双方向変換言語を用いたコードクローン管理に向けて

相原 崇弘[1,a)]　日高 宗一郎[1,b)]

概要：ソフトウェア開発において，コードクローンはソフトウェアの保守を困難にする原因であることが知られている．保守を困難にする理由としては，例えば，バグを含むコード片をコピーすることにより生じた，コードクローンにまたがるバグを，系統的に修正することが困難であることが挙げられる．このような修正の支援を行うツールは提案されているが，修正における性質は明らかではない．本研究では，プログラミング中に行われるカット，コピー，ペーストといった動作を追跡することによってコードクローンを特定し，その間に保持されるべき一貫性を双方向変換言語によって記述することにより，一貫性を保持したコードクローンの系統的な編集を柔軟に行う機構の提案を目指す．

キーワード：双方向変換，コードクローン，ソフトウェア保守

1. はじめに

ソフトウェアの大規模化に伴い，ソフトウェア保守は困難となっている．ソフトウェア保守の重要性は高く，システム総費用の中で占める割合は大きい．ソフトウェア保守が困難な理由の一つとして，コードクローンが存在する．コードクローンが発生する主な理由として，コピーアンドペーストが存在する．コピーアンドペーストによって発生したコードクローンは，拡張や修正がなされる．あるコードクローンへの拡張や修正は関係するコードクローンに変更を伝搬する必要性がある．コードクローンに対する変更を伝搬する仕組みとして，例えば，Wit ら [5] や Cheng ら [6] の手法がある．これらの手法では伝搬がどのような性質を持っているかは明らかではない．本研究では，コードクローンの編集の伝搬に性質を持たせることによって，伝搬規則を明らかにすることによって，一貫性を保持したコードクローンに対する系統的な編集を柔軟に行う機構の提案を目指す．編集の伝搬には双方向変換言語の一つである UnQL+ を用いることによって，UnQL+が持っている性質を満たす伝搬の仕組みを提供する．

2. 準備

2.1 抽象構文木

抽象構文木とは，プログラムのソースコードを木によって表現したものである．順序木であり，子の数に制限はない．肥後等 [4] の扱う抽象構文木はノードをソースコードに対応させているが，本研究では，用いる双方向変換言語のグラフデータモデルに基づき枝ラベルに情報を保持する．

[1] 法政大学大学院情報科学研究科
[a)] takahiro.aihara.3i@stu.hosei.ac.jp
[b)] hidaka@hosei.ac.jp

2.2 コードクローン

コードクローンとは，類似または一致したコード片のことである．コードクローンには，様々なものがあるが Bellon らよって以下の 3 つに分類されている [7].

- タイプ 1：空白や改行を除いて全く同じコード片
- タイプ 2：型や変数名，関数名が異なるコード片
- タイプ 3：タイプ 2 に加え，構造が異なるコード片

本研究では，タイプ 1，タイプ 2 に加え，タイプ 3 のコードクローンを扱う．

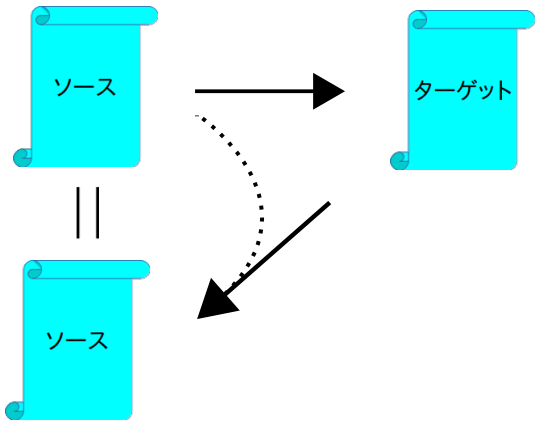

図 1 GETPUT

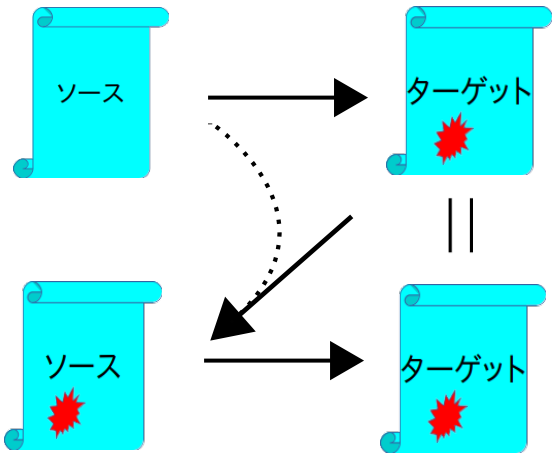

図 2 PUTGET

2.3 双方向変換言語

2.3.1 UnQL+

UnQL+とは，Bunemann ら [1] によるグラフ変換言語 UnQL にグラフ操作用の拡張 [2] を施したものであり，SQL と類似の記法でグラフへのクエリを記述出来る．UnQL+は，日高らによって双方向化された [3] コア言語 UnCAL への翻訳を通じて双方向に実行される．

2.3.2 双方向変換が満たすべき性質

双方向変換とは，二つ (もしくはそれ以上) の情報源の間で変換を介して一貫性を維持する仕組みのこと指す．一貫性を維持する仕組みのことを well-behavedness と呼ぶ．双方向変換では，変換対象の一方をソース，他方の集合をターゲットとすると，ソースからターゲットへの変換を get，ターゲットからソースへの変換を put と呼ぶ．双方向変換言語では，get 又は put を記述することによって双方向変換を行う．get を記述した場合には put を自動導出，put を記述した場合には get を自動導出することによって双方向変換を実現する．双方向変換が提供する well-behavedness は様々なものが存在するが，よく知られているものとして，GETPUT，PUTGET がある．図 1 に GETPUT の概略，図 2 に PUTGET の概略を示す．ソースを s，ターゲットを t とすると，GETPUT，PUTGET は以下のように定義出来る．

$$put(s, get(s)) = s \qquad \text{(GETPUT)}$$

$$get(put(s, t)) = t \qquad \text{(PUTGET)}$$

GETPUT はソースをターゲットへと get によって変換後，ターゲットを変更しない場合 put によってソースが変更されない性質を示す．PUTGET では，ソース s から変換されたターゲットが存在した時に，修正を加えたターゲット t(右辺) とその t を put した結果を用いた get の結果 (左辺) が等しい性質を示す．

UnQL+では，get を記述することによって，put を導出し，GETPUT, WPUTGET と呼ばれる well-behavedness を提供する．WPUTGET は以下のように定義出来る．

$$put(s, get(put(s,t))) = put(s,t) \quad \text{(WPUTGET)}$$

WPUTGET は, PUTGET を緩和した定義である.

2.4 プログラミング動作の追跡

一定の時間隔毎にコードクローンの編集前と編集後の差分を抽出することによって, プログラミング動作の追跡を行う. ただし, 時間間隔については別途定める. 差分を抽出する手法として, Higo ら [4] の手法を使う. この手法により抽象構文木の差分を insert, delete, update, move, duplicate の 5 つのスクリプトで表すことが出来る.

3. 提案手法

3.1 概要

コードクローンの編集は, 関わりのあるコードクローン集合全体に対し適切に行う必要がある. 適切な修正はコードクローン毎のプログラム構造を考慮にいれたコードの挿入, 削除, 移動, 更新, 複製を双方向変換言語によって記述し, 修正を双方向変換によって伝搬することによって実現出来る. 提案手法では, 抽象構文木を通してプログラムの構造を分析し, コードクローン毎への異なる変更を UnQL+により記述することで, 双方向変換による規則性をもった修正の伝搬を行う.

3.2 手順

提案手法では, コードクローンの編集を Higo らが提案する編集スクリプト [4] によって記述し, 編集スクリプトを UnQL+へと変換することによってコードクローン間の同期を行う. 提案手法の概要を図 3 に示す. STEP の詳細は以下の通りである.

STEP1(構文解析) コードクローンを含むソースファイルの構文解析を行い, 抽象構文木を生成する.

STEP2(コードクローンへの双方向変換) ソースファイルから抽象構文木形式でコードクローン部分へと双方向変換する.

STEP3(編集スクリプトの動的な生成) コードクローン毎に特有な編集を編集スクリプトによって表現する.

STEP4(UnQL+への変換) 編集スクリプトを UnQL+へと変換する.

STEP1 では, 抽象構文木の全ての要素を一意に特定可能なようにラベルを付ける. STEP2 では, STEP1 で生成した抽象構文木のコードクローンにあたる部分をコードクローンの木へと変換する. STEP1 で生成した抽象構文木の要素は一意に特定可能な為, コードクローンの木の要素と対応付けが可能である. すなわち, well-behavedness を満たすような変換がソースファイルの抽象構文木とコードクローンの木の間で記述可能である. STEP3 では, Higo らが提案する手法 [4] に従って, コードクローンの木毎の特有な変更を記述し, プログラミング動作の追跡を行う. ただし, Higo らの手法は抽象構文木に対してのみ適用可能な点に注意が必要である. コードクローンの木は, ソースコードの抽象構文木の部分木な為, 抽象構文木であるとは限らない. しかし, 抽象構文木に類似した木である為, Higo らの手法が適用可能であると推測される. STEP4 では, 編集スクリプトを UnQL+へと変換する.

4. 関連研究

4.1 コピーアンドペーストを利用したコードクローン編集

Wit らの手法 [5] では, ユーザの編集動作を追跡することでコードクローンの認識を行い, コードクローンの比較を字句比較することによって行っている. この手法では, コードクローンの比較を単純な字句比較によって行っているため, コードクローン内のあるコードが移動した場合に, 移動した箇所のコードの同期を取ることが出来ない. また, 自動的に表せない同期に対し, 人による同期内容の変更を行うことが出来ない. 提案手法では, 人による同期の変更とコードクローン内のコードの移動を考慮した同期が可能である.

4.2 ルールを用いたコードクローン同期

Cheng らの手法 [6] では, コードクローンの比較を差分を取ることによって行い, 自動的に生成したルールに合わせてコードクローンの編集の伝搬

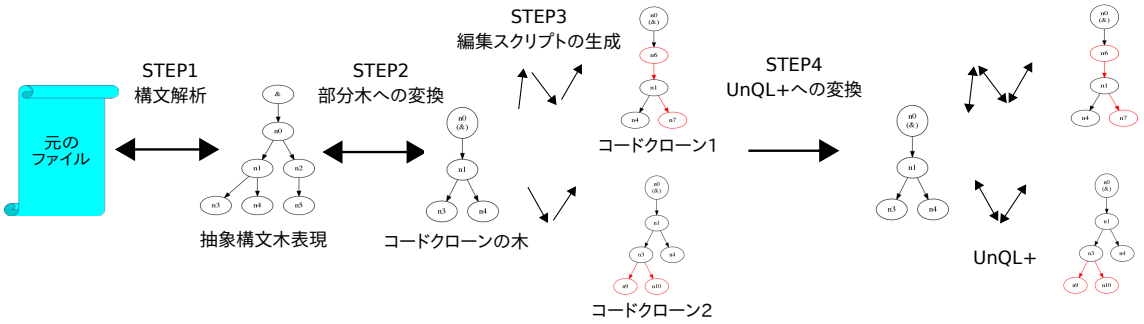

図 3 提案手法の概要

を行う．ただし，ルールの生成を人の手によって行う事も可能である．この手法では，編集の伝搬ルールは自動的に生成された場合には伝搬の規則の理解が困難な可能性があり，コードクローン毎に全ての伝搬ルールを人によって記述することは時間がかかる．提案手法では，一定の規則を持ったコードクローンの編集の伝搬が可能である．

5. おわりに

本論文では，コードクローンへの編集を伝搬する手法を提案した．双方向変換言語がコードクローンへの伝搬へと使われた例は私達が知る限りない．双方向変換によるコードクローンへの伝搬の有用性をこれから確認していく必要がある．

今後の研究課題は以下の通りである．

- 複数クローンへの拡張
- 提案手法の実装
- 実際の開発への適用可能性の分析

参考文献

[1] P. Buneman, M. Fernandez, D. Suciu, “UnQL : a query language and algebra for semistructured based on structural recursion,” *The VLDB Journal 9*, 2000, pages 76110.

[2] S. Hidaka, Z. Hu, H. Kato, and K. Nakano, ”Towards a Compositional Approach to Model Transformation for Software Development,” In *Proc. of the 2009 ACM symposium on Applied Computing*, 2009, pages 164-173.

[3] S. Hidaka, Z. Hu, K. Matsuda, and K. Nakano, ”Bidirectionalizing Graph Transformations,” In *Proc. of the 15th ACM SIGPLAN international conference on Functional programing*, 2010, pages 205-216.

[4] 肥後芳樹, 大谷明央, 楠本真二, ”編集スクリプトへのコピーアンドペースト操作の導入によるコード差分の理解向上の試み,” 情報処理学会論文誌, 2017, pages 833-844.

[5] M. de Wit, A. Zaidman, A. van Deursen, “Managing Code Clones Using Dynamic Change Tracking and Resolution,” In *Proc. of the 25th IEEE International Conference on Software Maintenance*, 2009, pages 169-178.

[6] X. Cheng, H. Zhong, Y. Chen, Z. Hu, J. Zhao, “Rule-Directed Code Clone Synchronization,” In *Proc. of the 24th IEEE International Conference on Program Comprehension*, 2016, pages 1-10.

[7] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, ”Comparison and evaluation of clone detection tools,” IEEE Trans. Software Engineering, vol.31, no.10, 2007, pages.804-818.