Sub-chaining style と flat-chaining style 双方の形式に対応した fluent interface 生成に向けた形式的な取り組み

山崎 徹郎 [1], 千葉 滋 [2]

[1] 東京大学大学院 情報理工学系研究科
yamazaki@csg.ci.i.u-tokyo.ac.jp

[2] 東京大学大学院 情報理工学系研究科
chiba@acm.org

概 要 構文解析を型検査器に埋め込むことで fluent interface の呼び出し規約をコンパイル時に検査する手法が研究されている．本稿では sub-chaining style と flat-chaining style の両形式に対応した検査付き fluent interface を生成する手法を提案する．既存の sub-chaining style と flat-chaining style の両形式に対応した生成手法には手順が操作的であるため理論的な取り扱いが困難であるという課題があった．本手法では単純な操作の組み合わせによって両形式に対応した検査付き fluent interface の生成を実現する．その結果，入力として受け入れることができる文法の制限を明らかにすることができた．

1 はじめに

メソッド呼び出しの連鎖を通して呼び出すように設計されたライブラリインターフェースを fluent interface という．今日では Java の Stream API[1]，jMock[2]，jOOQ[3] など fluent interface を提供するライブラリが散見される．

型検査を利用して fluent interface の呼び出し規約を検査する手法が研究されている．[11, 9, 8, 10, 6, 7, 12] これらの手法には処理系の改造や前処理の追加が必要ないため適切にライブラリを定義しさえすれば通常のライブラリと同様に利用するだけで検査の恩恵を得られるという利点がある反面，検査できるインターフェースに制限がある，ライブラリコードが巨大化する，検査のパフォーマンスチューニングが困難などの課題も残っており，研究の余地が多分に残されている．ここで型検査器を利用して fluent interface の呼び出し規約を検査するというものがどんな手法なのかについては第 2 章で詳しく述べる．

本稿で我々は，flat-chaining style と sub-chaining style の双方の呼び出し方で利用できるインターフェースを生成するように，型検査を利用した呼び出し規約の検査付き fluent interface を自動生成する生成器を拡張する．呼び出し規約の検査付き fluent interface は通常，インターフェース定義するコードが巨大になるため自動生成器によって生成する．しかしこのような生成器は生成器を設計した時点で想定されていたインターフェースしか生成できず，sub-chaining style の呼び出しを受け入れるなどのインターフェースは生成できない．

既存の検査付き fluent interface 生成器では sub-chaining style を受け入れるインターフェースの生成が難しいという課題を指摘し，sub-chaining style の呼び出しも受け入れることができる検査付き fluent interface 生成器を提案した既存の研究として中丸らの研究 [10] が挙げられる．彼らの提案では入力として与えられた文法からどのように fluent interface を生成するかが step-by-step に示されており，その生成過程を理論的に解析することは難しい．理論的解析が難しいという課題は

Listing 1. Fluent interface

```
class SQL {
    String selectList;
    /* ... 中略 ... */
    SQL select(String selectList) {
        this.selectList = selectList;
        return this;
    }
    SQL from(String tableSource) {
        this.tableSource = tableSource;
        return this;
    }
    SQL where(String searchCondition) {
        this.searchCondition = searchCondition;
        return this;
    }
    Ret run() {
        /* ... 略 ... */
    }
}
```

Listing 2. メソッド呼び出しの連鎖の例 (1/2)

```
Ret result = receiver.method1().method2();
```

入力文法が満たすべき条件の解明が難しいことや，生成器の拡張が不変条件を守っているかの検証が難しいなどの問題を引き起こす．

我々は文法書き換えに基づく flat-chaining style と sub-chaining style の両形式で呼び出すことができる fluent interface を生成する生成器を実装する方法を提案する．我々の手法は大きく (1) 文法書き換えによって sub-chaining style 形式の呼び出しを文法に埋め込み，(2) 既存の検査付き fluent interface 生成手法によってインターフェースを生成する という 2 つのステップから成る．それぞれのステップで行う操作は簡単なものであるため理論的解析も比較的難しくない．また理論的解析の一例として，我々の設計を採用した検査付き fluent interface 生成器が入力文法に要求する条件を明らかにした．我々の提案は fluent interface 生成器の設計とその性質の証明のみであり，ツールの実装や実証実験は行っていない．

以下，第 2 章では検査付き fluent interface とその生成手法，および flat-chaining style と sub-chaining style の両形式について詳細を述べ，第 3 章では我々が提案する検査付き fluent interface 生成手法を述べる．第 4 章では提案手法を実装した生成器が入力に求める条件を明らかにし，第 5 章ではまとめを述べる．

2 検査付き fluent interface と sub-chaining style

メソッド呼び出しの連鎖の形で呼び出されるように設計されたライブラリインターフェースを fluent interface [5] という．クラスやメソッド呼び出しなどの機能を提供する多くのオブジェクト指向言語ではソースコード 2 のようにメソッド呼び出しを並べて記述すると 1 つ目のメソッド呼び出しの返り値をレシーバーに 2 つ目のメソッドを呼び出すという意味になる．意味論の上ではソースコード 2 のように記述してもソースコード 3 のように記述しても意味は同じである．一般的にはソースコード 2 を指してメソッド呼び出しの連鎖 (method chaining) というが，本稿の関心はイン

Listing 3. メソッド呼び出しの連鎖の例 (2/2)

```
Mid receiver2 = receiver.method1();
Ret result    = receiver2.method2();
```

Listing 4. 検査付き fluent interface

```
class SQL {
    SQL2 select(String selectList) {
        /* ... 略 ... */
    }
}
class SQL2 {
    SQL3 from(String tableSource) {
        /* ... 略 ... */
    }
}
class SQL3 {
    SQL4 where(String searchCondition) {
        /* ... 略 ... */
    }
}
class SQL4 {
    Ret run() {
        /* ... 略 ... */
    }
}
```

ターフェースの使い方ではなく生成方法にあるため，どちらも区別せずにメソッド呼び出しの連鎖と呼ぶ．

ソースコード 1 に fluent interface の例を示す．`select`，`from`，`where` の各メソッドが `this` を返却するため，連鎖される次のメソッドは同じ `SQL` オブジェクトをレシーバに呼び出される．`run` はメソッド呼び出しの連鎖として記述された select クエリを実行するメソッドであり，`SQL` ではなく select クエリの実行結果を表す `Ret` 型の値を返却する．

Fluent interface の呼び出し規約を構文検査の技術を応用して検査する方法が研究されている．[11, 9, 8, 10, 6, 7, 12] 例えば，ソースコード 1 のように定義されたインターフェースでは，`new SQL().where("he is come").from("?");` のような間違った呼び出しも記述できてしまう．構文検査を型検査に埋め込むことでこのような間違ったインターフェース呼び出しをコンパイル時に検出する方法が研究されている．ある fluent interface への正しい呼び出しの集合を考えたとき，これはメソッド呼び出しの列の集合であるためある種の言語であるとみなすことができる．そのためあるメソッド呼び出しの列が与えられた時，それが正しい呼び出しの集合に含まれるかどうかは構文解析の技術によって判定することができる．さらにこのような構文解析をメタプログラミングの技術を応用して型検査に埋め込むことで，処理系の改造や前処理の追加を行わないまま，自動的に呼び出し規約が検査されるような fluent interface を実現することができる．

各メソッドがちょうど次に呼べるメソッドだけを持つクラスを返すようにすることで決定性有限状態オートマトン (Deterministic Finite Automaton, DFA) 相当の構文検査を型検査に埋め込み，fluent interface の呼び出し規約を検査する手法が知られている．ソースコード 4 に構文検査付き fluent interface の定義例を示す．このインターフェースは `new SQL().select(...).from(...).where(...).run()` という呼び出しだけを受け入れ，それ以外はコンパイル時にメソッドが未定義であるというエラーが検出される．このインターフェース呼び出しの中で返り値であり同時にレシーバでもある中間デー

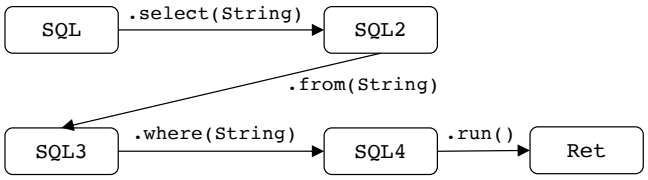


図 1. メソッド呼び出しによるレシーバ型の変遷

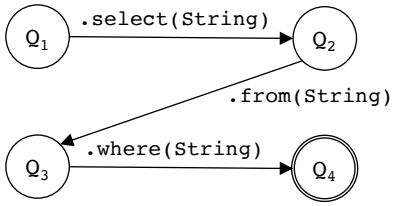


図 2. 決定性有限状態オートマトン

タの型がメソッド呼び出しによってどう変化するかを図 1 に示す．この中間データ型の変化は図 2 に示す決定性有限状態オートマトンの状態遷移と対応付けることが可能である．一般にある fluent interface の呼び出し規約がそれを言語とみなした時に正規言語に属するならば，その言語を認識する決定性有限状態オートマトンを型検査に埋め込むことで呼び出し規約が自動的に検査されるインターフェースを定義することが可能である．

スタックを型に埋め込むことで，単一状態リアルタイム決定性プッシュダウンオートマトン相当の構文検査を型検査に埋め込む手法も知られている．[11, 9] 適切な単一状態リアルタイム決定性プッシュダウンオートマトンを構築することで LL(1) 構文検査と同等の検査が可能である．LL(1) は正規言語を包含するより表現力の高い言語クラスである．

このような呼び出し規約が自動的に検査される検査付き fluent interface は通常，自動生成器を用いて生成される．ソースコード 4 のような単純なインターフェースでも 4 種類の互いに異なるクラス定義が必要となるが，呼び出し規約を表す文法が複雑になるにつれてその数は増加し，数百のクラス定義が必要となる場合もある．このような多数のクラスを手動で記述することは現実的ではないため自動生成器の開発が進められている．

このような生成器を用いて生成する fluent interface には提供できる機能の上限が生成器の機能によって制限され，生成器の開発時に想定されない機能の提供が困難であるという課題がある．そのような生成器の制限によって提供が困難な機能の例として sub-chaining style の呼び出しへの対応が挙げられる．Sub-chaining とはメソッド呼び出しの連鎖の一部を独立した連鎖として記述し，メソッドの引数に渡すような fluent interface の亜種である．以降では sub-chaining style と混同しないために，単なるメソッド呼び出しの連鎖の形で呼び出す形式を flat-chaining style と呼ぶ．ソースコード 5 に flat-chaining style と sub-chaining style それぞれの呼び出しの例を示す．Sub-chaining の例では `where` 以降の条件式に対応する部分が `whereClause` としてグループ化され，独立した呼び出しとして引数に渡されている．

既存の検査付き fluent interface 生成器には sub-chaining style への対応が欠けていることは中丸らによって指摘されている．[10] 彼らは sub-chaining style の呼び出しも受け入れることができる検査付き fluent interface 生成器の作成手法も提案している．しかし彼らはその作成手法として文法を有限状態オートマトンに変換する手続きしか示しておらず，その意味論も操作的にしか与えられていないと見なせる．このためその手法でインタフェースを生成可能な文法の条件がわからないなど，理論的解析が困難であった．

中丸らの手法では，非終端記号ごとに導出規則の右辺を決定性有限状態オートマトンに変換することで sub-chaining style 用インターフェースだけを先に生成する．次に，非終端記号に対応する

Listing 5. Sub-chaining style の呼び出し

```
/* Flat-chaining style */
new SQL().select("*")
         .from("members.db")
         .where()
             .column("member_id")
             .equals()
             .value("1002");
/* Sub-chaining style */
new SQL().select("*")
         .from("members.db")
         .whereClause(
             new SQL.WhereClause()
                 .where()
                 .column("member_id")
                 .equals()
                 .value("1002"));
```

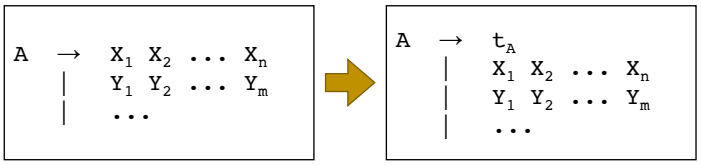


図 3. 文法書き換えのイメージ

遷移をインライン展開することで flat-chaining 用インターフェースの生成を試みるが，この展開は失敗することがある．展開に失敗するのはインライン展開の過程で自分自身が再帰的に出現する場合である．展開に失敗した自己参照を含む非終端記号については，導出規則の右辺から生成した有限状態オートマトンを (単一状態) リアルタイム決定性プッシュダウンオートマトンに変換することで flat-chaining 用インターフェースの生成を試みる．有限状態オートマトンからプッシュダウンオートマトンへの変換もまたインライン展開のような方法で行われる．２種類のインライン展開を経て得られる状態遷移を表すテーブルは LL(1) 構文解析に使用する LL(1) 構文解析表のようなものだと思われるが，先頭以外に出現する非終端記号も一部展開されるという特徴がある．

理論的解析が難しいという課題は，例えば入力として受け入れることができる文法の条件がわからなくなるという問題を引き起こす．[10] ではある文法が与えられた時に flat-chaining style を受け入れるインターフェースの生成に失敗することがあり，そのような場合には sub-chaining style のみを受け入れるインターフェースを生成するとされている．もしこの生成器を利用するユーザーが flat-chaining style 用インターフェースの生成に失敗し，かつ flat-chaining 用インターフェースを生成したい理由がある場合には，どのように文法を修正すれば flat-chaining 用インターフェースが生成されるようになるのか予想できないという問題が生じる．

また，理論的解析が難しいという課題は生成器の拡張を困難にするという問題も引き起こす．生成された検査付き fluent interface は指定された文法から外れた形式の呼び出しをコンパイルエラーとして指摘することや，flat-chaining style と sub-chaining style の両方の形式を受け入れるなど複数の性質が期待されるが，これらの性質がなぜ保証されているかの根拠が明らかではないため，拡張を加えた時にそれらの性質を破壊していないかを検証することが困難である．

3 提案: 文法書き換えによる sub-chaining 形式の解釈

本論文で我々は新しい fluent interface の生成手法を提案する．この手法を用いると sub-chaining style と flat-chaining style の両形式に対応する検査付き fluent interface を生成できる．提案手法は文法書き換えに基づいており，中丸らの手法 [10] より理論的な解析が容易である．我々の手法は (1) 文法書き換えによって sub-chaining style 形式の呼び出しを文法に埋め込み，(2) 既存の検査付き fluent interface 生成手法によってインターフェースを生成する の 2 ステップによって fluent interface を生成する．

(1) の文法の変形は，直感的には図 3 に示すように各非終端記号の導出規則の右辺に sub-chain に対応する終端記号を追加するだけである．図中では $\mathtt{t_A}$ が非終端記号 `A` に対応する終端記号である．Sub-chaining style はメソッド呼び出しの連鎖の一部を独立した呼び出しとして記述し，その結果をメソッドの引数として渡す．この sub-chain の結果を表す型は生成器によって自動生成される特別な型であるため同じシグネチャを持つメソッドは他に存在しない．そのため sub-chain を引数に受け取るメソッドは自然とオーバーロードによって他のメソッドと区別され，特別な終端記号とみなすことができる．文法，終端記号，非終端記号の定義は第 3.1 節で，文法書き換えの詳細については第 3.2 節で詳しく説明する．

(2) の非終端記号ごとのインターフェース生成はどんな手法で行ってもよい．我々の手法は flat-chaining style だけではなく sub-chaining style も受け入れる interface を生成するが，利用する fluent interface 生成手法は flat-chaining style 用のインターフェースしか生成しないものを使用する．Flat-chaining style 用インターフェースと sub-chaining 用インターフェースの間には受け入れることができるメソッドチェインの形式が異なる点だけではなく，sub-chaining 用では sub-chain を記述するためのインターフェースも提供しなければならないという違いがある．我々の手法では sub-chain 用インターフェースとして，対応する非終端記号を開始記号として生成したものを適当な名前空間に配置したものを使用する．Sub-chain 用のインターフェースを生成するために非終端記号の数だけインターフェース生成を繰り返す．開始記号も非終端記号のどれか 1 つであるため，ライブラリインターフェースとして sub-chain 用に生成したインターフェースを公開すればよい．

3.1 文法

形式的な議論では言語は記号列の集合であるとされ，通常その大きさは無限である．例としてプログラミング言語である C 言語をこの意味の言語と捉える場合は，コンパイラによる解釈方法が定義されているすべての文字列からなる集合こそが C 言語であると考える．有限の記述では特殊な場合を除いて無限の集合を表現できないため，言語を直接記述しようとしても簡単なものしか記述することはできない．

言語を有限の規則の組み合わせで記述したものを文法という．文法を記述するメタ言語としてバッカスナウア記法 (BNF)[4] などが知られている．以下，本稿では文法 G を文脈自由であると仮定して $\langle \Sigma, N, \delta, A_{init} \rangle$ の 4 つ組みで表現する．ここで，

- Σ はすべての終端記号からなる有限の集合である．終端記号は導出される言語を構成する文字のことであり，Σ は導出される言語で使うことができるすべての文字を含む集合である．
- N はすべての非終端記号からなる有限の集合である．非終端記号は導出過程の途中状態を表す記号であり，導出される言語では使用しない記号を用いる．
- $\delta : N \mapsto 2^{(\Sigma \cup N)*}$ は導出関数である．各非終端記号とそれから導出できる記号列の集合の間の対応を表す．
- $A_{init} \in N$ は開始記号である．

本稿で扱う文法は生成文法である．開始記号 A_{init} のみを含む列から始め，列中の適当な非終端記号に導出関数 δ を適用することを繰り返すことで言語を生成する．厳密には次の関数 $\overset{G}{derivation}$ を用いて，$\overset{G}{derivation}(A_{init})$ に含まれる終端記号のみからなる列の集合 $\overset{G}{derivation}(A_{init}) \cap \Sigma*$ を文法 G から生成される言語だと考える．

$$\overset{G}{derivation}(\overline{X}) = \{\overline{X}\} \cup \bigcup_{\overline{Y} \in \overset{G}{deriv_1}(\overline{X})} \overset{G}{derivation}(\overline{Y})$$

$$\overset{G}{deriv_1}(\overline{X}) = \bigcup_{\overline{Y_1}\, A\, \overline{Y_2} = \overline{X}, A \in N} \; \bigcup_{\overline{Z} \in \delta(A)} \overline{Y_1}\, \overline{Z}\, \overline{Y_2}$$

ここで，$\overline{X} \in \overset{G}{derivation}(\overline{Y})$ は「文法 G の下で記号列 $\overline{Y}$ から記号列 $\overline{X}$ を導出可能である」と読む．

我々の手法は fluent interface を生成するため，終端記号の集合 Σ はインターフェースが提供するすべてのメソッドからなる集合である．非終端記号 $A \in N$ は内部状態を表すもので flat-chaining style の呼び出しでは登場しないが，sub-chain 用のインターフェースを提供するクラス名や意味論を与えるために中間データとして構文木を生成する時のノード名などに使用することがある．例えばソースコード 5 中の `WhereClause` は `where` 以降に与えられる検索条件を表す非終端記号である．

3.2 文法書き換え f

我々の手法は (1) 文法書き換えによって sub-chaining style 形式の呼び出しを文法に埋め込み (2) 既存の検査付き fluent interface 生成手法によってインターフェースを生成する の 2 ステップによって fluent interface を生成する．(1) の文法書き換えを文法上の関数 f としてその定義を示す．図 3 にこの変形のイメージを示す．

$$f(G) = \langle \Sigma', N, \delta', A_{init} \rangle$$

ただし

- $\Sigma' = \Sigma \cup \{subchain(A) \mid A \in N\}$
- $\delta'(A) = \{subchain(A)\} \cup \delta(A)$

ここで，$subchain : N \mapsto \Sigma'$ は非終端記号に対して対応する sub-chain を表す終端記号を返す関数である．任意の $A \in N$ に対して $subchain(A) \notin \Sigma \cup N$ を満たす．

文法書き換えの例として，SQL の select クエリのようなライブラリインターフェースを表す文法の例 SQL を図 4 に，書き換え後の文法 $f(SQL)$ を図 5 に示す．SQL は fluent interface を表す文法であるため，その終端記号はメソッド呼び出しと対応する．引数の異なる 2 種類の `where` メソッドが Σ_{SQL} に含まれるが，これらはオーバーロードの解決が可能であり同じクラスに同時に定義できるため別々の終端記号として共存することができる．SQL の開始記号である *SelectQuery* から `.select(String selectList).from(String tableSource)` という列を導出することができる．これは，生成されるライブラリが `new SQL().select("*").from("members.db")` のような flat-chaining style の呼び出しを受け入れることを意味する．$f(SQL)$ では各非終端記号が導出規則の右辺に SQL には存在しなかった追加の要素を持つ．この `.selectQuery(SelectQuery t)` や `.whereClause(WhereClause t)` が sub-chain に対応する終端記号である．これらの規則が追加されたため，`.select(String selectList).from(String tableSource).whereClause(...)` のような sub-chaining style の呼び出しが可能となっている．

$SQL = \langle \Sigma_{SQL}, N_{SQL}, \delta_{SQL}, SelectQuery \rangle$

where

$\Sigma_{SQL} = \{$

`.select(String selectList)`,

`.from(String tableSource)`,

`.where(String searchCondition)`,

`.where()`,

`.column(String columnSpec)`,

`.equals()`,

`.value()`

$\}$

$N_{SQL} = \{SelectQuery, WhereClause\}$

$\delta_{SQL}(SelectQuery) = \{$

`.select(String selectList) .from(String tableSource)` $WhereClause$

$\}$

$\delta_{SQL}(WhereClause) = \{$

ϵ,

`.where(String searchCondition)`,

`.where() .column(String columnSpec) .equals() .value(String value)`

$\}$

図 4. select クエリ用インターフェースの文法 SQL

4 本手法によって fluent interface を生成できる入力文法の条件

この章では提案手法によって fluent interface を生成できるために入力文法が満たすべき条件を interface 生成のプロセスを逆にたどることで明らかにする．我々の手法は (1) 文法書き換えによって sub-chaining style 形式の呼び出しを文法に埋め込み，(2) 既存の検査付き fluent interface 生成手法によってインターフェースを生成する の 2 ステップによって fluent interface を生成する．(2) のインターフェース生成は Xu の方法 [11] を単純化した，LL(1) 文法から等価な単一状態リアルタイム決定性プッシュダウンオートマトンを生成し，その模倣を型検査に埋め込む方法で行うことを仮定する．

ある文法 G が LL(1) 文法であるとは，その文法から導出される文に含まれるどんな部分文でもある非終端記号に対応しているならば，その先頭の 1 文字を見ただけで導出規則の右辺に出現する複数の選択肢のどの選択肢によって導出されたかが一意に決定できることを意味する．我々の手法の (2) では各非終端記号それぞれに対応するインターフェースを生成するため，(1) で変形した後の文法は与えられた開始記号だけでなく，どの非終端記号を開始記号だと思ったとしても LL(1) 文法である必要がある．

ある文法 G がどの非終端記号を開始記号だと思ったとしても LL(1) 文法であるための条件を次に示す．

$f(SQL) = \langle \Sigma_{f(SQL)}, N_{SQL}, \delta_{f(SQL)}, SelectQuery \rangle$

$where$

$\Sigma_{SQL} = \{$

`.select(String selectList)`,

`.from(String tableSource)`,

`.where(String searchCondition)`,

`.where()`,

`.column(String columnSpec)`,

`.equals()`,

`.value()`,

`.selectQuery(SelectQuery t)`,

`.whereClause(WhereClause t)`

$\}$

$N_{SQL} = \{SelectQuery, WhereClause\}$

$\delta_{SQL}(SelectQuery) = \{$

`.selectQuery(SelectQuery t)`,

`.select(String selectList).from(String tableSource)` $WhereClause$

$\}$

$\delta_{SQL}(WhereClause) = \{$

`.whereClause(WhereClause t)`,

ϵ,

`.where(String searchCondition)`,

`.where().column(String columnSpec).equals().value(String value)`

$\}$

図 5. 書き換え後の文法 $f(SQL)$

- $\forall A \in N. \forall \overline{X}, \overline{Y} \in \delta(A). \overline{X} \neq \overline{Y} \implies \overset{G}{first}^{\Sigma}(\overline{X}) \cap \overset{G}{first}^{\Sigma}(\overline{Y}) = \phi$
- $\forall A \in N, \overset{G}{nullable}(A) \implies \overset{G}{first}^{\Sigma}(A) \cap \overset{G}{follow}^{\Sigma}(A) = \phi$
- $\forall A \in N. \forall \overline{X}, \overline{Y} \in \delta(A). \overline{X} \neq \overline{Y} \implies \text{not } (\overset{G}{nullable}(\overline{X}) \text{ and } \overset{G}{nullable}(\overline{Y}))$

ここで $nullable$，$first$，$follow$ はそれぞれ次のような関数である．

- $\overset{G}{nullable}(\overline{X}) = \epsilon \in \overset{G}{derivation}(\overline{X})$
- $\overset{G}{first}^{*}(\overline{X}) = \{Y \mid Y\overline{Z} \in \overset{G}{derivation}(\overline{X})\}$
- $\overset{G}{first}^{\Sigma}(\overline{X}) = \overset{G}{first}^{*}(\overline{X}) \cap \Sigma$
- $\overset{G}{first}^{\mathrm{N}}(\overline{X}) = \overset{G}{first}^{*}(\overline{X}) \cap N$
- $\overset{G}{follow}^{*}(A) = \{X \mid \exists B \in N. \overline{Y}\, A\, X\, \overline{Z} \in \overset{G}{derivation}(B)\}$
- $\overset{G}{follow}^{\Sigma}(A) = \overset{G}{follow}^{*}(\overline{X}) \cap \Sigma$
- $\overset{G}{follow}^{\mathrm{N}}(A) = \overset{G}{follow}^{*}(\overline{X}) \cap N$

$\overset{G}{nullable}(A)$ は非終端記号 A から空列を導出できるかどうかを，$\overset{G}{first}(\overline{X})$ は記号列 $\overline{X}$ から先頭の記号として導出可能な記号の集合を，$\overset{G}{follow}(X)$ はいずれかの非終端記号 B から導出できる文の中で記号 X の次に出現可能な記号の集合を表す．上付き文字の $*$，Σ，N はそれぞれ，集合がすべての記号，終端記号のみ，非終端記号のみを含むことを表す．上付きの $*$，Σ，N は集合に含まれる記号の種類を表す．上付き文字が $*$ ならばすべての記号を，Σ ならば終端記号のみを，N ならば非終端記号のみを含むことを意味する．

変換後の文法 $f(G)$ における $\overset{f(G)}{nullable}$，$\overset{f(G)}{first}$，$\overset{f(G)}{follow}$ を変換前の文法 G におけるそれらを用いて次のように書き表すことができる．

- $\overset{f(G)}{nullable}(\overline{X}) = \overset{G}{nullable}(\overline{X})$
- $\overset{f(G)}{first}^{\mathrm{N}}(\overline{X}) = \overset{G}{first}^{\mathrm{N}}(\overline{X})$
- $\overset{f(G)}{first}^{\Sigma}(\overline{X}) = \overset{G}{first}^{\Sigma}(\overline{X}) \cup \{subchain(A) \mid A \in \overset{G}{first}^{\mathrm{N}}(\overline{X})\}$
- $\overset{f(G)}{follow}^{\mathrm{N}}(A) = \overset{G}{follow}^{\mathrm{N}}(A)$
- $\overset{f(G)}{follow}^{\Sigma}(A) = \overset{G}{follow}^{\Sigma}(A) \cup \{subchain(B) \mid B \in \overset{G}{follow}^{\mathrm{N}}(A)\}$

以降ではここまでに議論した文法 G が LL(1) 文法である条件と変換後の文法 $f(G)$ における $nullable$，$first$，$follow$ を用いて我々の手法が fluent interface を生成できるための条件，すなわち変換後の文法 $f(G)$ が LL(1) 文法である条件を G の条件として求める．

4.1 G が LL(1) ならば $f(G)$ もまた LL(1) である

変換前の文法 G が LL(1) 文法であることを仮定する．これは第 4 章で触れた LL(1) の定義より，次の 3 条件が満たされることを意味する．

1. $\forall A \in N.\ \forall \overline{X}, \overline{Y} \in \delta(A).\ \overline{X} \neq \overline{Y} \implies \overset{G}{first}^{\Sigma}(\overline{X}) \cap \overset{G}{first}^{\Sigma}(\overline{Y}) = \phi$
2. $\forall A \in N, \overset{G}{nullable}(A) \implies \overset{G}{first}^{\Sigma}(A) \cap \overset{G}{follow}^{\Sigma}(A) = \phi$
3. $\forall A \in N.\ \forall \overline{X}, \overline{Y} \in \delta(A).\ \overline{X} \neq \overline{Y} \implies \text{not } (\overset{G}{nullable}(\overline{X}) \text{ and } \overset{G}{nullable}(\overline{Y}))$

以降では，この仮定の下で変換後の文法 $f(G)$ がこの 3 条件を満たすための条件を導出する．

4.1.1 $\forall A \in N.\ \forall \overline{X}, \overline{Y} \in \delta(A).\ \overline{X} \neq \overline{Y} \implies \overset{f(G)}{\boldsymbol{first}}{}^{\Sigma}(\overline{X}) \cap \overset{f(G)}{\boldsymbol{first}}{}^{\Sigma}(\overline{Y}) = \phi$ であるための条件

$\forall A \in N.\ \forall \overline{X}, \overline{Y} \in \delta'(A).\ \overline{X} \neq \overline{Y}$ である $\overline{X}$，$\overline{Y}$ を仮定する．$f(G)$ と G の関係より $\overset{f(G)}{first}{}^{\Sigma}(\overline{X}) = \overset{G}{first}{}^{\Sigma}(\overline{X}) \cup \{subchain(A) \mid A \in \overset{G}{first}{}^{\mathrm{N}}(\overline{X})\}$ であり，これは $\overline{Y}$ についても同様である．よって与式 $\overset{f(G)}{first}{}^{\Sigma}(\overline{X}) \cap \overset{f(G)}{first}{}^{\Sigma}(\overline{Y})$ は次のように変形できる．

$$
\begin{aligned}
&\overset{f(G)}{first}{}^{\Sigma}(\overline{X}) \cap \overset{f(G)}{first}{}^{\Sigma}(\overline{Y}) \\
&= (\overset{G}{first}{}^{\Sigma}(\overline{X}) \cup \{subchain(A) \mid A \in \overset{G}{first}{}^{\mathrm{N}}(\overline{X})\}) \cap (\overset{G}{first}{}^{\Sigma}(\overline{Y}) \cup \{subchain(B) \mid B \in \overset{G}{first}{}^{\mathrm{N}}(\overline{Y})\}) \\
&= (\overset{G}{first}{}^{\Sigma}(\overline{X}) \cap \overset{G}{first}{}^{\Sigma}(\overline{Y})) \\
&\quad \cup (\{subchain(A) \mid A \in \overset{G}{first}{}^{\mathrm{N}}(\overline{X})\} \cap \{subchain(B) \mid B \in \overset{G}{first}{}^{\mathrm{N}}(\overline{Y})\}) \\
&\quad \cup (\overset{G}{first}{}^{\Sigma}(\overline{X}) \cap \{subchain(B) \mid B \in \overset{G}{first}{}^{\mathrm{N}}(\overline{Y})\}) \\
&\quad \cup (\{subchain(A) \mid A \in \overset{G}{first}{}^{\mathrm{N}}(\overline{X})\} \cap \overset{G}{first}{}^{\Sigma}(\overline{Y}))
\end{aligned}
$$

どんな $C \in N$ に対しても $subchain(C) \notin \Sigma$ なので後者の 2 項は空集合である．また G は LL(1) 文法であるという仮定より $\overset{G}{first}{}^{\Sigma}(\overline{X}) \cap \overset{G}{first}{}^{\Sigma}(\overline{Y})$ もまた空集合である．

残る $\{subchain(A) \mid A \in \overset{G}{first}{}^{\mathrm{N}}(\overline{X})\} \cap \{subchain(B) \mid B \in \overset{G}{first}{}^{\mathrm{N}}(\overline{Y})\}$ もまたほとんどの場合で空集合だが，例外がある．まず $subchain(A) = subchain(B)$ であるような $A \in \overset{G}{first}{}^{\mathrm{N}}(\overline{X})$，$B \in \overset{G}{first}{}^{\mathrm{N}}(\overline{Y})$ の組みが存在することを仮定する．$subchain$ の定義より $A = B$ である．非終端記号である A，B は導出途中の状態を表しており，$A \in \overset{G}{derivation}(\overline{Z})$ ならば $\overset{G}{derivation}(A) \subseteq \overset{G}{derivation}(\overline{Z})$ である．これは $first$ でも同様であり，もし $A \in \overset{G}{first}{}^{\mathrm{N}}(\overline{Z})$ ならば $\overset{G}{first}{}^{\Sigma}(A) \subseteq \overset{G}{first}{}^{\Sigma}(\overline{Z})$ である．よって $\overset{G}{first}{}^{\Sigma}(A) \cap \overset{G}{first}{}^{\Sigma}(B) \subseteq \overset{G}{first}{}^{\Sigma}(\overline{X}) \cap \overset{G}{first}{}^{\Sigma}(\overline{Y})$ であるが，左辺は $A = B$ より $\overset{G}{first}{}^{\Sigma}(A) \cap \overset{G}{first}{}^{\Sigma}(B) = \overset{G}{first}{}^{\Sigma}(A)$ であり，右辺は G は LL(1) 文法であるという仮定より $\overset{G}{first}{}^{\Sigma}(\overline{X}) \cap \overset{G}{first}{}^{\Sigma}(\overline{Y}) = \phi$ である．すなわち，A，B の組みが存在するためには $\overset{G}{first}{}^{\Sigma}(A) = \phi$ の条件を満たす必要があるが，この条件を満たす非終端記号 A は $\overset{G}{derivation}(A) \cap \Sigma \subseteq \{\epsilon\}$ であるような A に限られる．空列しか導出しない非終端記号が文法上必要不可欠ということは考えにくいため，この例外ケースを無視したとしても実用上の問題は起きないだろうと考えられる．

4.1.2 $\forall A \in N, \overset{f(G)}{\boldsymbol{nullable}}(A) \implies \overset{f(G)}{\boldsymbol{first}}^{\Sigma}(A) \cap \overset{f(G)}{\boldsymbol{follow}}^{\Sigma}(A) = \phi$ であるための条件

第 4.1.1 節と同様の議論によって，$\overset{G}{derivation}(A) \cap \Sigma \subseteq \{\epsilon\}$ であるような A を除いて条件が成立することを示す．まず，$A \in N$ かつ $\overset{f(G)}{nullable}(A)$ である A を仮定する．$f(G)$ と G の関係より，

$$
\begin{aligned}
&\overset{f(G)}{first}^{\Sigma}(A) \cap \overset{f(G)}{follow}^{\Sigma}(A) \\
&= (\overset{G}{first}^{\Sigma}(A) \cup \{subchain(B) \mid B \in \overset{G}{first}^{\mathrm{N}}(A)\}) \cap (\overset{G}{follow}^{\Sigma}(A) \cup \{subchain(C) \mid C \in \overset{G}{follow}^{\mathrm{N}}(A)\}) \\
&= (\overset{G}{first}^{\Sigma}(A) \cap \overset{G}{follow}^{\Sigma}(A)) \\
&\quad \cup (\{subchain(B) \mid B \in \overset{G}{first}^{\mathrm{N}}(A)\} \cap \{subchain(C) \mid C \in \overset{G}{follow}^{\mathrm{N}}(A)\}) \\
&\quad \cup (\overset{G}{first}^{\Sigma}(A) \cap \{subchain(C) \mid C \in \overset{G}{follow}^{\mathrm{N}}(A)\}) \\
&\quad \cup (\{subchain(B) \mid B \in \overset{G}{first}^{\mathrm{N}}(A)\} \cap \overset{G}{follow}^{\Sigma}(A))
\end{aligned}
$$

最後の式の第１，３，４項は空であるが，その議論は第 4.1.1 節のものと同様なので省略する．また第２項が空でない条件もまた第 4.1.1 節のものと同様の議論によって $\overset{G}{derivation}(A) \cap \Sigma \subseteq \{\epsilon\}$ である場合のみに限られることを示すことができる．

4.1.3 $\forall A \in N.\ \forall \overline{X}, \overline{Y} \in \delta(A).\ \overline{X} \neq \overline{Y} \implies$ not $(\overset{f(G)}{\boldsymbol{nullable}}(\overline{X})$ and $\overset{f(G)}{\boldsymbol{nullable}}(\overline{Y}))$ であるための条件

$A \in N$ である A と $\overline{X}, \overline{Y} \in \delta'(A), \overline{X} \neq \overline{Y}$ である $\overline{X}$，$\overline{Y}$ を仮定する．$f(G)$ と G の関係より $\overset{f(G)}{nullable}(\overline{X}) = \overset{G}{nullable}(\overline{X})$ であり，これは $\overline{Y}$ についても同様である．よって not $(\overset{f(G)}{nullable}(\overline{X})$ and $\overset{f(G)}{nullable}(\overline{Y}))$ である．

4.2 G は LL(1) だが $f(G)$ は LL(1) ではない例

第 4.1 節の議論によって，提案手法には空文字列のみを導出する非終端記号やどんな記号列も導出しない非終端記号を含む文法は fluent interface の生成に失敗することがあることが明らかになった．本節ではこのような文法の例を示す．変換後の文法 $f(G)$ が LL(1) 文法にならないケースには $\forall A \in N.\ \forall \overline{X}, \overline{Y} \in \delta(A).\ \overline{X} \neq \overline{Y} \implies \overset{f(G)}{first}^{\Sigma}(\overline{X}) \cap \overset{f(G)}{first}^{\Sigma}(\overline{Y}) = \phi$ の条件に違反する場合と $\forall A \in N, \overset{f(G)}{nullable}(A) \implies \overset{f(G)}{first}^{\Sigma}(A) \cap \overset{f(G)}{follow}^{\Sigma}(A) = \phi$ の条件に違反する場合があった．前者の条件に違反する例として，次の文法 G_1 が挙げられる．

$$
\begin{aligned}
G_1 &= \langle \{a, b\}, \{S, A\}, \delta_1, S \rangle \\
\text{ただし} \quad \delta_1(S) &= \{A\,a, A\,b\} \\
\delta_1(A) &= \{\epsilon\}
\end{aligned}
$$

文法 G_1 から導出される言語は $\{a, b\}$ の２つの文のみから構成される有限の言語であり，文が a ならばその導出は $S \to A\,a \to a$ であり，文が b ならばその導出は $S \to A\,b \to b$ であり，どちらの方法で導出されたかを文の先頭１文字を先読みするだけで判定できるので G_1 は LL(1) 文法である．しかし $f(G_1)$ は LL(1) 文法ではない．なぜなら $f(G_1)$ から導出される言語は $\{a, b, subchain(A)\,a, subchain(A)\,b\}$ の４要素からなる有限の言語であるが，１文字先読みするだけでは先読みした先頭１文字が $subchain(A)$ だった場合に開始記号 S から $A\,a$ と $A\,b$ のどちらの方法で導出するかを決定することができないからである．

$\forall A \in N, \overset{f(G)}{nullable}(A) \implies \overset{f(G)}{first}{}^{\Sigma}(A) \cap \overset{f(G)}{follow}{}^{\Sigma}(A) = \phi$ の条件に違反する例として，次の文法 G_2 を挙げることができる．

$$G_2 = \langle \{a, b\}, \{S, A\}, \delta_2, S \rangle$$
$$\text{ただし} \quad \delta_2(S) = \{a\, S\, A, A\}$$
$$\delta_2(A) = \{\epsilon\}$$

文法 G_2 から導出される言語は $a^n \quad (n \geq 0)$ である．例えば文 $a\, a$ は $S \to a\, S\, A \to a\, a\, S\, A\, A \to a\, a\, A\, A \to a\, a\, A \to a\, a$ のように導出される．末尾に蓄積される A の列は空列としか対応しないため文法 G_2 が LL(1) 文法であるかどうかには影響しないが，先読みした 1 文字が $subchain(A)$ だった場合にそれが末尾に蓄積されたどの A によって消費されるのか決定できない．そのため $f(G_2)$ は本質的に曖昧であり，LL(1) 文法ではない．

4.3 G が LL(1) でないならば $f(G)$ もまた LL(1) ではない

第 4.1 節の議論によって我々の手法が fluent interface を生成できる文法の下限，十分条件が明らかになった．しかし第 4.1 節の議論だけでは上限は明らかではなく，LL(1) 文法ではない文法 G から fluent interface が生成できる可能性が残る．本節ではそのような可能性がないことを明らかにする．

変換前の文法 G が LL(1) 文法**ではない**ことを仮定する．つまり，次に示す 3 条件の少なくとも 1 つが充足されない．

1. $\forall A \in N.\ \forall \overline{X}, \overline{Y} \in \delta(A).\ \overline{X} \neq \overline{Y} \implies \overset{G}{first}{}^{\Sigma}(\overline{X}) \cap \overset{G}{first}{}^{\Sigma}(\overline{Y}) = \phi$
2. $\forall A \in N, \overset{G}{nullable}(A) \implies \overset{G}{first}{}^{\Sigma}(A) \cap \overset{G}{follow}{}^{\Sigma}(A) = \phi$
3. $\forall A \in N.\ \forall \overline{X}, \overline{Y} \in \delta(A).\ \overline{X} \neq \overline{Y} \implies \text{not } (\overset{G}{nullable}(\overline{X}) \text{ and } \overset{G}{nullable}(\overline{Y}))$

以降ではそれぞれの場合について，変換後の文法 $f(G)$ が同様の性質を継承することを示す．

4.3.1 $\forall A \in N.\ \forall \overline{X}, \overline{Y} \in \delta(A).\ \overline{X} \neq \overline{Y}$ and $\overset{G}{\boldsymbol{first}}{}^{\Sigma}(\overline{X}) \cap \overset{G}{\boldsymbol{first}}{}^{\Sigma}(\overline{Y}) \neq \phi$ を仮定する場合

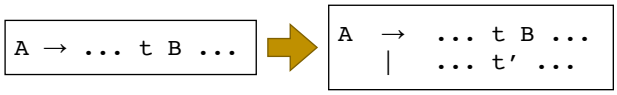
$f(G)$ と G の関係より $\overset{f(G)}{first}{}^{\Sigma}(\overline{X}) \supseteq \overset{G}{first}{}^{\Sigma}(\overline{X})$ である．よって $\overset{f(G)}{first}{}^{\Sigma}(\overline{X}) \cap \overset{f(G)}{first}{}^{\Sigma}(\overline{Y}) \supseteq \overset{G}{first}{}^{\Sigma}(\overline{X}) \cap \overset{G}{first}{}^{\Sigma}(\overline{Y}) \neq \phi$．$f(G)$ は G と同様の性質を継承しており，LL(1) 文法ではない．

4.3.2 $\forall A \in N, \overset{G}{\boldsymbol{nullable}}(A)$ and $\overset{G}{\boldsymbol{first}}{}^{\Sigma}(A) \cap \overset{G}{\boldsymbol{follow}}{}^{\Sigma}(A) \neq \phi$ を仮定する場合

$f(G)$ と G の関係より $\overset{f(G)}{nullable}(A) = \overset{G}{nullable}(A)$ であり，また $\overset{f(G)}{first}{}^{\Sigma}(A) \supseteq \overset{G}{first}{}^{\Sigma}(A)$ かつ $\overset{f(G)}{follow}{}^{\Sigma}(A) \supseteq \overset{G}{follow}{}^{\Sigma}(A)$ である．よって $\overset{f(G)}{first}{}^{\Sigma}(A) \cap \overset{f(G)}{follow}{}^{\Sigma}(A) \supseteq \overset{G}{first}{}^{\Sigma}(A) \cap \overset{G}{follow}{}^{\Sigma}(A) \neq \phi$. $f(G)$ は G と同様の性質を継承しており，LL(1) 文法ではない．

4.3.3 $\forall A \in N.\ \forall \overline{X}, \overline{Y} \in \delta(A).\ \overline{X} \neq \overline{Y}$ and $\overset{G}{\boldsymbol{nullable}}(\overline{X})$ and $\overset{G}{\boldsymbol{nullable}}(\overline{Y})$ を仮定する場合

$f(G)$ と G の関係より $\overset{f(G)}{nullable}(A) = \overset{G}{nullable}(A)$．よって $f(G)$ は G と同様の性質を継承しており，LL(1) 文法ではない．



ただし

```
t  = .m()
t′ = .m(B)
```

図 6. 便利メソッドを実現する文法書き換え

5 まとめ

我々は flat-chaining style と sub-chaining style の両形式を受け入れる静的呼び出し規約検査付き fluent interface を生成する生成器の新しい設計を示した．提案手法の特徴は単純な文法書き換えに基づいており，理論的に扱いやすい点にある．検査付き fluent interface 生成器を理論的に解析する例として，提案した設計を基に実装された生成器がインターフェース生成のために入力文法に要求する条件として次の 2 点を明らかにした．

- 入力が LL(1) 文法であるとき，そのような場合に限りインターフェースの生成に成功する．
- ただし例外的に，空列のみを導出する，あるいはどんな文も導出しない非終端記号が存在する場合は入力が LL(1) 文法であったとしてもインターフェースの生成に失敗する場合がある．

空列のみを導出する，あるいはどんな文も導出しない非終端記号は存在する意味がなく，また人手での除去が容易である．そのため例外ケースがインターフェース生成に失敗するより詳細な条件は調査していない．

我々の手法が単純な形の sub-chaining だけに対応しているのに対して，中丸らの手法 [10] ではより多様な機能を提供していることに注意されたい．例えば我々の手法では変換後の文法 $f(G)$ が LL(1) でなければ一切のインターフェースが生成されないのに対して，彼らの手法では一部で flat-chaining style が使えないかもしれないが何らかのインターフェースは生成される．また中丸らの手法では無引数のメソッドから何らかの非終端記号に対応するメソッドが呼べる場合，非終端記号に対応する sub-chain を直前のメソッドの引数に渡すことができる便利メソッドも生成される．このような便利メソッドは我々の文法書き換えに基づく手法でも図 6 に示すような書き換えを導入することで実現できるが，その場合の入力文法が満たすべき条件は再度の調査が必要である．

また我々の手法では書き換え後の文法からの fluent interface の生成を単純な方法で行うことを仮定したが，実際のツール開発ではより制限を緩和することができる．例えば非終端記号のインライン展開，導出規則中の共通部分のくくりだし，左再帰の除去などの変形を活用することでより様々な文法を LL(1) 文法に書き換えることが可能である．このように制限を緩和したツールは入力文法に対して我々の手法とは異なる条件を要求する．しかし同時に，このようなツールでも我々の結論を利用することで，少なくとも入力が LL(1) 文法ならばインターフェースの生成に成功するという結論は容易に導くことができる．

今後の課題として，より高度な fluent interface 生成手法を採用した場合に入力文法の制限がどう変化するかの調査と sub-chaining style への対応以外の追加機能を文法の書き換えとみなすことによる制限の調査の 2 つが挙げられる．我々の手法では fluent interface の生成に Xu らの手法 [11] を単純化した方法を利用することを仮定したため，文法書き換え後の文法が満たすべき条件は LL(1) であった．Gil らの手法 [6, 7] や山崎らの手法 [12] は LR 構文解析によって呼び出し規約を検査するため，より多様な文法から検査付き fluent interface を生成することができる．後段の fluent interface 生成をこれらの手法で置き換えた時，入力文法が満たすべき条件がどう変化するかは興味深い課題である．

また fluent interface 生成器が備えるべき便利機能は sub-chaining style への対応だけではない．例えば型パラメータを持つ総称メソッドを含む検査付き fluent interface は，生成器は開発されているが理論的な解析は困難である．総称メソッドを含む状態遷移はどのようなオートマトンのクラスに属するのか明らかではないため，我々の解析とは異なるアプローチが必要かもしれない．

参考文献

[1] Java Platform Standard Edition 8 Documentation - `interface Stream<T>`. `https://docs.oracle.com/javase/jp/8/docs/api/java/util/stream/Stream.html`.

[2] jMock - An Expressive Mock Object Library for Java. `http://jmock.org/`.

[3] jOOQ: The easiest way to write SQL in Java. `https://www.jooq.org/`.

[4] Control Data Corporation. Appendix D of the CDC Algol-60 Version 5 Reference Manual. `https://web.archive.org/web/20060925132043/https://www.lrz-muenchen.de/~bernhard/Algol-BNF.html`, 1979.

[5] Martin Fowler. FluentInterface. `https://www.martinfowler.com/bliki/FluentInterface.html`, 2005.

[6] Yossi Gil and Tomer Levy. Formal Language Recognition with the Java Type Checker. In *Proceedings of 30th European Conference on Object-Oriented Programming*, 2016.

[7] Yossi Gil and Ori Roth. Fling - A Fluent API Generator. In *33rd European Conference on Object-Oriented Programming (ECOOP 2019)*, 2019.

[8] Radu Grigore. Java Generics Are Turing Complete. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, 2017.

[9] Tomer Levy. A Fluent API for Automatic Generation of Fluent APIs in Java. Master's thesis, Israel Institute of Technology, 2017.

[10] Tomoki Nakamaru, Kazuhiro Ichikawa, Tetsuro Yamazaki, and Shigeru Chiba. Silverchain: a fluent API generator. In *Proceedings of the 16th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*, 2017.

[11] Hao Xu. *EriLex: An Embedded Domain Specific Language Generator*, p. 192–212. Springer-Verlag, 2010.

[12] Tetsuro Yamazaki, Tomoki Nakamaru, Kazuhiro Ichikawa, and Shigeru Chiba. Generating a fluent api with syntax checking from an lr grammar. *Proc. ACM Program. Lang.*, Vol. 3, No. OOPSLA, oct 2019.