類似性の複数の基準に対応可能な解析器の必要性について

吉　田　　敦[†]

著者はプログラム解析器の構築支援に関する研究やその応用としての差分抽出を行なっている。ソースプログラムの類似性の基準は、様々な観点によって異なるため、複数の基準を容易に実装したり、既存の類似性判定ツールを適宜カスタマイズできる環境が必要と考える。

表 1　設問への回答

ソース No.	A	B	C	D	X
1	no	yes	yes	yes	yes
2	yes	yes	yes	yes	yes
3	yes	yes	yes	yes	yes
4	no	yes	no	yes	no
5	no	no	no	no	yes
6	yes	yes	yes	yes	yes
7	yes	yes	yes	yes	no
8	no	no	no	no	no
9	no	yes	no	yes	no
10	yes	yes	yes	yes	yes
11	no	no	no	no	no

1. はじめに

著者は約 18 年ほどプログラム解析器の構築支援に関する研究を行なっており[1),2)]、その応用として差分抽出に関する研究も行なっている。差分抽出は、類似性の裏返しとも言えるため、共通する点も多い。研究対象は C 言語であり、Java のプログラムは読めるが、プログラムを書くことはほとんどない。

2. アンケート回答

設問の回答を表 1 に示す。観点 X については、学生のプログラミング演習を実施ている立場から、学生が複製をしたと考えられるという観点から回答を行なった。基本的には、類似している字句の割り合いが多かったり、識別子や何らかの属性だけが異なっているなど差分に含まれる字句の種類に偏りがある場合などが基準となる。

† 南山大学
Nanzan University

2.1 ソース No.1 についての回答理由

この断片は、現時点では、同じ振る舞いを持つクラスであるが、将来においてそのことは保証されず、また、たまたま違う概念のものを表現したときに類似しただけである可能性がある。この場合、継承を用いて共通部分の切り出しを行って管理することが望ましいと考えられる。よって、一つにまとめることは望ましくないが、関連性が強いと考えて他の観点では yes とした。

2.2 ソース No.2 についての回答理由

一つのメソッドの公開範囲が異なるだけであり、統合させることを強く望む記述である。したがって、すべての観点において yes とした。

2.3 ソース No.3 についての回答理由

同じインタフェース名を持ち、メソッドが一つだけ多く存在するかどうかだけの差異であるため、統合を強く望む記述であり、ソース No.2 と同じとなった。

2.4 ソース No.4 についての回答理由

IPv6 と IPv4 という二つの異なる概念を表現したクラスであり、これらは基本的には独立して扱うべきである。ただし、共通のインタフェースを持ち、振る舞いも近いところが多いとも想像できるため、類似している事実を記録しつつ、一方にバグがあれば、もう一方を調べた方が望ましい。ただ、それぞれ独立したプロトコルであり、それぞれが独立して進化する可能性があることや、効率の点を考えても、継承や委譲等を使った共通化は避けるべきと考える。

2.5 ソース No.5 についての回答理由

判断が難しいところではあるが、異なる概念を表現しており、また、特定のアプリケーション等に特化したクラス (例えばテスト用) のように読めるため、基本的には独立したものとして扱い、その関連性も弱いと判断した。

2.6 ソース No.6 についての回答理由

同じインタフェースを定義しているため、可能な限り一つにまとめるべきものである。ただし、Java のバージョンの違いを配慮して定義されている可能性がある。その場合、統合は避けた方が良い。よって、強い関連性があると判断して観点 1 のみを no とし、それ以外を yes とする回答もありえる。

2.7 ソース No.7 についての回答理由

一方は他方のリファクタリング結果であり、同時に存在する意味がないため、完全に統合するべきである。

2.8 ソース No.8 についての回答理由

性能が異なり、また、ソートのアルゴリズムも異なるため、別のクラスとして扱うことが妥当と言える。異なるクラス名を付けることが望ましい。

2.9 ソース No.9 についての回答理由

基本的な型の定義であり、共通化しない方が管理はしやすい。ただし、Byte と Short では概念的に類似しているため、関連性があるものとして管理することは不可欠である。

2.10 ソース No.10 についての回答理由

基本的に同じクラスであり、リファクタリングの有無だけの差であるため、統合するべきである。

2.11 ソース No.11 についての回答理由

基本的に異なる概念を表現したクラスが類似しているだけであり、統合の必要はない。テンプレートのような仕組みがあれば、共通化することで管理コストが下がる可能性があるが、積極的に行うほどではない。

3. 議　　論

単純にコード断片を見て類似性を判定することは難しく、将来的な改変も予測しつつ判定する必要がある。ただ、将来的な改変を予測することは困難であるため、以下のような点から総合的に判断する必要がある。

- 判定者がおかれている状況 (文脈) や責務
- 各断片が所属するプログラムやライブラリ等の成果物との関係
- その断片の過去の履歴

また、比較するときには識別子の名前に意味があるものと捉えるか、構造的に一致していれば良いと捉えるかなど、重視する要素は、必ずしも一律に定められるものではない。比較対象となるソースプログラムの個々の内容や状況に依存する可能性がある。したがって、類似性を判定するツールの構築や既存のツールのカスタマイズが容易に行える環境必要であり、そのためには比較をしやすく、かつ、比較に必要な情報を含んだソースプログラムの解析データが必要である。

Sapid[3] など細粒度の解析系の場合、提供されるデータは基本的には抽象構文木であり、また、差分を求める研究の経験上、抽象構文木の比較はアルゴリズムが複雑にも関わらず、必ずしも高い精度の結果は得られない。単純に字句の並びを比較した方が実装が簡単で、かつ、ある程度の実用性を持った精度で結果を得られることもある。ただし、純粋な字句解析の結果が提供されただけでは不十分なことも多い。字句の種類を区別し、種類によって取り扱いを変えたり、種類に基づく評価方法を用いる場合があるためであり、記号表の作成が必要になる。さらに、字句の並びも出現順にする場合と、抽象構文木に基づいて逆ポーランド形式で並べる場合、またそれらの混合など、複数の方法も考えられる。

さらに、ソースプログラムは構文解析が可能であることを必ずしも前提にはできない。何らかの開発・保守作業の中で比較を行い、作業計画を立てる可能性もある。また、C 言語の場合、前処理前と前処理後のどちらを比較するかという選択があり、前処理前の場合には、そもそも構文解析ができない。このことは、前述の記号表が必要という点と整合できない可能性がある。記号表を正確に構築するためには構文解析が必要になることが多いためである。したがって、記号表の精度を少し落しつつ、構文的に不完全なソースプログラムに対して字句分解と字句の種類の判別を行える環境が必要になる。

現在、そのような考えに基づいて、解析器のプロトタイプを構築している。必要となる精度または粒度の情報を得やすくするため、解析器は各機能のごとに独立し、それらを組み合わせて実現する方法で進めている。実行効率の点では少し問題があるが、ハードウェアの性能向上のおかげにより、実用的な時間内での処理は可能と考えている。

参　考　文　献

1) 吉田敦, 山本晋一郎, 阿草清滋,「意味を考慮した差分抽出ツール」, 情報処理学会論文誌, Vol.38, No.6, pp.1163–1171, 1997.
2) 吉田敦, 鯵坂恒夫,「細粒度の振舞いに基づくプログラム差分理解支援ツール」, ソフトウェア工学の基礎 XV レクチャーノートソフトウェア学 34, 近代科学社, pp.11–20, 2008.
3) http://www.sapid.org/