 6$
 - #query (merged, eager) : #query(eager) + #query(ite) + #query(disj)

$$(2+2) + 12 + 2 = 18 (*)$$

- ソルバは同一クエリ内でiteの結果を共有しないと仮定 (ソルバ、エンジン依存)
- ここではite式は記号値、具体値を含む

1. INTRODUCTION

マージの欠点

- ソルバへの負担が増加

```
a = ( b > 0 )? 0: 2;
//merge 1
```

```
c = ( a > 0 )? a: -1;
//merge 2
```

$S1 = \{\sigma[a \rightarrow 0], \pi = (b>0)^*\}, S1' = \{\sigma[a \rightarrow 2], \pi = !(b>0)^*\}$

$S1m = \{\sigma[a \rightarrow \text{ite}(b>0, 0, 2)^*], \pi = (b>0) \vee !(b>0)^* = \text{true}\}$

\downarrow

$S2 = \{\sigma[a \rightarrow \text{ite}(b>0, 0, 2)^*, c \rightarrow a^*], \pi = (a>0)^{**}\},$

$S2' = \{\sigma[a \rightarrow \text{ite}(b>0, 0, 2)^*, c \rightarrow -1], \pi = !(a>0)^{**}\}$

$S2m = \{\sigma[a \rightarrow \text{ite}(b>0, 0, 2)^*, c \rightarrow \text{ite}(a>0, a, -1)^{***}],$

$\pi = (a>0) \vee !(a>0)^{***} = \text{true}\}$

- ite (if-then-else) 式を解決するためには別途クエリが必要
 - #query (no merge, eager) : $2 + 2^* 2 = 6$
 - #query (merged, eager) : #query(eager) + #query(ite) + #query(disj)
 $(2+2) + 12 + 2 = 18 (*)$
 - ソルバは同一クエリ内でiteの結果を共有しないと仮定 (ソルバ、エンジン依存)
 - ここではite式は記号値、具体値を含む
- 直感：iteで表された変数が後々使用されるほど
マージのデメリットが増大
 - 使用：条件変数となる or 配列インデックスとなる

1. INTRODUCTION

本論文の基本的な直感・課題

- ・ソルバへの負担が少ないマージ ≈ 効率的なマージ
- ・マージ優先と競合する探索戦略との兼ね合いをどうするか
 - such as Maximize Line Coverage

アプローチ：

- Query Count Estimation (QCE: クエリ数推定)
 - 変数が後の条件分岐で何回出てくるかを静的に推定
- Dynamic State Merging (DSM: 動的状態マージ)
 - 任意の探索戦略ワークリストからマージに適した状態を検出

2. TRADE-OFFS IN SYMBOLIC PROGRAM ANALYSIS

2.1 GENERAL SYMBOLIC EXPLORATION

記号探索アルゴリズムのモデル化

- with merge
- 簡単のためにstatementは以下の通り
 - 代入
 - 条件付きgoto
 - assert
 - halt
- 以下の小泉翻案では
A Survey of Symbolic[CSUR'18]に準拠
 - s : 記号実行の状態
 - stmt : 評価文（ロケーション）
 - σ : シンボリックテーブル
 - π : パス条件

Input: Choice function pickNext , similarity relation \sim , branch checker follow , and initial location ℓ_0 .

Data: Worklist w and set of successor states S .

```

1   $w := \{(\ell_0, \text{true}, \lambda v.v)\};$ 
2  while  $w \neq \emptyset$  do
3     $(\ell, pc, s) := \text{pickNext}(w); S := \emptyset;$ 
    // Symbolically execute the next instruction
4    switch  $\text{instr}(\ell)$  do
5      case  $v := e$  // assignment
6         $S := \{(suc(\ell), pc, s[v \mapsto \text{eval}(s, e)])\};$ 
7      case  $\text{if}(e) \text{ goto } \ell'$  // conditional jump
8        if  $\text{follow}(pc \wedge s \wedge e)$  then
9           $S := \{(\ell', pc \wedge e, s)\};$ 
10         if  $\text{follow}(pc \wedge s \wedge \neg e)$  then
11            $S := S \cup \{(\text{suc}(\ell), pc \wedge \neg e, s)\};$ 
12      case  $\text{assert}(e)$  // assertion
13        if  $\text{isSatisfiable}(pc \wedge s \wedge \neg e)$  then abort;
14         $S := \{(\text{suc}(\ell), pc, s)\};$ 
15      case  $\text{halt}$  // program halt
16        print  $pc;$ 
17    // Merge new states with matching ones in  $w$ 
18    forall  $(\ell'', pc', s') \in S$  do
19      if  $\exists (\ell'', pc'', s'') \in w : (\ell'', pc'', s'') \sim (\ell'', pc', s')$  then
20         $w := w \setminus \{(\ell'', pc'', s'')\};$ 
21         $w := w \cup \{(\ell'', pc' \vee pc'', \lambda v.\text{ite}(pc', s'[v], s''[v]))\};$ 
22      else
23         $w := w \cup \{(\ell'', pc', s')\};$ 
24    print "no errors";

```

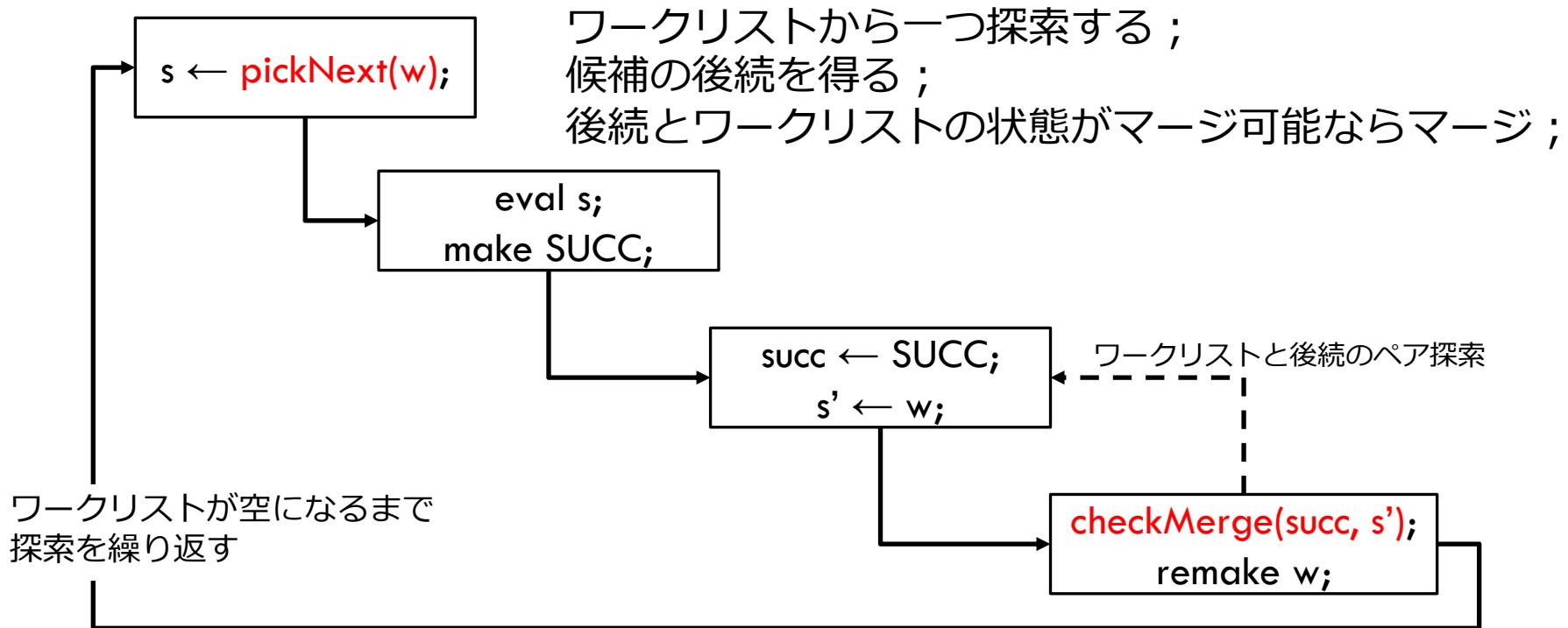
Algorithm 1. Generic symbolic exploration.

2. TRADE-OFFS IN SYMBOLIC PROGRAM ANALYSIS

2.1 GENERAL SYMBOLIC EXPLORATION

マージ有り記号探索アルゴリズムのモデル化

まとめるとこんな感じ



`pickNext`と`checkMerge`を抽象化することで
色々なマージ有り探索アルゴリズムを表現してるだけ

2. TRADE-OFFS IN SYMBOLIC PROGRAM ANALYSIS

2.1 GENERAL SYMBOLIC EXPLORATION

マージ有り記号探索アルゴリズムのモデル化

- ワークリスト w : 未探索エンジン状態の候補群
 - $\text{pickNext}(w)$: ワークリストから次の探索状態を選ぶ関数(= 探索戦略)
 - SUCC : stmt の後に評価されるロケーションの状態の集合

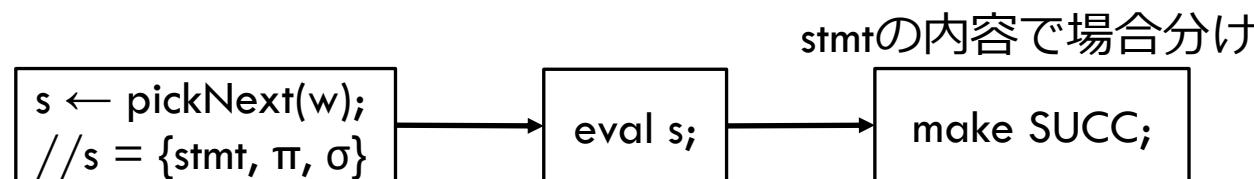


2. TRADE-OFFS IN SYMBOLIC PROGRAM ANALYSIS

2.1 GENERAL SYMBOLIC EXPLORATION

マージ有り記号探索アルゴリズムのモデル化

- ワークリスト w : 未探索エンジン状態の候補群
 - $\text{pickNext}(w)$: ワークリストから次の探索状態を選ぶ関数(= 探索戦略)
- SUCC : stmt の後に評価されるロケーションの状態の集合

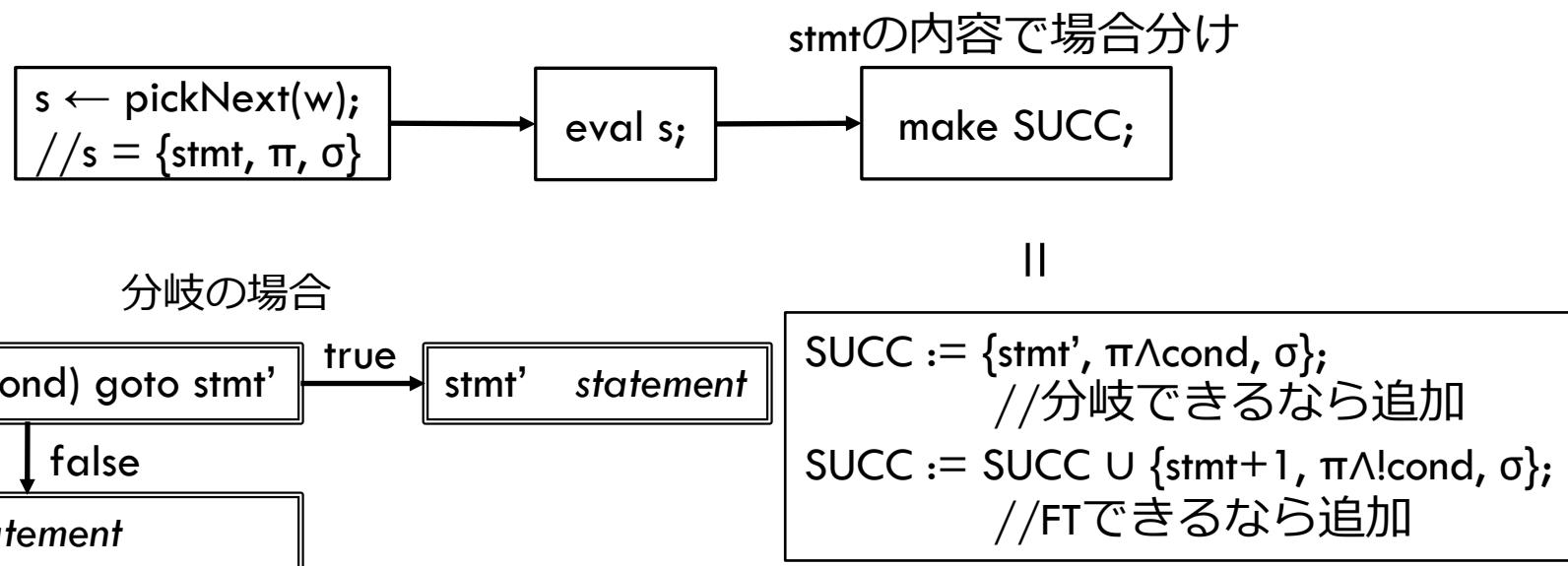


2. TRADE-OFFS IN SYMBOLIC PROGRAM ANALYSIS

2.1 GENERAL SYMBOLIC EXPLORATION

マージ有り記号探索アルゴリズムのモデル化

- ワークリスト w : 未探索エンジン状態の候補群
 - $\text{pickNext}(w)$: ワークリストから次の探索状態を選ぶ関数(= 探索戦略)
- SUCC : stmt の後に評価されるロケーションの状態の集合

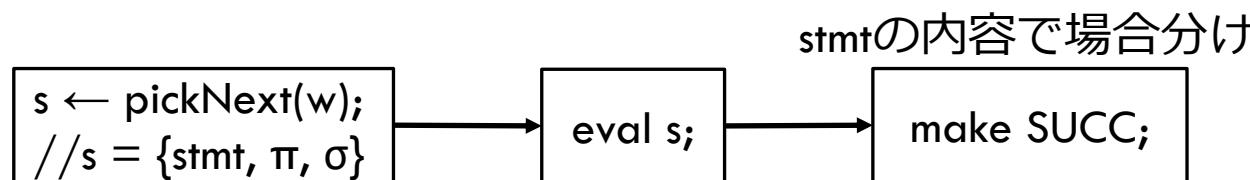


2. TRADE-OFFS IN SYMBOLIC PROGRAM ANALYSIS

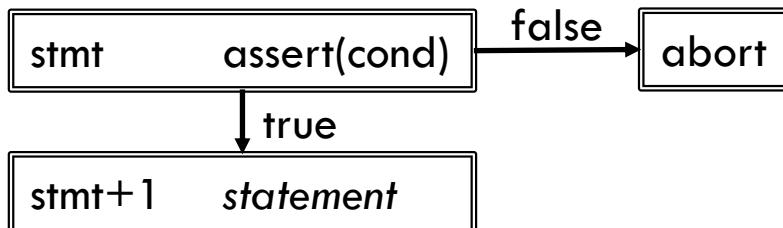
2.1 GENERAL SYMBOLIC EXPLORATION

マージ有り記号探索アルゴリズムのモデル化

- ワークリスト w : 未探索エンジン状態の候補群
 - $\text{pickNext}(w)$: ワークリストから次の探索状態を選ぶ関数(= 探索戦略)
- SUCC : stmt の後に評価されるロケーションの状態の集合



assertの場合



||

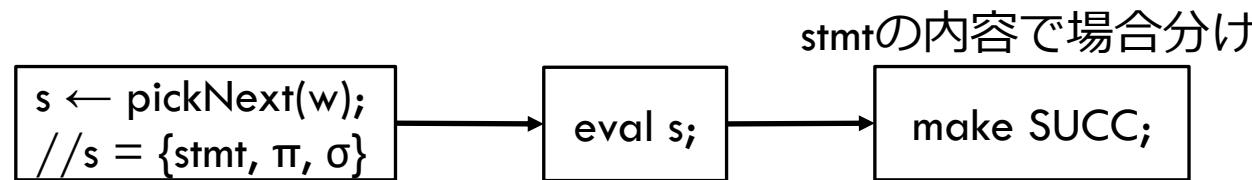
$\text{SUCC} := \{\text{stmt}+1, \pi, \sigma\};$

2. TRADE-OFFS IN SYMBOLIC PROGRAM ANALYSIS

2.1 GENERAL SYMBOLIC EXPLORATION

マージ有り記号探索アルゴリズムのモデル化

- ワークリスト w : 未探索エンジン状態の候補群
 - $\text{pickNext}(w)$: ワークリストから次の探索状態を選ぶ関数(= 探索戦略)
- SUCC : stmt の後に評価されるロケーションの状態の集合



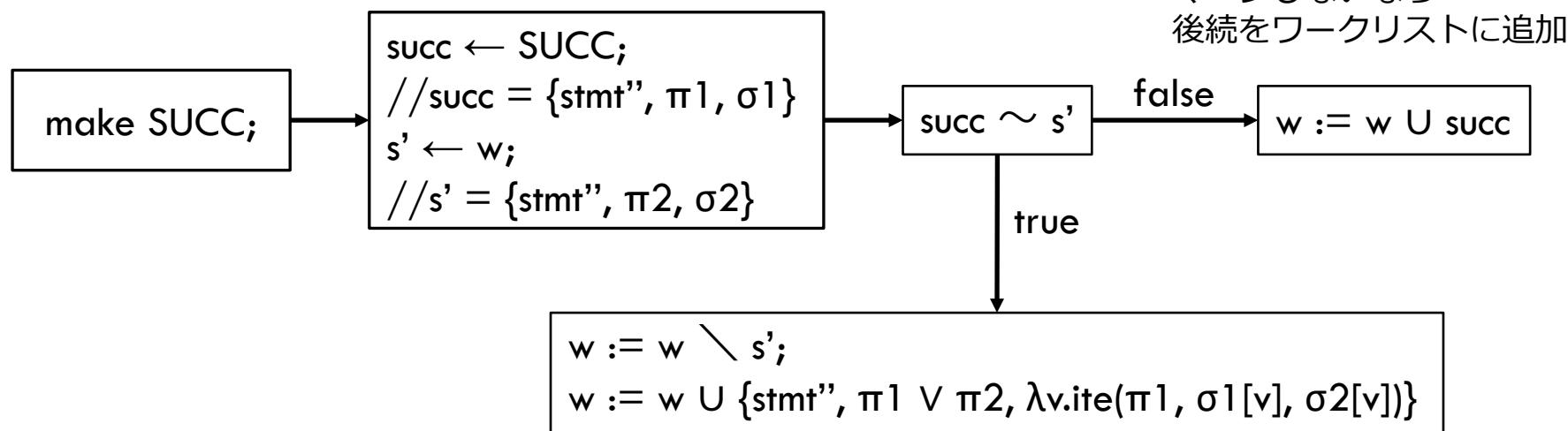
2. TRADE-OFFS IN SYMBOLIC PROGRAM ANALYSIS

2.1 GENERAL SYMBOLIC EXPLORATION

マージ有り記号探索アルゴリズムのモデル化

- ・ワーカリスト w : 未探索エンジン状態の候補群
- ・ $SUCC$: $stmt$ の次に評価されるロケーションの状態の集合
 - ・ $s_1 \sim s_2$: エンジン状態 s_1, s_2 をマージするなら真になるオペランド

$SUCC$ は最大でも2つ（分岐時）

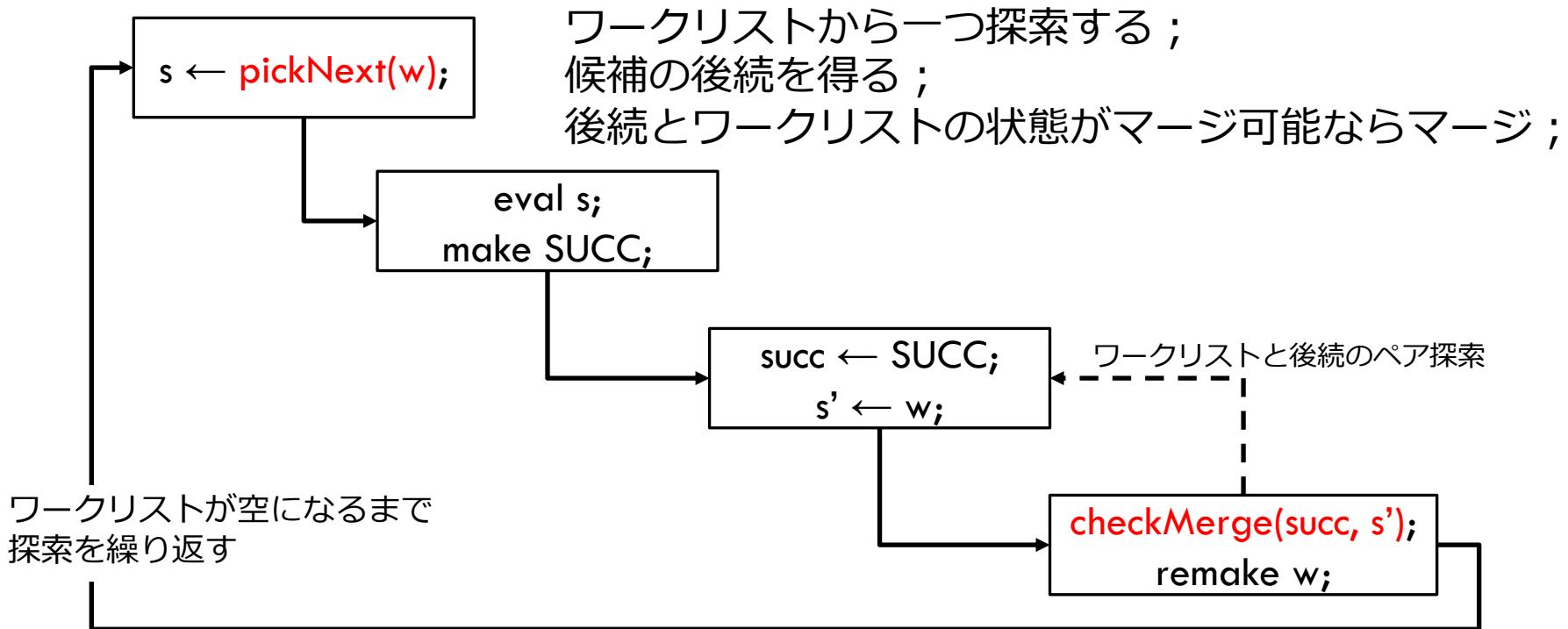


2. TRADE-OFFS IN SYMBOLIC PROGRAM ANALYSIS

2.1 GENERAL SYMBOLIC EXPLORATION

マージ有り記号探索アルゴリズムのモデル化

まとめるとこんな感じ



`pickNext`と`checkMerge(～)`を抽象化することで
色々なマージ有り探索アルゴリズムを表現してるだけ

2. TRADE-OFFS IN SYMBOLIC PROGRAM ANALYSIS

2.2 THE DESIGN SPACE OF SYM PROG ANA

記号プログラム解析のデザインは（大体）以下で決定

- | | |
|------------------|--------------------------|
| ①ループ・再帰への対処 | 健全性・完全性・効率に影響
効率のみに影響 |
| ②各分岐のfeasibility | |
| ③マージの有無、マージの仕方 | |
| ④関数サマリの有無 | |

本論文の提案デザインは

- ①はパラメータ固定
- ②～④は組み合わせる探索戦略に合わせて変化
 - 特に③は静的ではなく動的(On-the-fly)に決定

2. TRADE-OFFS IN SYMBOLIC PROGRAM ANALYSIS

2.2 THE DESIGN SPACE OF SYM PROG ANA

概説

①ループ・再帰への対処

- 探索するループ数、再帰数の上限を決めるのが一般的
 - 後の実験では10回ぐらいでええんでね、としている
- ループ不变条件 (loop invariant)
 - 弱い不变条件は誤検出へつながるので使いづらい
 - 本提案では未採用

②各分岐のfeasibility (実現可能性)

- 要するにunsatな分岐は弾くか否か、ということ
- ソルバを呼び出すタイミングなどが影響 (eager, lazy)
 - 本提案ではeager

2. TRADE-OFFS IN SYMBOLIC PROGRAM ANALYSIS

2.2 THE DESIGN SPACE OF SYM PROG ANA

概説

③マージの有無、マージの仕方

- ・マージする場合、正確性を担保するなら抽象化はしてはいけない
- ・一般に二極的に区別可能
- ・実際にはスペクトラム



EXE[5], KLEE[6], DART[18]
など

- ・マージ条件”～”は空
(＝マージしない)
- ・**pickNextは任意**

• [14]パスのサブセットごとに
マージ条件”～”を変化

など

”Dynamic State Merging”はこの辺

Calysto[2], F-soft[21]など
• マージ条件”～”を固定
• **pickNextはマージに特化**
• **すべての状態対をマージ**
(＝理想的には状態数を
CFGのノード数にする)

2. TRADE-OFFS IN SYMBOLIC PROGRAM ANALYSIS

2.2 THE DESIGN SPACE OF SYM PROG ANA

概説

④関数サマリの有無

- 正確性を担保しつつInter-proceduralな記号プログラム解析の実現
 - 案A：関数のインライン化
 - 欠点：インライン化された関数への解析が重複
 - 案B：関数サマリの利用
 - 利点？：関数出口でのマージによって生成可能
 - 欠点：サマリの生成が再解析と同程度には重い
 - 欠点：サマリ自体はintra-なのでinter- のチェックがどの道必要
 - 欠点：マージでサマリを作ったとき、ite式のソルバコストも大
 - Godefroidら[1][16]：
 - 入力と出力制約のペアをサマリとして収集
再利用可能なサマリがあれば再利用（徐々にサマリを構築）

本提案ではインライン化をサポート（Algo.1 はintra- を想定）

[1] Demand-driven compositional symbolic execution. [TACAS'08]

[16] Compositional dynamic test generation. [POPL'07]

1. INTRODUCTION

本論文の基本的な直感・課題

- ・ソルバへの負担が少ないマージ ≈ 効率的なマージ
- ・マージ優先と競合する探索戦略との兼ね合いをどうするか
 - such as Maximize Line Coverage

アプローチ：

- Query Count Estimation (QCE: クエリ数推定)
 - 変数が後の条件分岐で何回出てくるかを静的に推定
- Dynamic State Merging (DSM: 動的状態マージ)
 - 任意の探索戦略ワークリストからマージに適した状態を検出

2. TRADE-OFFS IN SYMBOLIC PROGRAM ANALYSIS

2.3 DYNAMIC NAVIGATE THE DESIGN SPACE

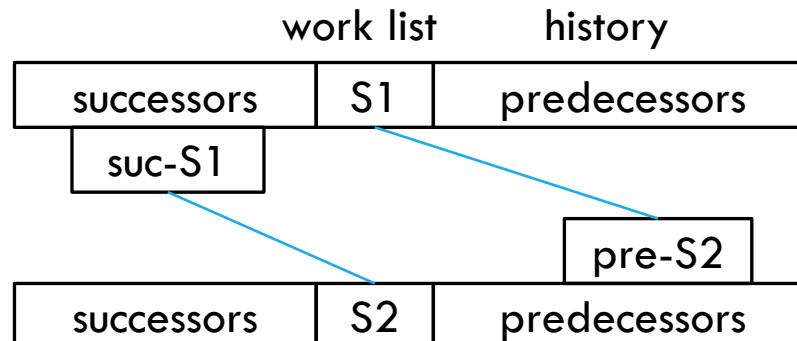
2.2節を踏まえて

本論文の2つの課題

- ① **単純な状態は多く、複雑な状態は少なくする**
探索に有利なバランスを自動で判定
- ② **任意のパス探索戦略と組み合わせられるマージ方法**

アプローチ：

- for ① **Query Count Estimation (QCE: クエリ数推定)**
 - 変数がCFG内のマージポイント（任意の点）までに条件分岐で使用された回数を静的に推定
- for ② **Dynamic State Merging (DSM: 動的状態マージ)**
 - ワークリストを拡張（ワーキリストへ至る状態を一定数保持）



3. APPROACH: QUERY COUNT ESTIMATION

3.1 MOTIVATING EXAMPLE

例：echo プログラム

仮定： $\text{argc} = N+1 \rightarrow L3$ は常に真 (N : 固定の自然数)

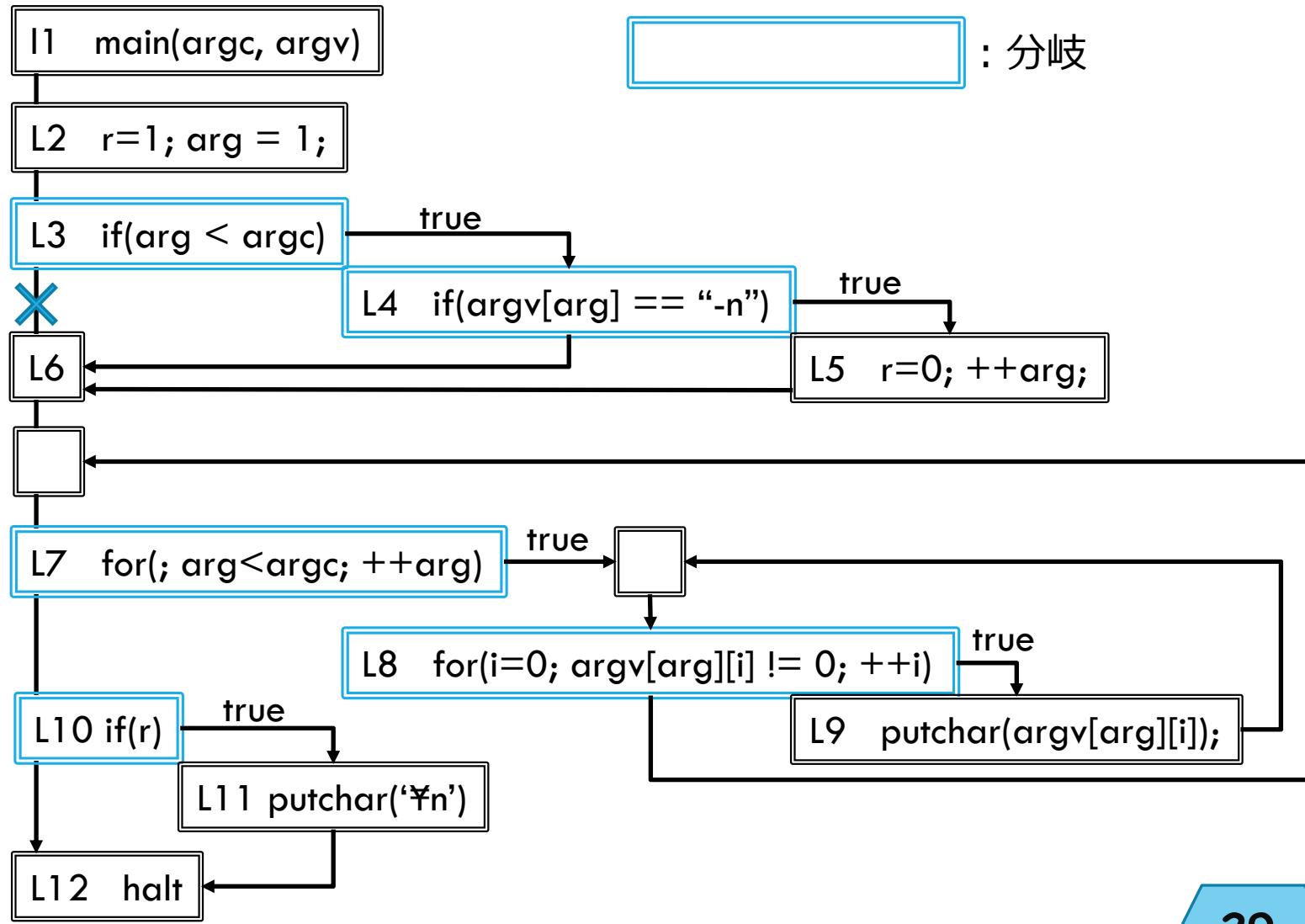
仮定：putchar, strcmpは探索しない

```
1 void main(int argc, char **argv) {
2     int r = 1, arg = 1;
3     if (arg < argc)
4         if (strcmp(argv[arg], "-n") == 0) {
5             r = 0; ++arg;
6         }
7     for (; arg < argc; ++arg)
8         for (int i = 0; argv[arg][i] != 0; ++i)
9             putchar(argv[arg][i]);
10    if (r)
11        putchar('\n');
12 }
```

Figure 1. Simplified version of the echo program.

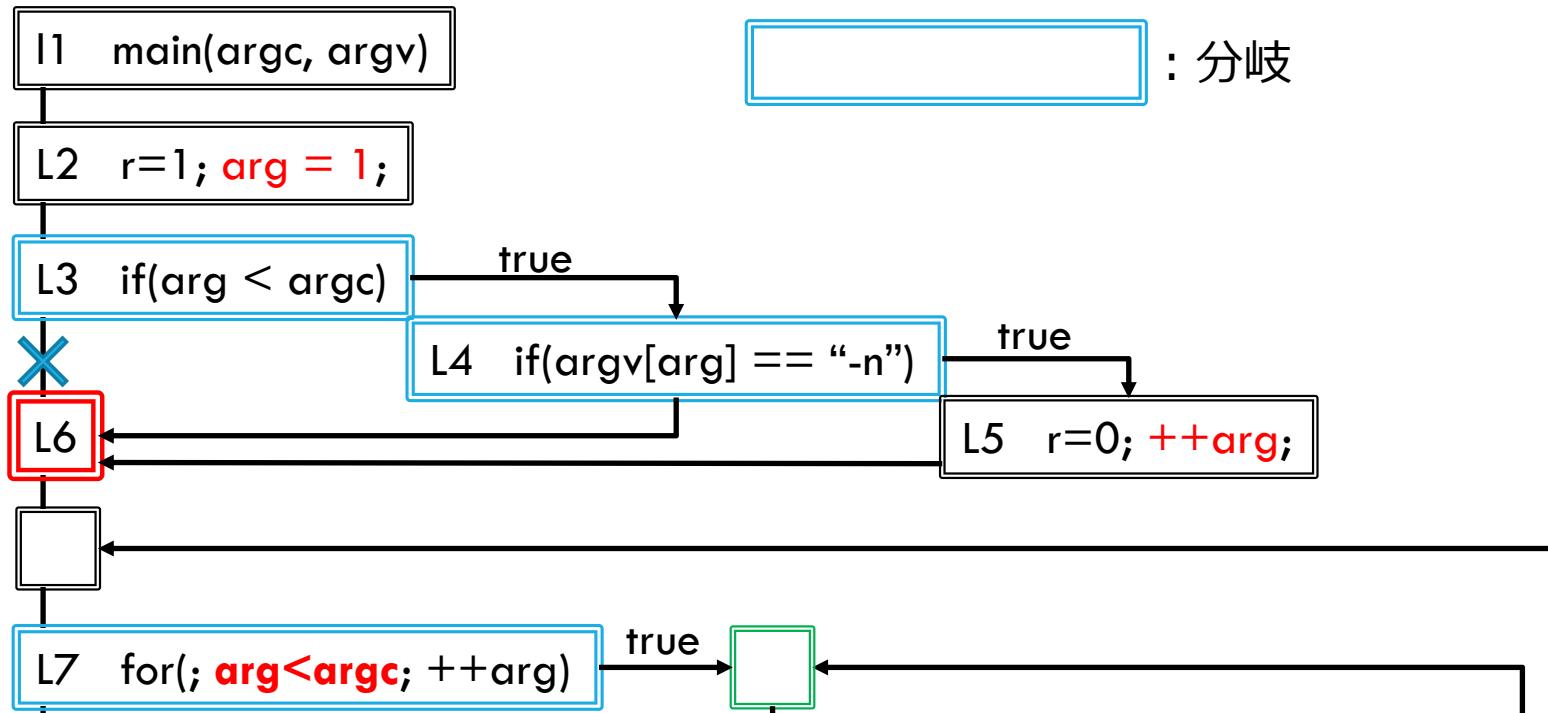
3. APPROACH: QUERY COUNT ESTIMATION

3.1 MOTIVATING EXAMPLE



3. APPROACH: QUERY COUNT ESTIMATION

3.1 MOTIVATING EXAMPLE



愚直なマージ案：

L6でのマージ (L3は常に真なのでL4のみ考慮)

$s6_merged_6 = \{L6, cond_L3, \sigma[r \rightarrow \text{ite}(cond_L4, 0, 1), arg \rightarrow \text{ite}(cond_L4, 2, 1)]\}$

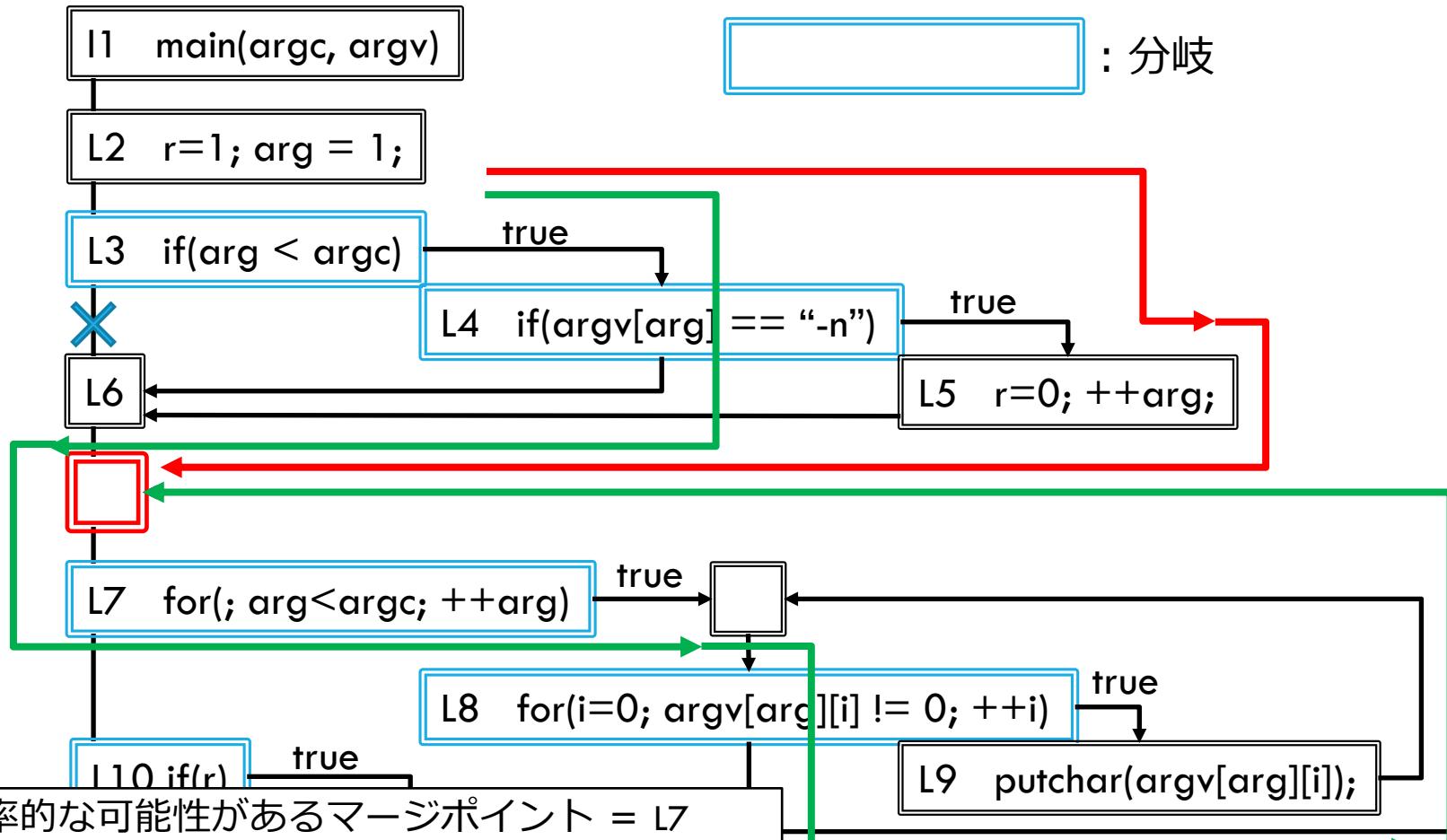
$s8_merged_6 = \{L8, cond_L3 \wedge (\text{arg} < N+1), \sigma\}$

$= \{L8, cond_L3 \wedge (\text{ite}(cond_L4, 2, 1) < N+1), \sigma'\}$

→条件変数がite式で表現され、ソルバの負担大 → L8後は更に悪化

3. APPROACH: QUERY COUNT ESTIMATION

3.1 MOTIVATING EXAMPLE



3. APPROACH: QUERY COUNT ESTIMATION

3.1 MOTIVATING EXAMPLE

例：echo プログラム

ここまで話

- L6よりもL7でのマージのほうが良さげ、というのを自動判定したい
 - 今の例題は手動判定、という位置づけ
- マージの効率を変数に着目して判定
 - → Query Count Estimation (QCE)

3. APPROACH: QUERY COUNT ESTIMATION

3.2 COMPUTING THE HEURISTICS

QCEのヒューリスティクス

- 記号実行を開始する前に、各口けーションstmtで
ホット変数の集合H(stmt)を計算
 - ホット変数の直感：
 - ソルバへのクエリが多くなる可能性が高い変数 = マージに不適

マージ判定式 " \sim_{qce} " を以下に定義

- $(stmt, \pi_1, \sigma_1) \sim_{qce} (stmt, \pi_2, \sigma_2) \stackrel{def}{\iff} \forall v \in H(stmt) \{ (\sigma_1[v] = \sigma_2[v]) \vee (I \blacktriangleleft \sigma_1[v]) \vee (I \blacktriangleleft \sigma_2[v]) \}$
 - すべてのホット変数vについて
vの値が同じ、もしくはvが記号値である ($I \blacktriangleleft \sigma[v]$)
- 意訳：
**クエリに頻繁に含まれる変数全部が
ite式で表現しなくて済むならマージしようす**

3. APPROACH: QUERY COUNT ESTIMATION

3.2 COMPUTING THE HEURISTICS

ホット変数の定義

- $H(\text{stmt}) = \{ v \in V \mid Q_{\text{add}}(\text{stmt}, v) > \alpha * Q_t(\text{stmt}) \}$
- $Q_{\text{add}}(\text{stmt}, v)$:
記号値ではない変数 v が stmt 後に条件変数となる見積もり数
= 変数 v についてのクエリ見積もり数
- $Q_t(\text{stmt})$:
 stmt 後に出てくる全変数のクエリ見積もり数
- Q_{add}, Q_t は以下の式 $q()$ で再帰的に見積もり
 - $Q_{\text{add}}(\text{stmt}, v) = q(\text{stmt}, \lambda(\text{stmt}', e).\text{ite}((\text{stmt}, v) \triangleleft (\text{stmt}', \text{cond}), 1, 0))$
 - $\lambda \dots$: stmt での変数 v が後の文 stmt' の条件 cond に影響するなら (\triangleleft) 1
影響しないなら 0 を返す関数 (c へ渡される)
- $Q_t(\text{stmt}) = q(\text{stmt}, \lambda(\text{stmt}', \text{cond}).1)$
- $q(\text{stmt}', c) =$

$\begin{cases} \beta * q(\text{stmt}' + 1, c) + \beta * q(\text{stmt}'', c) + c(\text{stmt}', \text{cond}) & // \text{if}(\text{cond}) \text{ goto } \text{stmt}'' \\ 0 & // \text{halt} \\ q(\text{stmt}' + 1, c) & // \text{otherwise} \end{cases}$

(°д°) 例題見たほうが早い

3. APPROACH: QUERY COUNT ESTIMATION

3.2 COMPUTING THE HEURISTICS

いいから例題だ！

- $\alpha = 0.5, \beta = 0.6$ で $Qadd(7, \text{arg})$, $Qadd(7, r)$, $Qt(7)$ を求めてみよう

```

7   for ( ; arg < argc; ++arg)
8     for (int i = 0; argv[arg][i] != 0; ++i)
9       putchar(argv[arg][i]);
10    if (r)
11      putchar('\n');
12 }
```

$c(\text{stmt}, \text{cond})$: condがargに依存なら1を返す

- $Qadd(7, \text{arg}) = q(7, c)$

$$\begin{aligned}
& //q(7, c) = \beta q(7+1, c) + \beta q(10, c) + c(7, \text{cond}) \because \text{if}(\text{cond}) \text{ goto stmt"} \\
& = \beta q(8, c) + \beta q(10, c) + c(7, \text{arg} < \text{argc}) \\
& = \beta q(8, c) + 0 + 1 \\
& = \beta (\beta q(9, c) + \beta q(10, c)) + c(8, \text{argv}[\text{arg}][\text{i}] \neq 0) + 1 \\
& = \beta (\beta q(9, c) + 0) + 1 + 1 \\
& = \beta (\beta q(10, c)) + 1 + 1 \\
& = \beta + 1 = \underline{\underline{1.6}}
\end{aligned}$$

$$\begin{aligned}
q(10, c) &= \beta q(11, c) + \beta q(12, c) + c(10, r) \\
&= 2\beta q(12, c) + 0 //q(11, c) = q(12, c) \because \text{otherwise} \\
&= 0 //q(12, c) = 0 \because \text{halt}
\end{aligned}$$

3. APPROACH: QUERY COUNT ESTIMATION

3.2 COMPUTING THE HEURISTICS

いいから例題だ！

- $\alpha = 0.5, \beta = 0.6$ で $\text{Qadd}(7, \text{arg})$, $\text{Qadd}(7, r)$, $\text{Qt}(7)$ を求めてみよう

```

7   for ( ; arg < argc; ++arg)
8     for (int i = 0; argv[arg][i] != 0; ++i)
9       putchar(argv[arg][i]);
10    if (r)
11      putchar(' \n');
12 }
```

$c'(\text{stmt}, \text{cond})$: condがrに依存なら1を返す

$\text{Qadd}(7, r) = q(7, c')$

$$= \beta q(8, c') + \beta q(10, c') + c'(7, \text{arg} < \text{argc})$$

$$= \boxed{} \quad \boxed{①}$$

$$+ \beta q(10, c') + \boxed{} \quad \boxed{②}$$

$$= \boxed{}$$

$$= \boxed{}$$

$$= \boxed{} \quad \boxed{③}$$

$$= \boxed{} \quad \boxed{④}$$

$$= \boxed{}$$

3. APPROACH: QUERY COUNT ESTIMATION

3.2 COMPUTING THE HEURISTICS

いいから例題だ！

- $\alpha = 0.5, \beta = 0.6$ で $\text{Qadd}(7, \text{arg}), \text{Qadd}(7, r), \text{Qt}(7)$ を求めてみよう

```

7   for ( ; arg < argc; ++arg)
8     for (int i = 0; argv[arg][i] != 0; ++i)
9       putchar(argv[arg][i]);
10    if (r)
11      putchar(' \n');
12 }
```

$c'(\text{stmt}, \text{cond})$: condがrに依存なら1を返す

$$\begin{aligned}
\text{Qadd}(7, r) &= q(7, c') \\
&= \beta q(8, c') + \beta q(10, c') + c'(7, \text{arg} < \text{argc}) \\
&= \beta (\beta q(9, c') + \beta q(10, c')) + \beta q(10, c') + c'(8, \text{argv}[\text{arg}][\text{i}] \neq 0) \\
&\quad + \boxed{\text{②}}
\end{aligned}$$

$$\begin{aligned}
&= \boxed{} \\
&= \boxed{}
\end{aligned}$$

$$\begin{aligned}
&= \boxed{} \text{③} \\
&= \boxed{} \text{④}
\end{aligned}$$

$$= \boxed{}$$

3. APPROACH: QUERY COUNT ESTIMATION

3.2 COMPUTING THE HEURISTICS

いいから例題だ！

- $\alpha = 0.5, \beta = 0.6$ で $Q_{\text{add}}(7, \text{arg}), Q_{\text{add}}(7, r), Q_t(7)$ を求めてみよう

```

7   for ( ; arg < argc; ++arg)
8     for (int i = 0; argv[arg][i] != 0; ++i)
9       putchar(argv[arg][i]);
10    if (r)
11      putchar(' \n');
12 }
```

$c'(stmt, cond)$: condがrに依存なら1を返す

- $Q_{\text{add}}(7, r) = q(7, c')$

$$\begin{aligned}
 &= \beta q(8, c') &+ \beta q(10, c') &+ c'(7, \text{arg} < \text{argc}) \\
 &= \beta (\beta q(9, c') &+ \beta q(10, c') &+ c'(8, \text{argv}[\text{arg}][i] \neq 0)) \\
 &&+ \beta q(10, c') &+ 0 \\
 \\
 &= \beta^2 q(9, c') + (\beta^2 + \beta) q(10, c') \\
 &= (2\beta^2 + \beta) q(10, c')
 \end{aligned}$$

$$= \boxed{\quad}$$

③

$$= \boxed{\quad}$$

④

$$= \boxed{\quad}$$

3. APPROACH: QUERY COUNT ESTIMATION

3.2 COMPUTING THE HEURISTICS

いいから例題だ！

- $\alpha = 0.5, \beta = 0.6$ で $Q_{\text{add}}(7, \text{arg}), Q_{\text{add}}(7, r), Q_t(7)$ を求めてみよう

```

7   for ( ; arg < argc; ++arg)
8     for (int i = 0; argv[arg][i] != 0; ++i)
9       putchar(argv[arg][i]);
10    if (r)
11      putchar(' \n');
12 }
```

$c'(stmt, cond)$: condがrに依存なら1を返す

$$\begin{aligned}
 & Q_{\text{add}}(7, r) = q(7, c') \\
 & = \beta q(8, c') + \beta q(10, c') + c'(7, \text{arg} < \text{argc}) \\
 & = \beta (\beta q(9, c') + \beta q(10, c') + \beta q(10, c')) + c'(8, \text{argv}[\text{arg}][i] \neq 0) + 0 \\
 & = \beta^2 q(9, c') + (\beta^2 + \beta) q(10, c') \\
 & = (2\beta^2 + \beta) q(10, c') \\
 & = (2\beta^2 + \beta) (\beta q(11, c') + \beta q(12, c') + c'(10, r)) \\
 & = \boxed{\quad} \quad \boxed{④} \\
 & = \boxed{\quad}
 \end{aligned}$$

3. APPROACH: QUERY COUNT ESTIMATION

3.2 COMPUTING THE HEURISTICS

いいから例題だ！

- $\alpha = 0.5, \beta = 0.6$ で $Q_{\text{add}}(7, \text{arg}), Q_{\text{add}}(7, r), Q_t(7)$ を求めてみよう

```

7   for ( ; arg < argc; ++arg)
8     for (int i = 0; argv[arg][i] != 0; ++i)
9       putchar(argv[arg][i]);
10    if (r)
11      putchar(' \n');
12 }
```

$c'(\text{stmt}, \text{cond})$: condがstmtに依存なら1を返す

$$\begin{aligned}
 & \bullet Q_{\text{add}}(7, r) = q(7, c') \\
 & = \beta q(8, c') + \beta q(10, c') + c'(7, \text{arg} < \text{argc}) \\
 & = \beta (\beta q(9, c') + \beta q(10, c') + \beta q(10, c')) + c'(8, \text{argv}[\text{arg}][\text{i}] \neq 0) + 0 \\
 & = \beta^2 q(9, c') + (\beta^2 + \beta) q(10, c') \\
 & = (2\beta^2 + \beta) q(10, c') \\
 & = (2\beta^2 + \beta) (\beta q(11, c') + \beta q(12, c') + c'(10, r)) \\
 & = (2\beta^2 + \beta) (0 + 0 + 1) \\
 & = 2\beta^2 + \beta = \underline{\underline{1.32}}
 \end{aligned}$$

3. APPROACH: QUERY COUNT ESTIMATION

3.2 COMPUTING THE HEURISTICS

いいから例題だ！

- $\alpha = 0.5, \beta = 0.6$ で $Q_{\text{add}}(7, \text{arg}), Q_{\text{add}}(7, r), Qt(7)$ を求めてみよう

```

7   for ( ; arg < argc; ++arg)
8     for (int i = 0; argv[arg][i] != 0; ++i)
9       putchar(argv[arg][i]);
10    if (r)
11      putchar(' \n');
12 }
```

$c''(\text{stmt}, \text{cond})$: 常に1を返す

$$\begin{aligned}
 \bullet \quad Qt(7) &= q(7, c'') \\
 &= \beta q(8, c'') + \beta q(10, c'') + c''(7, \text{arg} < \text{argc}) \\
 &= \boxed{ + \beta q(10, c'')} \quad \textcircled{5} \\
 &\quad + \boxed{} \quad \textcircled{6}
 \end{aligned}$$

$$\begin{aligned}
 &= \boxed{} \\
 &= \boxed{}
 \end{aligned}$$

$$\begin{aligned}
 &= \boxed{} \quad \textcircled{7} \\
 &= \boxed{} \quad \textcircled{8}
 \end{aligned}$$

$$= \boxed{}$$

3. APPROACH: QUERY COUNT ESTIMATION

3.2 COMPUTING THE HEURISTICS

いいから例題だ！

- $\alpha = 0.5, \beta = 0.6$ で $Q_{\text{add}}(7, \text{arg}), Q_{\text{add}}(7, r), Qt(7)$ を求めてみよう

```

7   for ( ; arg < argc; ++arg)
8     for (int i = 0; argv[arg][i] != 0; ++i)
9       putchar(argv[arg][i]);
10    if (r)
11      putchar(' \n');
12 }
```

$c''(\text{stmt}, \text{cond})$: 常に1を返す

$$\begin{aligned}
 \bullet \quad Qt(7) &= q(7, c'') \\
 &= \beta q(8, c'') + \beta q(10, c'') + c''(7, \text{arg} < \text{argc}) \\
 &= \beta (\beta q(9, c'') + \beta q(10, c'') + c''(8, \text{argv}[\text{arg}][\text{i}] \neq 0)) \\
 &\quad + \boxed{\textcircled{6}}
 \end{aligned}$$

$$\begin{aligned}
 &= \boxed{} \\
 &= \boxed{}
 \end{aligned}$$

$$\begin{aligned}
 &= \boxed{} \textcircled{7} \\
 &= \boxed{} \textcircled{8}
 \end{aligned}$$

$$= \boxed{}$$

3. APPROACH: QUERY COUNT ESTIMATION

3.2 COMPUTING THE HEURISTICS

いいから例題だ！

- $\alpha = 0.5, \beta = 0.6$ で $Q_{\text{add}}(7, \text{arg}), Q_{\text{add}}(7, r), Qt(7)$ を求めてみよう

```

7   for ( ; arg < argc; ++arg)
8       for (int i = 0; argv[arg][i] != 0; ++i)
9           putchar(argv[arg][i]);
10      if (r)
11          putchar('\n');
12 }
```

$c''(\text{stmt}, \text{cond})$: 常に1を返す

$$\begin{aligned}
 \bullet \quad Qt(7) &= q(7, c'') \\
 &= \beta q(8, c'') + \beta q(10, c'') + c''(7, \text{arg} < \text{argc}) \\
 &= \beta (\beta q(9, c'') + \beta q(10, c'') + c''(8, \text{argv}[\text{arg}][\text{i}] \neq 0)) \\
 &\quad + \beta q(10, c'') + 1 \\
 \\
 &= \beta^2 q(9, c'') + (\beta^2 + \beta) q(10, c'') + \beta + 1 \\
 &= (2\beta^2 + \beta) q(10, c'') + \beta + 1
 \end{aligned}$$

$$\begin{aligned}
 &= \boxed{\quad} \quad (7) \\
 &= \boxed{\quad} \quad (8)
 \end{aligned}$$

$$= \boxed{\quad}$$

3. APPROACH: QUERY COUNT ESTIMATION

3.2 COMPUTING THE HEURISTICS

いいから例題だ！

- $\alpha = 0.5, \beta = 0.6$ で $Qadd(7, \text{arg})$, $Qadd(7, r)$, $Qt(7)$ を求めてみよう

```

7   for ( ; arg < argc; ++arg)
8     for (int i = 0; argv[arg][i] != 0; ++i)
9       putchar(argv[arg][i]);
10    if (r)
11      putchar('\n');
12 }
```

$c''(\text{stmt}, \text{cond}):$ 常に1を返す

• $Qt(7) = q(7, c'')$

$$\begin{aligned} &= \beta q(8, c'') + \beta q(10, c'') + c''(7, \text{arg} < \text{argc}) \\ &= \beta (\beta q(9, c'') + \beta q(10, c'') + c''(8, \text{argv}[\text{arg}][\text{i}] \neq 0)) \\ &\quad + \beta q(10, c'') + 1 \\ &= \beta^2 q(9, c'') + (\beta^2 + \beta) q(10, c'') + \beta + 1 \\ &= (2\beta^2 + \beta) q(10, c'') + \beta + 1 \\ &= (2\beta^2 + \beta) (\beta q(11, c'') + \beta q(12, c'') + c''(10, r)) + \beta + 1 \end{aligned}$$

= ⑧

= _____

3. APPROACH: QUERY COUNT ESTIMATION

3.2 COMPUTING THE HEURISTICS

いいから例題だ！

- $\alpha = 0.5, \beta = 0.6$ で $Q_{\text{add}}(7, \text{arg}), Q_{\text{add}}(7, r), Qt(7)$ を求めてみよう

```

7   for ( ; arg < argc; ++arg)
8     for (int i = 0; argv[arg][i] != 0; ++i)
9       putchar(argv[arg][i]);
10    if (r)
11      putchar(' \n');
12 }
```

$c''(\text{stmt}, \text{cond})$: 常に1を返す

$$\begin{aligned}
 \bullet \quad Qt(7) &= q(7, c'') \\
 &= \beta q(8, c'') + \beta q(10, c'') + c''(7, \text{arg} < \text{argc}) \\
 &= \beta (\beta q(9, c'') + \beta q(10, c'') + c''(8, \text{argv}[\text{arg}][\text{i}] \neq 0)) \\
 &\quad + \beta q(10, c'') + 1 \\
 \\
 &= \beta^2 q(9, c'') + (\beta^2 + \beta) q(10, c'') + \beta + 1 \\
 &= (2\beta^2 + \beta) q(10, c'') + \beta + 1 \\
 \\
 &= (2\beta^2 + \beta) (\beta q(11, c'') + \beta q(12, c'') + c''(10, r)) + \beta + 1 \\
 &= (2\beta^2 + \beta) (0 + 0 + 1) + \beta + 1 \\
 \\
 &= 2\beta^2 + 2\beta + 1 = \underline{\underline{2.92}}
 \end{aligned}$$

3. APPROACH: QUERY COUNT ESTIMATION

3.2 COMPUTING THE HEURISTICS

いいから例題だ！

- $\alpha = 0.5, \beta = 0.6$ で $Q_{\text{add}}(7, \text{arg})$, $Q_{\text{add}}(7, r)$, $Qt(7)$ を求めてみよう

```

7   for ( ; arg < argc; ++arg)
8     for (int i = 0; argv[arg][i] != 0; ++i)
9       putchar(argv[arg][i]);
10    if (r)
11      putchar('\'n');
12 }
```

- よって

$$\begin{aligned}
 Q_{\text{add}}(7, \text{arg}) &= 1.6 \\
 Q_{\text{add}}(7, r) &= 1.32 \\
 Qt(7) &= 2.92
 \end{aligned}$$

より $H(7) = \{\text{arg}\} \because Q_{\text{add}} > \alpha * Qt(7)$

- 6行目は分岐ではないので $H(6) = H(7) = \{\text{arg}\}$
- すなわち、6, 7行目で arg が等しい値か記号値ならばマージに適当

3. APPROACH: QUERY COUNT ESTIMATION

3.2 COMPUTING THE HEURISTICS

QCEのまとめ

- プログラム位置 stmt について
- $H(\text{stmt})$: クエリに大きな負担をかけそうな変数 = ホット変数の集合
- ホット変数の判定：
各変数についてのクエリ見積もり $Q_{\text{add}}(\text{stmt}, v)$ を再帰的に求める
全体のクエリ見積もり $Qt(\text{stmt})$ を再帰的に求める
$$H(\text{stmt}) := \{ v \in V \mid Q_{\text{add}}(\text{stmt}, v) > \alpha Qt(\text{stmt}) \}$$
- マージ可能条件 " $\sim qce$ " :
 stmt でホット変数が同値もしくは記号値ならマージに適する
- QCEはパラメータ α, β, κ により調整
 - α : ホット変数を判定するしきい値
 - β : 分岐確率 (feasibility)
 - κ : ループ・再帰の上限

3. APPROACH: QUERY COUNT ESTIMATION

3.3 JUSTIFICATION

3.2節でのヒューリスティクスが定められるまでの仮定

- 仮定①クエリの解決時間を1単位時間とする
 ite 式の解決には $\xi (\geq 1)$ の固定解決時間がかかる
 →全体の解決時間はクエリ数に比例
- 仮定②マージ候補 s_1, s_2 について $Q_{t_s1}(\text{stmt}) = Q_{t_s2}(\text{stmt})$
 Q_{ite} : ite 式を解決するクエリ数
 $Q_{\text{add}'}$: 定数だが依存元が異なる場合の追加クエリ数
 Q_{disj} : 選言を解決するためのクエリ数

マージ前のコスト : $Q_{t_s1} + Q_{t_s2} = 2Q_t$

マージ後のコスト : $(Q_t - Q_{\text{ite}}) + \xi Q_{\text{ite}} + Q_{\text{add}'} + Q_{\text{disj}}$

ここで、選言解決のためのコストは無視

→大抵の場合互いに否定となる条件の選言だから

→コストから見たマージに適する条件 :

$$(Q_t(\text{stmt}) - Q_{\text{ite}}(\text{stmt})) + \xi Q_{\text{ite}}(\text{stmt}) + Q_{\text{add}'}(\text{stmt}) < 2Q_t(\text{stmt})$$

$$\rightarrow (\xi - 1)Q_{\text{ite}}(\text{stmt}) + Q_{\text{add}'}(\text{stmt}) < Q_t(\text{stmt})$$

3. APPROACH: QUERY COUNT ESTIMATION

3.3 JUSTIFICATION

3.2節でのヒューリスティクスが定められるまでの仮定

- 仮定③各分岐は固定確率 $0.5 < \beta < 1$ で分岐
分岐間の依存関係は考慮しない
→クエリ数を再帰的に見積もれるように
→で、この式

$$Q(\text{stmt}) = q(\text{stmt}, c) = \begin{cases} \beta * q(\text{stmt}+1, c) + \beta * q(\text{stmt}', c) + c(\text{stmt}', \text{cond}) & // \text{if}(\text{cond}) \text{ goto stmt}' \\ 0 & // \text{halt} \\ q(\text{stmt}+1, c) & // \text{otherwise} \end{cases}$$

$Q_t(\text{stmt})$, $Q_{ite}(\text{stmt})$, $Q_{add'}(\text{stmt})$ を計算するための $c(\text{stmt}', \text{cond})$ を以下に定義

$$\begin{aligned} Q_t : c(\text{stmt}', \text{cond}) \\ = \lambda(\text{stmt}', \text{cond}).\text{ite}(\exists v: (I \blacktriangleleft \sigma_1[v] \vee I \blacktriangleleft \sigma_2[v]) \wedge (\text{stmt}, v) \triangleleft (\text{stmt}', \text{cond}), 1, 0) \end{aligned}$$

$$\begin{aligned} Q_{ite} : c(\text{stmt}', \text{cond}) \\ = \lambda(\text{stmt}', \text{cond}).\text{ite}(\exists v: (I \blacktriangleleft \sigma_1[v] \vee I \blacktriangleleft \sigma_2[v]) \\ \wedge \sigma_1[v] \neq \sigma_2[v] \wedge (\text{stmt}, v) \triangleleft (\text{stmt}', \text{cond}), 1, 0) \end{aligned}$$

$$\begin{aligned} Q_{add'} : c(\text{stmt}', \text{cond}) \\ = \lambda(\text{stmt}', \text{cond}).\text{ite}(\exists v: \neg(I \blacktriangleleft \sigma_1[v] \vee I \blacktriangleleft \sigma_2[v]) \\ \wedge \sigma_1[v] \neq \sigma_2[v] \wedge (\text{stmt}, v) \triangleleft (\text{stmt}', \text{cond}), 1, 0) \end{aligned}$$

3. APPROACH: QUERY COUNT ESTIMATION

3.3 JUSTIFICATION

3.2節でのヒューリスティクスが定められるまでの仮定

- 仮定④入力に依存する条件分岐の数は分岐の総数の固定割合 ϕ で算出可能
→ $| \blacktriangleleft \sigma[v]$ の事前決定のため

$$\begin{aligned} &\rightarrow Q_t(\text{stmt}) \\ &= q(\text{stmt}, \lambda(\text{stmt}', \text{cond}).\text{ite}(\exists v: (| \blacktriangleleft \sigma_1[v] \vee | \blacktriangleleft \sigma_2[v]) \wedge (\text{stmt}, v) \triangleleft (\text{stmt}', \text{cond}), 1, 0)) \\ &= \phi \ q(\text{stmt}, \lambda(\text{stmt}', \text{cond}).1) \end{aligned}$$

$Q_{add}'(\text{stmt})$ with $Q_{ite}(\text{stmt})$

→ 記号値に限らず、値の異なる変数の集合についてのクエリ数

変数単位で見ると

$$\begin{aligned} Q_{ite}(\text{stmt}, v) &= Q_{add}(\text{stmt}, v) \\ &= q(\text{stmt}, \lambda(\text{stmt}, \text{cond}).\text{ite}((\text{stmt}, v) \triangleleft (\text{stmt}', \text{cond}), 1, 0)) \leq Q_{ite}(\text{stmt}), Q_{add}'(\text{stmt}) \end{aligned}$$

クエリの重複が起こり得るので、実際のクエリ数は

$$\max Q_{add}(\text{stmt}, v) \leq Q_{add}'(\text{stmt}) \leq \sum Q_{add}(\text{stmt}, v)$$

$$\max Q_{ite}(\text{stmt}, v) \leq Q_{ite}(\text{stmt}) \leq \sum Q_{ite}(\text{stmt}, v)$$

3. APPROACH: QUERY COUNT ESTIMATION

3.3 JUSTIFICATION

3.2節でのヒューリスティクスが定められるまでの仮定

- 仮定⑤総クエリ数は変数へのクエリ数の最大値の定数 ρ 倍である

$$\rightarrow Q_{ite}(\text{stmt}) \propto \rho \max Q_{ite}(\text{stmt}, v)$$

$$\rightarrow Q_{add}'(\text{stmt}) \propto \rho \max Q_{add}(\text{stmt}, v)$$

コストから見たマージに適する条件 from 仮定②

$$(\xi - 1)Q_{ite}(\text{stmt}) + Q_{add}'(\text{stmt}) < Q_t(\text{stmt})$$

仮定④, ⑤

$$\rightarrow (\xi - 1) \max Q_{ite}(\text{stmt}, v) + \max Q_{add}(\text{stmt}, v) < \phi / \rho Q_t(\text{stmt})$$

→唐突に $Q_{ite}(\text{stmt}, v)$ が消えて（実験でも影響が出ている模様）

$$\rightarrow \max Q_{add}(\text{stmt}, v) < \phi / \rho Q_t(\text{stmt})$$

$$\rightarrow \forall v \in V \mid \neg(I \blacktriangleleft \sigma_1[v] \vee I \blacktriangleleft \sigma_2[v]) \wedge \sigma_1[v] \neq \sigma_2 \\ Q_{add}(\text{stmt}, v) < \alpha Q_t(\text{stmt})$$

一部理解しきれていない by 小泉

4. APPROACH: DYNAMIC STATE MERGING

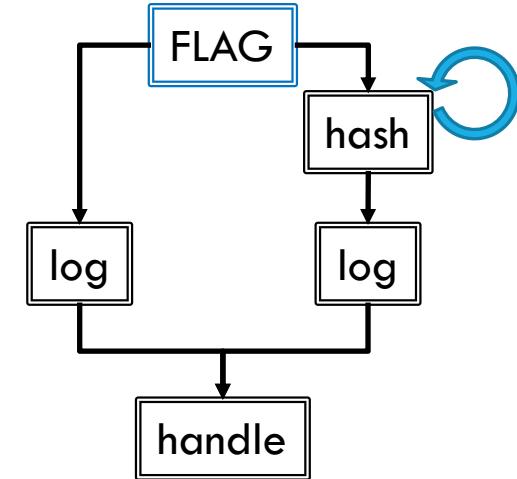
4.1 STATIC MERGE AND INCOMPLETE EXPLORATION

既存マージ手法（静的マージ）

- 全ジョインポイントでマージを試行
 - 完全なプログラム検証には適している
が、前述の通り記号実行のパス探索戦略の多くと競合

```

1 if (logPacketHash) {
2   hash = computeHash(pkt);
3   log("Packet: %s, hash: %s", pkt->name, hash);
4 } else {
5   log("Packet: %s", pkt->name);
6 }
7 handlePacket(pkt);
  
```



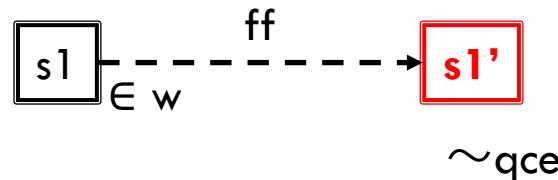
- 静的マージが探索戦略を妨害する例
 - 静的マージでは7行目の評価前にマージを試行
 - handlePacketの前にcomputeHashを全探索しようとする
 - handlePacket内部に到達するまで時間がかかる
 - カバレッジ最大化戦略などと競合**

4. APPROACH: DYNAMIC STATE MERGING

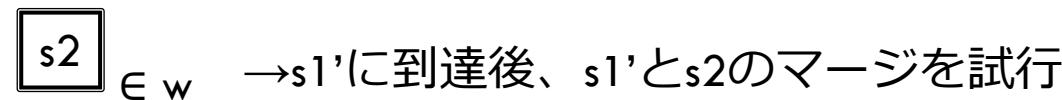
4.3 DSM ALGORITHM

Dynamic State Merging (DSM)

- ワークリスト内の状態 $s_1, s_2 = \{stmt_i, \pi_i, \sigma_i\}$ について



s_1' と s_2 がマージに適している"~qce"なら、探索戦略を超えて s_1 を優先 (fast-following)



$\rightarrow s_1'$ に到達後、 s_1' と s_2 のマージを試行

stmt_1 -----> stmt_2

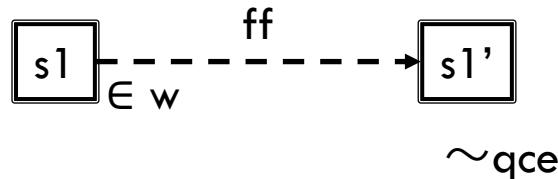
少ステップで到達

4. APPROACH: DYNAMIC STATE MERGING

4.3 DSM ALGORITHM

Dynamic State Merging (DSM)

- ワークリスト内の状態 $s_1, s_2 = \{stmt_i, \pi_i, \sigma_i\}$ について



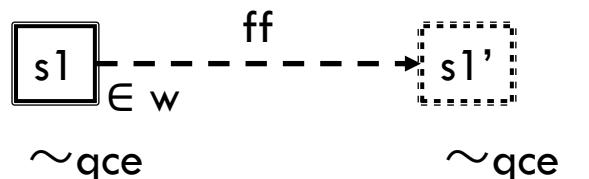
s_1' と s_2 がマージに適している"~qce"なら、探索戦略を超えて s_1 を優先 (**fast-following**)



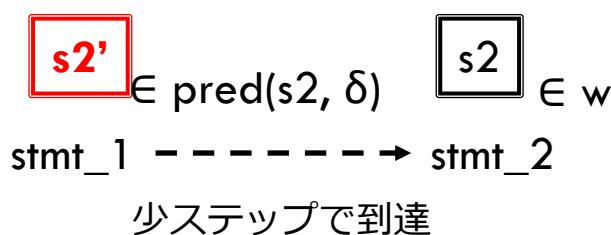
$\rightarrow s_1'$ に到達後、 s_1' と s_2 のマージを試行

$stmt_1 \dashrightarrow stmt_2$

少ステップで到達



s_1' がマージに適しているかを事前判定
→**s2へ至る状態群** $pred(s_2, \delta)$ に s_1 と
マージに適している状態があるか検索



δ : 「少ステップ」の長さ

→実験ではBBL 8つ分を指定 (根拠は特に無)

5. EXPERIMENT

5.1 PROTOTYPE

実装：

- KLEEエンジン上に構築
- for QCE : LLVMを静的に解析できるようにKLEEを拡張
 - $Qt(\text{stmt})$ と $Qadd(\text{stmt}, v)$ を計算できるように注釈を挿入
 - 対象変数：
 - ローカル変数、関数引数、メモリ内変数
 - メモリ内変数：
ローカル、関数引数、グローバル変数から固定オフセットで参照される変数
 - ローカル変数についてはデータ依存をチェック (by SSA形式)
- for DSM : KLEEの探索戦略レイヤーに $\text{pred}(s, \delta)$ を実装
 - 採用探索戦略 (KLEEがサポートしている戦略)：
 - ランダム探索 (for 完全探索 in 5.4節)
 - ステートメントカバレッジ探索 (for 部分探索 in 5.5節)

検体：GNU Core Utilities

- 小さすぎる検体は省略 (静的な検証が支配的になるため)

5. EXPERIMENT

5.2 EVALUATION METRICS

比較対象：

- KLEE as No-merge Search-based Symbolic Execution

指標：

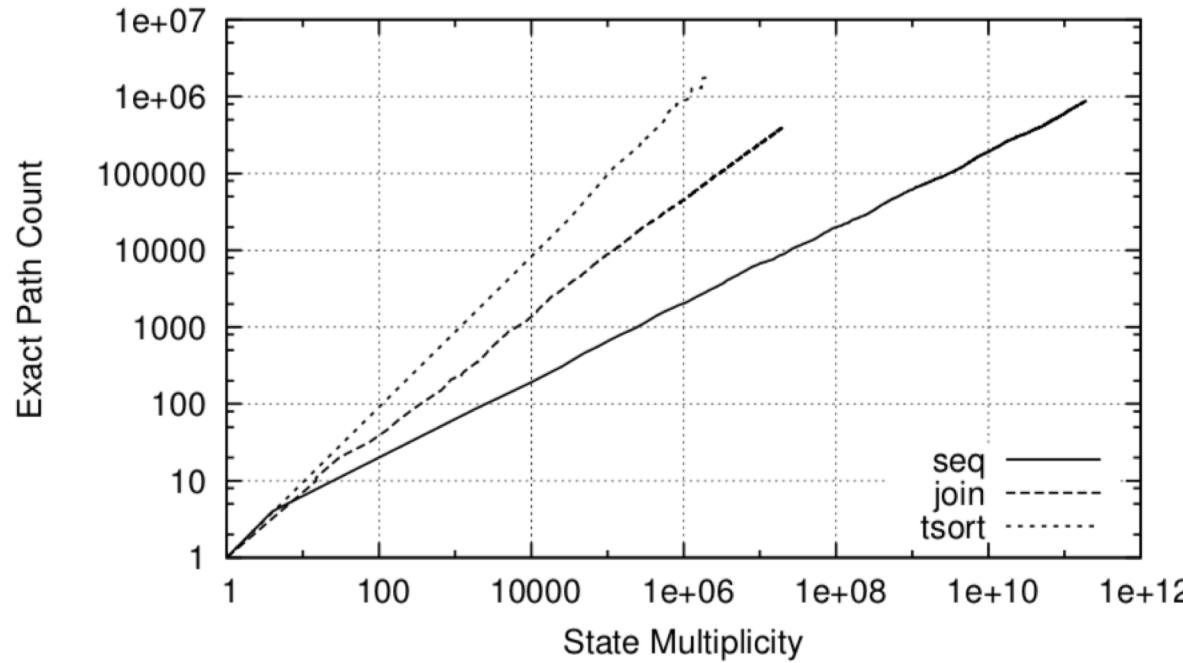
- ①固定時間内のパス探索量 →5.3節
- ②タスク完了時間 →5.4, 5.5節
 - タスク：ここでは探索戦略が`interest`とするパスの探索
- マージ時のパス探索量を正確に計算するのは困難
 - → **state multiplicity (状態多重度) で推定**
 - → 単一パスなら1、マージパスはマージ前のパスの多重度の合計

5. EXPERIMENT

5.2 EVALUATION METRICS

多重度の合計は重複が起こるので過大評価になる

- ・合計多重度 m と探索パス数 p を次の式でモデル化（詳細は省略）
 - $\log p = C_1 + C_2 \log m$
 - いくつかの検体(seq, join, tsort)の正確なパス探索数と比較



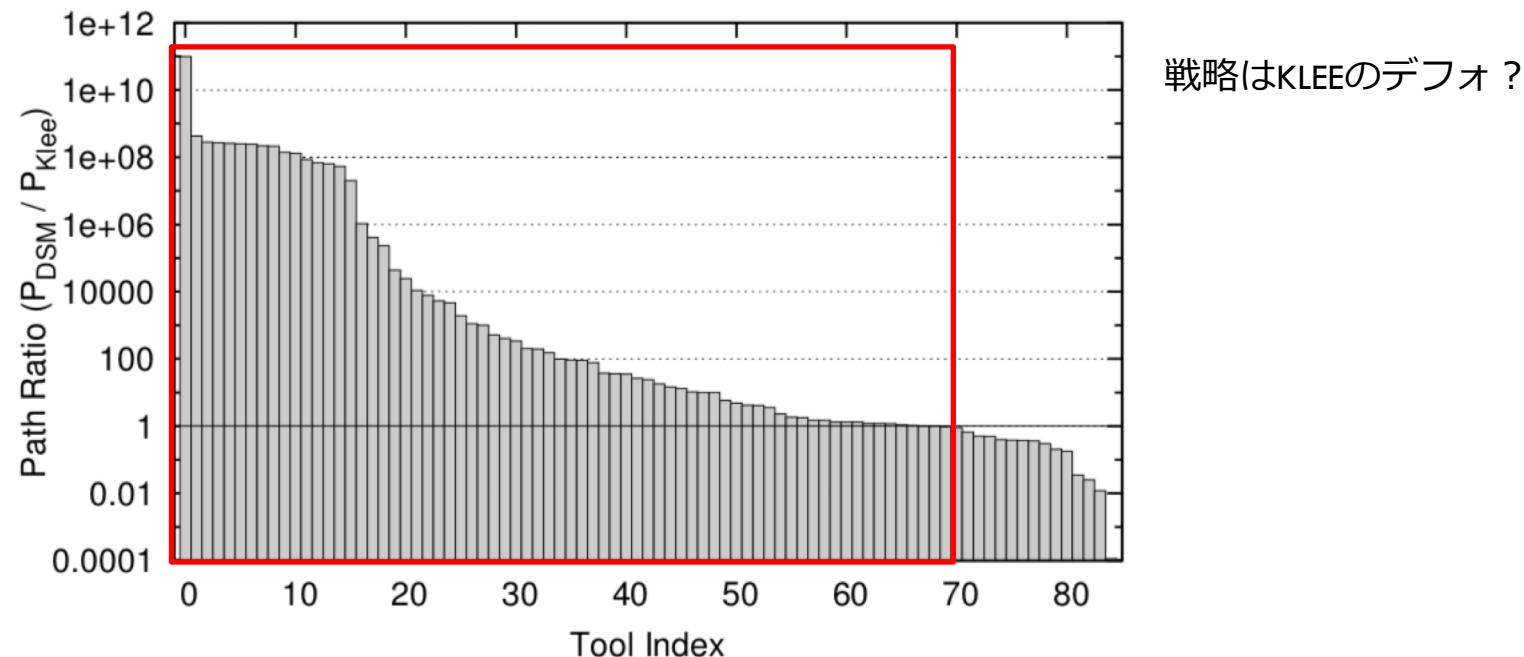
- ・正確なパス数と多重度が指数的に比例
 - 以降の実験ではここから得られた定数 C_1, C_2 を使用

5. EXPERIMENT

5.3 FASTER PATH EXPLORATION WITH DSM, QCE

DSMとQCEを適用したときパス探索が高速化するか検証

- ・パス数を比較（1以上なら提案手法優位）
- ・各検体の入力空間：1時間ずっとツールが“busy”になるぐらいには大



- 結果：
- ・ 14/83はKLEEよりもパス探索数が減少
 - ・ Qiteの無視などが原因
- ・ **69/83はパス探索数が増加（最大11桁の差）**

5. EXPERIMENT

5.4 ACHIEVING EXPONENTIAL SPEEDUP WITH QCE

QCE with DSMは高速化した（5.3節）

- QCE単体ではどうか？

QCE with Selective SSM (Static State Merging)

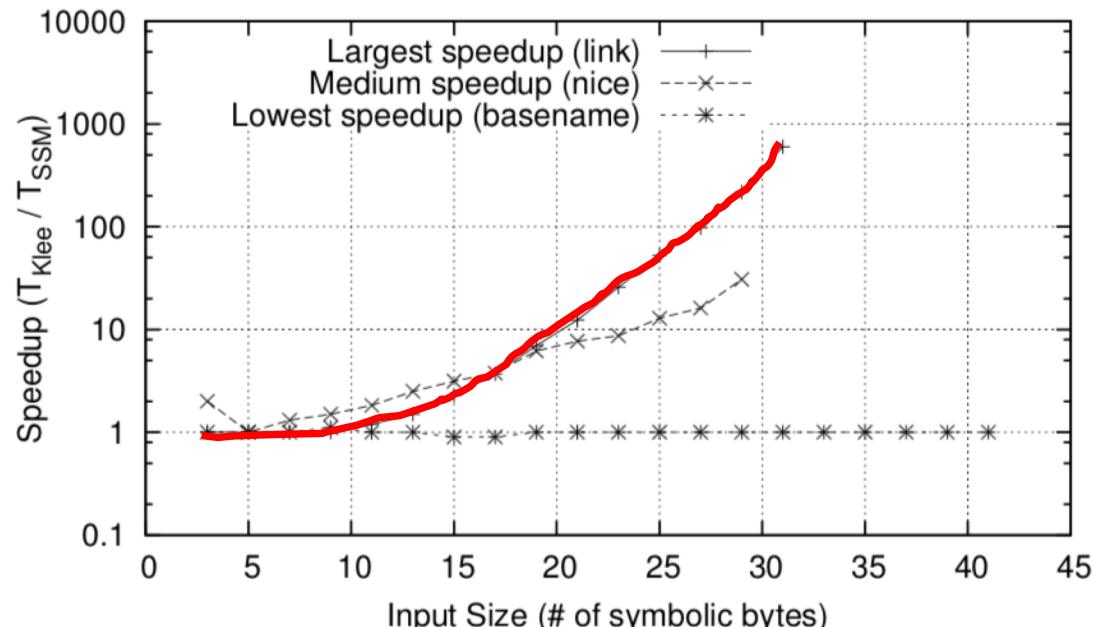
- Selective SSM：ワーカリスト内の状態をトポロジカルにマージ試行
- 評価ポイント
 - ②A：完全パス探索 with QCEの探索終了時間
 - ②B：各パラメータ (α, β, κ) のパフォーマンスへの影響

5. EXPERIMENT

5.4 ACHIEVING EXPONENTIAL SPEEDUP WITH QCE

②A : 完全パス探索 with QCEの探索終了時間

- vs. KLEE



代表例の結果 :

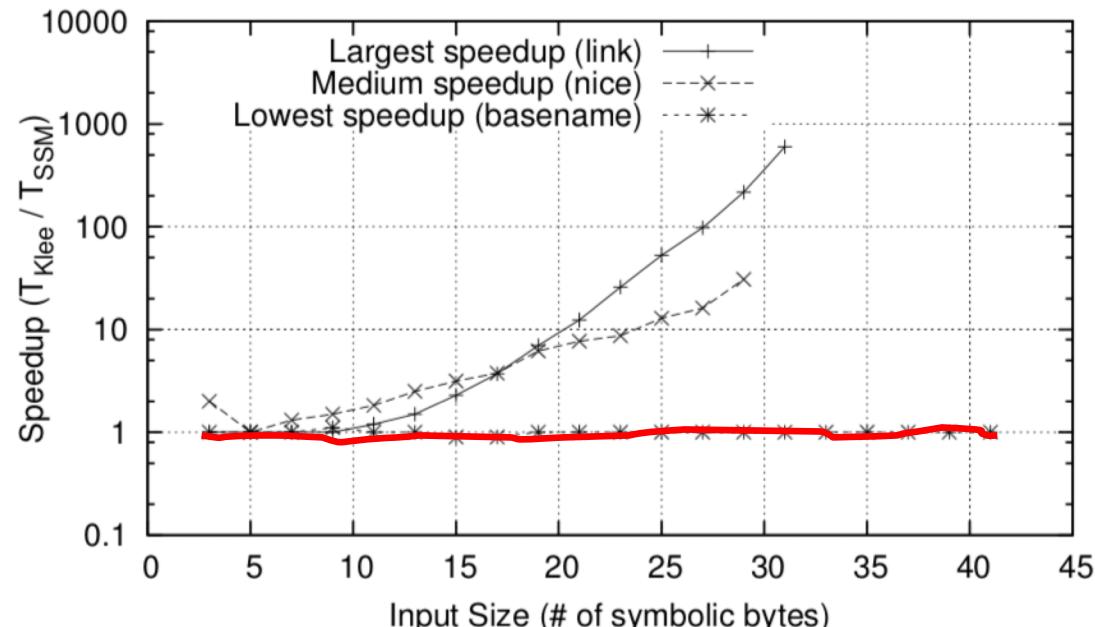
- 横軸 : 入力空間サイズ 縦軸 : 高速化比 (1以上でQCE優位)
- link** : メッチャスピードアップをした例
→入力サイズが大きいほど高速化
- nice** : 標準的なスピードアップをした例
- basename** : 全くスピードアップしなかった例
→が、これは稀 (次実験)

5. EXPERIMENT

5.4 ACHIEVING EXPONENTIAL SPEEDUP WITH QCE

②A : 完全パス探索 with QCEの探索終了時間

- vs. KLEE



代表例の結果 :

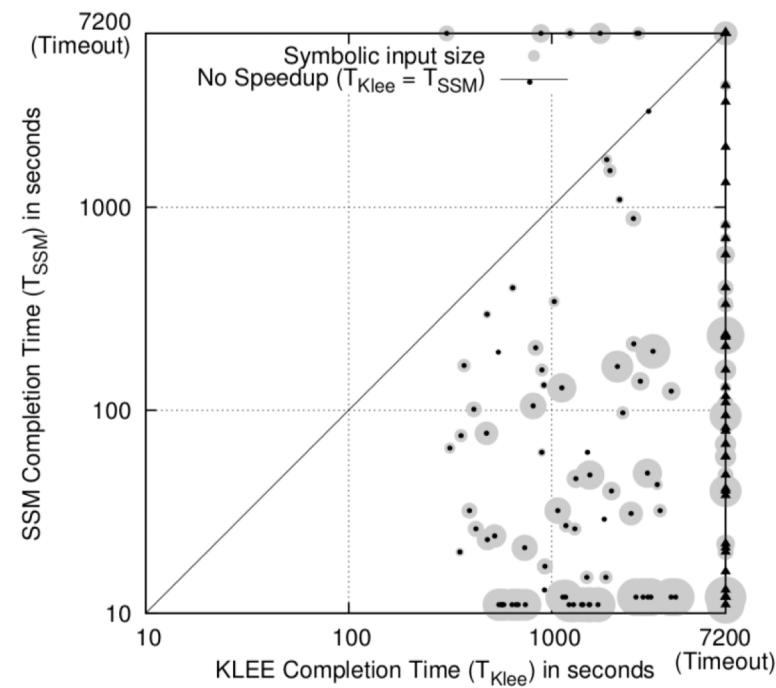
- 横軸 : 入力空間サイズ 縦軸 : 高速化比 (1以上でQCE優位)
 - link : メッチャスピードアップをした例
→入力サイズが大きいほど高速化
 - nice : 標準的なスピードアップをした例
 - basename** : 全くスピードアップしなかつた例
→が、これは稀 (次実験)

5. EXPERIMENT

5.4 ACHIEVING EXPONENTIAL SPEEDUP WITH QCE

②A : 完全パス探索 with QCEの探索終了時間

- vs. KLEE
- 各検体最大 2 時間の実行
- 各検体で様々な入力サイズを試行
 - 各点：スピードアップの下限
 - 灰円：各点での入力サイズ比
 - 大きいほどQCE優位



全体の結果（散布図）：

- **右下ほどQCE優位**（右端ならKLEEは探索終了せず）
- 両者で探索が終了する場合、灰円は大きくなる傾向
- →KLEEが苦手な大きな入力空間による状態爆発に対応できるように

5. EXPERIMENT

5.4 ACHIEVING EXPONENTIAL SPEEDUP WITH QCE

②A : 完全パス探索 with QCEの探索終了時間

- Case Study : sleep (Utils)
 - 整数リストをコマンドラインから取得し、合計を変数secondsに格納
- QCE : secondsをホット変数と見做さず
 - → 解析中に出現した全分岐でマージを敢行
 - → パス爆発を抑制（ソルバのコストも償却）
- その他KLEEより悪化した例
 - 原因：
 - **ite式が複雑になりすぎる** → **QCEの改善余地がまだ有**
 - 選言の解決にかかるコスト → ソルバとの相性
 - ソルバはKLEEなのでSTP

5. EXPERIMENT

5.4 ACHIEVING EXPONENTIAL SPEEDUP WITH QCE

②B : 各パラメータ (α, β, κ) のパフォーマンスへの影響

他の実験でのパラメータ : $\alpha = 10^{-12}$ 、 $\beta = 0.8$ 、 $\kappa = 10$

- 4つの検体から山登り法で α, β の最適値を導出

- 検体の選出は無作為

- κ はループ数が入力から影響を受けるという知見から平均入力サイズに対応する $\kappa = 10$ を選択 (?)

- → なんだかんだ十分に良い結果

パラメータのうち、 **α がもっともパフォーマンスへ影響**

- α : どれだけ積極的にマージするか、とも言える

- $\alpha = \infty$: 全変数がホットではない → 全部マージしようぜ !

- $\alpha = 0$: 異なる具体値を持つ変数は絶対にマージされない

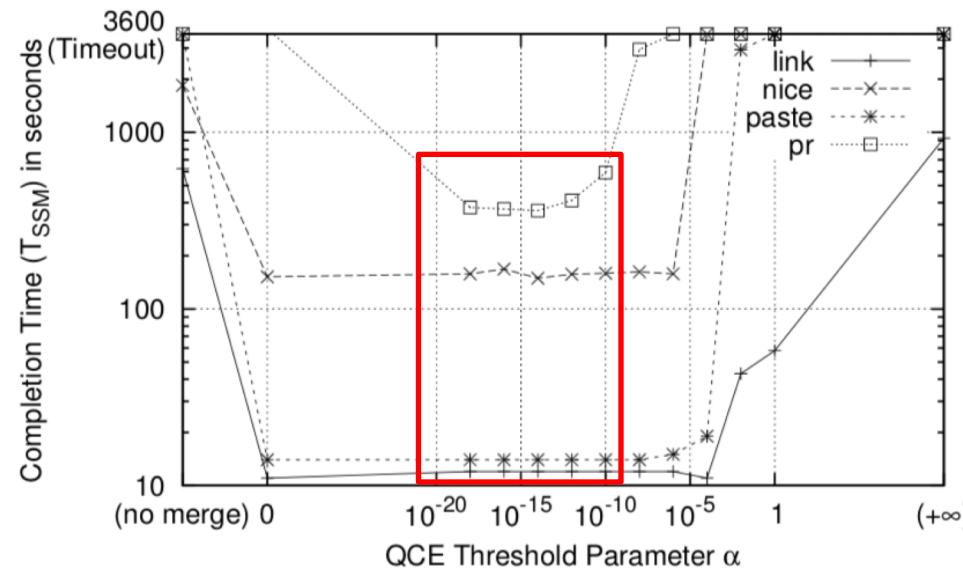
5. EXPERIMENT

5.4 ACHIEVING EXPONENTIAL SPEEDUP WITH QCE

②B : 各パラメータ (α, β, κ) のパフォーマンスへの影響

α の違いによるパフォーマンスへの影響

- ・検体 : link, nice, parse, pr (無作為に選出)
- ・色々な α について 1 時間実行



- ・結果 : $\alpha = 10^{-20} \sim 10^{-10}$ ぐらいならいい感じ

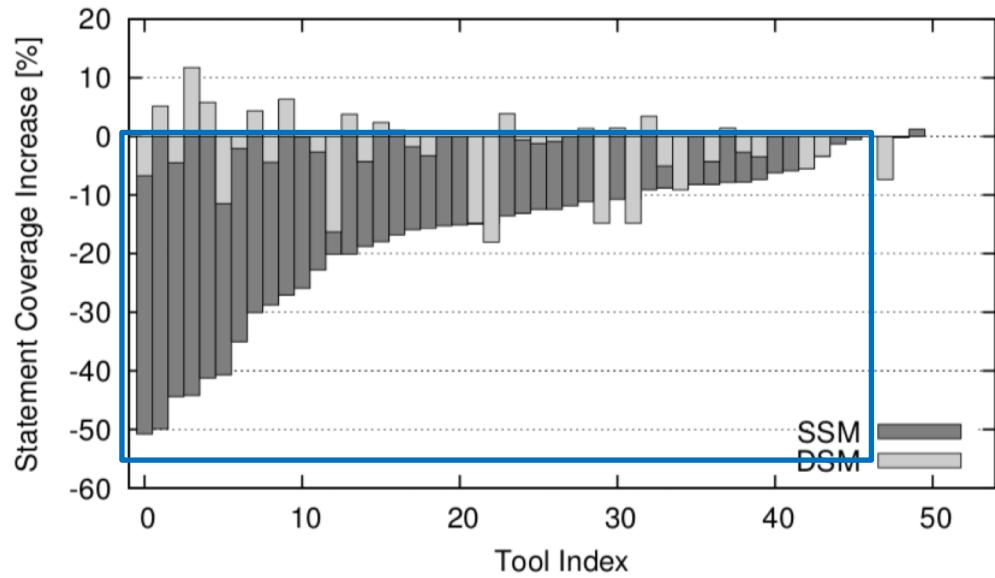
5. EXPERIMENT

5.5 REACHING AN EXPLORATION GOAL WITH DSM

DSMをパス探索戦略に適用したとき、タスクを達成できるか

- ・タスク：ここでは探索戦略がinterestとするパスの探索
- ・QCE with DSM vs. QCE with Selective SSM
- ・探索戦略：カバレッジヒューリスティクス[6] (TO : 1 時間)
 - ・保持するヒストリー長 δ ：試験的に BBL 8つ分
 - ・KLEEでのステートメントカバレッジを100%とする

- ・結果：
 - ・SSMはカバレッジ悪化
 - ・DSMはカバレッジ保持
 - ・タスクを阻害しない
 - ・5.3節から
パス数は増加しているが
カバレッジの増加には
つながらなかった
 - ・また、69%のfast-followingで選択された状態がマージ成功



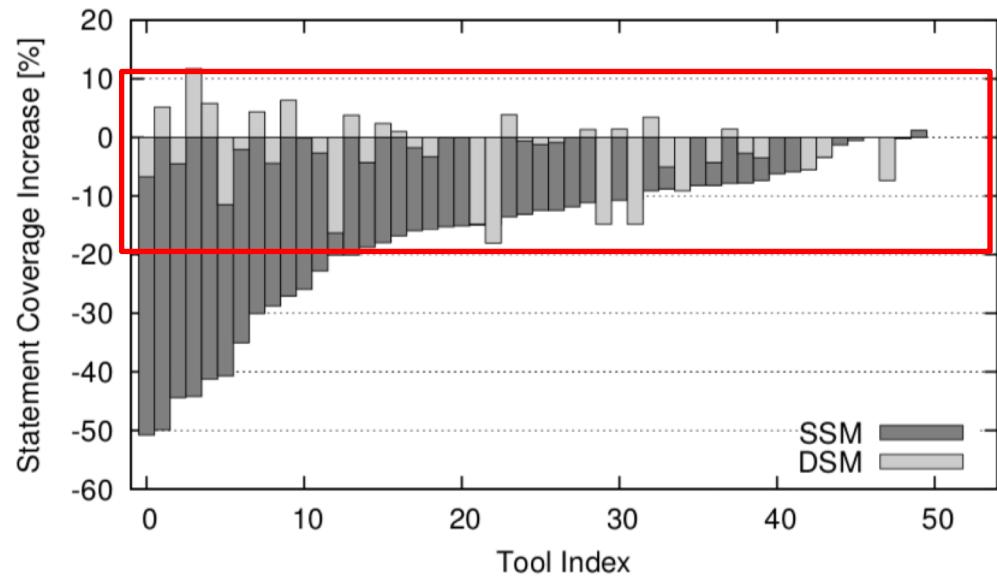
5. EXPERIMENT

5.5 REACHING AN EXPLORATION GOAL WITH DSM

DSMをパス探索戦略に適用したとき、タスクを達成できるか

- ・タスク：ここでは探索戦略がinterestとするパスの探索
- ・QCE with DSM vs. QCE with Selective SSM
- ・探索戦略：カバレッジヒューリスティクス[6] (TO : 1 時間)
 - ・保持するヒストリー長 δ ：試験的に BBL 8つ分
 - ・KLEEでのステートメントカバレッジを100%とする

- ・結果：
 - ・SSMはカバレッジ悪化
 - ・DSMはカバレッジ保持
 - ・タスクを阻害しない
 - ・5.3節から
パス数は増加しているが
カバレッジの増加には
つながらなかった
 - ・また、69%のfast-followingで選択された状態がマージ成功



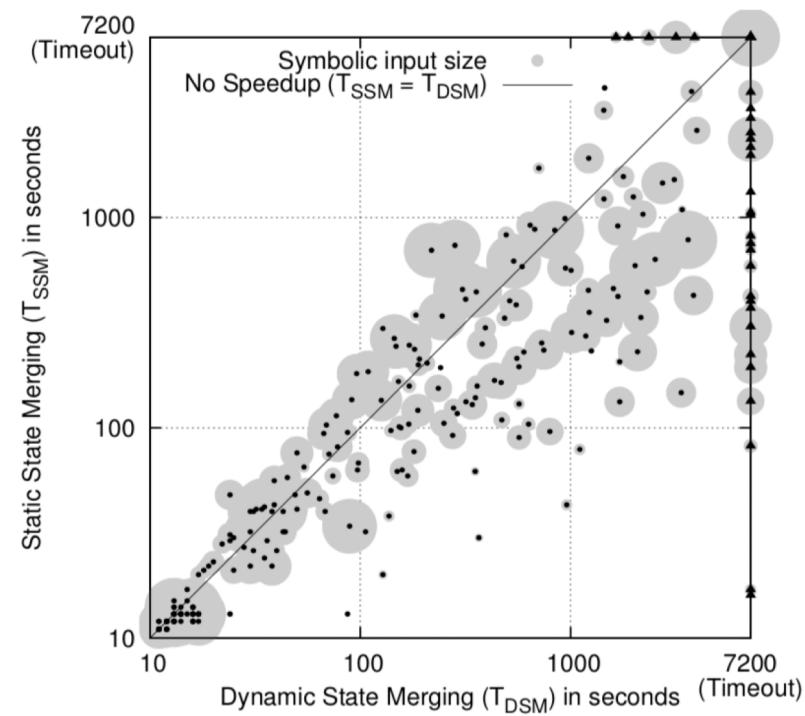
5. EXPERIMENT

5.5 REACHING AN EXPLORATION GOAL WITH DSM

DSMのペナルティの検証 (in 完全探索)

- QCE with DSM vs. QCE with Selective SSM
- パフォーマンス視点
- 右下ならDSM優位

- 結果（散布図）：
 - 多くの例で対角線上
 - →**極端な悪化はなし**
- DSMはSSMに比べて**15%程度の性能ダウン** in 完全探索
 - →しかし一般的に完全探索以外ではDSMが優れるので十分有用



実験結論：QCE with DSMは探索目的を阻害せずに指数的に高速化可能

CONCLUSION

背景 :

- 状態マージは状態数は減るがパフォーマンス悪化の可能性有
- ソルバへの負担が増加

提案手法 :

- **Query Count Estimation (QCE)** :
 - ソルバへの負担が少ないマージポイントを記号実行前に計算
 - ソルバへの負担をホット変数という概念で見積もり
 - ホット変数はパラメータ α, β, κ と再帰式で導出
- **Dynamic State Merging (DSM)** :
 - ワークリストから将来的にマージに適する状態を検出
 - 各状態のヒストリーでマージに適する状態があるかで判定
 - パス探索戦略よりも一時的にマージを優先

実験 :

- 非マージ記号実行と比較して効率化 (KLEE vs. KLEE with QCE)
- SSMと比較してパス探索戦略を阻害せずに高速化を達成 (SSM vs DSM)