4B1-R-2-5

文法推論のコンパイラ言語への適用

Applying Grammatical Inference to a Compiled Language

常見 一樹　上野 敦志　田窪 朋仁　辰巳 昭治
Kazuki Tsunemi　Atsushi Ueno　Tomohito Takubo　Shoji Tatsumi

大阪市立大学 大学院工学研究科
Osaka City University, Graduate School of Information Engineering

We aim to develop a method for mining grammatical rules of a programming language from a bunch of source codes and the corresponding byte codes. This method will lead to bridging the gap between the fields of grammatical inference and semantic understanding both of which are critical to language learning. We show the potential of our method by a thought experiment.

1. はじめに

プログラミング言語は，コンピュータに対する命令の記述をより分かりやすく行うために開発された人工言語の総称である．特に近年用いられているプログラミング言語は高級言語と呼ばれ，より人間の直感的に分かりやすい構文で記述できるように設計されている．ここで注目すべきは，人間にも事前知識さえあれば解読可能なプログラミング言語が，コンパイラやインタプリンタ，ミドルウェア，OS，BIOS などを経て最終的に CPU で処理することが出来る時系列シグナル情報に機械的に変換されるという点である．これはプログラミング言語という文法的に記述された構造情報が意味情報を保存したまま，手続き処理を纏めた構造へと変換されている事を意味する．しかもその変換は構文に関する情報さえ与えてやれば機械的に実行可能である．

構文に関する情報を用いてある構造を持つ表現から別の構造を持つ構造への変換が存在するとき（図1），相異なる2つの構造で表現される同じ情報から変換に必要な情報を抜き出すことが可能であると考えられる．例えば，プログラミング言語のソースコードと，コンパイル後のオブジェクトコードからコンパイラ推定が可能である．

本論文ではソースコードとバイトコードを記号列のベクトルとして抽象化し，ソースコード同士及びバイトコード同士の差分をベクトル表現することで，推定に有用な情報を行列として抽出する手法を提案する．また構文の重要な機能の一つである構造の階層化に関する情報の抽出も試みる．

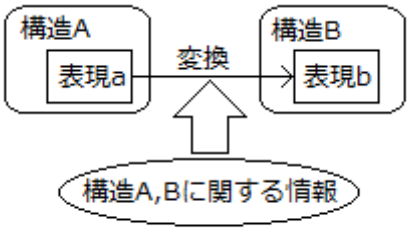

図1

連絡先：常見一樹，大阪市立大学大学院工学研究科，
〒558-8585　大阪市住吉区杉本 3-3-138
TEL：06-6605-2688
Mail：ksilver@kdel.info.eng.osaka-cu.ac.jp

2. 関連研究

2.1 文法推論

法則性のある記号列を元にして，その記号列に共通して存在する表現拘束情報を抽出する技法は文法推論(Grammatical Inference)と呼ばれ盛んに研究されている．しかしこれは意味情報には殆ど言及せず，入力記号列が文法的に正か誤かという判断を行うのが主流である [Clark 2010] .

2.2 文脈自由文法

文脈自由文法は二型文法ともいい，その強力な表現能力と扱いやすさからコンパイル言語の構文定義にも用いられている．自然言語文法の研究にも文脈自由文法を利用して文法の自動学習も試みられている[中村 2007]．また文脈自由文法中に出現する内部記号と概念に関する研究も行われている[花川 1998].

2.3 自然言語処理

機械翻訳や文章からの知識情報の自動獲得は自然言語処理の分野で非常に盛んに研究されている．また応用も行われており，実際にサービスとして提供されている．しかしその品質は未だに非常に悪く，構文情報などを盛り込むことでの性能改善が期待されている[Nichols 2010]．既知の Wikipedia 独自のページ構造情報を利用する Wikipedia マイニングが一例である．[玉川 2009]

3. 提案手法

Java ソースコードとそのソースコードをコンパイルした結果のバイトコードの対を1つの事例とし，この事例の集合を用いて構文に関する情報を抽出することを試みる．議論を簡単にするためにソースコードの字句解析が済んでいるものとし，バイトコードも1バイト単位でブロック化して扱う．

事例の有限集合を$\boldsymbol{E}$とする．i番目の事例$\boldsymbol{e}^i$は字句配列ベクトル$\boldsymbol{s}^i$とバイト配列ベクトル$\boldsymbol{b}^i$を用いて以下のように表現される．ここで T は転置を表す．$|\boldsymbol{E}|$は集合$\boldsymbol{E}$の要素数を表す．

$$\boldsymbol{e}^i = \left(\boldsymbol{s}^i, \boldsymbol{b}^i\right)^{\mathrm{T}}, \quad \boldsymbol{e}^i \in \boldsymbol{E}\ (\mathrm{i} = 1,2,\ldots,|\boldsymbol{E}|) \qquad (1)$$

また字句配列ベクトル$\boldsymbol{s}^i$とバイト配列ベクトル$\boldsymbol{b}^i$はそれぞれ，ソースコードを成す字句の集合 $\boldsymbol{C}$ とバイトデータの集合 $\boldsymbol{N}$ を用いて以下のように表される．

$$\boldsymbol{s}^i = \left(s_1^i, s_2^i, \dots, s_{l_i}^i\right)^{\mathrm{T}}, \quad s_j^i \in \boldsymbol{C}\ (j = 1,2,\dots,l) \tag{2}$$

$$\boldsymbol{b}^i = \left(b_1^i, b_2^i, \dots, b_{m_i}^i\right)^{\mathrm{T}}, \quad b_j^i \in \boldsymbol{N}\ (j = 1,2,\dots,m) \tag{3}$$

ここでl_iとm_iはそれぞれ$\boldsymbol{s}^i$と$\boldsymbol{b}^i$の次元の大きさを表す。

字句配列ベクトルの集合を$\boldsymbol{S} = \{\boldsymbol{s}^i | \boldsymbol{e}^i \in \boldsymbol{E}\}$, バイト配列ベクトルの集合を$\boldsymbol{B} = \{\boldsymbol{b}^i | \boldsymbol{e}^i \in \boldsymbol{E}\}$とする. コンパイラ推定問題は, コンパイラによる変換をfとすると, $f(\boldsymbol{s}^i) = \boldsymbol{b}^i (i = 1,2,\dots|\boldsymbol{E}|)$を満たす変換$f: \boldsymbol{S} \to \boldsymbol{B}$を求める問題と言うことが出来る.

3.1 構造の作成

(1) 次元一致ケース

相異なる2つの事例$\boldsymbol{e}^p, \boldsymbol{e}^q$について, 字句配列ベクトル$\boldsymbol{s}^p, \boldsymbol{s}^q$の次元とバイト配列ベクトル $\boldsymbol{b}^p, \boldsymbol{b}^q$の次元について以下の式が成り立つ場合を考える.

$$l_p = l_q, \quad m_p = m_q$$

相異なる2つの事例$\boldsymbol{e}^p, \boldsymbol{e}^q$に対して事例差$\boldsymbol{\Delta e}^{p,q}$, 及び字句差異ベクトル$\boldsymbol{\Delta s}^{p,q}$とバイト差異ベクトル$\boldsymbol{\Delta b}^{p,q}$を以下のように定義する.

$$\boldsymbol{\Delta e}^{p,q} = (\boldsymbol{\Delta s}^{p,q}, \boldsymbol{\Delta b}^{p,q})^{\mathrm{T}} \tag{4}$$

$$\boldsymbol{\Delta s}^{p,q} = \left(\Delta s_1^{p,q}, \Delta s_2^{p,q}, \dots, \Delta s_{l_p}^{p,q}\right)^{\mathrm{T}}$$

$$\Delta s_i^{p,q} = \begin{cases} 1\ (s_i^p \neq s_i^q) \\ 0\ (s_i^p = s_i^q) \end{cases}, \quad (i = 1,2,\dots,l_p) \tag{5}$$

$$\boldsymbol{\Delta b}^{p,q} = \left(\Delta b_1^{p,q}, \Delta b_2^{p,q}, \dots, \Delta b_{m_p}^{p,q}\right)^{\mathrm{T}}$$

$$\Delta b_i^{p,q} = \begin{cases} 1\ (s_i^p \neq s_i^q) \\ 0\ (s_i^p = s_i^q) \end{cases}, \quad (i = 1,2,\dots,m_p) \tag{6}$$

ここで$\boldsymbol{\Delta s}^{p,q}$と$\boldsymbol{\Delta b}^{p,q}$はそれぞれソースコード情報とバイトデータ情報の差分を表す。

$\boldsymbol{\Delta s}^{p,q}$が零ベクトルの時, $\boldsymbol{s}^p = \boldsymbol{s}^q$である. コンパイラは決定性を持つため$\boldsymbol{b}^p = \boldsymbol{b}^q$である. このとき$\boldsymbol{e}^p = \boldsymbol{e}^q$となり$\boldsymbol{e}^p, \boldsymbol{e}^q$が相異なる事例であることに反する. よって$\boldsymbol{\Delta s}^{p,q}$は零ベクトルになり得ない. $\boldsymbol{\Delta b}^{p,q}$が零ベクトルの時, コンパイルによる変換$f$について, $\boldsymbol{b}^p = \boldsymbol{b}^q$より$f(\boldsymbol{s}^p) = f(\boldsymbol{s}^q)$が言える.

$\boldsymbol{\Delta b}^{p,q}$が零ベクトルでない場合, $\boldsymbol{\Delta e}^{p,q}$はソースコードの違いによるバイトデータ情報の変化が記録されていることになる。ここで長さが 1 になるように$\boldsymbol{\Delta s}^{p,q}, \boldsymbol{\Delta b}^{p,q}$をそれぞれ正規化した正規字句差異ベクトル$\boldsymbol{\delta s}^{p,q}$と, 正規バイト差異ベクトル$\boldsymbol{\delta b}^{p,q}$を定義する.

$$\boldsymbol{\delta s}^{p,q} = \frac{\boldsymbol{\Delta s}^{p,q}}{\|\boldsymbol{\Delta s}^{p,q}\|}, \quad \boldsymbol{\delta s}^{p,q} = \left(\delta s_1^{p,q}, \delta s_2^{p,q}, \dots, \delta s_{l_p}^{p,q}\right)^{\mathrm{T}} \tag{7}$$

$$\boldsymbol{\delta b}^{p,q} = \frac{\boldsymbol{\Delta b}^{p,q}}{\|\boldsymbol{\Delta b}^{p,q}\|}, \quad \boldsymbol{\delta b}^{p,q} = \left(\delta b_1^{p,q}, \delta b_2^{p,q}, \dots, \delta b_{m_p}^{p,q}\right)^{\mathrm{T}} \tag{8}$$

$\boldsymbol{\delta s}^{p,q}, \boldsymbol{\delta b}^{p,q}$はそれぞれ差異が複数箇所有る場合に, 差異の箇所が増えるに従い各成分の値が小さくなる性質を持つ, 全体に対する差分量を表す量である. ソースコードの差異情報とバイトデータの差異情報の対応関係を表す構造対応行列$\boldsymbol{M}^{p,q}$を以下のように定義する.

$$\boldsymbol{M}^{p,q} = \boldsymbol{\delta s}^{p,q} (\boldsymbol{\delta b}^{p,q})^{\mathrm{T}} \tag{9}$$

ここで$\boldsymbol{s}^p$と$\boldsymbol{s}^q$, $\boldsymbol{b}^p$と$\boldsymbol{b}^q$が共に似ている場合, $\boldsymbol{\delta s}^{p,q}, \boldsymbol{\delta b}^{p,q}$は多くの成分が 0 で少数成分が 1 に近いベクトルとなる. このとき行列$\boldsymbol{M}^{p,q}$も定義から同様に少数成分が 1 に近い構造となる. このとき$\boldsymbol{M}^{p,q}$の(i,j)成分が非零であるとき, $\boldsymbol{\delta s}^{p,q}$の第$i$成分と$\boldsymbol{\delta b}^{p,q}$の第$j$成分が共に非零である. これは事例$\boldsymbol{e}^p, \boldsymbol{e}^q$において, $\boldsymbol{s}^p$の第i番目と$\boldsymbol{b}^p$の第j成分, $\boldsymbol{s}^q$の第i番目と$\boldsymbol{b}^q$の第j成分が対応していることを示す. また(i,j)成分の大きさが対応関係の強さを表している. この値を対応度と呼ぶことにする. 対応度は$\boldsymbol{s}^p$と$\boldsymbol{s}^q$, $\boldsymbol{b}^p$と$\boldsymbol{b}^q$が似ていない場合でも求めることが出来るが値は小さくなる. つまり$\boldsymbol{M}^{p,q}$は$\boldsymbol{e}^p, \boldsymbol{e}^q$から求まる対応関係の位置と関連度を表す行列である.

多くの事例ペアから同次元の構造対応行列が得られ, なおかつ特定の成分(i,j)が共通して 1 に近い時, ソースコードの第i番目の字句とバイトコードの第i番目のバイトが対応している可能性が高い. この字句とバイトの対応はコンパイラを構成する上で非常に重要である.

(2) 次元不一致ケース

相異なる2つの事例$\boldsymbol{e}^p, \boldsymbol{e}^q$について, 字句配列ベクトル$\boldsymbol{s}^p, \boldsymbol{s}^q$の次元が異なる. もしくはバイト配列ベクトル $\boldsymbol{b}^p, \boldsymbol{b}^q$の次元が異なる場合を考える.

字句配列ベクトル$\boldsymbol{s}^p, \boldsymbol{s}^q$について, $\dim(\boldsymbol{s}^p) > dim(\boldsymbol{s}^q)$であるとする. ただし$\dim(\boldsymbol{s})$はベクトルの次元を表す記号である.

$$\boldsymbol{s}^p = \left(s_1^p, s_2^p, \dots, s_{l_p}^p\right)^{\mathrm{T}}, \quad s_i^p \in \boldsymbol{C}, \quad i = 1,2,\dots,l_p \tag{10}$$

$$\boldsymbol{s}^q = \left(s_1^q, s_2^q, \dots, s_{l_q}^q\right)^{\mathrm{T}}, \quad s_i^p \in \boldsymbol{C}, \quad i = 1,2,\dots,l_q \tag{11}$$

$$l_p > l_q \tag{12}$$

このとき字句配列圧縮記号Xを用いて第k成分から$\dim(\boldsymbol{s}^p) - \dim(\boldsymbol{s}^q) + 1$個を置き換えた抽象字句配列ベクトル$\boldsymbol{s}^{p\prime}$を作る.

$$\boldsymbol{s}^{p\prime} = \left(s_1^p, s_2^p \dots, s_{k-1}^p, X, s_{k+n-m+1}^p, \dots, s_{l_q}^p\right)^{\mathrm{T}} \tag{13}$$

$$X = \left(s_k^p, s_{k+1}^p, \dots, s_{k+n-m}^p\right)^{\mathrm{T}} \tag{14}$$

$$\dim(\boldsymbol{s}^{p\prime}) = \dim(\boldsymbol{s}^q) \tag{15}$$

字句圧縮記号Xは一意に定義できないため, $\boldsymbol{s}^{p\prime}, \boldsymbol{s}^q$から得られる字句差異ベクトル$\boldsymbol{\Delta s}^{p,q}$の長さが最小になるように選ぶ. バイト配列ベクトル$\boldsymbol{b}^{p\prime}, \boldsymbol{b}^q$に対しても同様に, バイト配列圧縮記号を用いて抽象バイト配列ベクトルを作る. すると$\dim(\boldsymbol{s}^{p\prime}) = \dim(\boldsymbol{s}^q)$ かつ $\dim(\boldsymbol{b}^{p\prime}) = \dim(\boldsymbol{b}^q)$ であるため $\boldsymbol{e}^{p\prime} = (\boldsymbol{s}^{p\prime}, \boldsymbol{b}^{p\prime}), \boldsymbol{e}^q = (\boldsymbol{s}^q, \boldsymbol{b}^q)$について, 次元一致ケースの手続きを行うことが出来る.

3.2 統合と一般化

前節の手続きだけでは抽出した情報は事例のペアに依存しているため, 事例のペア毎に構文対応行列や字句圧縮記号といった構造情報が生み出されることになる. 事例の数に対して組み合わせの爆発が生じ, 分析した情報の急激な増加することが危惧される.

そこで多数生み出される構造情報のうち比較的似ている構造情報や字句圧縮記号, バイト圧縮記号を等価と見なす. 複数の構造情報から共通する性質を抽出し, 構造情報を一般化するなどの操作を加える. こうすることで分析情報の圧縮と表現の一般化を同時に行う事が出来る.

構造対応行列が直交行列となっており, 記号の対応度が1で有れば、等価変換と行列演算だけでコードの書き換えが可能である. 例えば未知のソースコード$\boldsymbol{s} = (x,*,y)^{\mathrm{T}}$を構造対応行列$\boldsymbol{M}$で書き換えることを考える.

$$\boldsymbol{Ms} = \begin{pmatrix}1 & 0 & 0\\ 0 & 0 & 1\\ 0 & 1 & 0\end{pmatrix}\begin{pmatrix}x\\ *\\ y\end{pmatrix} = \begin{pmatrix}x\\ y\\ *\end{pmatrix} \quad (16)$$

これはポーランド記法の変換である. 実際の Java バイトコードにおいても変数名格納場所などは別であるものの, load_1 命令→load_2 命令→imul 命令など似たような構造をとる。多くの事例ペアにおける構造対応行列に対し平均を取るなどの操作を行うことで, ソースコード中の位置とバイトコードの位置の対応関係が予測できる. つまり構文構造の対応関係の発見に繋がる.

また圧縮記号の導入も単に次元の調整ではなく, 一般化プロセスでは, 文法の入れ子構造を検出するのに役立つと考えられる. 例えば字句圧縮記号は字句配列ベクトルと同じ構造を取る. 他の字句記号配列ベクトルと字句圧縮記号の間に等価関係や構造情報が抽出できた際に字句記号配列ベクトルは子構造とみなす事が出来る. 以下の例は if 文的構造から入れ子にされた式の構造を抽出するプロセスである. ソースコード$\boldsymbol{s^p}, \boldsymbol{s^q}$を以下のように定義する.

$$\boldsymbol{s}^p = (if, a, \neq, b, then)^{\mathrm{T}}, \quad \boldsymbol{s}^q = (if, bool, then)^{\mathrm{T}} \quad (17)$$

ここで字句圧縮記号$\boldsymbol{A} = (a, \neq, \mathrm{b})^{\mathrm{T}}$を置くと,

$$\boldsymbol{s}^{p\prime} = (if, \boldsymbol{A}, then)^{\mathrm{T}}, \quad \boldsymbol{s}^q = (if, bool, then)^{\mathrm{T}} \quad (18)$$

ここから$\boldsymbol{A}$と$bool$, 即ち$a \neq b$と$bool$が同じ種類のクラスに分類されることが推察できる.

4. 実験

Java を含め近年開発されたプログラミング言語は, 人間が読みやすく, なおかつ汎用的である. その代償としてコンパイラの仕様は複雑になっており, これは本研究で扱うコンパイル時の構文情報もまた煩雑であることを意味する. 推定対象の構文が複雑になればなるほど, 実験するのに必要な計算量とデータセットが爆発的に増大するのは明白である. 同時に, プログラミング言語全体という広域の推定結果を実験的に求めることが出来たとしても, 内部でどのような処理が行われたか分析は困難である.

そこで本論文では限定された部位の構文情報の推定を行い, 提案手法の有用性及び, その挙動を分析する. 具体的な初期段階としては, Java コードのうちクラス定義を固定化し, 特定メソッド内部の手続きのみを変更して構文情報の推定を行う.

現在は準備段階として, 式の中間演算子表現とポーランド記法間の変換規則の推定を行っている. これはメソッドの手続き記述の中でも最も初歩的な機能の一つである式の計算において, Java のバイトデータはポーランド記法に非常に類似した構造を出力するためである. 中間演算子表現からポーランド記法への変換に関する情報の抽出例を以下に述べる.

中間演算子表現からポーランド記法表現の変換について考える. 以下に挙げる事例に対して提案手法を用いた場合に得られる構造情報について具体的に述べる. 中間演算子表現をソースコードとして扱い, ポーランド記法をバイトコードとして扱う.

4.1 次元一致ケース

(事例 1)ソースコード:$x + y$, バイトコード:(X, Y, Add)
(事例 2)ソースコード:$x - y$, バイトコード:$(X, Y, Minus)$
(事例 3)ソースコード:$z - y$, バイトコード:$(Z, Y, Minus)$
記号$\boldsymbol{e^1}, \boldsymbol{e^2}, \boldsymbol{e^3}$を用いて表現すると以下のようになる.

$$\boldsymbol{e}^1 = (\boldsymbol{s}^1, \boldsymbol{b}^1)^{\mathrm{T}},\ \boldsymbol{s}^1 = (x, +, y)^{\mathrm{T}},\ \boldsymbol{b}^1 = (X, Y, Add)^{\mathrm{T}} \quad (19)$$

$$\boldsymbol{e}^2 = (\boldsymbol{s}^2, \boldsymbol{b}^2)^{\mathrm{T}},\ \boldsymbol{s}^2 = (x, -, y)^{\mathrm{T}},\ \boldsymbol{b}^2 = (X, Y, Minus)^{\mathrm{T}} \quad (20)$$

$$\boldsymbol{e}^3 = (\boldsymbol{s}^3, \boldsymbol{b}^3)^{\mathrm{T}},\ \boldsymbol{s}^3 = (z, -, y)^{\mathrm{T}},\ \boldsymbol{b}^3 = (Z, Y, Minus)^{\mathrm{T}} \quad (21)$$

$\boldsymbol{e}^1, \boldsymbol{e}^2$を用いて差分をとる.

$$\boldsymbol{\Delta s}^{1,2} = (0,1,0)^{\mathrm{T}}, \quad \boldsymbol{\Delta b}^{1,2} = (0,0,1)^{\mathrm{T}} \quad (22)$$

正規化する.

$$\boldsymbol{\delta s}^{1,2} = (0,1,0)^{\mathrm{T}}, \quad \boldsymbol{\delta b}^{1,2} = (0,0,1)^{\mathrm{T}} \quad (23)$$

構造対応行列を求める

$$\boldsymbol{M}^{1,2} = \boldsymbol{\delta s}^{1,2}(\boldsymbol{\delta b}^{1,2})^{\mathrm{T}} = \begin{pmatrix}0 & 0 & 0\\ 0 & 0 & 1\\ 0 & 0 & 0\end{pmatrix} \quad (24)$$

(2,3)成分が 1 だから$\boldsymbol{s}^1, \boldsymbol{s}^2$第 2 成分と$\boldsymbol{b}^1, \boldsymbol{b}^2$第 3 成分を比較して, 字句"+"とコード"$Add$", 字句"−"とコード"$Minus$"がそれぞれ等しいことが分かる. 同様に事例$\boldsymbol{e}^2, \boldsymbol{e}^3$については以下の構造対応行列が得られる.

$$\boldsymbol{M}^{2,3} = \begin{pmatrix}1 & 0 & 0\\ 0 & 0 & 0\\ 0 & 0 & 0\end{pmatrix} \quad (25)$$

(1,1)成分が 1 だから$\boldsymbol{s}^2, \boldsymbol{s}^3$第 1 成分と$\boldsymbol{b}^2, \boldsymbol{b}^3$第 1 成分を比較して, 字句"$x$"とコード"$X$", 字句"$y$"とコード"$Y$"がそれぞれ等しいことが分かる. 同様に事例$\boldsymbol{e}^3, \boldsymbol{e}^1$については正規字句配列ベクトルと正規バイト配列ベクトルが以下のようになる.

$$\boldsymbol{\delta s}^{3,1} = \left(\tfrac{1}{\sqrt{2}}, \tfrac{1}{\sqrt{2}}, 0\right)^{\mathrm{T}}, \quad \boldsymbol{\delta b}^{3,1} = \left(\tfrac{1}{\sqrt{2}}, 0, \tfrac{1}{\sqrt{2}}\right)^{\mathrm{T}} \quad (26)$$

このため構造変換行列は以下の通りになる.

$$\boldsymbol{M}^{3,1} = \begin{pmatrix}1/2 & 0 & 1/2\\ 1/2 & 0 & 1/2\\ 0 & 0 & 0\end{pmatrix} \quad (27)$$

この構造変換行列には 1 の成分がないため対応関係は決定できない. 例えば(1,1)成分に注目すると対応度が 0.5 で, 対応する成分が"x"とコード"X", 字句"y"とコード"Y"の対応を示している. これは正しい対応である. 一方で(1,3)成分に注目すると同じく対応度が 0.5 で, 対応する成分が"x"とコード"Add", 字句"z"とコード"$Minus$"の対応を示している. これは誤った対応である.

このように対応度が 1 の場合は正しい対応が得られる一方で対応度が 1 より小さい場合は必ずしも正しい対応のみを表すとは限らない.

4.2 次元不一致ケース

(事例 1)ソースコード:$x + y$, バイトコード:(X, Y, Add)
(事例 2)ソースコード:$x + y * z$,
バイトコード:(X, Y, Z, Mul, Add)
記号$\boldsymbol{e}^1, \boldsymbol{e}^2$を用いて表現すると以下のようになる.

$$\boldsymbol{e}^1 = (\boldsymbol{s}^1, \boldsymbol{b}^1)^{\mathrm{T}},\ \boldsymbol{s}^1 = (x, +, y)^{\mathrm{T}},\ \boldsymbol{b}^1 = (X, Y, Add)^{\mathrm{T}} \quad (28)$$

$$\boldsymbol{e}^2 = (\boldsymbol{s}^2, \boldsymbol{b}^2)^{\mathrm{T}}$$

$$\boldsymbol{s}^2 = (x, +, y, *, z)^{\mathrm{T}},\ \boldsymbol{b}^2 = (X, Y, Z, Mul, Add)^{\mathrm{T}} \quad (29)$$

$\|\boldsymbol{\Delta s}^{1,2\prime}\|, \|\boldsymbol{\Delta b}^{1,2\prime}\|$がそれぞれ最小になるように字句圧縮記号$a$, バイト圧縮記号$A$, 及びを用いて事例$\boldsymbol{e}^2$を書き換えたに事例$\boldsymbol{e}^{2\prime}$を作る.

$$\boldsymbol{e}^{2\prime} = (\boldsymbol{s}^{2\prime}, \boldsymbol{b}^{2\prime})^{\mathrm{T}},\ \boldsymbol{s}^{2\prime} = (x, +, a)^{\mathrm{T}},\ \boldsymbol{b}^{2\prime} = (X, A, Add)^{\mathrm{T}} \quad (30)$$

$$a = (y, *, z)^{\mathrm{T}}, \quad A = (Y, Z, Mul)^{\mathrm{T}} \quad (31)$$

次元一致ケースと同様に計算する.

$$\boldsymbol{\delta s}^{1,2\prime} = (0,0,1)^{\mathrm{T}}, \quad \boldsymbol{\delta b}^{1,2\prime} = (0,1,0)^{\mathrm{T}} \quad (32)$$

$$\boldsymbol{M}^{1,2\prime} = \begin{pmatrix}0 & 0 & 0\\ 0 & 0 & 0\\ 0 & 1 & 0\end{pmatrix} \quad (33)$$

(3,2)成分が 1 だから$\boldsymbol{s}^1, \boldsymbol{s}^{2\prime}$第 3 成分と$\boldsymbol{b}^1, \boldsymbol{b}^{2\prime}$第 2 成分を比較して，字句"$y$"とコード"$Y$"，字句"$y,*,z$"とコード"$Y,Z,Mul$"がそれぞれ等しいことが分かる．字句"$y,*,z$"とコード"$Y,Z,Mul$"のペアは数式で言うと式の子構造である項と見なすことが出来る．

5. おわりに

今回提案した手法ではソースコードとバイトコードをベクトル化することで差分情報をベクトルとして求め，コンパイル前後の位置の対応関係を行列で表現した．その行列を用いて特別な事例ペアでは行列から一対一の対応が得られること，任意のペアから対応関係を定量的に導出できることを示した．

また比較するソースコードのベクトルとしての次元が一致しない，バイトコードのベクトルとしての次元が一致しない場合は圧縮記号を用いて任意の文字列を一纏めにして長さを縮める手法を導入し，スケールの変化に対して柔軟に対応できる可能性も示した．これは構文の階層構造を探る上で重要である．

抽出した構造情報を組み合わせるなどして適用範囲を広げるなどの一般化に関する操作をどのように行うかが今後の課題である．また実際に提案手法を実装し，どのような規模でどのような変換データが得られるのかの実験的確認が必要である．

参考文献

[Clark 2010] Alexander Clark, Christophe Costa Florêncio, Chris Watkins: “Languages as hyperplanes: grammatical inference with string kernels”, Mach Learn, 2011.

[中村 2007] 中村 克彦, 杉田 雄大: “文脈自由文法の漸次学習方式とその応用”, 東京電機大学総合研究所年報, 2007.

[花川 1998] 花川 賢治, 武田 英明, 西田 豊明: “文章から概念を獲得するための文法推論”, 電子情報通信学会, 1998.

[Nichols 2010] Nichols: “Applying Deep Grammars to Machine Translation, Paraphrasing, and Ontology Construction”, NAIST-IS-DD0561041, NAIST, 2010.

[玉川 2009] 玉川 奨, 桜井 慎弥, 手島 拓也: “日本語 Wikipedia からの大規模オントロジー学習”, 人工知能学会誌 25 巻 5 号 SP-C, 2010.