連続した条件分岐に対する プログラミング学習用プルーフリーダの提案
―変数の範囲に関する条件の区間を用いた判定―

2019SE035 宮島寛斗　2019SE049 小澤秀輔
指導教員：蜂巣吉成

1　はじめに

プログラミング演習では，学習者は課題に対するプログラムを作成し，期待される実行結果が得られるか確認して学習を進める．教育者は学習者に習得させたい事項である「教育意図」を満たすプログラムを期待するが，学習者は教育意図を意識せずにプログラムを記述することがある．学習者のソースコードが教育意図を満たすか確認するには，教育者が目視で確認する必要があり，大きな負担となる．また，学習者は作成したプログラムが教育意図を満たす適切なものか判断できない．我々の研究室では，学習者のプログラムが教育意図を満たすか判定するプログラミング学習用プルーフリーダを提案している [1][2]．学習者の解答と模範解答からチェック対象部分をパターンマッチングで抽出し，それらを比較して教育意図を満たすか判定する．

if 文や if-else 文はプログラミング演習で基本となる学習項目であり，実行結果が同じでも学習者によって条件の記述方法は様々であるので，適切な記述方法かチェックするのが望ましい．従来のプルーフリーダでは，if の制御式に対して演算子や閾値を模範解答と比較して判定しているが，else によって決まる暗黙的な条件や前後の if の条件を考慮しておらず，記述不要な条件を判定できない．また，プログラミング演習では else if で記述すべき箇所を連続した if 文で記述している学習者や，else if で記述していても else の特性を理解していない学習者がいる．さらに閾値や演算子を誤る場合があるが，テストケースに対する実行結果が変化しない場合は，学習者が不適切な記述に気づくことは困難である．

本研究では，学習者が if 文や if-else 文の条件を適切に記述できるように支援するプルーフリーダを提案する．if の制御式に記述された条件の範囲を区間で表現し，区間に対して重複や否定等の計算を行い，制御式や else が適切に記述されているか判定する．区間で表現した条件，排反条件，前提条件，実行条件と else の有無を用いて，else if で記述できているかや閾値や演算子の誤りの可能性等をチェックし，必要に応じて指摘する．判定に区間を用いることで，条件の範囲に対するチェックが可能となる．

2　関連研究

C 言語のコーディングチェッカやプログラミング教育を支援するツールが提案されている．C-Helper[3] は C 言語初学者向けの静的解析ツールである．初学者が起こしやすいミスを検出し，わかりやすい言葉で解決策を提示する．しかし if 文や if-else 文は検出対象ではないので，if の条件に対する教育意図を考慮した判定ができない．CX-Checker[4] は柔軟なカスタマイズ機能を有する C 言語のコーディングチェッカである．XPath，DOM，ラッパを用いたルールにより独自のコーディング規約を定義できる．教育意図に沿って独自のコーディング規約を定義すると教育意図の判定も可能だが，if 文の連続や if-else 文の制御式に対応するルールの記述が困難である．原田らのツール [1] は，教育者の判定基準に基づいた判定木を用いて教育意図を判定する．繰り返しを用いた計算を行うプログラムを対象としており，条件分岐に対応する判定要素抽出パターンの記述が困難である．蜂巣らのツール [2] は，学習者の解答と模範解答を比較して教育意図を判定する．if 文や if-else 文に対して演算子と閾値のチェックが可能である．連続の if 文や if-else 文に対して，else で暗黙的に決まる条件や前後の if の条件を考慮した判定はできない．

3　条件分岐における問題分析

次の条件を満たすものを適切な条件分岐とする．

- else で暗黙的に決まる条件を記述していない
- 重複や不足のない条件を記述している
- if 文の条件範囲が他の if 文の条件範囲と重ならない場合，else if で記述している

ソースコード 1 は，適切な条件分岐の条件を満たす模範解答である．ソースコード 2 は，else で暗黙的に決まる条件を記述している条件分岐である．同じ閾値が 2 箇所あり，閾値変更の際の手間が増えるため保守性が低い．ソースコード 3 は，閾値の誤りと '=' の不足がある条件分岐である．3, 4 行目の条件が重複し，80 に対する条件が不足しているため不適切な条件分岐である．ソースコード 4 は else if で記述すべき if 文である．else if で記述することで，条件を簡潔に記述できる．

ソースコード 1　模範解答

```
1 if (var >= 90) {
2 } else if (var >= 80) {
3 } else if (var >= 70) {
4 } else {
5 }
```

ソースコード 2　暗黙的に決まる条件が記述されたもの

```
1 if (var >= 90) {
2 } else if (var >= 80 && var < 90) {
3 } else if (var < 80) {
4 }
```

ソースコード 3 演算子と閾値に誤りがあるもの

```
1 if (var >= 90) {
2 } else if (var > 80 && var < 90) {
3 } else if (var >= 70 && var < 80) {
4 } else if (var < 75) {
5 }
```

ソースコード 4 else if で記述すべき if 文

```
1 if (var >= 90) {}
2 if (var >= 80 && var < 90) {}
3 if (var < 80) {}
```

4 連続した条件分岐に対するプルーフリーダ

4.1 概略

本研究では，連続した条件分岐における条件式に対する教育意図判定と誤りの可能性を指摘するプログラミング学習用プルーフリーダを提案する．本プルーフリーダでは if の条件を「区間」として表現し，3 節で述べた適切な条件分岐の条件を満たさない if 文や if-else 文をチェックする．

4.2 対象とするソースコード

次の条件をすべて満たす連続の if 文，if-else 文をチェック対象とする．

- 同一の変数に対する条件分岐をするもの
- 一連の条件分岐で変数の値が不変であるもの
- 整数型，実数型の変数に対し，ある値との等価，あるいは大小関係を比較するもの

4.3 判定項目

3 節で述べた適切な条件分岐をチェックするために，区間を用いて次の 5 項目をチェックする．

A if 文が連続している場合に範囲が重なっている条件が存在するか

B 一連の条件分岐で条件に重複がない場合，else if で記述しているか

C else により暗黙的に決まる条件が記述されていないか

D True にならない条件が記述されていないか

E 一連の条件分岐で条件の範囲が途切れていないか

4.4 区間

4.4.1 概略

本研究における区間とは，条件式に記述された条件の範囲を表したものである．明示的に決まる区間と暗黙的に決まる区間を調べることにより，条件式の区間の重複や不足をチェックする．区間を用いることにより，条件式の判定に必要な比較演算子や閾値を一つのデータ構造で取り扱うことが可能になる．また，else で暗黙的に決まる条件や True にならない条件の範囲が導出可能になる．

4.4.2 定義

閉区間，開区間，半開区間は数学の定義に準拠する．本研究で独自に用いる用語について定義する．

定義 4.1 一点集合は，閉区間 $[t_1, t_2]$ において $t_1 = t_2$ を満たすものである．

定義 4.2 区間 $[t_1, t_2]$ を s とするとき，s の下端点を $s.low = t_1$，s の上端点を $s.high = t_2$ と表現する．また，$s.low$ を下端値，$s.high$ を上端値と呼ぶ．

本研究では if の条件式に着目する性質上，複数の範囲に対する区間を取り扱う．そこで，「単純区間」，「複合区間」を定義し，区間で表現できる範囲を拡張する．複合区間の導入により，複数の範囲をまとめて表現できる．

定義 4.3 単純区間とは，閉区間，開区間，半開区間のいずれかで表現される区間 1 つを表す．

定義 4.4 複合区間とは，単純区間が複数個存在し，それらの単純区間の間に論理和 (OR) や論理積 (AND) の関係性が存在する区間を表す．

単純区間 s, t の関係性には，「重なっている」と「連結」がある．それぞれの定義を次に示す．

定義 4.5 「重なっている」とは，単純区間 s, t において，次の 2 つの条件のいずれかを満たす状態である:

(1) $s.low < t.high$ かつ $t.low < s.high$ を満たす．

(2) $s.low = t.high$ または $t.low = s.high$ を満たすとき，値の等しい端点は閉端点である．

定義 4.6 「連結」とは，単純区間 s, t において，$s.low = t.high$ または $t.low = s.high$ を満たし，値の等しい端点は一方が開端点，もう一方は閉端点の状態である．

4.4.3 演算

4.3 節で示した項目をチェックするためには，次の区間演算が必要である．複数の単純区間に対して「重なっている区間」を導出する重複演算，「排反」を導出する否定演算，区間の「減算」を定義する．

重複演算では，単純区間 s, t において $\exists i(i \in s \wedge i \in t)$ を満たす実数 i が存在する範囲 (以降，重なっている区間と呼ぶ) を導出する．重なっている区間の導出のために，単純区間 s, t の重なり方を考える．単純区間 s, t において，端点の関係性は「両端点が一致しない」，「両端点が一致する」，「上端点のみ一致する」，「下端点のみ一致する」の 4 通りである．そのうち，両端点が一致しない場合の s, t の重なり方は 6 通り，両端点が一致する場合は 3 通り，上端点のみ一致する場合は 5 通り，下端点のみ一致する場合 5 通りの重なり方が存在する．s, t の重なり方は 19 通り存在し，それぞれに対応する重なっている区間は一意に定まる．

否定演算では，区間が表現する範囲以外の範囲 (以降，排反と呼ぶ) を導出する．両端点が定数の単純区間の場合，開端点と閉端点を入れ替え，∞ を用いた 2 つの単純区間で排反を表現する．片端点が無限の単純区間の場合，両端の値を入れ替えた後に無限でない端点の閉と開を入れ替え，無限の端点は正負を反転して開端点にすると排反を導出できる．複合区間の排反を導出する場合には，複合区間を構成する単純区間の排反を導出し，論理演算子の and と or を入れ替えて導出する．

減算では，単純区間 s, t に対し $s - t$ を導出する．$s - t$ は，s から s, t の重なっている区間を除いた範囲である．t

の排反を否定演算で導出し，導出した排反と s の重なっている区間を導出すると $s-t$ が得られる．

4.5 判定要素

4.3 節で示した項目をチェックするため，条件，前提条件，実行条件，else の有無を判定要素とする．「条件」とは，制御式に記述された条件を区間に変換したものである．「前提条件」とは，if が実行される条件のうち，前までの if から暗黙的に決まる条件である．前提条件はひとつ前の if の排反条件と前提条件の重なっている区間に等しくなる．最初の if の前提条件は $(-\infty, \infty)$ とする．「実行条件」とは，制御式が True となる区間である．実行条件は，条件と前提条件の重なっている区間に等しくなる．True になる区間が存在しない場合は None とする．else の有無はソースコードを字句解析して導出する．

4.6 判定

判定項目 A, B の判定では，判定要素の「条件」と「else の有無」を利用する．一連の if 文における else の有無を確認し，すべての if 文に対して else が無いことをチェックする．続いて，それぞれの if の条件に着目し，互いに重なっている区間が存在するかを重複演算により調べる．重なっている区間が存在すれば誤りの可能性がある if 文として指摘し，そうでなければ else if で記述するようにアドバイスする．

判定項目 C の判定では，判定要素の「条件」と「前提条件」を利用する．前提条件は暗黙的に決まる条件であり，必ずしも条件として記述する必要がないので，条件に前提条件が含まれているかチェックする．チェックの前に，両端点が定数である単純区間に対して論理演算子 *and* を用いた複合区間に変換する．条件の単純区間と前提条件の単純区間が一致すれば，その区間に該当する条件記述が不要であることを指摘する．

判定項目 D の判定では，「実行条件が True にならない if」,「条件式に記述された条件の一部の範囲が True にならない if」をチェックする．前者は，判定要素の「実行条件」が None である if を指摘する．後者は，判定要素の「条件」と「実行条件」を利用する．減算により「条件 – 実行条件」を導出すると，条件式に記述されているが実行時には True にならない区間を導出できる．一方，模範解答のソースコード 1 のように，else で暗黙的に決まる条件を意識して条件を記述することが一般的であり，条件を簡潔に記述するためにあえて True にならない部分を含む条件記述をする場合がある．そこで，指摘が不要である条件記述を考察する．ソースコード 1 の 2 行目の if の判定要素を表 1 に示す．

表 1 ソースコード 1 の var >= 80 における判定要素

条件 (記述)	条件 (区間)	前提条件	実行条件	else
var >= 80	[80, ∞)	(−∞, 90)	[80, 90)	有

表 1 の条件 (区間)– 実行条件を計算すると，[90,∞) となり，これは前提条件である $(-\infty, 90)$ の排反と一致する．よって，True にならない区間が存在していても指摘が不要なソースコードの特徴は「True にならない区間と前提条件の排反が一致すること」とする．

条件式に記述された条件の一部の範囲が True にならない if を判定するアルゴリズムをアルゴリズム 1 に示す．for $e \leftarrow l$ はリスト l の要素を 1 つずつ e として処理し，for $(e_1, e_2) \leftarrow (l_1, l_2)$ はリスト l_1, l_2 に対して要素の直積 (e_1, e_2) を 1 つずつ処理する．

アルゴリズム 1 条件の一部の範囲が True にならない if

```
function 判定 (判定要素)
    z を空リストとする
    for x ← 一連の if の判定要素リスト do
        リスト c と c′ を空リストとする
        for (c1, c2) ← (x の条件の単純区間リスト, x の実行条件の単純区間リスト) do
            リスト c に c1 − c2 を追加する
        リスト c に格納された区間の重なっている範囲を求め，リスト c′ に追加する
        x の前提条件の排反を d とする
        for s ← c′ do
            if s ≠ d then
                (x, s) を z に追加する
    return z
```

判定項目 E の判定では，閾値が誤っている可能性のある if と '=' が不足している可能性のある if をチェックする．判定要素の実行条件を利用し，実行条件が連結でない if を指摘する．実行条件において連結でない区間は，隣り合う区間との連結部の端点の値が異なる場合と，端点の値は等しいが連結部の端点がどちらも開端点となる場合がある．前者の場合には閾値誤りの可能性を指摘し，後者の場合には '=' が不足している可能性を指摘する．

4.7 フィードバック

本研究で提案する判定に対するフィードバックは，教育意図の観点から修正を促すものと，誤りの可能性を指摘するものに区別できる．判定項目 A, D, E はソースコードの誤りの可能性を指摘するものであり，必ずしも指摘が正しいことを保証できないが，学習者に優先的に確認を促す項目である．判定項目 B, C は教育意図を満たすプログラムを記述できるようにアドバイスするものである．修正後に実行結果が変化しないので，学習者に修正を推奨する項目である．指摘の意味合いが異なるので，前者を「warning」，後者を「advice」と分類し，自然言語により指摘箇所の説明や修正の目的を学習者にフィードバックする．

5 プルーフリーダの設計と実現

5.1 設計

ツールの全体像を図 1 に示す．入力されたソースコードに対して，if 文, if-else 文の制御構造の波括弧を付与する正規化を行う．その後，パターンマッチングにより，連続の if 文と if-else 文を抽出する．抽出した if 文, if-else 文の字句系列から制御式と else の有無を抽出したのち，条件を区間に変換して判定要素を導出して判定する．

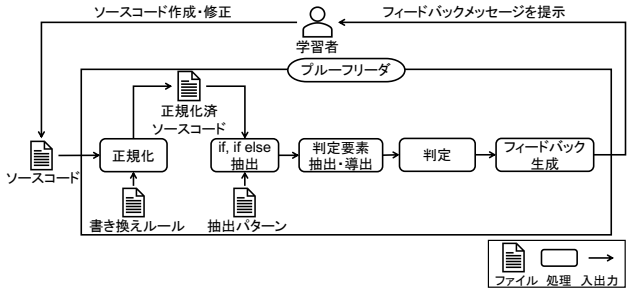

図 1 ツールの全体像

5.2 区間のデータ構造

単純区間の構成要素は，下端点の種類，下端値，上端値，上端点の種類である．ツールでは 4 つの要素を数値として保持し，単純区間クラスで区間を表現する．複合区間の構成要素は，論理演算子と単純区間のリストである．ツールでは論理演算子と単純区間クラスのインスタンスをリストで保持し複合区間を表現する．また，条件，前提条件，実行条件，排反条件を複合区間として表現する．判定要素とフィードバックに必要な情報を保持するデータ構造を判定要素クラスとする．クラス図を図 2 に示す．

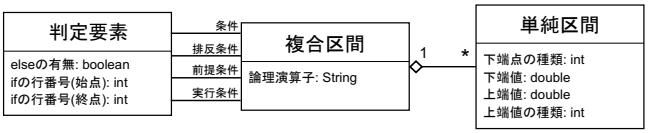

図 2 クラス図

5.3 実現

提案するプルーフリーダを Python を用いて実現した．ツールの規模は約 1,700 行である．

6 プルーフリーダの評価

提案したプルーフリーダの判定項目を網羅する 46 通りのテストケースを作成し，ツールの機能を評価する．例えば，判定項目 D の検証のため，大小関係を比較する指摘対象の条件式を「指摘可能な最初の if」,「最後の if」,「途中の if」に記述した 3 通り，それらの閾値の並び方を反転させた 3 通り，等価を判定する指摘対象の条件式をそれぞれの if に記述した 3 通りが作成できる．本ツールにテストケースのソースコードを入力して検証した結果，46 通りすべてにおいて期待通りのフィードバックが得られた．

提案したプルーフリーダがプログラミング演習において実用的であるか評価した．2019 年度に本学理工学部の「プログラミング基礎」を履修した学習者 4 名のソースコードのうち，判定可能な if 文, if-else 文を含む 39 個に対して正しく判定できるか検証した．また，フィードバック通りに修正することで教育意図を満たすソースコードに修正できるか検証した．結果，37 個で適切なフィードバックが得られた．そのうち，指摘が必要なソースコード 5 個に対して，最大 2 回ツール実行と修正を繰り返すことで教育意図を満たすソースコードに修正できることを確認した．残り 2 個のソースコードはチェック対象外の if が含まれており，ツールが正しく実行できなかった．

7 考察

7.1 模範解答の活用

本研究で提案したプルーフリーダは，学習者に修正を促す指摘が必ずしも教育者の意図に沿っているとは限らない．また，誤りの箇所を確定させるのが困難であるほか，修正方法によっては実行結果が変化する可能性があり，判定の精度が課題である．判定精度向上のために，模範解答を利用する方法がある．学習者の解答と模範解答の if の判定要素をそれぞれ作成し，それらを比較することで誤り箇所の特定や教育意図を適切に反映することが可能となる．また，ソースコード 1 のような条件分岐において '=' が不足している場合，実行条件が連結となるので，提案した手法では誤りの可能性を指摘できない．模範解答の活用による判定精度向上と判定可能範囲の拡張が課題である．

7.2 区間を用いた判定の計算量

入力する制御式の個数を n とする．判定項目 A，B では条件式 2 つの組合せを調べるので，計算量は $O(n^2)$ である．判定項目 C，D では全ての if の条件式を 1 回ずつ調べればよいので，計算量は $O(n)$ である．判定項目 E は判定自体の計算量は $O(n)$ だが，条件式の閾値が降順である前提なので，ソートの計算量が $O(n \log n)$ である．

区間を用いた判定の最悪計算量は $O(n^2)$ であるが，一般に n は高々十数個であるので，実用的な時間内で判定できる．実際に判定の計算量が $O(n^2)$ となる $n = 6$ の連続の if 文をツールに入力すると，実行時間は 0.40 秒であった．実行環境は MacBook Air (M1, 2020) である．

8 おわりに

本研究では，学習者が if 文や if-else 文の条件を適切に記述できるように支援するプルーフリーダを提案した．区間を用いることで条件の範囲に対する判定が可能となり，必要のない条件記述や，閾値や演算子誤りの可能性の指摘が可能になった．今後の課題として，チェック対象の拡張や模範解答を利用した判定精度の向上が挙げられる．

参考文献

[1] 原田知空, 前川雅貴: “判定木を用いたプログラミング学習用プルーフリーダの提案”, 南山大学理工学部 2021 年度卒業論文 (2022).

[2] 蜂巣吉成, 吉田敦, 桑原寛明, 阿草清滋: “プログラミング学習用プルーフリーダの試作”, コンピュータソフトウェア, Vol.35, No.4, pp.129-135 (2018).

[3] 内田公太, 権藤克彦: “C 言語初学者向けツール C-Helper の現状と展望”, 第 54 回プログラミング・シンポジウム, pp.153-160 (2013).

[4] 大須賀俊憲, 小林隆志, 渥美紀寿, 間瀬順一, 山本晋一郎, 鈴村延保, 阿草清滋: “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).