社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.

ソースコードのXML表現のための選択例を用いた対話的XPath生成支援

夏目　雅槻†　相澤　遥也†　渥美　紀寿††　小林　隆志†

† 東京工業大学 情報理工学院 情報工学系　〒 152–8552 東京都目黒区大岡山 2–12–1
†† 京都大学 学術情報メディアセンター　〒 606–8501 京都市左京区吉田本町
E-mail: {{mnatsume, yaizawa}@sa., tkobaya@}cs.titech.ac.jp, atsumi.noritoshi.5u@kyoto-u.ac.jp

あらまし　コーディング規約の汎用的な検査手法としてソースコードの構文情報を XML に変換し，XPath を用いて検査する手法が存在するが，構文情報と XML の構造の対応を熟知する必要があり，XPath の学習コストが高いという問題点があった．本研究ではソースコード中の選択箇所を正例・負例として対話的に入力することで，規約に対応するソースコード中の要素を適切に選択する XPath 群を生成し，XPath の複雑度を定義して順位付けすることで XPath を効率よく探索できる手法を提案する．手法をツールとして実装し，二つの課題を例に XML への変換ツールや XPath に関する深い理解を持ち合わせていない利用者が，容易に所望の XPath を生成できることを確認した．

キーワード　ソースコード，XML，XPath，スクレイピング，静的解析

1. はじめに

大規模なソフトウェアの開発現場においてはソースコードのレイアウトやコメントの書き方，識別子の名前等について共通の規約を定め，これをすべての開発者が遵守することで開発効率を高めている．広く知られた規約の例としては GNU コーディングスタンダード，MISRA-C 等が挙げられる [1].

規約は自然言語で定義され，ソースコードレビュー等のように人手によって遵守されているかの確認が行われることもある．また，形式的に定義可能な規約の場合，検査ツールを用いてソースコードを静的に解析することで，機械的に規約が遵守されているか判断する検査手法が有効である．コーディング規約を形式的に定義するためにはソースコード中の着目する要素を選択する方法が必要となるが，これには多くの手法が提案されている [2–4].

ソースコードがプログラム言語の構文規則によって規定される構造文書であることを利用し，ソースコードの解析や変換を容易にするために，構文情報などの静的解析結果を XML タグで表現し各構成要素をマークアップするための XML フォーマットがいくつか提案されている [5,6]．この XML フォーマットに変換したソースコードに対して，特定の規約に従って記述されているかどうかを XPath を用いて検査する手法も提案されている [1]．しかし，XPath は記述力の高さと引き換えに学習コストが高く，また XML への変換ツールがどのようにソースコードを XML へ変換するかという特性を熟知する必要がある．例えば XML への変換ツールの一つである srcML では name というノードを生成するが，このノードは変数の名前のみならず型の名前にも現れる，ということを知っていなければ XPath で変数に関係した規約を書くことはできない．

本研究ではこの問題点を解決するために，ソースコード中の選択したい部分を正例，選択したくない部分を負例として対話的に入力することで，例示されたノードを選択する XPath 群を生成し，コーディング規約に利用する XPath の生成を支援する手法を提案する．また，生成された複数の XPath からユーザが求める XPath を探し出す労力を低減するために，複雑度を計算し，XPath を順位付けする．

本研究ではこれらの機能を実装し，二つのノード選択問題を対象とした適用実験を行った．実験の結果，XML への変換ツールや XPath に関する深い理解を持ち合わせていない利用者が，容易に所望の XPath を生成できることを確認した．

2. XPath

XPath(注1)は 1999 年に World Wide Web Consortium(W3C) によって勧告された，XML 文書内のノードをノード名やノード間の親子関係，兄弟関係，ノードの持つ属性等の条件に基づいて選択する為の表現言語であり，XML 処理において広く用いられている．2018 年 9 月現在では XPath3.1 の策定が進んでいる．本稿で扱う XPath は XPath1.0 の範囲内のものである．以下では本稿で必要となる XPath の情報について説明する．詳細な説明は W3C のサイト(注1)から確認できる．

XPath は以下に示される “ロケーションステップ”(以下，ステップ) の繰り返しを “/” または “//” で区切ったものとして記述される．

　　軸::ノードテスト [述語]

XPath“/child::a” の “child” の部分を軸と呼ぶ．軸は “child” の他に，“descendant”，“parent” や “ancestor”，“preceeding”，“following”，“self” 等が存在し，選択するか判定するノードの XML 木構造上の方向や範囲を指定する．“/child::a” はルートノードが a であるならばそれを選択し，“/descendant::a” は XML 文書中の全てのノード a を選択する．軸を省略した場合は “child” とみなされる．すなわち “/child::a” と “/a” の二つの XPath は同じものである．

(注1)：https://www.w3.org/TR/xpath-3/

“/child::a” の “a” の部分をノードテストという．ノードテストはノードの意味を表し，“a” のようにノード名を指定するか，“node()” を使うことで，軸や直前までのローケーションステップで示された範囲の任意のノードを指定することができる．例えば “/a/node()” はルートノード a の任意の子ノードを選択する．自身と子孫ノード方向に存在する任意のノードを示す，“//” は “/descendant-or-self::node()/” を省略したものである．すなわち以下の二つの XPath は同じものである．

```
/child::a/descendant-or-self::node()/child::b
/a//b
```

述語はノードが満たすべき条件を付与するものであり，ノードテストの後ろに記述する．“/a/b[c]” と書いた場合 “/a/b” を満たすノード b のうち，子ノード “ c” を持つもののみが選択される．すなわち以下の XML において二つ目の b は “/a/b[c]” によって選択されるが，一つ目の b は選択されない．

```
<a><b></b><b><c></c></b></a>
```

述語には text() 関数を用いてノードが含むテキストを指定することもできる．以下の XML では “//b[text()=‘foo’]” と書けば一つ目の b のみを選択することが可能である．

```
<a><b>foo</b><b>bar</b><b><c></c></b></a>
```

複数の述語を and や or，あるいは not 等を用いて連結したり真偽を反転させたりすることが可能である．例えば，上の XML に対して “//b[text()=‘foo’ or c]” や “//b[not(text()==‘bar’)]” は一つ目と三つ目の b を選択することが可能である．

XPath にはこれらに加えて省略記法やインデックス記法等が存在し，似たような XPath でも異なる実行結果を与える等，学習コストは高いと言える．

3. 関 連 研 究

3.1 ソースコード向けの XML フォーマット

srcML [5] は Collard らによって開発された大規模なソフトウェアプロジェクトの効率的な探索，分析，操作のためのプラットフォームのための C/C++/C#/Java ソースコードの XML 表現であり，それを利用したプラットフォーム全体を指す名称でもある(注2)．ソースコードの抽象構文木の構造が XML のタグによって表現され，解析プログラムの記述が容易になることから，多くの解析ツールで利用されている [7].

我々も C 言語の XML 表現形式である CX-model [6] を提案してきた．CX-model では，静的解析プラットフォーム Sapid [8] を利用することで，複数のファイルに跨ってマクロを含めてソースコードを解析し，識別子に id を割り振ることで別の箇所に現れる同一の変数，関数等を判別することができる．

これらの XML 表現に対しては，特定の要素の指定や検索に XPath が用いられる．CX-Checker [1] は CX-model に基づいて XML に変換された C 言語のソースコードに対して XPath 等を用いてルールを定義できるカスタマイズ可能なコーディングチェッカーである．オープンソースの静的解析ツール PMD(注3) でも，ソースコードを XML 文書化し，簡易な検索条件を XPath で記述するインターフェースが用意されている．

XPath を用いない方法として，srcML では XPath に変わるノードの選択手法として srcQL [4] 等を提案している．srcQL は srcML 形式の XML 上で動作するクエリ言語で XPath 及び srcML プラットフォーム上の情報を利用してソースコード中のコード片を取得する．この他にも，XML 形式ソースコードの問い合わせに関しては，Python を基にした問い合わせ言語 CXmlPyQuery [9] なども提案されている．しかしながら，いずれの手法においても対象の XML フォーマットを熟知したうえで，新しい言語を学習する必要があり，その導入コストが課題となる．

3.2 Web スクレイピング

Web 上から情報を抽出する手法は一般に Web スクレイピングやラッピングと呼ばれ，数多くの研究事例やツールが存在する [10,11]．構造化文書である HTML を対象として情報の抽出を行うため，本研究との類似性が存在する．Web スクレイピングでは，コンテンツの内容に基づいて繰り返し構造を検出する手法と，構造的特徴に基づいて抽出する手法が存在する．前者の方法では，例えば，抽出するコンテンツの周辺の文字列を学習する手法 [12] がある．本研究と同様に，構造的特徴のみに基づいて抽出する後者の手法には，HTML 内での類似する繰り返し構造を探索する手法 [13] や，XML の構造を教師付きで学習し判定ルールを抽出する手法 [14]，XPath のグループ化を行うことで抽出ルールを構成する手法 [15] などが提案されている．しかしながら，ソースコードでは繰り返し構造が現れにくく，構造の差異が重要な意味を持つ場合が多いため適用は難しい．

4. 提 案 手 法

4.1 概　　要

本研究では正例・負例を反映した新しい XPath を生成するために，選択範囲の拡大を行う union と縮小を行う difference の二つの演算を定義する．これらはベースとなる XPath と一つの正例/負例から XPath の範囲を拡大/縮小するような XPath の集合を計算する．また，XPath に対して複雑度を定義する．

4.2 正例・負例ノードから XPath を取得

提案手法では，union と difference はそれぞれ二つの XPath に対する演算として定義し，選択された正例および負例に対して，そのノードを選択する XPath を取得して計算に用いる．例えば，以下のような XML においてノード b を選択した時 “/a/b” や “/a/b[text()=‘foo’]” のように，ルートノードから選択したノードまで XML の木構造を辿る上で通過したノードを XPath の各ステップのノードテストとするような XPath を生成し，計算に用いる．

```
<a><b>foo</b></a>
```

4.3 union

union は二つの XPath に対して両方の XPath が選択するノード集合全てを選択するような XPath の集合を計算する．union は以下のように再帰的に定義される．ここで，式中の a, b は任意のノードを表現し，X, Y は 1 ステップ以上の XPath のステッ

(注2)：http://www.srcml.org/about.html
(注3)：https://pmd.github.io/

プ列とする．また，union の演算は可換であり $union(X,Y) = union(Y,X)$ であるため X のステップ数 $\leqq Y$ のステップ数 であるものとする．

$$union(/a,/b) = \begin{cases} /a & if\ a = b \\ /* & otherwise \end{cases}$$

$$union(/a,/X/b) = \begin{cases} //a & if\ a = b \\ //* & otherwise \end{cases}$$

$$union(/a/X,/b/Y) = \begin{cases} // \oplus union(/a/X,/Y) \\ \quad \cup\ // \oplus union(/X,/b/Y) \\ \quad \cup\ /a/ \oplus union(/X,/Y) & if\ a = b \\ // \oplus union(/a/X,/Y) \\ \quad \cup\ // \oplus union(/X,/b/Y) & otherwise \end{cases}$$

$union(/a,/X/b)$ は a と b が同じ場合 $//a$ を，異なる場合 $//*$ を返す．XML においてルートノード a とルートノードでないノード b を選択する XPath を考えることと同義である．

$union(/a/X,/b/Y)$ は $\oplus$ を用いて定義される．a と b が異なる場合はそれぞれの XPath について先頭ノードを取り除いたものと取り除いていないものとの union を計算し，a と b が同じ場合はこれに加えて両方の XPath の先頭ノードを取り除いたもの同士の union を計算して先頭に"/a"を付加する．これは二つの XPath の前方が一致していた場合に残った部分の共通部分を union で計算すること，一致していない場合は片方の XPath の先頭が一致することを諦め，残った部分ともう一つの XPath の共通部分を求めることを意図している．$\oplus$ は，第 1 項に XPath の部分列 lhs，第 2 項に XPath の集合 RHS を取り，第 2 項の各 XPath $rhs \in RHS$ について $prepend(lhs,rhs)$ を計算した結果の集合を返す．$prepend$ は第 1 項が $/a/$ である場合は第 2 項の XPath の先頭に $/a$ を付加し，第 1 項が $//$ である場合は第 2 項の XPath の先頭を $//$ に変更する演算である．定義を以下に示す．

$$lhs \oplus RHS = \{\ prepend(lhs,rhs) \mid rhs \in RHS\ \}$$

$$prepend(lhs,rhs) = \begin{cases} /a//X & if\ lhs = /a/\ \wedge\ rhs = //X \\ /a/X & if\ lhs = /a/\ \wedge\ rhs = /X \\ //X & if\ lhs = //\ \wedge\ rhs = //X \\ //X & if\ lhs = //\ \wedge\ rhs = /X \end{cases}$$

実際の計算においては $union$ の計算において動的計画法を用いることによって二つの XPath のステップ数 N, M に対し $O(NM)$ 程度のステップ数で計算が可能である．

4.4 difference

difference は二つの XPath に対して一つ目の XPath をベースとして二つ目の XPath が指すノードを含まないような XPath の集合を計算する．union と異なり，除外対象とされたノード集合を指定する方法として “parent”，“ancestor”，“child”，“descendant” の四つの軸を用いた候補を生成する．difference は以下の手順で計算される．

（1） 二つの XPath でノードテストが同じであるようなステップのペア集合を計算する．

（2） 各ペアについてペアのうち二つ目の XPath のステップから相対的に見て (a) 親ノード，(b) 先祖ノード，(c) 親・先祖の構造，(d) 子ノード，(e) 子孫ノード，(f) 子・子孫の構造にあたるステップを見つけ，それらを除外するようペアのうち一つ目の XPath のステップに述語を付加した XPath を生成する．

実際に XPath p1:“/A//D” と p2:“/A/B/C/D” について $difference(p1,p2)$ を計算すると (a) から (c) についてノードテスト D に注目した場合，(d) から (f) についてノードテスト A に着目した場合に対応する結果の一つとしてそれぞれ以下の XPath が得られる．

(a) /A//D[not(parent::C)]
(b) /A//D[not(ancestor::B)]
(c) /A//D[not(parent::C/parent::B)]
(d) /A[not(B)]//D
(e) /A[not(descendant::C)]//D
(f) /A[not(B/C)]//D

(a) から (c) について，二つ目の XPath において注目したノードテスト D の親 (図 1)・先祖 (図 2) のノードとそれらの構造 (図 3) から “parent::C”, “ancestor::B”, “parent::C/parent::B” がそれぞれ得られる．これらを含まないようにベースとなる XPath“/A//D” のステップ D に述語を追加することで上記の XPath が得られる．(d) から (f) についても同様に，二つ目の XPath において注目したノードテストの A の子 (図 4)・子孫 (図 5) のノードとそれらの構造 (図 6) から “child::B”, “descendant::C”, “child::B/child::C” がそれぞれ得られる．これらを含まないようにベースとなる XPath“/A//D” のステップ A に述語を追加することで上記の XPath が得られる．図 1 から図 6 の各図は負例として与えたノードと生成された XPath による選択範囲を示している．負例として与えたノードを青で，生成された XPath の選択範囲を黄色で示した．

4.5 不適切な XPath の除去

union と difference は既存の XPath 候補と新たに選択した正例/負例から，新たに選択した正例/負例を正しく選択する XPath を計算するが，生成される XPath は必ずしも既存の正例・負例を正しく選択するわけではない．

union/difference の計算後，生成された XPath を評価し，既存の正例を選択できない，あるいは既存の負例を選択する XPath を除去し，新たな XPath 候補とする．

4.6 XPath の複雑度

同じ正例・負例の組から生成した XPath であっても，条件が少なく簡潔なもの，条件が多く複雑なものがある．また，同じ対象を選択する XPath が複数存在することもある．複数の XPath を生成したとき，よりシンプルな XPath の方が有用であろうという直感から，本研究では XPath に対して複雑度を定義し，生成された XPath を複雑度が低い順で順位付けする．複雑度はステップの数と述語の項数から計算する．

$Complexity$ = ステップ数 + 述語の項数 − 0.5 ∗ ”//”の数

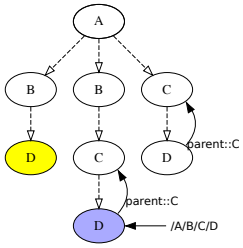

図 1　/A//D[not(parent::C)] の選択範囲

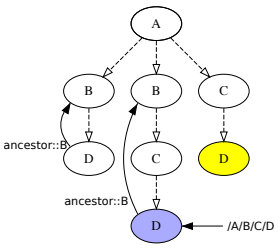

図 2　/A//D[not(ancestor::B)] の選択範囲

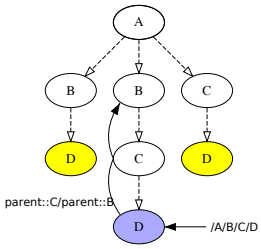

図 3　/A//D[not(parent::C/ parent::B)] の選択範囲

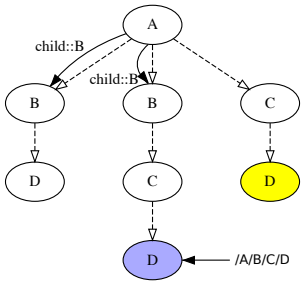

図 4　/A[not(B)]//D の選択範囲

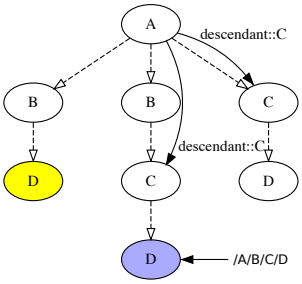

図 5　/A[not(descendant::C)]//D の選択範囲

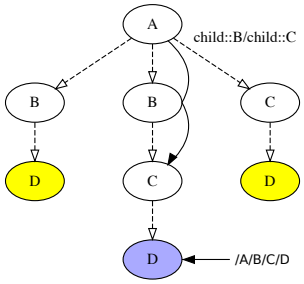

図 6　/A[not(B/C)]//D の選択範囲

5. 実　　装

本研究では提案手法をもとにした GUI による対話的な XPath の生成手法を実装した．提案手法はユーザに XPath の選択範囲を提示し，追加したいノードを正例，削除したいノードを負例として入力させ，より適切な新しい XPath を生成する．

5.1　XPath 生成の流れ

図 7 に XPath 生成処理の概略を示す．ユーザが XPath を得るまでの手順は以下の手順 (1)〜(7) から構成されている．

（1）　ソースコードを XML へ変換
（2）　XML を HTML へ変換
（3）　ユーザによる正例/負例の選択
（4）　ユーザが選択した正例/負例を追加
（5）　XPath 候補の生成
（6）　複雑度を用いた XPath の順位付け
（7）　生成された XPath の選択範囲を HTML に反映

本実装では，手順 (1) に srcML [5] を用いた．srcML を利用することで，図 8 に示すように，ソースコードは XML フォーマットに変換される．また，手順 (2) では，XML の構造を保持したままソースコードのように描画するために，XSLT を用い

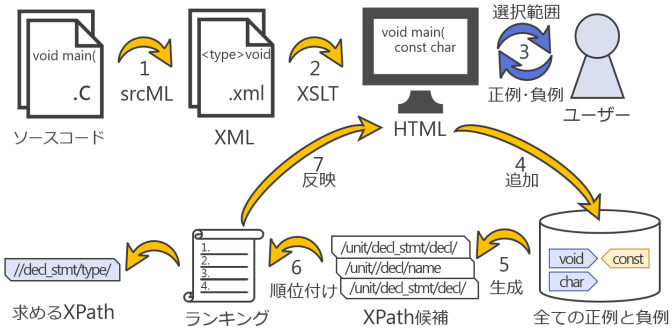

図 7　実装の概略図

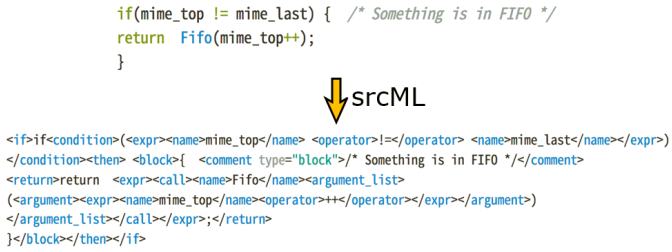

図 8　ソースコードから XML への変換

て HTML に変換する．HTML のノードは XML のノードと一対一に対応しているため XML のノードを選択することと同等である．この HTML を用いてブラウザを介してユーザにソースコードを提示し，ソースコードから HTML 上のノードを正例/負例として対話的に選択できる UI を実装した．

手順 (5) では，初めて XPath 候補を生成する場合は 4.2 節で述べた手法で XPath を生成し，これを XPath 候補とする．二度目以降の生成については提案手法に基づき，既存の XPath 集合と新たに選択した正例/負例から XPath 候補を新たに生成する．生成された XPath 候補のうち既存の正例・負例を正しく選択するものを新たな XPath 集合とする．

生成した XPath が選択する範囲は，手順 (7) で HTML にフィードバックし，意図したノードを選択できているかどうかをユーザがブラウザ上で確認することができる．意図したノードが選択できていなかったり意図しないノードを選択してしまった場合は手順 (3) に戻ってさらに正例/負例を追加する．正例・負例の追加を行って XPath について十分な確度が得られた場合，XPath を決定する．

5.2　計算量の低減

負例を追加した際の計算には既存の XPath 集合の各 XPath のステップ数 N_i と負例から得られた XPath のステップ数 M に対して $\sum_i N_i^2 M^2$ 程度のステップが必要なため XPath 集合と各 N, M が大きくなると生成に時間がかかる場合がある．この問題を低減するために，手順 (5) の際に，既に生成されている XPath の中に追加された正例・負例を正しく選択しているものがあれば XPath の再計算を行わずにその XPath だけをランキング対象とするようツールを実装した．

5.3　XPath の比較

生成される XPath は一般に複数存在するが全ての候補の中から適切な XPath を探し出す，あるいは適切な XPath が無い

ことを確認するのは非常に煩雑で困難である．

そのためツールの実装に際して二つの XPath の選択対象を比較し，その差分や共通範囲を表示する機能を実装することで所望の XPath が生成されているか，すばやく判断できるよう，また，所望の XPath が存在しない場合にすばやく次の正例/負例を追加できるようにした．

6. 評　　価

6.1 概　　要

本研究では評価すべき観点として，1) 実際に XPath を生成することができるか，2) 少数の例示によって XPath を生成できるか，3) 複雑度による順位付けは適切か，4) 実用的な時間で処理できるか，の四つの観点を設定した．この観点に基づいて，ケーススタディとして二つのノード選択問題を設定した．設定に際し union/difference の実行時間が XPath のステップ数に影響を受けることに鑑みて，選択対象のノードを示す XPath のステップ数が大きいものと小さなものとなるように注意した．

各問題に対して，規約設計者として筆頭筆者が実装ツールを用いて XPath を生成した．規約設計者は srcML の仕様を調査しないものとし，生成された XPath ランキングの最上位を確認し，条件を満たすまで正例か負例を追加することとした．評価のために，条件を満たす XPath を得るために要した正例と負例の数，提示された XPath ランキング上位の内容，各 XPath 候補計算にかかった時間を記録した．対象のソースコードには，nkf.c(バージョン 1.6) を利用した．

6.2 ケース 1: ステップ数が少ないケース

選択問題を「グローバル変数を選択する」と定め XPath 生成を行った．正例としてグローバル変数とグローバル配列変数の二つを選択した．取得された XPath はそれぞれ

```
x = /unit/decl_stmt/decl/name
y = /unit/decl_stmt/decl/name/name
```

であった．負例としてローカル変数を一つ選択し，取得された XPath は

```
z = /unit/function/block/decl_stmt/decl/name
```

であった．結果生成された XPath($difference(union(x,y),z)$) は`/unit/decl_stmt//name` と`/unit/decl_stmt/decl/name` の二つで，計算時間は 66ms であった．得られた 1 番目の XPath の選択範囲の一部を表したものが図 9 である．XPath の選択範囲は黄色，正例が赤，負例が青である．いずれの XPath もグローバル変数を選択対象とし，ローカル変数は選択対象としない．しかしこれらの XPath は，ソースコード上ではグローバル変数の型名 “int” 部分も選択していた．これは型名を表すノードにも “name” と名付ける srcML の仕様に起因する．実験ではこの知識を利用せず，最上位に提示された XPath を選択し，単に “int” を負例として追加した．

負例として追加した “int” から取得された XPath は

```
w = /unit/decl_stmt/decl/type/name
```

であった．$difference(difference(union(x,y),z),w)$ を計算し，計算時間は 182ms であった．計算の結果，10 個 XPath が生成され，複雑度が低い順に順位づけした上位 5 個の XPath は以下の通りであった．

```
int line = 0;    /* chars in line */
int prev = 0;
int      fold_f  = FALSE;
int      fold_len  = 0;
```

図 9 “/unit/decl_stmt//name” の選択対象の例

```
int line = 0;    /* chars in line */
int prev = 0;
int      fold_f  = FALSE;
int      fold_len  = 0;
```

図 10 “/unit/decl_stmt//name [not(./parent::type)]” の選択対象の例

```
/unit/decl_stmt//name[not(./parent::type)]
/unit/decl_stmt//name[not(./ancestor::type)]
/unit/decl_stmt//name
                [not(./parent::type/parent::decl)]
/unit/decl_stmt/decl//name[not(./parent::type)]
/unit/decl_stmt/decl//name[not(./ancestor::type)]
```

得られた XPath は全て，“int” を選択しないものであった．最上位の XPath で選択範囲の一部を図 10 に示す．

最上位の XPath の選択対象をさらに確認したところ，グローバル変数の初期化に使われたその他の変数 (`FALSE`) も選択対象としてしまうものであった．次の繰り返し処理として，この変数 “FALSE” を負例として追加した．取得された XPath は

```
v = /unit/decl_stmt/decl/init/expr/name
```

であった．difference を計算した結果 45 個の XPath が生成され，計算時間は 631ms であった．複雑度が低い上位 5 個の XPath は以下の通りであった．

```
/unit/decl_stmt//name[not(./parent::type)
                and not(./ancestor::expr)]
/unit/decl_stmt//name[not(./ancestor::type)
                and not(./ancestor::init)]
/unit/decl_stmt//name[not(./ancestor::type)
                and not(./ancestor::expr)]
/unit/decl_stmt//name[not(./parent::type)
                and not(./ancestor::init)]
/unit/decl_stmt//name[not(./ancestor::type)
                and not(./parent::expr)]
```

一番上位の XPath はルートノード直下の宣言文内の name で型及び式に含まれないものを指し，条件を満たす XPath であった．なお 2 番目と 4 番目の XPath はグローバル配列変数の大きさを指定する式中の識別子を誤って指定してしまう．これらは expr タグの中にあるが init タグの中にはないためであり．追加の正例・負例の選択が必要である．

6.3 ケース 2: ステップ数の多いケース

選択問題を「条件式中の代入文を選択する」と定め XPath の生成を行った．代入演算子を持つ文 (Expr ノード) を選択できる XPath は現在の手法では生成できないため，条件式中の代入演算子を選択する XPath を生成することを目標とした．正例として if 文，while 文の条件式中に存在する代入演算子をそれぞれ一箇所選択した．取得された XPath は以下に示すとおり，12 ステップ，7 ステップであった．

```
x = /unit/function/block/if/else/block/while
```

```
      /block/if/condition/expr/operator[text() = "="]
y = /unit/function/block/while/condition
      /expr/operator[text() = "="]
```

また，負例として条件式ではない箇所に存在する代入演算子を一箇所選択した．取得された XPath を以下に示す．

```
z = /unit/function/block/expr_stmt/expr
      /operator[text() = "="]
```

union(x,y) の計算の結果，z を選択しない XPath が生成されたため 5.2 節の通り difference の計算は省略され，正例・負例を適切に選択できる 68 個の XPath が生成された．計算時間は 1613ms であった．複雑度が低い順に順位付けした上位 5 個の XPath は以下の通りである．

```
//condition//operator[text() = "="]
//while//operator[text() = "="]
//condition/expr/operator[text() = "="]
/unit//condition//operator[text() = "="]
//while//expr/operator[text() = "="]
```

最上位のものは条件式 (condition ノード) 中の代入演算子を指し，求める XPath であった．なお，2 番目と 4 番目の XPath は while 文中の任意の代入演算子を選択しており不適である．正しい XPath を生成するには while 文外での条件式中の代入演算子を正例として追加したり while 文中の条件式外での代入演算子を負例として追加したりする必要がある．

6.4 議　　論

ケース 1 では，srcML の仕様に起因する問題を，srcML での XML 構造や XPath の知識を必要とせずに，最上位に提示された XPath によって選択されるプログラム中の要素を確認し，負例を追加することのみで不要な要素を除外して目的の XPath を生成できることを確認した．さらにケース 2 のように候補箇所が長いステップ数を要する対象であっても同様に XPath を生成できることを確認した．このことから，XML への変換ツールに対する理解の浅い，あるいは XPath に習熟していない規約設計者であっても，容易に所望の XPath を生成できるものと考える．また，いずれのケースも最上位の XPath を選択するだけで生成できており，ケース 2 のようにステップ数が大きくなった場合でも実用的な時間で生成できることも確認できた．

本項で挙げた二つの例については適切に動作する XPath を生成できたが，XML においてネストの深い位置にあるノードを負例として選択した場合に生成される XPath の数が多くなること，その結果，正例・負例集合に対する選択判定回数が増加し実行時間が増大するという問題が確認された．対話的な XPath 生成という形態では実行時間は重要であるため，複雑度の小さな XPath についてのみ判定を行う，“/unit//something” と “//something” 等の選択結果が等しくなる XPath を統合する等の改善を検討している．また，本研究ではソースコードを XML へ変換する手法として srcML を用いたが，他の変換手法を利用することで XPath の計算時間や表現力が改善するかの検証が必要がある．

7. おわりに

本研究ではコーディング規約の静的検査を行うツール等での利用を念頭に，XML 上のノードを正例・負例として与えることで，それらを適切に選択可能な XPath を生成する手法を提案した．また，XPath に対して複雑度のメトリクスを導入し，この手法によって生成された XPath を順位付けして規約設計者が求める XPath を発見するための労力を低減する手法を提案した．以上の手法を実装し，二つの例で XPath に関する知識がなくとも選択したいノードを対話的に選択することで，適切な XPath を生成できることを確認した．

今後は，コーパスを用いて C 言語特有の XML 構造を得ることで冗長な XPath を除外し，さらなる高速化を図ることや，srcML 以外の XML への変換手法の検討，及び兄弟ノードや子孫ノードに関する情報を利用した XPath の生成によって XPath の表現力を向上させることを目指している．

謝辞： 本研究の一部は科研費（#15H02683, #18H03221, #15K15973，#18K11241）の助成を受けた．

文　　献

[1] 大須賀, 小林, 渥美, 間瀬, 山本, 鈴村, 阿草. CX-Checker: 柔軟にカスタマイズ可能な C 言語プログラムのコーディングチェッカ. 情報処理学会論文誌, Vol. 53, No. 2, pp. 590–600, 2012.

[2] J. Cordy, T. Dean, A. Malton, and K. Schneider. Software engineering by source transformation - experience with TXL. In *Proc. SCAM'01*, pp. 168–178.

[3] P. Klint, T. Storm, and J. Vinju. RASCAL: A domain specific language for source code analysis and manipulation. In *Proc. SCAM 2009*, pp. 168–177.

[4] B. Bartman, C. Newman, M. Collard, and J. Maletic. srcQL: A syntax-aware query language for source code. In *Proc. SANER 2017*, pp. 467–471.

[5] M. Collard, H. Kagdi, and J. Maletic. An XML-based lightweight C++ fact extractor. In *Proc. IWPC'03*, pp. 134–143.

[6] 渥美, 小林, 山本, 阿草. ソフトウェア開発支援基盤のためのソースプログラムの XML 表現. 電子情報通信学会論文誌, Vol. D96, No. 11, pp. 2681–2691, 2013.

[7] D. Binkley, N. Gold, S. Islam, J. Krinke, and S. Yoo. Tree-oriented vs. line-oriented observation-based slicing. In *Proc. SCAM 2017*, pp. 21–30.

[8] 福安, 山本, 阿草. 細粒度ソフトウェア・リポジトリに基づいた CASE ツール・プラットフォーム Sapid. 情報処理学会論文誌, Vol. 39, No. 6, pp. 1990–1998, 1998.

[9] 岩間, 福原, 猪股ほか. C ソースコード静的解析のための問い合わせ言語 CXmlPyQuery とその応用. 情処研報 2017-SE-196-21, 情報処理学会, 2017. (7 pages).

[10] H. A. Sleiman and R. Corchuelo. A survey on region extractors from web documents. *IEEE TKDE*, Vol. 25, No. 9, pp. 1960–1981, 2013.

[11] 石井, 森嶋ほか. 抽象的な記述が可能な Web からのデータ抽出言語の提案. In *DEIM Forum 2011*, No. D2-6 (8 pages).

[12] Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, Vol. 118, No. 1, pp. 15 – 68, 2000.

[13] C. Chang and S. Lui. IEPAD: Information extraction based on pattern discovery. In *Proc. WWW'01*, pp. 681–688.

[14] M. Zaki and C. Aggarwal. XRules: An effective structural classifier for XML data. In *Proc. KDD'03*, KDD '03, pp. 316–325.

[15] J. Zhang, et al. Automatic extraction rules generation based on XPath pattern learning. In *WISE 2010 Workshops, LNCS 6724*, pp. 58–69, 2011.