6L-01

C++ コンセプトの自動生成による C++ テンプレートのエラーメッセージの改善

繁田 鈴之介 †　小宮 常康 ‡
電気通信大学　大学院情報理工学研究科 † ‡

1　はじめに

C++ テンプレートによって発生するエラーは読みにくく，エラーの量も膨大になりやすい．そこで，C++ ではコンセプトという機能を導入した．コンセプトを用いることでテンプレートを正しく使うための**要件**を記述でき，その要件に基づいたチェックを行える．この機能により，要件に基づいた分かりやすいエラーメッセージを出力するようになるが，現状要件の記述は手動で行う必要がある．そこで，本研究ではこれらの要件を自動生成することによって，出力されるエラーメッセージの改善を図る．本研究により，コンセプトを用いていないテンプレートがコンセプトによるチェックの恩恵を受けられるようになることが期待される．

2　C++ コンセプト

C++ テンプレートは複数の型に対応した定義を扱うための C++ の機能である．C++ のテンプレート関数の例を示す．

```
1 template<class T>
2 T sum(T a, T b) {
3     return a + b;
4 }
```

先頭でテンプレートパラメータ T を宣言している．テンプレートパラメータは関数の内部で型として使うことができる．このテンプレート関数は以下のように具体的な型を与えて使用することができる．指定した型によって T を置き換えた新しい関数定義が生成される．

```
1 sum<int>(1, 2); //int + int -> int
2 sum(1, 2); //int + int -> int
3 sum(1.0, 2.0); //double + double -> double
4 sum("abc", "def"); //エラー
```

<>の中で型を指定できる．また，2 行目以降のように引数から型を推論させることもできる．4 行目では const char*型を与えているが，+ によって const char*型どうしを足すことはできない．このように，指定した型による置き換えに失敗したときエラーとなる．今回の sum ような単純な関数の場合はエラーが発生したときの原因は単純で明らかであり，エラーメッセージも難しくはないが，置き換えが沢山発生するような複雑なテンプレート関数の場合，置き換え失敗によるエラーメッセージは膨大で複雑なものとなる．

このような膨大で複雑なエラーメッセージを改善することを目的の一つとして C++20 より導入されたのがコンセプトである．コンセプトを用いることでテンプレートを正しく使うための要件を記述できる．コンセプト定義の例を以下に示す．

```
1 template<class T>
2 concept Addable = requires (T a, T b) {
3     {a + b} -> std::convertible_to<T>;
4 };
```

T 型の a と b について + によって足し算ができ，その結果が T に変換できるという意味の要件を表すコンセプト定義である．

この定義は以下のように用いることで T として指定できる型を制限することができる．

```
1 template<class T>
2 requires Addable<T>
3 T sum(T a, T b) {
4     return a + b;
5 }
```

sum("abc", "def") とするとコンセプトを用いない場合と同じくエラーが発生する．コンセプトがない場合との違いはエラーが発生するタイミングである．コンセプトなしの sum では，テンプレート関数内部でエラーとなるが，コンセプトありの sum では，テンプレート関数の内部を見る前に制約によってエラーとなる．

3　コンセプトのエラーメッセージ

コンセプトを用いることで関数内部の個々の置き換えに反応したエラーメッセージではなく，要件に関わるエラーメッセージを出力するようになる．よって，より分かりやすいエラーメッセージとなることが期待される．例えば，以下のようなテンプレート関数 f，g があると

Improving error messages about C++ templates by automatically generating C++ concepts

† Suzunosuke Shigeta,
The University of Electro-Communications
‡ Tsuneyasu Komiya,
The University of Electro-Communications

する．

```
template<class S, class V>
void g() {
    V::inner;
}

template<class T, class U>
void f() {
    g<T, U>();
}
```

テンプレート関数 f の内部で別のテンプレート関数 g が呼ばれている．g の内部では V のメンバ inner のみの式がある．

この f に対して，f<int, int>() と型を与えた場合，以下のようなエラーメッセージが出力される（gcc version 10.3.0）．

```
4.cc: In instantiation of ‘void g() [with S = int; V =
      int]’:
4.cc:15:12: required from ‘void f() [with T = int; U =
      int]’
4.cc:19:17: required from here
4.cc:10:8: error: ‘inner’ is not a member of ‘int’
   10 | V::inner;
      | ~~~~~
```

このエラーメッセージからは int がメンバ inner を持っていないことによるエラーだとは分かるが，f に対して与えた 2 つの int 型のどちらがエラーの原因か分からない．

上記のプログラムに対して，コンセプトによる制約を追加したのが以下である．

```
template<class S, class V>
void g() {
    V::inner;
}

template<class T, class U>
concept C_f = requires () {
    g<T, U>();
    U::inner;
};

template<class T, class U>
requires C_f<T, U>
void f() {
    g<T, U>();
}
```

f のテンプレートパラメータに制約を課している．以下がエラーメッセージである．

```
4.cc: In function ‘int main()’:
4.cc:21:17: error: use of function ‘void f() [with T =
      int; U = int]’ with unsatisfied constraints
   21 | f<int, int>();
      | ^
4.cc:16:6: note: declared here
   16 | void f() {
      | ^
4.cc:16:6: note: constraints not satisfied
4.cc: In instantiation of ‘void f() [with T = int; U =
      int]’:
4.cc:21:17: required from here
4.cc:9:9: required for the satisfaction of ‘C_f<T, U>’ [
      with T = int; U = int]
4.cc:9:15: in requirements [with T = int; U = int]
4.cc:11:8: note: the required expression ‘U::inner’ is
      invalid, because
   11 | U::inner;
      | ~~~~~
4.cc:11:8: error: ‘inner’ is not a member of ‘int’
```

11 行目より，f に追加した制約 C_f を満たしていないことや，12〜16 行目より，f のテンプレートパラメータ U に与えた型 int が要件を満たしていないことによるエラーだと分かる．このように，コンセプトを用いることでエラーメッセージが改善される．

ただし，ここではコンセプトの要件を満たさないような入力の範囲が，元々のテンプレート関数の内部でエラーとなるような入力の範囲と一致することを仮定している．そうでなければ元々の関数でエラーとならないような入力がエラーとなってしまう．この問題はテンプレート関数内部の要件を列挙し，それをコンセプト定義に与えれば解決するが，現状そのような列挙は手動で行う必要がある．

4 提案手法

テンプレート関数本体のテンプレートパラメータに関係する式を列挙し，それらを要件として持つコンセプト定義を生成し元の関数に追加する．列挙される式の中に別のテンプレート関数の呼び出しがあれば，その呼び出し先のテンプレート関数本体も調べる．f を例にとると，C_f を生成し f の制約に設定する流れを自動化するということである．

5 関連研究

A. Sutton らの研究 [1] では，コンセプト定義の集合を用いて，テンプレート関数内の要件を表現することを試みている．本研究に似た部分が多い．異なる点として，彼らの研究ではエラーメッセージの改善について直接的には言及していない．また，内部で呼び出される別のテンプレート関数について扱ってはいない．彼らの研究では，より要件の表現を洗練させる方法について扱っている．

6 今後に向けて

提案手法を実装し，STL などでエラーメッセージについて実験を行う．要件の洗練やクラステンプレートへの対応についても考えていく．

参考文献

[1] A. Sutton and J. I. Maletic, ”Automatically identifying C++0x concepts in function templates,” 2008 IEEE International Conference on Software Maintenance, Beijing, China, 2008, pp. 57-66, doi: 10.1109/ICSM.2008.4658054.

[2] Working Draft, Standard for Programming Language C++.
https://timsong-cpp.github.io/cppwp