

BÁO CÁO THỰC HÀNH

XÂY DỰNG CHƯƠNG TRÌNH DỊCH

TUẦN 5: TYPE_CHECKING BUỔI 1

Họ và tên: Bùi Quang Hưng

Mã số sinh viên: 20225849

I. Tổng quan về Type checking

Type Checking (Kiểm tra kiểu) là một giai đoạn quan trọng trong phân tích ngữ nghĩa của trình biên dịch, đảm bảo rằng các phép toán và biểu thức trong chương trình sử dụng đúng kiểu dữ liệu. Trong ngôn ngữ KPL, type checking được thực hiện trong quá trình phân tích cú pháp (parsing) để phát hiện sớm các lỗi về kiểu dữ liệu.

- Các kiểu dữ liệu trong KPL:
 - Kiểu cơ bản (Basic Type): INTEGER, CHAR
 - Kiểu mảng (Array Type): ARRAY[size] OF element_type
 - Kiểu do người dùng định nghĩa: Thông qua khai báo TYPE
- Mục tiêu của Type checking:
 - Đảm bảo sự nhất quán kiểu trong khai báo hằng, biến, tham số
 - Kiểm tra tính tương thích kiểu trong biểu thức, lệnh gán, điều kiện
 - Xác minh tính đúng đắn của các phép toán số học và logic
 - Phát hiện lỗi sử dụng mảng (số chiều, kiểu chỉ số)

II. Mô tả hàm compileConst và ứng dụng

1. Vị trí trong mã nguồn

Hàm compileConstant() nằm trong parser.c và được gọi khi xử lý phần khai báo hằng (qua hàm xử lý CONST trong compileBlock / compileConstDecl), đồng thời cũng được gọi khi phân tích biểu thức (compileExpression / compileTerm) để đọc hằng.

2. Mã nguồn chi tiết của hàm

```

254 ConstantValue *compileConstant(void)
255 {
256     ConstantValue *constValue;
257
258     switch (lookAhead->tokenType)
259     {
260         case SB_PLUS:
261             eat(SB_PLUS);
262             constValue = compileConstant2();
263             if (constValue->type != TP_INT)
264                 error(ERR_TYPE_INCONSISTENCY, currentToken->lineNo, currentToken->colNo);
265             break;
266         case SB_MINUS:
267             eat(SB_MINUS);
268             constValue = compileConstant2();
269             if (constValue->type != TP_INT)
270                 error(ERR_TYPE_INCONSISTENCY, currentToken->lineNo, currentToken->colNo);
271             constValue->intValue = -constValue->intValue;
272             break;
273         case TK_CHAR:
274             eat(TK_CHAR);
275             constValue = makeCharConstant(currentToken->string[0]);
276             break;
277         default:
278             constValue = compileConstant2();
279             break;
280     }
281     return constValue;
282 }

```

3. Phân tích hoạt động

- Đọc token hiện tại; nếu gặp dấu + hoặc - ở vị trí tiền tố thì ghi nhận dấu và chuyển sang token tiếp.
- Nếu gặp NUMBER: tạo TypeDescriptor dạng INT, áp dụng dấu tiền tố (+/-) lên giá trị số.
- Nếu gặp CHAR_CONST: chỉ chấp nhận khi không có dấu - / + (tùy quy tắc ngôn ngữ); nếu có dấu tiền tố không hợp lệ => báo lỗi kiểu và trả về TYPE_ERROR.
- Nếu token không phải số/ký tự hợp lệ => báo lỗi “Invalid symbol” và trả về TYPE_ERROR để không làm gián đoạn toàn bộ phân tích.
- Hàm trả về cấu trúc mô tả kiểu và (tùy cần) giá trị hằng để các giai đoạn sau dùng.

4. Ứng dụng trong Type Checking

- compileConstant cung cấp kiểu và giá trị hằng cho:
 - Kiểm tra khai báo CONST (đảm bảo hằng tương ứng kiểu khai báo).
 - Xây dựng AST/descriptor cho biểu thức, giúp hàm kiểm tra kiểu (semantics) quyết định kiểu kết quả của phép toán.
 - Phát hiện trường hợp bất hợp lệ: ví dụ +/- áp trên CHAR => phát sinh thông báo lỗi kiểu sớm (granularity cao).
- Nếu trả về TYPE_ERROR, các kiểm tra tiếp theo sẽ lan truyền lỗi nhưng cho phép phân tích tiếp (error recovery).

5. Vị trí trong quy trình biên dịch

- Sau lexical analysis (tokenizer), parser gọi compileConstant khi đang phân tích expression/term hoặc khi đọc phần CONST trong chương trình.

- Kết quả của compileConstant được dùng ngay trong pha parsing để xây AST và cũng tham chiếu trong pha semantic/type checking để so sánh kiểu (type checking).

III. Kết quả thực hiện với Example12.kpl không lỗi

```
Type_checking > tests > Example12.txt
1 PROGRAM EXAMPLE12;
2 VAR A : INTEGER;
3 B:INTEGER;
4 C:INTEGER;
5 BEGIN
6 A:=10;
7 C:=0;
8 FOR B:=1 to A do
9 if B/2*2 != 0 then C:=C+B;
10 END.
11

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS
powershell - Type_checking + × └

● PS D:\CTD_Lab\Lab5\Type_checking> .\kpl.exe tests\Example12.txt
Program EXAMPLE12
Var A : Int
Var B : Int
Var C : Int
○ PS D:\CTD_Lab\Lab5\Type_checking> ┌
```

IV. Kiểm tra điều khiển lệnh FOR bằng biến ký tự

1. Mã nguồn và kết quả

```
Type_checking > tests > error_for_char.kpl
1 PROGRAM ERRORFORCHAR;
2 VAR C : CHAR;
3 | I : INTEGER;
4 BEGIN
5 I := 0;
6 (* Lỗi: Biến điều khiển FOR phải là INTEGER, không phải CHAR *)
7 FOR C := 'a' TO 'z' DO
8 | I := I + 1;
9 END.
10

PROBLEMS OUTPUT TERMINAL ...
powershell - Type_checking + × └ ...
● PS D:\CTD_Lab\Lab5\Type_checking> .\kpl.exe tests\error_for_char.kpl
7-5:Type inconsistency
○ PS D:\CTD_Lab\Lab5\Type_checking> ┌
```

2. Luồng hoạt động

- Quy tắc hiện tại: biến điều khiển và hai biểu thức giới hạn , phải có cùng kiểu (so sánh bằng nhau về kiểu). So sánh kiểu thực hiện bằng hàm checkTypeEquality (gọi compareType và báo ERR_TYPE_INCONSISTENCY nếu không khớp).
- Luồng chi tiết:
 - eat(KW_FOR) — đọc từ khóa FOR.
 - eat(TK_IDENTIFIER) — đọc identifier (tên biến điều khiển).
 - checkDeclaredVariable(name) — kiểm tra tên đã được khai báo là biến; lấy Type *varType của biến.
 - eat(SB_ASSIGN) — đọc dấu gán ":=".

- `exp1Type = compileExpression()` — phân tích biểu thức bắt đầu; `compileExpression` trả về con trỏ Type cho `exp1`.
 - `checkTypeEquality(varType, exp1Type)` — so sánh kiểu biến điều khiển và `exp1`; nếu khác nhau gọi `error(ERR_TYPE_INCONSISTENCY, line, col)`.
 - `eat(KW_TO)` — đọc từ khóa TO.
 - `exp2Type = compileExpression()` — phân tích biểu thức kết thúc; trả về Type * cho `exp2`.
 - `checkTypeEquality(varType, exp2Type)` — so sánh kiểu biến điều khiển và `exp2`; nếu khác nhau báo `ERR_TYPE_INCONSISTENCY`.
 - `eat(KW_DO)` — đọc DO và tiếp tục phân tích thân vòng lặp (các câu lệnh bên trong).
 - Nếu cả hai so sánh kiểu đều thành công thì tiếp tục xử lý thân FOR bình thường.

- Cơ chế lỗi và thông báo:

 - So sánh kiểu thực hiện qua `checkTypeEquality` → gọi `compareType`; nếu không bằng nhau sẽ sinh `ERR_TYPE_INCONSISTENCY` với vị trí token hiện tại.
 - Ví dụ với `tests/error_for_char.kpl`: biến C : CHAR và cả 'a'/'z' là CHAR → các kiểm tra kiểu đều PASS và chương trình được chấp nhận; nếu một thành phần khác kiểu (ví dụ `exp1` là INT) sẽ báo lỗi Type inconsistency tại vị trí tương ứng.

V. Thay đổi các ví dụ đã cho để gây ra lỗi

1. Lỗi không tương ứng kiểu khi khai báo hằng

```
Type_checking > tests > example1_error_const.kpl
1 PROGRAM EXAMPLE1ERROR;
2 (* Dựa trên Example1 - Lỗi: Không tương ứng kiểu khi khai báo hằng *)
3 CONST X = +'A';
4 | | Y = -'B';
5 VAR A : INTEGER;
6 BEGIN
7 | A := 10;
8 END.
9
```

- Thay đổi: trong phần CONST khai báo $X = +'A'$; $Y = -'B'$; (tiền tố + / - áp lên ký tự).
 - Tại sao lỗi (theo code hiện tại): trong pha phân tích hàng parser gọi `compileConstant()` → sau `SB_PLUS/SB_MINUS` gọi `compileConstant2()` và `compileConstant2()` chỉ chấp nhận `NUMBER` hoặc `IDENT`; `TK_CHAR` không được chấp nhận ở vị trí này.
 - Hàm phát hiện: `parser::compileConstant()/compileConstant2()`.
 - Thông báo mong đợi: `ERR_INVALID_CONSTANT` → "A constant expected." (ví dụ 3-13:A constant expected).

2. Lỗi dùng sai số chiều của mảng so với khai báo

- Thay đổi: truy cập mảng với số chiều không khớp khai báo (ví dụ khai báo A 1 chiều nhưng dùng A(i).(j) 2 chiều).
 - Tại sao lỗi: khi phân tích LValue parser/semantics kiểm tra số chỉ số và kiểu từng chỉ số; số chiều/kết quả truy xuất không khớp với khai báo → kiểu truy xuất không hợp lệ.
 - Hàm phát hiện: parser::compileLValue / semantics::checkArrayIndex / compareType.
 - Thông báo mong đợi: ERR_TYPE_INCONSISTENCY (hoặc ERR_INVALID_INDEX tùy vị trí) → "Type inconsistency" (ví dụ 13-11: Type inconsistency).

3. Lỗi một biểu thức bắt đầu bằng + nhưng phần sau đó lại có kiểu ký tự

```
Type_checking > tests > ⚒ example2_error_plus_char.kpl
1  PROGRAM EXAMPLE2ERROR;
2  (* Dựa trên Example2 - Lỗi: Biểu thức bắt đầu bằng + với kiểu ký tự *)
3  VAR N : INTEGER;
4      C : CHAR;
5
6  FUNCTION F(N : INTEGER) : INTEGER;
7  BEGIN
8      IF N = 0 THEN F := 1 ELSE F := N * F (N - 1);
9  END;
10
11 BEGIN
12     C := 'A';
13     (* Lỗi: biểu thức + với ký tự *)
14     N := + C;
15     FOR N := 1 TO 7 DO
16         BEGIN
17             CALL WRITELN;
18             CALL WRITEI(F(N));
19         END;
20     END.
21

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   PORTS   QUERY RESULTS   powershell - Type_checking + ✎ ✎ ... | { } ×

PS D:\CTD_Lab\Lab5\Type_checking> .\kpl.exe .\tests\example2_error_plus_char.kpl
14-10:Type inconsistency
PS D:\CTD_Lab\Lab5\Type_checking>
```

- Thay đổi: Trong thân chương trình thêm biểu thức như $N := + C$; với C là CHAR.
- Tại sao lỗi: Tiền tố $+$ chỉ hợp lệ khi toán hạng là số (INT); $+$ applied to CHAR không hợp lệ.
- Quy tắc semantic: compileExpression/compileConstant không chấp nhận $+-$ tiền tố trước giá trị CHAR.
- Hàm kiểm tra liên quan: compileConstant trả về TYPE_ERROR; checkChar/compareType dẫn đến ERR_TYPE_INCONSISTENCY.
- Thông báo mong đợi: Type inconsistency (vị trí toán hạng sau $+$).

4. Lỗi không tương ứng kiểu trong lệnh gán

The screenshot shows a code editor with a dark theme. The file is named `example5_error_assign.kpl`. The code contains several assignments between variables of different types:

```

1 PROGRAM EXAMPLE5ERROR;
2 (* Dựa trên Example5 - Lỗi: Không tương ứng kiểu trong lệnh gán *)
3 CONST C = 1;
4 TYPE T = CHAR;
5 VAR V : CHAR;
6   N : INTEGER;
7
8 BEGIN
9   V := 'A';
10  N := 10;
11  (* Lỗi: gán integer cho char *)
12  V := N;
13  (* Lỗi: gán char cho integer *)
14  N := V;
15 END.
16

```

The terminal below shows the execution of the program. It outputs two error messages:

```

PS D:\CTD_Lab\Lab5\Type_checking> .\kpl.exe .\tests\example5_error_assign.kpl
12-8:Type inconsistency
PS D:\CTD_Lab\Lab5\Type_checking>

```

- Thay đổi: Gán giá trị kiểu CHAR cho biến INTEGER hoặc ngược lại (ví dụ $V := N$ khi $V:int$, $N:char$).
- Tại sao lỗi: Lệnh gán yêu cầu kiểu bên trái và bên phải tương thích.
- Quy tắc semantic: trong compileAssignSt so sánh kiểu LValue và RHS bằng compareType; không tương thích \Rightarrow lỗi.
- Hàm kiểm tra liên quan: checkTypeEquality / compareType \rightarrow ERR_TYPE_INCONSISTENCY.
- Thông báo mong đợi: Type inconsistency (vị trí dấu $:=$ hoặc RHS).

5. Lỗi không tương ứng kiểu trong lệnh while

The screenshot shows a code editor window with a dark theme. The file being edited is `example3_error_while.kpl`. The code is as follows:

```
1 PROGRAM EXAMPLE3ERROR;
2 (* Dựa trên Example3 - Lỗi: Không tương ứng kiểu trong lệnh while *)
3 VAR I : INTEGER;
4   | N : INTEGER;
5   | C : CHAR;
6
7 BEGIN
8   | N := 5;
9   | C := 'X';
10  | I := 0;
11  (* Lỗi: so sánh integer với char trong while *)
12 WHILE N = C DO
13   | BEGIN
14   |   | N := N - 1;
```

Below the code editor is a terminal window showing the command and its output:

```
PS D:\CTD_Lab\Lab5\Type_checking> .\kpl.exe .\tests\example3_error_while.kpl
12-13:Type inconsistency
```

- Thay đổi: Thay điều kiện WHILE bằng phép so sánh giữa INT và CHAR hoặc dùng biểu thức trả về CHAR trong điều kiện cần boolean/INT.
- Tại sao lỗi: Điều kiện while yêu cầu kiểu phù hợp theo quy tắc (ví dụ phải là INT hoặc các hai vế so sánh phải cùng kiểu); so sánh giữa INT và CHAR là không tương thích.
- Quy tắc semantic: compileExpression kiểm tra kiểu điều kiện; compareType/checkIntType sẽ phát hiện khác kiểu.
- Hàm kiểm tra liên quan: checkTypeEquality / checkIntType → ERR_TYPE_INCONSISTENCY.
- Thông báo mong đợi: Type inconsistency (tại vị trí biểu thức điều kiện).