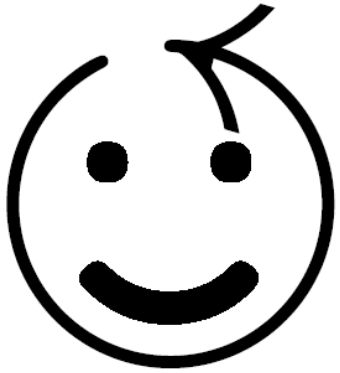
Citation Count: 76

“Whole Program Paths” (James R. Larus)

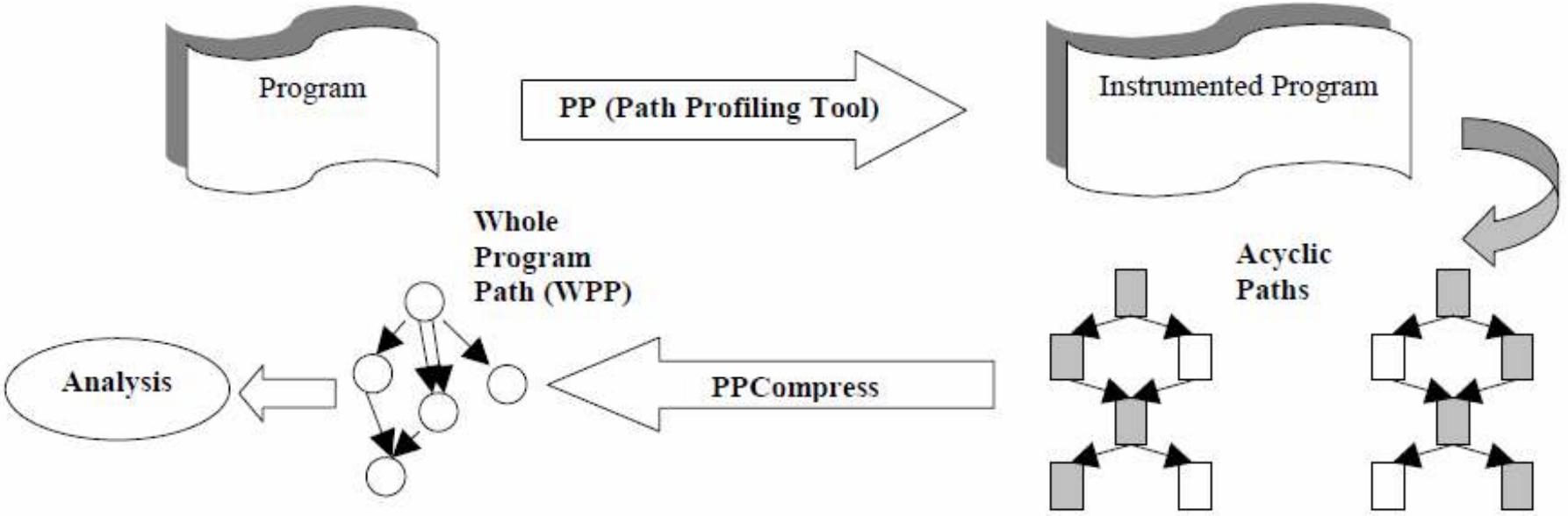
PLDIr #3 (PLDI’99) 2009-11-03

酒井 政裕



これは何?

- プログラム全体の実行トレースを
- 文脈自由文法としてコンパクトに表現、
- それを元に hot subpath を発見



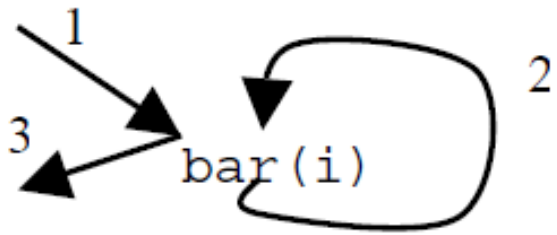
Code → Paths → Acyclic Path Trace

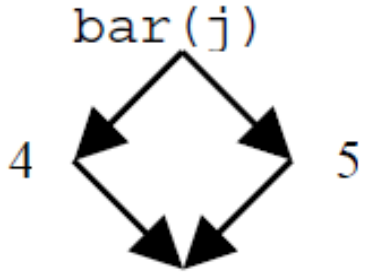
Code

```
for (i=0; i<9; i++)
  bar(i);
```

```
int bar(int j) {
  if (j < 5)
    return j;
  else
    return 0;
}
```

Paths





Acyclic Path Trace

14242424
25252525253

Acyclic Path Trace → SEQUITUR Grammar → WPP

Acyclic Path Trace

14242424
25252525253

SEQUITUR Grammar

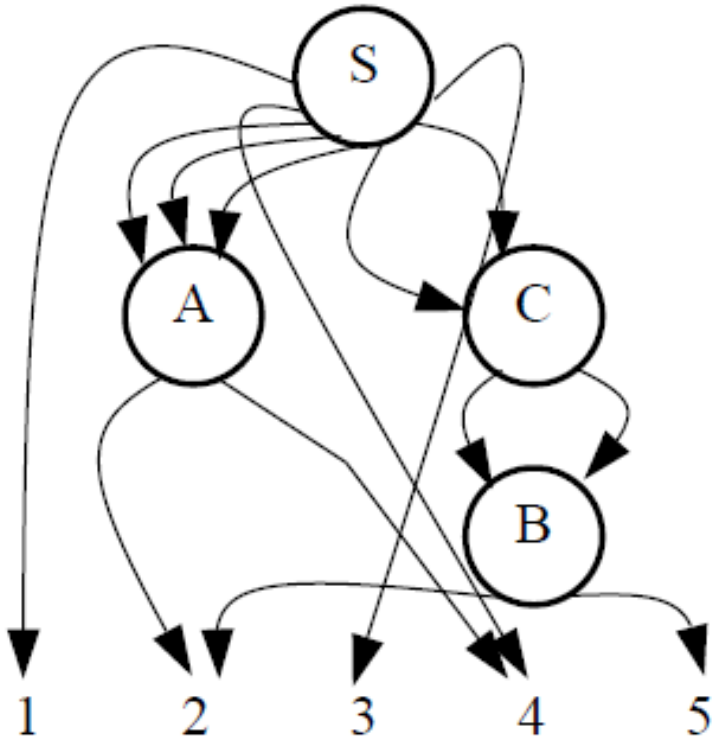
S → 14AAACC3

A → 24

B → 25

C → BB

WPP



文脈自由文法

そのDAG表現

SEQUITURアルゴリズム

- 入力文字列から文脈自由文法を生成するアルゴリズム
- 線形時間
- オンライン

- 次スライドの不変条件を保存しながら、一文字ずつ加えていく。
- この論文では若干改良しているけど、省略

SEQUITURアルゴリズムの不変条件

Diagram uniqueness property:

- ある記号列は、全ての生成規則の右辺のなかに、一度しか現れない
- S→abca の状態で b が追加されたら ab が重複して出現してしまう
- S→AcA, A→ab と変形して回避

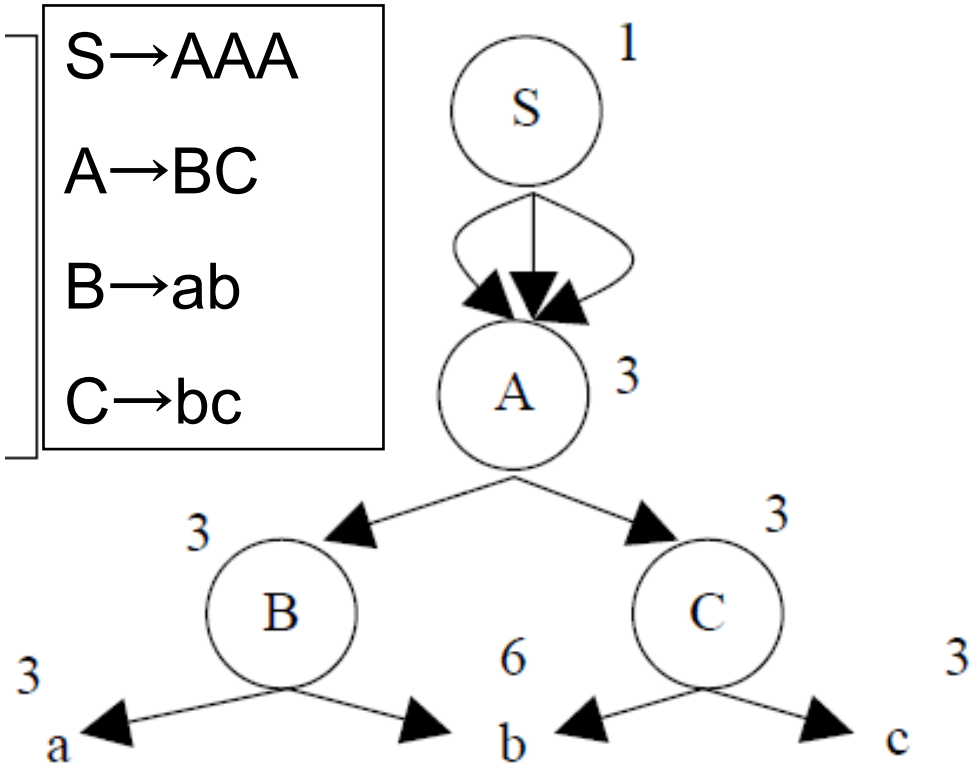
Rule utility property:

- S以外の非終端記号は、2回以上用いられる
- S→BCBA, A→ab, B→Ac, C→Ad の状態で dを追加すると S→DD, A→ab, B→Ac, C→Ad, D→BC となる
- BとCは一度しか使われていないので、インライン化して S→DD, A→ab, D→AcAd

Hot Subpath の解析

- WPPは色々な解析に便利
- ここでは Hot Subpath の発見を取り上げる
- Hot Subpath:
  Acyclic subpath の短い列で、実行コストが高いか、実行頻度が高いかによって、コストの大きいもの。
- 特に極小の Hot Subpath を知りたい。
  - 極小でないものは極小のものを拡大すれば得られるので

Hot Subpath の例



abbcabbcabbcのWPPと実行回数

- a, b, c の実行コストを1として、長さ4未満でコストが6以上の hot subpath は?
- ⇒ ab, bc, bb, ca
- この論文のアルゴリズムは ab, bc を発見。
  - bb, ca はこの二つをextendして得られるとあるが……

アルゴリズム

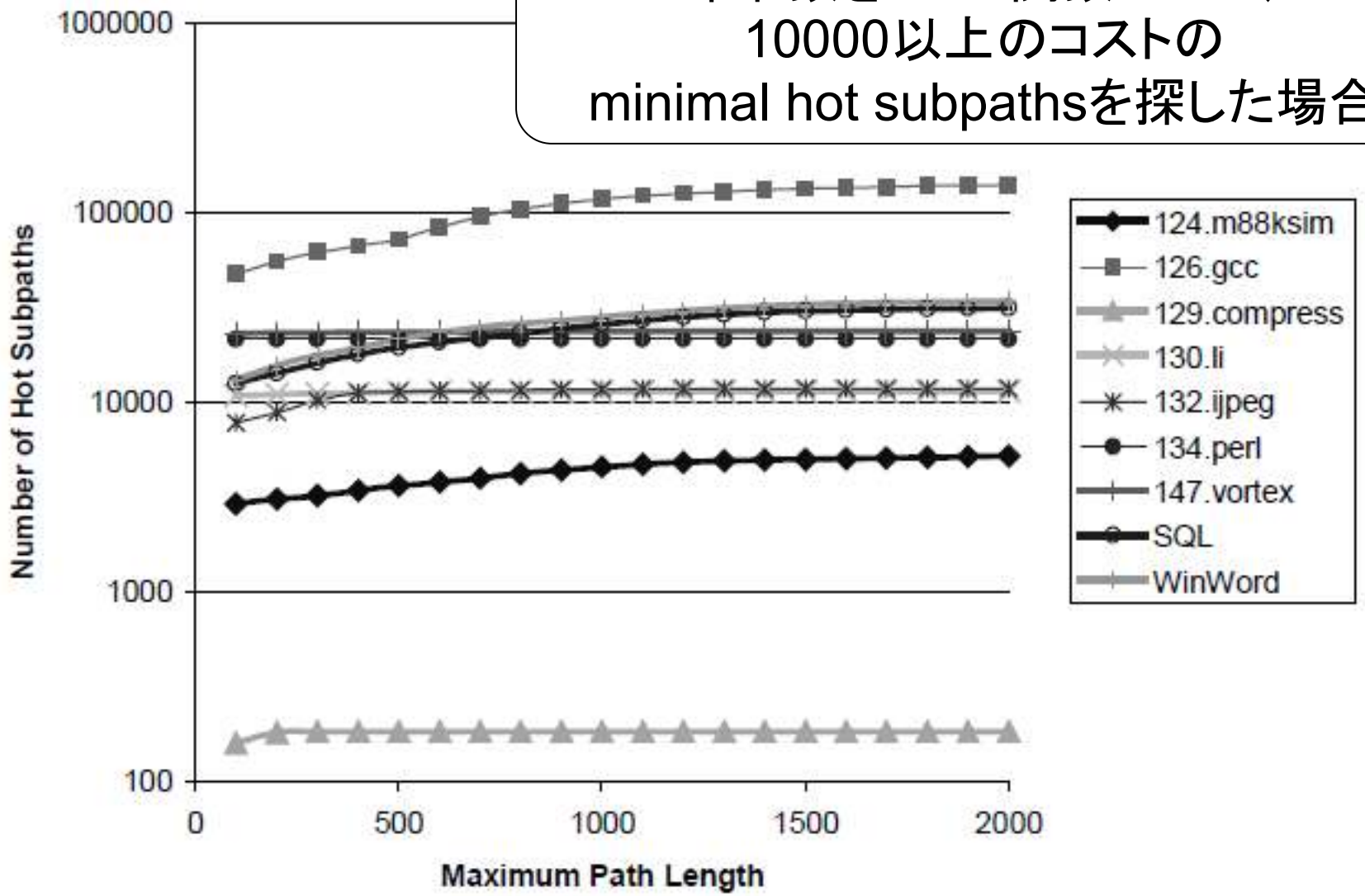
- 基本的にはDAGを post-orderで走査
- (詳細略)

結果とか

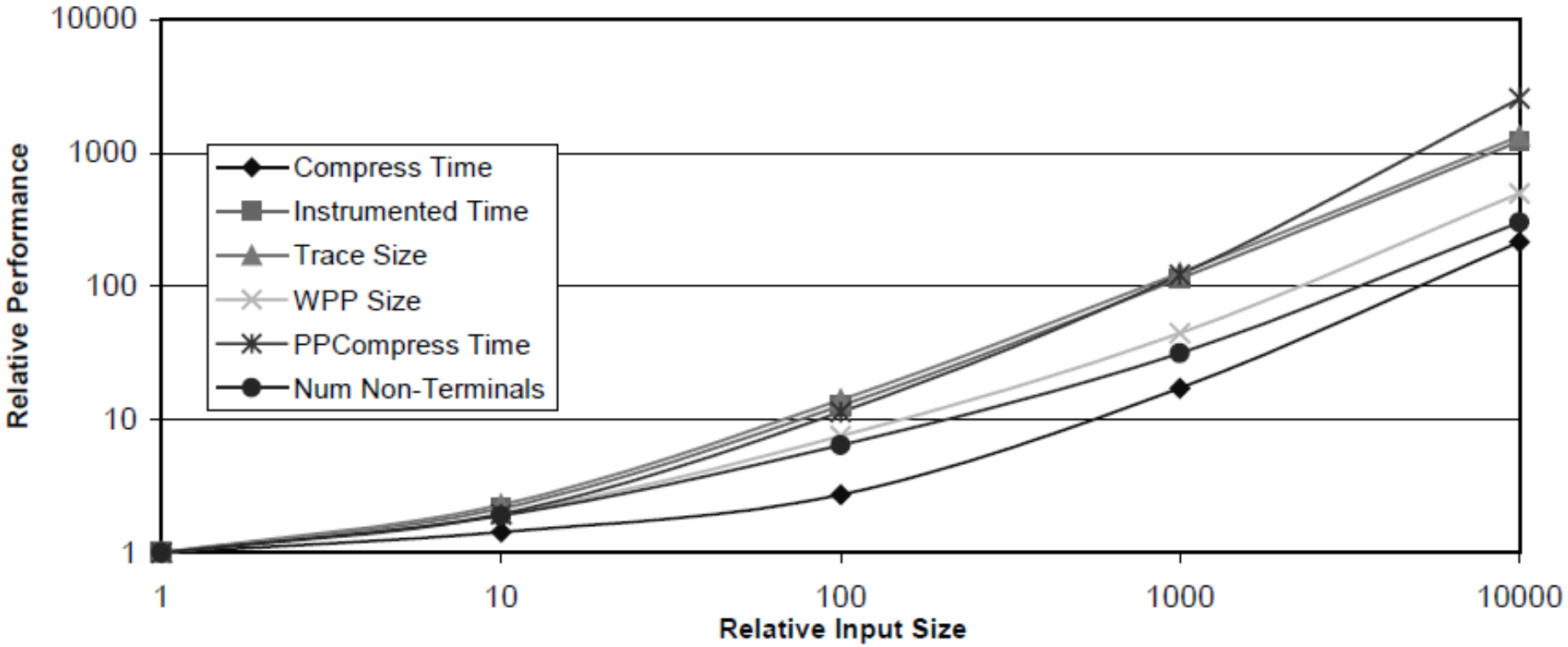
- ベンチマーク対象
  - SPECINT95, Microsoft SQL 7.0, TPC-C benchmark (on-line transaction processing benchmark)
- WPPにすることによるトレースの圧縮率
  - 7.3倍 ～ 392.8 倍
- 発見できた hot subpath
  - Hot path の相当部分は長さ 100 ～ 1000　程度の acyclic paths 。

パス長の上限と hot subpath の関係

命令数をコスト関数にして、
10000以上のコストの
minimal hot subpathsを探した場合



パフォーマンス



compressでベンチマークした結果

感想

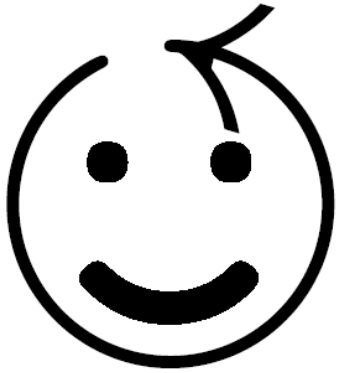
- プログラムのトレースを文脈自由文法を使ってコンパクトに表現するというアイディアが非常に面白かった。

Citation Count: 25

“What is a Recursive Module?” (Karl Crary, Robert Harper, Sidd Puri)

PLDIr #3 (PLDI’99) 2009-11-03

酒井 政裕



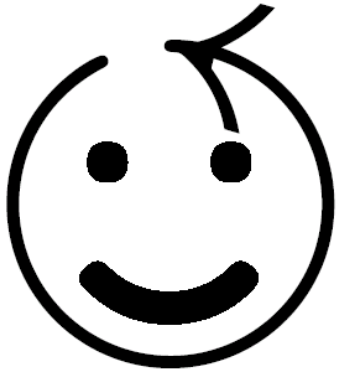
- 階層的なモジュールシステムは、大きなプログラムを構造化する効果的なツール
- だけど、厳密な階層性は、モジュールの依存関係にサイクルがないことを強制してしまう
- 相互依存するようなコンポーネントを、単一のモジュールにまとめることが強制されて、良くない。
- 相互依存を許す再帰的モジュールの提案は色々されているけど、それらを型理論的にきちんと分析してみました。

Citation Count: 19

“Efficient Incremental Run-Time Specialization for Free” (Renaud Marlet et al)

PLDIr #3 (PLDI’99) 2009-11-03

酒井 政裕



- 一度に全部特殊化するのではなく、データが入手可能になるごとに、段階的に特殊化
- 段階的特殊化が嬉しい例
  - 汎用的なマイクロプロセッサのシミュレータ
    - 命令セットによる特殊化と、シミュレーション対象プログラムによる特殊化
  - ネストしたループ
- Tempoを用いて実装
  - 対象言語 C
  - テンプレートはバイナリ。普通のコンパイラで作る。