日本語プログラミング言語「プロデル」の言語仕様と実装

馬場祐人

日本語によるプログラミングは 1980 年代より継続的に研究されてきたが, 初学者向けに限定しない汎用的な言語設計においては十分な検討が行われていない. 本稿では, 日本語プログラミング言語「プロデル」の言語仕様と実装について述べる. プロデルは予約語, 識別子, ライブラリ名称を日本語で統一するとともに, 日本語の語順に基づく構文を採用したプログラミング言語である. また, 助詞を用いた引数指定機構およびレシーバ補語の概念を導入し, 日本語表記とオブジェクト指向設計との両立を可能とした. 本稿では, プロデルの設計方針, 文法仕様, オブジェクト指向機構, ライブラリ設計および実装上で得られた知見について報告する.

Programming in Japanese has been continuously studied since the 1980s, but there hasn't been enough consideration for general-purpose language design that isn't limited to beginners. In this paper, I'll talk about the language specifications and implementation of the Japanese programming language "Produire." Produire is a programming language that unifies reserved words, identifiers, and library names in Japanese, and it uses syntax based on Japanese word order. It also introduces an argument specification mechanism using particles and the concept of receiver complements, making it possible to balance Japanese notation with object-oriented design. In this paper, we report on Produire's design policies, grammar specifications, object-oriented mechanisms, library design, and insights gained from its implementation.

1 はじめに

プログラミング言語の多くは英単語や数式表記を基盤として設計されており, ソフトウェア開発においても予約語, 識別子, ライブラリ名には英語表現が用いられることが一般的である. 一方で, 日本語によるプログラミングを実現しようとする研究は古くから存在し, さまざまな日本語プログラミング言語が提案されてきた.

日本語プログラミング言語が議論され始めた [7]1980 年代当時は, オペレーティングシステムや開発環境において日本語を扱うこと自体が困難であり, 日本語プログラミングの実現には技術的制約が存在した. しかし現在では Unicode の普及により, ソースコード中で日本語を利用するための技術的障壁は解消している. 識別子に日本語を用いることも一般的な処理系では可能となり, 日本語を活用した開発環境の整備は容易になった.

しかしながら, 多くの既存プログラミング言語では予約語や標準ライブラリが英語単語を用いて設計されているため, 日本語識別子を利用したとしてもソースコード全体としては英語と日本語が混在する表記となる. そのため, 日本語によるプログラム記述の利点や課題を十分に議論することは容易ではない.

そこで筆者は, 日本語表記によるプログラミングを実践的に検討するための環境として, 日本語プログラミング言語「プロデル」([2], [9]) を開発している. 本稿では, その言語仕様および実装について報告する.

2 言語設計の背景

2.1 日本語プログラミング言語の定義

本稿で扱う日本語プログラミング言語とは, 予約語や識別子に日本語表記を用いるプログラミング言語

The language specifications and implementation of the Japanese programming language 'Produire'

Yuto BAMBA, Personal, .

を指す. 単に日本語文字を識別子として利用できるだけでなく, 日本語の動詞や助詞を言語仕様として採り入れ, 日本語の語順に近い形でプログラムを記述できることを特徴とする.

また, 本研究は自然言語処理によって自由な日本語文を実行することを目的とするものではない. 日本語表記を用いつつも, 形式言語として厳密に定義可能な文法体系を持つプログラミング言語の実現を目的としている.

2.2 日本語プログラミング言語の意義

日本語話者を主たる利用者とするソフトウェア開発において, 設計対象となる概念の多くは日本語で認識され, 日本語で検討される. しかし実際のコードでは, それらを英語単語へ置換した名称を用いることが一般的である.

この置換過程では, 適切な英単語の選択が困難な場合や, 開発者間で用語選択が統一されない場合がある. また, スペルミスや英語表現の揺れによる問題も発生する. 日本語で設計される概念を日本語のままプログラム上に表現できれば, より直感的で適切な表現で設計できる.

もちろん, 国を跨がり母国語が異なる開発者が関わる開発や数学的表現を主体とする科学技術計算といった国際的に価値があるソフトウェアなど, 日本語表記を必ずしも必要としない分野も存在する. しかし, 日本語話者を対象としたソフトウェア開発に限定すれば, 日本語プログラミング言語が有効な選択肢となる可能性がある.

3 本研究が対象とする問題

3.1 既存の日本語プログラミング言語

既存の日本語プログラミング言語として代表的なものとして,Forth の逆ポーランド記法と日本語語順の対応関係に着目した Mind [8], Squeak 上で日本語文法によるプログラミングを実現したことだま on Squeak [6], 近年では日本語プログラミング言語なでしこ ([5], [3]) などが挙げられる. 教育用途ではドリトル [4] が挙げられる.

既存の日本語プログラミング言語処理系は教育用途や簡易スクリプト用途に留まり, 汎用的なソフトウェア開発やオブジェクト指向設計を前提とした言語機能は限定的である.

1. **日本語として自然な記述と, 形式言語として厳密かつ簡潔な記述の両立**
   日本語プログラミング言語は「〜なら」「〜の間繰り返す」「〜して」などの自然な論理表現を形式言語として扱うことが特徴である. しかし次の挙げたパーサ実装の制約から “日本語として自然な記述” と “プログラミング言語としての厳密性” の両立した言語仕様を設計することも難しい.
   - 単語と単語との間には空白や記号が必要になる
   - 特定の字句が解析に使われるため, それらを名前に用いることに制約がある

   日本語の特徴を活かしながら形式言語として設計することが求められる.
2. **日本語語順での記述とオブジェクト指向を記述する構文の体系化**
   一般的なオブジェクト指向言語では, 英語語順のようにレシーバ (操作対象) が行頭に固定されることを前提としている. 英語語順を前提としたレシーバ構造を言語仕様に適用すると日本語としての不自然さが生じる. 日本語プログラミング言語で,「名簿リストへ山田さんを加える」のような自然な日本語文をそのままオブジェクト指向の文法として表記する仕組みが必要である. 日本語の自然な語順とオブジェクト指向のメッセージ送信機構を体系的に統合する仕組みが必要である.
3. **汎用的なソフトウェア開発に耐える言語設計が不足**
   既存の日本語プログラミング言語は初学者や教育用途を主目的に設計されており, GUI アプリケーションやデータベースといった実用的な開発を前提としたライブラリ体系が十分に整備されていない. その結果, 実用開発には利用しづらいという課題が生じている.

3.2 本研究が解決する問題

前述の課題に対し, 本研究では次に挙げる点によっ

て解決する.

- 日本語の論理構造に基づく形式言語として厳密かつ簡潔な構文の体系化
- 日本語語順とオブジェクト指向のメッセージ送信を統合した構文の体系化
- 汎用的なソフトウェア開発に耐える日本語ライブラリ体系の設計と実装

これらにより, 日本語話者にとって自然な記述と, 形式言語としての厳密性を両立した汎用プログラミング言語の実現を目指す.

4 設計思想

プロデルの設計思想とし, 単に予約語や識別子を日本語化するなどの日本語の表層的な表記を利用するだけではなく「名詞句と動詞による文構成」「助詞による意味役割表現」をプログラミング言語仕様へ直接取り入れた点にある. これにより日本語話者にとって自然な記述と, 形式言語としての厳密な解釈の両立を試みている. それに加えて, プログラミング言語として簡潔な記述ができることも目指している.

4.1 名詞句と動詞による文構成

日本語では名詞句と動詞の組み合わせが文の基本構造となる. 例えば「デスクトップへファイルを移動する」という文では「移動する」が述語 (動詞) であり「デスクトップへ」と「ファイルを」がその補語として機能する.

プロデルではこの構造をそのまま言語仕様に取り込み, 「報告書ファイルを削除する」のように, 自然な命令文としてプログラムを記述できるようにした. これにより, コードが説明的となり, 結果的にコメントの必要性を減らすことを設計目標とした.

4.2 助詞による意味役割の表現

日本語では語順より助詞が文中の役割を示す特徴がある. プロデルではこの特徴を利用し, 助詞を引数の識別に利用する. 例えば「100 を 5 で割る」という文においては, 助詞が次のような意味を示している.

- を：割られる対象
- で：割る数

次に示すように助詞情報によって各引数が対応付けられる.

- 100 → を
- 5 → で

助詞によって対応付けられるため「5 で 100 を割る」のように語順を変更しても文法上は許容される. 助詞による意味指定は既存言語における名前付き引数と類似する機構である.

この特徴により, 自然言語のような読みやすさを維持する特徴としても作用する. コメントの自然文とコードの文体が一致しやすくなり, 可読性の向上を目指す.

5 言語仕様

本節では, プロデルの構文体系および言語機能について形式的な仕様を示す.

5.1 プログラム例

プロデルの構文は, 日本語話者が日常的に用いる論理表現をそのままコード化できるよう設計した. 条件分岐や繰り返しなどの制御構文も, 日本語の論理構造を反映しており, コードの意味が直感的に理解しやすくなることを目指した.

リスト 1 プロデルのプログラム例

```
1  //エラトステネスのふるい
2  総数は100
3  データは{}
4  Xを2から総数まで1ずつ増やしながら
5      データのX番目は1
6  そして
7
8  Xを2から総数まで1ずつ増やしながら
9      データのX番目が1なら
10         Xを報告する
11         YをXから総数まで1ずつ増やしながら
12             データの（X*Y）番目は0
13         そして
14     そして
15 そして
```

5.2 文の構造

プロデルの文は, 日本語文に近い構造で文を記述する. 述語（動詞）を文末に配置する日本語の構造をそ

のまま採用し, 処理対象は助詞によって明示される.

プロデルにおける文の形式的な文法をリスト 2 に示す.

リスト 2　文の構造 (BNF)

```
<Sentence> ::= <ComplementList> <
    DictionaryVerb>
<ComplementList> ::= { <Complement> }
<Complement> ::= <NounPhrase> <Particle>
```

補語 (Complement) は動詞 (Verb) に係る句であり, 引数として機能する. 補語は文末の動詞に対応し, 補語同士の順序には制約を設けない. 動詞は手順名や制御構文を表す基本要素である. 予約語として予め定義される語や標準ライブラリに定義された動詞を利用できるほか, ユーザが独自の動詞を定義することも可能である.

5.3　助詞

助詞 (Particle) は引数に意味的な役割を与える構文要素である. 既存言語における名前付き引数やキーワード引数に近い役割を持つ. 助詞を用いることで, 引数順序への依存を減らし, 可読性を向上させることができる.

助詞は引数の意味役割を表す構文要素であり, リスト 3 に示す語などが利用できる.

リスト 3　プロデルで用いる主な助詞 (BNF)

```
<Particle> ::= "を" | "へ" | "に" | "から" |
    "まで" | "として" | "という" | "が" |
    "の"
```

なお, これらは日本語文法上の助詞と必ずしも一致するものではなく, プログラミング言語における意味付けのために利用される.

例：「100 を 5 で割る」 → 「を」「で」により引数が束縛される.

5.4　名詞句

名詞句は “ファイル”,“ボタン” といった名詞を変数名・識別子として利用できる.

なお, 名詞句の部分には数式を記述することもでき, 算術演算記号 (+-/*) を用いて四則演算を記述できる. 自然文でも日本語文に数式を組み合わせて書くことと同様である.

リスト 4　名詞句 (BNF)

```
<NounPhrase> ::= <Identifier> | <Literal> |
    <BinaryExpr>
<BinaryExpr> ::= <NounPhrase> ("+" | "-" |
    "*" | "/") <NounPhrase>
```

5.5　動詞と形式補語

単独の動詞だけでは十分にプログラマが機能の意味を表現できない場合に, 形式補語を利用して動詞を補完して機能の名前の一部として宣言できる.

例えば,「四角形を描く」という機能であれば,

- 形式補語: 四角形を
- 動詞: 描く

で構成される.

5.6　複文

複文は, 複数の手順呼出し文を連用形で連結し,1 つの文として記述できる構文である. 動詞を連用形（〜して,〜て,〜んで等）にすることで, 次の手順呼出し文を連結して記述する言語仕様である. 複文を用いて日本語の自然な語順で処理の流れを記述できる.

リスト 5　複文の仕様 (BNF)

```
<CompoundSentence> ::= { <ComplementList> <
    ConjunctiveVerb> } <Sentence>
<ConjunctiveVerb> ::= <VerbHead> ("して" | "
    て" | "んで")
```

連用形で書かれた手順の戻り値は仮想の特殊変数「それ」に一時的に格納される. 例えば, リスト 6 に示す手順呼出し文では補語を省略でき, 暗黙的に「それ」が利用される.

リスト 6　複文のプログラム例

```
【一覧】は｛「トカゲ」,「エリマキトカゲ」,「
    エリ」｝
一覧から「カゲ」を取り出して「☆」で繋げて表
    示する
```

複文で記述しない場合, 例えばリスト 7 に示したよ

うな日本語文として冗長な文になってしまう.

リスト 7　複文を用いない冗長なプログラム例

```
1 一覧から「カゲ」を取り出す
2 [それを「☆」で繋げたもの]を表示する
```

複文を用いることで，リスト 7 に示したコード中の「〜たもの」という表記や「それ」変数を明示的に使わずに処理を表記できる. 複文で記述しない場合に比べてコードの表記を短縮できる.

なお, 連用形で連結できるのは戻り値を持つ手順のみであり, 補語は連用形動詞までにすべて消費される必要がある. また「〜とする」代入文は複文の末尾にのみ記述でき, 後続の連結はできない. このような制約は, 処理とデータの流れを必要以上に長文で書き下さず, 完結に記述するために設けている.

5.7　繰り返し文（反復処理）

プロデルの「繰り返し文」は, 日本語の反復表現のための構文である.「〜回繰り返す」「〜の間繰り返す」「〜になるまで繰り返す」など自然な日本語で反復を記述できる. なお「繰り返す」は省略できる.

リスト 8　反復処理仕様 (BNF)

```
1 <LoopSentence> ::= <CountLoop> | <WhileLoop>
      | <UntilLoop> | <ForEachLoop>
2
3 <CountLoop> ::= <NumberLiteral> "回" [ "繰り
      返す" ] <Block>
4 <WhileLoop> ::= <Condition> "の間" [ "繰り返
      す" ] <Block>
5 <UntilLoop> ::= <Condition> "になるまで" [ "
      繰り返す" ] <Block>
6 <Block> ::= { <Sentence> } "そして"
```

できる限り違和感ない日本語の自然な語順で反復処理を記述できること同時に, コードとしての記述の簡略化や可読性を両立する仕様を検討した.

自然な日本語において，プログラムの入れ子を含むブロック構造を表現することには限界がある．プロデルではブロックの終わりを「そして」というキーワードで示すこととした.

リスト 9　回数を指定する反復処理のプログラム例

```
1 5回
2     「わん」と表示する
3 そして
```

反復処理では行末が「〜回」「〜の間」「〜になるまで」「〜ながら」で終わる場合, 「繰り返す」を省略できる.

リスト 10　条件を指定するプログラム例

```
1 Aが10未満の間
2     Aを1増やす
3 そして
```

5.8　もし文（条件分岐）

プロデルの条件分岐は,「〜なら」で条件を記述する．「そうでなければ」で else に相当する分岐を記述できる.

リスト 11　条件分岐仕様 (BNF)

```
1 <IfStatement> ::= <IfStatementHead> <Block>
      [ "そうでなければ" <Block> ]
2 <IfStatementHead> ::= <Expression>
3 <CompoundIfStatementHead> ::= <
      ComplementList> <ConditionalVerb>
4 <ConjunctiveVerb> ::= <VerbHead> ( "なら" |
      "れば")
5 <Block> ::= { <Sentence> } "そして"
```

また「そして」はブロック終端を示す予約語である. これは日本語の段落構造とプログラムのブロック構造を対応付ける仕組みとして試みた.

リスト 12　条件分岐のプログラム例

```
1 AがBより大きければ
2     Aを表示する
3 そうでなければ
4     Bを表示する
5 そして
```

条件式内でも複文が記述できる. これにより, 条件式の補語「〜が」を省略し, 条件を自然文に近い違和感のない文体で記述できる.

リスト 13　複文を用いだ条件分岐のプログラム例

```
1 Aを比較してBより大きければCは1
```

5.9　手順による処理定義

プロデルでは関数やメソッドに相当する機能を “手順” と呼ぶ. 手順定義では, 仮引数, 対応する助詞, 形式補語, 動詞および戻り値を宣言する.

リスト 14　手順定義仕様 (BNF)

```
<ProcedureDef> ::=
  <ParamList> <DictionaryVerb> "手順"
  <Block>
  "終わり"

<ParamList> ::= { <Param> }
<Param> ::=
  "【" <Identifier> [ ":" <Type> ] "】" <
      Particle>
<Type> ::=
  "整数" | "文字列" | "配列" | "真偽値" | <
      Identifier>
```

BMI 計算する手順のコード例をリスト 15 に示す.

リスト 15　BMI 計算するプログラム例

```
54と160からBMIを計算して報告する

【体重:整数】と【身長:整数】から
    BMIを計算する手順
    【BMI】=体重/(身長/100)^2
    BMIを返す
終わり
```

この例では「と」「から」という助詞によって実引数と仮引数が対応付けられている. プログラムの利用者は引数順序を意識することなく手順を呼び出せる.

5.10　種類

プロデルではオブジェクト指向のクラスに相当する機能を「種類」と呼ぶ.

種類には手順および属性を定義できる. 種類を定義するコード例を, リスト 17 に示す. 種類定義は「とは」で開始し,「終わり」で終了する.

リスト 16　種類定義仕様 (BNF)

```
<ClassDef> ::= <Identifier> "とは"
               { <AttributeDef> | <
                   ProcedureDef> }
               "終わり"

<AttributeDef> ::= <Modifier> <Identifier> [
      ":" <Type> ]
<Modifier> ::= "+" | "-" | "#" | "~"
```

リスト 17　種類を定義するプログラム例

```
リストボックスとは
    +幅:整数
    +高さ:整数
    自分に【内容】を加える手順
    終わり
終わり
```

5.11　レシーバ補語

一般的なオブジェクト指向言語では, メソッド呼び出しの対象はレシーバとして行頭で指定する. 例えば Java では “list.add(item);” のように記述する.

一方, 日本語では対象物が助詞によって表現されるため, プロデルではレシーバも補語の一種として扱う.

リスト 18 に示すコード例では「名簿リストへ」がレシーバ補語であり, 「加える」という動作の対象となるオブジェクトを示している. このような仕様により, 日本語本来の文構造を保持したままオブジェクト指向のメッセージ送信機構を実現している.

リスト 18　レシーバ補語を含む文

```
名簿リストへ「山田さん」を加える
```

なお, 日本語では動詞ごとに適切な助詞が異なるため, レシーバ補語の助詞も固定ではない. プログラマは「へ」「を」「から」など, 操作内容に応じた助詞を用いることができる.

6　言語処理系の実装

6.1　解析処理

字句解析 → 構文解析 → 意味解釈 → 中間コード生成 の順で実行される.

6.2　字句解析

日本語プログラミング言語は, 自然な語順と表記でプログラムを記述できるよう設計されている. しかし, 一般的なプログラミング言語と異なり, スペースや記号による明確な字句区切りが存在しないため, 既存の字句解析器（Lex / Flex）では解析が困難である. 例

えば「値が 10 なら値に 1 を加える」のような文では, 語の境界が明示されていないため, 予約語・変数名・助詞などを正しく切り出す仕組みが必要となる.

自然言語処理では形態素解析が広く用いられるが, 辞書には日本語の膨大な語彙が必要であり, 言語処理系に組み込むには非現実的である. 軽量な言語処理系を目指す日本語プログラミング言語にとっては過剰である. 一方, プログラム言語として使われる語彙は自然言語より限定されているため, より軽量な辞書で解析できる.

プロデル処理系では, 言語仕様を実現するためには, パーサーを実装した. 形態素解析の最長一致法を応用し, 次の方針で字句解析と構文解析を平行して行う.

- 辞書に登録された語彙を最長一致で切り出す
- 品詞を名詞, 助詞, 動詞, 記号, 予約語に限定する
- 単語連接規則は使用せず, 構文解析で文法を判定する

プログラム内で宣言される種類・変数・手順は, 初回字句解析時には辞書に存在しないため未知語として扱う. 構文解析後に新たに定義された語彙を辞書に追加し, 再度字句解析し構文解析する.

6.3 構文解析

LL(k) パーサを採用し, 字句解析と構文解析を平行して行う.

1. **レシーバ補語解決**
   動詞が許容する助詞に基づきレシーバ候補を選択し, 曖昧性がある場合はエラーとする.
2. **複文処理**
   連用形動詞を連結し, 戻り値を「それ」に格納して次の手順に伝播する. 複文は「〜して」「〜て」などの連用形で手順を連結する構文である.
   複文処理の流れ
   - 連用形動詞を検出
   - 手順呼出しを AST の連結ノードとして構築
   - 手順の戻り値を仮想変数「それ」に格納
   - 次の手順呼出しの補語が省略されている場合,「それ」を補語として補完
   - 終止形の動詞または「とする」を検出すると複文の終端として扱う

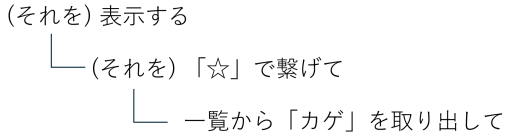

図 1　複文の AST

7 ライブラリ設計

日本語プログラミング言語の実用性を評価するためには, 言語仕様だけでなくライブラリ設計も重要である. プロデルでは.NET Framework 上で動作する処理系を実装し, 日本語表記に適したライブラリ体系を整備した.

ライブラリ構築では次の 2 つの方針を採用した.

- 既存クラスを継承し, 日本語名称へ翻訳する方式
- 日本語文法に適合するよう再設計する方式

GUI 部品などの視覚的な概念については 1. の方式を適用した. 一方, 正規表現や暗号化, 数学演算などについては単純な翻訳では不自然となるため,2. の方式により再設計を行った.

これらの文は形式言語として厳密に解釈可能である一方, 日本語文としても自然な読み方が可能である.

例えばファイルおよびフォルダ操作は, リスト 19 のコード例のように記述できる.

リスト 19　オブジェクト指向に則ったファイルとフォルダ操作のプログラム例

```
1 デスクトップに「書類」というフォルダを作成して、書類フォルダとする
2 デスクトップから「報告書.xlsx」というファイルを取得して、
3 報告書ファイルとする
4 報告書ファイルの絶対パスを報告する
5 報告書ファイルを書類フォルダへコピーする
6 書類フォルダを削除する
7 報告書ファイルをマイドキュメントへ移動する
```

8 本研究の新規性

本研究で提案する日本語プログラミング言語「プロデル」は, 既存の日本語プログラミング言語汎用的なソフトウェア開発やオブジェクト指向設計を十分に扱

えないという課題に対して, 次の新規性を有する.

1. **日本語話者にとって自然な記述と, 形式言語として厳密で簡潔な記述の両立**
   「〜なら」「〜回繰り返す」など, 日本語話者が日常的に用いる論理表現を形式言語として体系化した制御構文を設計した. 単に予約語を日本語で置き換える方法とは異なり日本語の論理構造を形式的に定義し, 汎用言語として実装している.
   また, 日本語の連用形 (〜して,〜て等) を形式言語として定義し, 複数の手順呼出しを自然文のように連結する「複文」構文を導入した. 連用形で連結された手順の戻り値を仮想変数「それ」に自動格納し, 後続の手順に暗黙的に伝播する仕組みは, 既存言語のパイプライン構文やメソッドチェーンを日本語の文法構造に基づいて表現している.
2. **レシーバ補語による日本語語順とオブジェクト指向の統合**
   一般的なオブジェクト指向言語では, メソッド呼び出しのレシーバは行頭に固定される. これに対しプロデルは, レシーバを補語の一種として扱う「レシーバ補語」を導入し, 「名簿リストへ山田さんを加える」のように, 日本語の自然な語順を保持したままオブジェクト指向のメッセージ送信を実現した. 助詞にレシーバの役割を柔軟に割り当てる仕組みは既存言語に見られず, 日本語語順と OOP の両立を形式的に達成した点が新しい.
3. **日本語文法に適合したライブラリ体系の設計**
   .NET Framework 上で動作する処理系を実装し, GUI・ファイル操作・正規表現・暗号化などの汎用ライブラリを日本語語順・助詞体系に適合するよう再設計した点も新規性がある. 単なる名称翻訳ではなく, 日本語の述語中心の構造に基づいて記述できるように API 設計を行った点が特徴的である.

9 利用事例

プロデルは,GUI アプリ, 業務アプリ, ファイル操作,Web API, 競技プログラミングなど多様な用途で利用できる.

1. **実装の公開**プロデルの実装は公式 Web サイトにて無料で一般公開している. 安定版であるプロデル 2.0 は,2024 年 1 月に公開して以来 2026 年 8 月 10 日時点で延べ 11631 件ダウンロードされている.
   公開しているプロデルのリリースには, 次のような分類のサンプルコードを 130 個同梱している. これらのサンプルコードは実際に動作する.
   - 一般的なアプリ (テキストエディタや画像加工ツール)
   - ゲーム (ブロック削り, リバーシ, シューティング)
   - 業務アプリ (在庫管理アプリ)
   - Web API を用いたデータ取得
   - Excel・CSV ファイルの業務自動化スクリプト
   - データベースを取り扱うアプリ
   - ファイル整理・変換処理
   - その他 GUI アプリ

   サンプルコードでは実用用途でソフトウェアを実装したものを動作させて確認できる.
2. **競技プログラミング (AtCoder)** プロデルは, 国内最大級の競技プログラミングサイト “AtCoder” において 2026 年現在コンテストに参加するために用いる提出プログラミング言語として選択できる (図 2).
   - 標準入出力・アルゴリズム実装が可能
   - 競技問題を解くための基本的なライブラリが実現可能
   - 実行性能が競技環境で許容されている

   AtCoder での言語別の提出状況をまとめた有志のサイト “Atcoder Problems” の “Language Owners” [1] では, AC(Accepted, 正解) となったプロデルの提出数が 2026 年 8 月時点で最高 276 AC であることが記録されている (表 1).
   プロデルは教育用途に留まらず競技用途でも利用可能である.

図 2　AtCoder におけるプロデル言語の選択

表 1　Atcoder Problems Language Owners Ranking の AC 数 (一部)

言語	最高値	平均値 (上位 20 名)
Python	6595	4195.7
なでしこ	252	79.1
プロデル	276	34.8

10　結論

本研究では, 日本語の文法特性を形式言語として再構成し, 日本語話者にとって自然な記述と形式的厳密性を両立する汎用プログラミング言語「プロデル」を解説した. 助詞構文, レシーバ補語, 複文構造, 日本語論理構造に基づく制御構文など, 既存言語では扱われてこなかった設計要素を体系化し, 実用的な開発環境として実装した. 今後は, 型推論,IDE 最適化などの観点からさらなる拡張を進め, 日本語プログラミング言語研究の基盤としていきたい.

参 考 文 献

[1] kenkoooo: Atcoder Problems, 2026. https://kenkoooo.com/atcoder/#/lang.
[2] ゆうと: プロデル, 2026. https://produ.irelang.jp/.
[3] クジラ飛行机: 日本語プログラミング言語「なでしこ」に関する解説, 情報処理, Vol. 62, No. 5(2021), pp. e26–e42.
[4] 進兼宗, 理英御手洗, 多哉子中谷, 眞吾福井, 靖久野: 学校教育用オブジェクト指向言語「ドリトル」の設計と実装, 情報処理学会論文誌. プログラミング（*PRO*）, Vol. 42, No. SIG11(PRO12)(2001), pp. 78–90.
[5] 峰章酒徳: 日本語プログラミング言語「なでしこ」, コンピュータ ソフトウェア, Vol. 28, No. 4(2011), pp. 23–28.
[6] 学杉浦: アルゴリズム構築能力育成の導入教育：実作業による概念理解に基づくアルゴリズム構築体験とその効果, 情報処理学会論文誌, Vol. 49, No. 10(2008), pp. 3409–3427.
[7] 神田泰典, 杉本正勝, 沢井進: エンドユーザのための日本語によるプログラミング, 情報処理, Vol. 21, No. 3(1980), pp. p208–213.
[8] 正樹平賀: 日本語プログラミング言語 Mind, コンピュータ ソフトウェア, Vol. 6, No. 2(1989), pp. 170–178.
[9] 馬場祐人, 筧捷彦: RA-009 日本語プログラミング言語における字句解析 (モデル・アルゴリズム・プログラミング, 査読付き論文), 情報科学技術フォーラム講演論文集, Vol. 8, No. 1(2009), pp. 61–64.