Variability and Type Calculation for Equation of Modelica model

Junjie Tang Jianwan Ding Liping Chen Xiong Gong
National CAD Center
1037 Luoyu Road, Wuhan, China
tjj.hust@gmail.com dingjw@hustcad.com Chenlp@hustcad.com Gongx@hustcad.com

Abstract:

Differential algebraic equations (DAEs), translated from Modelica model, is usually represented by bipartite graph. One of basic premises of creating bipartite graph is to determine types of variables and equations. Type calculation of Modelica equation has been researched and a serial of rules for variability and type calculation has been concluded in this paper.

Equation type is the type of variable that equation can solve. Equation type is calculated in symbolic by both variability and basic type of its sub-expressions. Generally, type calculation is a bottom-up way as expression is represented in form of tree. But, there are kinds of particular expressions, such as integer(), noEvent(), multi-output function call expression, etc, which may cause type and variability incompatible problem. The issue is discussed in the paper, and several rules for variability and type calculation are present. These rules will helps to debug out obscure errors, and several typical examples are present to show how the rules work.

Keyword: equation type; equation variability; compatibility of variability and type; model debug

1. Introduction

Differential algebraic equations (DAEs), generated by compiling and translating Modelica model, should be debugged by method of structural analysis [1], and be reduced and decomposed to subsystem serials [2], to reduce the scale of equation system and improve efficiency of numerical calculation. Most of these symbolic operations are usually processed based on representation of bipartite graph of DAEs, and one of basic premises of creating bipartite graph is determining type of variables and equations. The type of variable, obviously, is as defined in model, while the type of equation has to be deduced from its sub-expressions [3].

Equation type is the type of variable that equation can solve. Equation type is symbolic calculated by both variability and basic type of its sub-expressions. Generally, type calculation is a bottom-up way as expression is represented in form of tree. But, there are some particular expressions, such as integer(), noEvent(), multi-output function call expression, etc, which may cause type and variability incompatible problem, that is variability of equation is not compatible with type of it. In the paper, reason for this problem is discussed, and the way for debugging is introduced.

Section 2 shows basic rules of equation

type calculation. Section 3 analyzes type and variability incompatible problem and concludes corresponding rules for variability and type calculation. Section 4 complements an additional rule for variability calculation for symbolic transformation. Section 5 is the conclusion of this paper.

2. Basic Rules for Equation Type Calculation

There are four basic types of variable of Modelica model, as Real, Integer, Boolean, and String, predefined by Modelica. Correspondingly, there are four basic types for equation. Record equations should be split into basic types. The following rules of equation type calculation can be summarized from Modelica Language Specification (Modelica 3.2, 28, 61, 64) [4]:

Rule 1. The resulting type of equation is the same as of the type compatible expression of two sides.

Rule 2. The resulting variability of equation is the higher variability of two sides.

3. Compatibility of Type and Variability

As we known, Real type variable can hold all kinds of variability, while variability of Integer, Boolean or String Type variable is no higher than discrete. So does expressions and equations. That is the rule:

Rule 3. The resulting type and variability of equation variability must be compatible.

In following part, cases for Rule 3 are introduced, to show how it works.

3.1 integer() and noEvent()

There are kinds of pre-defined built-in function in Modelica (Modelica 3.2, 19, 109). Generally, type and variability calculation of call of these functions follows Rule 1 and 2. However, there are two particular functions, integer() and noEvent(), are quite different. According definition of them, there is a rule for their variability and type calculation.

Rule 4. Variability of integer() is no higher than discrete, and type of it is integer; variability of noEvent() is continuous, and type of it is the same as that of input argument.

integer() is more like a implicit type conversion function [5], like int() in C++, that means variability of integer(x) is no higher than discrete, even x is continuous, and type of it is integer, obviously. So, integer(x) indicates that x must be an independent variable. Here is an example:

```
function fn1
  input Real x;
  input Integer y;
  output Real z;
  annotation (derivative = derfn1);
algorithm
  z:=x^2+y^2;
end fn1;

function derfn1
  input Real x;
  input Integer y;
  input Real xder;
  output Real zder;
algorithm
  zder:=2*x*xder+y^2;
end derfn1;

Real x;
Real y;
Real u;
Real v;
equation
```

*u=sin(2*time); // eq1*
x =cos(time); // eq2
u= fn1(x, integer(y)); // eq3
v = der(y)+x; // eq4

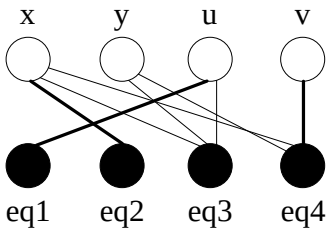

Figure 1 Bipartite graph of equations, with matching edges marked by thick lines

In the example, each equation is legal, all their types are continuous, and here is the bipartite graph for equations of model (Fig. 1). When finding matching in bipartite graph, it should be careful that argument of integer() cannot be the matching vertex with the same equation integer() present in. So, in this case, result of structural analysis is that, y is under-determined, while x and u are over-determined with equation eq1, eq2 and eq3.

noEvent() doesn't trig event as defined (Modelica 3.2, 26), even input argument is an event expression. When we combined noEvent() and integer() together, there was an interesting result. Here is the simple example.

Real x;
Integer i = 1;
Integer i2;
equation
i = x; // eq1, legal
i2 = noEvent(integer(x)); // eq2, illegal

In this case, eq1 is legal, as variability of eq1 is continuous (higher one of i and x), and type of it is Real (as the same type of type compatible variable of i and x). However, eq2 is illegal, variability of eq2 is continuous (right hand is continuous), while type of eq2 is Integer (type of two hands of eq2 are Integer), that is breach of Rule 3.

3.2 Call Expression of User-define Function

Type and variability calculation of function call follows Rule 5:

Rule 5. Assume that function is defined as single output, variability of function call expression is the same as the higher variability of input real arguments; type of function call is the same as the one of output formal parameter.

Variability of function call expression is the same as the higher variability of all input real arguments, is because that "all assignment statements within function are implicitly treated with the noEvent function" (Modelica 3.2, 87), which means if input arguments are continuous, it is impossible to generate a discrete output. For example:

parameter Real a = 10;
parameter Real b = a+2;
*parameter Real b = f(a+b, a)*2;*

Since both a+b and a are parameter, f(a+b, a) is parameter. But if model is:

Real a = 10;
parameter Real b = 2;
*parameter Real b= f(a,2)*2; //illegal*

Variability of f(a,2) is the higher variability of a and 2, that is continuous.

A more complex example is like follows:

function fn1
input Real x;
input Integer y;
output Real z;
annotation (derivative = derfn1);
algorithm
z:=x^2+y^2;
end fn1;

function derfn1
input Real x;
input Integer y;
input Real xder;
output Real zder;

```
algorithm
  zder:=2*x*xder+y^2;
end derfn1;

function fn2
  input Real a;
  output Integer b;
algorithm
  b:= integer(a);
end fn2;

Real x;
Real y;
Real u;
Real v;
equation
  u=sin(2*time);
  x =cos(time);
  u= fn1(x, fn2(y)); // eq1, illegal
  v = der(y)+x;
```

In eq1, the variability of fn1(x, fn2(y)) is continuous, as x is continuous, and the type of it is Real, then the right hand and left hand have the compatible variability and type. However, the variability of fn2(y) is continuous, and the type of it is Integer, that is breach of Rule 3. We could make a transformation to show this obscure error more clearly. Let introduce Integer variable, like:

```
Integer i = fn2(y); // eq2
```

Then eq1 is conceptually equivalent with:

```
u= fn1(x,i); // eq1', legal
```

After transformation, eq1' is legal, but eq2 is a wrong equation, obviously, as type of variable i is discrete, and variability of fn2(y) is continuous. It is impossible to assign a continuous value to a discrete variable.

3.3 Symbol "."

"." is a symbol for member access. Let extend its meanings to present split pattern of multi-output function call.

For call expression of multi-output function, variability and type calculation follows Rule 5, with a split transformation of function call. That is, equation that contains multi-output function call expression should be split into basic types before type calculation. Take following case as an example:

```
function fn
  input Real x;
  input Real y;
  output Real u;
  output Integer v;
algorithm
  u := x+y;
  v := integer(x-y);
end fn;
Real a,b,c,d;
Integer k;
equation
  c=3*sin(time);
  d=cos(time);
  (a,k)=fn(c,d); // eq1
  b=fn(2,if c>0 then 3 else -0.5); //eq2
```

In this case, eq1 should be split into following equations:

```
a=fn(c,d).u; // eq1-1
k=fn(c,d).v; // eq1-2
```

And eq2, though it is a basic type equation, should be equivalently transformed into:

```
b=fn(2,if c>0 then 3 else -0.5).u
```

Key of split transformation is to put the corresponding output formal parameter at the right position, with a symbol ".", as a member attached to its parent expression.

With these transformations, type of multi-output function call expression could be calculated by Rule 5. For example, the type of right hand of equation k=fn(c,d).v is the type of v, that is Integer, and the type of equation is Integer. Variability of right hand is continuous (higher variability of c and d), and variability of equation is continuous. It

indicates that there is an error in equation (a,k)=fn(c,d), with a breach of Rule 3.

3.4 If-Expression

If-expression is defined as "if expression1 then expression2 else expression3" (Modelica 3.2, 19). Rule for variability and type calculation of if expression is:

Rule 6. Variability of if-expression is the highest variability of expression1, expression2 and exression3, type of it is the type of expression2.

Type of if-expression is type of expression2, as expression2 and expression3 should be defined as type compatible, while expression1 affects variability of if-expression. For example:

Integer x=if noEvent(time>0) then 1 else 2;

Variability of equation in the example is continuous, as that of right hand is continuous, following Rule 6, while type of equation is Integer, as both hands are Integer. Thus, resulting variability and type breach Rule 3, means that equation is illegal.

3.5 Event Expressions

For event expressions, like event triggering mathematical functions (Modelica 3.2, 21), relational expressions, etc, calculation rule is:

Rule 7. Variability of event expression is no higher than discrete, unless it is present in when-clause.

For example:

```
  Integer x;
equation
  when time > 0 then
  x = if noEvent(time > 0) then 1 else 2;
  end when;
```

For equation x = if noEvent(time > 0) then 1 else 2 is present in when-clause, variability of equation is discrete, different with example in section 3.4, and type of it is Integer. So, in this case, Rule 3 is followed, and the equation is legal.

4. Rule for Symbolic Transformation

In the case where function call is inlined, part of assignment statements will become the part of equations, and the inlined result should be treated with noEvent function. Here is an example:

```
function f
  input Real in1;
  input Real in2;
  output Integer out1;
annotation(Inline=true);
    algorithm
      out1 := if in1>0 then in1 else in2;
    end f;
    Real x;
    Integer i = 2;
  equation
    i = f(x, time);// eq3
```

When f(x, time) of eq3 is inlined, the inlined result should be:

i =if noEvent (x>0) then x else time;

rather than:

i =if x>0 then x else time;

It is concluded as:

Rule 8. Symbolic transformation must not change basic type and variability of equation.

5. Conclusion

Type calculation of Modelica equation has been researched and a serial of rules for variability and type calculation has been concluded in the paper. Kinds of expression

are analyzed to explain possible variability and type incompatible problem, and more rules are introduced, with several examples to show how rules work. The rules for variability and type calculation for equation will helps to find out obscure errors in the model(such as examples in section 3), and to build more accurate bipartite graph for DAEs from Modelica model.

REFERENCES

[1]. Peter Bunus, Peter Fritzson. Methods for Structural Analysis and Debugging of Modelica Models. Proceedings of the 2nd International Modelica Conference, 2002, 10: 157~165

[2]. Ding Jianwan. Research on Methods for Consistency Analysis and Reduction of Declarative Simulation Models: [PhD thesis]. China: Huazhong University of Science & Technology, 2006

[3]. David Broman. Types in the Modelica Language. Proceedings of the 5th International Modelica Conference, 2006, 9:303~315

[4]. Modelica Language Specification V3.2. https://www.modelica.org/.

[5]. Peter Fritzson. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1. Wiley-IEEE Press, New York, USA, 2004

[6]. Futong Lv. Numerical Computing Methods, chapter 5. Tsinghua University Press, 2008.