Fast and Precise Code Clone Reduction by Incremental Analysis (2007/2/7)

米澤研究室 M2 佐藤秀明

コードクローンとは

- 文面が似ているソースコード断片の集合
  - ソースのコピー＆ペーストで発生
- 大規模なシステム開発では有害
  - メンテナンス性の低下

ソースコードを解析して
コードクローンを発見 / 解消できないか？

コードクローンの定義付け

- 「互いに等しいトークン列の組」と定義
  - ただし変数のリネームについては等しいとみなす

```
int f(int x){
    int a = 3;
    return a * x;
}
```

クローン

```
int g(int y){
    int b = 3;
    return b * y;
}
```

コードクローン発見の既存手法

- 接尾辞配列を利用 [Basit et al. 2005]
  - テキスト検索の手法を応用してクローンを発見
- 接尾辞配列：文字列の全接尾辞を辞書順にソート
  - 省スペース
  - 単純な構造

actaat$ 

index	pos	実体
0	3	aat$
1	0	actaat$
2	4	at$
3	1	ctaat$
4	2	taat$
5	5	t$
6	6	$

既存手法の問題

- クローンを発見するだけ
  - どう書き換えるべきか教えてくれない
- 理由：詳細な解析には膨大な時間がかかる

より高速な解析の必要性

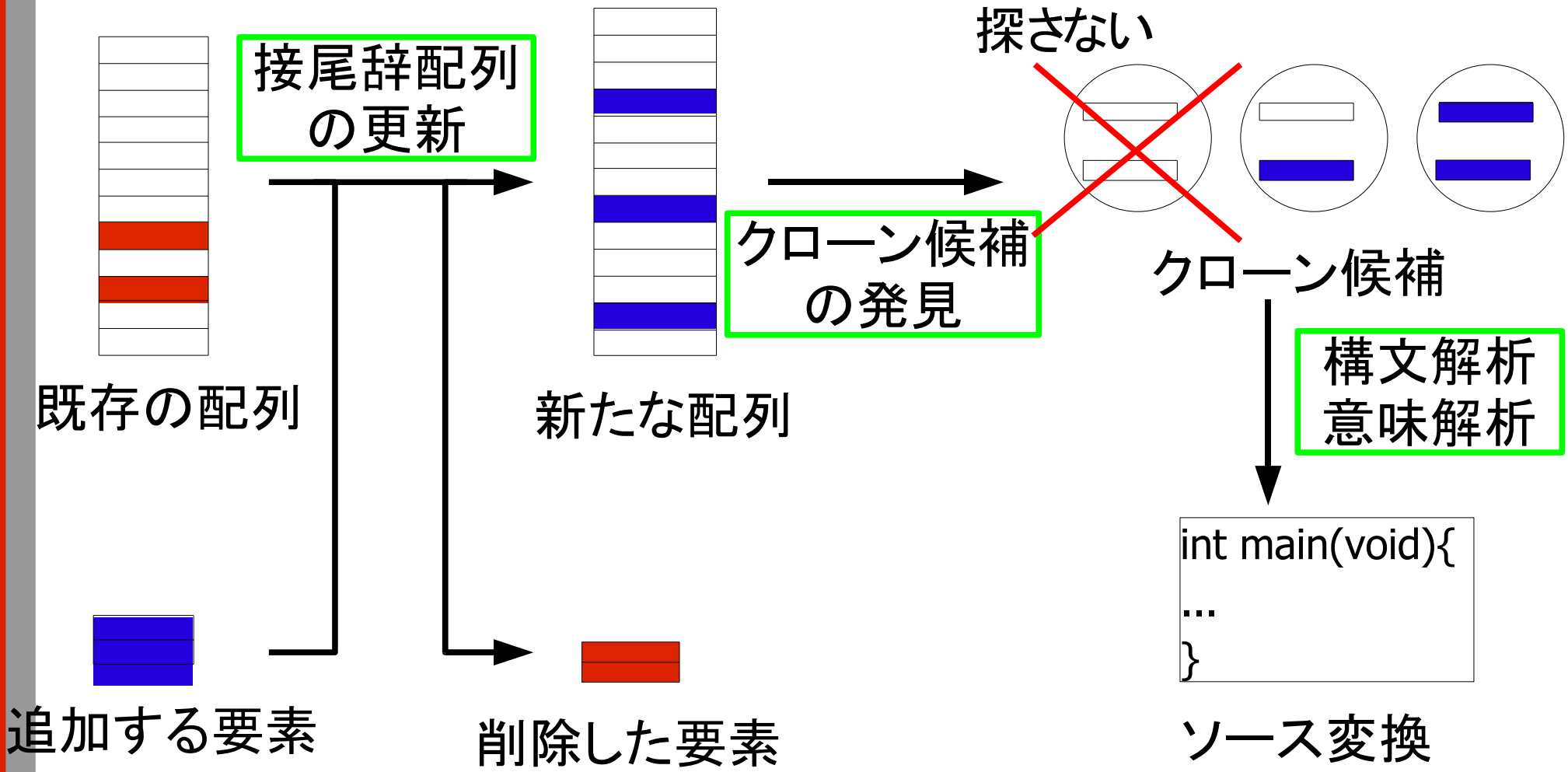
我々のアプローチ

- ソースコードの変化に沿って解析
  - 差分のみ解析することで毎回の計算量を削減

解析手順

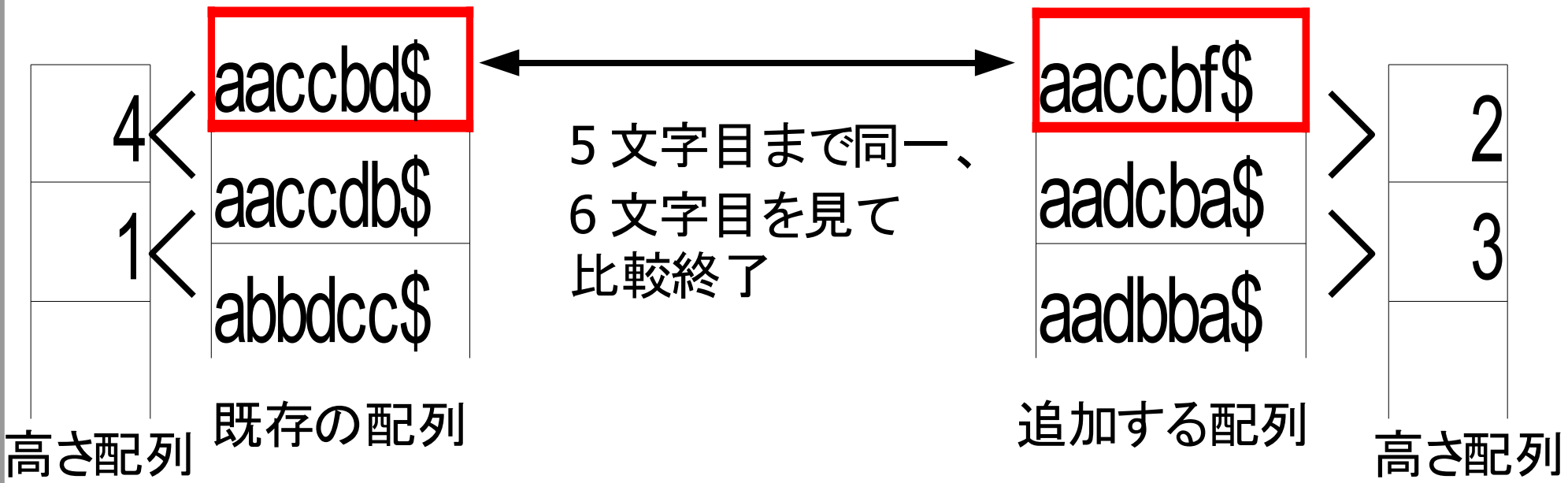
1. 接尾辞配列の更新
   - 前回作った配列に今回生まれた差分を反映
   - 毎回全てを作り直すより高速
2. クローン候補の発見
   - 今回新たに生まれたクローンのみ発見
   - 重い解析にかけるクローン候補の数を削減
3. 構文解析 / 意味解析
   - ソース変換によりクローンを解消
   - メモリ / 時間を大きく消費する解析を実行

解析手順の図


接尾辞配列
の更新
既存の配列
新たな配列
探さない
クローン候補
の発見
クローン候補
構文解析
意味解析
int main(void){
...
}
追加する要素
削除した要素
ソース変換

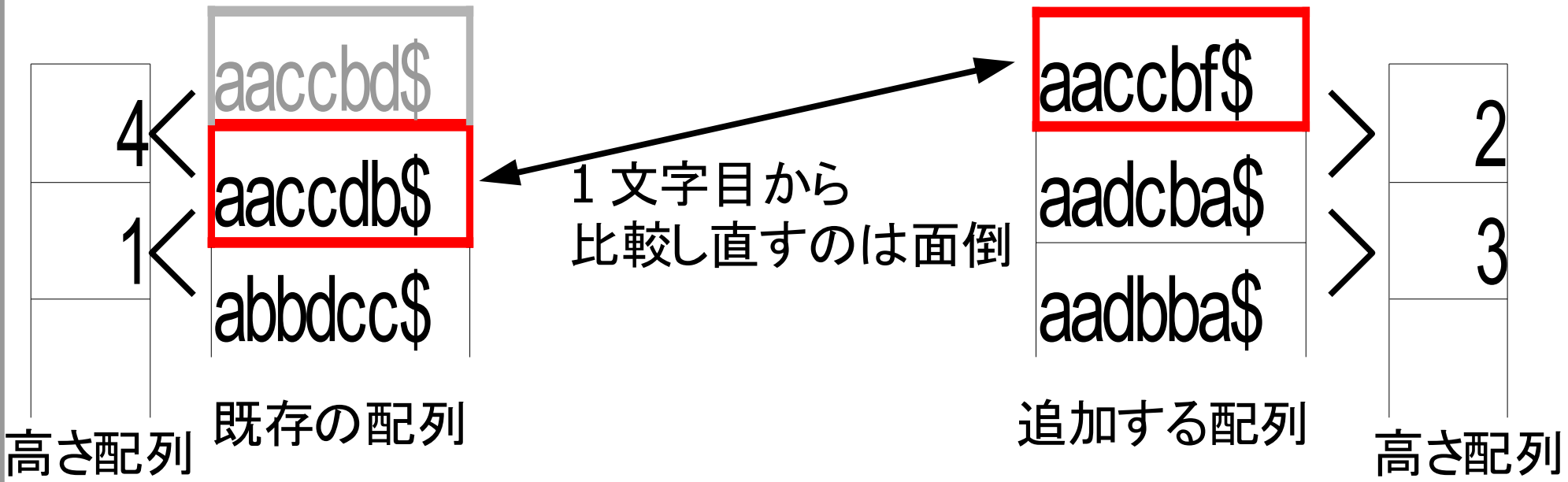
1. 接尾辞配列の更新

- 高さ配列を利用して効率的にマージソート
  - 接尾辞配列上で隣り合う要素間の類似度を保持
  - 各要素の比較にかかる手間を削減



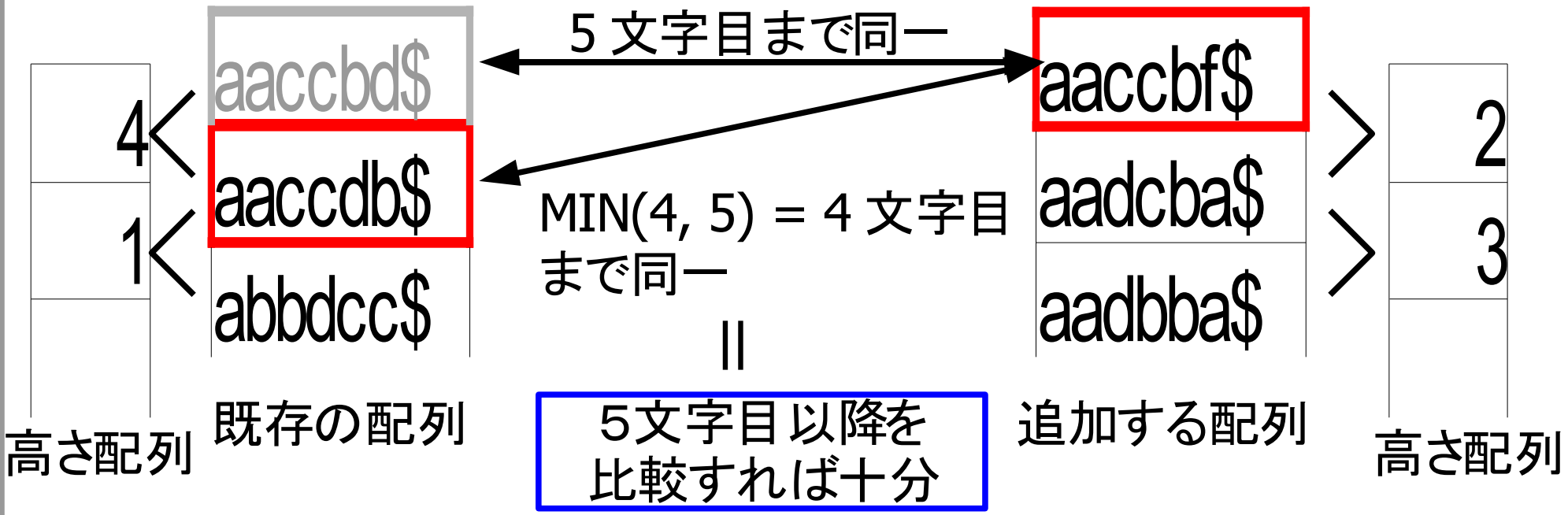
1. 接尾辞配列の更新

- 高さ配列を利用して効率的にマージソート
  - 接尾辞配列上で隣り合う要素間の類似度を保持
  - 各要素の比較にかかる手間を削減



1. 接尾辞配列の更新

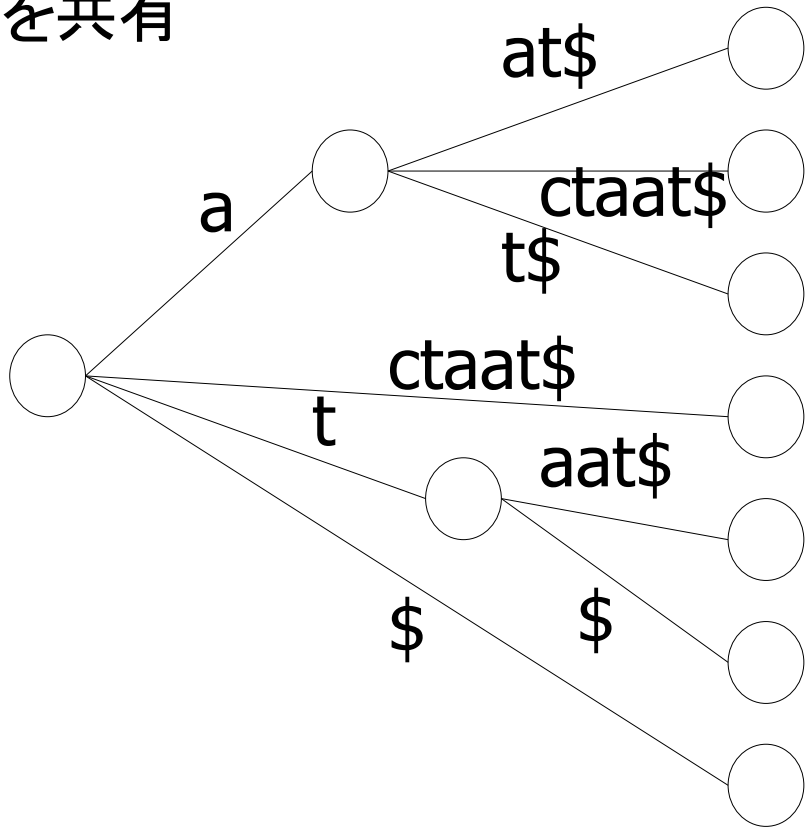
- 高さ配列を利用して効率的にマージソート
  - 接尾辞配列上で隣り合う要素間の類似度を保持
  - 各要素の比較にかかる手間を削減



2. クローン候補の発見

- 接尾辞配列とスタックで接尾辞木をシミュレート
  - 元アイデアは [Abouelhoda et al. 2002]

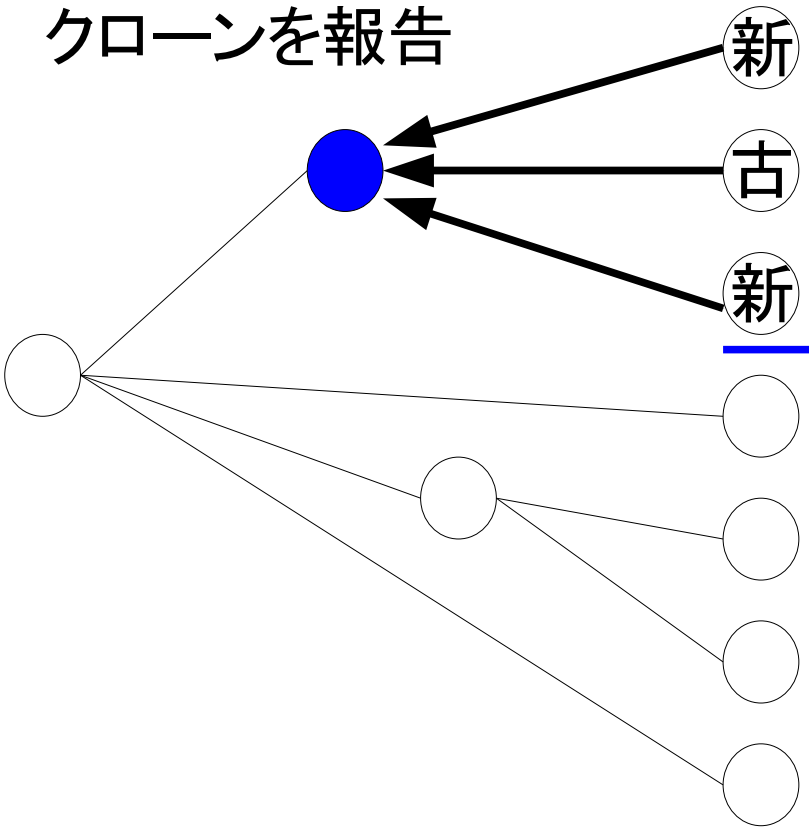
共通 prefix の部分は
枝を共有



index	pos	実体
0	3	aat$
1	0	actaat$
2	4	at$
3	1	ctaat$
4	2	taat$
5	5	t$
6	6	$

2. クローン候補の発見

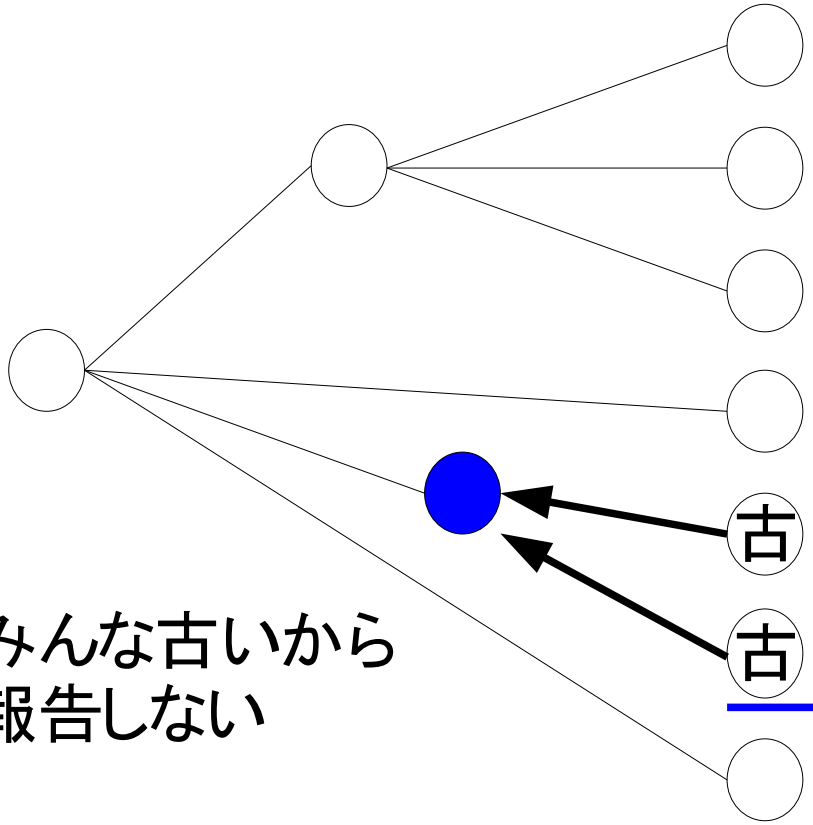
- 子孫ノードが新しい接尾辞かをボトムアップに伝達
  - ノードの pop 操作時に子孫ノードをクローンとして報告



index	pos	実体
0	3	aat$
1	0	actaat$
2	4	at$
3	1	ctaat$
4	2	taat$
5	5	t$
6	6	$

2. クローン候補の発見

- 子孫ノードが新しい接尾辞かをボトムアップに伝達
  - ノードの pop 操作時に子孫ノードをクローンとして報告

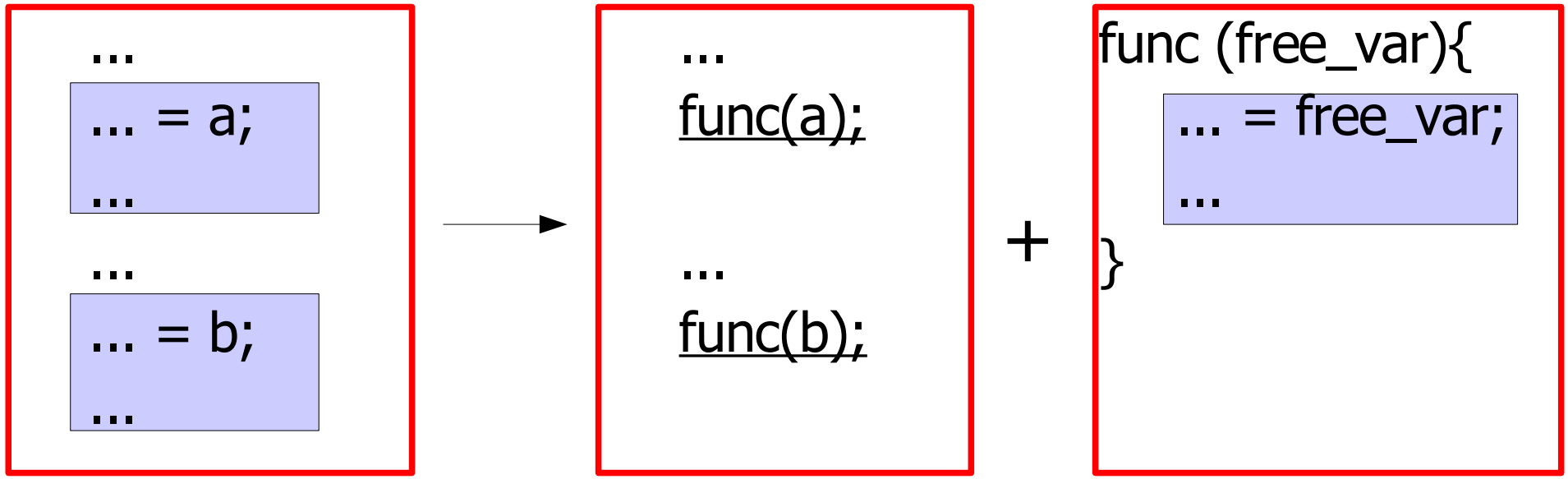


みんな古いから
報告しない

index	pos	実体
0	3	aat$
1	0	actaat$
2	4	at$
3	1	ctaat$
4	2	taat$
5	5	t$
6	6	$

3. 構文解析 / 意味解析

- トークン列ベースのクローン情報を構文木へマップ
  - 1 次元から 2 次元へ
- クローン内の自由変数を計算
- 自由変数を引数として関数抽出



言語選択

- 対象言語：Java
- 実装言語：C, OCaml

実装

- ソースコード読み込み
  - 変数名の違いを吸収 [Baker 1992]
- 接尾辞配列構成
  - [Larsson et al. 1999] を利用
- 接尾辞配列の更新
  - 変更箇所の情報はソースファイル単位で丸ごと差し替え
- ソース変換
  - 実際のソースファイルは改変しない
  - 構文木の変換が可能なことをメモリ上で確認するだけ

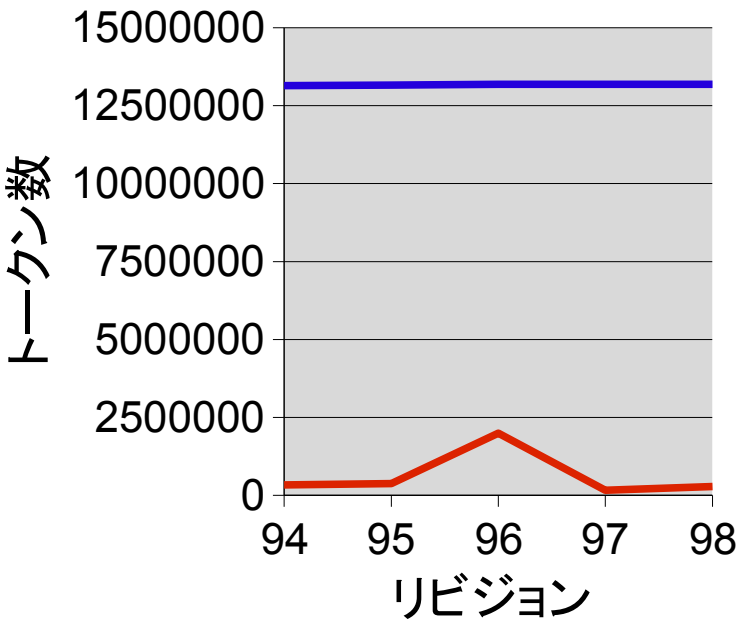
評価

- 目的：差分的手法による解析時間の削減を確認
  - 巨大なソースコードに対する有効性
- 対象ソースコード：Java SE 6 の weekly snapshots
  - ファイル数： 約 15,000
  - 総トークン数： 約 13,000,000
- 解析対象とするトークン列の長さ：100 以上
  - 小さすぎて無意味なクローンを排除

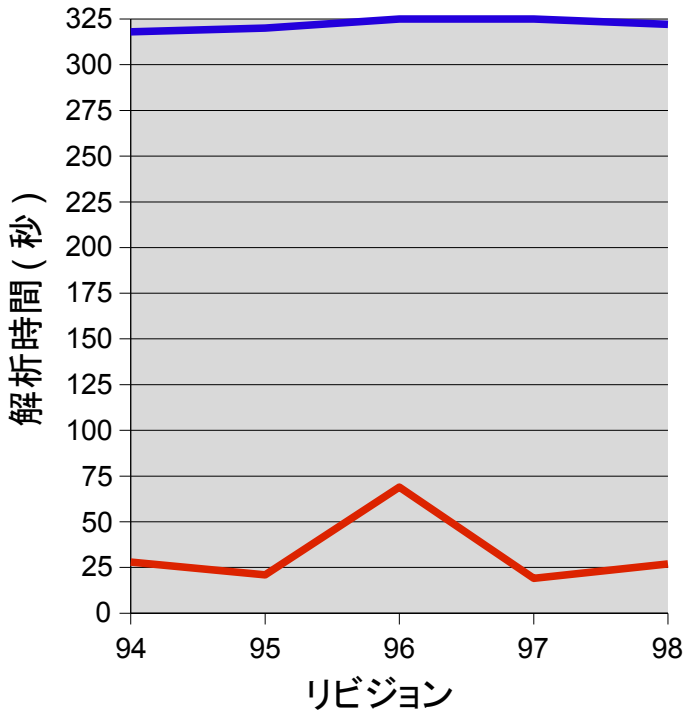
評価結果

- 解析時間を大幅に削減
  - 大きな更新があっても解析時間は許容範囲内

新たに読み込んだトークン数



解析時間



CPU: Pentium 4 2.6GHz
メモリ : 2GB
OS: Linux 2.6.16
コンパイラ : gcc 4.0.2

今後の展望

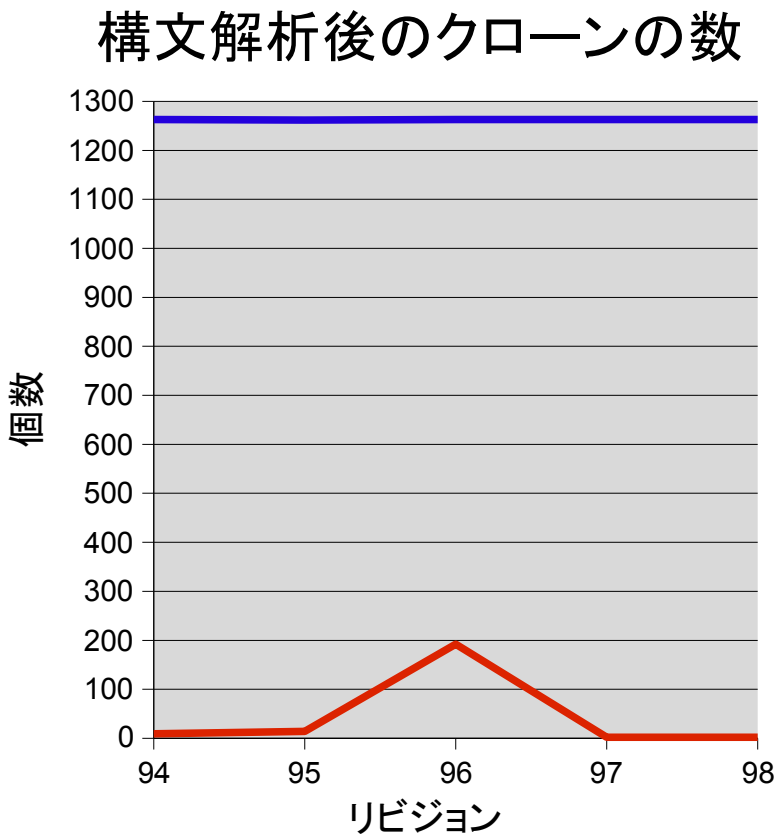
- 多様なリファクタリング手法の採用
  - Generics (Templates) で型を抽象化
- クローン取捨選択法の考案
  - なんらかの定量的指標の必要性
- アルゴリズムの数学的な検証
  - 接尾辞配列操作の正しさ
  - プログラム変換の正しさ

まとめ

- 差分的手法によるコードクローンの高速な解析
  - 接尾辞配列の差分構築
  - 解析するクローン候補数の絞りこみ
- 巨大なソースコードに対する効果を確認
  - ソース変換による冗長性解消までを高速に実現

クローン数の比較

- 重い構文解析前に候補数を絞り込み



CPU: Pentium 4 2.6GHz
メモリ : 2GB
OS: Linux 2.6.16
コンパイラ : gcc 4.0.2

変数名の違いの吸収

- 最近出現した同名変数の現在位置との距離を保持